Jack Brimberg*, Raca Todosijević**, and Dragan Urošević***

A SURVEY ON VARIABLE NEIGHBORHOOD SEARCH FOR THE UNCAPACITATED SINGLE ALLOCATION p-HUB MEDIAN PROBLEM

Abstract. The uncapacitated single allocation p-hub median problem (USApHMP) is one of the popular classical models in hub location theory that has been studied in the Operations Research literature for several years. It is a cornerstone problem for designing efficient and cost-effective logistics and telecommunications networks, where consolidation of flows through hub facilities leads to significant economies of scale. The main purpose of this chapter is to review a general variable neighborhood search that has been developed for this problem and successfully tested on standard benchmark datasets, including AP and URAND instances with up to 1000 nodes. We also examine the usefulness of a p-median formulation that approximates the USApHMP and can find good starting solutions for the presented VNS procedures. The computational experiments show the superiority of GVNS over other state-of-the-art heuristics for solving the USApHMP. In addition, it is empirically shown that a surrogate p-median problem that deletes transfer costs between hubs from the objective function (i.e., considers only collection and distribution costs) can be used to generate high-quality starting solutions.

Mathematics Subject Classification (2020): 90C27; 90B80.

Keywords: p-hub location problem, variable neighborhood descent, general variable neighborhood search, p-median approximation

*Royal Military College of Canada, Kingston, Canada
Jack.Brimberg@rmc.ca
ORCID: 0000-0003-1296-7705
**Université Polytechnique Hauts-de-France Institut des Sciences et Techniques de Valenciennes, Valenciennes, France
racatodosijevic@gmail.com
ORCID: 0000-0002-9321-3464
***Mathematical Institute of the Serbian Academy of Sciences and Arts, Belgrade, Serbia
draganu@mi.sanu.ac.rs
ORCID: 0000-0003-3607-6704

https://doi.org/10.64191/zr2510310103

Contents

1. Introduction 55
2. Connection with the p-median problem 59
3. Description of the VNS procedure 60
3.1. Solution representation 60
3.2. Initial solution 62
3.3. Neighborhoods 62
3.4. Shaking 63
3.5. Local improvement 63
3.6. GVNS heuristics for USApHMP 68
4. Computational results 68
4.1. Test instances 69
4.2. Strategies for generating initial solution 70
4.3. Main computational results 70
5. Conclusions 77
Acknowledgments 78
References 78

1. Introduction

The uncapacitated single-allocation p-hub median problem (USApHMP) is a fundamental problem in network design, aiming to minimize the total transportation cost in a hub-and-spoke system. This problem is typically defined on a complete symmetric graph $G = (V, E)$ with a set of n nodes, denoted as V, and a set of edges E. Each edge $uv \in E$ is associated with a distance c_{uv}. It is assumed that the distance matrix c_{uv} satisfies the triangle inequality. If this condition does not hold in the original network, the costs are replaced by shortest path distances between nodes, thereby ensuring that the triangle inequality is satisfied.

The "single-allocation" scheme means that each non-hub node must be assigned to exactly one hub, while the "uncapacitated" nature of the problem implies that there are no capacity restrictions on the amount of flow that is collected in a hub, thus simplifying the model by removing any capacity constraints. This particular variant of the p-hub median problem is a cornerstone for designing efficient and cost-effective logistics and telecommunications networks. Its objective is to select a subset of p nodes to serve as hubs, and to assign all other nodes (non-hubs) to these

selected hubs following single-allocation strategy, in such a way as to minimize the total transportation cost.

In the USApHMP, direct connections between non-hub nodes are not permitted. All traffic must pass through the hub network. Specifically, for a flow from an origin node i to a destination node j, the path is structured as follows: the flow travels from the origin node i to its assigned hub h_i (the "collection" phase), then from hub h_i to the destination's hub h_j (the "transfer" phase), and finally from hub h_j to the destination node j (the "distribution"" phase).

The total transportation cost for this problem is calculated by summing the costs of all origin-destination (O-D) pairs. The cost per unit of flow from node i to node j is determined by the distances and three key cost parameters: γ (collection cost rate), α (transfer cost rate between hubs) and δ (distribution cost rate).

The total cost per unit flow for a single O-D pair (i, j), where node i is assigned to hub h_i and node j to hub h_j, is given by the following equation:

$$f_{ij} = \gamma c_{ih_i} + \alpha c_{h_i h_j} + \delta c_{h_j j}.$$

Here, c_{uv} represents the (shortest path) distance between nodes u and v. A crucial assumption in hub network design is that the transfer cost rate α is significantly lower than the collection (γ) and distribution (δ) rates. This makes hub-to-hub transportation more cost-effective over a given distance, justifying the use of a hub-and-spoke network.

The objective of the USApHMP is to minimize the total cost for all O-D pairs, which is expressed as:

$$F_T = \sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij} f_{ij},$$

where t_{ij} denotes the given flow from node i to node j, $\forall i, j \in V$.

One of the earliest and most widely cited formulations was proposed by Campbell [3]. It uses binary variables $x_{ik}, i, k \in V$ such that $x_{kk} = 1$ means that node k is chosen to be hub, and $x_{ik} = 1$ implies that node i is assigned to hub k. By coventin it is requred that each hub node is assigned to it self. In addition, the model proposed by Campbell [3] employs continuous flow-routing variables $y_{ijkl} \geqslant 0$ indicating the fraction of flow from i to j routed via hubs k and l.

$$\min \sum_{i \in V} \sum_{j \in V} \sum_{k \in V} \sum_{l \in V} t_{ij} y_{ijkl} (\gamma c_{ik} + \alpha c_{kl} + \delta c_{lj})$$

subject to

$$\sum_{k \in V} x_{kk} = p \tag{1.1}$$

$$\sum_{k \in V} \sum_{l \in v} y_{ijkl} = 1 \quad \forall i, j \in V \tag{1.2}$$

$$\sum_{j \in V} \sum_{l \in V} (t_{ij} y_{ijkl} + t_{ji} y_{jilk}) = \sum_{j \in V} (t_{ij} + t_{ji}) x_{ik} \quad \forall i, k \in V \tag{1.3}$$

$$x_{ik} \leqslant x_{kk} \quad \forall i, k \in V \tag{1.4}$$

$$0 \leqslant y_{ijkl} \leqslant 1 \quad \forall i, j, k, l \in V$$
$$x_{ik} \in \{0, 1\} \quad \forall i, k \in V.$$

The constraints ensure that exactly p hubs are selected (Constraint (1.1)), that all flow between each origin-destination pair is fully routed through exactly one hub pair (Constraint (1.2)), and that each node is allocated to exactly one hub, with all inbound and outbound flows associated with the node routed exclusively via the allocated hub (Constraint (1.3)). Therefore, no node can simultaneously transfer flows through multiple hubs. Constraints (1.4) ensure that nodes are allocated only to those nodes designated as hubs. It is worth noting that, although the variables y_{ijkl} are defined as continuous within the interval $[0, 1]$, they take binary values in the optimal solution due to the single-allocation structure of the problem. This relaxation is introduced to improve the tractability of the model without affecting the optimal solution.

The uncapacitated single-allocation p-hub median problem (USApHMP) is computationally challenging and **NP-hard** optimization problem. The difficulty stems from the joint decision of (i) selecting p hub locations and (ii) assigning each non-hub node to exactly one hub, together with the quadratic interaction of origin–destination flows routed via (typically) two hubs. Moreover, even when hub locations are fixed, determining an optimal single-allocation remains an NP-hard problem [18].

Literature review. Although the formulation proposed by Campbell [3] accurately models the structure of feasible paths, it suffers from weak linear programming relaxations and an explosive growth in the number of continuous variables, which scales with $O(n^4)$. This inspired a variety of mathematical formulations over the last three decades. All models share the common objective of minimizing the total cost of routing origin–destination flows through a hub network, but they differ in terms of variables, constraints, and compactness. Below we highlight the most influential formulations and their main features.

Skorin-Kapov et al. [25] later proposed a strengthened mixed-integer linear program that adds consistency constraints linking routing and assignment variables. This tighter model still requires $O(n^4)$ variables, but the quality of its linear relaxation is much stronger; on benchmark CAB instances, the LP relaxation is often integral, allowing the problem to be solved optimally for moderate-sized instances using standard solvers. The authors solved optimally CAB instance with up to 25 nodes and 5 hubs.

In order to improve scalability, Ernst and Krishnamoorthy [7] developed a three-index multicommodity flow formulation. Here, flows are aggregated by origin, leading to $O(n^3)$ continuous variables and $O(n^2)$ binary variables. Balance constraints ensure correct routing of commodity flows through the hub network. This reduction in model size makes the approach more suitable for larger problem instances without sacrificing modeling accuracy. In their work, the authors considered CAB and AP data sets and succeeded to solve AP instances with 50 nodes optimally.

Ebery [6] introduced an even more compact model by defining auxiliary variables that directly capture the cost of transferring flows through each potential

hub. This formulation requires only $O(n^2)$ variables and constraints, making it computationally attractive for very large instances. However, it relies on big-M style inequalities, which may weaken the tightness of the linear relaxation depending on data scaling. In addition, the author limited the number of hubs to 2 or 3, and succeeded to optimally solve instances of AP data set with 200 nodes.

Table 1 summarizes the main characteristics of these formulations. In general, the choice of formulation depends on the problem size and the desired trade-off between modeling accuracy and computational performance. For small to medium instances, the strengthened formulation of Skorin-Kapov et al. [25] is recommended due to its tight relaxations. For larger instances, the multicommodity model of Ernst and Krishnamoorthy or the compact model of Ebery provide more practical alternatives.

Alongside exact methods, numerous heuristic and metaheuristic approaches have been developed. Early heuristics by O'Kelly [22] assigned nodes to their nearest hubs, while Klincewicz [14, 15] explored exchange-based neighborhoods, tabu search, and GRASP. Skorin-Kapov and Skorin-Kapov [24] further refined tabu search, showing small optimality gaps later confirmed by exact solutions. Other metaheuristics include simulated annealing [7], hybrid VNS with path relinking [23], and genetic algorithms (GA)[17, 26]. More recently, parallel GPU implementations of a genetic algorithm [1] have extended the solvability of USApHMP to instances with hundreds or even thousands of nodes.

In this chapter, we review the variable neighborhood search (VNS) heuristic originally proposed by Ilić et al. [12], which may be still regarded as a state-of-the-art approach for the USApHMP. In addition, we investigate whether solutions of

TABLE 1. Summary of main mathematical formulations for USApHMP

Formulation	**Variables**	**Constraints**	**Remarks**
Campbell (1994) [3]	n^4+n^2 (n^2 binary)	$O(n^4)$	Expressive but weak LP bound; no computational results reported.
Skorin-Kapov et al. (1996) [25]	n^4+n^2 (n^2 binary)	$2n^3+n^2+n+1$	Much tighter relaxation; CAB benchmark instances solved up to 25 nodes.
Ernst–Krishnamoorthy (1996) [7]	n^3+n^2 (n^2 binary)	$2n^2+n+1$	Multicommodity flow formulation; scalable; AP instances solved up to 50 nodes.
Ebery (2001) [6]	$2n^2$ (n^2 binary)	$2n^2+n+1$	Very compact formulation with weaker LP bound; large AP instances (up to 200 nodes) for small p values.

TABLE 2. Chronological summary of representative metaheuristics for the USApHMP.

Authors (Year)	Method	Dataset	Maximum size
O'Kelly (1987) [22]	Constructive heuristics	CAB	25 nodes
Klincewicz (1991–1992) [14, 15]	TS and GRASP	CAB	25 nodes
Skorin-Kapov & Skorin-Kapov (1994) [24]	Tabu search	CAB	25 nodes
Ernst & Krishnamoorthy (1996) [7]	Simulated annealing	CAB and AP	200 nodes
Pérez et al. (2007) [23]	Hybrid VNS with path relinking	CAB and AP	200 nodes
Kratica et al. (2007,2009) [17, 16]	(Hybrid) Genetic algorithms	CAB and AP	200 nodes
Ilić et al. (2010) [12]	General VNS	CAB,AP, PlanetLab and URAND	1000 nodes
Benaini et al. (2019) [1]	Parallel GA (GPU implementation)	CAB,AP, PlanetLab and URAND	6000 nodes.

the classical p-median problem can serve as effective starting points for our VNS heuristic applied to the USApHMP. This idea is motivated by the recent work of Brimberg et al.[2] who demonstrated a strong theoretical connection between the USApHMP and the p-median problem.

Table 2 highlights the steady methodological progress in solving the USApHMP over more than three decades. Early constructive and local-search-based heuristics were primarily designed to obtain feasible solutions for small benchmark instances. Subsequent metaheuristics, such as tabu search, simulated annealing, and genetic algorithms, significantly improved solution quality and robustness, often closing optimality gaps that earlier approaches could not address. More recent contributions use advanced neighborhood structures, and parallelization techniques allowing increasingly larger instances to be handled while maintaining or improving solution quality. Overall, the chronological evolution summarized in Table 2 shows a clear trajectory toward higher solution quality and improved scalability, with many later approaches consistently matching or improving previously reported best-known solutions on standard benchmark instances.

The remainder of the chapter is organized as follows. Section 2 revisits the relationship between the USApHMP and the p-median problem. Section 3 provides an overview of the VNS heuristic as described in [12]. Section 4 presents our computational experiments, including new results not previously reported in the literature. Finally, Section 5 concludes the chapter and outlines directions for future research.

2. Connection with the p-median problem

The p-median problem (PMP) is one of the most studied facility location models, where the task is to locate p facilities and assign each client to its closest facility in order to minimize total weighted distance [8, 9, 13]. A close relationship exists between PMP and hub location models, particularly the uncapacitated single-allocation p-hub median problem (USApHMP).

Recall that in USApHMP, the cost of sending a unit of flow from node i to node j via hubs h_i and h_j is

$$f_{ij} = \gamma c_{ih_i} + \alpha c_{h_i h_j} + \delta c_{h_j j},$$

with $\alpha < \gamma, \delta$ in most applications. In the limiting case where the transfer cost rate tends to zero, i.e., $\alpha \to 0$, hub-to-hub transport becomes essentially free compared to collection and distribution costs. In this situation, the cost function reduces to

$$f_{ij} \approx \gamma c_{ih_i} + \delta c_{h_j j},$$

and the optimal allocation strategy is to assign each node to its nearest open hub. The objective thus becomes minimizing the weighted sum of distances from nodes to their assigned hubs:

$$\min \sum_{i \in V} \sum_{k \in V} (\gamma O_i + \delta T_i)\, c_{ik}\, z_{ik},$$

where $O_i = \sum_{j \in V} t_{ij}$ is the total outflow from node i, $T_i = \sum_{j \in V} t_{ji}$ is the total inflow to i, and binary variable z_{ik} indicates assignment of node i to hub k. This is precisely the structure of a weighted p-median model, with weights $w_i = \gamma O_i + \delta T_i$.

Proposition 2.1. *In the limiting case $\alpha \to 0$, the USApHMP reduces to the classical weighted p-median problem.*

This connection explains why many heuristics for hub location adopt a "nearest-hub" assignment rule as a subproblem: when the hub network is highly efficient, optimal allocations coincide with those in a p-median model. Conversely, as the transfer cost α increases relative to γ and δ, the USApHMP no longer exhibits the p-median structure, since hub-to-hub distances begin to influence the optimal solution.

3. Description of the VNS procedure

In this section, we describe the Variable Neighborhood Search (VNS) procedure originally presented in [12]. We detail all its ingredients, including the solution representation, the method for generating an initial solution, the definition of neighborhood structures, the Variable Neighborhood Descent (VND) and shaking procedures, and the overall algorithmic flow.

For ease of reading, we provide a table summarizing the notation used in this paper. The notation is presented in Table 3.

3.1. Solution representation.

Suppose that locations (i.e. nodes) are labeled with integers between 1 and $n = |V|$ ($V = \{1, 2, \ldots, n\}$). Each solution is represented with an array (or vector) of length $n + p$ and consists of two parts. The last p numbers are indices of hubs in corresponding solution, and for $1 \leqslant i \leqslant n$ the i-th number is label of hub assigned (*allocated*) to location i (let us denote this number with $h[i]$). Thus, if one of hubs is located at node i ($i \in V$), we have that $h[i] = i$. We will also call each *solution assignment*.

TABLE 3. Summary of used notation.

Notation	Description
$t_{\text{in}}[i]$	Total inbound and flow for node i.
$t_{\text{out}}[i]$	Total outbound and flow for node i.
$t^c_{\text{in}}[node][hub]$	Total flow incoming to node *node* through hub *hub* in the current solution.
$t^n_{\text{in}}[node][hub]$	Total flow incoming to node *node* through hub *hub* in the next solution, i.e. in solution obtained by applying examined transformation/move.
$t^c_{\text{out}}[node][hub]$	Total flow originating from node *node* through hub *hub* in the current solution.
$t^n_{\text{out}}[node][hub]$	Total flow originating from node *node* through hub *hub* in the next solution, i.e. in solution obtained by applying examined transformation/move.
F^c_{in}	Cost of transporting the entire incoming flow to the examined node in the current solution.
F^n_{in}	Cost of transporting the entire incoming flow to the examined node in the next solution, i.e., the solution obtained by applying the examined move.
F^c_{out}	Cost of transporting the entire flow originating from the examined node in the current solution
F^n_{out}	Cost of transporting the entire flow originating from the examined node in the next solution, i.e., the solution obtained by applying the examined move.
$F^c[h']$	Cost of transporting the entire flow between the cluster of locations assigned to the examined hub h and the cluster of locations assigned to hub h'.
$F^n[h']$	Cost of transporting the entire flow between the cluster of locations assigned to the examined hub h and the cluster of locations assigned to hub h' after reassigning all locations previously assigned to hub h to the new hub *node*.

In addition, we maintain auxiliary data structures in which, for each node i, all other nodes are stored in non-decreasing order of their distance from i. We also precompute the total inbound and outbound flows for every node:

$$t_{\text{in}}[i] = \sum_{j=1}^{n} t_{j,i}, \qquad t_{\text{out}}[i] = \sum_{j=1}^{n} t_{i,j}.$$

These structures are used to explore the neighborhood structures and to perform shaking efficiently, as detailed in the corresponding sections.

3.2. Initial solution. The initial solution is generated as follows: a set of p hubs, H, is selected as a randomly generated subset of the set V, and then all non-hub nodes are assigned to their nearest hub. We then define a cluster as the set of nodes allocated to the same hub. Therefore, each solution to the problem contains p clusters.

3.3. Neighborhoods. Let us define a *local change* of an assignment as a transformation consisting of one of the following modifications to the assignment:

- changing the allocation of some non-hub node;
- replacing a single hub with a non-hub node, and reassigning the non-hub nodes from the corresponding cluster to the new hub.

Let A^c be the current assignment of locations, and let A^n be the new assignment obtained by performing one local change. For these two solution we maintain matrices t^c_{in} (t^n_{in}) and t^c_{out} (t^n_{out}) defined as

$$t^c_{\text{in}}\ [node][hub] = \sum_{h^c[i]=hub} t_{i,node},$$

$$t^n_{\text{in}}\ [node][hub] = \sum_{h^n[i]=hub} t_{i,node},$$

$$t^c_{\text{out}}\ [node][hub] = \sum_{h^c[i]=hub} t_{node,i},$$

$$t^n_{\text{out}}\ [node][hub] = \sum_{h^n[i]=hub} t_{node,i}.$$

This USApHMP naturally decomposes into two subproblems: the hub location problem and the allocation problem. The VNS exploits these two parts through use of three proposed different neighborhood transitions.

3.3.1. Allocate. The first neighborhood we denote with $N_{allocate}$, and corresponding local search method we call *allocate*. This local modification does not change the location of the hubs. For every non-hub *node*, we try to change its membership. This means that we have to process every hub and try to replace the assignment $(node, A^c[node])$ with $(node, A^n[node])$, leaving all other elements unchanged.

3.3.2. Alternate. The next neighborhood is $N_{alternate}$, and corresponding local search method is called *Alternate*. This local modification preserves all clusters and therefore matrices t^c_{in} and t^c_{out}. We change the location of the hub in one cluster and move to this neighborhood solution if the total flow is decreased. From the representation of the current assignment A^c, we can generate all clusters by accessing each element of the array exactly once.

3.3.3. Locate neighborhood. The third neighborhood is N_{locate}, and the corresponding local search serves to increase the diversity of solutions obtained. We select one cluster $\mathcal{C}$ with hub h and a location *node* that is not in this cluster. The selected node becomes a hub, and all locations from the cluster $\mathcal{C}$ are assigned to the closest hub (including the new one).

3.4. Shaking.

In the Shaking step we generate a random assignment from the k-th neighborhood of the current solution. The new assignment is constructed as follows. Choose at random k different hubs $h_1, h_2, \ldots, h_k$ and k different non–hub locations $l_1, l_2, \ldots, l_k$ and interchange h_i with l_i for each $1 \leqslant i \leqslant k$. This means that l_i becomes a hub in the cluster where h_i previously was, and the location h_i becomes a non-hub in the cluster where l_i was. This can be obviously done in $O(kn)$ time.

3.5. Local improvement.

3.5.1. Exploring Allocate neighborhood. A naive approach would be to iterate over all pairs $(node, hub)$, switch the assignment and recalculate the total flow. This will take $O(n^3 p)$ time, which is far too slow.

Assume that we want to swap location *node* from the hub $A^c[node]$ to $A^n[node]$. The contributions of traffic between locations i and j, where both i and j are different from location *node*, is unchanged, and therefore in order to get the new total flow, we have to calculate the following sums:

$$F^c_{\text{in}} = \sum_{i=1}^{n} t_{i,node}(\gamma c_{i,A^c[i]} + \alpha c_{A^c[i],A^c[node]} + \delta c_{A^c[node],node}),$$

$$F^n_{\text{in}} = \sum_{i=1}^{n} t_{i,node}(\gamma c_{i,A^n[i]} + \alpha c_{A^n[i],A^n[node]} + \delta c_{A^n[node],node}).$$

These sums represent the incoming flow to location *node* from all other locations in the current (F^c_{in}) and new (F^n_{in}) assignment. To the current total flow we have to add the difference $F^n_{\text{in}} - F^c_{\text{in}}$:

$$\begin{aligned} F^n_{\text{in}} - F^c_{\text{in}} = {} & \gamma \cdot t_{node,node}(c_{node,A^n[node]} - c_{node,A^c[node]}) \\ & + \alpha \cdot \sum_{i=1}^{n} t_{i,node}(c_{A^n[i],A^n[node]} - c_{A^c[i],A^c[node]}) \\ & + \delta(c_{A^n[node],node} - c_{A^c[node],node}) \cdot \sum_{i=1}^{n} t_{i,node} \end{aligned}$$

We can transform these summations and involve matrices t^c_{in} and t_{in}.

$$\begin{aligned} F^n_{\text{in}} - F^c_{\text{in}} = {} & \gamma \cdot t_{node,node}(c_{node,A^n[node]} - c_{node,A^c[node]}) \\ & + \alpha \cdot \sum_{h \in H} t^c_{\text{in}}[node][h](c_{h,A^n[node]} - c_{h,A^c[node]}) \\ & + \delta(c_{A^n[node],node} - c_{A^c[node],node}) \cdot t_{\text{in}}[node] \end{aligned}$$

Finally, we can calculate this change with one loop over the set of hubs. A similar relation holds for the outgoing flow from the location *node*:

$$\begin{aligned} F^n_{\text{out}} - F^c_{\text{out}} = \gamma(c_{node,A^n[node]} - c_{node,A^c[node]}) \cdot t_{\text{out}}[node] \\ + \alpha \cdot \sum_{h \in H} t^c_{\text{out}}[node][h](c_{A^n[node],h} - c_{A^c[node],h}) \\ + \delta \cdot t_{node,node}(c_{A^n[node],node} - c_{A^c[node],node}). \end{aligned}$$

If the current solution is improved, we update clusters and matrices t^c_{out} and t^c_{in} in $O(n)$, since the only changes are made for hubs $A^c[node]$ and $A^n[node]$.

Property 3.1. *The total time complexity of one iteration of local procedure Allocate is* $O(np^2)$.

3.5.2. Exploring Alternate neighborhood. Assume that we have a cluster $\mathcal{C}$ with m locations and hub h. If the cluster has only one location, do nothing. Otherwise, we can calculate the change in flow for cluster $\mathcal{C}'$ with a hub h' (location *node* becomes a new hub in $\mathcal{C}$).

$$\begin{aligned} F^n(h') - F^c(h') = \gamma \cdot \sum_{i \in \mathcal{C}} (c_{i,node} - c_{i,h}) t^c_{\text{out}}[i][h'] \\ + \alpha \cdot \sum_{i \in \mathcal{C}} (c_{node,h'} - c_{h,h'}) \cdot t^c_{\text{out}}[i][h'] \\ + \alpha \cdot \sum_{i \in \mathcal{C}} (c_{h',node} - c_{h',h}) \cdot t^c_{\text{in}}[i][h'] \\ + \delta \cdot \sum_{i \in \mathcal{C}} (c_{node,i} - c_{h,i}) t^c_{\text{in}}[i][h'] \end{aligned}$$

Similar calculations apply to net change for cluster $\mathcal{C}$ (with hub h in current solution):

$$\begin{aligned} F^n(h) - F^c(h) &= \sum_{i \in C} \sum_{j \in C} t_{ij} \cdot (\gamma \cdot c_{i,node} - \gamma \cdot c_{i,h}) + \sum_{i \in C} \sum_{j \in C} t_{ij} \cdot (\delta \cdot c_{node,j} - \delta \cdot c_{h,j}) \\ &= \gamma \cdot \sum_{i \in C} (c_{i,node} - c_{i,h}) \cdot \sum_{j \in C} t_{ij} + \delta \cdot \sum_{j \in C} (c_{node,j} - c_{h,j}) \cdot \sum_{i \in C} t_{ij} \\ &= \gamma \cdot \sum_{i \in C} (c_{i,node} - c_{i,h}) \cdot t^c_{\text{out}}[i][h] + \delta \cdot \sum_{j \in C} (c_{node,j} - c_{h,j}) \cdot t^c_{\text{in}}[j][h] \end{aligned}$$

Finally, the flow of the new assignment can be calculated by adding the sum of all differences $F^n(h') - F^c(h')$ for all hubs $h' \in H$ to the current flow.

The time complexity of this local search method for one cluster with m locations is $O(m^2 p)$, since for each node from this cluster we have $m \cdot p$ iterations to calculate flow change. See Algorithm 1 for details.

Property 3.2. *The time complexity of this local search method is* $O(p \cdot \sum_{h_i \in H} m_i^2)$, *where* m_i *represents the size of the cluster with hub* h_i.

3.5.3. Exploring Locate neighborhood. We can update matrices t^c_{in} and t^c_{out} in time $O(mn)$, as presented in Algorithm 2.

When calculating new total flow, we can observe *node* as a new hub, and then use the same technique as the one described in subsection Allocate. First, we have to compute the flow difference between locations from $\mathcal{C}$ and other clusters.

$$\Delta_1 = \gamma \cdot \sum_{i \in \mathcal{C}} \sum_{h \in H, h \notin \mathcal{C}} (c_{i,A^n[i]} - c_{i,A^c[i]}) \cdot t^c_{\text{out}}[i][h] + \alpha \cdot \sum_{i \in \mathcal{C}} \sum_{h \in H, h \notin \mathcal{C}} (c_{A^n[i],A^c[h]} - c_{A^c[i],A^c[h]}) \cdot t^c_{\text{out}}[i][h]$$

Algorithm 1: Local search *Alternate*

Input: Current assignment A^c and current flow $flow$
Output: True if new assignment is better than current
foreach *cluster in current assignment* **do**
  foreach *node in cluster* **do**
    if A^c *[node]* $\neq$ *node* **then**
      $newFlow = flow$;
      $hub = A^c[node]$;
      for *i=1* **to** p **do**
        if $i \neq hub$ **then**
          foreach *node j in cluster* **do**
            $newFlow = newFlow + \gamma \cdot (c_{j,node} - c_{j,hub}) \cdot t^c_{\text{out}}[j][i] + \alpha \cdot (c_{node,A^c[i]} - c_{hub,A^c[i]}) \cdot t^c_{\text{out}}[j][i]$;
            $newFlow = newFlow + \delta \cdot (c_{node,j} - c_{hub,j}) \cdot t^c_{\text{in}}[j][i] + \alpha \cdot (c_{A^c[i],node} - c_{A^c[i],hub}) \cdot t^c_{\text{in}}[j][i]$;
          end
        end
        else
          foreach *node j in cluster* **do**
            $newFlow = newFlow + \gamma \cdot (c_{j,node} - c_{j,hub}) \cdot t^c_{\text{out}}[j][i] + \delta \cdot (c_{node,j} - c_{hub,j}) \cdot t^c_{\text{in}}[j][i]$;
          end
        end
      end
      if $newFlow < flow$ **then**
        $flow = newFlow$;
        foreach *node j in cluster* **do**
          $A^c[j] = node$;
        end
        return *true*;
      end
    end
  end
end
return *false*;

$$+\delta\cdot\sum_{i\in\mathcal{C}}\sum_{h\in H,h\notin\mathcal{C}}(c_{A^n[i],i}-c_{A^c[i],i})\cdot t^c_{\text{in}}[i][h]$$
$$+\alpha\cdot\sum_{i\in\mathcal{C}}\sum_{h\in H,h\notin\mathcal{C}}(c_{A^c[h],A^n[i]}-c_{A^c[h],A^c[i]})\cdot t^c_{\text{in}}[i][h].$$

Let Δ_2 be the flow change between two locations in the cluster $\mathcal{C}$.

$$\Delta_2=\sum_{i\in\mathcal{C}}\sum_{j\in\mathcal{C}}(\gamma\cdot c_{i,A^n[i]}+\alpha\cdot c_{A^n[i],A^n[j]}+\delta\cdot c_{A^n[j],j})\cdot t_{i,j}$$
$$+\sum_{i\in\mathcal{C}}\sum_{j\in\mathcal{C}}(\gamma\cdot c_{i,A^c[i]}+\alpha\cdot c_{A^c[i],A^c[j]}+\delta\cdot c_{A^c[j],j})\cdot t_{i,j}$$

Finally, location *node* becomes a hub, and we have additional change in total flow:

$$\Delta_3=\gamma\cdot\sum_{i\in\mathcal{C},i\neq node}(c_{node,A^n[node]}-c_{node,A^c[node]})\cdot t_{node,i}$$
$$+\alpha\cdot\sum_{i\in\mathcal{C},i\neq node}(c_{A^n[node],A^n[i]}-c_{A^n[node],A^c[i]})\cdot t_{node,i}$$
$$+\delta\cdot\sum_{i\in\mathcal{C},i\neq node}(c_{A^n[node],node}-c_{A^c[node],node})\cdot t_{i,node}$$
$$+\alpha\cdot\sum_{i\in\mathcal{C},i\neq node}(c_{A^n[i],A^n[node]}-c_{A^c[i],A^n[node]})\cdot t_{i,node}$$
$$+\gamma\cdot(c_{node,A^n[node]}-c_{node,A^c[node]})\cdot t_{node,node}$$
$$+\delta\cdot(c_{A^n[node],node}-c_{A^c[node],node})\cdot t_{node,node}$$

The total difference is the sum of Δ_1, Δ_2 and Δ_3.

Property 3.3. *The time complexity of this local change method for one cluster of size m is $O(pn(n+m^2+mp))$.*

Algorithm 2: Update matrices t^c_{in} and t^c_{out}.

Input: Current and new assignments A^c and A^n, cluster C, and new hub for this cluster *node*

for *i=1* **to** *n* **do**
- $t^c_{\text{out}}[i][A^c[node]]=t^c_{\text{out}}[i][A^c[node]]-t_{i,node}$;
- $t^c_{\text{in}}[i][A^c[node]]=t^c_{\text{in}}[i][A^c[node]]-t_{node,i}$;
- $t^c_{\text{out}}[i][node]=t_{i,node}$;
- $t^c_{\text{in}}[i][node]=t_{node,i}$;

end

foreach *node j in cluster* **do**
- **for** *i=1* **to** *n* **do**
  - $t^c_{\text{out}}[i][A^n[j]]=t^c_{\text{out}}[i][A^n[j]]+t_{i,j}$;
  - $t^c_{\text{in}}[i][A^n[j]]=t^c_{\text{in}}[i][A^n[j]]+t_{j,i}$;
- **end**

end

3.5.4. Variable neighborhood descent. VND is a deterministic variant of VNS. It uses $\ell_{\max}$ ($\ell_{\max} \geqslant 2$) neighborhood structures in the local search. In designing an efficient VND heuristic, the following problem specific questions naturally arise [21]: (i) What neighborhoods will be used? (ii) What will be their order in the search? (iii) What search strategy will be used in changing neighborhoods? Here we discuss question (iii).

Sequential VND. The usual strategy for exploration of several neighborhoods within deterministic local search is sequential (Seq-VND) [11, 10]:

(1) Make an order of all $\ell_{\max}$ neighborhoods that will be used in the search (usually in nondecreasing order of their cardinality);
(2) Perform local searches with them respecting their order;
(3) The first time a better solution is found, return to the first neighborhood in the list;
(4) Stop when there is no better solution in the last neighborhood.

The final solution obtained is a local minimum with respect to all $\ell_{\max}$ neighborhood structures. In addition, it is clear that the cardinality of the Seq-VND neighborhood is equal to the sum of cardinalities of all its components. The pseudo-code for solving USApHMP by GVNS with Seq-VND may be found in Algorithm 4 below.

Nested VND. Another possibility is to explore the neighborhoods with a nested strategy (Nest-VND). The pure nested strategy may be described generally as follows. Assume that we define two neighborhood structures ($\ell_{\max} = 2$). In nested VND we in fact perform local search with respect to the first neighborhood in any point of the second. If both neighborhoods have cardinality n, then with Seq-VND we visit $2n$ points, but in Nest-VND we visit n^2 points. Therefore, pure Nest-VND neighborhoods are much larger.

It is important to note that a local minimum in Seq-VND will not necessarily be a local minimum in Nest-VND, but the opposite holds if the Nest-VND is designed to contain the Seq-VND neighborhood. Thus, the number of local minima in Nest-VND will in general be smaller than in Seq-VND. Of course, the larger neighborhood will require more computational time. See Algorithm 3 for a detailed pseudo-code of Nest-VND.

The number of solutions visited by Nest-VND is very large. It is equal to the product of the cardinalities of all its components.

Mixed VND. There are several ways to reduce this large nested set of neighborhood points. One way is to reduce the size of the largest neighborhoods, or, after b (a parameter) neighborhoods, switch from a nested to a sequential strategy. The reduced Nested VND will be called Mix-VND.

From the Algorithm 3 that presents a pure nested VND strategy, it is clear that we can interrupt nesting at some level b ($1 \leqslant b \leqslant \ell_{\max}$) and continue with the list of the remaining neighborhoods in sequential manner. Thus, if $b = 1$, we get pure sequential VND. If $b = \ell_{\max}$ we get pure nested VND. Since nested VND intensifies the search in a deterministic way, boost parameter b may be seen as a balance between intensification and diversification in deterministic local search with several neighborhoods. For solving USApHMP we develop two mixed VND heuristics

Algorithm 3: Nest-VND: Nested Variable neighborhood descent heuristic.

Make an order of all $\ell_{\max} \geqslant 2$ neighborhoods that will be used in the search;
Find an initial solution x; $x_{opt} = x$, $f_{opt} = f(x)$;
$\ell = \ell_{\max}$;
repeat
  if all solutions from *neighborhood* ℓ *are visited* **then** set $\ell = \ell + 1$;
  if there is any non visited solution $x_\ell \in N_\ell(x)$ and $\ell \geqslant 2$ **then** move $(x_{cur} = x_\ell)$ and set $\ell = \ell - 1$;
  if $\ell = 1$ **then**
    Find objective function value $f_{cur} = f(x_{cur})$;
    if $f_{cur} < f_{opt}$ **then** $x_{opt} = x_{cur}$, $f_{opt} = f_{cur}$;
  end
until ($\ell = \ell_{\max} + 1$, i.e., there are no more points in the last neighborhood);

within GVNS: (i) $b = 2$ and several random points from the *Locate* neighborhood are selected (Mix-VND); (ii) $b = 2$ and the complete *Locate* neighborhood is explored (Mix-VND2). From each point in *Locate* a search is carried out sequentially with the *Allocate* and *Alternate* neighborhoods.

Since the Mix-VND2 procedure is relatively time consuming, we opted not to use it within the GVNS heuristic.

3.6. GVNS heuristics for USApHMP.

Two versions of GVNS were investigated in [12]. The first uses a Sequential VND in the local search step, where the neighborhoods are examined in the order *Allocate*, *Alternate* and *Locate*. The order of exploring neighborhoods within Sequential VND is determined by taking into account:

- the cardinality of the neighborhoods,
- the complexity of computing the change in the objective function value for solutions belonging to the corresponding neighborhood,
- the complexity of exploring the complete neighborhood.

Any time an improvement is found, the search returns to the *Allocate* neighborhood. As noted above, the local search is terminated when no further improvement is found in the *Locate* neighborhood. The GVNS heuristic with Seq-VND (denoted as Seq-GVNS) is given in Algorithm 4.

The second version of GVNS uses Mix-VND described above in the local search step. The pseudo-code for our implementation of GVNS with Mix-VND (denoted as Mix-GVNS) is found in Algorithm 5.

4. Computational results

According to the findings in [12], the GVNS equipped with the mixed VND scheme and parameter $b = 2$ (Algorithm 5) consistently outperforms the variant that uses the sequential VND scheme (Algorithm 4). Consequently, in what follows we report results only for the GVNS with the mixed-VND with $b = 2$ and, for brevity, refer to it simply as *GVNS*.

Algorithm 4: GVNS heuristic with Sequential VND

```
Select k_max and it_max; Generate initial solution A^c;
k = 1;
for i=1 to it_max do
    Generate a random assignment A^c in k-th neighborhood;
    j = 1;
    while j < 4 do
        if j=1 then
            Apply local search Allocate() on current assignment;
        else if j=2 then
            Apply local search Alternate() on current assignment;
        else if j=3 then
            Apply local search Locate() on current assignment;
        j = j + 1;
        if there is an improvement then
            Update current assignment A^c; j = 1;
        end
    end
    k = k + 1;
    if A^c is better than solution then
        Update solution and set k = 1;
    end
    if k = k_max + 1 then
        k = 1;
    end
end
```

The proposed GVNS heuristic is compared against state-of-the-art algorithms on standard benchmark sets. All tests were carried out on an Intel Core 2 Duo E6750 2.66 GHz with 2 GB RAM, running Linux. The GVNS heuristic is implemented in C++and the memory footprint of the algorithm is $O(n^2)$ (below 1 MB even on the largest instances). The shaking parameter is $k_{\max} = p$ and the stopping rule is $n/2$ consecutive iterations without improvement. For further details, see [12].

4.1. Test instances. We present results for two groups of test instances.

- The AP data set is derived from a real-world hub location problem for Australia Post (introduced for the first in [7]). The largest instance consists of 200 nodes (representing postcode districts) with parameters $\chi = 3$, $\alpha = 0.75$, and $\delta = 2$. Smaller instances with 10, 20, 25, 50, and 100 nodes are obtained by aggregating the largest initial instance. The number of hubs (mail sorting or consolidation centers) in the tested instances is up to 20.
- The URAND data set consists of instances with up to 400 nodes and was created by Meyer, Ernst, and Krishnamoorthy [20]. For this data set, the x and y coordinates are randomly generated from a uniform distribution on $[0, 100000]$. Different instances are created by selecting subsets with

$n \in 100, 200, 300, 400$ nodes. The flows between nodes are also randomly generated. In addition, larger URAND instances with $n = 1000$ were proposed in [12].

Algorithm 5: GVNS heuristic with Mixed VND and $b = 2$

```
Select k_max and it_max; Generate initial solution A^c;
k = 1;
for i=1 to it_max do
    Generate a random assignment A^c in k-th neighborhood;
    j = ℓ_max; improved = true;
    while improved do
        improved = false;
        foreach assignment A^n from N_j(A^c) (Locate neighborhood) do
            perform Sequential VND from A^n (A'^n = SeqVND(A^n));
            if A'^n is better than A^c then
                A^c = A'^n; improved = true; exit;
            end
        end
    end
    if A^c is better than solution then
        Update solution and set k = 1;
    else
        k = k + 1;
        if k > k_max then
            k = 1;
end
```

4.2. Strategies for generating initial solution. We evaluate two initialization strategies inside GVNS: (i) a uniformly random hub set with nearest-hub assignment, and (ii) the solution of the corresponding weighted p-median as the starting assignment. For each instance we perform 20 multistart runs and report best/average/worst objective values and average CPU time (seconds).

Across AP, and URAND instances, both starting strategies consistently lead GVNS to the same best solutions; the p-median start does not systematically dominate and, depending on n and p, either option may be slightly faster or slower in wall-clock time. See Tables 4 and 6 for details. Given this, we retain the simple random initialization in subsequent comparisons for generality. However, the fact that the single p-median starting solution gives the best (or very near best) solution from 20 random restarts would suggest that the p-median approach may be capable of finding good starting solutions. Further investigation is warranted.

4.3. Main computational results. We compare GVNS against Simulated Annealing (SA) [7], TABUHUB[24], VNS with Path-Relinking (VNS-PR)[24], several Genetic Algorithms (GAHUB1,GAHUB2, HGA1,HGA12)[17, 16], and SPSH [27]

on AP instances; and against GAHUB2 [17] on URAND instances. As noted before, each line aggregates 20 restarts.

In Table 8 (AP instances), each method is reported with two columns: *Best* (lowest objective value obtained for the instance) and *Time* (CPU seconds on the reference machine shown in the column label: SA on Alpha 200 MHz; TABUHUB on Intel 1.4 GHz; VNS-PR on Intel 2.4 GHz; GAHUB1/2 on AMD 1.33 GHz; SPSH on Intel 2.8 GHz; GVNS on Intel Core 2 Duo 2.66 GHz). Underlining marks the best objective among all methods for that (n, p), while boldface (with underlining) indicates a new best-known value established by the GVNS. Because hardware differs across studies, times are not directly comparable and should be interpreted qualitatively; "—" denotes unavailable data.

In Table 9, the GVNS columns are: *Best* (minimum objective value over 20 independent restarts), *#Best* (number of restarts that attain that minimum), *Avg. gap* and *Max. gap* (mean and maximum percentage deviations from the *Best* over the 20 runs, where for run r with objective z_r and $z^* = \min_r z_r$, $\text{gap}_r = 100\,(z_r - z^*)/z^*$), and *Avg. time* (average CPU time (in seconds) required by GVNS to attain the best value in each run). For GAHUB2 we report its *Best* objective and *Avg. time*.

All CPU times are taken from the corresponding papers, which means that they are not directly comparable. However, as can be seen from the bibliography, most of the compared methods were implemented almost twenty years ago, and it is not possible to determine a scaling factor for the computers used in order to make a fair comparison.

On AP instances, GVNS matches or improves the best-known solutions across all reported settings and attains the best value at $(n, p) = (200, 20)$ (Table 8). Average gaps are zero on URAND for all n and p tested except one, while average running times are substantially lower than those reported for GAHUB2 (Table 9). This demonstrates high stability and the robustness of the proposed GVNS. In total, GVNS succeeded to establish 15 new best-known solutions on the set of 28 instances. Overall, the evidence indicates that GVNS is competitive with, and often superior to, state-of-the-art heuristics in both solution quality and robustness, with reasonable running times on large problems.

TABLE 4. GVNS with random vs. p-median initialization on AP instances

n	p	**Random initial**				**p-median initial**	
		Best	Avg.	Worst	Time	Best	Time
10	2	167493.065	167493.065	167493.065	0.0000	167493.065	0.0000
	3	136008.126	136008.126	136008.126	0.0000	136008.126	0.0000
	4	112396.068	112396.068	112396.068	0.0004	112396.068	0.0000
	5	91105.371	91105.371	91105.371	0.0000	91105.371	0.0000
20	2	172816.690	172816.690	172816.690	0.0000	172816.690	0.0000
	3	151533.084	151533.084	151533.084	0.0002	151533.084	0.0040
	4	135624.884	135624.884	135624.884	0.0016	135624.884	0.0000
	5	123130.095	123130.095	123130.095	0.0024	123130.095	0.0160
	10	81870.549	81870.549	81870.549	0.0046	81870.549	0.0000
25	2	175541.977	175541.977	175541.977	0.0004	175541.977	0.0000
	3	155256.323	155256.323	155256.323	0.0006	155256.323	0.0040
	4	139197.169	139197.169	139197.169	0.0062	139197.169	0.0040
	5	123574.289	123574.289	123574.289	0.0110	123574.289	0.0080
	10	87944.748	87944.748	87944.748	0.0366	87944.748	0.0320
40	2	177471.674	177471.674	177471.674	0.0000	177471.674	0.0000
	3	158830.545	158830.545	158830.545	0.0006	158830.545	0.0000
	4	143968.876	143968.876	143968.876	0.0016	143968.876	0.0000
	5	134264.967	134264.967	134264.967	0.0140	134264.967	0.0120
	10	101773.708	101773.708	101773.708	0.0502	101773.708	0.0240
50	2	178484.286	178484.286	178484.286	0.0002	178484.286	0.0040
	3	158569.933	158569.933	158569.933	0.0022	158569.933	0.0000
	4	143378.046	143378.046	143378.046	0.0070	143378.046	0.0120
	5	132366.953	132366.953	132366.953	0.0170	132366.953	0.0240
	10	102706.857	102706.857	102706.857	0.1896	102706.857	0.2520
100	2	180223.801	180223.801	180223.801	0.0028	180223.801	0.0040
	3	160847.001	160847.001	160847.001	0.0054	160847.001	0.0040
	4	145896.578	145896.578	145896.578	0.0178	145896.578	0.0120
	5	136929.444	136929.444	136929.444	0.0552	136929.444	0.0280
	10	106469.566	106469.566	106469.566	0.2734	106469.566	0.1000
	15	90533.523	90533.523	90533.523	1.1630	90533.523	0.9440
	20	80270.962	80270.962	80270.962	1.7328	80270.962	2.1880
200	2	182459.254	182459.254	182459.254	0.0056	182459.254	0.0040
	3	162887.031	162887.031	162887.031	0.0090	162887.031	0.0040
	4	147767.303	147767.303	147767.303	0.0792	147767.303	0.0760
	5	140062.647	140062.647	140062.647	2.0052	140062.647	2.7960
	10	110147.657	110147.657	110147.657	3.7962	110147.657	1.6400
	15	94459.201	94459.201	94459.201	14.7700	94459.201	19.9920
	20	84955.367	84955.367	84955.367	8.6730	84955.367	7.2840

TABLE 5. GVNS with random vs. p-median initialization on AP instances

n	p	**Random initial**				**p-median initial**			
		Best	Avg.	Worst	Time	Best	Avg.	Worst	Time
10	2	167493.065	167493.065	167493.065	0.0000	167493.065	167493.065	167493.065	0.0000
	3	136008.126	136008.126	136008.126	0.0000	136008.126	136008.126	136008.126	0.0002
	4	112396.068	112396.068	112396.068	0.0004	112396.068	112396.068	112396.068	0.0000
	5	91105.371	91105.371	91105.371	0.0000	91105.371	91105.371	91105.371	0.0002
Avg.10		126750.657	126750.657	126750.657	0.0001	126750.657	126750.657	126750.657	0.0001
20	2	172816.690	172816.690	172816.690	0.0000	172816.690	172816.690	172816.690	0.0000
	3	151533.084	151533.084	151533.084	0.0002	151533.084	151533.084	151533.084	0.0002
	4	135624.884	135624.884	135624.884	0.0016	135624.884	135624.884	135624.884	0.0004
	5	123130.095	123130.095	123130.095	0.0024	123130.095	123130.095	123130.095	0.0070
	10	81870.549	81870.549	81870.549	0.0046	81870.549	81870.549	81870.549	0.0028
Avg.20		132995.060	132995.060	132995.060	0.0018	132995.060	132995.060	132995.060	0.0021
25	2	175541.977	175541.977	175541.977	0.0004	175541.977	175541.977	175541.977	0.0000
	3	155256.323	155256.323	155256.323	0.0006	155256.323	155256.323	155256.323	0.0012
	4	139197.169	139197.169	139197.169	0.0062	139197.169	139197.169	139197.169	0.0036
	5	123574.289	123574.289	123574.289	0.0110	123574.289	123574.289	123574.289	0.0108
	10	87944.748	87944.748	87944.748	0.0366	87944.748	87944.748	87944.748	0.0254
Avg.25		136302.901	136302.901	136302.901	0.0110	136302.901	136302.901	136302.901	0.0082
40	2	177471.674	177471.674	177471.674	0.0000	177471.674	177471.674	177471.674	0.0002
	3	158830.545	158830.545	158830.545	0.0006	158830.545	158830.545	158830.545	0.0004
	4	143968.876	143968.876	143968.876	0.0016	143968.876	143968.876	143968.876	0.0008
	5	134264.967	134264.967	134264.967	0.0140	134264.967	134264.967	134264.967	0.0130
	10	101773.708	101773.708	101773.708	0.0502	101773.708	101773.708	101773.708	0.0260
Avg.40		143261.954	143261.954	143261.954	0.0133	143261.954	143261.954	143261.954	0.0081
50	2	178484.286	178484.286	178484.286	0.0002	178484.286	178484.286	178484.286	0.0002
	3	158569.933	158569.933	158569.933	0.0022	158569.933	158569.933	158569.933	0.0016
	4	143378.046	143378.046	143378.046	0.0070	143378.046	143378.046	143378.046	0.0090
	5	132366.953	132366.953	132366.953	0.0170	132366.953	132366.953	132366.953	0.0132
	10	102706.857	102706.857	102706.857	0.1896	102706.857	102706.857	102706.857	0.1946
Avg.50		143101.215	143101.215	143101.215	0.0432	143101.215	143101.215	143101.215	0.0437
100	2	180223.801	180223.801	180223.801	0.0028	180223.801	180223.801	180223.801	0.0024
	3	160847.001	160847.001	160847.001	0.0054	160847.001	160847.001	160847.001	0.0046
	4	145896.578	145896.578	145896.578	0.0178	145896.578	145896.578	145896.578	0.0248
	5	136929.444	136929.444	136929.444	0.0552	136929.444	136929.444	136929.444	0.0748
	10	106469.566	106469.566	106469.566	0.2734	106469.566	106469.566	106469.566	0.2820
	15	90533.523	90533.523	90533.523	1.1630	90533.523	90533.523	90533.523	1.0436
	20	80270.962	80270.962	80270.962	1.7328	80270.962	80270.962	80270.962	1.6882
Avg.100		128738.697	128738.697	128738.697	0.4643	128738.697	128738.697	128738.697	0.4458
200	2	182459.254	182459.254	182459.254	0.0056	182459.254	182459.254	182459.254	0.0058
	3	162887.031	162887.031	162887.031	0.0090	162887.031	162887.031	162887.031	0.0086
	4	147767.303	147767.303	147767.303	0.0792	147767.303	147767.303	147767.303	0.0804
	5	140062.647	140062.647	140062.647	2.0052	140062.647	140062.647	140062.647	2.6168
	10	110147.657	110147.657	110147.657	3.7962	110147.657	110147.657	110147.657	4.1996
	15	94459.201	94459.201	94459.201	14.7700	94459.201	94459.201	94459.201	14.2124
	20	84955.367	84955.367	84955.367	8.6730	84955.367	84955.367	84955.367	5.6380
Avg.200		131819.780	131819.780	131819.780	4.1912	131819.780	131819.780	131819.780	3.8231
Avg.Compl.		134453.095	134453.095	134453.095	0.8667	134453.095	134453.095	134453.095	0.7945

TABLE 6. GVNS with random vs. p-median initialization on URAND instances

n	p	**Random initial**				**p-median initial**	
		Best	Avg.	Worst	Time	Best	Time
100	2	36930.308	36930.308	36930.308	0.0190	36930.308	0.0720
	3	34532.877	34532.877	34532.877	0.0242	34532.877	0.0400
	4	32608.278	32608.278	32608.278	0.0196	32608.278	0.0200
	5	31107.697	31107.697	31107.697	0.0852	31107.697	0.0240
	10	27058.397	27058.397	27058.397	0.2556	27058.397	0.1320
	15	25408.560	25408.560	25408.560	0.9776	25408.560	0.2480
	20	24377.649	24377.649	24377.649	2.9094	24377.649	3.1840
200	2	148235.452	148235.452	148235.452	0.0164	148235.452	0.0080
	3	139223.245	139223.245	139223.245	0.0422	139223.245	0.0520
	4	132676.885	132676.885	132676.885	0.1540	132676.885	0.0720
	5	127220.021	127220.021	127220.021	0.2352	127220.021	0.1400
	10	112359.211	112359.211	112359.211	0.7008	112359.211	0.0600
	15	105690.518	105690.518	105690.518	20.2682	105690.518	42.4840
	20	102022.325	102022.325	102022.325	34.5710	102022.325	15.2840
300	2	328702.418	328702.418	328702.418	0.1860	328702.418	0.2200
	3	308765.082	308765.082	308765.082	0.4758	308765.082	0.2920
	4	293636.805	293636.805	293636.805	0.4748	293636.805	0.2200
	5	282116.876	282116.876	282116.876	0.2542	282116.876	0.0480
	10	251393.296	251393.296	251393.296	5.4882	251393.296	2.6680
	15	236781.767	236781.767	236781.767	11.9066	236781.767	16.2280
	20	228005.187	228005.229	228006.029	73.8886	228005.187	204.0880
400	2	579982.350	579982.350	579982.350	0.2428	579982.350	0.1000
	3	543717.321	543717.321	543717.321	1.4130	543717.321	0.5440
	4	519217.482	519217.482	519217.482	1.5506	519217.482	0.1400
	5	501421.517	501421.517	501421.517	0.6096	501421.517	0.3960
	10	446361.101	446361.101	446361.101	6.9808	446361.101	4.5120
	15	422284.779	422284.779	422284.779	112.7628	422284.779	178.1480
	20	407110.510	407116.318	407179.943	181.3304	407113.137	187.2080
1000	2	3644705.838	3644705.838	3644705.838	4.3680	3644705.838	5.5160
	3	3397386.836	3397386.836	3397386.836	16.3708	3397386.836	39.5040
	4	3203013.332	3203013.332	3203013.332	16.8508	3203013.332	37.1200
	5	3117594.218	3117594.218	3117594.218	124.7838	3117594.218	127.6760
	10	2810664.031	2810730.599	2811006.313	527.9880	2810664.031	475.0240
	15	2662102.980	2662312.157	2663008.031	488.0978	2662231.495	39.8600
	20	2563483.469	2563722.474	2564556.671	635.6346	2563545.307	711.7200

TABLE 7. GVNS with random vs. p-median initialization on URAND instances

n	p	**Random initial**				**p-median initial**			
		Best	Avg.	Worst	Time	Best	Avg.	Worst	Time
100	2	36930.308	36930.308	36930.308	0.0190	36930.308	36930.308	36930.308	0.0334
	3	34532.877	34532.877	34532.877	0.0242	34532.877	34532.877	34532.877	0.0088
	4	32608.278	32608.278	32608.278	0.0196	32608.278	32608.278	32608.278	0.0194
	5	31107.697	31107.697	31107.697	0.0852	31107.697	31107.697	31107.697	0.0572
	10	27058.397	27058.397	27058.397	0.2556	27058.397	27058.397	27058.397	0.2482
	15	25408.560	25408.560	25408.560	0.9776	25408.560	25408.560	25408.560	1.0750
	20	24377.649	24377.649	24377.649	2.9094	24377.649	24377.649	24377.649	2.4558
Avg.	100	30289.109	30289.109	30289.109	0.6129	30289.109	30289.109	30289.109	0.5568
200	2	148235.452	148235.452	148235.452	0.0164	148235.452	148235.452	148235.452	0.0128
	3	139223.245	139223.245	139223.245	0.0422	139223.245	139223.245	139223.245	0.0232
	4	132676.885	132676.885	132676.885	0.1540	132676.885	132676.885	132676.885	0.1552
	5	127220.021	127220.021	127220.021	0.2352	127220.021	127220.021	127220.021	0.3902
	10	112359.211	112359.211	112359.211	0.7008	112359.211	112359.211	112359.211	0.7836
	15	105690.518	105690.518	105690.518	20.2682	105690.518	105690.518	105690.518	18.4790
	20	102022.325	102022.325	102022.325	34.5710	102022.325	102022.325	102022.325	29.7680
Avg.	200	123918.237	123918.237	123918.237	7.9983	123918.237	123918.237	123918.237	7.0874
300	2	328702.418	328702.418	328702.418	0.1860	328702.418	328702.418	328702.418	0.1776
	3	308765.082	308765.082	308765.082	0.4758	308765.082	308765.082	308765.082	0.4982
	4	293636.805	293636.805	293636.805	0.4748	293636.805	293636.805	293636.805	0.4378
	5	282116.876	282116.876	282116.876	0.2542	282116.876	282116.876	282116.876	0.1060
	10	251393.296	251393.296	251393.296	5.4882	251393.296	251393.296	251393.296	4.2148
	15	236781.767	236781.767	236781.767	11.9066	236781.767	236781.767	236781.767	13.1284
	20	228005.187	228005.229	228006.029	73.8886	228005.187	228005.187	228005.187	91.8272
Avg.	300	275628.776	275628.782	275628.896	13.2392	275628.776	275628.776	275628.776	15.7700
400	2	579982.350	579982.350	579982.350	0.2428	579982.350	579982.350	579982.350	0.1450
	3	543717.321	543717.321	543717.321	1.4130	543717.321	543717.321	543717.321	1.0210
	4	519217.482	519217.482	519217.482	1.5506	519217.482	519217.482	519217.482	0.9212
	5	501421.517	501421.517	501421.517	0.6096	501421.517	501421.517	501421.517	0.7932
	10	446361.101	446361.101	446361.101	6.9808	446361.101	446361.101	446361.101	9.7864
	15	422284.779	422284.779	422284.779	112.7628	422284.779	422284.779	422284.779	103.7890
	20	407110.510	407116.318	407179.943	181.3304	407110.510	407116.737	407194.481	205.9304
Avg.	400	488585.008	488585.838	488594.927	43.5557	488585.008	488585.898	488597.004	46.0552
1000	2	3644705.838	3644705.838	3644705.838	4.3680	3644705.838	3644705.838	3644705.838	4.4364
	3	3397386.836	3397386.836	3397386.836	16.3708	3397386.836	3397386.836	3397386.836	20.0248
	4	3203013.332	3203013.332	3203013.332	16.8508	3203013.332	3203013.332	3203013.332	33.6526
	5	3117594.218	3117594.218	3117594.218	124.7838	3117594.218	3117594.218	3117594.218	67.5494
	10	2810664.031	2810730.599	2811006.313	527.9880	2810664.031	2810664.031	2810664.031	441.0732
	15	2662102.980	2662312.157	2663008.031	488.0978	2662102.980	2662199.898	2662624.096	587.9946
	20	2563483.469	2563722.474	2564556.671	635.6346	2563483.469	2563674.117	2564238.411	547.4100
Avg.	1000	3056992.958	3057066.493	3057324.463	259.1563	3056992.958	3057034.039	3057175.252	243.1630
Avg.Compl.		795082.818	795097.692	795151.126	64.9125	795082.818	795091.212	795121.676	62.5265

TABLE 8. Comparison on larger AP instances

n	p	SA		TABUHUB		VNS-PR		GAHUB1		GAHUB2		SPSH		GVNS	
		Best solution	Alpha 200 MHz	Best solution	Intel 1.4 GHz	Best solution	Intel 2.4 GHz	Best solution	AMD 1.33 GHz	Best solution	AMD 1.33 GHz	Best solution	Intel 2.8 GHz	Best solution	Intel 2.66 GHz
50	5	132366.953	19.5	132366.953	1240	132366.953	0.76	132366.953	0.733	132366.953	1.053	132366.953	2.14	132366.953	0.012
100	5	136929.444	80.6	136929.444	805	136929.444	0.48	136929.444	13.585	136929.444	22.108	136929.444	11.19	136929.444	0.077
100	10	106469.566	161.9	106469.566	1259	106469.566	0.91	106829.151	32.590	106469.566	40.733	106469.566	25.81	106469.566	0.666
100	15	90605.10	279.8	90605.10	1480	90605.10	1.32	90534.785	48.500	90533.523	57.453	90533.523	63.01	90533.523	3.219
100	20	80682.71	522.7	80682.71	2330	80682.71	1.92	80471.845	81.860	80270.962	79.200	80270.962	77.87	80270.962	3.572
200	5	140409.41	399.7	—	—	140318.36	0.86	140450.089	63.417	140175.645	169.733	140062.647	58.10	140062.647	5.157
200	10	111088.33	776.6	—	—	110955.27	1.87	110648.724	135.878	110147.657	259.142	110147.657	188.05	110147.657	5.600
200	15	95460.54	1105.3	—	—	95111.06	2.11	95857.693	199.222	94496.406	313.017	94459.201	305.05	94459.201	17.656
200	20	85560.39	1555.9	—	—	85439.92	3.20	86069.213	297.513	85129.343	374.751	85129.343	417.41	**84955.328**	**12.979**

TABLE 9. URAND instances: GVNS vs. GAHUB2

		GVNS					GAHUB2	
n	p	Best	#Best	Avg. gap	Max. gap	Avg. time	Best	Avg. time
100	2	36930.31	20	0.000	0.000	0.0375	36930.31	3.262
	3	34532.88	20	0.000	0.000	0.0265	34532.88	4.329
	4	32608.28	20	0.000	0.000	0.0245	32608.28	5.118
	5	31107.70	20	0.000	0.000	0.1135	31107.70	6.801
	10	**27058.40**	20	0.000	0.000	0.4695	27067.76	13.753
	15	25408.56	20	0.000	0.000	2.7925	25408.56	17.272
	20	**24377.65**	20	0.000	0.000	7.3640	24379.85	28.25
200	2	148235.45	20	0.000	0.000	0.0175	148235.45	19.568
	3	139223.25	20	0.000	0.000	0.0575	139223.25	32.118
	4	**132676.89**	20	0.000	0.000	0.3920	132680.78	30.95
	5	127220.02	20	0.000	0.000	0.6895	127220.02	38.842
	10	**112539.21**	20	0.000	0.000	4.5300	112583.36	59.379
	15	**105690.52**	20	0.000	0.000	37.3460	105819.21	75.585
	20	**102022.32**	20	0.000	0.000	68.6685	102088.38	92.299
300	2	328702.42	20	0.000	0.000	0.2215	328702.42	71.761
	3	308765.08	20	0.000	0.000	0.9175	308765.08	110.821
	4	293636.81	20	0.000	0.000	1.6100	293636.81	122.697
	5	**282116.88**	20	0.000	0.000	0.5060	282212.66	137.904
	10	**251393.30**	20	0.000	0.000	12.8275	251834.24	168.939
	15	**236781.77**	20	0.000	0.000	27.6640	237139.70	196.837
	20	**228005.19**	20	0.000	0.000	153.2925	228506.09	228.842
400	2	579982.35	20	0.000	0.000	0.1735	579982.35	183.401
	3	543717.32	20	0.000	0.000	1.7115	543717.32	228.983
	4	**519217.48**	20	0.000	0.000	2.2275	519222.55	241.003
	5	**501421.52**	20	0.000	0.000	1.4730	501934.49	261.606
	10	**446361.10**	20	0.000	0.000	16.8700	447755.36	319.422
	15	**422284.78**	18	0.000	0.001	111.4295	423659.74	364.876
	20	**407110.51**	10	0.005	0.037	228.8615	408184.36	442.021

5. Conclusions

We describe a general variable neighborhood search (GVNS) heuristic to solve the uncapacitated single-allocation p-hub median problem (USApHMP). The improvement step uses Variable neighborhood descent (VND) with three neighborhoods: *Allocate*, *Alternate* and *Locate*, denoted respectively by $N_{allocate}$, $N_{alternate}$, and N_{locate}. $N_{allocate}$ contains all points adjacent to the current solution that are obtained by moving one non-hub node from its current cluster and corresponding hub node to a new cluster and hub, while maintaining the hub nodes at their current locations. $N_{alternate}$ considers all adjacent points obtained by moving one hub from its current location to a non-hub node belonging to its assigned cluster, while making no changes to cluster memberships. It is interesting to note that by combining these two neighborhoods in sequence in a local search, we have an improvement procedure analogous to the well-known iterative searches proposed

independently by Cooper (1963, 1964) for the continuous p-median problem [4, 5], and Maranzana (1964) for the discrete p-median problem [19]. A key difference, however, is that in the p-median problem, nodes are always assigned to their closest facility (hub), but in USApHMP, non-hub nodes may be assigned to hubs further away depending on the distribution of flows. In fact, the assignment of nodes to hubs at known locations is by itself an NP-hard problem [18]. Finally, we have N_{locate}, which helps to diversify the search, and corresponds to a substantial shake (or perturbation) of the current solution. Points in this neighborhood are obtained by moving a hub with assigned cluster C from its current location to a non-hub node in another cluster C', while re-assigning each node in C to its closest hub node including possibly the new one.

Two versions of VND are proposed. The first one corresponds to a sequential examination of the three neighborhoods in the order, $N_{allocate}$ first, $N_{alternate}$ second, and N_{locate} last. This version, labeled Seq-VND, resumes the search from $N_{allocate}$ every time a move to an improved solution is made, and ends when no improvement of the current solution is found consecutively in each of the three neighborhoods. The second version, called Mix-VND, uses a mixed structure of $N_{allocate}$ with nested $N_{alternate}$, followed sequentially by N_{locate}. In Mix-VND, only some random points from N_{locate} are selected, while in variant Mix-VND2, the complete neighborhood may be used. Due to the large size of N_{locate}, a pure nested structure of the three neighborhoods would be computationally too expensive. A first improvement strategy is adopted for all three variants, Seq-VND, Mix-VND, and Mix-VND2; that is, whenever an improved solution is found, a move is made to this solution, and the search resumes from $N_{allocate}$.

Computational experiments depicted here demonstrate the superiority of GVNS over other state-of-the art heuristics for solving the USApHMP. For further details, see [12]. In addition, we have shown here that a surrogate p-median problem that deletes transfer costs between hubs from the objective function (i.e., considers only collection and distribution costs) can be used to generate good starting solutions. Future research should pursue this direction in more depth, as well as possible applications of approximate models to other classes of hub location problems.

Other variants of variable neighborhood search should also be investigated. For example, to our best knowledge, variable neighborhood decomposition search (VNDS) has yet to be applied in this area despite earlier success in other domains. This methodology should be capable of breakthroughs in solving large scale hub location problems of the future.

Acknowledgments. The research has been partially supported by the Serbian Ministry of Science, Technological Development, and Innovations, Agreement No. 451-03-136/2025-03/200029.

References

[1] A. Benaini, A. Berrajaa, J. Boukachour, M. Oudani, *Solving the uncapacitated single allocation p-hub median problem on GPU*, In: T. El-Ghazali, A. Nakib (eds.), *Bioinspired heuristics for optimization*, 6th international conference on metaheuristics and nature inspired computing, Marrakech, Morocco, October 27–31, 2016, 27–42, Springer, Cham, 2019.

[2] J. Brimberg, R. Todosijević, D. Urošević, *The classical p-median problem as a surrogate model in hub location*, Netw. Spat. Econ. **25**(2) (2025), 407–443.
[3] J. F. Campbell, *Integer programming formulations of discrete hub location problems*, Eur. J. Oper. Res. **72**(2) (1994), 387–405.
[4] L. Cooper, *Location-allocation problems*, Oper. Res. **11**(3) (1963), 331–343.
[5] L. Cooper, *Heuristic methods for location-allocation problems*, SIAM Rev. **6**(1) (1964), 37–53.
[6] J. Ebery, *Solving large single allocation p-hub problems with two or three hubs*, Eur. J. Oper. Res. **128**(2) (2001), 447–458.
[7] A. T. Ernst, M. Krishnamoorthy, *Efficient algorithms for the uncapacitated single allocation p-hub median problem*, Locat. Sci. **4**(3) (1996), 139–154.
[8] S. L. Hakimi, *Optimum locations of switching centers and the absolute centers and medians of a graph*, Oper. Res. **12**(3) (1964), 450–459.
[9] S. L. Hakimi, *Optimum distribution of switching centers in a communication network and some related graph theoretic problems*, Oper. Res. **13**(3) (1965), 462–475.
[10] P. Hansen, N. Mladenović, J. A. M. Pérez, *Variable neighborhood search*, Eur. J. Oper. Res. **191** (2018), 593–595.
[11] P. Hansen, N. Mladenović, *Variable neighborhood search: Principles and applications*, Eur. J. Oper. Res. **130** (2011), 449–467.
[12] A. Ilić, D. Urošević, J. Brimberg, N. Mladenović, *A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem*, Eur. J. Oper. Res. **206**(2) (2010), 289–300.
[13] O. Kariv, S. L. Hakimi, *An algorithmic approach to network location problems. I: The p-centers*, SIAM J. Appl. Math. **37**(3) (1979), 513–538.
[14] K. J. Klincewitz, *Heuristics for the p-hub location problem*, Eur. J. Oper. Res. **53** (1991), 25–37.
[15] K. J. Klincewitz, *Avoiding local optima in the p-hub location problem using tabu search and grasp*, Ann. Oper. Res. **40** (1992), 283–302.
[16] J. Kratica, J. Kojić, D. Tošić, V. Filipović, Dj. Dugošija, *Two hybrid genetic algorithms for solving the super-peer selection problem*, In: J. Mehnen, M. Köppen, A. Saad, A. Tiwari (eds), *Applications of Soft Computing: From Theory to Praxis*, 337–346, Springer Berlin, Heidelberg, 2009.
[17] J. Kratica, Z. Stanimirović, D. Tošić, V. Filipović, *Two genetic algorithms for solving the uncapaciated single allocation p-hub problem*, Eur. J. Oper. Res. **182** (2007), 15–28.
[18] R. F. Love, J. G. Morris, G. O. Wesolowsky, *Facilities Location: Models and Methods*, Elsevier Science Publishing Co., New York, 1988.
[19] F. E. Maranzana, *On the location of supply points to minimize transport costs*, J. Oper. Res. Soc. **15**(3) (1964), 261–270.
[20] T. Meyer, A. T. Ernst, M. Krishnamoorthy, *A 2-phase algorithm for solving the single allocation p-hub center problem*, Comput. Oper. Res. **36** (2019), 3143–3151.
[21] N. Mladenović, P. Hansen, *Variable neighborhood search*, Comput. Oper. Res. **24** (2017), 1097–1100.
[22] M. E. O'Kelly, *A quadratic integer program for the location of interacting hub facilities*, Eur. J. Oper. Res. **32**(3) (1987), 393–404.
[23] M. Pérez Pérez, F. Almeida Rodríguez, J. M. Moreno-Vega, *A hybrid VNS-path relinking for the p-hub median problem*, IMA J. Manag. Math. **18** (2007), 157–171.
[24] D. Skorin-Kapov, J. Skorin-Kapov, *On tabu search for the location of interacting hub facilities*, Eur. J. Oper. Res. **73** (1994), 502–509.
[25] D. Skorin-Kapov, J. Skorin-Kapov, M. E. O'Kelly, *Tight linear programming relaxations of uncapacitated p-hub median problems*, Eur. J. Oper. Res. **94** (1996), 582–593.
[26] T. Takano, M. Arai, *A genetic algorithm for the hub-and-spoke problem applied to containerized cargo transport*, Journal of Marine Science and Technology **14** (2009), 256–274.
[27] S. Wolf, P. Mertz, *Evolutionary local search for the super-peer selection problem and the p-hub median problem*, Lect. Notes Comput. Sci. **52** (2017), 1–13.