Optimizing Geospatial Accessibility to Healthcare Services in Low- and Middle-Income Countries

Parvathy Krishnan Krishnakumari
Analytics for a Better World, University of Amsterdam, Amsterdam, The Netherlands

Joyce Antonissen
Faculty of Economics and Business, University of Amsterdam, Amsterdam, The Netherlands

Fleur Theulen
Department of Econometrics, Tilburg University, Tilburg, The Netherlands

Joaquim Gromicho
Faculty of Economics and Business, University of Amsterdam, Amsterdam, & ORTEC, Zotermeer, The Netherlands

Dick den Hertog
Faculty of Economics and Business, University of Amsterdam, Amsterdam, The Netherlands

Kai Kaiser
The World Bank, Washington DC, United States

Goos Kant
Department of Econometrics, Tilburg University, Tilburg, The Netherlands

Good geospatial accessibility to healthcare is extremely important for good health. Many low- and middle-income countries do not satisfy the healthcare accessibility goals defined by the United Nations. We develop a geospatial analytics toolbox that can both quantify the current healthcare accessibility in a country and improve it by optimizing the locations of healthcare facilities. Since the amount of data is huge and the underlying optimization problem is huge scale, much attention is paid to the scalability of the methodology. For example, the heuristic developed is able to approximately solve a $400,000$ by $350,000$ uncapacitated facility location problem. Two case studies demonstrate the usefulness of the toolbox. The first one is on accessibility to primary healthcare centers in Timor-Leste and the second one is on accessibility to stroke care specialty centers in Vietnam.

Key words: Healthcare services, geospatial accessibility, optimization, developing countries
History:

1. Introduction

In this section we first describe the importance of geospatial accessibility to healthcare services, the role of analytics to measure and improve it, and the analytics toolbox we developed. Moreover, we describe the relevant literature, the contributions of this paper, and the managerial insights and use of the techniques developed in this paper.

1.1. Geospatial analytics

Adopted by the General Assembly of the United Nations in 2015, the 17 Sustainable Development Goals (SDGs) are the layout to achieve a better and more sustainable world for all by 2030. These goals are a call for action to address a series of global challenges, such as: poverty, inequality, climate, environmental degradation, and justice. Each SDG is associated with a set of targets (149 in total) towards which organisations, people, researchers, etc., are working. SDG 3 regarding Good Health and Well-being has 13 targets and 28 indicators to measure progress toward targets. Key target SDG 3.8.1[1]is to achieve universal health coverage, including financial risk protection, access to quality essential healthcare services and access to safe, effective, quality and affordable essential medicines and vaccines for all. In this paper, we develop an analytics toolbox for this target. More precisely, we develop *descriptive analytics* techniques to measure current geospatial accessibility to healthcare services, and *prescriptive analytics* techniques to optimize the accessibility.

Currently, half of world's population do not have access to the healthcare they need[2]. Different terms such as access, utilization, availability and coverage are often used interchangeably to reflect on whether people are receiving the services they need. Access is a broad term with different dimensions. Comprehensive measurement of access requires a systematic assessment of physical, financial and socio-psychological access to services. Measuring, monitoring and optimizing these measurements over time in a data-driven and data-informed manner is important to ensure no one is left behind.

In rural parts of low- and middle-income countries, where most patients travel on foot and few have access to motorized transportation, there is a strong negative relationship between distance to a health facility and the rate of utilization of its services (Stock (1983)). For patients with life-threatening conditions, travel distance to a hospital is associated with mortality. A 10 km increase in straight-line travel distance to a hospital appears to be associated with around a 1% absolute increase in mortality (Nicholl et al. (2007)). The paper Kelly et al. (2016) reviews over one hundred studies to investigate whether there is an association between differences in travel time/travel distance to healthcare services and patient's health outcomes. It concludes that a relationship between traveling further and having worse health outcomes should be considered within the healthcare services debate.

The analytics toolbox. To estimate physical access at an acceptable standard for decision making certain data assets are essential - geographical distribution of beneficiaries, location of service delivery points, and the travel distance for a beneficiary to reach service delivery points using different modes of transport. An integrated analysis of this baseline data can form the foundation

on how accessibility is measured and monitored. Such an analysis can also form the basis of geospatial planning and budgeting to determine where further investments have to be made to improve accessibility.

The Analytics Toolbox *Public Infrastructure Services Access (PISA)*, described in this paper, provides supporting spatial data and a set of on-line decision support tools to help finance, health, education, and transport authorities to optimize the annual and medium term planning and budgeting processes. PISA leverages a set of digital workflow and spatial data innovations to realize transparent and ultimately credible decision support. PISA leverages cloud based open source optimization techniques, coupled with the systematic use of different satellite imagery manual and machine learning/AI applications in support of human capital development decision support. The PISA application also serves to identify physical infrastructure gaps and potentially ways to best address them given budget constraints.

1.2. Literature review

Geospatial accessibility. Many papers have been devoted to the study of geospatial accessibility to healthcare facilities. Dejen et al. (2019) analyses spatial accessibility of healthcare centers by using geospatial technologies in Gamo Gofa, Ethiopia. Juran et al. (2018) combines geographical locations of hospitals in relation to the population with spatial ancillary data on roads, elevation, land use or land cover to estimate travel-times. Besides analysing the best routes to healthcare facilities in the island of Malaysia, Jalil et al. (2018) identifies a suitable site to build a new location of a healthcare facility.

Weiss et al. (2020) produces global maps of travel time to healthcare facilities with and without access to motorized transport. One of the findings is that currently 8.9% of the global population (646 million people) cannot reach healthcare within one hour by motorized transport and 43.3% (3.16 billion people) cannot reach a healthcare facility by foot within one hour.

Healthcare Facility (HCF) location problem. The facility location problem has become the preferred approach for dealing with important strategic issues in healthcare systems, disaster management, and humanitarian logistics (Boonmee et al. (2017)). This healthcare facility (HCF) location problem has attracted considerable attention from the operations research community over nearly four decades. Ahmadi-Javid et al. (2017) provides a framework to classify different types of non-emergency and emergency HCFs in terms of location management, and reviews the literature based on the framework.

Boonmee et al. (2017) presents a survey on the facility location problems that are related to emergency humanitarian logistics based on both data modelling. This survey examines four main

problems: deterministic facility location problems, dynamic facility location problems, stochastic facility location problems, and robust facility location problems. For each problem, facility location type, data modeling type, disaster type, decisions, objectives, constraints, and solution methods are evaluated, including real-world applications and case studies. In a recent work, Dönmez et al. (2021) presents a comprehensive review of the research done on facility location problems under uncertainty in a humanitarian context. It includes an extensive literature list, reviewed from different perspectives: in terms of the type of facilities involved, the decisions to make, the criteria to optimize, the paradigm used for capturing uncertainty, and the solution method adopted.

There are several kinds of facility location problems, each alternative formulation serves a slightly different goal. Daskin (2008) distinguishes three main categories: covering problems, median problems and others. Covering problems use an all-or-nothing approach for the demand, according to which a demand point is counted as covered if it lies within the service distance of one the facilities. Median problems aim to minimize the average distance between the demand points and their appointed facility, where the average distance is weighted by the respective demand. The residual category contains location problems that do not have a clear common component. An example of a problem that falls in this category is the p-dispersion model, which maximizes the minimum distance between any two facilities.

Covering problems are more suitable for non-profit purposes such as placing hospitals, schools or voting offices, because the aim is to make these facilities accessible to all people. Since covering all demand is often very expensive, Church and ReVelle (1974) proposes the Maximum Covering Location Problem (MCLP). Instead of covering all demand, the coverage is maximized with a predefined number of facilities. This formulation is more realistic, since it enables to calculate an optimal solution for any given budget.

Garey and Johnson (1979) proves that the MCLP is non-deterministic polynomial time hard, or NP-hard in short, like many other facility location problems. Hence, for large MCLPs it might be better to use heuristics.

Heuristics. Heuristics do not guarantee to find an optimal solution, but generally they are more efficient and scale better than exact methods, making it possible to handle larger problems. Table 1 shows an overview of many of the heuristics that have been developed for the standard (uncapacitated) MCLP problem. We only included papers in this overview if their approach to solve MCLPs is novel, or if the maximum size of the test instances is larger than other papers reported so far.

The heuristic developed in Galvão and ReVelle (1996) uses the Lagrangian Relaxation (LR) of the maximal covering problem. They relax the covering constraint, resulting in a problem that

Author(s)	Solution method	Max instance*
Galvão and ReVelle (1996)	Lagrangian heuristic	150 × 150
Resende (1998)	Greedy Randomized Adaptive Search Procedures (GRASP)	10,000 × 1,000
Galvão et al. (2000)	Comparison Lagrangian and surrogate relaxations	900 × 900
Lorena and Pereira (2002)	Lagrangian and surrogate heuristic using Hillsman's edition	900 × 900
Pereira et al. (2007)	Column generation adjusted with Langragian	800 × 800
ReVelle et al. (2008)	Heuristic concentration	900 × 900
Senne et al. (2010)	Decomposition heuristic	3,000 x 3,000
Zarandi et al. (2011)	Genetic algorithm	2,500 × 2,500
Rodriguez et al. (2012)	Population-based iterated greedy	3,000 × 3,000
Calderín et al. (2017)	Algorithm portfolio	2,500 × 200
Máximo et al. (2017)	Intelligent guided GRASP	8,000 × 8,000
Máximo and Nascimento (2019)	Intelligent guided GRASP with path-relinking	8,000 × 8,000
Cordeau et al. (2019)	Benders decomposition	40,000,000 × 100

*largest instance solved by the heuristic in paper: # demand points × # potential facility locations

Table 1 The main solution methods proposed for the MCLP

can be solved easily for a fixed set of Lagrange multipliers. Resende (1998) develops a Greedy Randomized Adaptive Search Procedure (GRASP) for the MCLP. Each GRASP iteration consists of constructing a solution with an adaptive greedy approach and improving this solution using local search techniques. This GRASP-method is able to handle instances of 10,000 customers and 1,000 potential locations.

Galvão et al. (2000) compares heuristics based on the Lagrangian relaxation and a surrogate relaxation, and shows that there is no significant difference between these two methods in solving MCLP. Lorena and Pereira (2002) constructs a hybrid Lagrangian/surrogate relaxation that outperforms both heuristics separately. In that approach, the MCLP is formulated as a P-median Problem (PMP), which allows using solution methods that are actually designed for solving a PMP.

The heuristic proposed in Máximo et al. (2017) is based on Resende (1998). Every solution found at the end of the local search phase, is used to build and improve a neural network. The neural network functions like an intelligent adaptive memory, which enables the algorithm to learn from past iterations. Máximo et al. (2017) observes that this heuristic, called Intelligent-Guided Adaptive Search (IGAS), finds the same or better solutions in less time than GRASP on all of their test cases. Máximo and Nascimento (2019) extends both IGAS and GRASP with path-relinking. Path-relinking is the exploration of paths connecting two local optima, and is usually executed at the end of each GRASP iteration. Path-relinking is used to improve the solution after the local search, whereas the memory in IGAS is used to improve the construction phase.

Cordeau et al. (2019) manages to solve instances with a huge number of demand points, up to 40 million, using an algorithm based on Benders decomposition. However, the paper remarks that this heuristic only works on instances for which the number of potential facility locations is very small in comparison to the number of demand points.

We observe that the instances on which the MCLP heuristics have been tested in the literature so far, are not very large compared to the size of the problem that we are aiming to solve. To the

best of our knowledge, the largest case in the literature is 8,000 demand points by 8,000 potential facility locations, whereas the Vietnam case, that we will solve in this paper, is of size 400,000 by 350,000. However, there are several papers that approximately solve the p-median problem for instances of 90,000 by 90,000 nodes, which is already a lot closer to our desired size. In this paper we show that the Timor-Leste case with 15,000 by 190,000 nodes can be solved by a smart implementation of the exact model of Church and ReVelle (1974) and using state-of-the-art solvers. Moreover, we develop a new heuristic that is able to approximately solve the 400,000 by 350,000 Vietnam case.

1.3. Contributions of this paper

The contributions of this paper can be summarized as follows:

- We develop a geospatial analytics toolbox to quantify, visualize, and optimize the accessibility to healthcare services in a country.
- We propose a smart implementation of an existing facility location model, and show that state-of-the-art MIP solvers can solve large scale problems in relatively short time. For example, the Timor-Leste case leads to a 15,000 by 190,000 facility location problem which is solved in less than 3.5 seconds on the average. To the best of our knowledge, MCLPs up to sizes of 8,000 by 8,000 have been *approximately* solved in the literature.
- We develop a heuristic that can approximately solve huge scale uncapacitated facility location problems in relatively short time. For example, the Vietnam case leads to a 400,000 by 350,000 facility location problem which is approximately solved in less than 2 hours.
- Two real-life applications show the usefulness of the methodology: the first one is on accessibility to healthcare centers in Timor-Leste and the second one is on accessibility to stroke centers in Vietnam.

All code of PISA is publicly available at:
https://github.com/Analytics-for-a-Better-World/GPBP_Analytics_Tools.

1.4. Managerial insights and use

The PISA toolbox equips decision-makers with the capability to use data in an intuitive and iterative manner to get insight into the current geospatial accessibility to healthcare. Moreover, it helps them to improve this accessibility by adding healthcare facilities in an optimal way. Efficient use of budgets to improve healthcare infrastructure by using a new generation of digital decision support platforms in low- and middle-income countries is important to ensure better accessibility to healthcare services for the whole population.

1.5. Structure of the paper

The remainder of this paper is structured as follows: in the next section, the geospatial analytics methodology and the tools in the toolbox PISA are described. In Section 3 and 4, descriptive and prescriptive results are given for geospatial accessibility to healthcare centers in Timor-Leste and stroke centers in Vietnam, respectively. Section 5 contains some concluding remarks.

2. Methodology and Tool

In this section we describe the data layer model, the model to calculate geospatial accessibility, the integer optimization model, the heuristic to solve the huge scale problems, and the PISA toolbox.

2.1. Data layer model and sources

The basic data needed for a baseline analysis for accessibility are as conceptualized in Figure 1. At the heart of the model lies mapping of service delivery points, beneficiaries, and the road network that connect them. Once this geospatial data model is in place, it can be further augmented with a range of attributes about the service delivery points, beneficiaries and the road network. The model can then also be enhanced to consider different spatial exposures/risks (e.g., floods) or health challenges (e.g., dengue hotspots) as available.

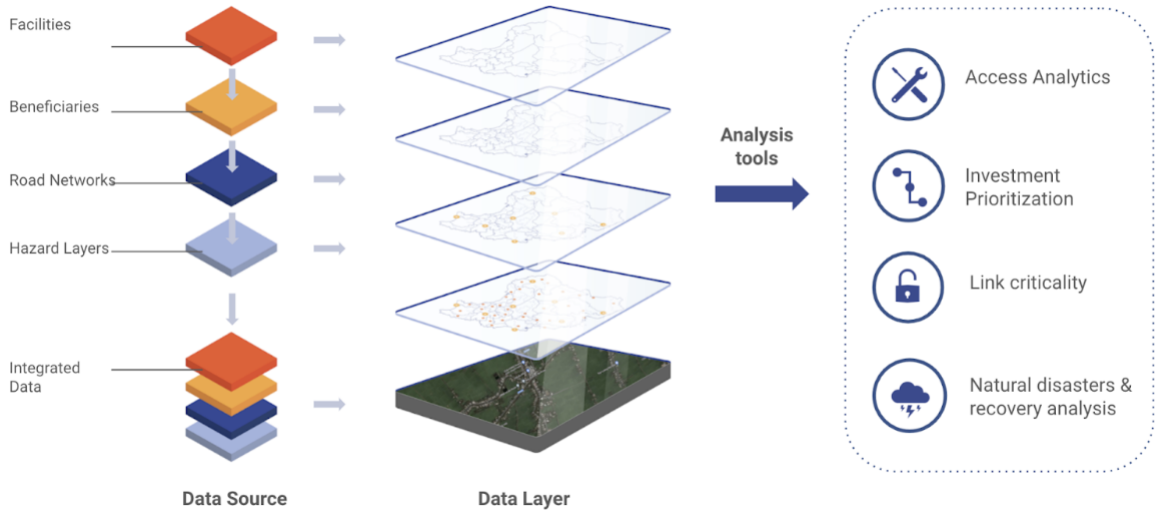


Figure 1 The different data layers of PISA.

Geospatial data is typically available in vector (e.g., roads, administration boundaries, point household data) or raster format (population counts, flood exposure). Vector and raster data is typically quite easy to merge. Data wrangling becomes more time consuming when some type of

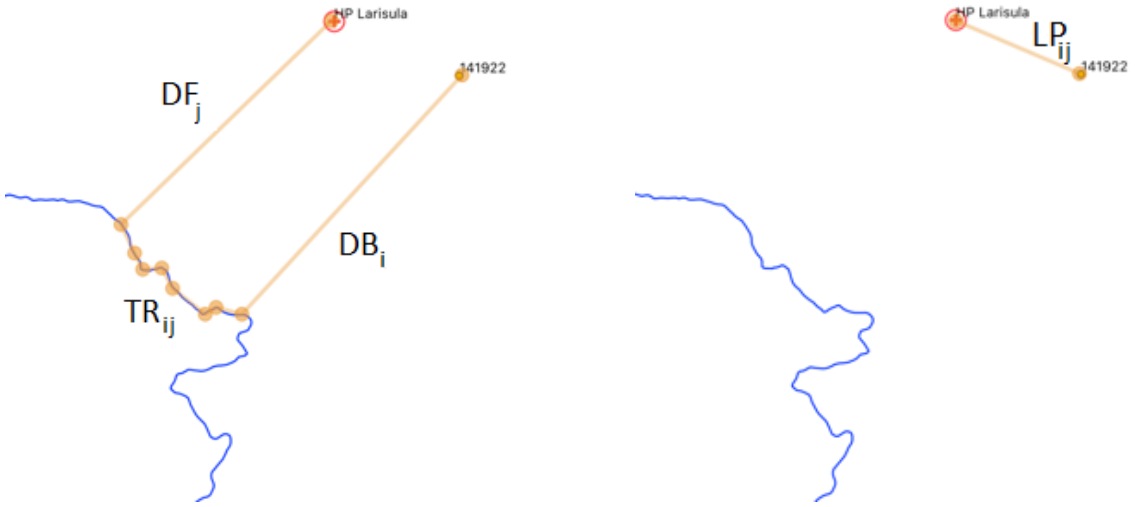
matching is required, and there are not clear IDs in place. For example, health facilities may be mapped, but attribute data is only available in a tabular dataset. Other examples include data that is available at the commune level, but the polygon matting data IDs to fully match this other data (e.g., due to boundary changes or non-consistent coding). The PISA decision support protocol is to identify for each layer of interest the ideal official layer, as well as identify any other, ideally open access layers. Official layers are typically preferred in projects with governments since they are likely to "own" this data, and reference decisions to this data. However, in many settings this data may only be partial, outdated or for a number of other reasons not accessible. Other global sources like OpenStreetMap (OSM) and projects such as World Pop or the Facebook Research High Resolution Human Settlements Layer provide rich resources to quickly stand-up a proof of concept for engagement. The advantage of these layers is that they are available globally, thereby making PISA use-cases replicable across different countries.

2.2. Geospatial accessibility model

Modeling true access behavior by, e.g., Timorese households to a health facility presents some additional challenges. For those households very far away from a road, it may make little sense to first walk to a road and than travel through the road network to a healthcare facility. Rather, these beneficiaries may choose to directly walk to a facility by ways of a path not mapped into the data, so they travel 'off road'. In our research, we assume that a beneficiary first finds the shortest distance to the nearest road. We define this distance as DB_i for household i. Next, the model finds the closest hospital facility j by traveling via the road network TR_{ij}, and back in turn a road connection to the hospital facility DF_j. Differently, we can consider the linear path from a household to a hospital facility as LP_{ij}. Those distances are illustrated in Figure 2 below.

We assume in this research that a beneficiary uses the linear path to the hospital facility (LP_{ij}) if the distance from a beneficiary to the nearest road (DB_i) is greater than the linear path (LP_{ij}). In that case, we assume a household rather travels 'off road' to the hospital facility instead of traveling to the main road first. Additionally, we assume that beneficiaries travel directly to the hospital, so uses the linear path LP_{ij}, if they have to travel more than 1 kilometer to reach the road network.

To speed up the distance matrix calculation we exclude the set of households that can already reach an existing hospital facility within the smallest threshold. Since they can already reach a facility, there is no need to include them in the optimization. Moreover, before calculating the travel distance over the road network, the Euclidean distance is checked. If this distance is larger than the largest threshold, the household will never be able to reach a facility within any of the



(a) Distance over the road (b) Distance by linear path

Figure 2 Calculation of travel distance

thresholds, so there is no need to calculate the shortest path. This reduces the number of shortest path calculations and thus reduces running time.

Special attention has been given to the calculation of the huge distance matrix. Since digital road networks may have a few million nodes and arcs, a single shortest path query can easily cost a number of seconds to compute, even by the fastest computer of nowadays. We therefore use Contraction Hierarchies (Foti and Waddell (2012)), more precisely the python package Pandana (Pandana (2023)). Pandana enables us to use road networks as in OSM and to compute matrices of distances on them in a scalable way. The generation of the distance matrix for Vietnam (350,000 by 400,000) took in total 20 minutes.

2.3. Optimization model

The objective of the optimization model is to maximize the number of inhabitants that can reach a facility within the preset maximum desired travel distance S. The capacity of the facilities is not taken into account. We use the model proposed by Church and ReVelle (1974), since the number of variables of that model is linear in the number of potential facility locations and the number of households. We first introduce the sets and parameters that are used in the model:

$\mathcal{N}$: the index set of households, or clusters of households, indexed by $i = 1, ..., n$
$\mathcal{M}$: the index set of all facility sites, where indices $j = 1, ..., \bar{m}$ correspond to the already existing facilities and indices $j = \bar{m}+1, ..., m$ correspond to potential facility locations
v_i the number of people in (cluster of) household(s) i
d_{ij} the travel distance from (cluster of) household(s) i to facility j
S the maximum travel distance from a household (or cluster) to a facility
p number of additional facilities located,

and additionally, we have the following decision variables:

$$y_j = \begin{cases} 1 & \text{if facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$x_i = \begin{cases} 1 & \text{if there is an opened facility within } S \text{ kilometers travel distance} \\ & \text{of the (cluster of) household(s) } i \\ 0 & \text{otherwise.} \end{cases}$$

We can now formulate the optimization model:

$$\text{Maximize} \quad \sum_{i \in \mathcal{N}} v_i x_i \tag{1}$$

$$\text{Subject to} \quad y_j = 1, \quad \forall j = 1, ..., \bar{m} \tag{2}$$

$$\sum_{j=\bar{m}+1}^{m} y_j \leq p \tag{3}$$

$$x_i \leq \sum_{j | d_{ij} \leq S} y_j, \quad \forall i \tag{4}$$

$$y_j, x_i \in \{0, 1\}, \quad \forall i \in \mathcal{N}, \forall j \in \mathcal{M}. \tag{5}$$

The objective function (1) maximizes the population that can reach a facility within the maximum travel distance S. The first constraint (2) ensures y_j is set to one for the already existing facilities. The second constraint (3) limits the number of additional facilities opened. When there is a monetary budget for new facilities, the following constraint can replace constraint (3):

$$\sum_{j=\bar{m}+1}^{m} c_j y_j \leq B, \tag{6}$$

where c_j are the costs of opening facility j and B is the available budget. This constraint (6) ensures that opening the facilities stays within the budget. Constraint (4) forces x_i to be zero when there are no opened facilities within the maximum travel distance S. This means that household i is not served by any facility in that case. Since we assume x_i to be binary in constraint (5), every household can only be served once. Additionally, y_j is assumed to be binary.

In practice decision makers are interested to see what the optimal coverage is for different values of p, the number of facilities. The Pareto curve shows this coverage for different values of p. Hence, to generate such a Pareto curve the optimization model has to be solved for different values of p.

The optimization model explained above differs from the maximum covering model of Church and ReVelle (1974) in constraint (3): we limit the number of facilities to be at most p while Church and ReVelle (1974) takes exactly p. A practical advantage of our version lies in the computation of

the Pareto curves, as discussed in Section 3.5: the solution found for p remains feasible for higher values of p and therefore allows the solver to continue optimization after adjusting the constraint.

When implementing the model care should be taken in the definition of constraints (4). A careless implementation searches the set of $\{j \mid d_{ij} \leq S\}$ for each i with effort proportional to $|\mathcal{N}| \times |\mathcal{M}|$ while these lists can be efficiently obtained by reverting the so-called *catchment areas* of the facilities.

The model is implemented in Python, and the Gurobi Optimizer[3] is used to solve the model instances. The optimizer stops once the best solution found is within a prescribed tolerance from the true optimum, the so-called optimality gap. In Section 3.5 we illustrate the trade-off between solution quality and the time taken to optimize.

2.4. Heuristic

The exact approach described in Section 2.3 can be used for large scale cases as Timor-Leste, but is not suitable for huge scale MCFL problems as the Vietnam case. In this section we explain the main concepts of the heuristic we developed, which is based on GRASP; see Resende (1998), Resende and Ribeiro (2003), Resende and Werneck (2004, 2007), Resende et al. (2010), Resende and Ribeiro (2010, 2019). GRASP consists of multiple iterations, where each iteration consists of a construction and a local search phase. We explain the main ideas behind these phases and how to create a very fast implementation. Details can be found in the Online Supplement.

Construction heuristic. There are numerous methods that can be used in the construction phase to build up the solution, the most common ones are discussed by Resende and Werneck (2004) and by Resende and Ribeiro (2010). A good construction method is efficient and yields high quality solutions, but also ensures there is enough variation between solutions in different iterations. In a standard GRASP implementation, the randomized greedy (`rgreedy`) method is most often used. The first facility is chosen uniform randomly from the top $\alpha\%$ of potential facilities, which are ranked according to their greedy value. Here, the greedy metric measures the additional demand covered by a facility if that facility is chosen. The top $\alpha\%$ of facilities form the Restricted Candidate List (RCL). Once the first facility is chosen, all remaining facilities are reevaluated with the greedy metric, and the next facility is selected in the same way. This process is repeated until p facilities are chosen. Every evaluation of the greedy metric has complexity $O(nm)$, so the whole selection process has complexity $O(pnm)$.

There are a few variants which we tested, like `random`, where p facilities are selected randomly from all m potential facility locations, random plus greedy (`rpg`), where randomness and greediness are combined, proportional greedy (`pgreedy`), where an underlying distribution is used with a bias towards choosing a facility with a higher rank, and sample greedy (`sample`), which is closely

related with the randomized greedy approach. See Resende and Ribeiro (2010) for more details.

Local search and path-relinking. In the local search phase, the solution found in the construction phase is improved by investigating its neighbourhood, and adapting the solution if a better solution is found. This process continues until a local optimum is obtained. The time that it takes to find a local optimum and the quality of such a solution mainly depends on the strategy of the local search, the quality of the starting solution, and the definition of a neighbourhood.

For the strategy, we distinguish first improvement local search, during which a move is made as soon as a better solution is found, and best improvement local search, during which the whole neighbourhood is investigated before the best move is executed. Resende (1998) shows that it is preferable to spend some more time in the construction phase to find a higher quality solution. Regarding the neighbourhood of a solution, a smaller neighbourhood will likely be more efficient, but could also yield lower quality local optima. Within GRASP, it is common practice to define the neighbourhood as all the solutions that differ from the considered solution by one facility. The best solution in this neighbourhood is called the 1-swap local optimum, because it can be found by 'swapping' one active facility with one inactive facility.

As we have explained in the literature review, the independence between iterations is a big disadvantage of GRASP. Therefore, Laguna and Marti (1999) proposes to apply path-relinking to the solutions found in each GRASP iteration. The general idea of path-relinking is to explore a path connecting two promising solutions, in order to investigate whether an even better solution can be found on this connecting path. The reason is that many good solutions occur in each others vicinity in the solution space, and therefore a better solution possibly can be found on the path connecting two already high-quality solutions. Path-relinking thus focuses on researching a promising part of the solution area more intensively. Details can be found in Resende and Ribeiro (2003) and Resende and Werneck (2007).

Creating a fast implementation. In our heuristic, many different solutions will have to be evaluated in an attempt to find a high-quality solution. In the construction, local search and path-relinking phase, the new solution that is considered always differs by only one facility from the previous solution. Hence, instead of calculating the objective from scratch for every new solution, it is beneficial to have a way of efficiently computing the objective change caused by a small modification in the solution.

To do so, let C be the binary matrix, where a cell is 1, if the corresponding facility covers the corresponding population point. Based on this we create the coverage vector cov, specifying how many facilities cover each population point in the current solution. If one active facility is swapped

with an inactive facility, you can simply take the coverage vector of the old solution, subtract the row of C corresponding to the leaving facility and add the row of the entering facility. Adding and subtracting a row of C has complexity $O(n)$.

Although the binary matrix C is a clear and intuitive way to represent which facilities can cover which population points, the full binary matrix would have $350,000 \times 300,000$ entries if it were constructed for the Vietnam case with grid size equal to 1 and $S = 50$. Since the sparsity of each case varies between 0.09% and 1.91%, converting C to sparse matrix results in a large memory space reduction. We do this by introducing two dictionaries, *row* and *col*. Dictionary *row* represents the facilities, and each value is a vector containing all population points that can be covered by the corresponding facility. Dictionary *col* represents the population points, and each value is a vector containing all facilities that can cover that point. Suppose we want to swap facility f, and let k be the number of elements in $row[f]$. Then the complexity of updating the coverage vector is equal to $O(k)$, because only k additions need to be executed. If we redefine k as the maximum length of $row[f]$ for any $f = \bar{m}+1, \ldots, m$, we can state that any addition, removal or swap of facilities has complexity $O(k)$. The fact that we have sparse data implies that $k << n$, so the complexity of $O(k)$ is a substantial improvement compared to $O(n)$. Using these principles, the construction heuristic becomes very fast.

For the local search improvement heuristic, the strength of our implementation is that we try to minimize executing unpromising calculations by restricting the neighbourhood of each chosen facility, and prioritize facilities which have been investigated the least. However, these advantages are lost once our approach is transformed into a best improvement local search. To speed up this implementation, we created an advanced approach: we first calculate the objective change corresponding to all possible swaps, and save these outcomes. Then we pick the best one, and determine which of other outcomes are affected by the chosen swap. After updating these affected values, we repeat this process, until a local optimum is found. Resende and Werneck (2007) were the first ones to implement a best improvement local search in this advanced way, but developed for a p-median problem. We translated it to be applicable to MCLP. To the best of our knowledge, there do not exist any papers which discuss how this translation from PMP to MCLP can be accomplished. As this translation is far from trivial, you can find a detailed explanation how we have achieved this in the Online Supplement.

2.5. PISA Analytics Toolbox

The analytics toolbox Public Infrastructure Services Access (PISA) is a suite of analytic tools and data layers designed to help finance and sectoral authorities. PISA is developed to support country's decisions on facility location. It contains the data layer model, the geospatial accessibility

model, the optimization model and heuristic developed in this paper and described in this section. The goal is to find out what is needed to ensure accessibility to healthcare services to most of the population. In particular, we want to know how many health facilities need to be added in low- and middle-income countries, and determine where the investments have to be made to optimally improve accessibility to facilities. As demographic conditions are often changing rapidly in low- and middle-income countries, this requires a more timely understanding and benchmarking of annual and medium term investment scenarios. New data sources (including big data) and modern analytics provide a rich new way to improve decision support. However, many low- and middle-income countries struggle to harness this. Learning by doing through specific use cases opens up avenues for policy makers to become more aware of these resources, especially in the context of tools that are easier to use for non-IT specialists. As computational power is democratized through cloud technologies, analytic tools and decision support are more capable, less complex and costly friendly for end users.

The use of PISA in low- and middle-income countries is important for various reasons. First, such countries will benefit from recurrent decision support. PISA will deliver tangible improvements to annual and medium-term planning and budgeting practices. The platform can be used by policy and practices as it is related to digital assets. Moreover, PISA maps access to public health facility assets in such countries. It maps facility locations and the status of those facilities. Additionally, with PISA it is possible to validate the access and quality of relevant geospatial data. The data can be shared and feedback can be given and implemented. Satellite image feeds and Machine assisted Infrastructure Detection (MaIDs) can be used to get the right input data. PISA contains the optimization methodology described in Sections 2.3 and 2.4 to obtain optimal results.

3. Timor-Leste Case

This case study implementation of the Public Infrastructure Service Access tool was implemented in partnership with the World Bank Group and used a data stack consisting of high-resolution gridded population estimates, existing health facilities, and travel distance calculations to analyze the current accessibility of the population to health facilities in Timor-Leste. We then employed an optimization strategy to identify the best locations for new health facilities.

3.1. Background

The Republic of Timor-Leste (Timor-Leste) is a small, low-income country located in Southeast Asia. The country has a population of 1.36 million people (2022), and the majority of the population lives in rural areas. Access to healthcare in Timor-Leste is a challenge, and there are a number of

barriers that prevent people from accessing care. The country has a rugged, mountainous terrain, affected by seasonal flooding and landslides. Reaching health facilities can be a herculean task, especially when over 90% of the roads are in poor condition. This is further exacerbated by the 70% of the population residing in rural areas. Even though healthcare in Timor-Leste is free at the point of delivery, there are often out-of-pocket costs for transportation, medications, and other expenses. This can be a significant financial burden for people who live in poverty.

The disparity in health facility access is evident, with wealthier patients utilizing hospital care nearly twice as much as poorer patients. This is likely due to a combination of the physical, financial, and socio-cultural barriers that prevent people from accessing healthcare. A recent report by UNICEF found that the under-five mortality rate in Timor-Leste is 51 deaths per 1,000 live births and the maternal mortality ratio is 280 deaths per 100,000 live births. Only 54% of children under the age of five are fully vaccinated against measles with around 38% of births attended by a skilled health professional. These health indicators are significantly worse than the average for Southeast Asia. The report also identifies a number of challenges facing the health sector in Timor-Leste, including. The report concludes by calling for increased investment in the health sector in Timor-Leste with a specific focus on challenges such as poor quality of infrastructure and lack of access to healthcare in rural areas.

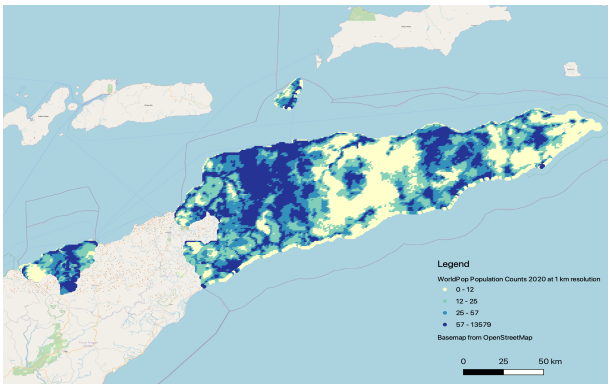
3.2. Data Stack

This case study uses a data stack consisting of high-resolution gridded population estimates, existing health facilities, and travel distance calculations to analyze the current accessibility of the population to health facilities in Timor-Leste. We then employed an optimization strategy to identify the best locations for new health facilities. Starting with the creation of a grid spanning the country with 1 km intervals, this served as a foundational layer to determine potential facility sites. Using the Mixed Integer Optimization (MIO) model, each grid point was evaluated for its suitability as a health facility location. The optimization model then selected the most strategic sites from the grid that maximized healthcare access to the population of Timor-Leste.

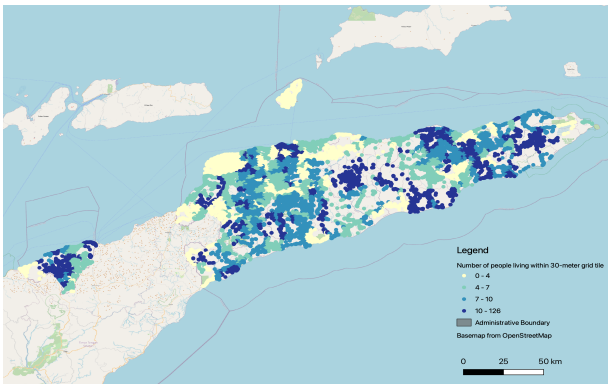
There are a number of different datasets available for population estimates in Timor-Leste; see Table 7. These datasets vary in terms of their accuracy, granularity, and coverage. Some datasets are based on census data, while others are based on surveys and/or satellite data. The same applies to health facilities, some datasets are maintained by the government and other public sector organizations while others are crowd-sourced and validated by the open-source community.

In this analysis, various datasets have been employed (see Table 7), each bringing its distinct perspective on population, transportation, and health facility accessibility. Regarding population data, though WorldPop Population Counts (WorldPop (2020)) offer datasets with a granularity of

100m resolution, for the purpose of sensitivity analysis in our study, we have utilized the WorldPop data with a 1 km resolution (Figure 3a), and the High-Resolution Population Density Maps by Meta's Data for Good Program Lab and for International Earth Science Information Network CIESIN Columbia University (2016) at 30-meter resolution (Figure 3b) intending to understand how variations in data resolution might influence the outcomes. WorldPop employs two significant methodologies for population count disaggregation: 'unconstrained' and 'constrained'. The unconstrained approach predicts population numbers for all 100x100m land grid cells globally, without limiting the predictions to known settlements. Its advantage lies in its applicability where settlement detection from satellite data is uncertain, especially concerning small rural settlements. This method spans from 2000-2020, making it apt for historical or change analyses. Conversely, its drawback is potential misallocation to uninhabited areas, causing underestimates in some urban regions. The constrained estimation, on the other hand, limits the population counts to areas identified with built settlements. Leveraging satellite-derived building footprint data, this method gives a more accurate population distribution when settlements are accurately mapped. However, its reliability is heavily dependent on the precision of satellite-based mapping, with challenges in missed settlements causing population allocation discrepancies.



(a) WorldPop UN Adjusted Population Counts for Timor-Leste (2020) at 1 km resolution



(b) High-Resolution Population Density Maps by Meta Data for Good (2020) at 30-meter resolution

Figure 3 Population Count Datasets

Regarding the geotagged health facility data (Figure 4), the World Health Organization (WHO) dataset, comprising information on 347 health facilities, offers a comprehensive view, but it is worth noting that unlike the curated platform Healthsites.io (Saameli et al. (2018)), which provides open access to an online structured API, the WHO dataset was a result of a one-time data collection to create an asset registry. Healthsites.io, meanwhile, has data on 187 hospitals and clinics readily accessible online and is updated frequently by the open-source community ensuring scalability and replicability.

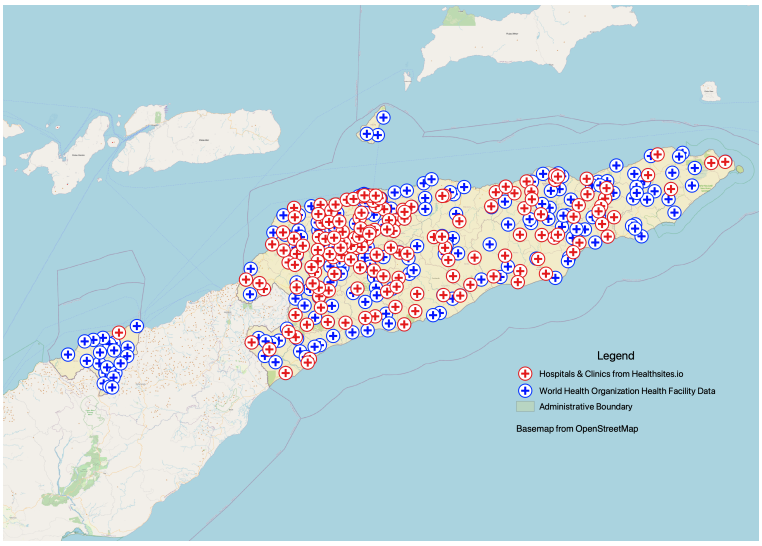
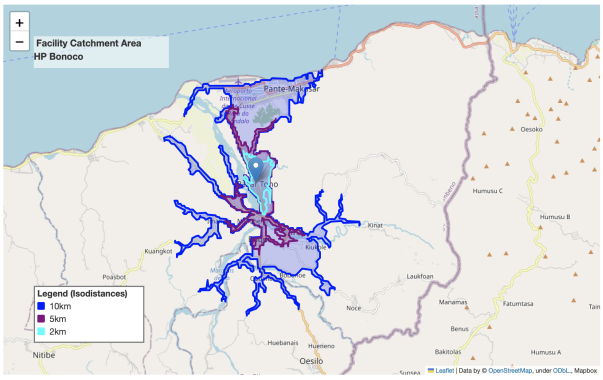
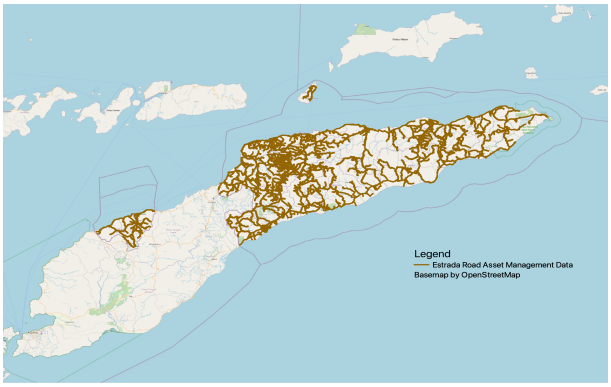


Figure 4 Geotagged Health Facilities Data from Healthsites.io and World Health Organization

For transportation insights, we tap into two primary sources: The MapBox Isochrone API, rooted in OpenStreetMap (OSM) data calculates areas accessible from a specified location within a certain timeframe or distance (Figure 5a), and the Estrada Road Asset Management System by Catalpa.io, which is the official road asset management system (RAMS) of Timor-Leste (Figure 5b).



(a) The MapBox Isochrone API computing areas that are reachable within a specific travel distances is showcased for the Health Post Bonoco



(b) Official Road Asset Management Registry of Road Network Data

Figure 5 Travel Distance Data Categories

Different data sources offer varying perspectives, with some being comprehensive while others provide insights on niche aspects. This inherent variability amplifies the need for a sensitivity analysis. The reasons for this are manifold: (i) *Robustness:* Ensuring that results are not drastically altered by minor data tweaks is essential. This underscores the stability of the derived conclusions,

regardless of the data nuances. (ii) *Data Reliability:* Data sources differ in terms of their granularity, and precision. Sensitivity analysis helps pinpoint the datasets that align best with the study's goals, lending more weight to the ensuing findings. (iii) *Highlighting Key Variables:* Understanding which datasets or parameters predominantly sway the results can shape subsequent research or data assimilation efforts. (iv) *Data-informed Decision-making:* For stakeholders, consistent outcomes across varied data sources bolster confidence, underscoring the reliability of the analysis. (v) *Data Uncertainties:* Data, by nature, comes with uncertainties - whether from temporal changes, measurement inaccuracies, or other dynamics. Sensitivity analysis not only acknowledges these uncertainties but also quantifies their potential impact. (vi) *Resource Mobilization:* Discerning the data sources that make minimal contributions to the outcome allows for the focused channeling of efforts and resources toward more significant datasets. (vii) *Risk Mitigation:* Especially when the stakes are high, understanding how outcomes oscillate based on different data sources is tantamount to risk mitigation. It provides a spectrum of scenarios from best to worst.

Given Timor-Leste's health access and data availability challenges, the emphasis was on ensuring that our analysis was resilient to variations in data sources. Misdirection based on untrustworthy data might worsen existing health disparities or divert critical resources inefficiently. As such, we employed different configurations to calculate the population's access percentage.

3.3. Descriptive analytics results

To address the distinct challenges of Timor-Leste's landscape and infrastructure, the study set travel distance thresholds of 2, 5, and 10 km to optimize healthcare accessibility. The 2 km threshold ensures rapid access for urgent medical needs, especially vital given the region's rugged terrain and often unreliable roads. Such thresholds also aim to bridge the noticeable gap in healthcare utilization between different socioeconomic groups, fostering equal access. By facilitating proximity to healthcare centers, the study encourages routine medical consultations, promoting early disease detection and overall public health. Furthermore, this approach minimizes transportation challenges in areas where mobility might be limited or expensive. Accessible healthcare facilities not only provide essential services but also fortify trust between the community and healthcare professionals. The variation in thresholds represents a tiered strategy, ensuring that while immediate healthcare is widely accessible, specialized facilities remain within reach for the majority.

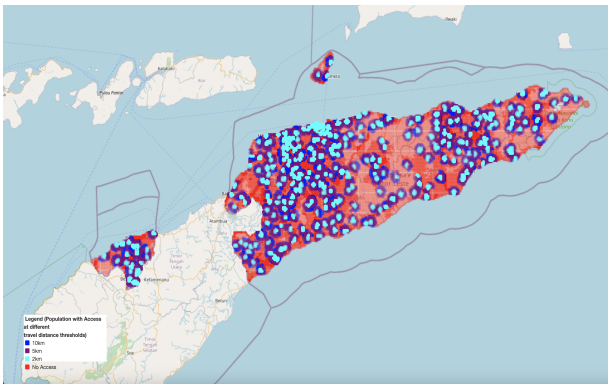
Comparative analysis of health facility accessibility for Timor-Leste's population across different data stack configurations is available in Table 2. The variations in the percentage of the population with access to health facilities at different travel distances depend on the chosen data stack configuration. Configuration A and B, using WorldPop Population Counts, showcase relatively similar percentages. However, when transitioning to the High-Resolution Population Density Maps from

Configuration B to D, a substantial increase in the percentage is observed. This comparison underscores the profound sensitivity of the results to the choice of data layers especially the granularity and resolution of population counts. Our analysis leverages openly available data sources, but integrating official population counts with finer granularity would undoubtedly offer enhanced insights. However, a challenge lies in that official census data, while invaluable, is typically aggregated at sub-national boundaries. Moreover, it often lacks the high resolution we seek and sees updates only every 5 to 10 years. Considering the ever-evolving nature of both populations and infrastructure needs, relying solely on this kind of aggregated data might pave the way for less-than-ideal planning decisions.

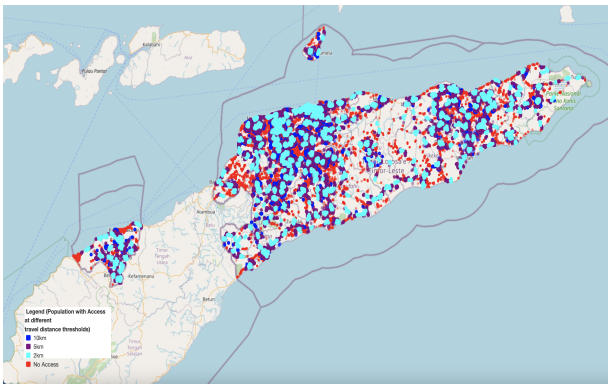
Configuration	**Data stack**			**% of population with access to health facility at travel distance in km**		
	Population	Transport	Geotagged Health Facilities	2 km	5 km	10 km
Configuration A	WorldPop Population Counts	MapBox Isochrone API	Healthsites.io	22	36	48
Configuration B	WorldPop Population Counts	MapBox Isochrone API	World Health Organization	24	40	53
Configuration C	WorldPop Population Counts	Estrada Road Asset Management System	World Health Organization	37	66	76
Configuration D	High Resolution Population Density Maps	MapBox Isochrone API	World Health Organization	43	72	87
Configuration E	High Resolution Population Density Maps	Estrada Road Asset Management System	World Health Organization	51	76	84

Table 2 Percentage of population with access to health facilities in Timor-Leste under different data stack configurations

Configuration C, incorporating the Estrada Road Asset Management System, exhibits the highest percentage compared to B across all travel distances, emphasizing the difference that specific road data can make. MapBox is predominantly based on OpenStreetMap (OSM) data. While OSM is a robust and continually updated platform, its completeness and accuracy can sometimes be less than that of official road asset management systems. Official road asset management systems often have detailed records of all road networks, including those that might not be popular or frequently traveled, but are crucial for connectivity. In contrast, OSM, being a crowd-sourced platform, might occasionally miss out on these lesser-known road links. OSM relies heavily on user contributions. While this has its strengths, it can sometimes lead to inaccuracies or outdated information, especially in regions with fewer active contributors. Official systems, on the other hand, are typically



(a) Configuration C showcasing the access of population at 2, 5, and 10 km travel distance using Unconstrained WorldPop Population Counts using the Official Road Asset Management Systems Data



(b) Configuration E showcasing the access of population at 2,5, and 10 km travel distance using the 30-meter high resolution population dataset from Meta's Data for Good using the Official Road Asset Management Systems Data

Figure 6 Visualizing the population with and without access to a health facility using the official road asset management system data and the two different population data layers

updated and vetted by professionals, ensuring higher accuracy. The official road asset management system is designed for specific tasks like road maintenance and planning, so it tends to have a "clean" network. This means fewer extraneous data and more consistent data quality. In OSM, on the other hand, being open to a wider range of contributors, we see more inconsistencies or "noisy" data. The effect on the coverage for Configuration C and E are depicted in Figure 6.

3.4. Tuning the grid size and the solver

The primary objective of this paper is to maximize the percentage of the population in Timor-Leste with access to healthcare facilities. A 1 km x 1 km grid was superimposed on Timor-Leste's geographical administrative boundary. Each grid point was then treated as a potential site for a new health facility. This systematic approach ensured a comprehensive coverage of the region, while the granularity of the grid provided an acceptable trade-off between computational feasibility and resolution.

To analyze the effect of the grid size on the coverage, we first calculate the coverage when a facility is placed in each potential location. Table 3 shows that full coverage is attainable for $S = 3$ kilometers. The table uses enough decimal places to disclose that for 2 kilometers coverage is just almost 100%, but for $0.1, 0.5, 1.0$ kilometers it is (much) lower than 100%. This is of course due to the grid size of 1 km x 1 km that we used. We conclude that this grid size is accurate enough when we consider accessibility distances of $S = 2$ km or higher.

	0.1 km	0.5 km	1.0 km	2.0 km	3.0 km	5.0 km	10.0 km
present	1.06234	10.63255	29.61263	37.53874	49.58958	66.40395	76.21680
max attainable	7.20859	82.07126	99.87826	99.99962	100.00000	100.00000	100.00000

Table 3 Coverage in Timor-Leste with the present facilities and if all potential facilities would be open.

An important solver parameter is the upper bound for the optimality gap. We first set this upper bound to 0.1, which means that the solution we obtain might be 10% worse than the true optimal solution. The curves in Figure 7 show the coverage for different values of S and the budget p. The expectation is that the curves will finally reach 100% coverage, which is not the case. Moreover, we would expect the curve for $S = 5$ kilometers to be above the curve for $S = 3$ km, and the curve for $S = 10$ km to be above the curve of $S = 5$ km, which is not the case. This happens since we allow the solver to be satisfied with solutions within 10% of the optimum. For Timor-Leste this is more than one hundred and thirty thousand people.

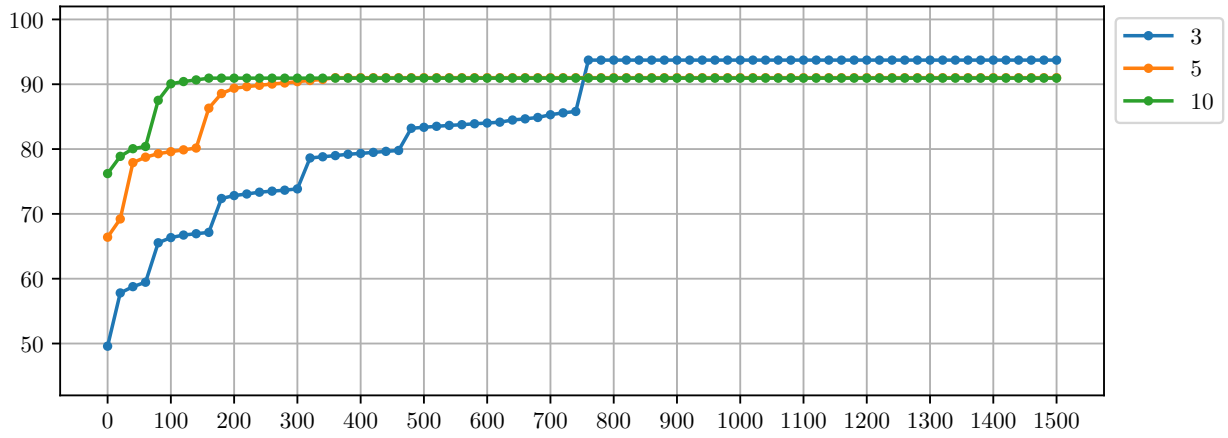


Figure 7 Coverages (in %, on vertical axis) found by the solver for different values of the budget (p, on horizontal axis) for $S = 3$ km (blue curve), $S = 5$ km (orange curve), $S = 10$ km (green curve), using 'optimality gap'=0.1.

To ensure true optimality with an objective function that can add above 10^6 we need an 'optimality gap' of 10^{-7}. This is, unfortunately, unrealistic for hard problems such as this one. Fortunately, optimality within reasonable tolerances becomes useful for less ambitious 'optimality gap' values, as Figure 8 shows. It takes about 13 minutes to obtain these curves. To create these curves in total $3 \times 75 = 225$ optimization runs are carried out, which means that on the average one run takes only 3.5 seconds.

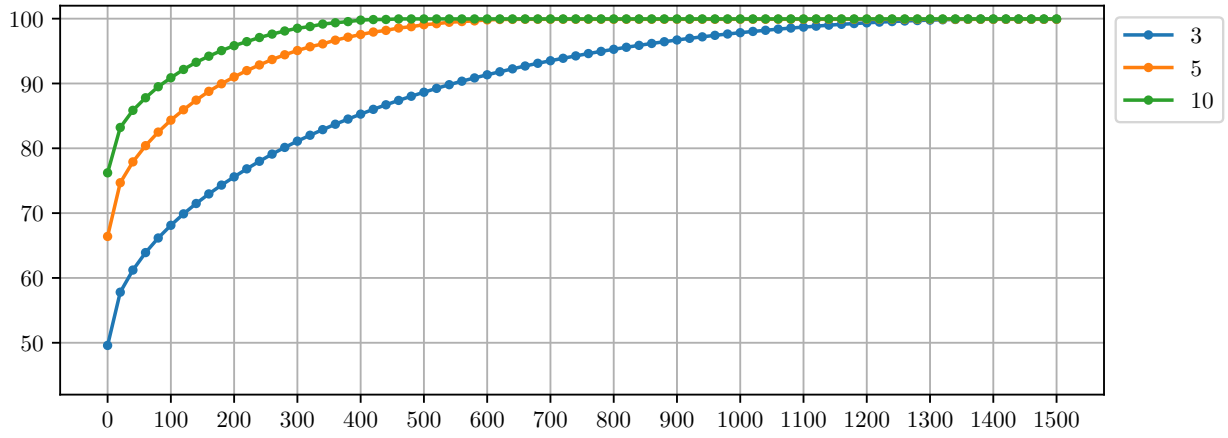


Figure 8 Coverages (in %, on vertical axis) found by the solver for different values of the budget (p, on horizontal axis), for $S = 3$ km (blue curve), $S = 5$ km (orange curve), $S = 10$ km (green curve), using 'optimality gap'=0.001

Let us analyze in more detail the trade-off between accuracy and calculation time, which consists of modeling and solving time. As Table 4 shows, the modelling times scale very well across S, the distance threshold, which affects the number of coefficients in constraint (4). However, the solving times are very much influenced by the value of the 'optimality gap'.

		3 km	5 km	10 km
0.1	modeling	00:00:01.12	00:00:01.24	00:00:01.34
	solving	00:00:31.28	00:00:14.65	00:00:07.32
0.01	modeling	00:00:01.17	00:00:01.20	00:00:01.35
	solving	00:05:08.87	00:01:34.02	00:00:24.79
0.001	modeling	00:00:01.19	00:00:01.24	00:00:01.32
	solving	00:10:16.68	00:02:15.65	00:01:27.30
0.0001	modeling	00:00:01.24	00:00:01.27	00:00:01.40
	solving	06:15:12.59	00:09:18.11	00:03:54.55

Table 4 Times taken for 'optimality gap' $= 0.1, 0.01, 0.001, 0.0001$, and distance threshold $S = 3, 5$, and 10 km.

Table 5 shows the statistics of the absolute gaps reported by the solver for each solution found while computing the Pareto curves. We conclude that taking the value 0.0001 for the 'optimality gap' is a good trade-off between solution accuracy and calculation time.

		mean	std	25%	50%	75%	max
0.1	3	75298	24292	73205	82764	82764	112486
	5	101347	33594	118468	118468	118468	118468
	10	111192	26374	119465	119465	119465	119465
0.01	3	7682	4229	5054	8012	11748	12778
	5	9857	3698	8684	12046	12046	12661
	10	10579	3150	11977	11977	11977	11977
0.001	3	611	508	116	536	1216	1315
	5	967	477	453	1289	1289	1312
	10	522	254	570	570	570	1296
0.0001	3	92	46	68	113	126	132
	5	99	37	113	113	113	131
	10	104	39	120	120	120	131

Table 5 Headcounts that can be missed per optimality gap and threshold

3.5. Prescriptive analytics results

We consider the Pareto curves in Figure 8. The current coverages (i.e., $p = 0$) are 50%, 66%, 76% for $S = 3, 5, 10$ km, respectively. To increase this to 95% coverage, we need $820, 300, 180$ more facilities, respectively. Due to the strong concave behaviour of the Pareto curves, we observe that especially big coverage improvements can be obtained for adding the first number of facilities. For example when adding 100 facilities, the increase in coverage is about 19%, 17%, and 15% for $S = 3, 5$ and 10 km, respectively.

4. Vietnam case

In this section, we apply the analytics techniques described in this paper to stroke centers in Vietnam. We start with a general introduction to Vietnam and then describe how to improve the geospatial accessibility by adding more healthcare facilities in an optimal way. For this case we will not discuss the data collected in such a detail as we did for Timor-Leste. This is to save space, and because the focus in this application is to show the scalability of especially the heuristic. Since for this case, we could not use the exact integer optimization approach described in Section 2.3, we will use the heuristic described in Section 2.4.

4.1. Background

Stroke and ischemic heart disease were the leading two causes of death and disability-adjusted life years (DALYs), in Vietnam in 2019. Health insurance inpatient admission data in Vietnam indicates that there were $45,000$ admissions for AMI (an acute exacerbation of ischemic heart disease requiring emergency care). The Vietnamese National Stroke Association estimates that there are a total of $200,000$ strokes each year and $104,000$ deaths from stroke. Treatment for

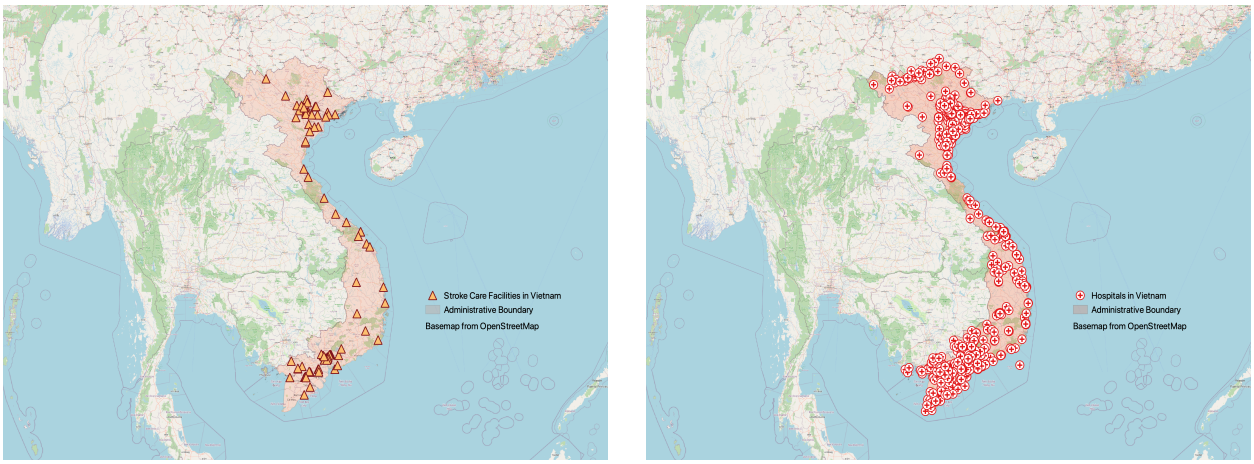
stroke and AMI require timely interventions within short windows from time of onset before death or permanent damage occurs. It is therefore important that not only do families and medical practitioners immediately recognize the signs and symptoms, but that the patients can access the necessary specialist services without delay. They may be accessing care directly from home (population concentrations) or from another facility (like a commune health station or district hospital). In recent years Vietnam has developed stroke guidelines (5331/2020/QD-BYT) and ischemic heart disease guidelines (2187/2019/QD-BYT). In addition, the Ministry of Health has specified the conditions a facility must meet to provide the necessary treatment services in these two emergency case types. Currently, there are over 50 stroke centers nationwide that are capable of responding effectively to these two medical emergencies.

These centers are not evenly distributed across the country, and patients and primary and medical people often do not know which is the closest center to provide effective care. This leads to substantial unmet needs and delays in treatment with catastrophic consequences for patients. To support Vietnam's medical referral system, it would be helpful to use geospatial information systems to set up a smarter referral system for these two emergency conditions.

Vietnam is bigger compared to Timor-Leste both with respect to the surface and the number of inhabitants. Vietnam has a three-tier local government structure: provincial, district, and commune levels. In 2017, Vietnam's local administrations consisting of People's Councils and People's Committees are established in 63 provincial-level administrative units (58 provinces and 5 centrally run cities), 713 district-level ones and 11,162 commune-level ones. For Vietnam, we have multiple sets of open data available. For the beneficiaries layer, we have World Pop or the Facebook Research High-Resolution Human Settlements Layer available. The existing health facilities and the road network are mapped in OSM as well.

4.2. Tuning the heuristic

For the heuristic described in Section 2.4, we have developed a set of algorithms, which can be combined in various ways in order to find a solution to a maximum covering problem. There are multiple options for the construction algorithm, for the local first versus best improvement) and for path-relinking. For the construction algorithm, we used various values of α in `pgreedyexp`, `rpg`, `pgreedylin` and `rgreedy`, and we used $q = 0.02M$ in `sample`. Note that `greedy` is simply the `rgreedy` with $\alpha = 0$. We tested different values of p, S and for gridsize equal to 1 and 5 km, and analyzed the outcomes on maximum, average and standard deviation of the objectives, the average amount of overlap between two solutions, and the average time it took to construct the solutions. Based on all the tests, we conclude `sample` is the best construction method, and $\alpha = 0.02$ the best parameter setting. This construction method might not be the best method for every individual



(a) Stroke Care Facilities in Vietnam

(b) Hospitals in Vietnam (as proxy for potential stroke facilities)

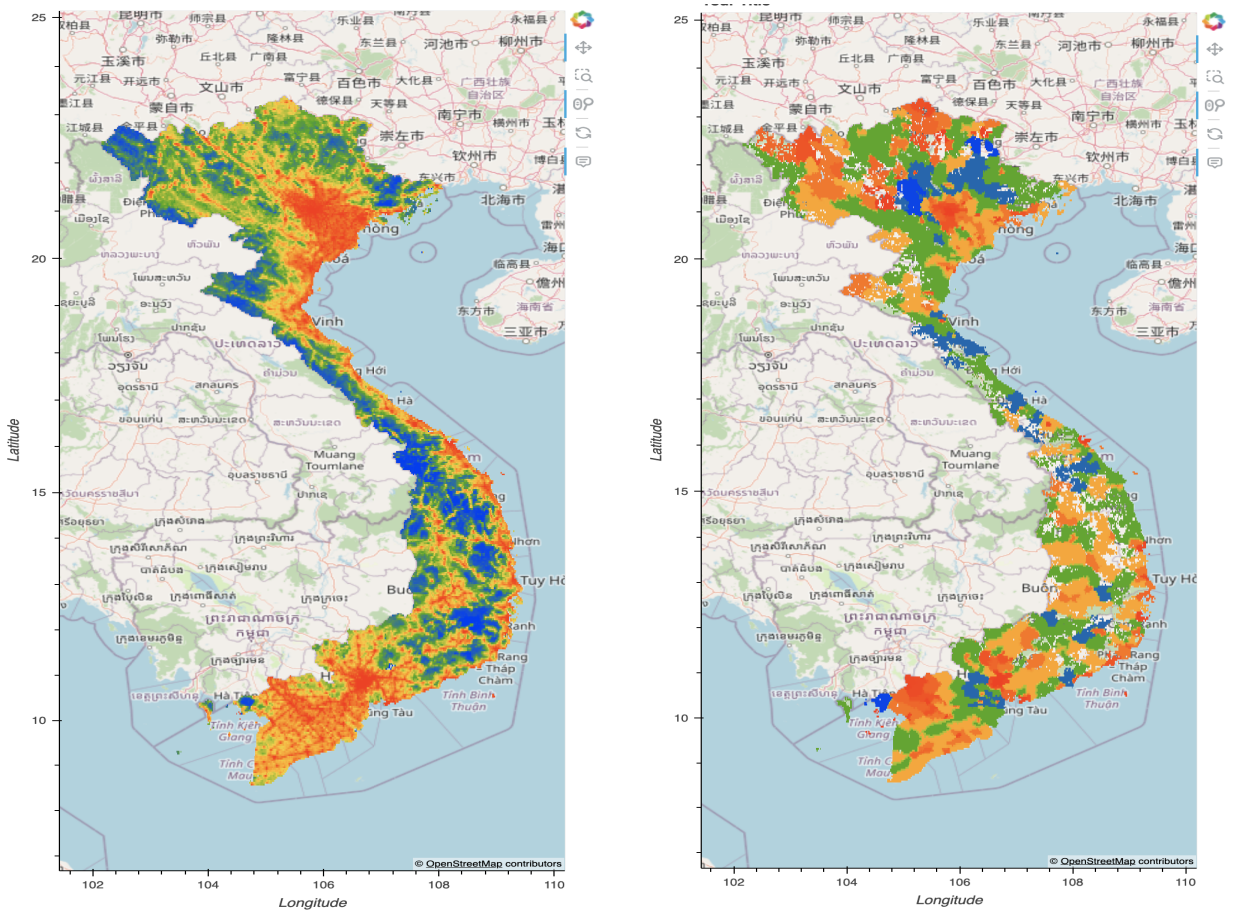
Figure 9 Existing Stroke Care Facilities and Hospitals in Vietnam

case, but we prefer to have a method that works well for a variety of test instances. See the Online Supplement for the details.

For local search, we have the options between best and first improvement local search. For best improvement, we can implement an advanced approach, where, as a result, we can finish the best improvement in a similar time as a first improvement local search. This is a strong achievement. It is now realistic to use the best improvement local search as a part of the GRASP heuristic, which would never have been possible with the simple approach.

Once a local optimum is found, we will explore the path between this local optimum and one of the elite solutions via path-relinking. In the Online Supplement, we describe a simple and advanced approach to path-relinking. Note that neither of these approaches is always faster than the other one. The reason is that the advanced approach has an initialization cost for filling the auxiliary structures, and sometimes the number of swaps that have to be considered in path-relinking is too small to compensate for this initialization cost.

Analyzing the contribution of path-relinking in the solving process, we found that 98% of all improving solutions found across all cases and methods were found during the path-relinking procedure. Of these improving solutions, 29% were found during forward path-relinking, and the other 71% during backward path-relinking. The proportion of the total available time spent on path-relinking ranges from $< 0.01\%$ up to 15% in some cases. This percentage is usually relatively high when the improvement search is restricted because relatively little time is spent in the local search phase in those cases.



(a) WorldPop UN Adjusted Population Counts for Vietnam (2020) at 1 km resolution

(b) High-Resolution Population Density Maps by Meta Data for Good (2020) at 30-meter resolution

Figure 10 Vietnam Population Count Datasets

4.3. Prescriptive analytics results

We divide this section into two parts: one based on using the existing hospitals in Vietnam, see Figure 9, as potential sites to install new stroke facilities and the other using a grid of candidate locations for new facilities.

Case 1: Only using existing hospital locations

Figure 11 shows the Pareto curves of the highest attainable coverage within 30 and 60 minutes driving time when expanding the set of stroke care units with optimal selections of hospitals, taken in steps of 20, until exhausting the list of candidates. After 360 hospitals the curve flattens for 60 minutes and the same happens for 30 minutes at 500. Note that as explained before the optimality is within a tolerance, specified in this case by an 'optimality gap' of 10^{-5}. We observed therefore fluctuations in the 'optimal' coverage of the flat areas of $\pm 5 \times 10^{-6}$. The choice of a MIPGap of 10^{-5} in this case is justified by the time needed to compute these Pareto curves being less than

one and a half minutes. The conclusion is that only using existing hospital locations as potential locations for stroke centers the maximal coverage is 87% for a 60 minutes threshold, and 71% for a 30 minutes threshold.

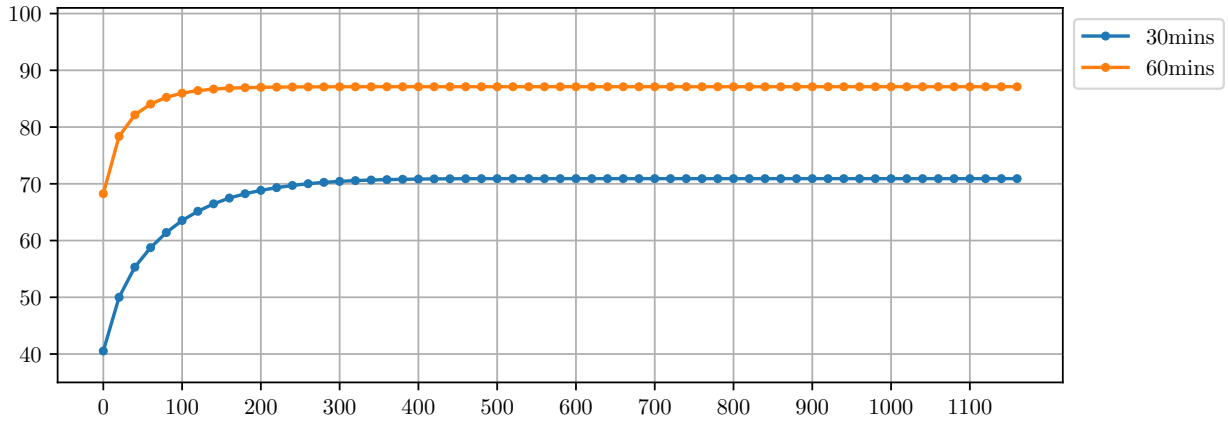


Figure 11 Pareto curves for expanding the existing stroke care facilities with a selection from the existing hospitals in Vietnam

Case 2: Using a huge grid of potential locations

Everything that we have discussed so far serves the goal of solving a huge scale maximal covering problem. Specifically, we want to be able to optimally allocate the stroke facilities in Vietnam, while considering every square kilometer of Vietnam as a potential facility location.

		Coverage (%)						
		grid-size = 5 km		**grid-size = 1 km**		**Time (s)**		
S	p	Opt	GRASP_5	GR+BI	GRASP_1	Opt	GR+BI	GRASP_1
20	20	48.868	48.868	49.447	49.484	11.3	8	42.7
	40	55.464	55.464	56.224	56.319	12.2	10.4	44.4
	60	60.483	60.483	61.325	61.452	15.7	17.2	769.3
	80	64.581	64.581	65.495	65.670	17.9	26.2	3099.5
	100	68.127	68.125	69.208	69.275	19.5	69.9	577.1
	200	80.038	80.013	81.137	81.396	32.6	274.1	6644.6
50	20	83.574	83.574	83.833	83.848	106.7	24.3	256
	40	90.137	90.137	90.353	90.469	33.9	40.1	2112.8
	60	93.777	93.773	93.938	94.064	177.7	65.2	4062.5
	80	96.091	96.091	96.196	96.377	157.3	96.1	4768.5
	100	97.550	97.541	97.592	97.743	436.8	162	3902.8
	200	99.896	99.877	99.876	99.913	96.1	367.4	7054.5
100	20	98.266	98.266	98.220	98.303	266.7	75.9	3280.5
	40	99.565	99.556	99.521	99.569	288.2	105.4	3440.3
	60	99.891	99.882	99.886	99.892	179.0	131.3	2743.1
	80	99.972	99.969	99.978	99.983	50.2	125.2	2169.8

Table 6 Numerical results for the Vietnam case. GRASP_1 and GRASP_5 are the GRASP heuristic for 1 km and 5 km grid, respectively.

Table 6 summarizes the most important results regarding the final outcomes of the developed heuristic. We run the heuristic both on the 5 km and 1 km grid. Gurobi could only solve the problem for the 5 km grid. We used Gurobi to find the optimal solution, with an allowed MIP-GAP of 0.01%. GRASP was run with `sample`, $\alpha = 0.02$ in the construction phase, combined with a best improvement local search, and back- and forward path-relinking. In each iteration, a choice for which path-relinking approach is used is made, based on the number of facilities that have to be swapped. GRASP uses the time limits given in Table 12 in the Online Supplement.

We observe that the GRASP outcomes on the 5 km grid are almost identical to the optimal values found by Gurobi: the largest difference is only 0.025%. When plotting the chosen facilities on the map of Vietnam, we observe that the accessibility of stroke facilities for people living in rural areas has increased substantially, whereas no extra facilities have been allocated in the areas where there were already a lot of facilities. In the North, almost all of the facility locations chosen by the heuristic, are also chosen in the optimal solution. In the South, there are some small differences.

We have also implemented a fast approximation approach `GR+BI`, which stands for greedy plus best improvement. This means we constructed one completely greedy solution and applied the best improvement local search to it. For 12 of the 16 cases, the outcome of this quick approach applied to the 1 km grid is already higher than the optimal value in the 5 km grid. In the Time columns `Opt` and `GR+BI`, we also compare the time taken by Gurobi to find the optimum (for the 5 km grid, and the time taken by the approximate approach to find a solution (for the 1 km grid). Sometimes our approximation method is much faster than the exact optimizer, and sometimes the opposite is true. Our method becomes relatively slow when p increases because the local search takes a lot longer to complete. In these cases, using the restricted first improvement strategy could be a good alternative.

The two GRASP_1 columns contain the objective values which are found by applying the full GRASP algorithm to the 1 km grid, and the time taken to reach this objective, respectively. For each case, the GRASP outcome for the 1 km grid is higher than the optimum of the 5 km grid. Especially for the instances covering a distance 20, the increase is substantial. If $p = 80$, 100 or 200, more than 1% of the Vietnamese inhabitants can be covered extra, as a direct result of using more detailed data. If we take into account that around 97 million people live in Vietnam, it is worthwhile to solve the problem at this level and to spend a maximum of 2 hours to run each instance. In other words, with our advanced optimization heuristic we can deliver access to about 1 million more people in Vietnam, with the same number of stroke centers.

5. Concluding remarks

The geospatial analytics techniques developed in this paper and implemented in the PISA toolbox have already shown their value in the applications in Timor-Leste and Vietnam. We now discuss

some limitations of the current toolbox PISA. The current methodology does not factor in the capacity of health facilities, nor does it consider the population's vulnerability to specific diseases or their relative wealth indexes. It is essential to recognize that beyond mere proximity, the capacity of a facility and the specific needs and capabilities of the population can drastically alter healthcare access dynamics.

While travel distance serves as the primary metric for accessibility in this study, it is not always the best indicator of actual accessibility. In many instances, distance does not equate to travel time. This discrepancy is particularly pronounced in regions with challenging terrains or roads that are in poor condition. A shorter distance might take a disproportionately long time to traverse, and vice versa, which could affect real-world health facility access.

The data sources, especially the road networks from OSM or Mapbox API and the health facilities data, pose another limitation. Without rigorous on-ground validations, there's an inherent risk of inaccuracies. These datasets, while comprehensive, might not capture the full picture or might include outdated or incorrect information, impacting the study's outcomes.

Identifying potential locations on a grid is a starting point for the prescriptive analytics deployed in this study, but determining the optimal placement of a health facility requires a more nuanced approach. Various considerations, such as being near resilient (all-season) and major roads, ensuring accessibility even during adverse conditions, and logistical considerations, must be evaluated in depth. A grid-based identification might suggest potential zones, but the exact location within that grid warrants a deeper dive.

One of the most important extensions for this study will be incorporating forward-looking population projections in the analysis. Infrastructure, especially health facilities, is designed for long-term service. Making decisions based solely on current data can be short-sighted. Factors such as urban migration, birth and death rates, and the overarching implications of climate change are pivotal in shaping future populations. Overlooking these can lead to infrastructural developments that might not align with the population's evolving needs in the coming decades.

For the future we further aim to extend the toolbox to be able to solve a larger scope of geospatial analytics problems. We describe six main directions:

1. In low- and middle-income countries often the waiting times for healthcare services are long. We would like to develop analytics techniques to predict waiting times and to optimize the assignment of extra capacities over healthcare centers such that the waiting times are reduced as much as possible. An example is the number of beds in stroke centers in Vietnam or the number of workers in healthcare centers in Timor-Leste.

2. We would like to develop analytics methods for mobile clinic planning. Although techniques developed in this paper could be partially used for these types of problems, the dynamic aspect

necessitates the development of additional methods. We started a project for mobile clinic planning in Kenya in cooperation with Amref.

3. We would like to develop analytics methods for emergency healthcare, such as those offered by ambulances. The emergency aspect adds new elements to the model for which new techniques have to be developed.

4. We would like to enhance the toolbox with techniques that can cope with different types of healthcare services. An example is the "normal" hospital service and stroke care service. The vicinity requirements may differ for these different services. Hence, we would like to optimize the locations of normal hospitals, and "super' hospitals that also offer the stroke care service.

5. The model could also be enhanced to consider different spatial exposures or risks (e.g., floods and avalanches), or health challenges (e.g. dengue hotspots). For example, we can detect those areas that may be most prone to pluvial or fluvial flood risk according to global models. Being able to observe risks, can be ensured a hospital is always available even when the risks occur. Next to the allocation of the inhabitants to a hospital within S kilometers, we can for example implement the criterion that, when risks occur, one should be able to reach the hospital within S_{risk} kilometers. Since for example flooding is a common problem in multiple low- and middle-income countries, this criterion can ensure better healthcare accessibility for the population in all situations.

6. Adding roads to the network or improving roads to make them, e.g., flood resilient, can improve accessibility significantly. Moreover, constructing a new road instead of building new hospitals can financially be more efficient. However, optimizing the road network involves a network structure with dependencies between roads, and the techniques developed in this paper are not suitable for this. In a current project we are developing such road network optimization techniques.

Moreover, besides applying our PISA toolbox to other healthcare services in other countries, we would like to apply our PISA toolbox to other applications in low- and middle-income countries. We started a project for optimizing water boreholes in Sudan with the Red Cross, and for waste collection in Nairobi with UN Environmental Programme.

We expect that geospatial analytics techniques, and especially optimization, could have a huge impact on a better accessibility of healthcare and other services in low- and middle-income countries. However, much research is needed to fully exploit these opportunities.

Acknowledgments

We want to thank Gurobi Optimization, LLC (2023) for being a supporter for Analytics for a Better World research projects and allowing us to use Gurobi Solver licenses free of charge.

References

Ahmadi-Javid A, Seyedi P, Syam SS (2017) A survey of healthcare facility location. *Computers & Operations Research* 79:223–263.

Boonmee C, Arimura M, Asada T (2017) Facility location optimization model for emergency humanitarian logistics. *International Journal of Disaster Risk Reduction* 24:485–498.

Calderín JF, Masegosa AD, Pelta DA (2017) An algorithm portfolio for the dynamic maximal covering location problem. *Memetic Computing* 9(2):141–151.

Church R, ReVelle C (1974) The maximal covering location problem. *Papers of the Regional Science Association*, volume 32, 101–118 (Springer-Verlag).

Cordeau JF, Furini F, Ljubić I (2019) Benders decomposition for very large scale partial set covering and maximal covering location problems. *European Journal of Operational Research* 275(3):882–896.

Daskin MS (2008) What you should know about location modeling. *Naval Research Logistics* 55(4):283–294.

Dejen A, Soni S, Semaw F (2019) Spatial accessibility analysis of healthcare service centers in gamo gofa zone, ethiopia through geospatial technique. *Remote Sensing Applications* 13:466–473.

Dönmez Z, Kara BY, Özlem Karsu, da Gama FS (2021) Humanitarian facility location under uncertainty: Critical review and future prospects. *Omega* 102:102393.

Foti F, Waddell P (2012) A generalized computational framework for accessibility: from the pedestrian to the metropolitan scale. *Technical Report, Institute of Theoretical Informatics, Karsruhe Institute of Technology* URL https://onlinepubs.trb.org/onlinepubs/conferences/2012/4thITM/Papers-A/0117-000062.pdf.

Galvão RD, Espejo LG, Boffey B (2000) A comparison of lagrangean and surrogate relaxations for the maximal covering location problem. *European Journal of Operational Research* 124(2):377–389.

Galvão RD, ReVelle C (1996) A lagrangean heuristic for the maximal covering location problem. *European Journal of Operational Research* 88(1):114–123.

Garey MR, Johnson DS (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)* (W. H. Freeman), first edition, ISBN 0716710455.

Gurobi Optimization, LLC (2023) Gurobi Optimizer Reference Manual. URL https://www.gurobi.com.

Jalil IA, Rasam ARA, Adnan NA, Saraf NM, Idris AN (2018) Geospatial network analysis for healthcare facilities accessibility in semi-urban areas. *International Colloquium on Signal Processing & its Applications* 14:255–260.

Juran S, Broer PN, Klug SJ, Snow RC, Okiro EA, Ouma PO, Snow RW, Tatem AJ, Alegana JGMVA (2018) Geospatial mapping of access to timely essential surgery in sub-saharan africa. *BMJ Global Health* 3:e000875.

Kelly C, Hulme C, Farragher T, Clarke G (2016) Are differences in travel time or distance to healthcare for adults in global north countries associated with an impact on health outcomes? a systematic review. *BMJ Open* 6:e013059.

Lab FC, for International Earth Science Information Network CIESIN Columbia University C (2016) High-resolution settlement layer (hrsl) source imagery for hrsl. accessed july 2023. URL https://www.worldpop.org/doi/10.5258/SOTON/WP00671.

Laguna M, Marti R (1999) Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing* 11(1):44–52.

Lorena LA, Pereira MA (2002) A lagrangean/surrogate heuristic for the maximal covering location problem using hillman's edition. *International Journal of Industrial Engineering* 9:57–67.

Máximo VR, Nascimento MC (2019) Intensification, learning and diversification in a hybrid metaheuristic: an efficient unification. *Journal of Heuristics* 25(4):539–564.

Máximo VR, Nascimento MC, Carvalho AC (2017) Intelligent-guided adaptive search for the maximum covering location problem. *Computers & Operations Research* 78:129–137.

Nicholl J, West J, Goodacre S, Turner J (2007) The relationship between distance to hospital and patient mortality in emergencies: an observational study. *Emergency Medicine Journal* 24:665–668.

Pandana (2023) URL http://udst.github.io/pandana/.

Pereira MA, Lorena LA, Senne EL (2007) A column generation approach for the maximal covering location problem. *International Transactions in Operational Research* 14(4):349–364.

Resende MG (1998) Computing approximate solutions of the maximum covering problem with grasp. *Journal of Heuristics* 4(2):161–177.

Resende MG, Martí R, Gallego M, Duarte A (2010) Grasp and path relinking for the max–min diversity problem. *Computers & Operations Research* 37(3):498–508.

Resende MG, Ribeiro CC (2003) A grasp with path-relinking for private virtual circuit routing. *Networks* 41(2):104–114.

Resende MG, Ribeiro CC (2010) Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. *Handbook of Metaheuristics*, 283–319 (Springer).

Resende MG, Ribeiro CC (2019) Greedy randomized adaptive search procedures: Advances and extensions. *Handbook of Metaheuristics*, 169–220 (Springer).

Resende MG, Werneck RF (2004) A hybrid heuristic for the p-median problem. *Journal of Heuristics* 10(1):59–88.

Resende MG, Werneck RF (2007) A fast swap-based local search procedure for location problems. *Annals of Operations Research* 150(1):205–230.

ReVelle C, Scholssberg M, Williams J (2008) Solving the maximal covering location problem with heuristic concentration. *Computers & Operations Research* 35(2):427–435.

Rodriguez FJ, Blum C, Lozano M, García-Martínez C (2012) Iterated greedy algorithms for the maximal covering location problem. *European Conference on Evolutionary Computation in Combinatorial Optimization*, 172–181 (Springer).

Saameli R, Kalubi D, Herringer M, Sutton T, de Roodenbeke E (2018) Healthsites.io: The global healthsites mapping project. *Technologies for Development*, 53–59 (Springer International Publishing), URL http://dx.doi.org/10.1007/978-3-319-91068-0_5.

Senne EL, Pereira MA, Lorena LA (2010) A decomposition heuristic for the maximal covering location problem. *Advances in Operations Research* 2010.

Stock R (1983) Distance and the utilization of health facilities in rural nigeria. *Social Science and Medicine* 174:563–570.

Weiss D, Nelson A, CA VR, et al (2020) Global maps of travel time to healthcare facilities. *Nature Medicine* 26:1835–1838.

WorldPop (2020) Global 1km population total adjusted to match the corresponding unpd estimate. URL http://dx.doi.org/10.5258/SOTON/WP00671.

Zarandi MH, Davari S, Sisakht SA (2011) The large scale maximal covering location problem. *Scientia Iranica* 18(6):1564–1570.

Appendix A. Data Sources for Timor-Leste Case

Category	Data Source	Description
Population	High-Resolution Population Density Maps (Population: 1.32 million with 164978 points)	A high-resolution (30-meter) population density map of the world, created by Meta's Data for Good Program using satellite imagery and ground surveys. 2020 which is the most recent version is used for this study
	WorldPop Population Counts (Population: 1.32 million with 18089 points)	The spatial distribution of population in 2020 with country total adjusted to match the corresponding UNDP estimate. The dataset is available to download in GeoTIFF and ASCII XYZ format at a resolution of 30 arc (approximately 1 km at the equator)
Transport	MapBox Isochrone API	The Mapbox Isochrone API computes areas that are reachable within a specified amount of time from a location and returns the reachable regions as contours of polygons or lines that you can display on a map. This API also supports contours based on distance. (Accessed July 2023)
	Estrada Road Asset Management System	Parvathy: please add some text here
Geotagged Health Facilities	Healthsites.io	The Global Health Sites Mapping Project is an initiative to create an online map of every health facility in the world and make the details of each location easily accessible. This study used 187 hospitals and clinics downloaded from Healthsites.io in July 2023.
	World Health Organization	347 Health Facilities data collected by World Health Organization given access for research by World Bank
Potential Locations	Generated by scripts	A 1 km x 1 km grid was superimposed on Timor-Leste's geographical administrative boundary data from Global Administrative Boundary Database (GADM)

Table 7: Data Source, Category, and Description for Timor-Leste Case

A Implementation of the GRASP heuristic

In this Appendix we describe an efficient implementation of the GRASP heuristic in details, since it has to be efficient in both time and memory. We use the notation of Sections 2.3 and 2.4 of the main paper, and only introduce new variables.

A.1 Sparse datastructures

For illustration purpose, we will use a small example throughout the paper.

$$\mathcal{M} = \overbrace{(0 \quad 1 \quad 2 \quad 3 \quad 4)}^{m} \qquad \mathcal{N} = \overbrace{(0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6)}^{n}$$

$$v = \overbrace{(20 \quad 10 \quad 5 \quad 10 \quad 3 \quad 30 \quad 5 \quad 8)}^{n}$$

We define the binary matrix $C \in \{0,1\}^{m \times n}$, in which $C_{ij} = 1$ if the distance between facility i and population point j is at most α km. The binary matrix for our small example is:

$$C = \overbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}}^{n} \Bigg\} m$$

We introduce coverage vector *cov*, which specifies how many facilities cover each population point in the current solution. Hence, we have $cov = C^T y$. A population point is covered if at least one active facility is covering it. More precisely: we have $x = \mathbb{1}_{\{cov \geq 1\}}$. Again, we illustrate this with our example:

$$y = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \Bigg\} m, \qquad C = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \begin{aligned} &\rightarrow cov = \overbrace{(1 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 1)}^{n} \\ &\rightarrow x = \mathbb{1}_{\{cov \geq 1\}} = (1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1) \end{aligned}$$

Note that x can be calculated from *cov* in $O(n)$ time. For the objective function value *obj*, which is the total number of covered people, we have $obj = x^T v$, and has complexity $O(n)$. For the example we have:

$$obj = x^T v = 20 + 10 + 10 + 5 + 8 = 53.$$

In the construction, local search and path-relinking phase, many different solutions have to be evaluated, and a new solution that is considered always differs by only one facility from the previous solution. So instead of calculating the objective value from scratch for every new solution, it is beneficial to have a way of efficiently computing the objective value change caused by a small modification in the solution. The coverage vector *cov* allows us to do the latter. For example, if one active facility is swapped with an inactive facility, we can simply take the coverage vector of the old solution, subtract the row of C corresponding to the leaving facility and add the row of the entering facility. The resulting vector is the coverage vector of the new solution, the same vector as you would obtain by summing over the p rows of C corresponding to the new solution. Adding and subtracting a row of C has complexity $O(n)$. Summing over the p rows of C on the other hand, has complexity $O(pn)$. As we will test instances up to $p = 200$, using the coverage vector is much more efficient. Both operations have complexity $O(n)$, so the total complexity of calculating an objective value change is $O(n)$.

Since the matrix is in general very sparse, we can save much memory by using a sparse matrix representation. We use two dictionaries. The first dictionary, *row*, represents the rows of the binary matrix. The potential facility IDs are the keys of the dictionary, and each value is a vector containing all population points that can be covered by the corresponding facility. The *row* dictionary makes it easy to look up which population points are covered by a certain facility, but it is very inefficient to determine which facilities can cover a specific population point. Therefore, we create a second dictionary, *col*, which saves the same information, but structured in a different way. Each key is a

population point, and the array that belongs to it contains all the potential facilities IDs that can cover that point. The *row* and *col* dictionary for the example are:

$$C = \overset{n}{\overbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}}}^{m} \qquad row = \begin{Bmatrix} 0: & [0,3,6] \\ 1: & [1,2] \\ 2: & [1,3,5] \\ 3: & [3,5] \\ 4: & [4] \end{Bmatrix}, \qquad col = \begin{Bmatrix} 0: & [0] \\ 1: & [1,2] \\ 2: & [1] \\ 3: & [0,2,3] \\ 4: & [4] \\ 5: & [2,3] \\ 6: & [0] \end{Bmatrix}$$

In this appendix, $row[f]$ represents the array with population point IDs that are covered by facility f, and $col[i]$ represents the array with facility IDs that can cover population point i. The information in the dictionaries can only be accessed by iterating over the keys, so logical indexing does not work anymore. Therefore, we do not store the solution as the binary vector y, but we store the solution as an array with the active facility IDs, which we name F^*. For the example we have:

$$y = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \rightarrow F^* = \begin{pmatrix} 0 & 2 \end{pmatrix}$$

Table 1 shows the total memory usage of the two dictionaries for each Vietnam case.

grid-size	$S = 20$	$S = 50$	$S = 100$
10	0.04	0.07	0.11
5	0.05	0.18	0.40
1	0.51	4.30	9.70

Table 1: Total memory usage of *row* and *col* in GB

Table 1 shows that the memory usage is reasonable. Even for a maximum distance of $S = 100$ km and a grid-size of 1 km, the total memory usage fits within 10 GB. The sparse matrix implementation with dictionaries also allows us to adjust the *cov* vector and calculate the objective value change in a more efficient way. We check for each point in $row[f_e]$ if its coverage value is equal to zero, and collect the points for which this is true in a set named $\mathcal{IN}$. These are the points that contribute to the objective value increase caused by f_e entering the solution, because they were not yet covered in the old solution, but will be in the new solution. After constructing $\mathcal{IN}$, we increase the *cov* value by one for all of the points in $row[f_e]$.

The mathematical formulation of $\mathcal{IN}$ is as follows: $\mathcal{IN} \leftarrow \{i \in row[f_e] : cov(i) = 0\}$. Note that in the code, $\mathcal{IN}$ is actually a Numpy array and not a set, to increase compatibility with the other data structures. Even so, we have chosen for the set notation to improve readability, and because $\mathcal{IN}$ actually represents a set of points. The values in $\mathcal{IN}$ are the population points that are newly covered by the facility f_e, because if they were already covered in the old solution, the coverage value would be at least 1. This means that the objective value increase can be calculated by summing $v(i)$ for all $i \in \mathcal{IN}$. Following a similar reasoning, $\mathcal{OUT} \leftarrow \{i \in row[f_l] : cov(i) = 1\}$ contains all population points which become uncovered if f_l leaves the solution. Summarizing, the following two formulas can be used to calculate the objective value change caused by a facility leaving or entering the solution:

$$\Delta obj = \sum_{i \in \mathcal{IN}} v(i) \qquad \text{with} \qquad \mathcal{IN} \leftarrow \{i \in row[f_l] : cov(i) = 0\}$$

$$\Delta obj = -\sum_{i \in \mathcal{OUT}} v(i) \qquad \text{with} \qquad \mathcal{OUT} \leftarrow \{i \in row[f_l] : cov(i) = 1\}.$$

To illustrate this process, we will show the steps that follow facility 2 entering our example solution, which only contains facility 0 so far:

$$F^* = (0)\,, f_e = 2 \rightarrow F^* = \begin{pmatrix} 0 & 2 \end{pmatrix}$$

$$
\begin{aligned}
row[2] &= [1, 3, 5] \\
cov &= (1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1) \\
\mathcal{IN} &= \{i \in [1, 3, 5] : cov(i) = 0\} \\
&= \{1, 5\} \quad (\text{since } \{cov(1) = 0, cov(3) = 1, cov(5) = 0) \\
cov &= (1 \quad 0^{+1} \quad 0 \quad 1^{+1} \quad 0 \quad 0^{+1} \quad 1) \rightarrow (1 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 1) \\
\Delta obj &= v(1) + v(5) = 10 + 30 = 40.
\end{aligned}
$$

Let k be the number of elements in $row[f_e]$, then the complexity of updating the coverage vector is equal to $O(k)$, because only k additions are executed. Determining which elements are in $\mathcal{IN}$ also takes $O(k)$ time, because the coverage value is only checked for the k values in $row[f_e]$. Calculating the Δobj again consists of summing a maximum of k values, so the whole process has complexity $O(k)$. If we redefine k as the maximum length of $row[f]$ for any $f \in \mathcal{M}$, we can state that any addition, removal or swap of facilities has complexity $O(k)$, following the same reasoning as above. The fact that we have sparse data implies that $k \ll n$, so the complexity of $O(k)$ is a substantial improvement compared to $O(n)$, which was the complexity of calculating an objective value change with the old approach.

A.2 Construction phase

Greedy approach

The greedy value of a facility is often not affected when another facility is added to the solution. For example, if a facility in the south of Vietnam is chosen, it will not affect the greedy values of the potential facilities in the North. In fact, the greedy value of a facility f only changes if there is overlap between the population points that are covered by f, and the population points that are uniquely covered by the chosen facility f_{sel}. If this is the case, the greedy value of facility f will decrease, because there is no added value in covering a population point by more than one facility.

There are two ways to determine the new greedy values of the affected facilities. The first option is to iterate over all population points that are newly covered by f_{sel}, and adjust the greedy values of the other facilities that could have covered this point by the appropriate amount. The second method is to find the set of affected facilities, and simply recalculate the greedy value from scratch for each of these affected facilities. This second option turned out to be faster, so we opted for this method.

With the help of both the *row* and *col* dictionary, we find the list of affected facilities. Once the coverage vector has been updated, the set $\mathcal{IN}_{sel} \leftarrow \{i \in row[f_{sel}] : cov(i) = 1]\}$ contains all population points that are uniquely covered by the last selected facility. For each point in $\mathcal{IN}_{sel}$, we look up which facilities can cover it in the *col* dictionary, and concatenate the corresponding array to Z. Z is the array with all the facility IDs for which the greedy metric changes. Many facility IDs will occur multiple times in Z, thus after iterating over all points in $\mathcal{IN}_{sel}$, we remove all duplicates, ending up with the desired list of affected facilities. Despite that this leads to a big reduction in the run-time, we can see in line 8 of Algorithm 1 that the worst-case efficiency of the greedy algorithm is not affected by this trick, because in theory all m facilities can occur in Z. In practice this is never the case however, which explains the observed time reduction.

The last big improvement was utilizing the sparsity of the data. Computing the greedy metric is reduced from $O(n)$ to $O(k)$ time, with $k = \max_{f_e} |row[f_e]|$, because the coverage values are only checked for the population points in $row[f_e]$. Another advantage of this trick is that we do not need to update the coverage vector before calculating the greedy value of a candidate facility. It is valuable that *cov* can be used without updating it, because many greedy values have to be calculated before picking the best one. Now, the same *cov* vector can be used for all of these calculations.

In algorithm 1, the pseudo-code of the resulting greedy construction algorithm is given. In the comments, we have indicated the time complexity of every line, however we have omitted $O(1)$ to improve readability. The indentation of $O(*)$ reflects the indentation of the loop in which the line is executed, which shows how often each line has to be executed. We will use these comments to analyze the complexity of the full algorithm. Finally, $k = \max_f |row[f]|$ is the maximum number of population points that a potential facility can cover, and $l = \max_i |col[i]|$ is the maximum number of facilities

by which a population point can be covered. Note that the sparsity of the data implies $k \ll n$ and $l \ll m$.

Algorithm 1: Greedy Function

```
1  Function Greedy(M, N, v, row, col, p)
2      Z: Array with all potential facilities for which the greedy metric changes
3      IN: Set of additionally covered population points
4
5      Initialize greedy and cov as zero vectors of length |M| and |N| respectively
6      Initialize obj ← 0, Z ← M, and F* as an empty list
7      for iter = 1, ..., p do                                          // O(p)
8          for f ∈ Z do                                                 //   O(m)
9              IN ← {i ∈ row[f] : cov(i) = 0}                           //     O(k)
10             greedy(f) ← Σ_{i∈IN} v(i)                                //     O(k)
11         end
12         f_sel ← argmax_f greedy(f)                                   //   O(m)
13         Append selected facility f_sel to solution F*
14         Update obj: obj ← obj + greedy(f_sel)
15         Update cov: let cov(i) ← cov(i) + 1  ∀i ∈ row[f_sel]          //   O(k)
16         Reset Z to an empty array
17         IN_sel ← {i ∈ row[f_sel] : cov(i) = 1}                       //   O(k)
18         for i ∈ IN_sel do                                            //   O(k)
19             Concatenate all elements in col[i] to Z                  //     O(l)
20         end
21         Remove duplicate values from Z                               //   O(kl)
22     end
23     return F*, cov, obj
24 end
```

Since $k \ll n$, the complexity of the greedy algorithm is O(pmk).

Combine randomness and greediness

In the construction phase we combine randomness and greediness in four ways, aiming to generate high quality solutions with sufficient variation between them:

- *randomized greedy (rgreedy)*: For this approach only the selection criterion in line 12 of Algorithm 1 has to be changed. Once the *greedy* vector has been computed, we sort the values in descending order. We save which index the sorted values had in the original *greedy* vector. We then uniform randomly select one of the top $\lfloor \alpha * p \rfloor$ indices, this will be the selected facility f_{sel}. This facility will be added to the solution, and the rest of the algorithm continues unchanged.

- *random plus greedy (rpg)*: After line 5 and 6 of Algorithm 1, we add some intermediate steps. The first $\lfloor \alpha * p \rfloor$ facilities are chosen at random from the list of potential facilities. These facilities are added to F^*, then *cov* and *obj* are updated accordingly. Line 7 iterates from 1 up to $p - \lfloor \alpha * p \rfloor$, such that in total p facilities are selected.

- *proportional greedy (pgreedy)*: We make the same adjustments as in $rgreedy$, except that the random selection has a different underlying distribution, which biases the selection towards facilities with a higher rank. There are several bias functions, and they can be used to assign a probability to each of the top $\lfloor \alpha * p \rfloor$ facilities. The selected facility f_{sel} will be chosen at random from these facilities, according to the customized probability distribution.

- *sample greedy (sample)*: This approach differs the most from the standard greedy algorithm. The greedy vector needs to be reset as a zero vector at the beginning of each iteration. Then, Z is filled with a random sample of q elements from $\mathcal{M}$. For these q elements, the greedy value is calculated, the best facility is selected, and F^*, *obj* and *cov* are updated. The steps in line 16 up to and including 21 in Algorithm 1 are not executed. At each iteration we just calculate the q greedy values of the facilities in the random sample, so determining which greedy values of the other facilities would be affected, is irrelevant.

A.3 First improvement local search

Once a solution has been constructed, the local search phase starts. In this section we will discuss a first improvement local search, in the next section we will elaborate on the best improvement strategy. In Algorithm 2 we show the basic structure of a first improvement local search. For each facility f_l in the current solution F^*, the objective value decrease w_{loss} associated with f_l leaving the solution is calculated. This is compared to the objective value increase w_{gain} caused by f_e entering the solution, taking the place of f_l. For a given leaving solution, each facility in the set of candidates is considered. The set of candidates C consists of all potential facilities that are not in the current solution. As soon as we find a profitable swap, i.e. $w_{gain} > w_{loss}$, we execute the swap and restart the search. We also show the time complexity of each line, so we can determine the time complexity of 1 iteration. Given that an objective value change can be calculated in $O(k)$ time, the total time complexity is $O(pmk)$.

Algorithm 2: Basic First Improvement Local Search

```
1  Function Local_Search(F*, C)
2      for f_l ∈ F* do                                              // O(p)
3          Compute objective value decrease w_loss by f_l leaving F*  //   O(k)
4          for f_e ∈ C do                                           //   O(m)
5              Compute objective increase w_gain by f_e entering F*\{f_l}  //   O(k)
6              if w_gain > w_loss then
7                  F* = F* ∪ {f_e}\{f_l}
8                  C = C ∪ {f_l}\{f_e}
9                  Return to line 2
10             end
11         end
12     end
13 end
```

w_{gain} and w_{loss} can be calculated efficiently using the technique discussed earlier. The sum of the population of all points in $\mathcal{OUT} \leftarrow \{i \in row[f_l] : cov(i) = 1\}$ is equal to w_{loss}, and similarly, all points in $\mathcal{IN} \leftarrow \{i \in row[f_e] : cov(i) = 0\}$ contribute to w_{gain}. An important step in the local search process, is that the coverage vector needs to be updated after calculating w_{loss}, as if f_l already left the solution. With the new coverage vector as a starting point, calculating w_{gain} of a candidate facility is equivalent to calculating its greedy value. However, if w_{gain} is smaller than w_{loss} for all $f_e \in C$, f_l will stay in the solution, so the change of the coverage vector needs to be undone, and the local search continues with the next facility in F^*.

If F^* and C are searched in the same order each time, the same swaps are considered many times. Strictly speaking, whether a certain swap is profitable also depends on the solution F^*, so the same swap might not be profitable the first time you consider it, but it could become profitable as the incumbent solution changes. However, in each step of the local search only one facility changes in the solution, so it might be more efficient to focus on examining new swaps. A simple way to make sure that the solution and candidate list are not searched in the same order each time, is to apply a random shuffle to F^* and C each time a new solution is found. Using the sparse matrix implementation, exploiting the sparsity to efficiently calculate objective value changes, and applying a random shuffle to F^* and C^* after each swap, accelerates the first improvement local search substantially.

To speed up the process even more, we reduce the neighbourhood of each facility, by only allowing swaps with facilities which are relatively close. As a result, less swaps have to be considered, which accelerates the local search. On the other hand, the fact that we consider fewer swaps could also result in a worse local optimum. We take the following steps to determine the restricted neighbourhood: let f_l denote a facility that is considered to leave the solution, then calculate the haversine distance from f_l to all other facilities, and only keep the ones which are located within *border* km. Next, sort the remaining candidates according to their distance to f_l, such that the local search will start with the closest facility.

There are two more additions that we made to the algorithm. The first one, is that we store the restricted candidate list of each facility in a dictionary C_{dict}. At the start of the algorithm, we calculate and save the candidate list for each facility in solution F^*. Then each time a swap is executed, we

add the candidate list of the entered facility to the dictionary. Using the dictionary saves computation time, because we only determine which candidates are close enough once. The second extension, is the vector *no_improv*, which keeps score of how many times the restricted candidate list of a given facility has been searched without success. After the random shuffle of F^*, we sort the facilities in the solution according to their corresponding values in the *no_improv* vector, such that the facility which has been researched the least is prioritized.

The pseudo-code of the final algorithm, including the time complexity of each line, is given in Algorithm 3. We define z as the maximum number of facilities that lie within the *border* of any of the facilities. The size of z compared to m depends on the value chosen for *border*, but we assume that we choose *border* in such a way that $z \ll m$. Consider the 5km grid data with $border = 50$ for example, then $z = 349$, which is 40 times smaller than $m = 14,235$.

Algorithm 3: Local Search Function

```
1  Function Local_Search(F*, cov, obj, M, v, row, border)
2      d_le: haversine distance between facility f_l and facility f_e
3      OUT: Set of additionally uncovered population points
4      C_dict: Dictionary, stores border restricted candidate list for a given facility
5      no_improv: Counts # unsuccessful searches of each facility's candidate list
6
7      flag = True
8      Initialize no_improv as a zero vector of length |M|
9      for f_l ∈ F* do                                              // O(p)
10         Calculate d_le ∀f_e ∈ M                                  //   O(m)
11         C_dict[f_l] ← {f_e ∈ M : d_le ≤ border}                  //   O(m)
12     end
13     while flag = True do
14         flag ← False
15         First random shuffle F*, then sort according to no_improv // O(p)
16         for f_l ∈ F* do                                          // O(p)
17             C ← C_dict[f_l]                                      //   O(z)
18             Sort C in increasing order according to distance d_le //   O(z)
19             OUT ← {i ∈ row[f_l] : cov(i) = 1}                    //   O(k)
20             w_loss ← Σ_{i∈OUT} v(i)                              //   O(k)
21             Update cov: cov(i) ← cov(i) − 1 ∀i ∈ row[f_l]        //   O(k)
22             for f_e ∈ C do                                       //   O(z)
23                 IN ← {i ∈ row[f_e] : cov(i) = 0}                 //     O(k)
24                 w_gain ← Σ_{i∈IN} v(i)                           //     O(k)
25                 if w_gain > w_loss then
26                     flag ← True
27                     obj ← obj + w_gain − w_loss
28                     Update cov: cov(i) ← cov(i) + 1 ∀i ∈ row[f_e] // O(k)
29                     Delete f_l from F* and insert f_e
30                     Calculate d_ek ∀k ∈ M                        // O(m)
31                     C_dict[f_e] ← {f_k ∈ M : d_ek ≤ border}       // O(m)
32                     Return to line 9
33                 end
34             end
35             if flag = False then
36                 no_improv(f_l) ← no_improv(f_l) + 1
37                 Undo change in cov: cov(i) ← cov(i) + 1 ∀i ∈ row[f_l]
38             end
39         end
40     end
41     return F*, cov, obj
42 end
```

The local search algorithm has an initialization cost of $O(pm)$, which is needed to construct the candidate dictionary for the facilities in the starting solution. We note that the complexity indentation

of lines 28, 30 and 31 shows that you pass these lines only once per local search iteration. To see why this is true, recall that a swap is executed as soon as the algorithm finds an improving swap, which finishes that local search iteration. We conclude that one local search iteration has worst-case complexity $O(pzk + m)$, with a one-time initialization cost $O(pm)$. Given that $z \ll m$, this is a substantial improvement if we compare it to the basic local search algorithm without the restriction on distance, which had complexity $O(pmk)$.

A.4 Best improvement local search

In this section we discuss two ways to implement a best improvement local search. The first one is the simple approach, which only requires a few adaptations to our first improvement local search in Algorithm 3. The alternative method is not as straightforward and requires a more extensive explanation, therefore we will refer to it as the advanced approach.

The alterations needed to transform Algorithm 3 into a best improvement local search are trivial. The random shuffle of F^* and the *no_improv* vector are removed, because the order in which you consider the swaps is not relevant anymore. Furthermore, the candidate list C consists of all potential facilities not in the current solution, without a restriction on the distance. Finally, instead of executing a profitable swap immediately, the algorithm keeps track of the best swap, which is executed once all possible swaps have been considered. The rest of the algorithm remains the same.

The strength of our own implementation of the first improvement local search is that we try to minimize executing unpromising calculations. To this goal, we restrict the neighbourhood of each chosen facility, and prioritize facilities which have been investigated the least. However, these advantages are lost once our approach is transformed into a best improvement local search. Considering all swaps in every iteration implies that many calculations are repeated in subsequent iterations, so it takes longer to reach a local optimum. Ideally, we would like to eliminate these repetitive computations, which is where the advanced approach comes in.

In the advanced approach, we first calculate the objective value change corresponding to all possible swaps, and save these outcomes. Then we pick the best one, and determine which of other outcomes are affected by the chosen swap. After updating these affected values, we repeat this process, until a local optimum is found. Note the similarity to our greedy algorithm, in which we calculate the greedy metric for every facility only once, and update the affected greedy values as facilities are added to the solution. To implement this idea for a local search is more complicated than for the greedy algorithm however, as it considers swaps instead of only adding facilities.

In the rest of this section will describe the implementation for the advanced approach for the best improvement local search. Algorithm 4 shows the resulting algorithm.

Constructing *loss*, *gain*, and *extra*

This implementation makes use of three auxiliary data structures, in which certain calculations are saved, such that these do not have to be repeated. The first structure is the *loss* vector, each item in the *loss* vector reflects the added value of that facility in the current solution. In other words, it is the amount with which the objective value would decrease if that facility would be removed from the current solution. The second structure is the *gain* vector, which indicates for each potential facility which is not in the current solution by how much the objective value would increase if that facility would be added. This vector is identical to the greedy vector, and each entry is computed using $\mathcal{IN}$. More precisely, the loss and gain values are computed as follows:

$$loss(f_l) = \sum_{\{i \in row[f_l] : cov(i)=1\}} v(i)$$

$$gain(f_e) = \sum_{\{i \in row[f_e] : cov(i)=0\}} v(i)$$

Below the *loss* and *gain* vector are given for the example that we have used throughout this section. The numbers on the right of the dashed line indicate to which facilities the values correspond. Take facility 0, this facility covers population points 0, 3 and 6. Of these population points, 3 is also covered by facility 2 in the current solution, so only the number of inhabitants belonging to 0 and 6, which

is 20 and 8, contribute to the value in *loss*. The *loss* value for facility 2 can be computed similarly. Next, we consider facility 1 in the *gain* vector, it could cover population points 1 and 2 if it would be added to the solution. However, point 1 is already covered by facility 2, so the *gain* value only reflects the number of inhabitants of population point 2, which is 8. The rest of the *gain* vector can be filled likewise.

$$loss = \begin{matrix} p \\ \left\{ \begin{pmatrix} 20+8 \\ 10+5 \end{pmatrix} \right. \end{matrix} \begin{matrix} \vdots \\ 0 \\ \vdots \\ 2 \end{matrix} \qquad gain = \begin{matrix} m-p \\ \left\{ \begin{pmatrix} 8 \\ 0 \\ 12 \end{pmatrix} \right. \end{matrix} \begin{matrix} \vdots \\ 1 \\ \vdots \\ 3 \\ \vdots \\ 4 \end{matrix}$$

Using the *loss* and *gain* vector, it is easy to calculate the effect of either adding or removing a facility to or from the current solution. However, this is not the same as executing these two actions simultaneously, i.e. executing a swap. To understand this concept, consider swapping facilities 2 and 1 in our example. Facility 2 covers population point 1 uniquely, so the corresponding 10 people contribute to its *loss* value. As this point is already covered in the current solution, it does not contribute to the *gain* value of facility 1, even though this facility could also cover population point 1. Indeed, if facility 0 would be swapped for facility 1, it would have no extra value that facility 1 covers this point. However, because it is facility 2 that is leaving the solution, population point 1 would have been uncovered, if was not for facility 1 covering it in the new solution. In this case it is valuable that facility 1 covers population point 1, but this is not visible from the *gain* value. There clearly exists interaction between leaving and entering facilities which has effect on the objective value change caused by a swap. This interaction term is captured in the third auxiliary structure, *extra*.

The formula used to compute the *extra* value for a given entering facility f_e and leaving facility f_l, is shown below. The sum is taken over all population points in $\mathcal{OUT}$, so they are uniquely covered by the leaving facility in the current solution. These points will become uncovered unless the entering solution covers them, and that is exactly what the *extra* value should capture. For each of these population points i we check if the entering facility f_e can cover this population point, which is the case if f_e occurs in $col[i]$. Summarizing, the *extra* value is the sum of the number of inhabitants that correspond to the population points that are uniquely covered by f_l, and can be covered by f_e. Note that this interaction term can never be negative.

$$extra(f_e, f_l) = \sum_{\{i \in row[f_l] : cov(i) = 1\}} \mathbb{1}_{\{f_e \in col[i]\}} * v(i)$$

All the *extra* values are collected in a matrix of size p by $m - p$. The matrix displays the interaction term for each pair of facilities that can be swapped. Below the *extra* matrix for our simple example is depicted, with under and to the right of the dashed lines the facilities to which the numbers correspond. Facility 0 has no interaction with the candidate facilities that can be swapped in. Population points 0 and 6 can only be covered by facility 0, so for sure they do not cause any interaction. Population point 3 is covered by facility 0, but also by facility 2 in the current solution. Regardless of the facility that leaves the solution, population point 3 will stay covered, so it if the entering facility also covers this point, this has no added value. We already discussed the interaction between facility 2 and 1, which is also visible in the matrix now. Similarly, replacing facility 2 by 3 results in an interaction term caused by population point 5.

$$extra = \begin{matrix} p \\ \left\{ \overbrace{\begin{pmatrix} 0 & 0 & 0 \\ 10 & 5 & 0 \end{pmatrix}}^{m-p} \right. \\ \underbrace{\quad 1 \quad 3 \quad 4 \quad}_{} \end{matrix} \begin{matrix} \vdots \\ 0 \\ \vdots \\ 2 \end{matrix}$$

Once *loss*, *gain* and *extra* have been computed, the profit formula given below can be used to calculate which swap generates the best profit. It is equal to the added value of the entering facility, minus the lost value of the leaving facility, plus the interaction term of these two facilities. Let f_l^* and f_e^* be the leaving and entering facility-pair that maximizes this profit function for all possible swaps. If $profit(f_l^*, f_e^*)$ yields a profit larger than 0, the swap will be executed. Otherwise, there exists no swap which improves the objective function value, which means that the current solution is a local optimum, and the best improvement local search is finished.

$$profit(f_l, f_e) = gain(f_e) - loss(f_l) + extra(f_l, f_e)$$

Turning back to our example, the profit matrix can be generated from the previously computed data structures, and is given below. We observe there is only one swap that results in a positive profit, thus facility 2 will leave the solution such that facility 1 can enter.

$$profit = \begin{array}{c|c} \begin{pmatrix} -20 & -28 & -16 \\ 3 & -10 & -3 \end{pmatrix} & \begin{matrix} 0 \\ 2 \end{matrix} \\ \hline \begin{matrix} 1 & 3 & 4 \end{matrix} & \end{array}$$

Solving discrepancy between indices and facility IDs

In our explanation until now, we have assumed that it is possible to translate a facility number into the correct index of a data structure. For example, if we want to know the *extra* value for $f_l = 2$ and $f_e = 1$ in our own illustrative case, we can simply look up $extra(2, 1) = 10$. However, in coding language this is not so straightforward. We use Numpy arrays to store all the large data structures, and the fastest way to access and alter values is via their indices. Thus, in our code we would need to look up $extra(1, 0)$ to get the desired *extra* value for $f_l = 2$ and $f_e = 1$. Clearly, there exists a discrepancy between the indices in the data structures and the facility ID they represent.

In order to deal with this discrepancy, we make some adjustments. First, the width of the *extra* matrix and the length of the *gain* vector is increased to m, instead of $m - p$. The altered versions of these data structures are shown below. In our application $p << m$, so the increase of memory usage is negligible. The big advantage is that there exists no more discrepancy in *gain* and in the columns of *extra*. As we will explain later on, the facility IDs in *col* will be used to indicate which columns in *extra* and which rows in *gain* are affected by a swap. Being able to skip the step of translating the values in *col* to their corresponding indices improves the speed of the algorithm significantly.

$$gain = \begin{array}{c|c} \begin{pmatrix} 0 \\ 8 \\ 0 \\ 0 \\ 12 \end{pmatrix} & \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \end{array} \qquad extra = \begin{array}{c|c} \overbrace{\begin{pmatrix} 28 & 0 & 0 & 0 & 0 \\ 0 & 10 & 15 & 5 & 0 \end{pmatrix}}^{m} & \begin{matrix} 0 \\ 2 \end{matrix} \\ \hline \begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix} & \end{array}$$

We could solve the discrepancy for in the *loss* vector and in the rows of the *extra* matrix in the same way, but this would not be feasible in terms of memory usage. The *extra* matrix would need $m \times m$ data points, and we would run into the same memory issues as discussed in section A.2 of this appendix. Therefore, we have come up with an alternative way to solve this issue. Constructing *loss* and *extra* for the first time, can simply be done row by row. Updating them after a swap is more complicated, and will be explained in detail later on. For now we just note that our method needs the vector ϕ. This vector indicates for every uniquely covered population point, the index of its covering facility in the current solution. The ϕ corresponding to our example is:

$$\phi = \begin{pmatrix} 0 & 1 & * & * & * & 1 & 0 \end{pmatrix}.$$

Updating *loss*, *gain* and *extra*

Once a swap has been executed, the auxiliary data structures have to be updated. Recomputing everything from scratch is an inefficient option, because many values do not change from one iteration to the next. Only changing the affected values and reusing the rest of the computations saves a lot of time. There are four cases which cause a value to change in one of the data structures, which we

1. A population point was first covered by two facilities, in the new solution by one.
2. A population point was first covered by one facility, in the new solution by two.
3. A population point was first covered by one facility, in the new solution by none.
4. A population point is uniquely covered by the entered facility.

Case 1. We collect all the points that belong to the first case in a vector named *cov_21*. The decrease in coverage can only be caused by the leaving facility. Therefore, to find the population

points that belong to *cov_21*, we first restrict to the points that were covered by the leaving facility: $row[f_l^*]$. Among these, we select the ones that are covered by only one facility in the new solution, thus: $\{i \in row[f_l^*] : cov(i) = 1\}$. Finally, of these points we discard the ones that are covered by both the leaving and entering facility. We collect the population points that are covered by both facilities in *overlap*: $overlap = \{row[f_e^*]\} \cap \{row[f_l^*]\}$. If a point is in *overlap* and has single coverage in the new solution, it was uniquely covered by the leaving facility in the old solution and is uniquely covered by the entering facility in the new solution. Thus its coverage did not decrease from two to one, so it should not be in *cov_21*. Summarizing:

$$cov\_21 = \{i \in row[f_l^*] : [cov(i) = 1] \wedge [i \notin overlap]\}.$$

Consider a population point $i \in cov\_21$ for which the coverage decreases from 2 to 1 by the executed swap. The *loss* value of the facility that still covers i will increase with $v(i)$, because it is the only remaining facility covering point i. As i is now uniquely covered, $\phi(i)$ gives the index of this covering facility in F^*, which allows us to define $f_a = F^*(\phi(i))$ as the affected facility. $\phi(i)$ is also the index of the value that needs to be adjusted in *loss*, as this is the loss value of f_a. Furthermore, $\phi(i)$ also gives the index of the affected row in the *extra* matrix, because the interaction term of f_a with all the facilities in $col[i]$ increases by $v(i)$. To understand this, recall that all facilities which can cover point i are found in $col[i]$. If any of these facilities would be swapped in the next iteration with f_a, the entering facility would take overtake the unique covering position of i from f_a, thus $v(i)$ is exactly the interaction term.

Case 2. We name the collection of points for which the coverage increases from one to two, *cov_12*. In this case, the points have to be covered by the entering facility, and their updated coverage value has to be equal to two. Following a similar reasoning as in the previous case, the points in *overlap* are excluded because these were already covered by two facilities in the old solution, not one. We thus obtain the following equation:

$$cov\_12 = \{i \in row[f_e^*] : [cov(i) = 2] \wedge [i \notin overlap]\}.$$

Again, the *loss* vector and *extra* matrix are affected. In order to know which row we have to adjust, we need ϕ of the old solution. Therefore, we save $\phi_{old} \leftarrow \phi$ at every step, before we update ϕ to coincide with the new solution. For a point i in *cov_12*, $\phi_{old}(i)$ indicates the index of the facility that was the unique covering facility of i in the old solution, and we define $f_a = F^*(\phi_{old})$. In the old solution $v(i)$ contributed to the loss value of f_a and to the interaction term of f_a with all other facilities that can cover i. In the new solution this is no longer the case, because f_a as well as the entering facility f_e^* can cover i. To process this, we subtract $v(i)$ from $loss(\phi_{old}(i))$ and from all elements in $extra(\phi_{old}, col[i])$ for every i in *col_12*. There is no need to alter the *loss* value or *extra* row corresponding of f_e*, because these will be filled correctly as the final step of the updating process.

Case 3. In this case population points are uniquely covered in the old solution and are not covered at all in the new solution. Clearly, these points have to be in $row[f_l^*]$ and will not occur in *overlap*. Therefore, all points in $row[f_l^*]$ for which the new coverage is zero belong to this group:

$$cov\_10 = \{i \in row[f_l^*] : [cov(i) = 0]\}$$

The *loss* value and *extra* row belonging to f_l^* will be overwritten by the loss value of f_e^*, so we leave them untouched for now. The *gain* vector is affected however, because in the old solution it was worthless if a candidate facility could cover a population point $i \in cov\_10$, because it was already covered. In the new solution this is no longer the case, so the *gain* value of all facilities that can cover point $i \in cov\_10$, listed in $col[i]$, increases by $v(i)$.

Case 4. Lastly, we have the points that are uniquely covered by the entered facility in the new solution. The definition of this group differs from the other ones, because this group is used to fill the rows of *extra* and *gain* corresponding to facility f_e^* from scratch. The definition of *cov_x1* in line 53 is equal to the definition of $\mathcal{OUT}$ in line 7, which implies we can fill the *extra* matrix in the same way as well. Note that we do not use *cov_x1* to fill the *loss* value of f_e^*, because it is simply equal to its former gain value plus the interaction term which belonged to the executed swap. We save this value in *aux* as a first step, so we do not need to recompute it. Finally, the *gain* vector can also be influenced by population points in *cov_x1*, but only if their coverage in the old solution was zero. To extract these points, we check whether a population point is in *overlap*. If not, the coverage must

have increased from 0 to 1, thus the gain value of other facilities that can cover that point decreases.

Algorithm 4: Best Improvement Local Search Function

Function `Best_improvement_LS`$(F^*, cov, obj, \mathcal{M}, \mathcal{N}, v, row, col, p)$
  Initialize the vector of candidate facilities $C \leftarrow \mathcal{M} \backslash \{F^*\}$
  Initialize $loss$, $gain$ and ϕ as zero vectors of length p, $|\mathcal{M}|$, $|\mathcal{N}|$ respectively
  Initialize $extra$ as zero matrix of size $p \times |\mathcal{M}|$
  Initialize F^*_{idx} as an index vector of length p with values $0, \ldots, p-1$
  for $f_{idx} \in F^*_{idx}$ and $f \in F^*$ **do** // $O(p)$
    $\mathcal{OUT} \leftarrow \{i \in row[f] : cov(i) = 1\}$ // $O(k)$
    $loss(f_{idx}) \leftarrow \sum_{i \in \mathcal{OUT}} v(i)$ // $O(k)$
    $\phi(i) \leftarrow f_{idx}\ \forall i \in \mathcal{OUT}$ // $O(k)$
    for $i \in \mathcal{OUT}$ **do** // $O(k)$
      $extra(f_{idx}, col[i]) \leftarrow extra(f_{idx}, col[i]) + v(i)$ // $O(l)$
    end
  end
  for $f \in C$ **do** // $O(m)$
    $\mathcal{IN} \leftarrow \{i \in row[f] : cov(i) = 0\}$ // $O(k)$
    $gain(f) \leftarrow \sum_{i \in \mathcal{IN}} v(i)$ // $O(k)$
  end
  $improv \leftarrow$ True

```
19
20  While improv = True Execute vectorized profit function to obtain profit matrix: // O(pm)
21    profit(f_l, f_e) ← gain(f_e) − loss(f_l) + extra(f_l, f_e) ∀f_l ∈ F*_idx, ∀f_e ∈ C
22  Let f*_{l,idx}, f*_{e,idx} be the coordinates of the highest value in profit // O(pm)
23  Let f*_l ← F*(f*_{l,idx}) and f*_e ← C(f*_{e,idx}) be the corresponding facilities
24  if profit(f*_{l,idx}, f*_{e,idx}) > 0 then
25      aux ← gain(f*_e) + extra(f*_{l,idx}, f*_e)
26      Replace leaving by entering facility in current solution: F*(f*_{l,idx}) ← f*_e
27      Replace entering by leaving facility in candidate vector: C(f*_{e,idx}) ← f*_l
28      Update obj: obj ← obj + profit(f*_{l,idx}, f*_{e,idx})
29      Increase coverage cov: cov(i) ← cov(i) + 1  ∀i ∈ row[f*_e]          // O(k)
30      Decrease coverage cov: cov(i) ← cov(i) − 1  ∀i ∈ row[f*_l]          // O(k)
31      φ_old ← φ                                                          // O(n)
32      Reset φ to a vector of zeroes                                      // O(n)
33      for f_idx ∈ F*_idx and f ∈ F* do                                   // O(p)
34          OUT ← {i ∈ row[f] : cov(i) = 1}                                //   O(k)
35          φ(i) ← f_idx ∀i ∈ OUT                                          //   O(k)
36      end
37      overlap ← {row[f*_e]} ∩ {row[f*_l]}                               // O(k)
38      cov_21 ← {i ∈ row[f*_l] : [cov(i) = 1] ∧ [i ∉ overlap]}            // O(k)
39      for i ∈ cov_21 do                                                  // O(k)
40          loss(φ(i)) ← loss(φ(i)) + v(i)
41          extra(φ(i), col[i]) ← extra(φ(i), col[i]) + v(i)               //   O(l)
42      end
43      cov_12 = {i ∈ row[f*_e] : [cov(i) = 2] ∧ [i ∉ overlap]}            // O(k)
44      for i ∈ cov_12 do                                                  // O(k)
45          loss(φ_old(i)) ← loss(φ_old(i)) − v(i)
46          extra(φ_old(i), col[i]) ← extra(φ_old(i), col[i]) − v(i)       //   O(l)
47      end
48      cov_10 ← {i ∈ row[f*_l] : [cov(i) = 0]}                            // O(k)
49      for i ∈ cov_10 do                                                  // O(k)
50          gain(col[i]) ← gain(col[i]) + v(i)                             //   O(l)
51      end
52      Reset extra row of leaving facility extra(f*_{l,idx}, :) ← 0       // O(m)
53      cov_x1 ← {i ∈ row[f*_e] : [cov(i) = 1]}                            // O(k)
54      for i ∈ cov_x1 do                                                  // O(k)
55          extra(f*_{l,idx}, col[i]) ← extra(f*_{l,idx}, col[i]) + v(i)   //   O(l)
56          if i ∉ overlap then
57              gain(col[i]) ← gain(col[i]) − v(i)                         //   O(l)
58          end
59      end
60      loss(f*_{l,idx}) ← aux
61  else
62      improv ← False
63  end
64  return F*, cov, obj
65 end
```

In Algorithm 4, the time complexity of each line is given, so we can compare the worst-case complexity of the simple and advanced approach. With the simple approach, one local search iteration has complexity $O(pmk)$. The advanced approach on the other hand, has an initialisation cost of $O(plk + mk)$, which are relatively small with respect to $O(pmk)$, since $l \ll m$. Each iteration in the advanced approach takes $O(p(m+k)+lk+n)$ which is much lower than $O(pmk)$. Summarizing, the worst-case complexity shows that the advanced approach can execute each local search iteration more efficiently than the simple approach. This is in line with our numerical experiments.

A.5 Path-Relinking

There are only two important differences in the implementation between path-relinking and best improvement local search:

1. Stopping criterion: In path-relinking, we always accept the best swap, even if it results in a negative profit. The process continues until the initial solution is identical to the guiding solution. Local search on the other hand, stops as soon as the best swap generates a profit smaller or equal to 0, as this means we have arrived at a local optimum.

2. Candidate moves: In the best improvement local search, we can swap any active facility with any inactive facility. In path-relinking however, many moves are not allowed. A facility is only allowed to leave the initial solution if it does not occur in the guiding solution, and vice versa, the only candidates that are allowed to enter, are the facilities in the guiding solution. When the best of these candidate moves is executed, the initial solution moves one step towards the guiding solution, which implies even fewer possible swaps are left to consider.

We will indicate which lines of the pseudo-code of Algorithm 4 for the best improvement local search need to be adjusted such that it can handle the differences between local search and path-relinking. See Algorithm 6 in the appendix for the pseudo-code of the resulting path-relinking algorithm.

We start by initializing the candidate vector C with all facilities that are in the guiding solution, but not in the initial solution. We also introduce the vector with leaving facilities L, which are in the initial but not in the guiding solution. Lines 2 up to 18 of Algorithm 4 can be executed without changing anything, until we arrive at the stopping criterion in line 20. The path-relinking is finished when the initial solution is identical to the guiding solution, so the stopping criterion consists of a test whether there are any elements left in L. Next, we calculate the profit matrix with a row for every $f_l \in L$ and a column for every $f_e \in C$, and determine which facilities correspond to the highest value in the profit matrix. The if-statement of line 24 is removed, because the best swap must always be executed. The swapped facilities cannot be used in future steps, so they are removed from L and C, which requires a small change in lines 26 and 27. Finally, we need to save the best solution found on the path connecting the initial and guiding solution, because this is not necessarily the last solution. After computing a new objective, we check whether it is higher than the best objective value found so far, and if so, we save the current objective, solution and coverage vector. At the end of the algorithm, the properties of the best found solution are returned.

We now discuss some more adjustments that we can make to improve this implementation. Note that a large part of the *extra* matrix is never used, because we only consider the rows and columns that correspond to facilities in L and C respectively. Constructing and adjusting the *extra* matrix takes a large portion of the total running time, so we can save time if we know which parts of *extra* are unnecessary. As we have seen in the previous section, *extra* and *loss* are filled row by row. We can check whether a row is relevant in a relatively quick way, so we have implemented this check in the algorithm. The columns of *extra* and the rows of *gain* are filled using the *col* dictionary. In order to know which of the facilities returned by the *col* dictionary are relevant, we would need to take the intersection with the candidate vector C. We found that taking this intersection is more expensive than simply filling all the columns indicated by *col*. Therefore, we adjusted the code such that only the relevant rows of *loss* and *extra* are constructed and updated, but this distinction is not made in the columns.

In line 6, we replace F^*_{idx} and F^* by L_{idx} and L, such that only the rows that correspond to facilities in L are constructed. We split the solution F^* in a constant part F^{con} and a variable part F^{var}. The variable part is initialized with all the elements of L, and in each path-relinking step, the entering facility replaces the leaving facility in F^{var}. If an improvement on the incumbent solution is found, we save the current composition of F_{var} and the corresponding objective value and coverage vector. Splitting the solution avoids indexing problems due to the altered dimensions of *extra* and *loss*. At the end of the algorithm, the variable part corresponding to the best objective value is concatenated to the constant, to obtain the full solution again.

In addition to restricting the construction, we also restrict the updating process. Updating *loss* and *extra* is done one row at a time, so per facility. Clearly, it is only necessary to update the rows which belong to the facilities in L, the ones that still have to leave the solution. We use the vector ϕ to make this distinction, this vector will indicate for every population point in *cov_21* and *cov_12* whether we need to process its associated change. Instead of initializing ϕ as a zero vector, we set every entry equal to -1. Next, for every facility f in L, we find the population points it uniquely covers, and fill their entries is ϕ with the index of f in F^{var}. In algorithm 4, $\phi(i)$ and $\phi_{old}(i)$ indicate which row of *extra* and *loss* need to be updated for every $i \in$ *cov_21* and $i \in$ *cov_12* respectively. Now, $\phi(i)$ fulfills the same function, but it is also possible that a -1 is returned. In that case, i is uniquely covered by a facility in the constant part of the solution, or by one of the facilities that have entered the solution from C. Either way, that facility is not allowed to leave the solution, so we do not need to update the corresponding rows in *extra* and *loss*. Checking whether the value returned by ϕ is unequal to -1, suffices make sure we only update the relevant rows. Lastly, we delete lines 52, 55 and 60, in which the values corresponding to the leaving facility are replaced by the values of the entering facility. As the entered facility is not allowed to leave the solution in future steps, these lines are not necessary anymore.

A.6 The full algorithm

The full algorithm is given in Algorithm 5.

Algorithm 5: The full GRASP algorithm

Set p, S and *grid-size* to the desired values
Set a maximum running time t_{max}, and measure starting time *start*
Choose a local search strategy *best* or $first$, and if $first$ is chosen, set *border*
Choose a path-relinking strategy $forward$, *backward*, or *both*
Load the population data: $\mathcal{N}$, v, coordinates of $\mathcal{N}$
Load the potential facility data: $\mathcal{M}$, coordinates of $\mathcal{M}$
Load the pre-processed dictionaries *row* and *col*
Adapt `Greedy` as described in 24 to add randomness
$F^*, cov, obj \leftarrow$ `Greedy(`$\mathcal{M}, \mathcal{N}, v, row, col, p$`)`
if *best* **then**
    $F^*, cov, obj \leftarrow$ `Best_improvement_LS(`$F^*, cov, obj, \mathcal{M}, \mathcal{N}, v, row, col, p$`)`
else
    $F^*, cov, obj \leftarrow$ `Local_Search(`$F^*, cov, obj, \mathcal{M}, v, row, border$`)`
end
If F^* satisfies conditions section 4.3 of main article store F^*, *cov* and *obj* in the elite pool
while $time - start < t_{max}$ **do**
    $F^*, cov, obj \leftarrow$ `Greedy(`$\mathcal{M}, \mathcal{N}, v, row, col, p$`)`
    if *best* **then**
        $F^*, cov, obj \leftarrow$ `Best_improvement_LS(`$F^*, cov, obj, \mathcal{M}, \mathcal{N}, v, row, col, p$`)`
    else
        $F^*, cov, obj \leftarrow$ `Local_Search(`$F^*, cov, obj, \mathcal{M}, v, row, border$`)`
    end
    Randomly select an elite solution from the pool of elite solutions
    $forward$: set elite as guiding solution, *backward*: set elite as initial solution, *both*: perform $forward$ and *backward*, then pick the best outcome
    $F^*, cov, obj \leftarrow$ `PathRelink(`$F^*_{init}, cov_{init}, obj_{init}, F^*_{guide}, \mathcal{M}, \mathcal{N}, v, row, col$`)`
    If F^* satisfies conditions section 4.3 of main article: store F^*, *cov* and *obj* in elite pool
end

Advanced Path-relinking Algorithm

Algorithm 6: Path-Relinking Function

Function `Path_Relink`$(F^*, cov, obj, F^*_{guide}, \mathcal{M}, \mathcal{N}, v, row, col)$
- Initialize the vector of candidate facilities $C \leftarrow \{F^*_{guide}\} \backslash \{F^*\}$
- Initialize the vector of leaving facilities $L \leftarrow \{F^*\} \backslash \{F^*_{guide}\}$
- Split F^* in a constant and variable part: $F^{con} \leftarrow \{F^*\} \cap \{F^*_{guide}\}$, $F^{var} \leftarrow L$
- Set p equal to the length of L
- Set $obj_{best} \leftarrow obj$, $cov_{best} \leftarrow cov$, $F^{var}_{best} \leftarrow F^{var}$
- Initialize $loss$ and $gain$ as zero vectors of length p and $|\mathcal{M}|$ respectively
- Initialize ϕ as a vector with all entries equal to -1 and length $|\mathcal{N}|$
- Initialize $extra$ as zero matrix of size $p \times |\mathcal{M}|$
- Initialize L_{idx} as an index vector of length p with values $0, \ldots, p-1$
- **for** $f_{idx} \in L_{idx}$ and $f \in L$ **do**
  - $\mathcal{OUT} \leftarrow \{i \in row[f] : cov(i) = 1\}$
  - $loss(f_{idx}) \leftarrow \sum_{i \in \mathcal{OUT}} v(i)$
  - $\phi(i) \leftarrow f_{idx}\ \forall i \in \mathcal{OUT}$
  - **for** $i \in \mathcal{OUT}$ **do**
    - $extra\Big(f_{idx}, col[i]\Big) \leftarrow extra\Big(f_{idx}, col[i]\Big) + v(i)$
  - **end**
- **end**
- **for** $f \in C$ **do**
  - $\mathcal{IN} \leftarrow \{i \in row[f] : cov(i) = 0\}$
  - $gain(f) \leftarrow \sum_{i \in \mathcal{IN}} v(i)$
- **end**
- **while** $L \neq \emptyset$ **do**
  - Execute vectorized profit function to obtain profit matrix: $profit(f_l, f_e) \leftarrow gain(f_e) - loss(f_l) + extra(f_l, f_e)\ \forall f_l \in L_{idx}, \forall f_e \in C$
  - Let $f^*_{l,idx}$, $f^*_{e,idx}$ be the coordinates of the highest value in $profit$
  - Let $f^*_l \leftarrow L(f^*_{l,idx})$ and $f^*_e \leftarrow C(f^*_{e,idx})$ be the corresponding facilities
  - Delete entered facility from candidate list: $C \leftarrow \{C\} \backslash \{f^*_e\}$
  - Similarly, update L and L_{idx}: $L \leftarrow \{L\} \backslash \{f^*_l\}$, $L_{idx} \leftarrow \{L_{idx}\} \backslash \{L_{idx}(f^*_{l,idx})\}$ Replace f^*_l by f^*_e in F^{var}
  - Update obj: $obj \leftarrow obj + profit(f^*_{l,idx}, f^*_{e,idx})$
  - Increase coverage cov: $cov(i) \leftarrow cov(i) + 1\ \ \forall i \in row[f^*_e]$
  - Decrease coverage cov: $cov(i) \leftarrow cov(i) - 1\ \ \forall i \in row[f^*_l]$
  - **if** $obj > obj_{best}$ **then**
    - Set $obj_{best} \leftarrow obj$, $cov_{best} \leftarrow cov$, $F^{var}_{best} \leftarrow F^{var}$
  - **end**
  - $\phi_{old} \leftarrow \phi$
  - Reset ϕ to a vector with all entries equal to -1
  - **for** $f_{idx} \in L_{idx}$ and $f \in L$ **do**
    - $\mathcal{OUT} \leftarrow \{i \in row[f] : cov(i) = 1\}$
    - $\phi(i) \leftarrow f_{idx}\ \forall i \in \mathcal{OUT}$
  - **end**
  - $overlap \leftarrow \{row[f^*_e]\} \cap \{row[f^*_l]\}$

```
42
43  |
44  |   |  cov_21 ← {i ∈ row[f_l^*] : [cov(i) = 1] ∧ [i ∉ overlap]}
45  |   |  for i ∈ cov_21 do
46  |   |  |  if φ(i) ≠ −1 then
47  |   |  |  |  loss(φ(i)) ← loss(φ(i)) + v(i)
48  |   |  |  |  extra(φ(i), col[i]) ← extra(φ(i), col[i]) + v(i)
49  |   |  |  end
50  |   |  end
51  |   |  cov_12 = {i ∈ row[f_e^*] : [cov(i) = 2] ∧ [i ∉ overlap]}
52  |   |  for i ∈ cov_12 do
53  |   |  |  if φ_old(i) ≠ −1 then
54  |   |  |  |  loss(φ_old(i)) ← loss(φ_old(i)) − v(i)
55  |   |  |  |  extra(φ_old(i), col[i]) ← extra(φ_old(i), col[i]) − v(i)
56  |   |  |  end
57  |   |  end
58  |   |  cov_10 ← {i ∈ row[f_l^*] : [cov(i) = 0]}
59  |   |  for i ∈ cov_10 do
60  |   |  |  gain(col[i]) ← gain(col[i]) + v(i)
61  |   |  end
62  |   |  cov_01 ← {i ∈ row[f_e^*] : [cov(i) = 1] ∧ [i ∉ overlap]}
63  |   |  for i ∈ cov_01 do
64  |   |  |  gain(col[i]) ← gain(col[i]) − v(i)
65  |   |  end
66  |   end
67  |   F^* = F^{con} ∪ F^{var}_{best}
68  |   return F^*, cov_{best}, obj_{best}
69  end
```