Metaheuristic Optimization

Sagdullin Damir
ING3 IA2
CY Tech
Cergy, France
sagdullind@cy-tech.fr

Laiolo Léo
ING3 IA2
CY Tech
Cergy, France
laiololeo@cy-tech.fr

Mensah Yann
ING3 IA2
CY Tech
Cergy, France
mensahyann@cy-tech.fr

***Abstract*—Nowadays, we are trying to optimize our device usage. Indeed, we are improving our computers, antennas, servers, etc, capabilities but the demand keeps growing. Researches have shown that edge computing was a good way to do horizontal scaling. This method uses a lot of connected devices which can be very expensive. In this paper, we focus on optimizing user coverage while reducing the cost of devices.**

***Index Terms*—Edge, site deployment, meta-heuristic, optimization**

I. Introduction

Cloud computing is great but not the best for real time tasks in mobile networks because of high latency. Popular solution is edge computing : computing and storage are closer to the end user. There are multiple challenges : caching, offloading, server placement etc. We will concentrate on the question of what out of available sites should be used as edge sites : edge site deployment problem.

II. Problem statement [1]

We consider a certain area A.

There are P candidate places for edge site placement in A.

Each placement p_i , $1 \leq i \leq P$, has coordinates $(\theta_i^P, \varphi_i^P)$.

Each deployed edge site p_i has a monetary cost m_i .

Once edge site p_i is deployed, it covers users in S radius around it.

The decision variable x_i , $1 \leq i \leq P$ is binary. $x_i = 1$ means p_i is selected and 0 otherwise.

Total monetary cost is

$$M = \sum_{i=1}^{P} m_i x_i$$

.

There are U users in A.

Each user u_j , $1 \leq j \leq U$, has coordinates $(\theta_j^U, \varphi_j^U)$.

We will need distance $d_{i,j}$ between p_i and u_j. Since we're going to use Cartesian coordinate system,

$$d_{i,j} = \sqrt{(\theta_i^P - \theta_i^U)^2 + (\phi_i^P - \phi_j^U)^2}$$

.

Coverage $c_{i,j}$ between p_i and u_j is binary. $c_{i,j} = 1$ means u_j is covered by p_i and 0 otherwise.

So

$$c_{i,j} = \begin{cases} 1, & \text{if } d_{i,j} \leq S \\ 0, & \text{if } d_{i,j} > S \end{cases}$$

The set of users covered by p_i is $C_i = \{u_j | c_{i,j} = 1\}$.

And the set of all users covered by selected placements is

$$C = \bigcup_{x_i=1} C_i$$

.

Overall user coverage $Q = \frac{|C|}{U} \times 100\%$.

Finally our objectives are :

- maximize Q
- minimize M

III. Dataset [2]

This section provides an overview of the datasets used in our study, detailing their structure and the initial preprocessing steps applied.

A. User Dataset (`users-melbmetro-generated.csv`)

Total Records: 131,312 users in Melbourne metropolis
Columns:

- `Latitude`: The latitude of the user in degrees.
- `Longitude`: The longitude of the user in degrees.

B. Antenna Dataset (`site.csv`)

Total Records: 95,562 sites across Australia
Columns:

- `SITE_ID`: Unique identifier for the antenna.
- `LATITUDE`: The latitude of the antenna in degrees.
- `LONGITUDE`: The longitude of the antenna in degrees.
- `NAME`: Name of the site where the antenna is located.
- `STATE`: State or territory where the antenna is located.
- `LICENSING_AREA_ID`: Identifier for the licensing area.
- `POSTCODE`: Postal code for the site.
- `SITE_PRECISION`: Precision of the site's location.
- `ELEVATION`: Elevation of the site (not specified in examples).
- `HCIS_L2`: Classification code for the site.

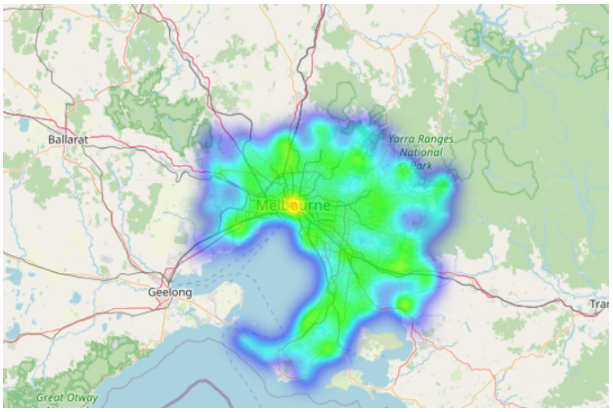

Fig. 1: Preprocessed Sites Heatmap

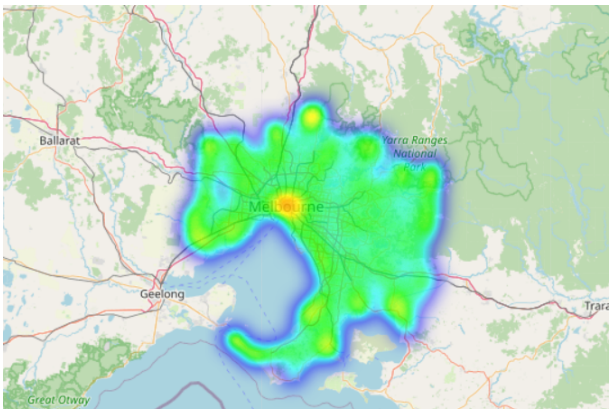

Fig. 2: Preprocessed Users Heatmap

C. Data Preprocessing

Upon loading the datasets, several preprocessing steps were applied to prepare the data for further analysis:

1) **Loading Data:** The data was loaded from `users-melbmetro-generated.csv` and `site.csv` into pandas DataFrames, referred to as `df_users` and `df_servers` respectively. In our dataset, antennas and users are located with spherical coordinates.
2) **Data Cleaning and Preparation:**
   - For `df_users`: Columns were renamed from `Latitude` and `Longitude` to `lat` and `lon` for consistency and clarity.
   - For `df_servers`: Unnecessary columns (`SITE_ID`, `NAME`, `STATE`, `LICENSING_AREA_ID`, `POSTCODE`, `SITE_PRECISION`, `ELEVATION`, `HCIS_L2`) were removed, keeping only geographical location information (`LATITUDE` and `LONGITUDE`). Remaining columns were renamed to `lat` and `lon`.
3) **Coordinate Conversion:** Geographic coordinates (latitude and longitude) from both datasets were converted into Cartesian coordinates (x, y) using the Universal Transverse Mercator (UTM) projection. This step facilitates spatial distance calculations and other spatial analyses. The `pyproj` library was utilized to define a projection and apply this transformation to the latitude and longitude columns, resulting in new `x` and `y` columns for both DataFrames.
4) **Filter out unnecessary site :** Since our users are located in Melbourne metropolis area but sites are across Australia. To filter out unnecessary sites that are further than 1000m (S than we used) from any of users, we used kd-trees to represent users and servers. Kd-trees cut the space (2d since we work with Cartesian coordinates) to have a tree representation which is way better adapted for distance calculations. Thanks to it we passed from 95,562 to 9,229 servers. Note that we also used kd-tree data format for coverage calculations as well.

IV. Meta-heuristic Optimization Methodology

For this multi-objective problem, we decided to use the Pareto dominance to compare solutions and keep the best. The two implemented algorithms were slightly modified to generate Pareto front solutions.

A. Simulated annealing

Simulated Annealing (SA) is a well known algorithm for optimization. It prioritises finding of a local optimum in a given time rather than searching a global optimum for an unpredictable amount of time. This is why it is often use for NP-hard problems, as the *Travelling salesman problem*.

Algorithm 1 Simulated annealing (original)

$s \leftarrow s_0$
$T \leftarrow 1000$
$\alpha \leftarrow 0.9$
$i_{max} \leftarrow 1000$
for i through i_{max} **do**
  $s_{new} \leftarrow neighbour(s)$
  $r \leftarrow random(0, 1)$
  if $f(s_{new}) > f(s)$ **then**
    $s \leftarrow s_{new}$
  else if $r < e^{\frac{f(s)-f(s_{new})}{T}}$ **then**
    $s \leftarrow s_{new}$
  end if
  $T \leftarrow \alpha \times T$
end for

In the original pseudo-code 1, s is a solution, T corresponds to the temperature, α is the variation of temperature at each step i, i_{max} corresponds to the number of iterations, r is a random float picked between 0 and 1, and f corresponds to the evaluating function.

Originally, simulated annealing is used in problems where the solutions are defined in one dimension (example: $s \in \mathbb{R}$). To use the algorithm, we had to adapt it for solutions in P dimensions ($x \in \{0,1\}^P$). As we are looking for several

Algorithm 2 Simulated annealing (Pareto adapted)

Require: $x \in \{0,1\}^P$
  $x \leftarrow x_0$
  $T \leftarrow 1000$
  $\alpha \leftarrow 0.9$
  $i_{max} \leftarrow 1000$
  $S \leftarrow [x]$
  for i through i_{max} **do**
    $x_{new} \leftarrow neighbour(x)$
    Keep in S only solutions that dominates x_{new}
    if no solution from S dominates x_{new} **then**
      $S.insert(x_{new})$
    end if
    $r \leftarrow random(0,1)$
    if $Q_{x_{new}} > Q_x$ and $M_{x_{new}} < M_x$ **then**
      $x \leftarrow x_{new}$
    else if $r < e^{\frac{(Q_{x_{new}}+M_{x_{new}})-(Q_x+M_x)}{T}}$ **then**
      $x \leftarrow x_{new}$
    end if
    $T \leftarrow \alpha \times T$
  end for

solutions, we had to create an archive S to store them and keep only those that dominates the others.

In the Pareto adapted pseudo-code 2, x is a solution, T corresponds to the temperature, α is the variation of temperature at each step i, i_{max} corresponds to the number of iterations, S is the set of Pareto solutions, r is a random float picked between 0 and 1, M_x corresponds to the cost of the x solution and Q_x is the coverage of the x solution.

B. Multi-Objective Firefly Algorithm

The Multi-Objective Firefly Algorithm (MOFA) algorithm is also a well known meta-heuristic optimization algorithm. It is based on the animals behaviour, especialy fireflies, that moves as one group. This algorithm simulates swarms of candidate solutions that will move in a search space, looking for an optimum.

Here, we implemented our own simplified MOFA. Fireflies are moving toward those with the best solutions. To explore other solutions, we added a mutation factor (generate with random values) to fireflies movements.

In the simplified MOFA pseudo-code 3, P is a set of fireflies p, S corresponds to a set of Pareto solutions, d is the direction of the firefly, α corresponds to the attraction coefficient, β is the randomness coefficient, m corresponds to the mutation and γ is the mutation coefficient.

V. Implementation

The optimization algorithm were implemented in python on Jupyter Notebook, on Google Colab. All the code is accessible at this link: Google Colab . The dataset was uploaded on a Google Drive as CSV files. There is no need for GPU to run the code.

Algorithm 3 Simplified Multi-Objective Firefly Algorithm

  Initialize fireflies P
  $S \leftarrow [p_0]$
  for $p_i \in P$ **do**
    for $p_j \in P$ **do**
      if p_i dominates p_j **then**
        $d \leftarrow \alpha \times (p_j - p_i) + \beta \times random(0, 0.5)$
        $m \leftarrow \gamma \times random(0, 0.5)$
        $p_j \leftarrow p_j + d + m \times random(0, 1)$
      end if
    end for
    $p_i \leftarrow round\_to\_array\_of\_ints(p_i)$
    Keep in S only solutions that dominates p_i
    if no solution from S dominates p_i **then**
      $S.insert(p_i)$
    end if
  end for

TABLE I: Algorithms parameters

Algorithm	Parameter	Role	Value
SA	initializor	Initialize the first solution	$[1, ..., 1]$
	T	Initial temperature T	1000
	alpha	Temperature variation α	0.9
	iterations	Number of iterations i_{max}	100
	flip_bits	Number of value that can mutate in a solution	100
MOFA	max_generations	Number of swarm generation	1
	population_size	Number of fireflies in the swarm	20
	attraction	Attraction coefficient α	2
	randomness	Randomness coefficient β	1
	mutation	Mutation coefficient γ	1

VI. Performance comparison

In this section, the performance of Simulated annealing and MOFA will be compared. The studied solutions were generated by the algorithms with the parameters listed in this table I.

Solutions found by the two algorithms are listed in this table II.

A Pareto front of the proposed solutions was drawn in the graphic 3.

We believe that everyone has its own vision of cost and

TABLE II: Solutions

Algorithm	Cost range	Coverage range	Number of solutions
SA	$[9000, 19000]$	$[82, 89]\%$	26
MOFA	$[6500, 9500]$	$[80, 84]\%$	8

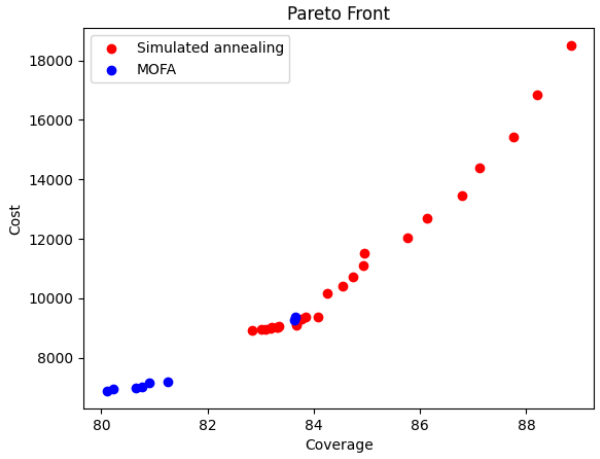

Fig. 3: Pareto front

TABLE III: Comparison metrics

Algorithm	Efficiency	Contribution	Number of dominating solutions
SA / MOFA	21	76%	5
MOFA / SA	6	24%	2

objective. To let the choice to edge computing investors, we compared the optimization algorithms using Pareto metrics. Those are described below.

A. *Pareto efficiency*

Pareto efficiency is the number of solutions from a set S_1 that are not dominated any solutions from S_2.

B. *Pareto contribution*

Pareto contribution is described by the following formula, where C corresponds to the set of solutions that are in both sets of solutions, $W1$ is the set of solutions from S_1 dominating solutions from S_2, $W2$ is the set of solutions from S_2 dominating solutions from S_1, $N1$ corresponds to the set of solutions from S_1 that are neither dominated by nor dominating solutions from S_2 and $N2$ corresponds to the set of solutions from S_2 that are neither dominated by nor dominating solutions from S_1.

$$CONT(S_1, S_2) = \frac{\frac{||C||}{2} + ||W1|| + ||N1||}{||C|| + ||W1|| + ||W2|| + ||N1|| + ||N2||}$$

C. *Analysis*

Based on those metrics III, we observed that most of the time Simulated Annealing was better than MOFA. Indeed, the efficiency of SA on MOFA is greater than MOFA on SA. Moreover, the contribution of SA on MOFA is also greater than MOFA on SA. The low cost solutions found by SA have greater coverage than the most expansive MOFA solutions. Those metrics show that SA is more interesting than MOFA to resolve our problem.

Yet, MOFA isn't useless. This algorithm find solutions that have low cost, compared to SA solutions, and still keep a coverage greater than 80%. Those low cost solutions were found because MOFA can much easily explore than SA because of its candidate population.

SA has a computing time much shorter than MOFA. This is due to the firefly swarm simulation. With an optimized MOFA, we could have found better solutions with bigger firefly population and several swarm generations.

Our different trials showed that SA seems more interesting than MOFA to solve our optimization problem. Even though, they are complementary.

VII. Conclusion

We studied the edge site deployment problem for edge computing by trying to cover a set of users with a set of antennas. The aim was to reduce the cost of antennas deployment while covering the greatest number of users. To solve this optimization problem, we used two algorithm : SA and a simplified MOFA. Our results showed that SA seemed proposing more interesting solutions than MOFA, comparing Pareto metrics. However, they both found diverse solutions so edge computing investors could chose which one suits them best.

If the choice is still too hard to make, it would be interesting to combine the exploration ability of MOFA and the exploitation of SA by writing a crossover algorithm.

References

[1] X. Xing, Y. Song, and B. Wang, "A hybrid metaheuristic algorithm for edge site deployment with user coverage maximization and cost minimization," *International Journal of Advanced Computer Science and Applications*, vol. 14, 01 2023.

[2] S. Edge, "Eua datasets: Edge server, user dataset for edge computing research)," 2021. [Online]. Available: https://github.com/swinedge/eua-dataset