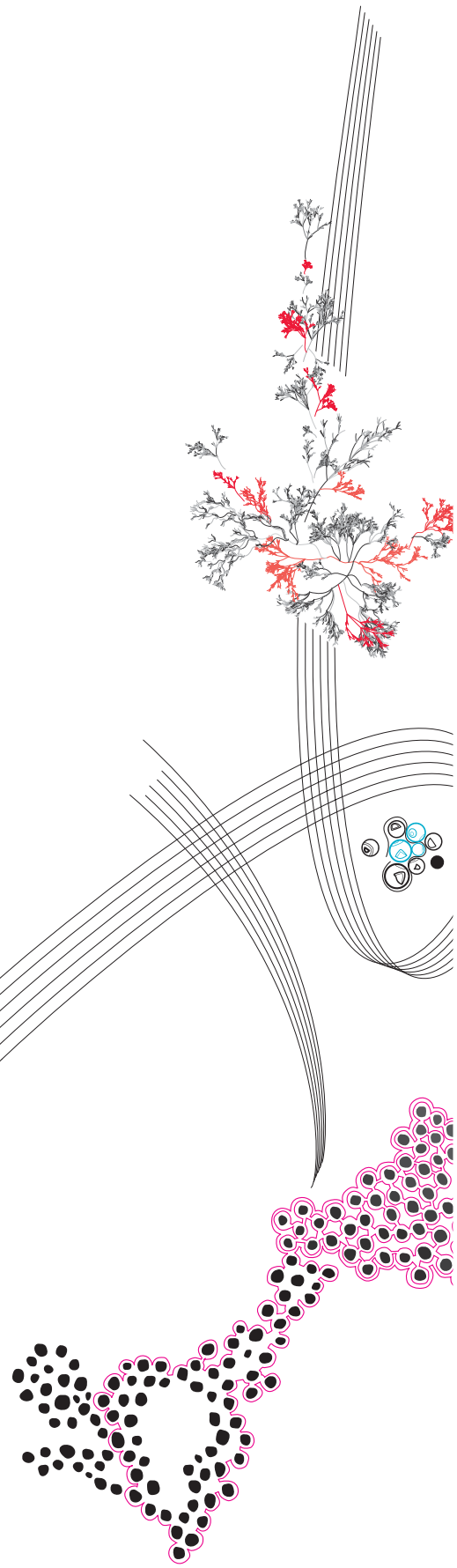


BSc Thesis Applied Mathematics

Efficiency and convergence of strategic two-sided facility location games on random graphs

Eline Peeters

Supervisor: A. Skopalik

June, 2021

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Efficiency and convergence of strategic two-sided facility location games on random graphs

E. W. H. Peeters

June 26, 2021

Abstract

Location models are used to analyse and predict customer and company behaviour with regard to their location and attraction range. We look into location models that are represented by graphs. Existing algorithms can run simulations on the graphs that lead to an equilibrium state. In an equilibrium state, no single company can change their location without ending up with lower profits. It can be beneficial for companies to compute these equilibria, to gain insight in how their decisions will influence their profit in the equilibria. By implementing these algorithms into programming software, simulations can be ran using different input graphs. We look into the effect of different graph properties on the time complexity and efficiency of the algorithm. It shows that the distribution of the improvement step follows a normal distribution, and that the efficiency of the algorithm is maximum in the vast majority of the simulations. A low number of facilities and high edge density are beneficial for the time complexity of the algorithm, as they lower the number of improvement steps needed until an equilibrium is found.

1 Introduction

When opening a new facility, companies aim for maximizing their profit by having the best possible location. However, finding that optimal location is not too obvious. The profit at a certain location depends on the locations of all other companies, and therefore companies are constantly adjusting their location strategies to the changes of other parties. As a consequence, the distribution of customers continuously changes as well, since facilities move closer and further away from them. Such customer and company behaviour can be represented and analysed using mathematical location models.

1.1 Related work

The fundamental location model is written by Hotelling [1]. A continuous line represents infinitely many customers, uniformly distributed over the full length of the line. Facilities can strategically choose a location on the line segment, after which each facility attracts a certain fraction of the customers. Facilities continuously relocate themselves to improve their profit, until an equilibrium is reached: there is no single facility in the game that could benefit from a relocation.

This one-dimensional model can be extended into two-dimensional models, where the network of customers is represented by, for example, a graph. Every customer and facility is assigned a vertex, and a connection occurs if two customers and/or facilities can reach each other. To solve this location problem, algorithms exist to find an equilibrium. The main focus of this paper is on the algorithm established by Krogmann et al. [2]. Given a graph, their algorithm computes

the locations of the facilities in an equilibrium, and the amount of customers they attract in this equilibrium. Node weights are taken into account to represent the spending capacity of the cusomters.

1.2 My contribution

Before reaching an equilibrium, the algorithm goes through several stages to improve the profit of the facilities. The number of stages that is being visited, depends on the size of the graph. However, it is not known what exact relation they show. Another property of an equilibrium is the efficiency, which is an estimation of the ratio of the total profit compared to the optimal profit. This number can be different for every equilibrium, and therefore is dependent on the used graph type. There may also be many other graph properties that influence the efficiency and the number of visited stages. Therefore, the algorithm is applied to multiple graphs with different properties. We investigate what effects these properties have on the running time of the algorithm and efficiency of equilibria, and how the algorithm behaves when multiple properties are being varied at once. These results can be used to improve the usage of the algorithm.

2 Model and notation

The algorithm is applied to a directed graph H consisting of weighted vertices $V = \{v_1, ..., v_n\}$ and edges. Every vertex represents exactly one customer, where the vertex weight $w(v_i)$ is the spending capacity of customer v_i. Meanwhile, the vertices are also possible locations for facilities to settle. The set $F = \{f_1, ..., f_k\}$ represents all facilities in the game.
Once a graph is created, the positions of the customers are fixed and therefore they cannot reposition themselves. However, the facilities strategically choose one location vertex v each to locate themselves. They can strategically reposition to a new location to influence which equilibrium the game ends up in. These location choices of all facilities are represented by the strategy profile $s = \{s_1, ..., s_k\}$, where s_i denotes the location vertex of facility f_i. The way a customer v_i divides their spending capacity among all facilities in their neighbourhood, depends on the strategy profile s and is denoted by $\sigma(s, v_i)$. When there is one facility f_j that chooses a different location, while all other facilities stick to their original strategy, the strategy profile changes to $s = (s'_j, s_{-j})$.

Every strategy profile s leads to a certain division of spending capacity. The result is a vector of facility loads $\ell(s, \sigma)$, which are the total amounts of spending capacities the facilities receive, $\ell_k(s, \sigma) = \sum_{i=1}^{n} \sigma_k(s, v_i)$ for facility f_k. The goal is to find a strategy profile s such that there is no single facility agent f_k that would have higher facility loads when moving to another location. Such a situation is called an equilibrium, where it holds that $\ell_k(s, \sigma) \geq \ell_k((s'_j, s_{-j}), \sigma)$, $\forall\ f_j \in F$, $\forall\ s'_j \in V$. Another condition that holds at an equilibrium is that the customer distribution shows no changes anymore. Krogmann et al.[2] proved that there exists at least one equilibrium in every location game. It is also possible for a location game to have multiple equilibria.

3 Methods

We apply the algorithm to several directed graphs with different properties, of which further details are explained in Section 3.1. We look for similarities and differences between the found equilibria when applying it to the different graph types. One of the parameters we keep track of is the time complexity: the number of computations that are needed until an equilibrium is found.

The second parameter we look at is the efficiency of the computed equilibrium. In the equilibrium, there are two possible states for the client vertices. Either they are covered, or they do not. For a vertex to be covered, at least one facility should be located on either the vertex itself or on one of its neighbouring vertices. In this case, vertex v_b is a neighbour of vertex v_a if the edge (v_a, v_b) exists. All vertices that satisfy the requirement of coverage, together form the covered clients set. By defining $N_s(v_i)$ as the set of facilities in the shopping range of vertex v_i, $N_s(v_i) = \{f_j | s_j \in N(v_i)\}$, we can define the covered client set as follows: $C(s) = \{v_i | v_i \in V, N_s(v_i) \neq \emptyset\}$.
Using the previous definition, we can obtain a formula for the social welfare of the equilibrium, also known as as the weighted participation range $W(s)$. This is equal to sum of all client weights, for all clients that are covered by a facility in the equilibrium.

$$W(s) = w(C(s)) = \sum_{v_i \in C(s)} w(v_i) \tag{1}$$

The efficiency of the equilibrium, denoted by η, can then be computed by comparing its weighted participation rate to the highest possible social welfare. Let S be the set of all possible location strategies, then

$$\eta = \frac{W(s)}{max_{s_i \in S} W(s_i)}. \tag{2}$$

3.1 Graphs

The Python library features many different random graphs, of which a small selection will be used to apply the algorithm on. The chosen graphs are listed below with their properties.

Erdös Renyi
One of the most used random graph type is the *Erdös-Renyi* graph. This graph is constructed from an empty graph with n nodes. Between these nodes, edges are created with a probability $0 < p < 1$ per edge. The probability of an edge addition is completely independent of all other edges.

Newman-Watts-Strogatz
The Newman-Watts-Strogatz graph is a small-world network: any two random nodes are connected to each other with a relatively short path. The base is a ring consisting of n nodes. Each node in the ring is connected to its m nearest neighbours (or $m - 1$ neighbours if m is odd). Then shortcuts are created by adding new edges. With probability p, a new edge (u, w) is added with randomly-chosen existing node w.

Barabási-Albert model
A Barabási-Albert model graph contains clustering and shows a power law. The model starts with an initial connected network of n_0 nodes. Then, $n - n_0$ new nodes are added to the network one at a time, until the network consists of n nodes. Each new node is connected to m existing

nodes with a probability that is proportional to the number of links that the existing nodes already have. Formally, the probability p_i that a new node is connected to node i is $p_i = \frac{k_i}{\sum_j (k_j)}$, where k_i is the degree of node i, summing over all existing nodes j. Heavily linked nodes (also called hubs) tend to quickly accumulate even more links, while nodes with only a few links are unlikely to be chosen as the destination for a new link.

Power law cluster
The power law cluster graph is comparable to the Barabási-Albert model, but it enables even higher clustering. The inputs are the same, where n is the number of nodes and m is the number of edges that the extra nodes are connected with. The addition to the Barabási–Albert growth model is an extra step after creating a random edge. In this extra step, the edge is connected to one of its neighbours with a probability p, causing a triangle of edges.

Regular graph
A regular graph is a random graph with n nodes, where each node has degree d. From the set of all possible graphs, one graph is uniformly randomly chosen. There is one condition this graph should satisfy, which is $\sum_{v \in V} d(v) = 2|E|$. Therefore, the number of nodes n and the degree d cannot both be odd.

3.2 Parameters

In addition to the different graph types, the influence of other variables on the time complexity and efficiency η are investigated as well. The variables used in this research are stated below with a short explanation.

Number of nodes
The number of nodes, n, is a natural number, $n \in \mathbb{N}$. Since each node represents a client with a spending capacity, a higher number of nodes implies there are more clients in the game, and also a higher total spending capacity. These two properties influence the equilibria. In addition, each node is a possible location for the facilities. An increase in the number of locations each facility can choose from, results in an increase in possible location strategies s. Every location strategy s needs to be compared to the equilibrium strategy $s_{\text{equilibrium}}$ to validate that it is an equilibrium. Therefore, the higher the number of nodes in the graph, the more time the computation will take.

Number of facilities
The number of facilities f also increases the number of possible strategy profiles s. Therefore the strategy comparison will probably have a longer computation time as well.

Edge density
The edge density determines the total number of edges in the graph, and how this number relates to the number of nodes in the graph. Its value lies between 0 and 1, and represents the ratio between the number of edges in a graph and the maximum possible number of edges. For directed graphs, the following formula for the edge density holds: $d = \frac{|E|}{|V|(|V|-1)}$. A high edge density implies that each vertex is connected to relatively many other vertices. This translates to the clients having many different locations to visit, and the facilities being in the neighbourhood of many clients. As a result, the weights of the client vertices will be divided over more facilities.

Weight distribution
The spending capacity of each client is represented by a vertex weight $w(v)$. These vertex weights can be created according to various distributions. The first distribution is the uniform distribution $U(a, b)$, where a and b are the lower and upper bounds, respectively. The probability of the vertex weight being assigned the value x is $P(w(v) = x) = \frac{1}{b-a}$ for all values $x \in [a, b]$. In this distribution, as well as the other distributions mentioned next, the probability of a weight value $w(v)$ is independent of the values of all other weight values.
The second distribution is the Poisson distribution, where $P(w(v) = x) = \frac{\lambda^x}{k!}e^{-\lambda}$, with $\lambda = E(w(v))$.
The last distribution is the binomial distribution: $P(w(v) = x) = \binom{n}{k}p^k(1-p)^{n-k}$. Here n is again the number of nodes, and $p[0, 1]$ is the probability of k successes in n independent trials.

Initial strategy
The initial strategy profile s is the first profile to which all other profiles will be compared. In this comparison it is checked whether there is a facility f_j that could improve its facility load by moving to another location vertex. In case a graph has multiple equilibria, the initial strategy profile could influence at which equilibrium the algorithm ends, and thus what the value of the Efficiency η of the first found equilibrium will be.

4 Implementation

4.1 Algorithm

The algorithm starts with an initial strategy profile s. The facility loads $\ell_j(s, \sigma)$ are computed for all facilities $f_j \in F$, which are also known as the utilities. The utilities of the chosen strategy s are compared to the utilities of all other possible strategies (s'_j, s_{-j}) where exactly one facility f_j is located at a different vertex. If a strategy profile (s'_j, s_{-j}) is found where at least one facility f_j has a higher utility compared to the old strategy s, it means the initially chosen strategy s is not an equilibrium. The game moves to the situation in which the initial strategy profile is given by the new (s'_j, s_{-j}), that caused a utility increase for facility f_j. This is the so-called 'improvement step'. The process of comparing strategy profiles continues until a strategy profile is found in which there is no benefit for any facility to move to another location vertex.

The utilities of a given strategy profile s are computed using two short algorithms. It is known that if a certain facility agent f_j has the lowest costs in the graph, all customers in its range will choose to visit that facility. Therefore, one of the first steps of the algorithm is to obtain the set of vertices that have the smallest possible facility load, also called the Minimum Neighbourhood Set (MNS). These MNS's are found by reducing the graph to a maximum-flow problem and solving this problem. Further details of this process can be found in the original paper in which the algorithm is introduced [2].

Once the MNS is found, all neighbouring vertices of this set will equally divide their spending capacity among the facilities. The vertices of the MNS are deleted from the graph, and a new MNS is computed in the remaining graph. This process repeats until the whole network is subdivided in shared client sets, and thus all customers have divided their spending capacity among the facilities. This gives the final division of customer loads among all facilities, $\sigma(s, v_i)_j$, from which the utilities of each facility f_j can be deduced using $\ell_j(s, \sigma) = \sum_{i=1}^{n} \sigma(s, v_i)_j$.

4.2 Python implementation and limitations

The algorithms stated above are translated into Python code which can be applied to any nonempty directed input graph. This input graph is represented by a set of vertices, $V = \{v_1, v_2, ..., v_n\}$, with vertex weights $w(v_i)\ \forall\ v_i \in V$, and set of edges $E = \{e_1, e_2, ..., e_m\}$. The code consists of mainly dictionaries and arrays to represent the vertices and edges, and to store all valuable information.

The input graphs can be created using standard packages in Python. Some of these packages create undirected graphs only. These graphs can be transformed into directed graphs by replacing every undirected edge (u, v) by two directed edges (u, v) and (v, u).

One small issue occurs when using multiplication on so-called 'float' integers in Python, which are integers represented by base 2 binary fractions. Not all decimal fractions can be exactly represented by a binary fraction, causing a slight inaccuracy in the approximation of the decimal fraction. When these decimal fraction with slight inaccuracies are multiplied, the inaccuracy gets multiplied as well. In some parts of the code, two different decimal fractions are checked to be the same. However, the small inaccuracies can give a misleading result. Therefore, a small margin of 0.001 is taken into account, when checking for the values to be the same.

When looking for the best possible location strategy, the process of comparing strategies can take up a lot of time. Any location strategy s needs to be compared to all possible (s'_j, s_{-j}) for all facilities f_j. In case of n vertices and k facilities, this could add up to a total of $k^{(n-1)}$ com-

parisons. To speed up this process, the facility utilities of the location *distributions* are stored. The location distribution describes only how many facilities are placed at every location. After all, the specification of which facility is located at what vertex does not influence the utilities of a location. If, for instance, facility f_j is placed at vertex v_j and f_k at v_k, then a location swap of the two facilities would not influence the utilities at vertices v_j and v_k. Therefore, the stored utility values can be used if a certain location configuration has already been analysed before, but where some facilities have swapped positions. In both location configurations, the utilities for a facility at a certain location remain the same. Therefore, the same value for the facility utility can be used, which can save a significant amount of time.

4.3 Data collection

While running the algorithm, a counter keeps track of how many strategy improvement steps are taken until an equilibrium is reached. As a consequence, the relation between the number of improvement steps and a to-be-investigated variable can be depicted in a figure.

The social welfare of the final location strategy is computed by adding all vertex weights of all covered vertices, like in Equation (1). The optimal social welfare is computed using another method, as it is not desirable to loop over all possible location strategies s. Instead, the graph instance can be represented by a linear program, for which two new variables are introduced.

$$x_e = \begin{cases} 1 \text{ if } v_e \in s, \text{ so there is a facility located at vertex } v_e \\ 0 \text{ if } v_e \notin s, \text{ so there is no facility located at vertex } v_e \end{cases}$$

$$y_e = \begin{cases} 1 \text{ if } N_s(v_i) \neq \emptyset, \text{ so there is a facility located in the neighbourhood set of vertex } v_e \\ 0 \text{ if } N_s(v_i) = \emptyset, \text{ so there is no facility located in the neighbourhood set of vertex } v_e \end{cases}$$

The goal is to maximize the weighted participation range. With the use of the variables x_e and y_e, the following objective function can be set up with its constraints. Here, v is the total amount of vertices of the graph.

$$\begin{aligned} \max \quad & \sum_{i \in V} \big(w(v_i) \cdot y_i\big) & \\ \text{s.t.} \quad & x_i \in \{0,1\} & \forall i \ \in [0, v] \\ & y_i \in \{0,1\} & \forall i \ \in [0, v] \\ & y_i \leq 1 & \forall\ i \in [0, v] \\ & y_i \leq \sum_{f \in N_s(v_i)} x_f & \forall\ v_i \in V \\ & \sum_{i=0}^{v} x_i \leq k & \end{aligned}$$

The solution of this linear program can be found using Gurobi, a Python add-on for solving optimization problems. The solution is equal to $max_{s_i \in S} W(s_i)$, the highest possible value of $W(s)$. As a result, the efficiency η can be computed using Equation (2). Again, figures can be computed to show the dependence of the efficiency on graph properties. This is an interesting result, because it shows the time complexity and efficiency of the algorithm on different graph types.

5 Results

Since there are six different parameters which all can have multiple values or distributions, the number of different parameter combinations is too high to look into detail in all of them. Therefore, only the most interesting relations have been investigated.

5.1 Distribution of the improvement step

For this simulation, we look into the distribution of the improvement step. The following values are constant throughout the whole simulation.

Description	Symbol	Value
number of nodes	n	25
weight distribution	w	constant: 4
number of facilities	k	$\lceil n/4 \rceil = 7$
probability of edge	p	0.3
graph type	G	Erdös-Renyi
location strategy	s	$\{v_0, v_0, ..., v_0\}$
number of runs	r	100

Table 1: Used parameters and their values

The algorithm is applied a total of 100 times, $r = 100$, where in every run a new random graph is created with the properties mentioned above: a random $G_{20,0.3}$ graph with 30 nodes, each with weight 4, and a probability of an edge addition of 0.3. There are 7 facilities and their initial strategy is to all start on vertex v_0. The number of improvement steps and efficiency are stored for every run, which results in the following distributions.

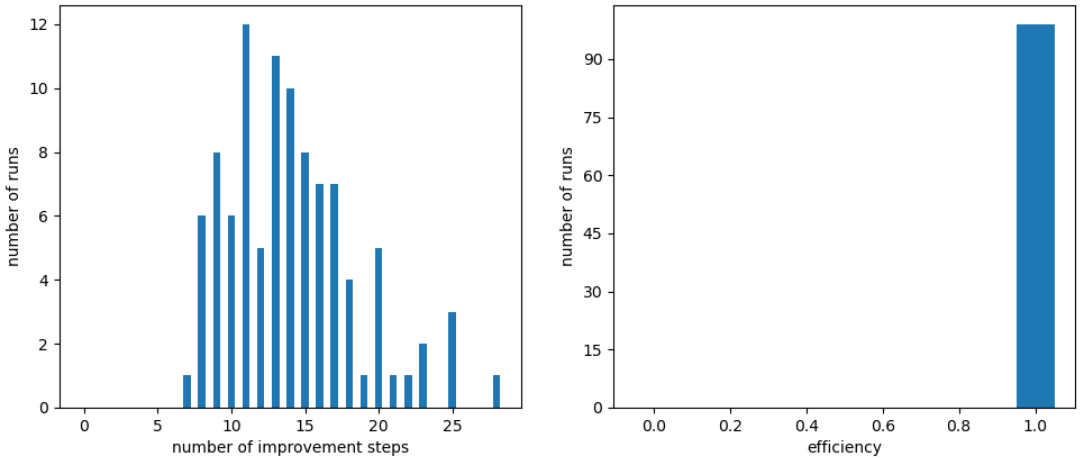
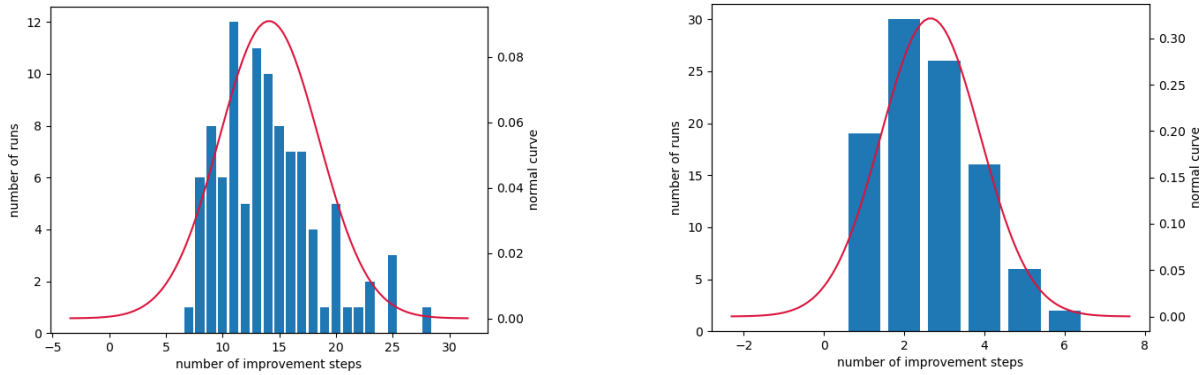


Figure 1: Distribution of number of improvement steps and efficiency for 100 runs with 25 nodes and 7 facilities

The efficiency seems to almost always be equal to (or close to) 1, which indicates that the equilibrium has the highest possible social welfare.

The number of improvement steps resembles a normal distribution. To validate this assumption, we fit a normal distribution to the data in Python. This can be seen in figure 2. The data fits a normal distribution with $\mu = 14.101$ and $\sigma = 4.3820$. This figure also shows a comparison to the distribution of the improvement step when applying the algorithm under similar condition to graphs of only 7 nodes, and thus $\lceil 7/4 \rceil = 2$ facilities.



(a) Normal fit with n=25

(b) Normal fit with n=7

Figure 2: Normal fits to the distribution of the number of improvement steps

It clearly shows that the simulation with a smaller number of nodes shows a higher accuracy with the normal fit. This result may be surprising at first thought, because bigger graph instances usually result in a higher accuracy. However, a higher number of nodes means the total number of possible random graphs is also higher. Graphs with many nodes can show more dissimilarities between each other than small graphs, and thus the results will be less accurate. Another cause of the inaccuracy could be that the number of improvement steps is in general lower for small graphs. Therefore the range of possible values it can take becomes smaller as well, and thus the data shows a normal distribution more clearly.

5.2 Relation of the fraction of facilities

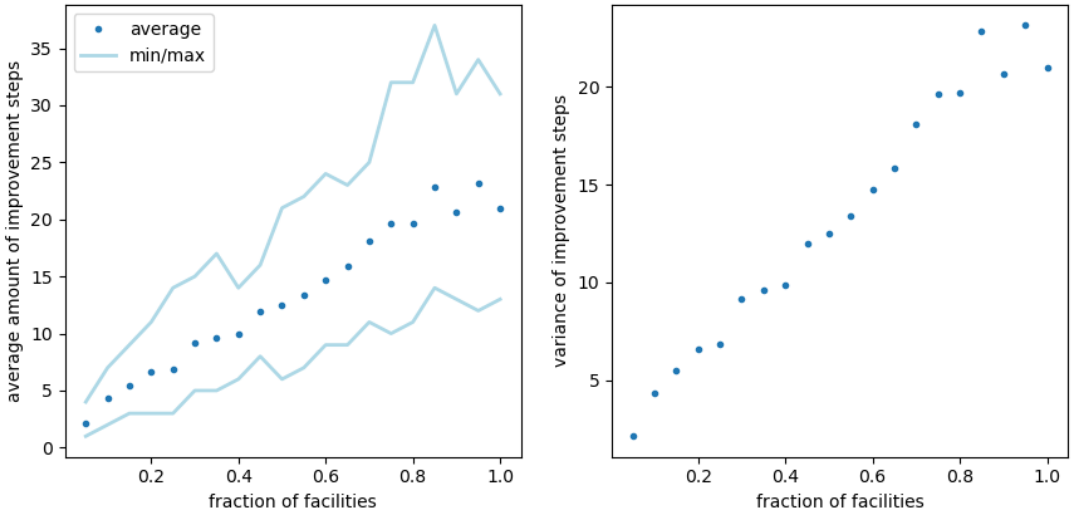
The number of facilities in comparison to the total number of nodes, $f = \frac{k}{n}$, can influence the two algorithm properties. To find the relation, we run the following simulation. The graph properties are similar to those of Section 5.1.

Description	Symbol	Value
number of nodes	n	20
weight distribution	w	constant: 4
number of facilities	k	[1, 20]
probability of edge	p	0.3
graph type	G	Erdös-Renyi
location strategy	s	$\{v_0, v_0, ..., v_0\}$
number of runs	r	30

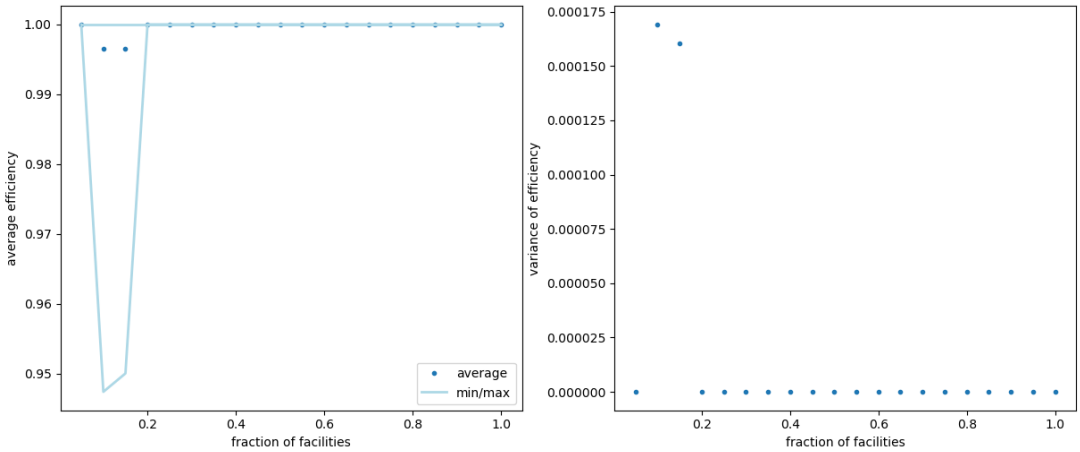
Table 2: Used parameters and their values

The number of facilities, k, is variable. We start with $k = 1$, one facility in the whole game, and we apply the algorithm to a random graph 30 times. Every run, a new random graph is created. Again, the number of improvement steps and efficiency are stored. The average value and variance of these properties are computed and linked to the corresponding value of k. Also,

the minimum and maximum value for each k is stored. This process continues for $k \in [1, 20]$, so in total the algorithm is applied to $20 \cdot 30 = 600$ random graph instances. The relation between the fraction of facilities, $\frac{k}{20}$, can be plot against the average value, minimum, maximum, and variance.



(a) Number of improvement steps dependent on the fraction of facilities



(b) Efficiency dependent on the fraction of facilities

Figure 3: Influence of the fraction of facilities

In figure 3a, the average value of the improvement steps shows a clear linear relation. As the ratio between the number of facilities and vertices grows closer to 1, more improvement steps need to be made until an equilibrium is found. This makes sense, since a higher number of facilities leads to a larger set of possible strategies that need to be compared. The range of the values also grows larger, which can also be seen in the linear growth of the variance.
The efficiency is remarkably high for all fractions of facilities, as the lowest value is 0.95. The lowest fractions show some instability, but as soon as the fraction of facilities reaches 0.2, every equilibrium has an efficiency of 1.0. This means that a game with at least $0.2n$ facilities will find an efficient equilibrium with the highest possible social welfare. In contrast to the improvement step, the variance of the efficiency grows to 0 quickly as more facilities are introduced.

5.3 Relation of number of nodes

The number of nodes represents the total number of clients and locations. We vary the number of nodes in a graph for different graph types. The other properties can be seen in the table below.

Description	Symbol	Value
number of nodes	n	[5, 25]
weight distribution	w	constant: 4
number of facilities	k	$\lceil n/4 \rceil$
probability of edge	p	0.3
graph type	G	*variable*
location strategy	s	$\{v_0, v_0, ..., v_0\}$
number of runs	r	50

Table 3: Used parameters and their values

To minimize the influence of the number of facilities, the fraction is kept roughly the same at $k = \lceil n/4 \rceil$. All facilities still start on vertex v_0 and all vertex weights are equal to 4. The graph types that are used have different inputs, of which further explanation can be found in Section 3.1.

Erdös-Renyi: $p = 0.3$

Newman-Watts-Strogatz: $m = 1 + \lceil n/5 \rceil$, $p = 0.3$

Barabási-Albert: $m = \lceil n/5 \rceil$

Power law: $m = 1 + \lceil n/5 \rceil$, $p = 0.3$

Regular: $d = \lceil n/5 \rceil$.
If n and d are both odd, the value of d is increased by 1 to satisfy the property that $nd = 2|E|$.

The value of n ranges from 5 to 25. For every value of n, a random graph is created and the algorithm is applied to find the equilibrium. This happens 50 times for every value of n, for the 5 different graph types. The average value, range, and variance of the number of improvement steps and efficiency are stored and matched to the corresponding value of n. This gives the following results.

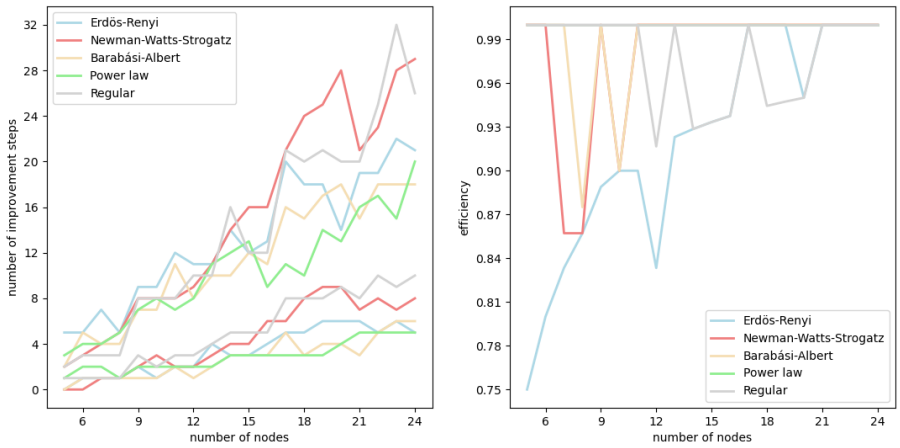
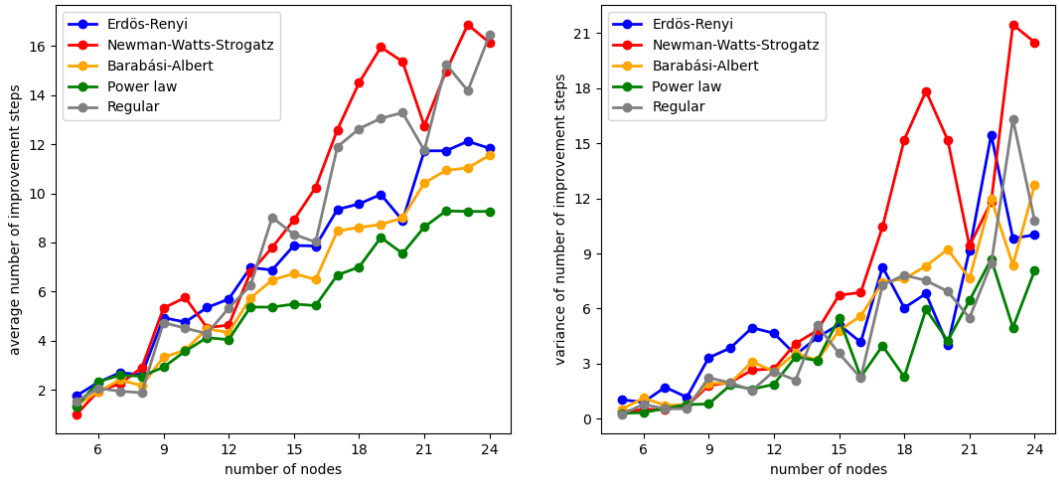


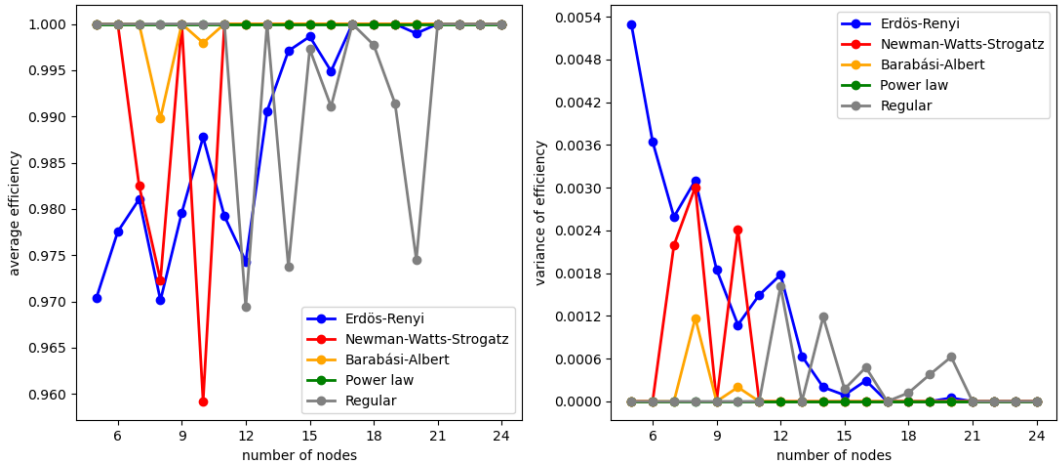
Figure 4: Minimum and maximum value of the number of improvement steps and efficiency dependent on graph size for different graph types

Figure 4 shows that the minimum and maximum number of improvement steps show an increase for all five graph types for increasing values of n. This result was expected, as the number of possible strategies depends on n. The Newmann-Watts-Strogatz graph, with the small-world property, shows the biggest increase. The power law graph, on the other hand, has the smallest increase and thus an equilibrium can be found quickest in a graph of this type. This can also be seen in the average values in Figure 5a. The variance shows similarities with the average. It grows largest for the small-world graph and least for the power law graph.

The efficiency increases for all graph types as the number of nodes increases. This means the found equilibrium has higher social welfare as the graph becomes bigger. Figure 5b indicates that the efficiency goes to 1 for all graph types. Although the efficiency is already high for all graph types and the variances are relatively small, the improvement of the efficiency can be seen the most in the Erdös-Renyi graph.



(a) Number of improvement steps dependent on graph size for different graph types



(b) Efficiency dependent on graph size for different graph type

Figure 5: Distributions of the number of improvement steps and efficiency for different graph types dependent on number of nodes

5.4 Influence of initial strategy profile

In previous simulations, the initial strategy was $s = \{v_0, v_0, ..., v_0\}$, so all facilities starting on vertex v_0. To investigate the influence of the initial strategy, the simulations are repeated with other initial strategies.

First, we look into the situation where the facilities are evenly spread throughout the graph. For facility f_j, the input location strategies will be $v_{\lfloor nj/k \rfloor}$: $s = \{v_{\lfloor \frac{n}{k} \rfloor}, v_{\lfloor \frac{2n}{k} \rfloor}, ..., v_n\}$
Another possibility is to first compute the maximum weighted participation range, as explained in Section 4.3. For the initial strategy, the facilities are divided according to the maximizing distribution found in the linear program. This initial strategy gives the highest possible social welfare, but does not necessarily need to be an equilibrium.
Lastly, we look into the strategy where all facilities are located on the vertex with the highest indegree. This implies that this vertex has the most in-going edges. The initial strategy becomes $s = \{v_j, v_j, ..., v_j\}$ where $v_j = max_{\{v \in V\}} deg^-(v)$.

Description	Symbol	Value
number of nodes	n	[5, 25]
weight distribution	w	constant: 4
number of facilities	k	$\lceil n/4 \rceil$
probability of edge	p	0.3
graph type	G	Erdös-Renyi
location strategy	s	*variable*
number of runs	r	50

Table 4: Used parameters and their values

The number of nodes of the graphs is varied between 5 and 25, where each node has a weight of 4. An Erdös-Renyi graph is created with n nodes and an edge probability of 0.3. Again, we keep the fraction of facilities somewhat the same at $k = \lceil n/4 \rceil$. For each n, 50 random graphs are created and the algorithm performance is saved. This repeats for all values of n, for all different initial location strategies.

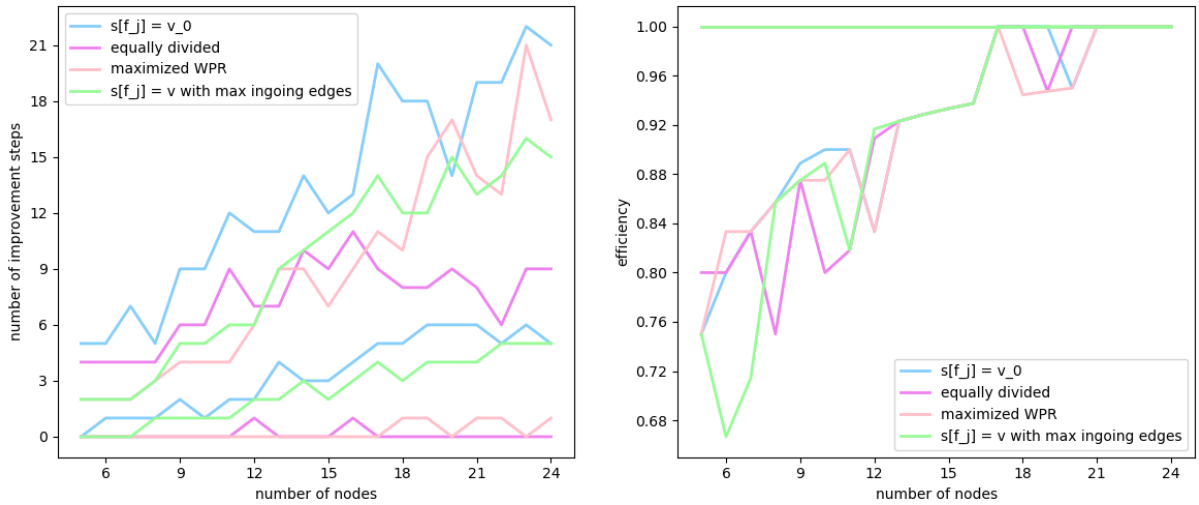
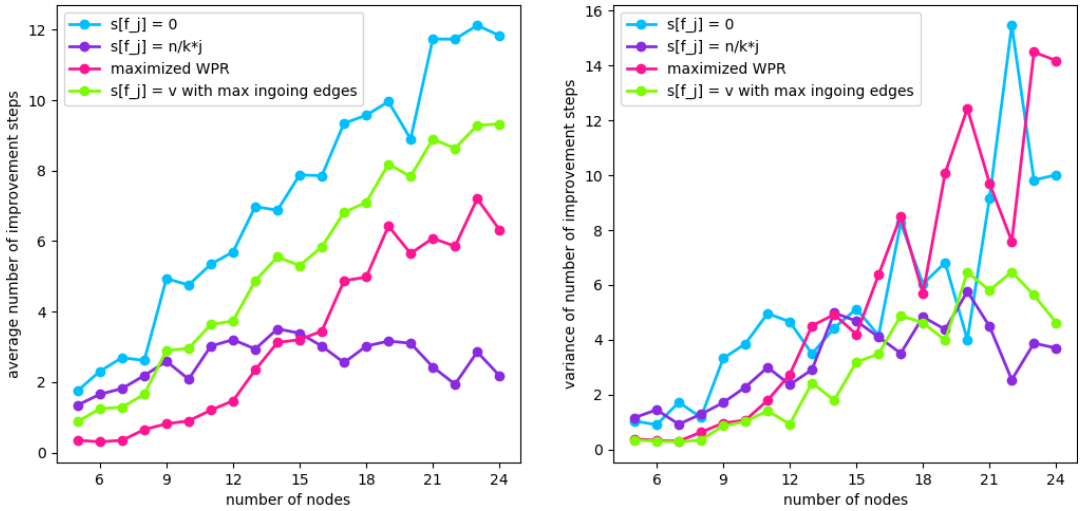


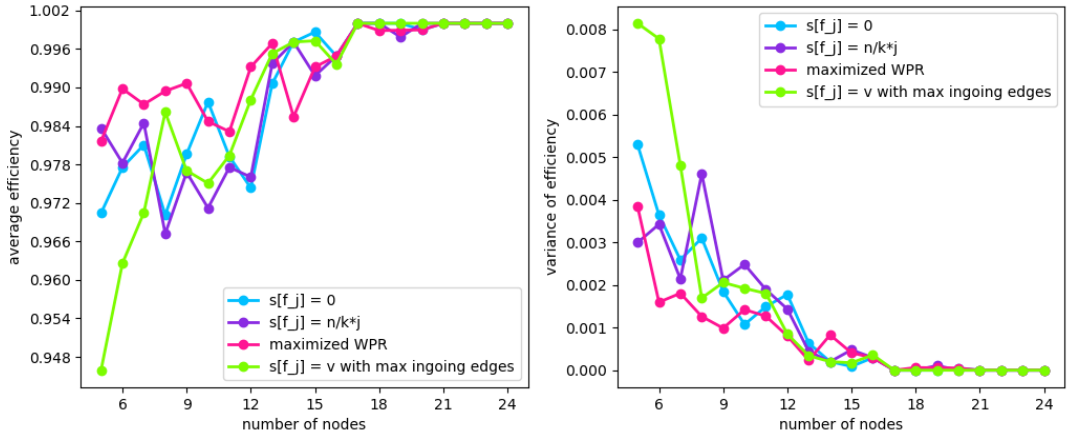
Figure 6: Range of the number of improvement steps and efficiency dependent on the number of nodes for different initial strategies

The results show that the minimum number of improvement steps is significantly low for the strategies where the facilities are equally divided over the graph, and where the facilities are located to maximize the weighted participation range (W). This also gives them a low average, and thus they are good initial strategies to quickly find an equilibrium. We also see that these two strategies have a low variance as n increases, compared to the other strategies. Placing all facilities at vertex v_0 seems to be most time consuming as many improvement steps need to be made. This result is not too surprising. In general, the equilibrium strategy is most likely to be spread over many location vertices. Therefore, starting with all facilities on one vertex takes many improvement steps until they are spread over the graph.

The efficiencies of the strategies are comparable, and again they all have a low variance and grow towards an efficiency of 1. It is least efficient to place all facilities on the vertex with the highest indegree. and most efficient to locate them according to the solution of the linear program.



(a) Number of improvement steps dependent on graph size for different initial strategies



(b) Efficiency dependent on graph size for different initial strategies

Figure 7: Distributions of the number of improvement steps and efficiency dependent on number of nodes for different initial strategies

5.5 Influence of the edge density

The edge density resembles the connections between customers and facilities. If a node has many outgoing edges, it means they can visit many other location vertices within the graph. We look into the impact of the edge density on a graph with 20 nodes and 5 facilities.

Description	Symbol	Value
number of nodes	n	20
weight distribution	w	constant: 4
number of facilities	k	5
probability of edge	p	[0.05, 1]
graph type	G	*variable*
location strategy	s	{0, 0,..., 0 }
number of runs	r	50

Table 5: Used parameters and their values

Again, all vertices start on v_0 and have a vertex weight of 4. We use the five different graph types listed in Section 3.1, with the following properties.

Erdös-Renyi: $p = [0.05, 1]$

Newman-Watts-Strogatz: $m = 1 + np$, $p = [0.05, 1]$

Barabási-Albert: $m = np$

Power law: $m = 1 + np$, $p = [0.05, 1]$

Regular: $d = np$

If n and d are both odd, the value of d is decreased by 1 to satisfy the property that $nd = 2|E|$.

The value of p starts at 0.05 and increases with steps of 0.05 to 1. For every value of p, an equilibrium is computed for 50 different random graphs. The average number of improvement steps and efficiency are stored, as well as their variances. The results provide a clear relation between the edge density and the performance of the algorithm.

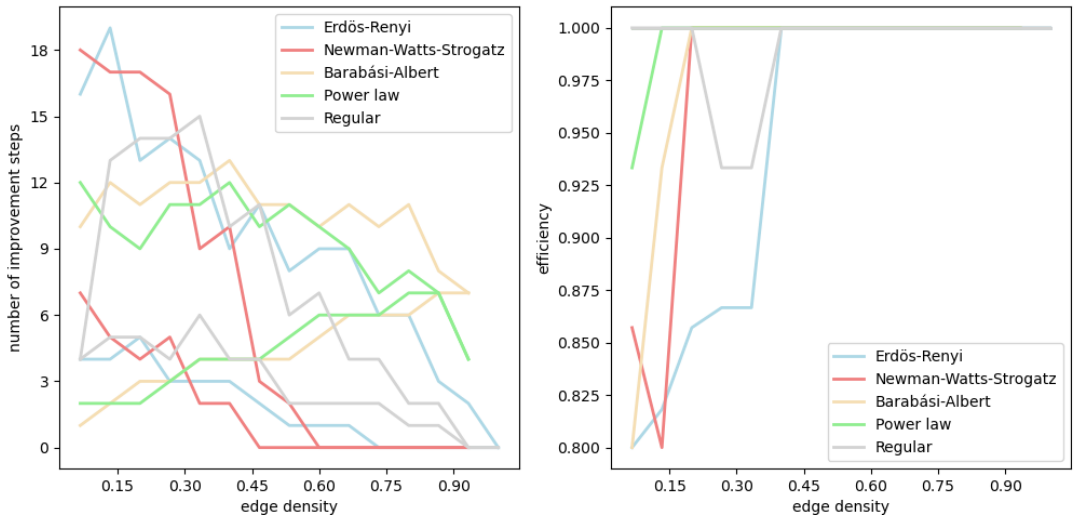
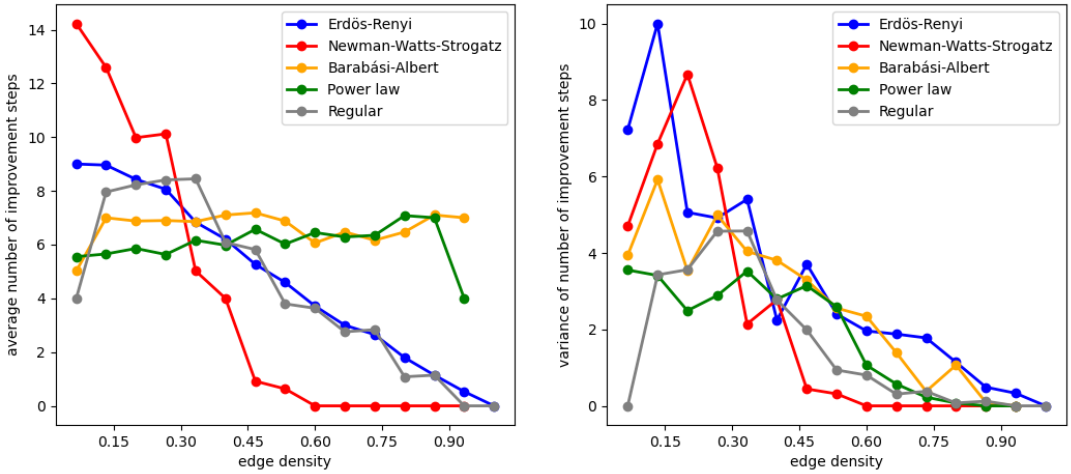


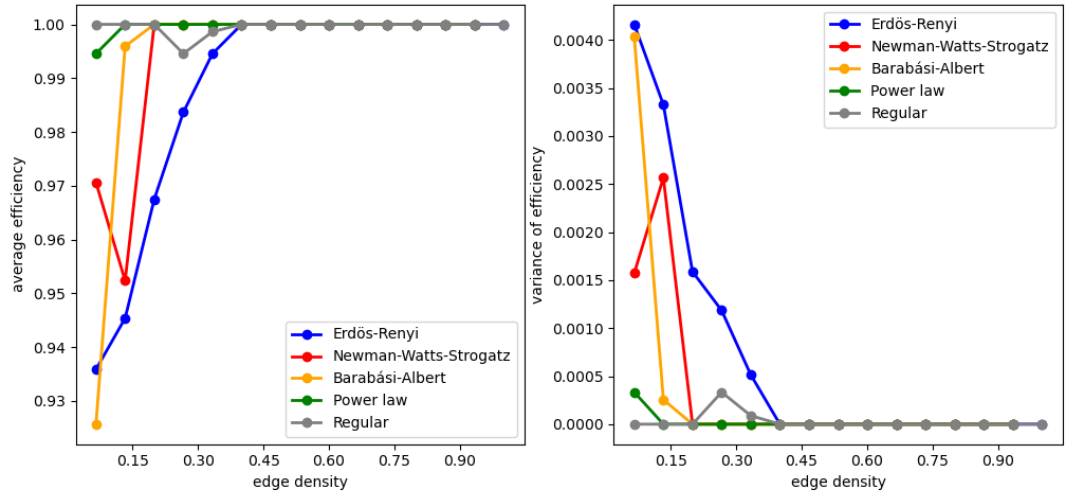
Figure 8: Minimum and maximum values of the number of improvement steps and efficiency for different graph types dependent on edge density

The Newman-Watts-Strogatz graph needs the least amount of improvement steps to find an equilibrium as the edge density increases. This graph type reaches a maximum value of 0 quickly, as can be seen in Figure 8. Another decrease can be seen in Figure 9a for the regular graph and Erdös-Renyi graph, where the relation with the latter seems to be linear. When applying the algorithm to the two remaining graph types, Barabási-Albert and Power law, the number of improvement steps stays on the same level, regardless of the edge density of the graph. This could be explained by the fact that these two graph types contain clusters with high edge density already. The minimum and maximum value of the number of improvement steps also grow closer together as more edges are added to the graph, causing the variance to drop. In general, we see for all graph types that the variance decreases for a higher value of p.

Figure 9b shows that the average efficiency is 1.0 for the regular graph for all values of p. The efficiency for the Barabási-Albert model graph starts relatively low but increases quickest to a value of 1.0. The efficiency when applying the algorithm to an Erdös-Renyi graph starts in the same range, but increases slower. However, it also eventually reaches an efficiency of 1.0.



(a) Number of improvement steps dependent on edge density for different graph types



(b) Efficiency dependent on edge density for different graph types

Figure 9: Distributions of the number of improvement steps and efficiency for different graph types dependent on edge density

5.6 Influence of the weight distribution

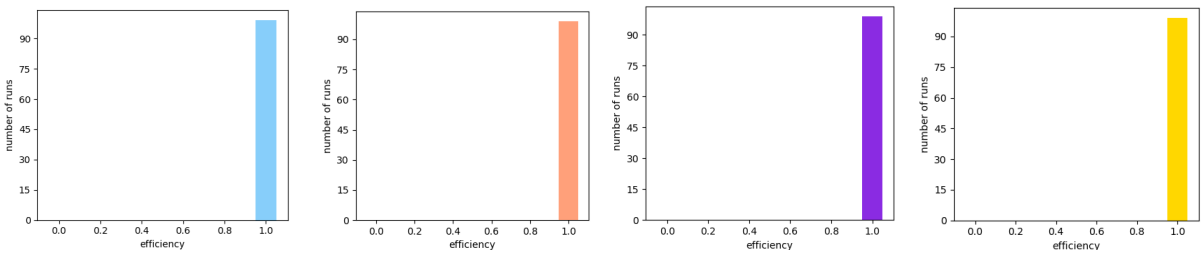
The weight distribution has been constant for all previous simulations, as all vertices had a vertex weight of 4. We now look into the distribution of the number of improvement steps and efficiency, for Erdös-Renyi graphs with 25 nodes, 7 facilities, and the other properties from the table below.

Description	Symbol	Value
number of nodes	n	25
weight distribution	w	*variable*
number of facilities	k	7
probability of edge	p	0.3
graph type	G	Erdös-Renyi
location strategy	s	$\{v_0, v_0, ..., v_0\}$
number of runs	r	100

Table 6: Used parameters and their values

Every facility starts at v_0. For each weight distribution, 100 random graphs are created to which the algorithm is applied. Before the algorithm is applied, all vertices get assigned their vertex weight by randomly choosing an integer according to the chosen weight distribution. The number of improvement steps and efficiency are tracked, of which histograms can be made to visualize the distribution.

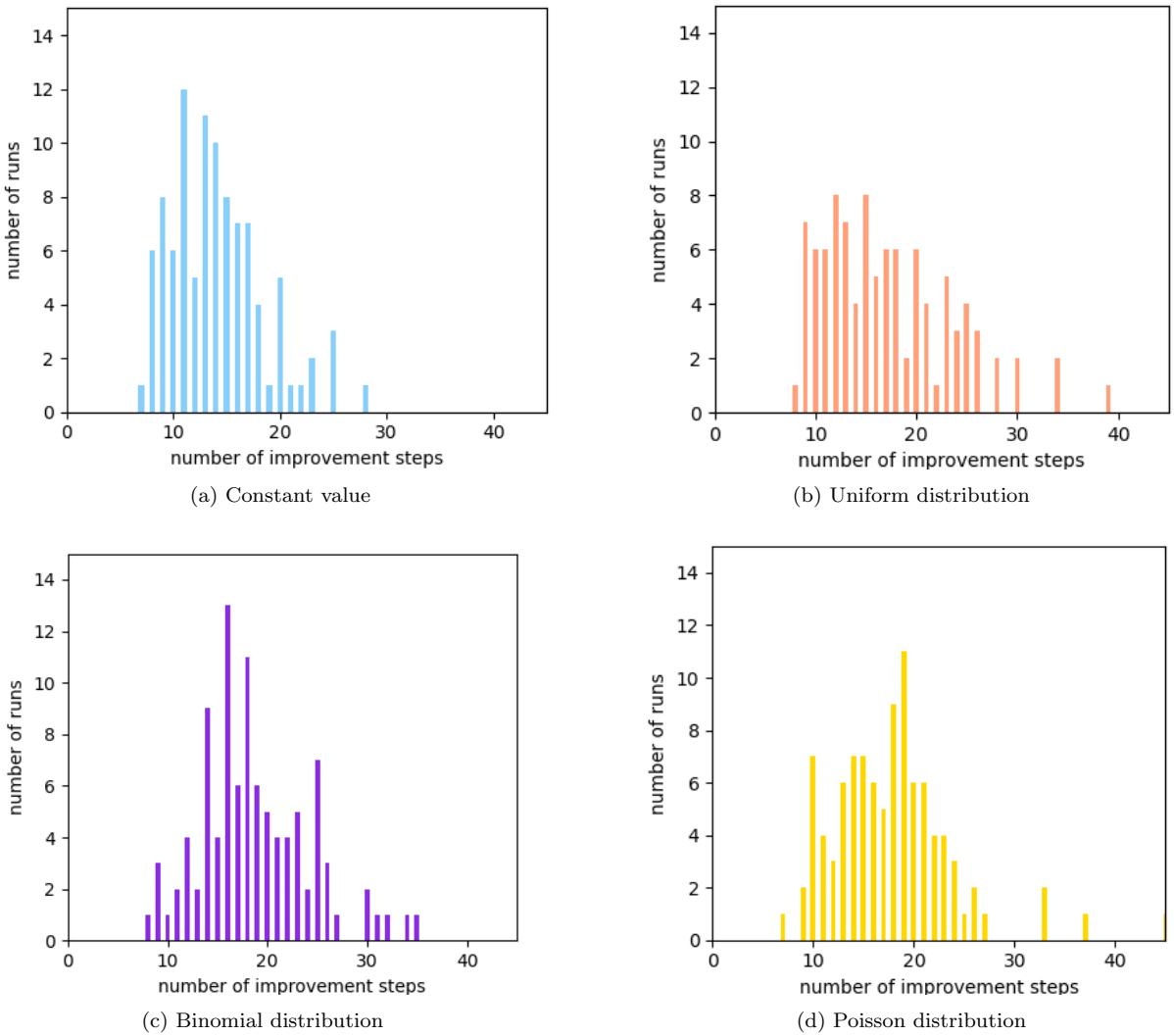
The first distribution is the constant value, used in previous simulations: $w(v_i) = 5 \; \forall \; v_i \in V$.
Next is the uniform distribution $U(a, b)$, of which further details are explained in Section 3.2. We choose $a = 2$ and $b = 8$, such that the average value is 5.
The third distribution is the binomial distribution $B(n, p)$. The parameters are chosen as $n = 15$ and $p = 1/3$ to again make the average value $np = 5$.
The last distribution is the Poisson distribution. In the Poisson distribution, the expectation is equal to λ, and therefore we choose $\lambda = 5$.



(a) Constant (b) Uniform (c) Binomial (d) Poisson

Figure 10: Distributions of the efficiency for different weight distributions

As can be seen from the figure above, the efficiency is (close to) 1.0 for all 100 runs for all weight distributions. A significantly high efficiency is a familiar result we have seen in previous simulations as well.



(a) Constant value

(b) Uniform distribution

(c) Binomial distribution

(d) Poisson distribution

Figure 11: Distributions of the number of improvement steps for different weight distributions

The number of improvement steps, however, does show different results. The average value and standard deviation of all histograms can be seen in the table below.

	μ	σ
constant	14.101	4.382
uniform	17.192	6.392
poisson	17.758	5.982
binomial	18.748	5.522

Table 7: Average value and standard deviation of distributions

The average number of improvement steps is highest for the binomial distribution and lowest for the constant value. The graphs where a uniform distribution is applied, show a high variance compared to the others. This is probably caused by the fact that the variance of the uniform distribution itself is quite high as well, causing big differences in the vertex weights and thus the equilibria. Again, the constant value has the lowest variance. This makes sense because the constant value does not include a random factor, while all other three distributions do.

6 Conclusion

We have found that the algorithm behaves differently when certain graph properties are changed. The simulations show that the number of improvement steps follows a normal distribution, and that the efficiency is maximum for the vast majority of the graph inputs.
If one would want to minimize the number of improvement steps that have to be taken before an equilibrium is found, it would be most beneficial to apply the algorithm to a power law graph with as few facilities and nodes as possible, where every node has the same spending capacity. Additionally, a high edge density and an even spread of all facilities through the graph have a positive influence on the number of necessary improvement steps, as it will decrease. However, it is not too likely that a situation in real life can be represented by a graph with all of these properties.
It would be interesting to study the influence of varying even more parameters at once. This gives a better analysis of the behaviour of the algorithm. Also, it would be beneficial to look into the details of which graph properties mostly can be found in realistic representations of customer and company locations, and how the algorithm behaves on these.

References

[1] Hotelling H. Stability in competition. *The Economic Journal*, (39(153)):41–57, 1929.

[2] Simon Krogmann, Pascal Lenzner, Louise Molitor, and Alexander Skopalik. Two-stage facility location games with strategic clients and facilities. 2021.