Analyzing the Computing Time to Solve Single Row Facility Layout Problems by Simulated Annealing in a Python Framework

Alexandre Miccoli
School of Business, University of Applied Sciences and Arts Northwestern Switzerland, Olten, Switzerland
alexandre.miccoli@students.fhnw.ch

Thomas Hanne*
Institute for Information Systems, University of Applied Sciences and Arts Northwestern Switzerland, Olten, Switzerland
thomas.hanne@fhnw.ch

Rolf Dornberger
Institute for Information Systems, University of Applied Sciences and Arts Northwestern Switzerland, Basel, Switzerland

ABSTRACT

The goal of this paper is to assess the Python computing time to solve a single row facility layout problem (SRFLP) by Simulated Annealing. The optimization problem is introduced, systematically modelled and then optimized numerically using a particular Python framework. The computing time and the results of experiments with various problem sizes and parameters are analyzed and discussed.

CCS CONCEPTS

• **Mathematics of computing** → Discrete mathematics; Combinatorics; Combinatorial optimization; • **I Computing methodologies** → Artificial intelligence; Search methodologies; Randomized search.

KEYWORDS

Single row facility layout problem, Layout design, Simulated annealing, SRFLP, FLP

ACM Reference Format:
Alexandre Miccoli, Thomas Hanne, and Rolf Dornberger. 2023. Analyzing the Computing Time to Solve Single Row Facility Layout Problems by Simulated Annealing in a Python Framework. In *2023 7th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (ISMSI 2023), April 23, 24, 2023, Virtual Event, Malaysia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3596947.3596953

1 INTRODUCTION

The correct arrangement of resources such as machines in a production plant is an important factor for the success of manufacturing companies [10]. In such industries, the costs of handling materials can make up 20-50% of the total costs of production [10],[16], and optimizing the positioning of machines, personnel, and any other resource used in production may help reducing these costs by 10-30% [4]. Optimizing the facility layout to find a (nearly) optimal solution in terms of minimized costs is therefore a key tactical issue [4].

*Corresponding author.

ISMSI 2023, April 23, 24, 2023, Virtual Event, Malaysia

ACM ISBN 978-1-4503-9992-0/23/04.
https://doi.org/10.1145/3596947.3596953

In manufacturing, human resources, machines, materials, and pieces of equipment can be regarded as the main resources to be distributed in the workshop [13], but the facility layout problem (FLP) can be applied to many other industries. For example, how to place books in a library, how to locate products in grocery stores, or how to align departments of a service company or wards of a hospital are questions that can be answered by applying the FLP [16]. Facility layout problems therefore have a broad scope of applicability and have thus been studied for several decades [16].

The first research contributions on the topic of FLPs date back to the late 1950s [4],[10]. Since then, many attempts have been made to find better solutions to solve such problems, which are generally classified as non-polynomial hard (NP-hard) so that finding an optimal solution within reasonable time is often not possible. Instead, the algorithms aim to find good solutions rather than an optimal solution [4],[10],[13],[16].

Many subtypes of FLPs are being studied. The single row facility layout problem (SRFLP) is one of them and might be described as one of the easier FLPs, due to its lower number of parameters and constraints. In the SRFLP, all facilities (e.g., machines) consist of equally sized rectangles, which are distributed on a straight line [16]. The problem will be covered in more detail in the next section, because solving SRFLPs of various sizes is the focus of this paper. Further details of FLP variants and surveys of the field can be found in [2], [7], [12], and [17].

When looking for software solutions to the FLP, packages written in languages such as MATLAB or SCALA are quickly found. In recent years, an increasing number of solutions written in Python can be found among the plethora of software packages. This programming language was introduced by the Dutch software engineer Guido van Rossum in 1991 [14]. An improved code readability and good performance are important attributes of this language [14]. These attributes have certainly contributed to Python repeatedly being among the top three most popular programming languages in the last couple of years [11],[15]. Another aspect that makes Python highly popular is the low entry barriers for new developers, as described in [6].

Many benchmark studies on different problems and various programming languages and paradigms can be found online. Concrete applications of Python to solve the FLP can be found as well, but judging their performance based on existing publications is tedious. For this reason, we present a benchmark study for solving an SRFLP by Simulated Annealing (SA) with Python in this work.

Section 2 presents the considered model in detail, whereas Section 3 discusses the used optimization method. Details of the experimental setup are presented in Section 4, followed by a discussion of results in Section 5. The paper concludes with Section 6.

2 PROBLEM MODEL

The goal of every FLP is to find facility arrangements which minimize material handling costs (MHC) [10],[16], while certain quantitative and/or qualitative constraints must be satisfied [13]. Among the MHC, fixed costs such as equipment costs, efforts to rearrange facilities or the costs for the handling itself can be included [10],[13]. Under these preconditions, the problem is said to be multi-objective [10],[13]. The FLP can be formulated mathematically in different ways, depending on how heterogeneous the facilities to be modelled are [10].

Problems with facilities of equal sizes can be represented discretely as quadratic assignment problems (QAP) [10]. Thereby, the available shopfloor space is divided into equally sized squares on which the facilities are placed [10].

FLPs with heterogeneous facility sizes can be represented as quadratic set problems (QSP) [10]. There, the real surface covered by the facilities is considered and divided into smaller parts, on which the facilities are placed, whereby each of these parts can only be covered by one facility [10].

Mixed integer programming (MIP) can also be used to represent problems with unequally sized facilities, according to [10]. Therefore, a linear objective function is required, in which constraints are represented as equations and inequations [10]. The complete shopfloor surface can then be used [10].

Finally, graphs can be used to represent the FLP as nodes, connected by edges [10]. The costs or benefits of each material move are then represented by weights assigned to each edge [10]. The optimization then takes place by minimizing the costs (i.e., weights) to pass through each node [10].

The FLP has been generalized by [10] as follows:

$$P = (S,\ \Omega,\ f)$$

S: The solution space, including the set of discrete decision variables X_i $(i = 1,\ \ldots,\ n)$

Ω : The set of constraints, which can include area, position, and budget limitations

f : An objective function to be maximized or minimized

The set of feasible solutions, where the decision all constraints Ω are met by the decision variable, is denoted as $s \in S$. The optimum solution is then defined as the one among all feasible solutions S with the best objective function value.

The SRFLP considered in this paper is about finding an arrangement of facilities (i.e., the permutation of facilities) that minimizes the cost of movements. The latter is defined as the sum of costs times distances. Both inputs (i.e., costs and distances) are defined as matrices, and serve as inputs for the algorithm.

An example cost matrix is depicted in Fig. 1, showing the costs to move from a facility in the first column to any other (e.g., costs of 5 to move from AAAA to AAAB). Staying in-place means costs of 0. Additional details on the experimental setup are provided in the introduction of the numerical experiment.

	AAAA	AAAB	AAAC	...	ZZZZ
AAAA	0	5	9	...	4
AAAB	2	0	2	...	6
AAAC	7	7	0	...	2
...	...	...	...	...	...
ZZZZ	3	5	1	...	0

Figure 1: Example cost/distance matrix

In this problem, only the product of cost times distance is optimized. There is no additional constraint (e.g., order of facilities to keep or positioning of facilities to respect). Still, it is possible to enforce a particular order of facilities by manipulating distances or costs in the input data.

3 OPTIMIZATION METHODS

Numerous methods to solve the FLP have been introduced [1]. For small optimization problems where only a few facilities need to be arranged, solutions that aim for a perfect solution (i.e., exact methods) can be applied [10]. As soon as the problem to be solved gets larger, heuristics or intelligent approaches could be applied, as the computational effort to find a perfect solution would be too high [10]. In this paper, we focus on methods for static FLPs, where the shopfloor arrangement is not meant to change over time, in contrast to dynamic FLPs (DFLP).

Simulated annealing, an approach that belongs to the so-called "intelligent methods", can be used to solve larger FLPs with complex objective functions [4],[10],[16]. Other classifications, such as the one presented by [4], assign SA to the class of meta-heuristic algorithms. SA, like many other algorithms (e.g., Genetic Algorithm or Particle Swarm Optimization), is nature-inspired and shows an evolution over several iterations [13]. The probabilistic method presented in the early 1980s aims at emulating the minimum energy configuration that a solid attains when temperature is decreased, and the material gets "frozen" [3]. The algorithm is capable of "jumping out" of local minima, which otherwise could prevent finding a global minimum [3].

To reach the liquid state, the temperature parameter T needs first to be increased to its maximum [5]. In this state, the particles of the material will arrange themselves randomly, which represents the initial random solution that is used as a starting point for the SA algorithm [5].

The temperature is carefully reduced, and the particles are rearranged [5]. If the costs have been reduced between one iteration and the next, we keep the new solution, otherwise, a probability value based on the Boltzmann constant is used to decide whether to keep or replace the current solution [5]. This concept may help to get out of local minima by allowing a temporary decline of the solution quality, especially while the temperature is high (i.e., in the beginning), while later, already found solutions will be kept with a higher probability [5]. In our case, the temperature, initialized by T0, is reduced by a factor α in each of the M iterations. In the

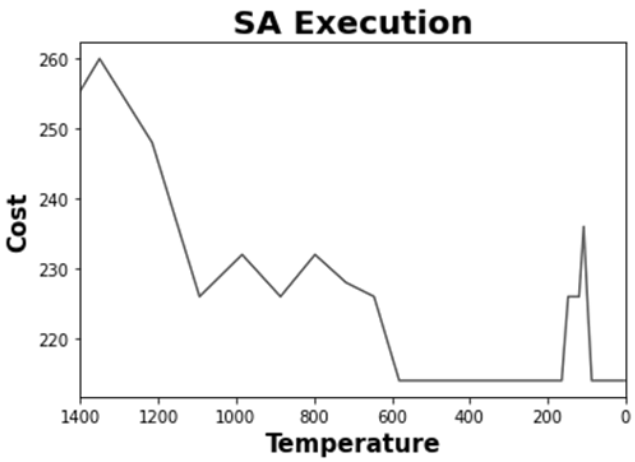

Figure 2: Example execution of the SA algorithm

solution herein discussed, a parameter N is used to determine the number of searches withing a neighborhood during an iteration of the algorithm.

In Figure 2 below, the execution of the SA algorithm is exemplified. The initial (random) solution at the beginning is optimized over 250 iterations in this case (i.e., as temperature decreases). Several local minima have been found, but the algorithm was able to find better alternative solutions later.

4 SETTING UP THE EXPERIMENT ENVIRONMENT

To provide suitable information for performance comparisons, transparency of the processes and repeatability must be ensured. Any disturbing factors affecting the experiment runtime must be excluded so that the results can be trusted. Therefore, an isolated computing environment for the experiment was set up along with randomized input data and a Python script.

4.1 Computing Environment

To conduct experiments without any unnecessary processes possibly affecting the performance of Python, a virtual server was set up, the main characteristics of which are summarized in Table 1.

Debian 11 was chosen as an operating system (OS). The server OS is delivered without graphical user interface (GUI), which allows most resources to be available for the experiment processes.

On top of the OS, Python 3 was added, including PIP, the Python package manager for the dependencies of the experiment script to be installed. Further, the Datadog Agent was installed and configured, so that the infrastructure and the processes could be monitored during the execution of the numerical experiment.

4.2 Generating Input Data

A cost matrix and a distance matrix for each FLP size were generated. To guarantee reproducible results, the input data for each FLP size are kept equal (i.e., each run is performed with the same data).

The inputs are made of comma separated value (CSV) files, having the name of the facilities in both the first row and column, and random integers between 1 and 9 as costs or distances. Thereby, the inputs stay rather homogeneous with no outlying, larger value, which could impact the results.

The naming of the facilities was set to a 4-character string (e.g., AAAA; AAAB; AAAC; . . .; ZZZZ) to always require the same amount of memory and prevent a falsification of the results by different memory occupation.

4.3 Preparing a Python Script

The numerical experiment is based on the work of [8]. The provided script was chosen for its simplicity and transparency in the sense of calculations that are clearly readable. As the work is based on loops with few dependencies, the code is easy to understand, and no hidden background tasks are to be expected. As explained in [9], Python's performance can be negatively affected by such loops. Yet, transparency is judged more important for comparing the overall performance.

The following main steps can be identified from the script provided by [8]:

- Writing input data in so-called "DataFrames"
- Defining execution parameters
- Generating an initial, random solution
- Running the SA algorithm
- Plotting the results (not further considered)

To allow the script to be executed multiple times from the command line interface (CLI), some modifications have been made. First, the execution parameters were defined as environment variables instead of local ones (step 2 above). In addition, the input data is

Table 1: Key characteristics of the computing environment

Characteristic	Value
Core processing unit (CPU)	AMD EPYC 7282 16-Core Processor
Number of cores	6 at 2.8 GHz
CPU cache size	512 KB
Random Access Memory (RAM)	16 GB
Storage	100 GB NVMe
Architecture	ARM 64-bit
Operating system (OS)	Debian 11

```
# python3 /root/python/flp/sa.py 32 1500 250 20 0.8
# <application> <path> <flp size> <T0> <M> <N> <alpha>
```

Figure 3: Example line of code to run the script (line 1) with explanations on the used variables (line 2)

now gathered from the CSV files previously generated (replacing step 1 above). With these adaptions, the script can be executed in CLI as depicted in Fig. 3

For the results to be analyzed later, a logging mechanism was implemented. Thereby, the parameters and results are recorded as entries of a JSON-file in every execution. A timestamp is collected at different events during the execution:

- Once the script has been started and the environment variables have been read, but before loading the input data (i.e., the start event)
- As soon as the input files have been loaded
- Once the initial, random solution has been generated, but before running the SA algorithm
- After the execution of the SA algorithm, once the results have been computed

Note that point 2. above has no direct impact on the execution of the algorithm, but since the input file size is highly dependent on the FLP size (about 73 Bytes for a FLP of size 4 up to 134 Megabytes for a FLP of size 8192), Python's performance at loading such files is worth analyzing.

5 RUNNING THE EXPERIMENTS

Several hours of computation were required to process the larger FLPs. Therefore, batch scripts were generated in advance to run all experiments in an automated fashion, while considering the different execution parameters to be analyzed.

To check the similarity of the results, every experiment (i.e., set of facilities with its distances and costs and parameter sets M, N, T0 and alpha) was conducted ten times. To demonstrate the impact of each parameter on the overall result, only a single parameter was altered for each series of executions. The details on the executions performed are presented in the following two subsections.

5.1 Increasing the Number of Facilities

In the first set of executions, only the number of facilities was modified, following the powers of 2 with a limit of 8192 (=2^{13}) facilities. Stopping at 8192 is an arbitrary decision, based on the highly increased computation time and the estimation that the number of facilities no longer resembles a realistic use case.

The summarized results of the experiment are depicted in Table 2. All time values are expressed in seconds. It can be said that the time-consuming part of the execution is running the SA algorithm. The hypothesis stating that loading the larger CSV files could take considerable time can be rejected, as with the second-largest input (approximately 34 MB for 4096 facilities), no more than 3 seconds were needed. A considerable increase of 21 seconds can be observed for the next-larger file (approximately 134 MB for 8192 facilities), but still, the loading time is a neglectable part of the execution time. The same applies for the time to generate an initial solution, which exceeds only one second for the largest experiment.

Overall, the experiment execution seems to follow an exponential curve, but no clear function could be identified. With up to 256 facilities, execution times of less than one minute could be observed. As the number of facilities increases, a rapid increase in execution time can be observed, leading to up to six hours on average for the largest SRFLP. When considering the deviation, no significant variability in the total execution time can be observed. The average total execution time by experiment size (in minutes on a logarithmic scale) is depicted in Figure 4.

Regarding the cost optimization, the observed results can be considered stable with a standard deviation of 1.3-1.9% of the average optimized costs with the set parameters. In the second set of executions presented below, these parameters were altered to show changes in the algorithm's performance.

5.2 Altering Other Execution Parameters

For the second set of executions, all other parameters were modified. The number of facilities was fixed at 32 (=2^5). This number was chosen arbitrarily, as it represents a not too trivial FLP, but still comes with a fair computation time, as seen in the first round of executions.

In Table 3, the results with different input parameters can be read. For every ten executions, one parameter was altered. Differences can be observed in both computation time and results.

When modifying the *alpha* parameter only, one can see that the standard deviation in the optimized costs is increased by up to 40% compared to the default value of 0.9. It can be assumed that 0.9 is a fair value for this parameter since the results are kept rather constant. The same effects can be observed for changes on parameter $T0$.

If M is increased, the optimized costs do not seem to be affected, but the execution time is different as well. For parameter N, the same effect on execution time can be observed, but since the acceptance of intermediary results is affected by N as well, increasing this parameter leads to a higher variation of the optimized costs.

5.3 Impact on the Infrastructure

Since a Datadog Agent was installed on the machine running the experiments, the load on the infrastructure could be monitored during the experiments.

Considering the memory usage (i.e., RAM occupation), it becomes clear that the larger experiments with 2^{13} facilities have a considerable impact (see Figure 5). Up to 40% of the memory is then used by the Python script. A factor of four can be observed in the required memory to move from 2^n to 2^{n+1} facilities. It can hence be assumed that doubling the number of facilities one more time would have led to the failure of the script, as the available RAM would have been exceeded on the machine used for the experiments described.

As depicted in Figure 6, the CPU usage of the Python process (see purple surface) is close to 16.7% during the executions, which corresponds to one of the six CPU cores (i.e., one thread) used to its full capacity. This result shows that to increase the performance of such a script, re-writing it as a multi-thread application might

Table 2: Summary of results for different numbers of facilities (shown in first row)

	4	8	16	32	64	128	256	512	1024	2048	4096	8192
n	10	10	10	10	10	10	10	10	10	10	10	10
Avg. time to load CSV	0	0	0	0	0	0	0	0	0	1	3	24
Avg. time to generate initial solution	0	0	0	0	0	0	0	0	0	0	1	4
Avg. time to run SA	14	14	15	16	18	23	37	100	305	1'349	5'789	22'068
Min. total time	14	14	14	15	18	23	36	99	304	1'343	5'714	21'269
Avg. total time	14	14	15	16	18	23	37	100	306	1'350	5'793	22'096
Max. total time	14	14	15	16	20	23	38	101	307	1'358	6'335	23'362
Min. costs	296	1'127	5'244	22'488	92'791	389'237	1'588'176	6'437'675	25'951'506	104'287'565	418'259'898	1'675'400'889
Avg. costs	300	1'148	5'315	22'627	93'265	389'894	1'589'396	6'439'873	25'956'764	104'304'538	418'278'781	1'675'474'636
Max. costs	306	1'202	5'400	22'731	93'510	390'586	1'590'861	6'443'880	25'961'061	104'316'709	418'307'709	1'675'507'951
Stdev. costs	4	26	54	76	214	478	939	1'866	3'119	8'795	14'751	31'450

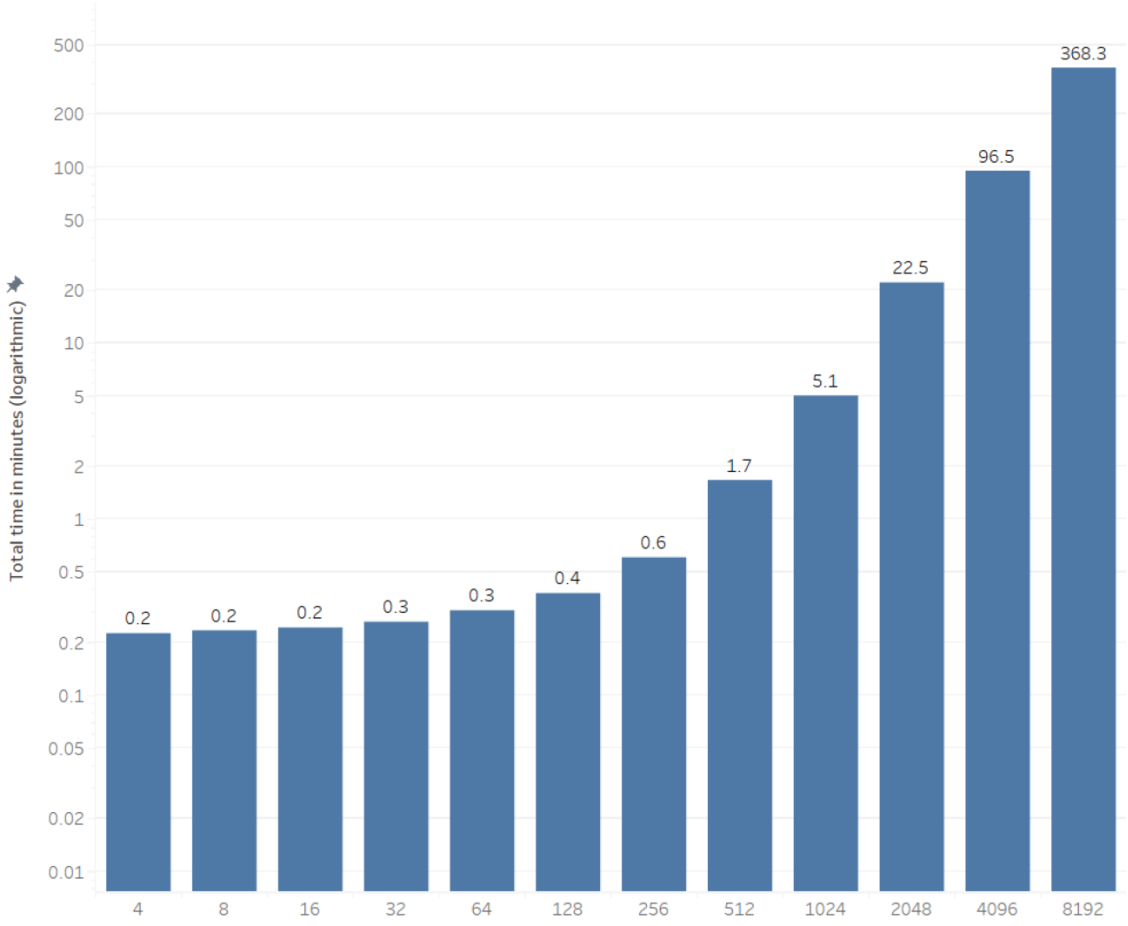

Figure 4: Time increases according to number of facilities

be of interest. Otherwise, the computing power of the single core can be considered the bottleneck for the application.

Since the experiments have been conducted on an isolated virtual machine, and no network or resource-intense I/O-operations were performed by the Python script, no additional parameters were monitored.

6 CONCLUSION

With the experiments presented, we analyzed the single row facility layout problem solved by SA in a Python environment. We analyzed problem instances with different sizes and with various settings of input parameters.

The experiments have demonstrated that the Python implementation used is suitable to solve SRFLPs even with larger problem sizes (i.e., high number of facilities), showing an average computation time of roughly five minutes for a 1024 facility problem with

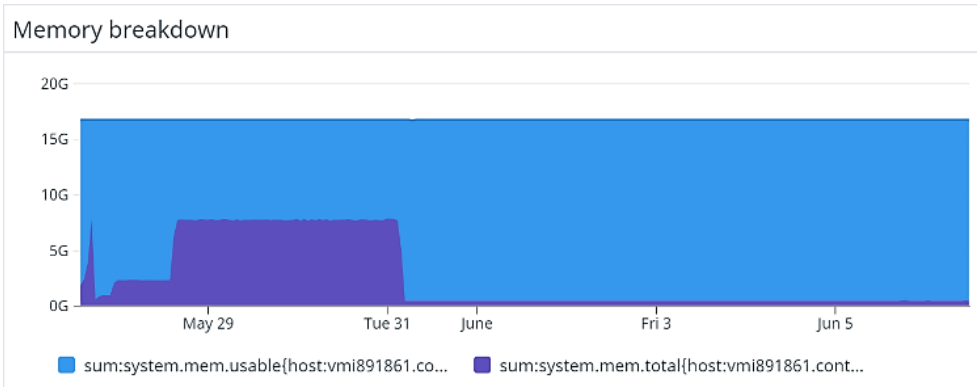

Figure 5: Memory (RAM) consumption during the execution

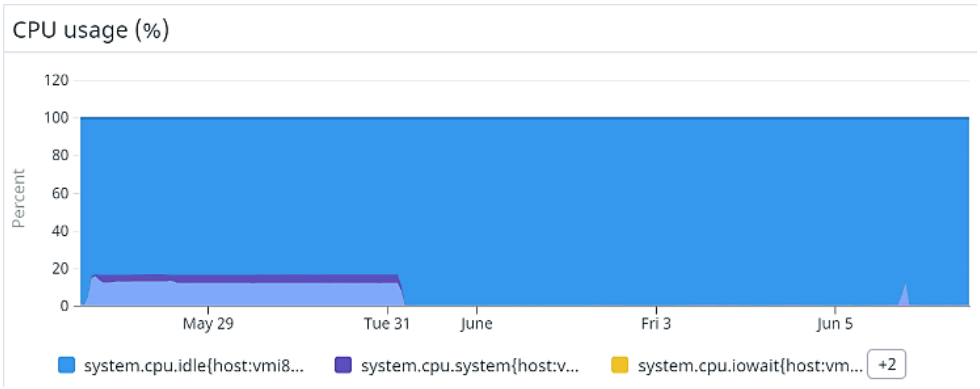

Figure 6: CPU Usage during the execution

Table 3: Summary of results when altering other parameter values

	Alpha / M / N / T0						
	0.8	0.9					0.99
	250	250				500	250
	20	20			40	20	20
	1500	750	1500	3000	1500	1500	1500
Avg. total time	32	30	31	31	66	65	33
Min. costs	22'500	22'381	22'566	22'419	22'472	22'490	22'48
Avg. costs	22'641	22'598	22'718	22'652	22'651	22'642	22'61
Max. costs	22'774	22'823	22'785	22'814	22'894	22'802	22'77
Stdev. costs	105	158	73	135	108	72	10

the used computing infrastructure. However, the experiments have shown potential issues with memory usage. With the increasing number of facilities in the problem, not only the computation time is drastically increased, but the required RAM is much higher. To prevent the Python script from failing, it must be made sure that enough memory is available on the system.

The impact of changes in the set of parameters (M, N, T0, alpha) could be demonstrated, which could also indicate that the set used for the first part of the experiments leads to plausible results and offers a fair performance in terms of computation time.

The single-thread computation of Python (i.e., Python using only one CPU core by default) has been identified as the potential bottleneck in terms of computation speed. Therefore, additional research might be conducted to clarify whether a multi-thread execution of such a script is beneficial.

One limitation of our study is the consideration of only one Python implementation. Therefore, we suggest comparing the presented results with those of other Python scripts or implementations in other programming environments (e.g., MATLAB or SCALA). It can also be used to compare the performance with that of other algorithms. In addition, deeper investigation of efficient data structures may be useful to improve computation time and memory consumption, which may be relevant especially for very large facility layout problems.

REFERENCES

[1] M. Al-Haidary, M. A. Ajlouni, M. A. Talib, S. Abbas, Q. Nasir, and E. Basaeed. 2021. Metaheuristic Approaches to Facility Location Problems: A Systematic Review. In 2021 4th International Conference on Signal Processing and Information Security (ICSPIS) (pp. 49-52). IEEE.

[2] S. Q. D. Al-Zubaidi, G. Fantoni, and F. Failli. 2021. Analysis of drivers for solving facility layout problems: A Literature review. Journal of Industrial Information Integration, 21, 100187.

[3] D. Bertsimas and J. Tsitsiklis. 1993. Simulated Annealing. Stat. Sci. 8, 1, doi: 10.1214/ss/1177011077.
[4] P. Burggräf, J. Wagner, and B. Heinbach. 2021. Bibliometric Study on the Use of Machine Learning as Resolution Technique for Facility Layout Problems. IEEE Access 9, pp. 22569–22586, doi: 10.1109/ACCESS.2021.3054563.
[5] E. K. Burke and G. Kendall, Eds. 2014. Search Methodologies. Boston, MA: Springer US. doi: 10.1007/978-1-4614-6940-7.
[6] H. Fangohr. 2004. A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. In Computational Science - ICCS 2004, vol. 3039, M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1210–1217. doi: 10.1007/978-3-540-25944-2_157.
[7] H. Hosseini-Nasab, S. Fereidouni, S. M. T. Fatemi Ghomi, and M. B. Fakhrzad. 2018. Classification of facility layout problems: a review study. The International Journal of Advanced Manufacturing Technology, 94, 957-977.
[8] kevingeorge0123, Facility Layout Simulated Anealing. 2022. Accessed: Jun. 03, 2022. [Online]. Available: https://github.com/kevingeorge0123/Facility-Layout_Simulated-Anealing
[9] J. Kouatchou. 2020. Basic Comparison of Various Computing Languages. Zenodo, doi: 10.5281/ZENODO.3675797.
[10] G. Moslemipour, T. S. Lee, and D. Rilling. 2012. A review of intelligent approaches for designing dynamic and robust layouts in flexible manufacturing systems. Int. J. Adv. Manuf. Technol. 60, 1–4, 11–27, doi: 10.1007/s00170-011-3614-x.
[11] J. Orizet. These are the most popular programming languages 2020; German: 'Das sind die beliebtesten Programmiersprachen 2020,"' Netzwoche, May 29, 2020. https://www.netzwoche.ch/news/2020-05-29/das-sind-die-beliebtesten-programmiersprachen-2020 (accessed Jun. 03, 2022).
[12] P. Pérez-Gosende, J. Mula, and M. Díaz-Madroñero. 2021. Facility layout planning. An extended literature review. International Journal of Production Research, 59(12), 3777-3816.
[13] R. K. Phanden, H. I. Demir, and R. D. Gupta. 2018. Application of genetic algorithm and variable neighborhood search to solve the facility layout planning problem in job shop production system. In 2018 7th International Conference on Industrial Technology and Management (ICITM), pp. 270–274. doi: 10.1109/ICITM.2018.8333959.
[14] L. S. Vailshery. Python - Statistics & Facts. Statista. https://www.statista.com/topics/9361/python/ (accessed Jun. 03, 2022).
[15] Stack Overflow. Most used languages among software developers globally 2021. Statista. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ (accessed Jun. 03, 2022).
[16] W.-C. Yeh, C.-M. Lai, H.-Y. Ting, Y. Jiang, and H.-P. Huang. 2017. Solving single row facility layout problem with simplified swarm optimization. In 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 267–270. doi: 10.1109/FSKD.2017.8393199.
[17] T. Zhu, J. Balakrishnan, and C. H. Cheng. 2018. Recent advances in dynamic facility layout research. INFOR: Information systems and operational research, 56(4), 428-456.