A decision support system with 3D visualization for box space optimization using the locust-based particle swarm optimization algorithm

To cite this article: C Janea et al 2019 IOP Conf. Ser.: Mater. Sci. Eng. 482 012048

View the article online for updates and enhancements.
A decision support system with 3D visualization for box space optimization using the locust-based particle swarm optimization algorithm

C Janae¹, B L Alcisto² and C Maderazo³

Department of Computer and Information Sciences, School of Arts and Sciences, University of San Carlos, Cebu City, Philippines

¹ janeacarlo@gmail.com
² bojoluis21798@gmail.com
³ cvmaderazo@usc.edu.ph

Abstract. The non-stop increase in world population has resulted to an increase of the demand of space for human habitability. Space allocation is a problem mostly dealt with those with limited to scarce resources, and those who cannot afford or are not willing to spend for an alternative or an additional space as a solution. The objective, then, is to aid people in maximizing their current available space without having to sacrifice cost, time, and effort. This study has developed a decision support system that provides the optimum arrangement between a set of objects, selected by the user from sample models from the ShapeNet dataset presented by the system through 3D Visualization, to maximize the space of the given box. The developed system obtained the optimal arrangement, which maximized 60% to 91% space of the given box, of the set of objects by utilizing the locust-based Particle Swarm Optimization algorithm.

1. Introduction
Space is a resource that few disregard as important but is actually critical with the limited resources made available by the environment. Space allocation is a problem dealt with those with limited to scarce resources, and those who cannot afford or are not willing to spend for an alternative or an additional space as a solution. This problem comes into play when objects within a given space is misarranged and then take more space than they are intended to, which then cause the person to sacrifice cost, time, and effort in the solution of finding more space. This study proposes a decision support system, aided by 3D Visualization, which provides the optimum arrangement between a set of objects from sample models from the ShapeNet dataset, to maximize the space within a given box by utilizing the locust-based Particle Swarm Optimization algorithm.

2. Related works
These are the past literatures that were utilized by the researchers in the creation of this study.

2.1. Space optimization
The problem of space optimization is not new in the computing field, as there have been many studies made to solve different variations and applications of such problem such as the research of [1] regarding the use of a greedy algorithm paired up with a tabu search algorithm in solving the office
space allocation problem which is to optimally arrange entities in an office. The problem of space optimization can also be linked up with the well-known knapsack problem due to its spatial scope. The knapsack problem was solved by [2] through a two-step ants algorithm, which is a swarm-based algorithm, where the approach started by making use of an anti-pheromone technique that was capable of warning ants that this specific food is not a good choice to include in the knapsack, and then, applying the ant colony optimization algorithm which does the decision making on what objects to put in the knapsack after such filtering by the anti-pheromone technique.

2.2. Locust-based Particle Swarm Optimization architecture
The locust-based particle swarm optimization architecture was made by [3] with the goal of solving combinatorial optimization problems with a general architecture. According to their model, the algorithm is divided into four different stages which mirrors on how locust swarms are formed during an attack – Identification, Verification, Analysis, and Solution.

Identification dwells on the swarm’s initialization and the corresponding parameters such as dimensions, number of iterations, the size of search space, and etc. Verification is checking the variation of the data compared to the last iteration and/or the current best solution of the swarm. This also includes updating the overall best for the entire group whenever possible. Analysis is saving the best solutions to a separate location for comparison for future iterations. Solution is the stage where the distribution of the best solution towards the problem happens.

This general architecture has been adapted to several combinatorial optimization problems such as the knapsack problem, traveling salesman, cellular robotic systems, and routing telecommunication networks due to its flexibility and usability.

2.3. Improving the Particle Swarm Optimization algorithm
The hyper parameters – such as the number of particles, the cognitive and social constants, the inertia constant, and etc. – of most optimization algorithms like the Particle Swarm Optimization (PSO) affect the solutions and result of the optimization significantly as these factors dictate the behavior and the interaction between particles within the swarm, and even the swarm itself.

Automatic tuning of parameters was used by [4] to obtain the ideal preliminary parameters for the Particle Swarm Optimization. They discovered that having a relatively larger population, contrast to standard PSO parameters, does improve the quality of the solutions given by the algorithm as tested by the researchers of the said study. Table 1 shows the obtained parameter changes from the study made by [4].

| Table 1. Tuned parameters for PSO by [4]. |
|-----------------------------------------|
| Parameter | Standard PSO | Meta-optimized PSO |
| Population Size | 40 | 246 |
| Inertia Factor (w) | 1 | 0.687378 |
| Cognitive constant (c₁) | 0.5 + ln(2) | 0.246448 |
| Social constant (c₂) | 0.5 + ln(2) | 0.701503 |

The performance of the standard PSO was improved by [5] through determining a stop condition for the swarm iterations on the basis of the swarm behavior during convergence. This allows the algorithm to put a dynamic limit on the swarm iterations and not exhaust the system for the same quality of solutions.

The quality of the solutions found by the swarm was enhanced by [6] through a stagnation detection method wherein the swarm’s search process is restarted when the fitness of the solutions stagnate. This allows the swarm to potentially avoid local optima within the search space.
3. Proposed method

In this section, the conceptual framework of the proposed decision support system is discussed. The summary of the conceptual framework is shown in figure 1.

![Figure 1. Development approach for the proposed decision support system.](image)

3.1. ShapeNet object model dataset

The system uses object model data from ShapeNet, a collaborative effort by Princeton, Stanford, and TTIC that provides large-scale and defined datasets for researches in computer graphics, computer vision, robotics, and other fields related. The researchers then filtered out object model data, from the said dataset, that had incomplete fields in their metadata, and those that were relatively larger than the large-sized moving box (18 x 18 x 24 inches).

3.2. Object model visualization

The system visualizes a sample handpicked by the researchers from the filtered dataset through Panda3D so the user can proceed in selecting from the visualized sample.

3.3. User intervention

The user can now then select a set of objects from this visualized sample and also set the dimensions of the box in inches whose space will be optimized. These dimensions are used in constructing the box, represented as a three dimensional array, for the algorithm to work on. With array’s index having to only handle discrete numbers, the system converts the given dimensions to millimetre and disregards the remaining decimal points to minimize inaccuracy of solutions.

3.4. Locust-based Particle Swarm Optimization

The system receives the set of objects, with their nearest bounding box dimensions, selected by the user and the box-container dimensions and then proceeds with the optimization with the locust-based particle swarm optimization algorithm. This optimization generates an array of inserted objects and an array of coordinates – a pair between object and best location within the box.

| Table 2. Stages in solving the box space optimization based on the architecture by [3]. |
|-------------------------------|-------------------------------|-------------------------------|-------------------------------|
| **Locust-based Particle Swarm Optimization** | **Identification** | **Verification** | **Analysis** | **Solution** |
| Initialization of parameters | Check number of free space around object | Analysis of locations (coordinates) with smaller free spaces around it | Insert object to the best location with minimum free space around the object |
| Determine problem dimensions | Check box’s available space | Store location (coordinates) with minimum free space | |
| Object combination | | | |
3.4.1. Finding the best combination of objects. Before the system determines the optimal arrangement of objects within the box, it first finds the best combination of objects that will occupy the most space within the box through a simple greedy algorithm.

3.4.2. Finding the best arrangement. After determining what combination of objects to insert, the locust-based Particle Swarm Optimization algorithm, with the same parameters as [4], then comes into play. The basis of the system to determine which location and what position of the object is optimal is through calculating the free space around the object. The system considers 6 rotations of the object which is shown in figure 2 and calculates the free space around the object per rotation for each location, and since all objects are treated as rectangles (nearest bounding box) the system counts the free space within the 6 sides per object.

![Figure 2. Six rotations of the object covered by the system.](image)

The lesser the free space around the object of the given location and the specific rotation, the more likely the swarm will choose it as the solution. The objective function whose goal is to find the optimal location with the least amount of free space around the object can then be stated as

$$\min f(x) = \sum_{i=1}^{6} \sum x \in p_i * q_i * r,$$

where \(i\) refers to the sides of the object’s bounding box, \(p_i\) and \(q_i\) are the length of the two edges of the \(i^{th}\) side being operated on, \(r\) is the distance either between the \(i^{th}\) side and the corresponding border of the box-container opposite to the object’s \(i^{th}\) side, or between the \(i^{th}\) side and the nearest opposite bounding box side of another inserted object, and \(x\) is the free cell found within the volume of the rectangular prism calculated by \(p_i * q_i * r\). After the swarm finishes the analysis between the solutions, the best solution is then selected, thus, inserting the object within the box through updating the three-dimensional array representation of the box-container and the list of inserted objects with their corresponding coordinate locations. The four stages of the locust-based Particle Swarm Optimization algorithm used by the system are shown in table 2.

3.4.3. Finding the percentage of optimized box space. After an object is inserted, the system checks the box-container’s optimized space through

$$\text{box\_percentage} = \frac{\sum_{i=0}^{n} v_i}{t} * 100,$$

where \(n\) is the number of objects inserted in the box, \(v\) is the solid volume of the object’s bounding box, and \(t\) is the total volume of the box-container. If the calculated percentage is already at its maximum (100%) then the system stops any further insertion since the box-container’s space are already fully occupied, but if the space of the given box isn’t fully optimized, the process resumes with section 3.4.2 until the box space is fully maximized or until no object in the given combination can be further inserted.

This shows that the system supports the possibility of having to insert only one object, given that it occupies the entire box space or somewhere near 100%, since the proposed system focuses on
maximizing box space, not the number of inserted objects. This also shows that if the given combination’s total volume doesn’t sum up to the total box-container’s volume, there is no way the system can maximize the box-container’s space to 100%.

3.5. Visualization of result
The system then passes the array of inserted objects and the array of coordinates to Panda3D for the visualization of the optimal arrangement as a result from the optimization. The system can also notify the user when there is not enough space to insert an object/s or when the box is too small to fit anything inside it.

4. Results

![Figure 3. Visualization of results (best) using Panda3D for (a) objects with varying dimensions and (b) objects with uniform dimensions.](image)

To test the effectiveness of the proposed methodology, the algorithm was tested on two kinds of object combinations – the first five combinations were composed of objects with varying dimensions, and the second five combinations were composed of objects with uniform dimensions. Both combinations totalled or exceeded the given box volume and was ran 5 times per combination with the same initial parameters for the swarm by [4]. For each run, the percentage of the box space optimization, calculated by equation 2, was saved and was averaged when the 5 runs were completed. The results of the tests are shown in table 3 and figure 3.

| Combination # | Average Space Optimized (%) | Best Space Optimized (%) |
|---------------|-----------------------------|--------------------------|
| 1<sup>a</sup> | 68.8220                     | 72.4241                  |
| 2<sup>a</sup> | 65.8629                     | 68.8544                  |
| 3<sup>a</sup> | 69.8358                     | 72.4614                  |
| 4<sup>a</sup> | 69.3207                     | 77.7381                  |
| 5<sup>a</sup> | 60.8461                     | 72.1862                  |
| 6<sup>b</sup> | 78.7142                     | 81.0294                  |
| 7<sup>b</sup> | 84.8720                     | 88.9788                  |
| 8<sup>b</sup> | 72.3733                     | 79.1583                  |
| 9<sup>b</sup> | 75.7440                     | 77.7728                  |
| 10<sup>b</sup> | 90.2290                    | 91.4008                  |

<sup>a</sup>Combination composed of objects with varying dimensions
<sup>b</sup>Combination composed of objects with uniform dimensions

The results show that the first five combinations that were comprised of objects with varying dimensions were optimized only by 77% at best. This is mainly caused by the locusts converging to local optima since the objective function used only takes into account the free space around a current
object. It should take into account the effect of inserting such object in that position to the rest of the objects to be inserted in the future in order for it to peak the solution towards the global optima. The second half of the test, which was comprised of objects with uniform dimensions, show promising results as it peaked to 91% space optimization. This is possible because having to deal with objects that have uniform dimensions, the swarm would not have to worry about how the position of the current object would affect the future objects to be inserted.

5. Conclusion and recommendations

The use of the simple greedy algorithm combined with the locust-based PSO algorithm showed a 60% to 77% space optimization for object combinations with varying dimensions and 72% to 91% space optimization for object combinations with uniform dimensions as seen from the results from table 3. Although having objects with varying dimensions resulted to a low space optimization, the cause was determined and was due to the used objective function in the locust-based algorithm. This was proven by testing the algorithm on objects with uniform dimensions, as the objective function would have little to no effect on the solutions, granted 91% space optimization, at best.

This research can be further improved by utilizing a different data structure to represent the box where the swarm can operate on since a 3D array showed inaccuracy and inefficiency for locating the optimal solution per object. Another improvement is by using a more exclusive objective function that considers not only the optimal space for an object, but also what object is the most optimal to insert at that specific location and iteration. This will allow the algorithm to peak its solution towards the global maxima. Another possible improvement towards this study is considering more test cases during testing as the researchers showed only the minimal combinations that prove that the algorithm can perform its objective. Lastly, to improve the realism of the research, the researchers recommend to operate on object’s real dimensions, not just its bounding box, and to handle containers which allow insertion of one object inside another object.

References

[1] Castillo F, Riff M-C and Montero E 2016 New bounds for office space allocation using tabu search Proc. of the Genetic and Evolutionary Computation Conf. 2016 (Denver, Colorado)

[2] Rojas N, Montero E and Riff M-C 2015 Using anti-pheromone to identify core objects for multidimensional knapsack problems: a two-step ants based approach Proc. of the Companion Publication of the 2015 Annual Conf. on Genetic and Evolutionary Computation (Madrid, Spain)

[3] Ribiero T and Paucar V P 2017 Problem solving in combinatorial optimization based on the algorithmic model of attack behavior of locust swarm Proc. of the 2nd Int. Conf. on Internet of Things, Data and Cloud Computing (Cambridge, United Kingdom)

[4] Ugolotti R and Cagnoni S 2015 Automatic tuning of standard pso versions Proc. of the Companion Publication of the 2015 Annual Conf. on Genetic and Evolutionary Computation (Madrid, Spain)

[5] Perroni P F, Weingaertner D and Delgado M R 2017 Estimating stop conditions of swarm based stochastic metaheuristic algorithms Proc. of the Genetic and Evolutionary Computation Conf. (Berlin, Germany)

[6] Ahmed H R 2014 An efficient fitness-based stagnation detection method for particle swarm optimization Proc. of the Companion Publication of the 2014 Annual Conf. on Genetic and Evolutionary Computation (Vancouver, BC)

Acknowledgements

The researchers would like to thank the Department of Computer and Information Sciences of the University of San Carlos for the research opportunity and for the knowledge and guidance during the said study.