# Optimizing Open Space for Furniture Layout using Quantized Firefly Algorithm

Randall Fowler and Conor King

***Abstract*—Abstract goes here...**

***Index Terms*—Firefly Algorithm, Discrete Optimization, Contour Mapping**

## I. Introduction

The introduction goes here...

## II. Methodology

### A. Firefly Algorithm

The original FA mimics the behavior of tropical fireflies. Each firefly will represent a solution, and a uniform distribution will be used to initialize the location of each firefly throughout the solution space. This follows a similar method to the particle swarm optimization (PSO) algorithm, but there is a major difference between FA and PSO [1]. While particles in PSO will move in the direction of a local better solution, fireflies expand upon this by moving toward the current global best solution. Movement toward the global best solution is handled by shifting fireflies toward the brightest firefly with respect to distance.

The FA is designed for continuous optimization with a maximization objective defined as

$$Maximize\ \ f(x), x \in \mathbb{R}^D. \tag{1}$$

It is an unconstrained problem that requires constraints to be added during movement of the fireflies. For a discrete optimization problem, such as the problem described in this paper, new solutions formed after movement can be quantized to integer values.

Movement of these fireflies is defined by an equation that weights distance similar to absorption of light intensity. Calculation of movement is defined as

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{i,j}^2}(x_j^t - x_i^t) + \alpha \epsilon_i^t, \tag{2}$$

where $x_i^t$ represents the location of the $i$th firefly in the $t$th iteration. While the distance between fireflies is a main component in the equation, the weighting is known as the attractiveness. Hyperparameters for this attractiveness term would be $\beta_0$, the attractiveness at zero distance, and $\gamma$, the absorption coefficient. Distance between solutions is calculated using Cartesian distances, and $r_{i,j}$ would be

$$r_{i,j} = \sqrt{\sum_{k=1}^{D}(x_{i,k} - x_{j,k})^2}, \tag{3}$$

an $l2$ norm of the two fireflies. This attractiveness term will reduce movement between fireflies if the distance is relatively large, and this will enable movement toward the global best solution while encouraging more movement toward local bests.

The final term in the movement equation 2 represents the exploration or noisy movement of the fireflies. While the vector $\epsilon_i^t$ will contain random numbers following a normal distribution with zero mean and unity variance, the hyperparameter $\alpha$ will be a scaling factor. Both $\beta_0$ and $\alpha$ will depend greatly on the scaling of the solution vector.

Another parameter in FA would be the time allowed for movement. With FA being a search algorithm to reduce the need for exhaustive searching. Algorithm 1 describes the pseudocode for FA. For the specified amount of time, the algorithm will iterate over all fireflies, check if the value of other fireflies is greater, and move if so. After every firefly has been visited, the known best will be saved.

**Algorithm 1** Firefly Algorithm

**Input:** Number of fireflies $N$, number of iterations $T$
**Output:** Best solution found

```
Initialize fireflies x_i, i = 1, 2, ..., N.
Evaluate the objective function f(x_i) for each firefly.
Set time t = 0.
while t < T do
  for i = 1 to N do
    for j = 1 to N do
      if f(x_j) > f(x_i) then
        Move firefly i toward j using equation 2.
        Evaluate the objective function f(x_i).
      end if
    end for
  end for
  Sort fireflies by objective function and update the best solution.
  t = t + 1.
end while
```

### B. Defining Objects in the Room

To reduce the complexity of objects shape, location, and orientation, the optimization of the room layout is defined as a discrete problem with rectangular objects. Each object will have a width, depth, rotation, reserved space, and a unique identification (uid). Rotation is defined by an enumeration of up, right, down, and left with values of 0, 1, 2, and 3 respectively. For values larger or smaller than the enumerated values, the value will be shifted to the respective values, 4 would be equivalent to 0, and -1 would be equivalent to 3.

The names of the enumeration can be interchangeable with north, east, south, and west. Reserve space will be considered to be the space in front of the object from wherever it is facing. When something is facing up, the reserve space of the object will be above the object.

Width and depth are the size parameters defining the size by orientation. If an object is facing up or down, the width will represent the distance in the x direction while depth would be in the y direction. This will be the opposite case for facing left and right as depth will be in the x direction and width will be in the y direction. Reserve space will be the distance in front of the object that is reserved open space, and the width of the reserved space will be the same as the width of the object. The purpose of the reserved space is to allow access to the object; most objects, such as a desk, door, or bookshelf, would require space in front of it for someone to stand or sit. Objects that need reserved space in more than one direction are not considered in this paper, but it wouldn't be a large change to add it.

For the uid, objects will have an identification equal to a prime number larger than 1. The reason for this is due to the nature of solving conflicts when moving objects. An open space would be given a value of 1 and adding an object would be to multiply the rectangular area by its uid. If a point in the graph has a non-prime number, this will be an indication of an overlap between objects.

### C. Defining the Room

A room will be defined simply as its width, height, and graph. The graph will be defined as an array with relative units shared between the sizing of the objects and room. This graph will hold the value 1 for being an open space and a value larger than one is an object taking up that space. For the uid graph, the reserved space will be included in the graph, but for open space graph, the reserved space will not be included. Since reserved space will be open, it will be considered for the evaluation of the layout.

### D. Solution Vector

Initialization of fireflies requires a uniform distribution amongst the solution space. Each firefly will have an associated solution vector holding the location and rotation of every object in the room.

The dimensions of the solution vector will be defined by the number of objects in the room, and the order of these objects will be by shape and distance from a reference point. This reference point is chosen to be the origin, bottom left corner of the room. One issue that arises from having a set index for the solution vector is that multiple solutions could be the same due to objects having the same shape. This shape is defined by its width, depth, and reserved space. Indexing of the solution vector will be iterated by the different shapes, and for each shape, there will be an indexing in the order of distance to the origin.

Some objects, such as doors and windows, are not moveable or rotatable in the room. For these objects and others that may not want to be moved, these will be left out of the solution vector. However, if a shape is moveable, but not rotatable, it will be included in the solution vector without a rotation variable and as its own shape. Any additional objects with the same shape and without rotation will be grouped together in the solution vector, similar to shapes with rotation.

Random placement of objects within the room will be bound to the size of the room. Coordinates and rotation will be gathered from a uniform distribution, and if the object does not fit in the room, a new coordinate and rotation will be gathered. This technique for generating initial solutions may require more time for more objects as objects will often fail when fitting.

## III. Design Decisions

### A. Objective Function

The purpose of this project is to design an optimal layout for a room with known objects. However, this is a challenging problem to address as optimizing the layout of a room is biased toward the opinion of the user. A common preference for individuals when designing their room is to have the largest amount of open space possible. With a set number of objects, the amount of open space would be the same for every possible room arrangement, but maximizing open space for this project holds a different definition. The objective of maximizing open space consists of maximizing the roundness of open areas. This means that larger open spaces would be preferred, but stretched open space would be less preferable than more circular open spaces; A large rectangular open space would be preferred over a long rectangle with the same area.

At first, this objective function was thought to be its own optimization problem where largest ratio of area to perimeter would be useful for estimating value of the room. However, another method emerged to try and satisfy the objective function. A pseudo contour map of the room could be created to find the distance between objects and maximize on spreading objects around the room. For less objects, the optimal solution could be to push everything against the walls, but for a larger number of objects, the layout would need to be creative about placement.

A contour map for the room would be an estimate of distance between objects at every point in the room. Objects themselves would have a value of 0 since that is the location of an object, and the walls could be either seen as an object or not. For this project, it was decided to treat the walls as an object, but exploring the other option could be insightful.

The first idea of a contour map thought of would be known as open distance, and this represents the product of width and height of the open space at a given position of the graph. This algorithm is simply created by iterating over all positions of the graph and getting the width by moving up and down till hitting a wall or object, and the height similarly with left and right. Results for this method are shown in figure 1, and it is difficult to spot where the objects actually are.

The second idea of a contour map was to use Euclidean distance to the nearest object or wall. This algorithm would

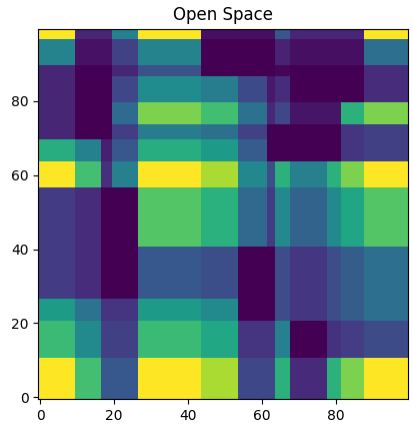


Fig. 1. Open Distance Contour Map

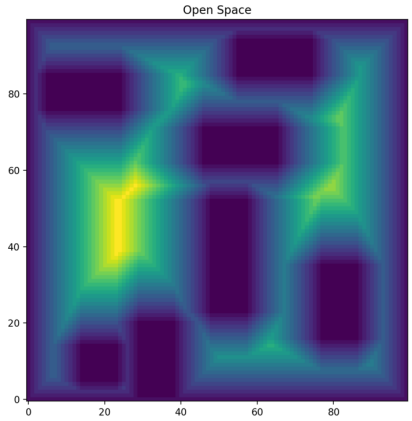


Fig. 2. Taxicab Distance Contour Map

require a good search method that could keep its searching distance to under the distance to the nearest wall. However, Taxicab geometric distance would provide a similar contour map, but the algorithm could be greatly improved.

Algorithm 2 shows the method for calculating distances. All points would be initialized as a number larger than possible, objects would be added as 0 values, and the search would begin by adding all nodes with 0 value to a list of nodes. While there are nodes in the list, each node will be iterated through, and if a neighbor has a larger value than the current node, the neighbor will be given a value of one larger than its own. After iterating through all nodes, it will then add all nodes with a value of one larger than the nodes it just visited. Eventually, all points in the map will have a distance. Results of this algorithm can be seen in figure 2.

**Algorithm 2** Taxicab Distance Contour Map

**Input:** Room with objects
**Output:** Contour map of room
  Initialize map to height or width of room.
  Add all objects as 0 values.
  Add all 0 value nodes to a list of nodes.
  **while** There are nodes in the list **do**
    **for** Each node in the list **do**
      **for** Each neighbor of the node **do**
        **if** Neighbor has a larger value than the current node **then**
          Neighbor is given a value of one larger than the current node.
        **end if**
      **end for**
    **end for**
    Add all nodes with a value of one larger than the nodes it just visited to the list.
  **end while**

Once the contour map has been found, evaluation of the map will provide the objective function. This value can be found by summing the entire contour map. Larger values will be calculated if there is more space between objects, and the calculated values could be normalized using the size of the room.

### *B. Types of Conflicts*

As objects begin to move in the room, there will be conflicts emerging from objects trying to move past the boundaries of the room and overlapping objects. For locations that conflict with the boundaries of the room, the object coordinates will be truncated and set to the position closest to the wall. However, overlapping object is a larger consider with a need for resolve to have a valid solution. Figure 3 displays an example of the algorithm starting with one room in 3a, moving towards another room in 3d, conflicting with overlapping objects in 3b, and ending in a new position 3c.

There are 3 known types of conflicts that relate to overlapping objects:

1) One object didn't move or rotate.
2) One object didn't move but rotated.
3) All objects moved.

The first type of conflict is the easiest to resolve as the object that didn't move will be left in its original position. However, it's debatable how to handle the other two conflicts. Some ideas would relate to filling the overlapped position using a random distribution with more weighting given to the objects that didn't move as far. This would require a bit more computation and could provide new conflicts with other objects along the movement path. Another idea would be to apply noise to the objects that have conflicts. The next section discusses this idea where noise is applied to all objects simultaneously.

### *C. Noise Separation*

To resolve conflicts, one technique would be to iteratively increase noise applied to an object's location till there is no longer a conflict. Algorithm 3 demonstrates the proposed noise separation algorithm to follow this procedure. Figure 4 contains the results of each iteration for the same example shown in figure 3. The algorithm begins by creating a list of

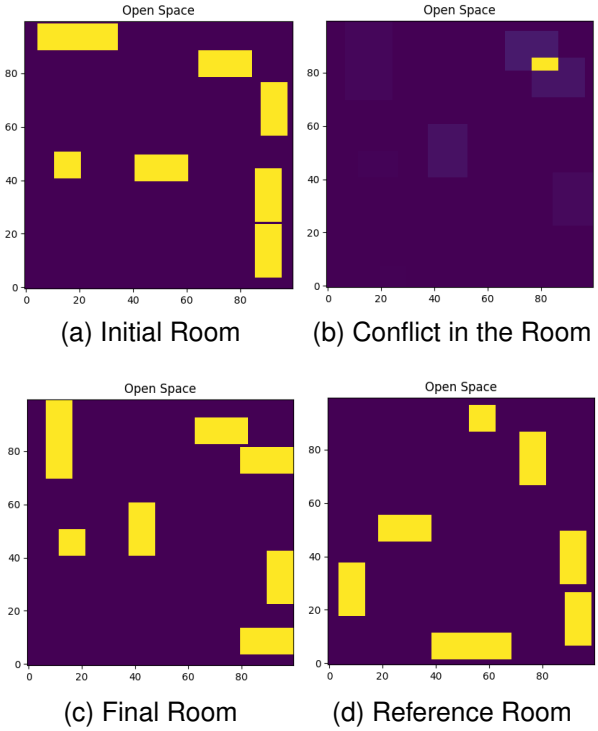


Fig. 3. Moving Objects in a Room

conflicts and removing all of them as shown in 4a. Objects that didn't move are not considered as conflicting objects and remain in the plot. All conflicts in the list will be added back to the plot with noisy new positions. If the new positions fail, the amount of noise will increase and try again, and this can be seen through three iterations between figures 4b to 4d.

Once an object resolves its position, it will be given that position and the algorithm with check for remaining conflicts for other objects. It will no longer be receiving noise for a new position once a conflict has been resolved. Figure 4d shows the result of the noise separation with no overlapping objects.

### D. Hyperparameters

The new solution vector, calculated from fireflies moving toward each other, will contain the locations for objects to move to and updated rotations. Since the rotation has a much smaller set of possible numbers than the coordinates, the hyperparameters $\alpha$ and $\beta_0$ may need to be defined differently. These parameters can be treated as a vector with a value relating to position and a value relating to rotation. To get started with testing, $\alpha$ and $\beta_0$ were defined as 0.2 and 0.5 for rotation respectively, and these parameters for coordinates were 2 and 5 respectively. Ideally, the movement equation should be adjusted for rotation as it may be preferable for the object to rotate more as it gets closer to the other fireflies but rotate less the further away. For now, the firefly movement remains unchanged for rotation, but there is a large exploration that can be done on the hyperparameter space.

**Algorithm 3** Noise Separation

**Input:** New uid space with conflicts
**Output:** New uid space without conflicts
  Initialize list of conflicts
  Remove conflicts from new uid space.
  Initialize noise parameter $\sigma$
  **while** There are conflicts **do**
    Initialize resolve space as the new uid space..
    **for** Each object with a conflict **do**
      Scale noise using $\sigma$.
      Apply noise to the objects position.
      Add object to the resolve space.
      **if** There is a conflict **then**
        Add conflict to new list of conflicts.
      **end if**
    **end for**
    Get resolved conflicts from difference between new conflicts and old conflicts.
    Add resolved conflicts to new uid space.
    Set new conflicts as the conflicts.
    Increment $\sigma$.
  **end while**

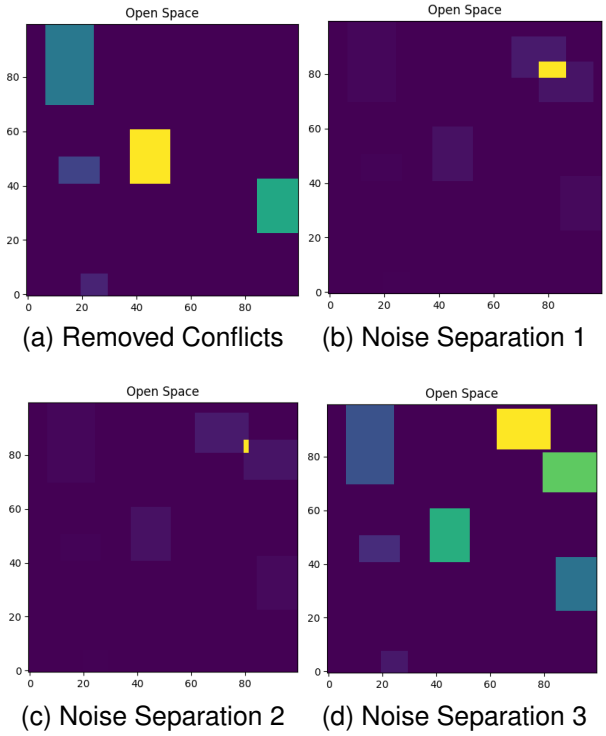


Fig. 4. Moving Objects in a Room

## IV. Experimental Tests

Experimental tests go here...

## V. Conclusions

Conclusions go here...

## ACKNOWLEDGMENT

Thankful for...

## REFERENCES

[1] Slowik, A. (Ed.). (2020). Swarm Intelligence Algorithms: A Tutorial (1st ed.). CRC Press. https://doi.org/10.1201/9780429422614
[2] Reference 2 goes here...