Optimizing Study Space in Odegaard Library

Hongyu Cao and Christina Stanfield

ABSTRACT

In this paper we present our method to arrange study furniture in Odegaard Undergraduate Library so as to maximize seated capacity while maintaining functional usability of the space. The arrangement of furniture within a large space in order to maximize aesthetics and functionality is a uniquely defined and difficult combinatorial optimization problem. Increasing study capacity in Odegaard library is important for combating pervasive overcrowding while ensuring efficient and accessible use of academic facilities. We employed the technique of simulated annealing to model and approximate a solution to this problem to optimality.

INTRODUCTION

Libraries are the foundation of academic life outside of the classroom. At the University of Washington, 16 different libraries see 5.6 million annual visitors[1], each offering a distinct breadth of specialized services to aid student's scholastic goals. Odegaard Undergraduate Library is one of the largest and most frequently trafficked on the Seattle campus with 1.6 million annual visitors supporting a Research and Writing Center, Learning Commons, and Active Learning Classrooms. As the admitted undergraduate classes grow successively larger each year, libraries face the challenge of meeting the pressing demand for more study space within their current facilities. Libraries will become significantly overcrowded without some kind of change.

Possible solutions to overcrowding could include major structural changes to the campus, such as building a new library or perhaps adding-on floors to the current library, or even reducing attendance to current libraries through deterrents. Yet these changes are drastic and expensive—too difficult to reasonably implement. In this paper we approach the issue of overcrowding in a different manner: changing the layout of existing study furniture within the existing facility to accommodate more students.

[1] "Libraries Fact Sheet 2018". https://www.lib.washington.edu/assessment/statistics/facts.

GOAL

Our objective is to find a pleasant floor plan which maximizes the amount of seating in Odegaard Library in such a way that each seated person would have enough space to comfortably study and all legal requirements are satisfied.

We define a *pleasant floor plan* as one with the following features: proximity to windows and proximity to power outlets. These enhancements will be discussed in further detail below.

ODEGAARD LIBRARY

We limit our study to the first floor of Odegaard library which contains various types of study spaces and classrooms.

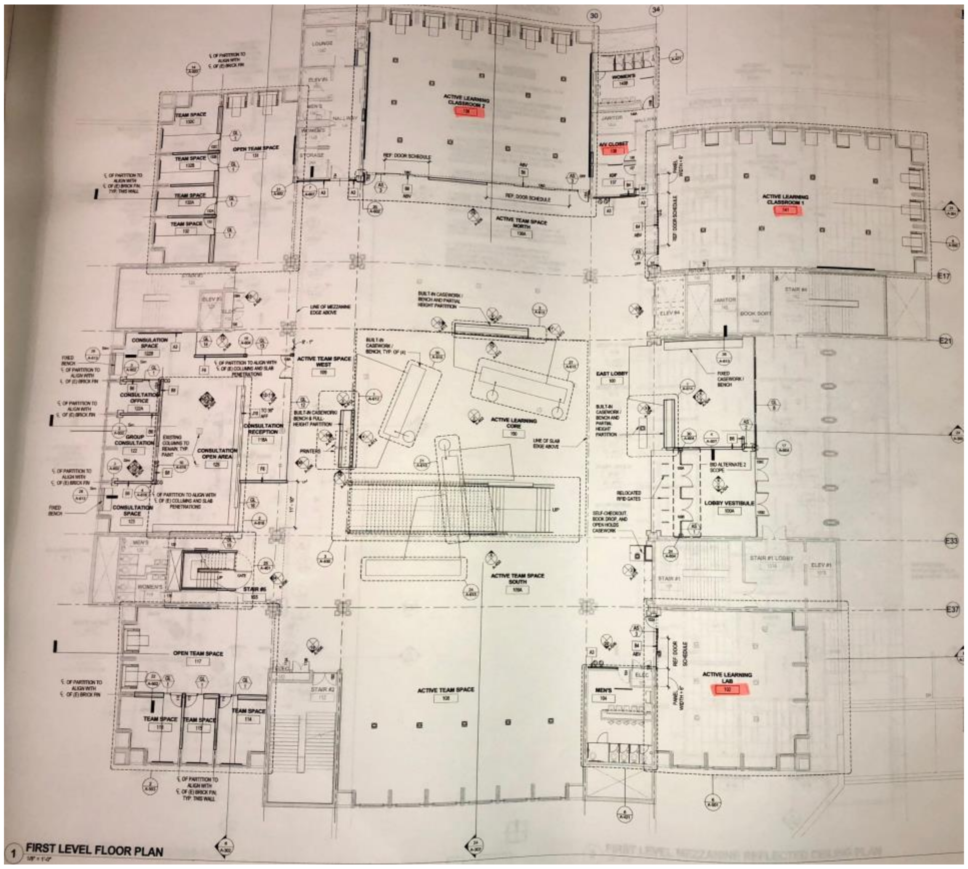


Figure 1: The first level floor plan for Odegaard library.

To simplify the problem, we consider Active Team Space 108 and Active Team Space North 136A as separate feasible study rooms. We create a system of room codes to differentiate each in our program; let Active Team Space 108 be room 1 and Active Team Space North 136A be room 2.

LAYOUT REQUIREMENTS

The floor plan must satisfy certain layout requirements to ensure student safety, comfort, and ease of accessibility. For our model relevant considerations include foot traffic pathways, minimum distance between tables, minimum distance between chairs, government fire escape and emergency regulations, and the maximum capacity of persons allowed on the first floor.

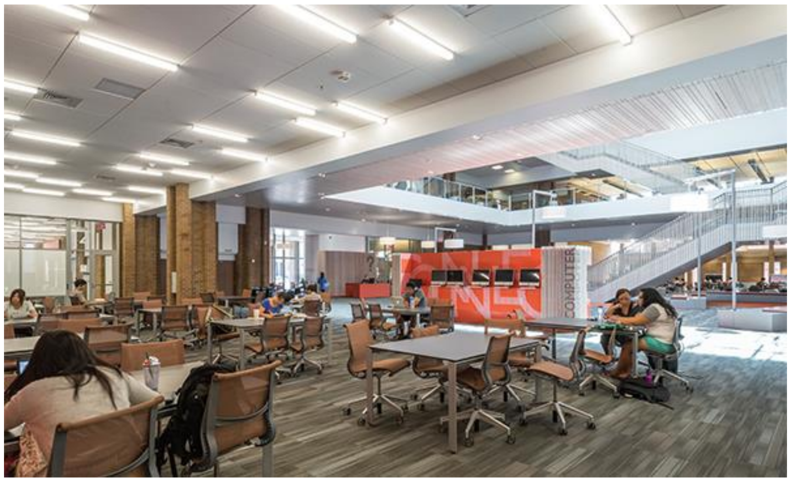


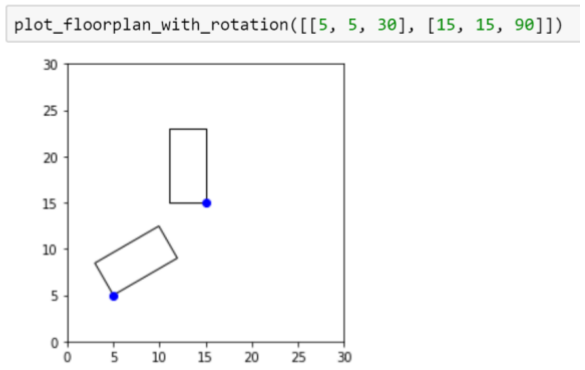
Figure 2: Active Team Space North 136A or 'room 2' of Odegaard library. Here we see the standard tables and chairs we can arrange in our model.

To create our pleasant floor plan, we consider the layout of study furniture in each room. We assume we have enough tables to fill the room and that these tables are the existing model in Odegaard measuring 1.5m x 1.05m and requiring 1.4m of padding on each side to comfortably accommodate chairs.[2] For this model, we assume each table can comfortably fit 4 chairs to seat 4 students. We consider no other types of furniture in these rooms.

A 3-dimensional array [x, y, r] is assigned to each table to model its arrangement, and a floorplan for a single room is represented by a list of those arrays, e.g. $[[x_1, y_1, r_1], [x_2, y_2, r_2]]$ represents a single room with 2 tables. Coordinates (x, y) indicates a table's bottom left point in a 2-dimensional coordinate system and r indicates the degree at which the table is rotated

[2] Brooks, Dillon. "How Much Space Between Tables And Chairs?". https://www.deltatables.com/2017/08/29/how-much-space-between-tables-and-chairs/

clockwise along its (x, y) point. To simplify the possible combinations of tables, let r ∈ {0, 30, 60, 90}.



```
plot_floorplan_with_rotation([[5, 5, 30], [15, 15, 90]])
```

Figure 3: Using our coordinate system, an example of a floorplan for a room with two tables: one at (5m, 5m) rotated at 30 degrees, another at (15m, 15m) rotated at 90 degrees.

PLEASANT FEATURES

In addition to minimum layout requirements, the floor plan should consider functional and aesthetic features that are visually enhancing and conducive to studying. We simplify the problem to only consider features that are contained within the scope of Odegaard library and can be finitely quantified for our mathematical model, including proximity to windows and proximity to power outlets.

– We calculate each table's distance to its closest window, and assign a score based on this distance. Tables closer to windows are considered more pleasant.
– We also calculate each table's distance to its nearest power outlet, and assign a score based on this distance. Tables closer to power outlets are considered more pleasant.

Together these generate our cost function, a measure of the "unpleasantness" of a given floorplan:

$$C = \sum_{i=0}^{n} \sum_{j=i+1}^{n} dist(state_i, state_j)$$

where

n = total number of tables

$state_i$ = the i^{th} table of a floorplan

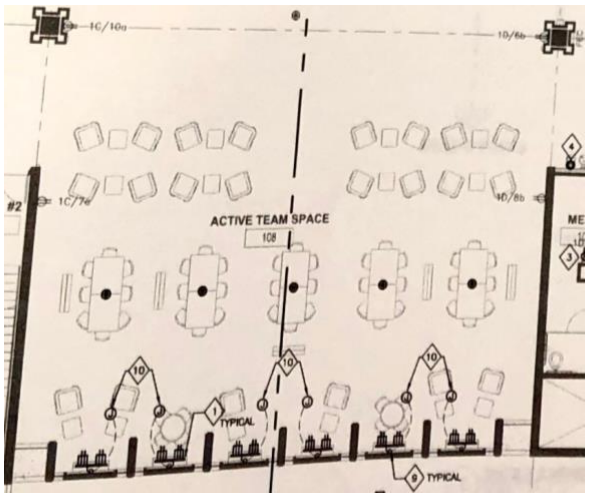


Figure 4: A section of the electrical floor plan of the first floor of Odegaard library showing the locations of power outlets (denoted by) in Active Team Space 108 or 'room 1'.

METHODS

We first considered a brute-force approach to generate a precise solution: enumerate all possible floor plans that do not violate constraints for a room, then evaluate each floorplan on its pleasantness. Then we simply choose which floorplan outputted the greatest pleasantness score as our solution.

Yet the problem of enumerating all possible floor plans for a room is NP-hard, and thus not computationally feasible. We therefore turn to approximation methods to generate a prediction of the most pleasant floor plan. Simulated annealing "is a probabilistic technique for approximating the global optimum of a given function; specifically, it is a metaheuristic to approximate the global optimum in a large search space used when the search space is discrete and the approximate global optimum is more important than finding a precise local optimum in a fixed amount of time."[3]

Simulated annealing originates from annealing in metallurgy, a technique "involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects." Wikipedia describes the algorithm as the following: given an initial state, at each step the simulated annealing heuristic considers some neighboring state s* of the current states, and probabilistically decides between moving the system to state s* or staying in-state s. The probability is based on some cost function it wishes to minimize and ultimately leads the system

[3] "Simulated Annealing." *Wikipedia: The Free Encyclopedia*, https://en.wikipedia.org/wiki/Simulated_annealing.

to move to states of lower energy, or lower cost. This step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.

For our purposes we wish to increase the number of tables while reducing the table's unpleasantness within a finite room. The program will start by generating a random feasible floor plan, then at each step will consider another pseudo-randomly chosen floor plan *f'* to compare to the current floor plan *f*. We decide whether to reject or keep *f'* based on its pleasantness score compared to *f*. We will repeat this process until we find a desirable floor plan or run out of computational space.

To implement this strategy, we modify and employ a Python module created by Matthew Perry publicly available on GitHub.[4] From this module we import its *Annealer* class that provides the base implementation of the simulated annealing algorithm. We also created our own *Rec* class that encapsulates information on a single table, its chairs, and padding, and contains a method *valid()* that returns true if its current state is within room boundaries and does not overlap other currently placed *Rec* objects. We then define two required methods for *Annealer*:

1. *move()* will be called at the beginning of each iteration. Here we generate another random floor plan *f'* by randomly selecting one *Rec* object from *f* and moving it to another random location and orientation within the room boundaries such that the object returns *valid() = true.*
2. *energy()* will be called after *move()* to define the cost, or total unpleasantness, of the newly generated floor plan *f'* compared to the previous floor plan *f*. If *energy(f') > energy(f)*, we reject *f'* and keep *f* as the most pleasant floor plan found so far.

COMPUTATIONAL RESULTS AND ANALYSIS

We ran our implementation of simulated annealing on each room for 10,000 iterations to obtain a reasonable local optimum. In the images below, each rectangle represents the safe area containing a table, its four chairs, and padding. For our first attempt at the algorithm our *valid()* method was incomplete since we were unable to find a computationally feasible way to determine if tables overlapped each other; we thus obtained the following results:

[4] Matthew Perry, https://github.com/perrygeo/simanneal.

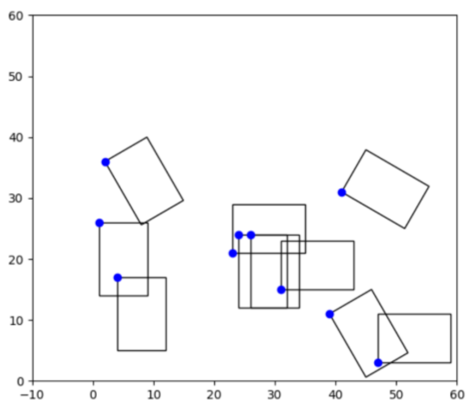


Figure 5: The plot of results obtained from running our first implementation of the simulated annealing algorithm on room 1.

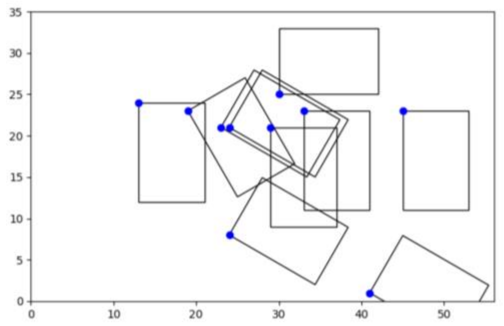


Figure 6: The plot of results obtained from running our first implementation of the simulated annealing algorithm on room 2.

The results show an immediate infeasibility: we cannot stack tables on top of each other. Yet they are revealing as each cluster of tables indicates the areas of the room that are most pleasant, the areas that have minimized both a table's distance to a window and distance to a power outlet. It is reasonable to expect a more relevant solution might first place tables in these clustered and desirable areas, then to branch out and place tables around these areas until it runs out of time or space.

We then were able to implement into *valid()* a means of checking whether a *Rec* object overlaps with any other *Rec* objects, drastically changing the outputted results. Running the same simulated annealing algorithm with a new *valid()* method reveals the following floor plan for room 1:

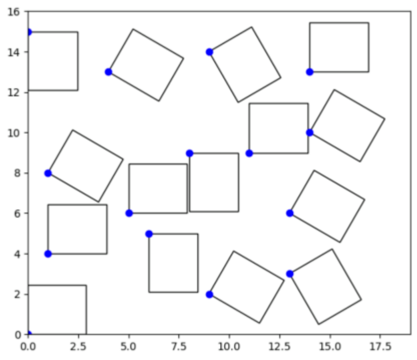


Figure 7: The plot of results obtained from running our second implementation of the simulated annealing algorithm on room 1 with improvements to the valid() method in the Rec class.

We can see from these results a much more feasible floor plan that comfortably seats 60 students. While the rectangles are no longer overlapping, the density of tables is perhaps sparser than desired as the rotation of the rectangles prevents additional tables from being placed more compactly. Additionally, the *move()* method of the simulated annealing algorithm moves the selected table at each iteration to a completely random location that satisfies *valid()*.

For our final implementation of the algorithm we chose to fix the rotation of the tables in each room, that is for a given room all *Rec* objects contained in that room have the same *r* rotation value. This will allow us to have a more uniform, dense, and aesthetically pleasing solution. We also modified the *move()* method to move the selected table at each iteration to a location based on the normal distribution as a function of the number of iterations. For example, for 'earlier' iterations say 0-1,000, the selected table has a higher likelihood of moving farther away from its current location than during 'later' iterations, say 9,000-10,000. Thus the Annealer will explore more 'extreme' solutions at the start, then begin to converge to a local optimum.

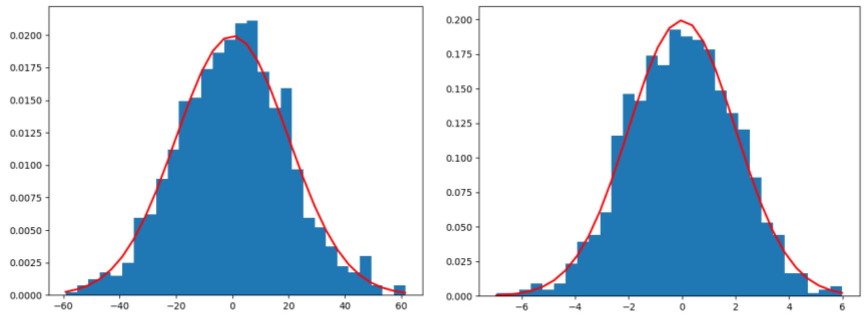


Figure 8: The normal distribution on the left was used by the Annealer for earlier iterations; the normal distribution on the right was used for later iterations. Comparing the two, we see from their x-axis values that a selected table is more likely to stay close to its origin when the move() method is called over successive iterations.

With these improvements, we ran our final implementation of the simulated annealing algorithm on room 1 and room 2 for 50,000 iterations, obtaining the following results. Again, note each rectangle on the plot represents a table, its chairs, and minimum padding.

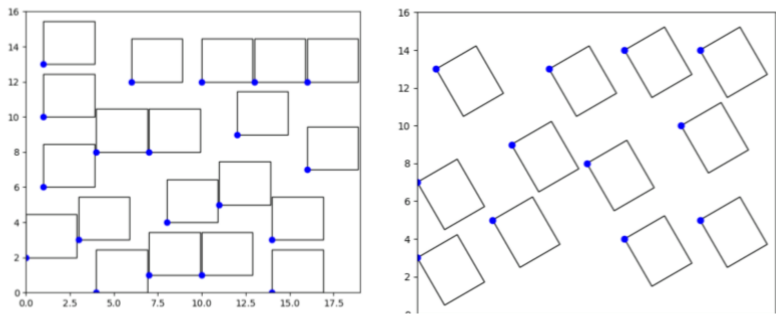


Figure 9: The plots of results obtained from running our final implementation of the simulated annealing algorithm on room 1. The plot on the left has a fixed rotation of r = 0; the plot on the right has a fixed rotation of r = 60.

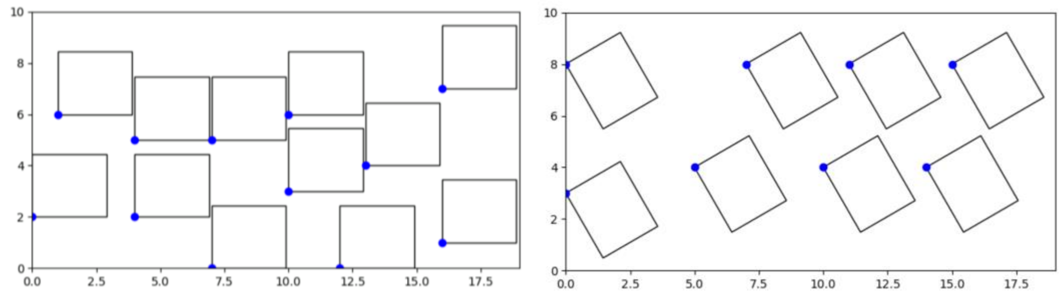


Figure 10: The plot of results obtained from running our final implementation of the simulated annealing algorithm on room 2. The plot on the left has a fixed rotation of r = 0; the plot on the right has a fixed rotation of r = 60.

We can see in Figure 9 the zero-degree rotation allows the algorithm to pack *Rec* objects more efficiently allowing for 20 tables comfortably seating 80 students. When the tables are rotated, we find a rotation of 60 degrees allows for a maximum of 12 tables comfortably seating 48 students. This is less than the zero-degree rotation, yet the tables are more evenly distributed and perhaps more visually appealing than the densely-packed plan. We observe the same pattern in room 2, where the zero-degree rotation allows for more tables but the 60 degree rotation outputs a more evenly-spaced layout.

Thus we observe a trade-off between these floorplans: more tables at the expense of higher cost or ‘unpleasantness’, or less tables with a comparatively lower cost. Considering our original goal, we must recommend choosing the floorplans with the most tables as this maximizes the number of seated students. This choice still ensures students are seated comfortably while optimizing accessibility to pleasant features.

IMPROVEMENTS

This model has many simplifying assumptions that could be expanded upon to provide additional complexity and solution accuracy. For example, we first lessened the scope of our problem to only two sections within Odegaard and considered those sections as uniform rectangles; it is possible to expand our inquiry into additional smaller areas within the first floor to generate a more complete floor plan that encompasses all possible study areas. It is also worthy to note a large section we omitted optimizing on the first floor of Odegaard does not contain our standard tables and chairs, rather it has immovable benches, small circular tables, and chairs meant for more relaxed seating. We would need to adjust our furniture parameters and personal space constraints to allow for better object pairing around immovable structural features. Adding this complexity to our program might also allow our model to generate floor plans for the 2nd and 3rd floors of Odegaard with less work so we could generate a complete optimized plan for the entire library.

A great deal of simplification also went into defining a pleasantness function to measure how functional and aesthetically pleasing a floor plan was. We only considered brute Euclidean distance measurements to windows and outlets as ways to score the pleasantness of a given table; more precise ways of measuring this may include scoring a table based on its *line of sight* to any given windows, and scoring outlets only if they are functional, i.e. not assigning a score for a table 40ft away from an outlet. We may also consider adding additional parameters to our pleasantness function that affect the quality of our floorplans, including measuring spatial symmetry, feng shui, lighting, and proximity to frequently trafficked areas.

CONCLUSIONS

Optimizing study furniture arrangements can have a conducive effect on the study environment and experience of students in the large first-floor Odegaard library space. In this paper we discussed our methods for generating such floor plans that not only optimize the floor layouts to increase available study space, but to do so in a way that studying is made more pleasant and comfortable. We found the simulated annealing algorithm to be the most relevant to our unique combinatoric optimization problem; this algorithm automatically generates feasible arrangements and scores them based on our pleasantness function. We have made incrementally better advances in how we define the methods of the algorithm, our objective function, and supporting classes to find local optimums that suit the needs of the library. With further study, it is possible to refine our methods and tackle improvements by adding additional complexity to create the ideal pleasant floor plan suiting the needs of students.

ACKNOWLEDGEMENTS
We acknowledge the warm support of John Danneker, Director of Odegaard Undergraduate Library, and Steve Weber, Manager of Access and Building Services, both of whom enthusiastically supported our project. We also gratefully acknowledge our instructor Sara Billey and Nico Courts whose help was invaluable in defining, modeling, and analyzing our project, and we thank our course peers for their constructive comments and ongoing feedback.

REFERENCES

Our source code is available at https://github.com/759340153/OnlineData/blob/master/test.py.

Brooks, Dillon. "How Much Space Between Tables And Chairs?". *Delta Tables*, 29 August 2017, https://www.deltatables.com/2017/08/29/how-much-space-between-tables-and-chairs/.

Thompson, Gary M. "Optimizing Restaurant-Table Configurations: Specifying Combinable Tables". *Cornell Hotel and Restaurant Administration Quarterly*, vol. 44, pp. 53-60, 2003, https://scholarship.sha.cornell.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1739&context=articles.

Yu, Lap-Fei, et. al. "Make it Home: Automatic Optimization of Furniture Arrangement." *ACM Transactions on Graphics*, vol. 30, no. 4, Article 86, 2011, http://doi.acm.org/10.1145/1964921.1964981.

"Simulated Annealing." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 27 October 2018. Web. 14 November 2018, https://en.wikipedia.org/wiki/Simulated_annealing.