Approaching the Floor Layout Problem with Genetic Algorithms and Convex Optimization

Name: Griffin Holt
Department of Electrical Engineering
Stanford University
gholt@stanford.edu

Name: Andrew Zhang
Department of Electrical Engineering
Stanford University
azhang82@stanford.edu

***Abstract*— In this paper we present a preliminary method of optimizing a floor layout given a set of desired proximity relations between a number of rooms or *cells*. The unconstrained problem is non-convex; however, if a relational constraint is given for each pair of cells, then the problem becomes convex and can be solved using standard convex optimization methods. The problem then becomes how to search this combinatorially large space to restrain the original problem. We first utilize the Fruchterman-Reingold algorithm, which models the cells as point masses connected to one another by springs with spring constants that correspond to the corresponding proximity relation. The equilibrium state of this spring-mass system can then be sampled to produce relational constraints. We can further leverage this to create a population input to a genetic algorithm, which can be used to find more optimal solutions in relation to the objective function. We find in the end that this procedure gave us a solution that reaches the global optimum of our objective for our particular problem instance.**

Code for this project can be accessed via Github repository: https://github.com/griffinbholt/Floor-Layout-Optimization.

I. Introduction

This project is in collaboration with the company Intralog[1]. As one of its services, Intralog sets up warehouses and distribution centers for other companies. Michael Schulte, a manager at Intralog, communicated to us that one of the more time-consuming tasks in facility planning is finding a floor layout plan that meets certain constraints, especially distance constraints between different rooms of the warehouse. Some rooms need to be close to each other because consistent travel occurs between the areas and, in Mr. Schulte's words, "travel time is the heaviest cost in warehouse operations." Other rooms need to be as far as from each other as possible; for example, high-value item storage areas should not be near any easily accessible large exits (such as a loading dock) for security purposes.

According to Mr. Schulte, although large companies such as Amazon and Walmart have the engineering resources and personnel to properly mathematically optimize warehouse distribution designs and processes, smaller companies do not have those same opportunities. In his facility planning course at Georgia Tech, Mr. Schulte was taught how to design a floor layout *by hand*, a process which can take a significant amount of time (on average, 3-4 possible layouts for a single warehouse can be produced per hour).

For a given warehouse, if we can specify at least one relative position (e.g., "the bathroom is to the *left* of the office", or "the docking area is *below* the storage room) between each of the rooms, then the problem of determining a feasible layout can be reduced to a simple convex feasibility problem [1]. We can also add an objective function minimizing distances between the centroids of each room, as well as minimum spacing constraints, minimum room areas, aspect ratio constraints, alignment constraints, symmetry constraints, similarity constraints, containment constraints, and distance constraints [1]. (A specific enumeration and explanation of the objective function and constraints we include in our problem formulation is included in Section III-C.)

However, if the $\frac{N(N-1)}{2}$ relative positions across the N rooms are *unknown*, then the problem of determining an optimal layout becomes NP-hard: it results in a Mixed Integer Program, requiring combinatorial optimization techniques. For Intralog's problem in particular, although a list of relative positions is not known, we do know a list of $\frac{N(N-1)}{2}$ *proximity preferences* on a six-point scale between each of the N cells. The purpose of this paper is to use convex optimization techniques, in combination with genetic algorithms as an intelligent search method, to search the space of possible relative cell positionings compatible with a given set of proximity preferences and automate the design of a floor layout for a given facility.

II. Related Work

Gaue and Meller provide an overview of various algorithms for searching the space of relative positionings to find optimal facility layouts [2]. Jankovits et al. created a convex optimization relaxation of the facility layout problem that achieved promising results for large facilities (i.e., facilities with about 30 rooms), but did not perform as well for facilities with 10-12 rooms [3]. Dunker, Radons, and Westkämper developed a coevolutionary method for floor planning of large facilities, whereby rooms are clustered into groups of rooms first, and then floor planning optimization is performed with genetic algorithms at two parallel levels between and within groups [4]. Several groups found genetic algorithms to be a successful method of searching the combinatorial space of relative positions [5], [6], [7]. The largest differences between our work and past work lie in two respects:

1) the input of our algorithm, which is $\frac{N(N-1)}{2}$ proximity preferences of the N rooms in the facility; and

[1] https://intralog.io

2) our use of the Fruchterman-Reingold algorithm as an initialization technique for the chromosomal population within the genetic algorithm (see Section IV).

III. Problem Formulation

Suppose that we have N rooms or areas $C_1, \ldots, C_N$ that need to be included in the floor layout design (e.g., the dock area, the main office, restrooms, storage rooms, etc.). We will refer to these rooms as *cells* to stay consistent with the literature [1]. The overall goal is to identify an arrangement of these N cells within a proposed building perimeter that meets necessary operational constraints. We will also suppose that the the proposed building for the facility is a rectangle with width W and length L, with its lower left corner at the origin $(0, 0)$ of the $\mathbb{R}^2$ plane.

We will specify the location of each cell $i = 1, \ldots, N$ by four decision variables: the coordinates (x_i, y_i) of the lower-left corner of the cell; and the width and height (w_i, ℓ_i) of the cell. Naturally, we will require that all cells lie inside the perimeter of the facility, giving us the following linear inequality constraints:

$$x_i \geq 0, i = 1, \ldots, N \tag{1}$$
$$y_i \geq 0, i = 1, \ldots, N \tag{2}$$
$$x_i + w_i \leq W, i = 1, \ldots, N \tag{3}$$
$$y_i + \ell_i \leq L, i = 1, \ldots, N \tag{4}$$

We will also suppose that we know the desired proximity relationships between each of these N rooms. We will frame these relationships as a fully-connected, undirected graph $\mathcal{G} = (C, E)$, where the nodes $C = \{C_1, \ldots, C_N\}$ are each of the N warehouse cells and E represents the set of labeled edges between each node. Each edge e_{ij} has an accompanied label which represents the desired distance relationship between cell C_i and cell C_j, on a scale from a_4 (*absolutely necessary to be close together*) to a_{-1} (*must be as far apart as possible*). The complete scale of labels is displayed in Table I. The actual numerical scale of these values will be a heuristic choice in our algorithm.

Edge Label	Interpretation
a_4	Absolutely necessary to be as close as possible
a_3	Especially important to be close
a_2	Important to be close
a_1	Ordinary closeness
a_0	No Preference
a_{-1}	Must be as far apart as possible

TABLE I: The scale of relationship labels for an undirected edge e_{ij} in the graph $\mathcal{G}$. The label describes the desired closeness of cells C_i and C_j in the final facility layout.

A. Objective Functions

Some of the relationships in the graph $\mathcal{G}$ lend themselves to be good objectives for scoring the quality of a floor layout plan. For example, if we want to minimize the distance between two cells C_i and C_j, we can utilize a *convex* objective function f_{ij} which is the Manhattan ($l1$) distance between the center of the cells C_i and C_j:

$$f_{ij}(x, w, y, \ell) = \left\| \begin{bmatrix} x_i + \frac{w_i}{2} \\ y_i + \frac{\ell_i}{2} \end{bmatrix} - \begin{bmatrix} x_j + \frac{w_j}{2} \\ y_j + \frac{\ell_j}{2} \end{bmatrix} \right\|_1, \tag{5}$$

We can then use f_{ij} to describe a multi-objective function f that prioritizes the proximity relationships defined by our graph $\mathcal{G}$. Let $\mathcal{A}_k$ represent the set of all edges e_{ij} with a relationship label of a_k. Also, let $\alpha_4 > \alpha_3 > \cdots > \alpha_0 > \alpha_{-1} = 0$ be heuristic objective weights we assign to describe the importance of each proximity set $\mathcal{A}_k, k = 4, 3, \ldots, 0, -1$. (Note that $\alpha_{-1} = 0$, which assigns *no* importance to the corresponding proximity relationships.) Then, to encourage the proximity relationships to be met, we introduce the following *convex* objective function:

$$f(x, w, y, \ell) = \sum_{k=0}^{4} \alpha_k \left[\sum_{(i,j) \in \mathcal{A}_k} f_{ij}(x, w, y, \ell) \right] \tag{6}$$

We do not include $\mathcal{A}_{-1}$ in this objective function, as the hope is that by prioritizing the proximity of the other pairs of cells within a limited space, we implicitly discourage the proximity of pairs of cells included in $\mathcal{A}_{-1}$ (cells that must be as far apart as possible).

Finally, we want to maximally utilize the space inside the warehouse (i.e., minimize wasted space between cells, not accounting for hallways). We can accomplish this by maximizing a *concave* objective function h that represents the sum of the log areas of each cell:

$$h(x, w, y, \ell) = \sum_{i=1}^{N} [\log w_i + \log \ell_i] \tag{7}$$

All of these objectives can be combined into a single objective function, which we will use to score a layout. Our *convex* objective function which we will aim to *minimize* is given by

$$\phi(x, w, y, \ell) \quad = \quad f(x, w, y, \ell) \; - \; \lambda\, h(x, w, y, \ell). \tag{8}$$

where $\lambda > 0$ represents the relative importance of maximizing utilized area. The value of this hyperparameter is another algorithm design choice (in addition to the values of $\alpha_k, k = 0, \ldots, 4$) that must be specified.

B. Relative Position Constraints

To ensure that there is no overlap between the boundaries of each of the cells, we need exactly $\frac{N(N-1)}{2}$ constraints defined as follows: for each pair of cells $(C_i, C_j), i \neq j$, *exactly one* of the following must be true:

1) C_i must be *left* of C_j;
2) C_j must be *left* of C_i;
3) C_i must be *below* C_j; or
4) C_j must be *below* C_j.

However, the space of all such possible relative positionings is exponentially large. As stated in Section I, exploring this space is the primary challenge in identifying an optimal layout for a given facility.

If we are able to select a set of $\frac{N(N-1)}{2}$ such constraints, they are specified mathematically as follows:

$$x_i + w_i \leq x_j \qquad \text{if } C_i \text{ is } \textit{left} \text{ of } C_j \tag{9}$$
$$x_j + w_j \leq x_i \qquad \text{if } C_j \text{ is } \textit{left} \text{ of } C_i \tag{10}$$
$$y_i + \ell_i \leq y_j \qquad \text{if } C_i \text{ is } \textit{below} \ C_j \tag{11}$$
$$y_j + \ell_j \leq y_i \qquad \text{if } C_j \text{ is } \textit{below} \ C_i \tag{12}$$

C. Other Constraints

In addition to the relative position constraints and the warehouse boundary constraints, we can implement each of the following constraints as desired:[2]

1) Minimum Area: If we want to require cell C_i to have a minimum area of A_i, then we introduce the following *concave* constraint:

$$\log w_i + \log \ell_i \geq \log A_i \tag{13}$$

2) Minimum or Maximum Length/Width: We can impose minimum or maximum length/width linear constraints for a specific cell C_i as follows:

$$w_{i,lo} \leq w_i \leq w_{i,hi} \tag{14}$$
$$\ell_{i,lo} \leq \ell_i \leq \ell_{i,hi} \tag{15}$$

3) Minimum or Maximum Aspect Ratios: To prevent any single room from being to "skinny" in either the horizontal or vertical directions, we want to impose bounds on the aspect ratio of each room:

$$\frac{w}{\ell} \leq \beta, \quad \frac{\ell}{w} \leq \beta \tag{16}$$

We can rewrite these two constraints as linear constraints, as follows:

$$w - \beta\ell \leq 0 \tag{17}$$
$$\ell - \beta w \leq 0 \tag{18}$$

IV. Methodology

A. Initial Approach: Finding Relative Positions with Fruchterman-Reingold

As stated earlier, the primary obstacle in producing an optimal floor layout is the selection of $\frac{N(N-1)}{2}$ relative positions between the N cells of the warehouse. If these relative positions are specified, then we only have to solve the convex optimization problem defined by the objective function $\phi(x, y, w, \ell)$ in Equation (8), the boundary constraints (1), the relative position constraints, and any of the other accompanying constraints in Section III-C one would wish to include.

To transform our desired closeness relationship graph $\mathcal{G}$ into a relative positioning, we utilized the Fruchterman-Reingold algorithm for graph drawing by force-directed placement [8]. The algorithm takes as input a graph $\mathcal{G} = (C, E)$ with edge weights ω. Each node C_i is treated as a mass that repels every other node in the graph. Each edge e_{ij} is treated as a spring with spring constant k_{ij} proportional to the edge's weight ω_{ij}. The algorithm then simulates the dynamic system of the spring network until the positions reach near equilibrium. The output of the graph is a set of positions (x_i, y_i) of each node at equilibrium. Naturally, nodes with a high edge weight between them will be drawn closer together, and nodes that are not as well connected will be farther away. Thus, running the Fruchterman-Reingold algorithm on our specific desired closeness relationship graph $\mathcal{G}$ (using edge weights ω corresponding to the edge labels) gives us candidates for the position of each cell C_i in the graph.

Using the candidate positions, we can then extract relative positions between the cells (essentially softening the positions returned by Fruchterman-Reingold) as follows: for all $i \neq j$, if cell C_i is to the left of or below cell C_j in the Fruchterman-Reingold graph layout, then we impose that same constraint on our floor layout. For our algorithm implementation in particular, we include a tolerance level to measure "to the left of" or "below"; that is, cell C_i must be to the left of or below C_j by a distance of at least ϵ in order to include the constraint in our convex optimization problem. This thins out the number of constraints at least a little bit, preventing a total ordering of cells from occurring but also potentially leading to insufficient constraints to prevent cell overlap: the exact value of ϵ is adjusted to achieve a correct balance between preventing total ordering and overlap.

In addition, we utilize an algorithm for finding the minimal equivalent graph of a directed graph [9] to remove redundancies in the set of $\approx N(N-1)$ constraints as much as possible (for example, if we know cell C_1 is to the left of C_2 and C_3, and C_2 is to the left of C_3, we can simplify these three constraints into two by only including that C_1 is to the left of C_2 and C_2 is to the left of C_3).

Using the final set of relative position constraints, we then formulate and solve a single convex optimization problem using CVXPY [10], [11] to produce a floor layout that is optimal with respect to $\phi(x, y, w, \ell)$ under the given constraints.

B. More Complete Approach: Genetic Algorithms

For reasons that will become apparent in Section V, our initial approach was ineffective at creating a layout that would actually work for an industrial facility. To better explore the space of possible relative positionings, we elected to utilize genetic algorithms [12], both because of their successful use by other researchers in exploring this space (see Section II) and because of the convenient way in which

[2]There are many more types of constraints we could include (see [1], Section 8.8.2), but the ones included here were a good starting point for an initial design.

the relative positions can be formulated as a "chromosome" in the genetic algorithm itself.

At a high level, genetic algorithms work as follows:

1) A *genome* g is defined which represents the space to be explored;
2) A *population* P of individuals/chromosomes, each with a different genome instantiation, is initialized (either randomly or by some chosen strategy);
3) For T generations:
   a) The fitness of each individual in the population is evaluated by a *fitness function*;
   b) The top-performing individuals are allowed to *survive* and/or *reproduce*;
   c) Reproduction occurs via genetic *crossover*, where some genes of one parent and another parent are switched;
   d) *Mutation* of genes in newly produced children occurs with some probability;
   e) The new children replace some proportion of the low-performing individuals of the previous generation, maintaining the same number of individuals $|P|$ within the population.
4) The top-performing individual in the final generation is returned as the best solution.

Thus, to define a genetic algorithm which allows us to explore the space of relative position constraints and find a "best" floor layout, we simply need to define the algorithm's *genome*, the *initialization strategy*, and the *fitness function*:

1) Genome: Let $g \in \{L \rightarrow, L \leftarrow, B \rightarrow, B \leftarrow\}^{\frac{N(N-1)}{2}}$ represent the genome for our genetic algorithm. Each element $g_{(i,j)}$ represents the relative position constraint between cell C_i and C_j:

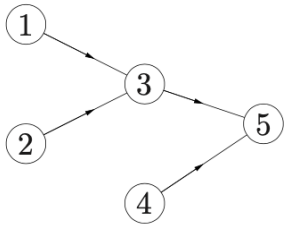
- $g_{(i,j)} = L \rightarrow$ means that C_i is *left* of C_j;
- $g_{(i,j)} = L \leftarrow$ means that C_j is *left* of C_i;
- $g_{(i,j)} = B \rightarrow$ means that C_i is *below* C_j; and
- $g_{(i,j)} = B \leftarrow$ means that C_j is *below* C_i.

For example, a genome g for a graph with $N = 5$ cells could be as follows: (note that the example genome described below is equivalent to the two directed acyclic graphs presented in Figure 1)

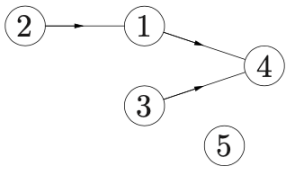
$g_{(1,2)}$	$g_{(1,3)}$	$g_{(1,4)}$	$g_{(1,5)}$	$g_{(2,3)}$	$g_{(2,4)}$	$g_{(2,5)}$	$g_{(3,4)}$	$g_{(3,5)}$	$g_{(4,5)}$
$B \leftarrow$	$L \rightarrow$	$B \rightarrow$	$L \rightarrow$	$L \rightarrow$	$B \rightarrow$	$L \rightarrow$	$B \rightarrow$	$L \rightarrow$	$L \rightarrow$

2) Initializing the Population P: Building on our initial approach (described in Section IV-A), we make use of the Fruchterman-Reingold (FR) algorithm [8] to provide an initial set of feasible relative positions. As described earlier, the Fruchterman-Reingold algorithm returns a set of exact positions for each of the N cells. We extract $N(N-1)$ relative positions from these exact positions: $\frac{N(N-1)}{2}$ *left* relative positions, and $\frac{N(N-1)}{2}$ *below* relative positions. (An example is shown in Figure 7.) We then sample half of these $N(N-1)$ relative positions to produce a chromosome g as follows:

1) With probability $p = \frac{1}{2}$, $g_{(i,j)}$ is a *left* constraint; otherwise it is a *below* constraint.



(a) Edge (i, j): C_i is *left* of C_j



(b) Edge (i, j): C_i is *below* C_j

Fig. 1: A complementary set of DAGs that are equivalent to the genome g. Images from an example in [1].

2) If $g_{(i,j)}$ is a *left* constraint:
   - If C_i is *left* of C_j in the Fruchterman-Reingold relative positions, then $g_{(i,j)} = L \leftarrow$;
   - otherwise, $g_{(i,j)} = L \rightarrow$.
3) If $g_{(i,j)}$ is a *below* constraint:
   - If C_i is *below* C_j in the Fruchterman-Reingold relative positions, then $g_{(i,j)} = B \leftarrow$;
   - otherwise, $g_{(i,j)} = B \rightarrow$.

Therefore, to completely initialize a population of $|P|$ individual chromosomes, we run Fruchterman-Riengold $|P|$ times and, each time, sample half of the returned relative positions as described above to produce $|P|$ chromosomes. (The Fruchterman-Reingold algorithm is nondeterministic, as it depends on the initial positions of the cells. The Python library we used to run FR determines these initial positions with a random seed. Thus, this adds a level of diversity to the resulting population: instead of all chromosomes in the population being sampled from the same FR graphs, they are each sampled from a different graph.)

3) Fitness Function: Our fitness function for a given chromosome g is the optimal value $\phi^\star$ of the convex optimization problem described previously in Section III, translating the chromosome g into the corresponding $\frac{N(N-1)}{2}$ relative position constraints (see Section III-B).

If a set of relative position constraints are infeasible, then the fitness function returns a score of ∞.

V. Results and Discussion

A. Initial Approach

We implemented this first method on the relationship graph defined by Figure 5 attached at the end of this paper. We used the edge weights $\omega = \{100, 50, 25, 10, 5, 0\}$ corresponding to each respective edge label in Table I. The multiobjective weighting parameters were $\alpha_k = 1, k = 0, \ldots, 4$ and $\lambda = 10$ (prioritizing the maximization of the sum of the log areas h). A visual progression of the algorithm is presented in Figures 6, 7, and 8 (attached at the end of the paper). The optimal layout returned by the algorithm is presented in Figure 2.

As you can see from the figure, this layout–although optimal according to the algorithm–is far from optimal in terms of facility floor layout design. There is still too much

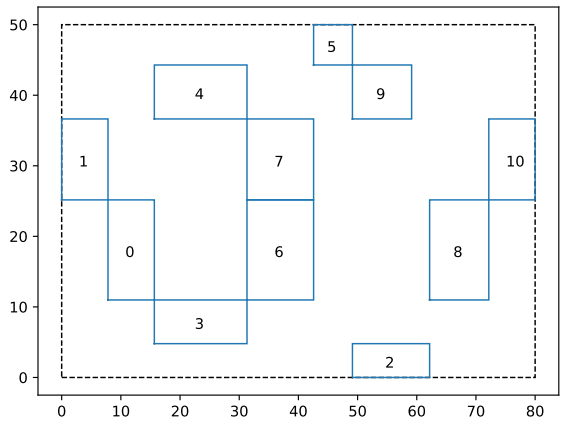


Fig. 2: The optimal floor layout returned by our initial appraoch (using just Fruchterman-Reingold) for the input relationship graph defined in Figure 5.

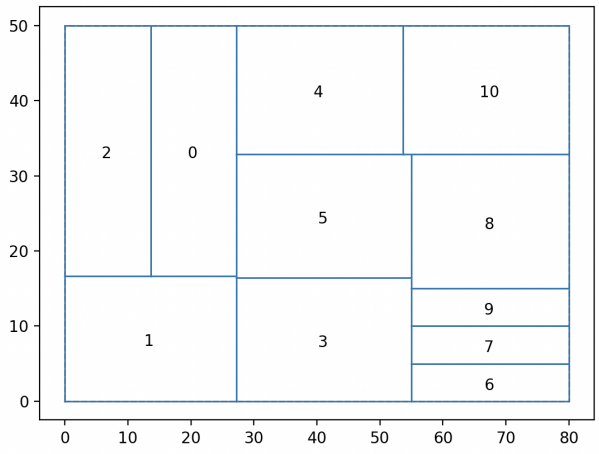


Fig. 3: The optimal floor layout returned by our genetic algorithm (with the population initalized by Fruchterman-Reingold) for the input relationship graph defined in Figure 5.

wasted space outside of the cells and, as a result, the design is not terribly practical. This is because the relative positionings extracted from the Fruchterman-Reingold layout are too restrictive. Too many relative positions are specified, which is what places the cells into a column-like and row-like organization. For example, consider cell C_1, which has empty space above and to the right. If we remove the constraints that C_1 has to be to the left of C_0 and below cells C_4, C_5, C_0, then C_1 could expand to fill that entire space.

In addition, because the Fruchterman-Reingold algorithm is not deterministic (it depends on a randomization seed), this layout is only one possible optimal layout that could be returned by our algorithm. There are many other "optimal" layouts that could be considered.

B. Genetic Algorithm Approach

For the genetic algorithm, we utilized $\alpha_4 = 1$ and $\alpha_0 = \alpha_1 = \alpha_2 = \alpha_3 = 0$ for the proximity objective weights (i.e., we only included the highest priority proximity relationships in the objective function – this was because the objective function became too complex for the CVXPY compiler). We also utilized $\lambda = 1$ (equating the priority of maximizing total utilized area with minimizing the proximity objectives). We used a steady-state parent selection strategy, single-point crossover, and a 5% probability of random gene mutation, all within the PyGAD [13] genetic algorithm framework. We initialized the population with $|P| = 1000$ individuals and ran the algorithm for $T = 100$ generations. We experimented with multiple different sets of constraints to understand their affect on the final optimal layout. Our best optimal floor layout returned by the algorithm utilized all the constraints presented in Section III-C (with $\beta = 5$, and $w_{\min} = \ell_{\min} = 5$). This best layout is presented in Figure 3; the progression of the best fitness score in the population $|P| = 1000$ over all $T = 100$ generations is presented in Figure 4.

You will notice that the resulting optimal layout in Figure 3 strongly resembles what could be an actual building floor plan. For this particular problem, we aimed for cells C_6 and C_7, and C_7 and C_9 to be as close together as possible (i.e., $\mathcal{A}_4 = \{(6,7),(7,9)\}$): this goal was definitely achieved. In fact, cells C_6 and C_7, as well as C_7 and C_9, are as close together as is allowed by our included constraint $\ell_{\min} \geq 5$: therefore, we know that f is at a minimum. In addition, there is no wasted space in the entire layout, which means that total maximum area ($h(x, y, w, \ell)$) was at its maximum possible value. Thus, we have actually found a *globally* optimal solution for our particular constraints and objectives (this is not universally guaranteed by our algorithm, but is a nice result for our particular instance).

However, even though this layout is globally optimal for our specific objective and constraints, it does not mean that this design is as functional as it could be. For example,

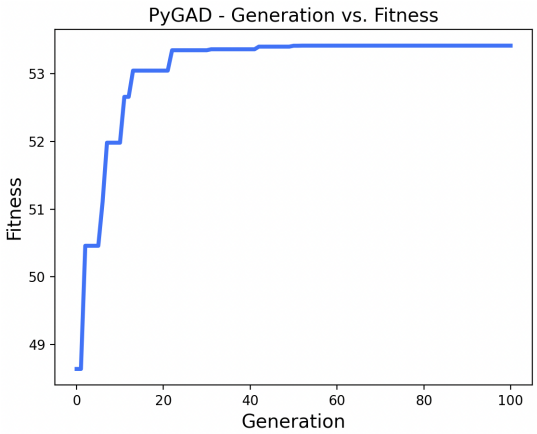


Fig. 4: Best fitness score in the population vs. generations of the genetic algorithm, for the rendition of the algorithm that output the final floor layout in Figure 3

certain corners and edges do not line up: it would be more practical in terms of construction for the corners of C_4, C_5, C_8, C_{10} to all line up, or for the top edge of C_9 to line up with the bottom edge of C_5. In addition, because two of our objectives were to minimize the $l1$-distance between the centers of cells C_7 and C_9, as well as the $l1$-distance between the centers of cells C_6 and C_7, the optimal layout squishes all three of those rooms on top of each other and pushes them to be as small as possible. These two results suggest that our objective function and/or constraints may need to be reworked to better reflect our desires for the building.

An additional drawback for this approach is the amount of time required to run the genetic algorithm. It took approximately 2 hours to complete $t = 100$ generations. The significant runtime was due to the computational cost of computing the fitness function with the ECOS solver in CVXPY for each individual at every generation. In addition, you will notice that all 100 generations were likely unnecessary–the algorithm converged around 35 iterations.

VI. Conclusion

The ultimate goal is to provide Intralog with an interactive software tool that utilizes optimization techniques to automate the design of a facility. Facility designers, such as Michael Schulte, will be able to input desired constraints into the software and then the software will attempt to find an optimal floor layout for the given constraints. This project was a significant step towards the completion of such a tool. Our genetic algorithm did produce a globally optimal floor layout for a facility with $N = 11$ cells: the layout both maximized total area used and minimized the distance between the highest priority cell proximity relationships.

Part of the success of our genetic algorithm may be due to the use of the Fruchterman-Reingold algorithm to initialize the population. Fruchterman-Reingold initialization provides a clear advantage over random initialization: random initialization has no guarantee of producing any feasible relative position constraints, whereas we already know that the relative position constraints sampled from the results of Fruchterman-Reingold are feasible. Therefore, the genetic algorithm does not have to waste any time finding feasible layouts: it can concentrate on simply combining the best features of the already feasible initial population. Thus, although our initial approach, which consisted of only using Fruchterman-Reingold to find a single set of relative positions, did not return a good floor layout, it ultimately led us to a better algorithm.

A. Future Work

One of the primary issues with our final optimal layout was the misaligned corners and edges present within the final layout. To encourage corners and edges within the floor layout to align, we conjecture it would be useful to experiment with adding the following convex function to our objective function in Equation 8:

$$q(x, y, w, \ell) = \sum_{(i,j):i<j} [|x_i - x_j| + |y_i - y_j| + |w_i - w_j| + |\ell_i + \ell_j|] \quad (19)$$

Minimizing the sum of the absolute differences of each element in x, y, w, ℓ, respectively, will naturally cluster the values in x, y, w, ℓ: therefore, including this objective will hopefully result in the alignment of corners and edges.

An additional issue was the "squished" layout of those cells which we wanted close together. Fortunately, we believe that this is just a result of our particular problem instance, where we only minimized the distances between the high priority cell relationships (C_6, C_7) and (C_7, C_9). By including all of the proximity relationships (i.e., setting $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ to nonzero values), we may be able to counteract some of that "squished" behavior. (We were unable to experiment with this for the time being because the resulting objective function was too complicated, or too lengthy, for the CVXPY compiler–experimentation with other solvers and vectorization of our objective function and constraints could mitigate this issue.)

To improve the runtime of the algorithm (currently, $\approx$ 2 hours), it would be beneficial to utilize a faster solver for the fitness function (at an acceptable cost of accuracy). Experimentation with multiple solvers will be necessary to understand the resulting time-accuracy tradeoffs. We also should experiment with the number of generations of the genetic algorithm that are required to return a good floor layout across many different problem instances.

We could also make additional improvements by experimenting with parallelization; loading the fitness function (finding the optimal objective value $\phi^\star$) onto a GPU; a thorough experimental comparison of our method with the methods proposed by other researchers (our paper currently makes no claims about the performance of our algorithm against theirs – this would require much more experimentation); and perhaps a combination of several proposed methods, including our own, to produce a more robust algorithm. All of these experiments will hopefully create a tool that is beneficial to Intralog in automating facility layout design.

References

[1] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, March 2004. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike-20&path=ASIN/0521833787

[2] K.-Y. Gau and R. Meller, "An iterative facility layout algorithm," *International Journal of Production Research*, vol. 37, no. 16, pp. 3739–3758, 1999.

[3] I. Jankovits, C. Luo, M. F. Anjos, and A. Vannelli, "A convex optimisation framework for the unequal-areas facility layout problem," *European Journal of Operational Research*, vol. 214, no. 2, pp. 199–215, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221711003560

[4] T. Dunker, G. Radons, and E. Westkämper, "A coevolutionary algorithm for a facility layout problem," *International Journal of Production Research*, vol. 41, no. 15, pp. 3479–3500, 2003. [Online]. Available: https://doi.org/10.1080/0020754031000118125

[5] J. Diego-Mas, M. Santamarina-Siurana, J. Alcaide-Marzal, and V. Cloquell-Ballester, "Solving facility layout problems with strict geometric constraints using a two-phase genetic algorithm," *International Journal of Production Research*, vol. 47, no. 6, pp. 1679–1693, 2009. [Online]. Available: https://doi.org/10.1080/00207540701666253

[6] M. Mazinani, M. Abedzadeh, and N. Mohebali, "Dynamic facility layout problem based on flexible bay structure and solving by genetic algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 65, 03 2012.

[7] A. McKendall and A. Hakobyan, "An application of an unequal-area facilities layout problem with fixed-shape facilities," *Algorithms*, vol. 14, no. 11, 2021. [Online]. Available: https://www.mdpi.com/1999-4893/14/11/306

[8] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380211102

[9] H. T. Hsu, "An algorithm for finding a minimal equivalent graph of a digraph," *J. ACM*, vol. 22, pp. 11–16, 1975.

[10] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[11] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[12] V. Chahar, S. Katoch, and S. Chauhan, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools and Applications*, vol. 80, 02 2021.

[13] A. F. Gad, "Pygad: An intuitive genetic algorithm python library," 2021.

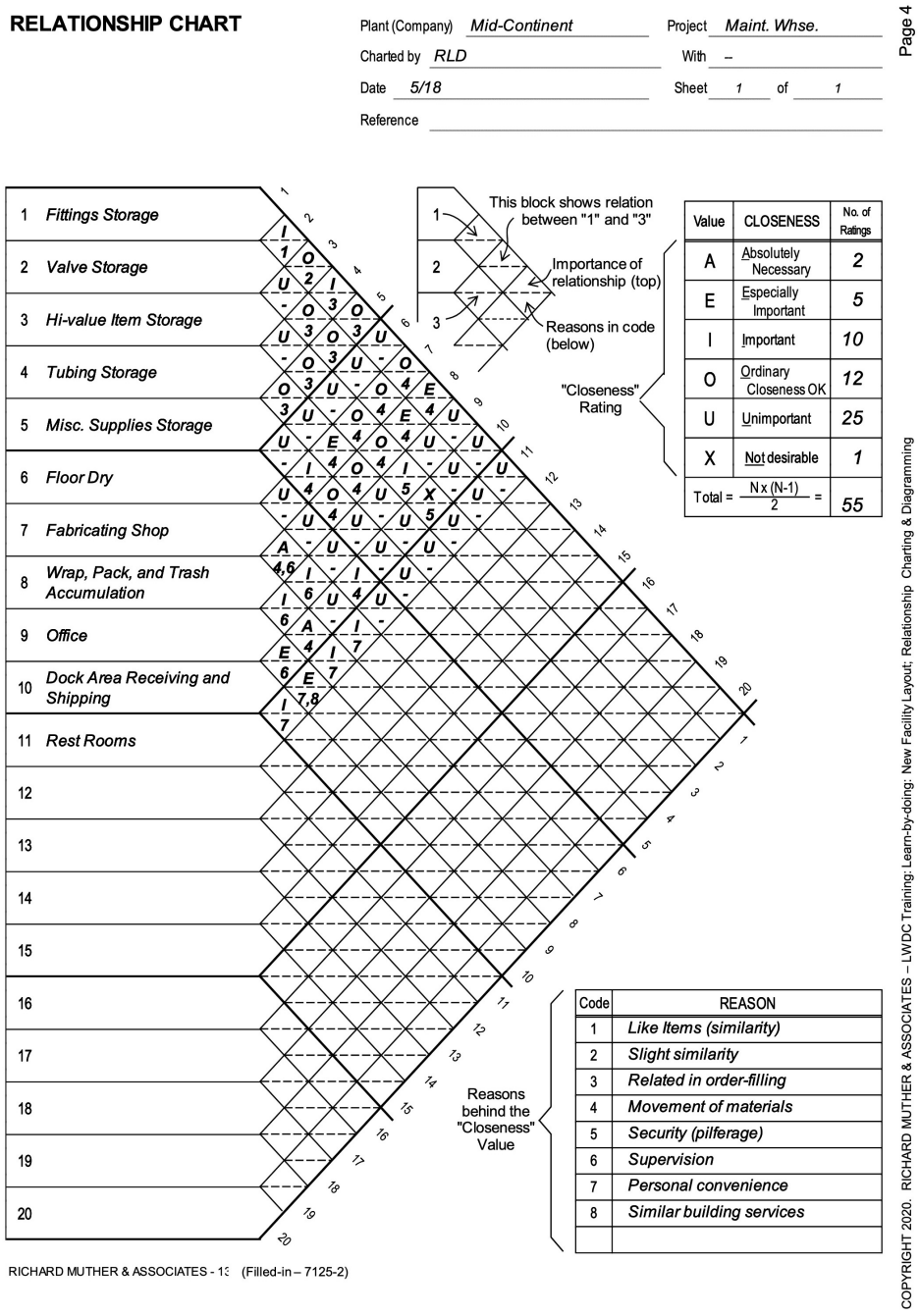


Fig. 5: An example relationship chart, from the facility planning course at Georgia Tech, displaying the $\frac{n(n-1)}{2}$ nearness preferences between the $n = 11$ areas required in the facility. The nearness preferences are on a six-point scale, from $A =$ Absolutely Necessary and $X =$ Not Desirable. The cells are one-indexed.

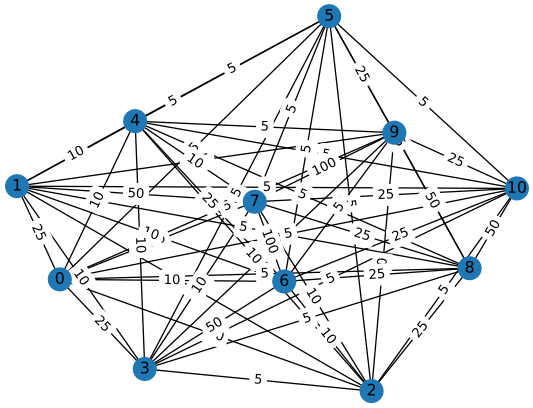
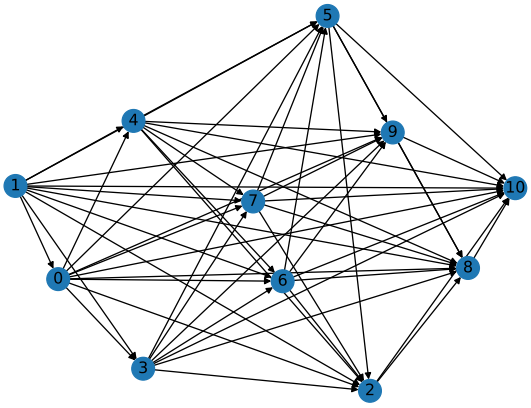
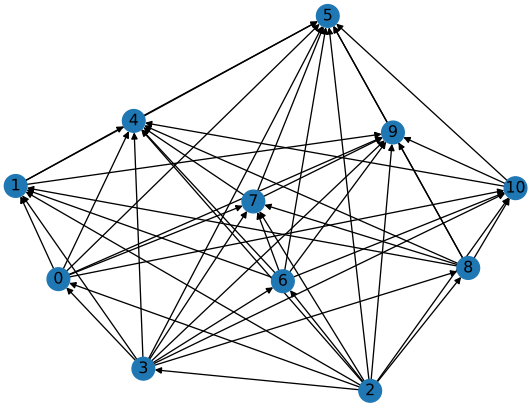


Fig. 6: The layout produced by running the Fruchterman-Reingold algorithm on our relationship graph $\mathcal{G}$ defined in Figure 5, including the assigned edge weights. The cells are zero-indexed.

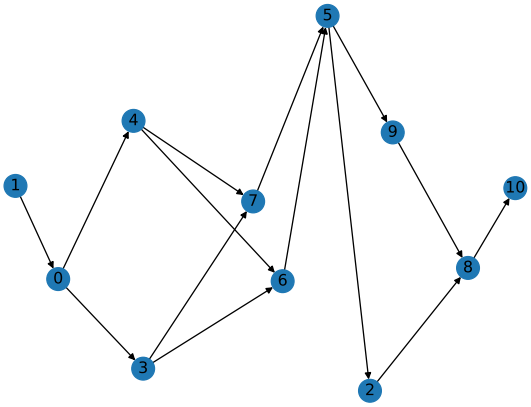


(a) "Left" relative positions: an edge from node i to node j means that cell C_i is *to the left of* C_j.

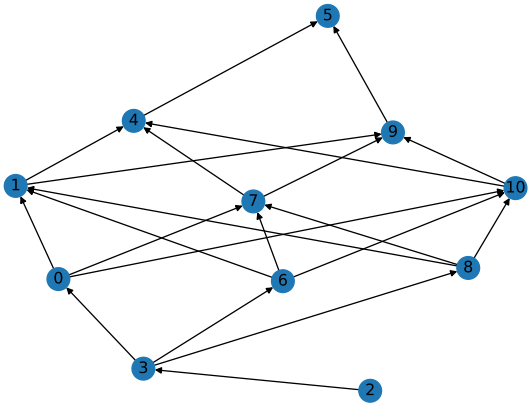


(b) "Below" relative positions: an edge from node i to node j means that cell C_i is *below* C_j.

Fig. 7: Relative positions extracted from the Fruchterman-Reingold layout in Figure 6.



(a) "Left" relative positions



(b) "Below" relative positions

Fig. 8: Minimal equivalent graphs of the relative positions in Figure 7 (i.e., all redundant constraints are removed).