ELOPE: Evolutionary Layout Optimization Playground and Evaluator

Open source evaluator, Search Optimization , Layout Optimization, Evolutionary Algorithms, Genetic Algorithms

Abstract

Current ameliorations in Evolutionary Strategies have allowed us to achieve phenomenal accuracy in variants of optimization problems. However, for the layout optimization problem, there doesn't exist a platform adept enough to challenge off-the-shelf algorithms to the fullest extent. For addressing the mentioned problem statement, we introduce Evolutionary Layout Optimization Playground and Evaluator (ELOPE), a platform for solving a large spectrum of layout optimization problems. ELOPE's purpose is to provide the community with the platform required for understanding and evaluating algorithms for solving the layout optimization problem. ELOPE's major contributions are its proposed $2D$ rectangular representation for the environment, multi-objective optimization, pre-defined plugins for evolutionary algorithms and visualization of the generated layouts. As an illustration, we optimize the layout of an environment with various modules considering multiple constraints by employing the genetic algorithm and the plugins available in ELOPE.

1 Introduction

Platforms which make it easy to implement and evaluate algorithms are very crucial in reducing friction while developing solutions to the real-world problems. It also reduces the level of expertise required to build solutions especially when there are plethora of strategies being proposed in the field of Artificial Intelligence(AI). Such platforms tend to provide boilerplate code to the community leading to accelerated adaptation of algorithms in applied sciences. The Arcade Learning Environment(ALE) [Bellemare *et al.*, 2013] facilitated the development of learning algorithms by providing standard benchmarks for evaluation and comparison. Open AI Gym [Brockman *et al.*, 2016] has grown as a de-facto bench-marking toolkit for reinforcement learning research. Eos [Bonsma *et al.*, 2000] supports rapid implementation of evolutionary algorithms, ecosystem simulations and hybrid models, and SPGAO [Liu, 2012] provides a simulation platform for the genetic algorithm along with visualisation. But, there is no platform that provides a base structure with plugins for solving the layout optimization problem using evolutionary algorithms. ELOPE, is a platform for evaluating and comparing different evolutionary algorithms for solving layout problems using evolutionary algorithms.

These are the major contributions made in this paper:

- **ELOPE provides a scalable playground** as it can be used to generate diverse set of layout customizations for various applications.
- **It provides a rich set of plugins** that makes it easy to use for developing various functionalities on the top of it.
- **It provides visualisation module** which can be used to visualise layouts generated by the algorithm.

The remainder portion of paper is organised as follows. Section 2 describes the literature review for the problem statement and solutions for the same. Section 3 contains the architectural overview of the ELOPE. In Section 4, we have explained possible interaction of ELOPE with any algorithm by taking Genetic Algorithm as an example. Finally, we have made conclusion in the Section 5 along with discussing future possible contributions.

2 Related Work

Layout Optimisation Problem refers to the problem of determining positions of modules in a layout for optimizing multiple constraint objectives specific to the problem statement. The discussed problem statement can be divided into two subdivisions as continuous layout optimization problem and discrete layout optimization problem. Researchers like [Tam and Li, 1991], [Van Camp *et al.*, 1992] [Tam, 1992b], [Tam, 1992a] have contributed immensely for continuous layout optimisation problem, whereas [Armour and Buffa, 1963], [Lee, 1967], [Khalil, 1973] made numerous efforts for discrete layout optimisation problem. Apart from mentioned categorization, there are also additional possible ways of categorizing Layout optimization problem as equal area problem, unequal area problem [Lee *et al.*, 2005], Quadratic Assignment Problem (QAP) and Quadratic Set Covering Problem (QSCP) [Rajasekharan *et al.*, 1998].

For solving the above mentioned problem scenarios, a wide array of algorithms have been employed. genetic algorithms

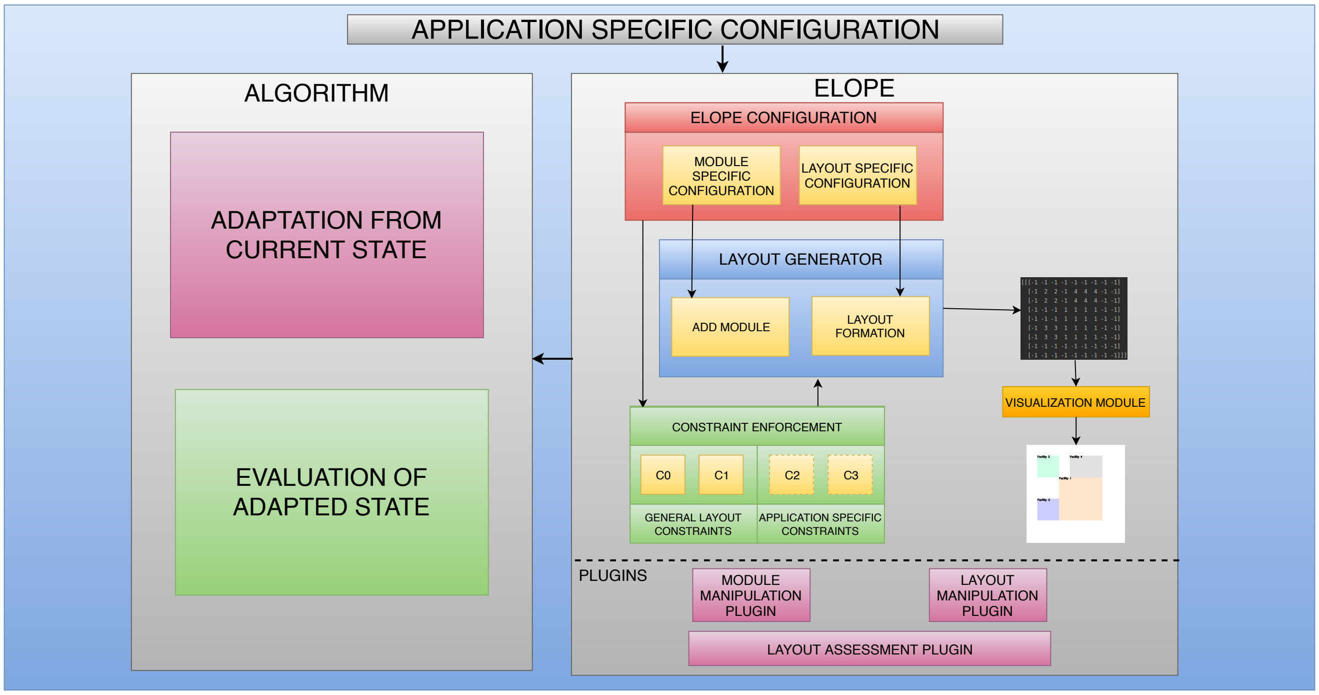

APPLICATION SPECIFIC CONFIGURATION
ALGORITHM
ADAPTATION FROM CURRENT STATE
EVALUATION OF ADAPTED STATE
ELOPE
ELOPE CONFIGURATION
MODULE SPECIFIC CONFIGURATION
LAYOUT SPECIFIC CONFIGURATION
LAYOUT GENERATOR
ADD MODULE
LAYOUT FORMATION
VISUALIZATION MODULE
CONSTRAINT ENFORCEMENT
C0
C1
C2
C3
GENERAL LAYOUT CONSTRAINTS
APPLICATION SPECIFIC CONSTRAINTS
PLUGINS
MODULE MANIPULATION PLUGIN
LAYOUT MANIPULATION PLUGIN
LAYOUT ASSESSMENT PLUGIN

Figure 1: ELOPE's architecture and interaction with the algorithm.

were used by [Rajasekharan *et al.*, 1998] for flexible manufacturing system optimisation, genetic search by [Tate* and Smith, 1995] for unequal-area facility layout optimization, genetic programming by [Zheng *et al.*, 2006] for Geometry and sizing optimisation of discrete structure. This problem statement has also been assessed by researchers from diverse applied engineering fields. [Ismail *et al.*, 2012] have explored the same for optimizing placement of components on the PCB using genetic algorithms. [Mohamed *et al.*, 2015] have analyzed layout improvement for steel fabrication works.Though community has explored the wide range of layout optimisation problems and employed diverse strategies, it lacks a playground which can be used as a standard tool for evaluating the progress of the community. For the same purpose, ELOPE aims to be the standard platform which can be used to test and benchmark the evolutionary algorithms. [Hosseini-Nasab *et al.*, 2018] study that genetic algorithm is the most used algorithm for solving the layout optimization problem, this makes it obvious to equip ELOPE with plugins to support the development of various variants of the genetic algorithm. This paper also briefly explains using the genetic algorithm with ELOPE plugins to solve a multi-objective layout optimisation problem.

3 ELOPE Architecture

The key idea for ELOPE is to avail the community with a comprehensive platform for solving the layout optimisation problems. Hence, in this section architecture of ELOPE is discussed in detail. ELOPE is written in Python to avail the implemented operations for matrix manipulation and its matured range of libraries for applied programming. This choice also caters the developers to prototype the ideas relatively faster and help direct the efforts on the algorithm than the interfaces.

ELOPE stands out as a bridge between the problem statement and the algorithm employed to solve it. Figure 1 illustrates that relationship via the user-defined data-flow originating from the problem statement at hand. The interactions among the sub-modules of ELOPE and the optimization strategy of choice are also represented. Furthermore, the section describes each building block of ELOPE for the ease of development.

3.1 Application specific configuration

The information corresponding to the layout being optimised acts as an input for the ELOPE environment. This information is represented in a configuration file consisting of fields related to the physical floor-space, constraints and specifications about the components to be placed optimally in the layout.

3.2 ELOPE

ELOPE represents the multi-floor space as a set of $3D$ grid boxes, which behave as the fundamental cell. Each grid box can be represented as $G_{x,y,z}$ where x, y and z where $x \in \{1, ..., X\}, y \in \{1, ..., Y\}$ and $z \in \{1, ..., Z\}$. The X, Y and Z are the inputs derived from the application specific configuration which denote the width, length and number of floors respectively for the floor space. Each grid box acquires a token value (g) from the grid token set (G).Here $g \in G$ where,

$$G = \{-2, -1, 0, M\}$$

where,

-2	represents blocked or unusable space
-1	represents vacant or usable space
0	represents the connections between modules
M	represents module ID (explained later in this section)

Module

The physical components which are supposed to occupy the floor-space or in other words whose positions are to be optimized are addressed as module henceforth. The properties of module (P_M) for single floor is represented as follows:

$$P_M = \{M_{x,y}, M_{co}, M_d, M_{ui}, l, w\}$$

where,

$M_{x,y}$	represents x, y coordinates of the module
M_{co}	represents connectable grid boxes of the module
M_d	represents orientation direction of the module
M_{ui}	represents user block
l	represents length of the module
w	represents width of the module

Layout

In ELOPE, the floor space along with its modules is addressed as the layout. Using the application specific configuration file, ELOPE represents a single floor-space with a set of $2D$ rectangular grid boxes. Further, the layout can contain m number of modules of width w_i and length l_i where $i \in \{1,......m\}$. So the number of modules contained in the environment should satisfy the following condition:

$$\sum_{i=1}^{m} (w_i \times l_i) \leq (X \times Y) \tag{1}$$

ELOPE Configuration

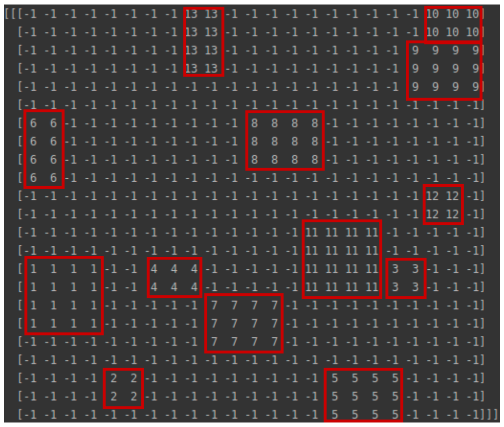
The inputs from the application specific configuration file are inherently used by ELOPE in two contexts to generate the complete layout. They are identified as specifications or constraints pertaining to the Module or Layout.

Layout Generator

The Layout Generator is responsible for generating layouts by placing each module as per the user defined strategies. The layout is generated considering the layout specific configuration parameters. Most of the evolutionary approaches place the modules at random location within the layout. Hence the add module functionality facilitates the placement by default at random locations considering the specifications and constraints from respective configuration parameters for the module. Figure 2 illustrates the sample from layout generator.

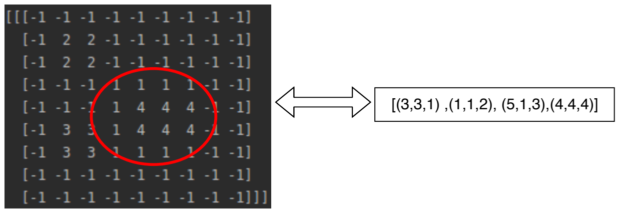
Constraint Enforcement

The ELOPE configuration file generates constraint enforcement which acts as an input for layout generator. Constraint enforcement can be categorized as general layout constraints and application layout constraints. Constraints are defined using the information from grid token set G. For example, general layout constraints includes layouts containing non-overlapping modules, modules that should be represented within the boundary of layout whereas application specific constraints includes constraints such as user accessibility constraint, blockage constraint and module adjacency constraint.



```
[[[-1 -1 -1 -1 -1 -1 -1 -1 13 13 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 10 10 10]
  [-1 -1 -1 -1 -1 -1 -1 -1 13 13 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 10 10 10]
  [-1 -1 -1 -1 -1 -1 -1 -1 13 13 -1 -1 -1 -1 -1 -1 -1 -1 -1  9  9  9  9]
  [-1 -1 -1 -1 -1 -1 -1 -1 13 13 -1 -1 -1 -1 -1 -1 -1 -1 -1  9  9  9  9]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  9  9  9  9]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
  [ 6  6 -1 -1 -1 -1 -1 -1 -1 -1 -1  8  8  8  8 -1 -1 -1 -1 -1 -1 -1 -1]
  [ 6  6 -1 -1 -1 -1 -1 -1 -1 -1 -1  8  8  8  8 -1 -1 -1 -1 -1 -1 -1 -1]
  [ 6  6 -1 -1 -1 -1 -1 -1 -1 -1 -1  8  8  8  8 -1 -1 -1 -1 -1 -1 -1 -1]
  [ 6  6 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 12 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 12 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 11 11 11 11 -1 -1 -1 -1 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 11 11 11 11 -1 -1 -1 -1 -1]
  [ 1  1  1  1 -1 -1  4  4  4 -1 -1 -1 -1 -1 11 11 11 11  3  3 -1 -1 -1]
  [ 1  1  1  1 -1 -1  4  4  4 -1 -1 -1 -1 -1 11 11 11 11  3  3 -1 -1 -1]
  [ 1  1  1  1 -1 -1 -1 -1 -1  7  7  7  7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
  [ 1  1  1  1 -1 -1 -1 -1 -1  7  7  7  7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1  7  7  7  7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
  [-1 -1 -1 -1  2  2 -1 -1 -1 -1 -1 -1 -1 -1 -1  5  5  5  5 -1 -1 -1 -1]
  [-1 -1 -1 -1  2  2 -1 -1 -1 -1 -1 -1 -1 -1 -1  5  5  5  5 -1 -1 -1 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  5  5  5  5 -1 -1 -1 -1]]]
```

Figure 2: The internal representation of the layout is demonstrated. The empty space is represented with -1 and the highlighted portion in red represents the modules with their unique module ids(M).



```
[[[-1 -1 -1 -1 -1 -1 -1 -1 -1]
  [-1  2  2 -1 -1 -1 -1 -1 -1]
  [-1  2  2 -1 -1 -1 -1 -1 -1]
  [-1 -1 -1  1  1  1  1 -1 -1]
  [-1 -1 -1  1  4  4  4 -1 -1]
  [-1  3  3  1  4  4  4 -1 -1]
  [-1  3  3  1  1  1  1 -1 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1]
  [-1 -1 -1 -1 -1 -1 -1 -1 -1]]]
```

[(3,3,1) ,(1,1,2), (5,1,3),(4,4,4)]

Figure 3: The modules represented with ids 1 and 4 can be easily identified using a $2D$ rectangular matrix. Whereas, vectors are represented as a list of tuples. Each tuple shows the x-coordinates, y-coordinates of the module and module id respectively. So, visualization using $2D$ matrix is very effective as compared to a vectored representation.

Plugins

ELOPE's plugins are divided into 3 major categories:

1. **Module manipulation plugin**: This set of plugin is used to move the modules within the layout. They include:

   Shift Plugin: This plugin takes in the module id and shifts that particular module by n position in the given direction. Mathematical formulation of shift plugin is as follows:
   For right/left shift:

   $$(c', d') = (c, d \pm n)$$

   For top/down shift:

   $$(c', d') = (c \pm n, d)$$

   where,

   c, c' represents the previous and new top left corner's

	row position respectively
d, d'	represents the previous and new top left corner's column position respectively
n	represents number of blocks to move

Rotate Plugin: This plugin takes in the module id and rotates the module by 90 or 180 degrees, if and only if there is sufficient space to rotate the module.

2. **Layout manipulation plugin** This set of plugin modifies the layout as a whole. Geometric transformations applied on the layout can be included in this category.

3. **Layout assessment plugin** These plugins are used for assigning a score to the layout by assessing various parameters of the layout. ELOPE is equipped with various plugins like Euclidean distance, Manhattan distance, minimal path plugin and minimal area plugin. What follows is mathematical description of minimal area and minimal path plugins.

Minimal path plugin: The lowest length of connecting path between 2 modules is the number of minimum steps required to move from module i to module j represented as $p_{i,j}$. The overall path length among the modules is represented as p_l which is calculated as follows:

$$p_l = \sum_{i,j=1,j\neq i}^{n} p_{i,j} \quad (2)$$

Minimal area plugin: The minimal effective area covered by the modules is calculated as follows:

$$a_m = (x_{max} - x_{min}) \times (y_{max} - y_{min}) \quad (3)$$

where,

a_m	represents overall area occupied by the modules
x_{max}, x_{min}	represents the maximum and minimum x-coordinates of the module respectively
y_{max}, y_{min}	represents the maximum and minimum y-coordinates of the module respectively

To bring above objectives on the small scale and giving equal weight to all the objectives, we do normalization:

$$p_n = \frac{(p_l - p_{min})}{(p_{max} - p_{min})} \quad (4)$$

where,

p_n	represents the normalized path
p_l	represents overall path length
p_{min}	represents minimum path length between modules
p_{max}	represents perimeter of the layout

$$a_n = \frac{(a_m - a_{min})}{(a_{max} - a_{min})} \quad (5)$$

where,

a_n	represents the normalized area
a_m	represents overall area occupied by the modules
a_{min}	represents sum of area of all the modules
a_{max}	represents area of the layout

Visualization Module: The layout that is internally represented as a Numpy array is not a good way when it comes to visualizing the layout. This is the reason ELOPE comes with a visualization module for converting all the information from the Numpy array to a form that can be easily understood and visualized. Figure 4 shows the layout generated using Visualization Module.

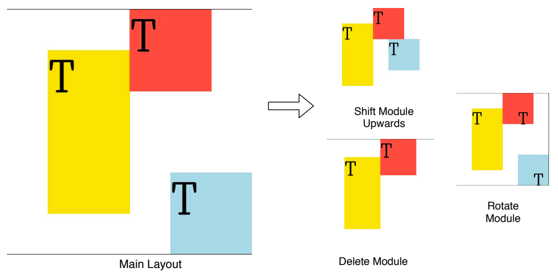


Figure 4: The internal representation of the layout is converted to the form which is convenient for visualization. Demonstration of how main layout looks after applying shift, delete and rotate plugin is shown.

3.3 Algorithm

Paradigm of any search algorithm can be represented as 2 block entity, Adaptation From Current State and Evaluation of Adapted State. In figure 1, we have tried to explain how any generic search algorithm can be represented and in the figure 5 we have explained how blocks of any search algorithm can interact with ELOPE's modules by considering genetic algorithm as an example.

4 Locus Classicus

What follows is a case study representation of layout optimization problem statement, where we have exemplified layout optimization in an industrial warehouse environment and provided an experimental analysis of solution using the genetic algorithm. In this section, we have demonstrated how functionalities and plugins of ELOPE can be used as higher level building blocks for constructing various subdivisions of the algorithm.

4.1 Constraint Overview

Constraints are essential part of any search optimization problem as it curtails search space to a considerable amount. Any search algorithm adapts the state of the search space such that exploration and exploitation are balanced while the constraints are fulfilled simultaneously. In the ELOPE, for facilitating a diverse variety of constraints, they have been classified into two possible approaches. 1) Constraint guided state adaptation: These set of constraints guide the algorithm in the process of state formation. 2) Constraint satisfied state adaptation: These set of constraints evaluate the state after it's creation.

In this implementation, three constraints were defined,

1. User accessibility constraint: For module with ID M user box is defined as $M_{ui,uj}$ where i, j are the coordinates of the user blocks in $G_{i,j}$ where $i \in \{1, ..., X\}$ and $j \in \{1, ..., Y\}$.

   $M_{ui+n,uj+n} = -1$ is the condition for constraint satisfaction. Where,
   n is the minimum distance for user accessibility. This constraint is considered under guided state adaptation category.

2. Blockage constraint: This constraint defines the grid boxes which are unusable. $G_{i,j} = -2$ is the condition for constraint satisfaction. This constraint is considered under satisfaction state adaptation category.

3. Module adjacency constraint: This constraint allows to keep some reserved space around any given side of the module. For example, a module with module ID M needs a grid blocks towards the left side. Then

   $$M_{x,y+l+a} - M_{x,y+a} : M_{x+l+a,y+l+a} - M_{x+l,y+l} = -1$$

   is the constraint that needs to be satisfied.

   This constraint is considered under guided state adaptation category.

4.2 Optimization Objective

For the expounded problem statement of industrial warehouse layout optimization, we want to optimize the machinic module configuration for facilitating the lowest length of conveyor and minimize the effective area covered by the modules. For calculating optimization objective mentioned, we are using ELOPE plugins for minimal path and area calculations. We are getting scores p_n and a_n as defined in equations 4 and 5 respectively from the plugins and calculating overall score as mentioned below.

The objective function considering above normalized values is:

$$Ob_f = \frac{1}{(p_n + a_n)} \tag{6}$$

where,

Ob_f represents the objective function

4.3 Genetic Algorithm and Implementation

The genetic algorithm is a type of evolutionary algorithm that tries to mimic the actual biological process in a hope of discovering good solutions which was first introduced by [Holland and others, 1992]. The genetic algorithm is analogous to Darwinian natural mutation and selection, it is a part of 'randomized heuristics' which does not depend on the prior knowledge of various features of the domain whereas it depends on the randomized choice of operators.

Algorithm 1 Genetic Algorithm

```
Input: desired score, maximum number of generations
Parameter: Fitness strategy
 1: Initialize initial population
 2: Set population score to zero
 3: Set generation number to zero
 4: while population score<desired score or generation
    number<maximum generations do
 5:     Get sorted population, population score and best individual score
 6:     Perform selection
 7:     Perform crossover
 8:     Sort population
 9:     Perform mutation
10:     Increment generation number
11: end while
```

Specifically, for solving this problem statement using genetic algorithm we have developed various plugins, they are as follows:

1. **Initial population generator**: This plugin takes in a number of individuals in the initial population (Step 1 in Algorithm 1), and simply generates that many individuals using individual generator. The user can easily add some special individuals that it wishes to be a part of the initial population. Population is represented as $P = \{L_0, L_1, ..., L_n\}$ where, n is number of individuals in the population.

2. **Fitness strategies**: There are various fitness strategies that are used for getting fitness value for various constraints (Step 5 in Algorithm 1). As defined in section 'n' we are using the fitness strategies in the following manner. Fitness is calculated as $F_p = \{S_{L_i}\}$ where, $S_{L_i} = f(L_i)$ here $i = 0, ..., n$.

3. **Crossover strategies**: Crossover is used for exploring the sample space in the Genetic Algorithm (Step 7 in Algorithm 1), we implement 2 different strategies in our plugin, they are as follows:

   (a) In the most basic strategy, we choose any environment at random and take the module's position from it for creating a new environment. If there is any conflict in creating the environment then a completely new environment is used as a child.

   (b) The second strategy involves using the fitness score of the environment. The environment with a higher score is given an 80% probability of contributing in the crossover process, whereas the one with the lower score is given the remaining 20% probability.

   For old population P, new population P' is generated as $P' = C(P)$ where C is crossover function.

4. **Mutation strategies**: Mutation is used for exploiting the sample space in the Genetic Algorithm (Step 9 in Algorithm 1), we implement 3 different strategies in our plugin, they are as follows:
   (a) **Shift mutation**: This strategy involves moving the module up, down, right or left by n number of blocks.
   (b) **Tilt mutation**: This strategy involves rotating the module by 180 degrees.
   (c) **Rotate mutation**: This strategy involves rotating the module by 90 degrees.
5. **Selection strategy**: This plugin is used for selecting individuals for the next generation. Every alternate environment is chosen from the entire population and these selected individuals take part in the selection process.

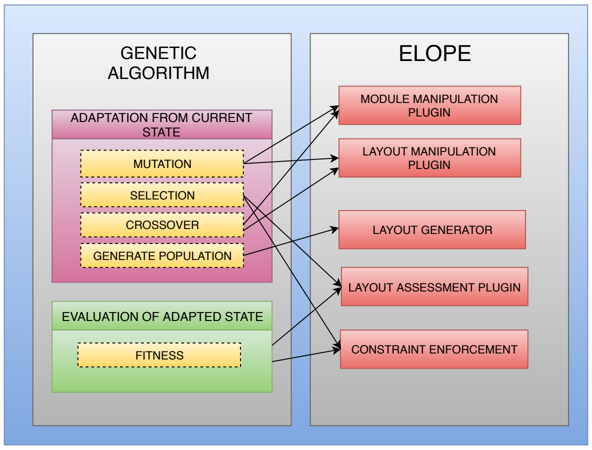


Figure 5: The use of ELOPE to develop genetic algorithm is demonstrated. The mapping of different strategies of genetic algorithm is shown with the corresponding modules of ELOPE used.

All these plugins of the genetic algorithm are developed using ELOPE. As shown in Figure 5, generate population is build using ELOPE's Layout Generator, crossover and mutation is build using module and layout manipulation plugin, whereas selection and fitness strategy is developed using constraint enforcement and layout assessment plugin. This demonstrates the purpose of ELOPE in developing different algorithms.

4.4 Results

For getting an optimized industrial warehouse environment the algorithm was made to evolve until the desired score was achieved. In Figure 6, the tracks of the conveyor are represented between the modules. Clearly, in Figure 6 (a) the conveyor length is higher than as shown in Figure 6 (b) and also the area occupied by the modules and conveyor is reduced to a certain extent. Overall we get an optimized layout after evolving for 160 generations through the genetic algorithm.

5 Conclusion and Future Work

This paper has revealed the design policies of ELOPE and its suitability to the optimization process using evolutionary

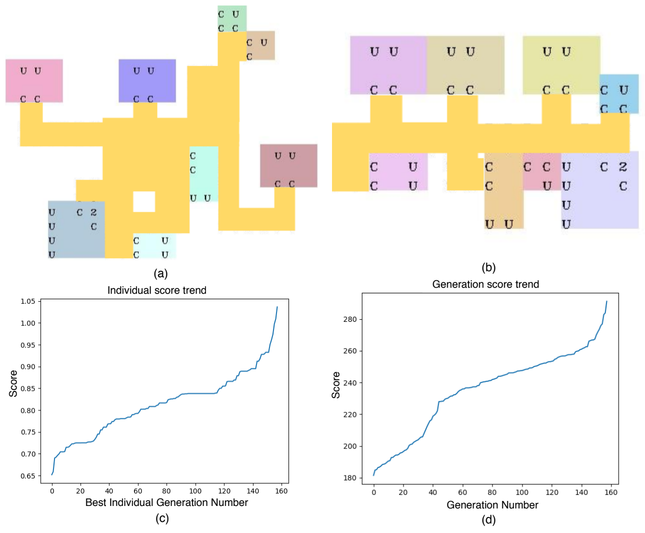


Figure 6: (a) shows the initial layout where modules are placed randomly, (b) shows the optimized layout evolved after 160 generations, (c) represents the best layout score and (d) represents the overall generation score.

algorithms. ELOPE can be elegantly used as a playground for developing, understanding and evaluating not just evolutionary algorithms but also Reinforcement Learning (RL) and other search optimization techniques. ELOPE in its future version aims to support various nuances of layout optimisation problems along with $3D$ visualisation modules. ELOPE also needs a standard leaderboard for these environments in maintaining the community's progress.

References

[Armour and Buffa, 1963] Gordon C Armour and Elwood S Buffa. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, 9(2):294–309, 1963.

[Bellemare *et al.*, 2013] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[Bonsma *et al.*, 2000] Erwin Bonsma, Mark Shackleton, and Rob Shipman. Eos—an evolutionary and ecosystem research platform. *BT Technology Journal*, 18(4):24–31, 2000.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Holland and others, 1992] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[Hosseini-Nasab *et al.*, 2018] Hasan Hosseini-Nasab, Sepideh Fereidouni, Seyyed Mohammad Taghi Fatemi Ghomi, and Mohammad Bagher Fakhrzad. Classification of facility layout problems: a review study. *The International Journal of Advanced Manufacturing Technology*, 94(1-4):957–977, 2018.

[Ismail *et al.*, 2012] Fatimah Sham Ismail, Rubiyah Yusof, and Marzuki Khalid. Optimization of electronics component placement design on pcb using self organizing genetic algorithm (soga). *Journal of intelligent Manufacturing*, 23(3):883–895, 2012.

[Khalil, 1973] Tarek M Khalil. Facilities relative allocation technique (frat). *International Journal of Production Research*, 11(2):183–194, 1973.

[Lee *et al.*, 2005] Kyu-Yeul Lee, Myung-Il Roh, and Hyuk-Su Jeong. An improved genetic algorithm for multi-floor facility layout problems having inner structure walls and passages. *Computers & Operations Research*, 32(4):879–899, 2005.

[Lee, 1967] Robert C Lee. Corelap-computerized relationship layout planning. *J. Ind. Eng.*, 18(3):195–200, 1967.

[Liu, 2012] Wentao Liu. Spgao: a simulation platform for genetic algorithm based on oop. 2012.

[Mohamed *et al.*, 2015] NMZN Mohamed, MFF Rashid, AN Rose, and WY Ting. Production layout improvement for steel fabrication works. *Journal of Industrial and Intelligent Information*, 3(2), 2015.

[Rajasekharan *et al.*, 1998] Maheswaran Rajasekharan, Brett A Peters, and Taho Yang. A genetic algorithm for facility layout design in flexible manufacturing systems. *International Journal of Production Research*, 36(1):95–110, 1998.

[Tam and Li, 1991] Kar Yan Tam and Shih Gong Li. A hierarchical approach to the facility layout problem. *The International Journal of Production Research*, 29(1):165–184, 1991.

[Tam, 1992a] Kar Yan Tam. Genetic algorithms, function optimization, and facility layout design. *European Journal of Operational Research*, 63(2):322–346, 1992.

[Tam, 1992b] Kar Yan Tam. A simulated annealing algorithm for allocating space to manufacturing cells. *The International Journal of Production Research*, 30(1):63–87, 1992.

[Tate* and Smith, 1995] David M Tate* and Alice E Smith. Unequal-area facility layout by genetic search. *IIE transactions*, 27(4):465–472, 1995.

[Van Camp *et al.*, 1992] Drew J Van Camp, Michael W Carter, and Anthony Vannelli. A nonlinear optimization approach for solving facility layout problems. *European Journal of Operational Research*, 57(2):174–189, 1992.

[Zheng *et al.*, 2006] QZ Zheng, OM Querin, and DC Barton. Geometry and sizing optimisation of discrete structure using the genetic programming method. *Structural and Multidisciplinary Optimization*, 31(6):452–461, 2006.