# Automatic Generation of LEGO Models

**Kang Juan**
B.S. (Huazhong University of Science and Technology) 2012

A THESIS SUBMITTED FOR THE DEGREE OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE
January 2015

# Acknowledgements

First, I would like to express my appreciation to Dr Low Kok Lim for his guidance and supervision. He has spent much time discussing with me about my topic, and gives helpful advices.

I would also like to thank my lab mates. I would like to thank Wenting for his implementation of the system. I would also like to thank Conrado and Philip for their valuable comments on this thesis.

# Contents

**1 Introduction** **1**

**2 Related Work** **4**

**3 Background** **7**

3.1 The LEGO problem description . . . 7

3.2 Simplified Approach . . . 8

3.3 Formulated as a combinatorial optimisation problem . . . 9

3.4 The cost function . . . 9

**4 Algorithm** **13**

4.1 General pipeline . . . 13

4.2 Step 1:Voxelizing and hollowing out . . . 14

4.3 Step 2:Balancing . . . 15

4.4 Step 3:Merging . . . 19

4.5 Step 4:Improving connections . . . 20

4.5.1 Connect disconnected components . . . 23

4.5.2 Remove weak points . . . 25

**5 Experimental Results** **32**

5.1 Balancing Results . . . 32

5.2 Improving connections Results . . . 33

5.2.1 Connect disconnected components . . . 33

5.2.2 Remove weak points . . . 33

**6 Limitations** **41**

**7 Conclusion** **43**

# List of Figures


3.1 The unit-volume LEGO brick . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.2 The yellow bricks are perpendicular to the blue bricks, which increases the stablity and strength of the LEGO model. . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3.3 The blue and yellow bricks are the original bricks. Suppose we want to add the red bricks above or below them and the red bricks form a T shape. Then we should align the center of the red bricks with the boundary formed (as shown by the dashed line) between the blue and yellow bricks. . . . . . . . . . . . . . . . . . 10

3.4 The red brick must be centered on the vertical boundary which is indicated by the dashed line and formed by the blue bricks to increase the stability. . . . . . . 11

3.5 The yellow bricks are all the neighbouring bricks of the blue brick. The red brick, although touching the corners of the blue brick, do not touch the sides and as such, are not neighbouring bricks. . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

3.6 The blue brick has a vertical direction and the yellow brick has a horizontal direction while the red brick has horizontal and vertical directions . . . . . . . . 11

3.7 The red lines are the vertical boundaries of the bricks. . . . . . . . . . . . . . . . . 12

4.1 A hollowed out layer of the Dinosaur and the voxelization result . . . . . . . . . . 15

4.2 The support region is the convex hull of the lowest layer of a model . . . . . . . . 16

4.3 Example of cut plane at top view . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

4.4 According to the relative position of center of mass and support region, we obtain all the cut planes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

4.5 We choose the most significant voxel that has the largest distance to the cut plane to add. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

4.6 We get the composed distance by combining the distances from the voxel and the center of mass to all the cut planes . . . . . . . . . . . . . . . . . . . . . . . . . 19

4.7 We get the voxel-adding space(light blue area) by considering the V-shape limitation. We give an example of generating inefficient move path by adding voxels(in purple) outside the V-shaped voxel-adding space. When we add voxels(in green) that are inside the area, the center of mass would only move inside the V-shape. 20
4.8 We get the extended voxel-adding space by generating planes that go through the center of mass and are parallel to the leftmost and rightmost cut planes. This area must contain the support region. . . . . . . . . . . . . . . . . . . . . . . . 21
4.9 The set of legal LEGO bricks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
4.10 The result of merging for a $10 \times 10$ grid . . . . . . . . . . . . . . . . . . . . . . . . . . 22
4.11 Two LEGO bricks layouts(left) and their corresponding graph representations . . 22
4.12 An example for weak points . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
4.13 The relative positions of bricks coming from different components at the same layer. We use different color to represent different components. . . . . . . . . . . 23
4.14 There are five disconnected components(in different colours) in the figure. The two components in green and in mauve only touch each other with bricks A and B. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
4.15 The connect directions according to the relative positions . . . . . . . . . . . . . . . 25
4.16 The added voxels (in purple shadow) at different situations of the brick and its neighbor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
4.17 Two different solutions of connecting a pair of knobs. The bricks in blue are the added bricks. We choose the solution of a), because b) will weaken the model. . . 27
4.18 An example of extending a brick. We will extend the red brick and the grey bricks which are the exist bricks. we get 6 possible knobs by extending the original red brick. The yellow rectangle represents the position of the knob and the blue brick is the brick that contains the knob and requires the fewest number of voxels. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
4.19 Two ways to add new bricks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
4.20 An example of all the possible optimal solutions to connect a pair of knobs. The grey bricks are the knobs to connect and the blue bricks are the added bricks. The number of added voxels is 7. . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

4.21 An example of the possible optimal solutions at all the position by using the dynamic programming technique. The positive number in each position represents the amount of added voxels of the optimal solution and -1 represents no solution in this position. We get the optimal solution of connecting the source knob to the target knob at the bottom-left position which is 7. . . . . . . . . . . . 30
4.22 At a), we show an example that could not generate the optimal solution because of the existence of original bricks. At b), we extend the source knob by adding a brick above it so that we could across the obstacle and then generate an optimal solution. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

5.1 We give an example of balancing a dinosaur model. The resolution of a) is 32. The blue voxels is the newly added voxels after balancing in c). In d), the black polygon is the support region and the blue dot is the projection of the original centre of mass onto the plane of support region in gravity direction as in blue line and the red dot is the result after balancing. . . . . . . . . . . . . . . . . . . 34
5.2 We give an example of balancing a three-overlaid-spheres model. The resolution of a) is 64. The blue voxels is the newly added voxels after balancing in c). In d), the black polygon is the support region and the blue dot is the projection of the original centre of mass onto the plane of support region in gravity direction as in blue line and the red dot is the result after balancing. . . . . . . . . . . . . . . 35
5.3 We give an example of balancing a bear model. The resolution of c) is 16. In order to state the correctness of our algorithm, we build the actual LEGO model before and after balancing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
5.4 We give an example of balancing a pig model. The resolution of c) is 16. In order to state the correctness of our algorithm, we build the actual LEGO model before and after balancing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
5.5 We give an example of connecting multiple components together. a) shows the original disconnected solution which has 4 disconnected components where each has a different color. In b), the red voxels are the original existing voxels and the blue voxels are the added voxels to connect different components. c) is our fixed connected solution after we have added the needed voxels and rerun the merge algorithm. d) is the graph for the connected solution. . . . . . . . . . . 38

5.6 We give an example of removing weak points. a) shows the original solution which has 7 weak points(red bricks). c) shows the corresponding graph for the original solution and the nodes marked by red circles are the weak points. b) is our solution to remove these weak points. We could remove a few weak points which are in adjacent layers at the same time. As shown in d), when we deal with the nodes marked by blue(yellow) triangles, we can remove these weak points marked by blue(yellow) circles simultaneously. . . . . . . . . . . . . . . . . 39
5.7 We give another example of removing weak points. . . . . . . . . . . . . . . . . . . . 40

6.1 Although the centre of mass of the bird is inside the support region, it might break at the thin leg part. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42

# Abstract

In this thesis, we focus on the problem of automatically generating LEGO models similar in appearance to the given 3D digital models. Building 3D LEGO models manually requires considerable trial-and-error even for small models. Even for experienced designers, it is not easy to generate balanced and structurally strong LEGO models on a large scale. Our system automatically generates a set of instructions for building balanced and structurally strong LEGO models from 3D digital models, which could simplify the fabrication of LEGO models.

In our algorithm, we first voxelize the input 3D mesh into the smallest bricks which are $1 \times 1$ in size, and then hollow out the voxelized model in order to reduce the time and cost of building the LEGO model. Secondly, we balance the object by adding the extra voxels inside the voxelized and hollowed model. Then, we merge the voxels into larger bricks until no further merging is possible. We prefer larger bricks because this reduces the brick count and increases structural strength. Finally, we reduce the problem of improving structural strength to improving connections between bricks. We use a graph to represent the connections symbolically. After that we find the badly connected parts and repair them by adding extra bricks while keeping as close as possible to the original model in volume. After these processes, we print out the set of instructions for building the LEGO model.

The user can control the scale of the built LEGO model by setting the resolution of voxelization. Our system can render LEGO models in both bricks and graph representations. The user can choose the thickness of the shell of hollowing and view the model layer by layer. The experimental results for both real and virtual models show that our algorithm can automatically generate balanced and structurally strong LEGO models.

# Chapter 1

# Introduction

Recently, the generation of 3D models for fabrication has attracted great interest from the computer graphics community. The idea of moving creations from virtual space to physical reality is intriguing. Usually the physical realization of 3D models is done by sophisticated milling and rapid prototyping processes. However, these state-of-the-art technologies usually require expensive equipment such as a laser cutter or 3D printer. On the other hand, the famous LEGO brick construction system, which is comparatively cheap, could also be used to generate physical versions of digital models. Thus, we focus on the fabrication of LEGO models in this thesis.

Building 3D LEGO models manually, however, requires considerable trial-and-error even for small models and it can be very cumbersome and time-consuming to generate balanced and structurally strong LEGO models at a large scale even for experienced designers. As such, the LEGO company has twice openly proposed the problem to the scientific community in 1998 [1] and again in 2001 [2] specifically, "Given any 3D body, how can it be built from LEGO bricks?". Because of the variety of types of bricks available and the multiple ways in which an object can be built from these bricks, the problem then becomes intractable.

The LEGO construction problem can be treated as the area filling problem in three dimensions which is considered to be NP-complete [1]. In order to solve it, we need to make some reasonable assumptions and simplify the problem. The first assumption is to use bricks with the same height. Under this assumption, we only consider one layer at a time and build the model layer by layer. In this way, we transform the problem from three dimensions to two dimensions. However, we can not just consider each layer separately and then join these separate layers together to build the final model, since it may lead to a LEGO sculpture that is

not connected. Thus, we need to take into account the connections between layers. As there are multiple ways to construct a LEGO model, we prefer "more stable" models [1], which are made with larger bricks that have more knobs connected to each other.

Using this assumption, we can solve the problem as an optimisation problem. We minimise the cost function of building the LEGO model to generate models with high connectivity. A number of researchers have used techniques such as simulated annealing [1], evolutionary algorithms [2], beam search [3] and cellular automata [4] to solve the optimisation problem.

However, minimising the cost function which represents the connectivity of bricks can not ensure that the actual LEGO model is strong enough to stand without collapsing. We still need extra work to explicitly consider the strength of the model. The first thing to consider is making the whole model stand without falling down, or in other words, keeping the model balanced. In addition, we need to make sure that there are no bricks that will break apart from the rest of the model under the force and stress between bricks when we build the actual LEGO model. Since the distribution of the force and stress between bricks depends on the structure of the model, we need to investigate the actual structure of the model explicitly and make sure the model is structurally strong.

Therefore, in this thesis we focus on the problems of constructing balanced and structurally strong LEGO models.

Since balance is a basic requirement to make our LEGO model strong, we first make sure that the model is balanced, or in other words, is in static equilibrium. That is, we consider the weight of each brick and the gravity to ensure that the model can stand stably. As such, we need to ensure that the projection of the center of mass of the LEGO model along the gravity direction is inside the base of support. As we intend to use as few bricks as possible, we hollow out the interior space. Thus, if the center of mass is not properly placed, we can balance it by adding bricks inside the hollowed portion.

Secondly, we reduce the problem of ensuring the structural strength of the model to checking the connectivity of each brick. As the LEGO model is built by joining different bricks together, the connections between bricks is likely to break, causing the part of the built model to collapse. The ideal way to solve this problem is to analyse the force and stress between bricks and then change the structure of the model to make it strong enough. However, physical analysis of the force and stress between bricks is quite complicated. Therefore, we will use a graph to represent the actual connections between bricks symbolically and analyse these structures to find the badly connected parts. Then repair them by adding extra bricks while keeping as close as possible to the original model in volume. After these processes, we can

improve the connections of the model at the badly connected parts and build a model that is structurally stronger.

# Chapter 2

# Related Work

In order to solve the LEGO construction problem, Gower *et al* [1] gave a formulation of the problem. Given the input 3D model, they first generated a *legoised* model, which could be represented as a 3D matrix containing one in the places where unit-volume LEGO bricks should be put and zeros where there should be empty spaces. Then the task became to make an algorithm which took the *legoised* 3D model as input and produced a way of constructing the model from actual LEGO bricks so that the model "stands" connected. As Gower *et al* [1] stated, since there were more than one way of building any model of even quite small size, they would seek "more stable" models which were made with bigger bricks and with bricks that have more studs connected to other bricks. Under these assumptions, they formulated the problem as a combinatorial optimization problem. They proposed a cost function which represented the connectivity of bricks. Actually, the cost function was a map from the set of all allowable brick configurations into the real numbers. They could decide whether a brick placement was good or not by calculating the cost function.

After Gower *et al* [1] gave the formulation, there were many studies on solving the optimization problem. After giving the formulation, Gower *et al* [1] described a few methods to optimise the cost function. One method was to use a pattern filling library. The library would include the optimal placement of bricks for known primitive shapes. Therefore, they could try to fill each layer with these primitive shapes and then find optimal solutions in the library. As they mentioned, it should be careful when consecutive layers have the same size as it would lead to an unstable and disconnected LEGO model if these layers were built in the same way. Another intuitive method was using local search. That was, they would place one or a few bricks at a time by searching through the possible brick placements for a small

subregion. After dealing with one subregion, they would continue to find the next new subregion that overlapped with the previous subregion until the entire layer was filled. The next method was by using simulated annealing, which they think was the most promising method. They would first divide each layer of the LEGO model into smaller sized subregions and each subregion was treated separately using simulated annealing. After that they would fill each subregion with an arbitrary initial brick placement as the starting state and then iteratively moved some small number of bricks and replaced them by a few new bricks. The simulated annealing method would lead to near optimal solution for each subregion. Although they suggested these methods to solve the optimisation problem, they did not implement and test them. The second attempt used an evolutionary algorithm to solve the optimisation problem [2]. They evaluated the performance of the evolutionary algorithm itself, but did not consider whether the solutions could lead to constructable LEGO models or not. They also reported that the required time for optimization was 5 to 11 hours. The next attempt used the beam search technique, but they did not provide data for the experiment results [3]. After that, [4] proposed an approach that was based on cellular automata and compared it with the existing methods. They used a merge and split method to generate the brick configurations. The optimization time was about a few minutes.

However, all the methods described above did not consider the strength of the LEGO model explicitly. They neither considered whether the result LEGO model could be balanced nor the structural strength of the model to avoid collapse at the thin parts. Recently, [5] had proposed a graph-based algorithm to explicitly explore the structure strength of the resulting LEGO model. They reduced the problem of improving structural strength to improving connections between bricks. They used graph to represent the connections between bricks symbolically and then found the badly connected parts by analysing the graph. They defined the *weak articulation point* to represent badly connected parts. After that they removed these *weak articulation points* by changing the brick layout. However, for some specific input models, there may still have disconnected components or thin regions where *weak articulation points* cannot be removed by only changing the brick layout. To resolve this problem, the original model need to be changed. However, since changing the original model would increase the complexity of the LEGO construction problem, they didn't attempt to solve it. Our work in the thesis would solve this problem. We would improve the structure strength of the resulting LEGO model while changing the original model. As there were multiple ways to change the model, we would prefer a way that could remove all the *weak articulation points* while keeping as close as possible to the original model in volume. We would also keep the model balanced

as it was a basic requirement to make the resulting LEGO model strong. None of the previous work considered the balance problem while solving the LEGO construction problem.

There were also plenty of studies on improving strength for other kinds of physical model fabrications. The process of physical model fabrication was not straightforward, and it need to follow several design constraints. A lot of recent papers tried to assist users in designing models that would be physically sound after fabrication. [6] presented a system that would perform real time simulation of the physics of a chair so that the users could visualize the equilibrium or stability issues for the final models. [7] extended the principle of [6] and gave suggestions for users to modify their current designs so that it would meet the requirements of stability and durability. [8] proposed a method that iterated between carving and deformation to reach equilibrium constraints. Their approach redistributed weights by jointly optimizing for the interior and the surface shape of an object. They excluded the use of struts or heavy pedestals to keep the shape as close as possible to the original model. In addition, the structural analysis was also a well-studied subject in disciplines such as mechanical engineering but the printability of virtual 3D objects was addressed only in a limited extent within the context of Computer Graphics [9]. In general, the structural analysis problem can be addressed by using methods that were based on shape optimization [10, 11], which modified the shape of the 3D object to minimize the objective function. However, this method was computationally very expensive even for moderately simple 3D models. [12] proposed an automatic method to detect the structural issues for printed models. The method would check whether the thin and thick parts were strong enough to support its attached parts by using a set of basic geometric rules. [9] presented a system that could detect and correct the major structural problems for 3D models before they were printed. They attempted to minimally modify the models to reinforce its physical realization. They first detected the major printability issues of the input 3D models by printability assessment. Then, they corrected the model with three different approaches: thickening thin parts of the object, inserting struts to support remaining excessive weight, and hollowing out parts to reduce the weight and stress on certain parts. However, for all the methods mentioned above, we could not apply it to improve the strength of our LEGO models directly as the LEGO models were constructed by separate bricks.

# Chapter 3

# Background

As mentioned in the previous chapter, Gower *et al* [1] gave a formulation of the LEGO construction problem. All the following studies including our work for the LEGO construction problem was based on this formulation. Therefore, in this chapter we provide more details of the problem formulation.

## 3.1 The LEGO problem description

The LEGO construction problem can be described as: given any 3D object, generate the instructions to build a LEGO sculpture for that object.

In order to make the problem more precise, Gower *et al* [1] gave more detailed assumptions about the problem:

- They assumed that they have a *legoised* representation of the real-world object as input. The *legoised* representation was a 3D matrix that contained ones for spaces that contained unit-volume Lego bricks and zeros for empty spaces. A unit-volume LEGO brick was the smallest possible brick and has only one knob on the top for connection with other bricks(see Figure 3.1).

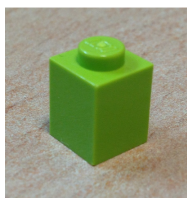


Figure 3.1: The unit-volume LEGO brick

- They only used the "family" LEGO bricks and LEGO DUPLO bricks as the legal bricks to build the model. These permitted bricks' dimensions were integer multiples of the dimensions of the unit-volume LEGO brick in each direction. Thus, each of the permitted bricks can be replaced by several unit-volume bricks which were stacked together . The use of DUPLO bricks can reduce the overall cost of the model, since DUPLO bricks were larger and a few large bricks cost less than many small bricks. However, these DUPLO bricks can only be connected to family bricks with even length and width.

- Since there were multiple ways in which an object can be built from these bricks, they preferred "more stable" models. More stable models were models that were made with larger bricks and with bricks that had more knobs connected to other bricks.

- They could also allow a few changes for the original model when building the actual LEGO model. That was they could change the model by very small amounts if the change could bring benefits to other aspects, such as increasing the strength.

- In order to reduce the time and cost of building the actual LEGO model, the object should be hollowed out if the model was large enough.

- In the first stage, they could ignore the colour of the bricks to generate a monochromatic LEGO model. But at the later stages, they should consider the colour of the outside of the model.

- Finally, the algorithm should give an acceptable solution within a reasonable time. In addition, the algorithm should be able to deal with large models, since the original purpose of the program was to find a solution for building LEGO models which were used in LEGO theme parks or at exhibitions for advertising purposes.

## 3.2 Simplified Approach

Even with these assumptions for the LEGO construction problem, it was still a large and complicated problem. Therefore, it was still necessary to simplify it. Gower *et al* [1] gave more detailed assumptions to simplify the problem:

- Don't use DUPLO bricks. Although the use of DUPLO bricks could reduce the overall cost of the model, it would increase the complexity of the problem, since DUPLO bricks could only be connected to bricks that have even numbered lengths and widths and

other bricks must also have a height of at least three unit-volume bricks to be connected to a DUPLO brick.

- Only use bricks with the same height. In this way, they could transform the problem from a three-dimensional problem to a two-dimensional one. This could be achieved by dividing the 3D model into different layers and only considering bricks one layer at a time; building the model layer by layer.
- Keep the size and shape of the given *legolised* representation. Although changing the inside of the *legolised* representation could potentially reduce the number of bricks used and increase the stability of the model, determining where and how to change it and what the global effects would be would increase the complexity. The purpose of them stating this assumption was to reduce the complexity of the problem, but we would consider changing the original shape to increase structure strength in this thesis.

## 3.3 Formulated as a combinatorial optimisation problem

After the assumptions and simplifications, Gower *et al* [1] used a cost function to represent the overall structure for the LEGO model. Then they formulated the LEGO problem as a combinatorial optimisation problem: minimise the overall cost function which mapped a finite discrete set(the set of all possible brick configurations) into the real numbers .

To solve the combinatorial optimisation problem, they need to calculate a cost function to decide whether a brick placement was good or not. The concept of "good" brick placements come from the experience of the LEGO designers who play with LEGO. They assumed that "the whole was not more than the sum of its parts" in this problem so that they can calculate the overall cost function by adding up all the cost functions for individual bricks.

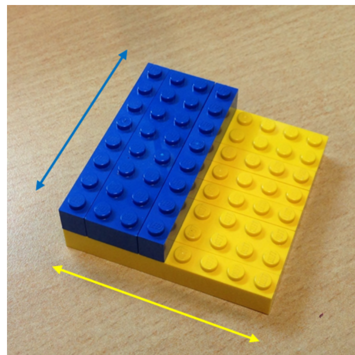


Figure 3.2: The yellow bricks are perpendicular to the blue bricks, which increases the stablity and strength of the LEGO model.

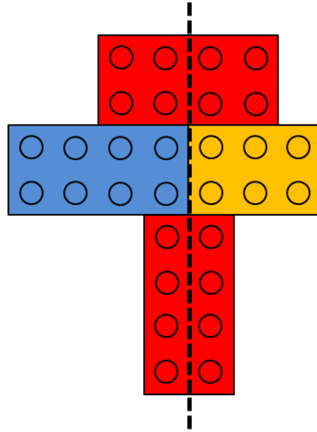


Figure 3.3: The blue and yellow bricks are the original bricks. Suppose we want to add the red bricks above or below them and the red bricks form a T shape. Then we should align the center of the red bricks with the boundary formed (as shown by the dashed line) between the blue and yellow bricks.

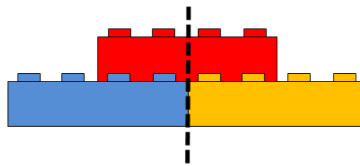


Figure 3.4: The red brick must be centered on the vertical boundary which is indicated by the dashed line and formed by the blue bricks to increase the stability.

## 3.4 The cost function

Although they could get the cost function by imitating the learning of humans through the use of neural networks [13], it could make the problem too complicated. So, instead of using a neural network, they made use of heuristics to calculate a cost function for the LEGO model.

The following heuristics were given by Gower *et al* [1] and the aim of these heuristics was to help ensure a stable and connected LEGO model.

- Factor 1: A high percentage of the area of each brick should be covered, above and below, by other bricks. This could help to increase the stability and also reduce the number of bricks that were not attached to the LEGO model as much as possible.

- Factor 2: The use of larger bricks. Using larger bricks could not only increase the overall stability and connections, but also reduce the cost of the LEGO model as a few larger bricks costed less than many smaller bricks.

- Factor 3: Alternating directionality of bricks in consecutive layers. In this way, a brick could connect to more bricks in the previous layer and this increased the stability and strength of the LEGO model (see Figure 3.2).

- Factor 4: A high percentage of the vertical boundaries of each brick should be covered by bricks in the above and below layers. This heuristic tried to avoid a brick being placed such that its boundaries matched the boundaries of bricks in the previous layer, which would potentially generate an unconnected LEGO model.

- Factor 5: When either the short or the long side of the brick formed a T-shaped boundary with its neighbouring bricks, the middle of the side should be placed at the boundary of the neighbouring bricks(see Figure 3.3).

- Factor 6: If a brick covered a boundary of bricks in the previous lower layer, it should be placed such that it was centered on the boundary(see Figure 3.4)

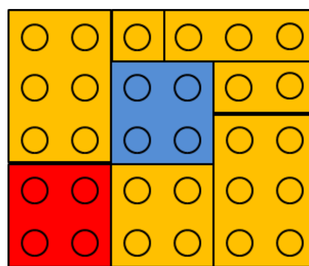


Figure 3.5: The yellow bricks are all the neighbouring bricks of the blue brick. The red brick, although touching the corners of the blue brick, do not touch the sides and as such, are not neighbouring bricks.

After giving the heuristics above, Gower *et al* [1] suggested using the following cost function:

$$P = C_1P_1 + C_2P_2 + C_3P_3 + C_4P_4 \tag{3.1}$$

where $C_i$'s were weighting constants and $P_i$'s were described as follows:

- $P_1$ corresponds to perpendicularity: This term was used to penalise the placement of bricks which do not fit with factor 3 above.

- $P_2$ relates to vertical boundaries: This term was used to penalise bricks which go against factor 4 above.

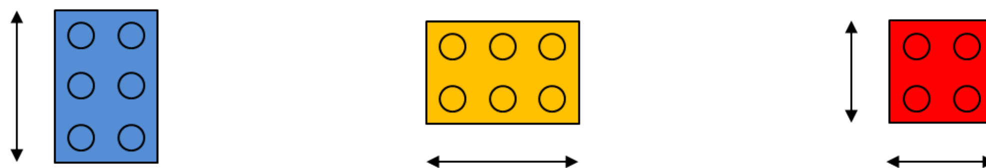


Figure 3.6: The blue brick has a vertical direction and the yellow brick has a horizontal direction while the red brick has horizontal and vertical directions

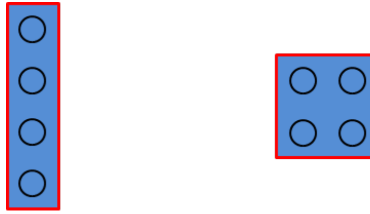


Figure 3.7: The red lines are the vertical boundaries of the bricks.

- $P_3$ relates to horizontal alignment: This was used to penalise bricks placed in a way that do not fit with factor 5.
- $P_4$ was used to favour the use of larger bricks explicitly which correspondences to factor 2.

Although factor 1 was important, there was not much they could do since they assumed that the shape of the model was fixed.

In order to understand the heuristics more naturally, we need to introduce some terminology which was from [14].

- Attached brick: if a brick is joined to at least one other LEGO brick in the LEGO model, it is treated as an attached brick.
- Neighbouring bricks: The neighbouring bricks of a brick are those bricks that touch at least one side of the brick directly and are within the same layer(see Figure 3.5).
- Brick direction: The direction of a brick depends on whether the brick is lying vertically or horizontally when looked at from the top. The direction could be horizontal or vertical. Rectangular bricks are defined to only have one direction while square bricks have both horizontal and vertical directions(see Figure 3.6)
- Parallel and perpendicular bricks: If two bricks have the same direction then they are parallel, otherwise they are perpendicular.
- Brick boundaries: The sides of a brick create a border which could distinguish the brick from its neighbouring bricks when we look at it from the top. The lines that define the borders are called the vertical boundaries [1] of the brick(see Figure3.7)

# Chapter 4

# Algorithm

In this chapter, we will describe our algorithm for automatically generating balanced and structurally strong LEGO models which are similar in appearance to the given 3D digital models. We first voxelize the input 3D model into the smallest bricks which are $1 \times 1$ in size, and then hollow out the voxelized model in order to reduce the time and cost of building the LEGO model. Secondly, we balance the object by adding the extra voxels inside the voxelized and hollowed model. Then, we merge the voxels into larger bricks until no further merging is possible. We prefer larger bricks because this reduces the brick count and increases structural strength. Finally, we reduce the problem of improving structural strength to improving connections between bricks. We use a graph to represent the connections symbolically. After that we find the badly connected parts and repair them by adding extra bricks while keeping as cloase as possible to the original model in volume. We will first give a brief description of the general pipeline of our algorithm and then explain each step in more detail.

## 4.1 General pipeline

In this section, we give a brief description of the general pipeline of our solution. The steps are the following:

- Step 1: Voxelize and hollow out. As in the previous work, we first represent the 3D digital model as $1 \times 1$ LEGO bricks. We treat these $1 \times 1$ LEGO bricks as voxels and call this process of transforming a 3D mesh representation into a discrete set of voxels voxelization. In order to reduce the time and cost of building the actual LEGO model, we hollow it out after the voxelization.

- Step 2: Balance. As mentioned in the introduction, we would like to generate a stable LEGO model. We will consider the weight of each brick and the gravity to check whether the LEGO model could stand on its own without falling. If the center of mass of the LEGO model is not inside the support region, then it is not balanced. In order to balance the model, we add voxels inside the hollowed and voxelized model so that we can move the center of mass into the support region. Since there are multiple ways to add voxels, we try to find a way that can balance the model more quickly while using as few voxels as possible.
- Step 3: Merge. After voxelization and balancing, we merge the voxels into bricks as large as possible and within the legal brick sets until no further merging is possible. We prefer using larger bricks because it will reduce the brick count and increase connectivity. As in previous work, we use the cost function to decide how to merge so that we can generate LEGO models with large connectivity.
- Step 4: Improve connections. After the merge step, we are still not sure whether the LEGO model is completely and strongly connected or just weakly connected and even disconnected. Therefore, we use a graph to represent the actual connections between the LEGO bricks. We can easily check whether the model is completely connected by observing the graph. If we find some bricks that should be connected to be separated, we join them together by adding extra bricks. After that, we check whether the LEGO model is weakly connected. To make it clearer, we first need to give a precise definition for the weakly connected points. The definition should correspond to the actual weak connected part of the real LEGO model. If we find the weak connected points inside the LEGO model, we will repair it by adding extra bricks. The added bricks should be as few as possible so that we can keep the original model as much as possible.

After these processes, we generate a constructable and structurally strong LEGO model. The following sections explain each step in more detail.

## 4.2 Step 1:Voxelizing and hollowing out

For the voxelization part, we use the open source program called binvox [15] which reads a 3D model file and rasterizes it into a binary 3D voxel grid. The binvox uses the parity count method and a slight variation of ray stabbing method described in [16].

We should be careful about the choice of voxel resolution. A higher resolution can generate a better approximation of the original model, but will cost more bricks and time to build the model. A lower resolution may lead to the loss of thin parts. We can adjust and choose a proper resolution in the binvox software.

For the hollowing out part, we first choose a shell size and then we just delete the inside voxels such that their distances to the outside of the model in all six directions are larger than the shell size. Figure 4.1 shows one layer of a hollowed Dinosaur with shell size 2 and the voxelized result.

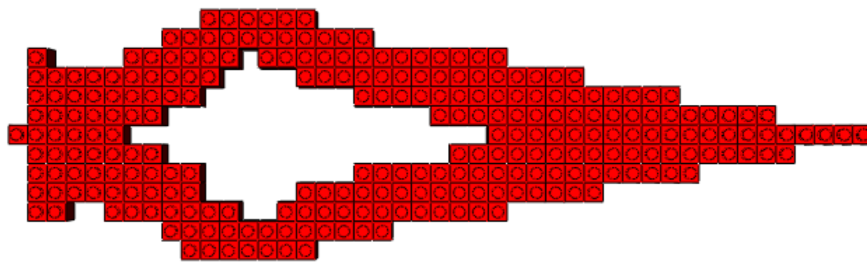


Figure 4.1: A hollowed out layer of the Dinosaur and the voxelization result

## 4.3 Step 2:Balancing

As stated previously, the model is balanced only when the center of mass is projected down onto the support region. Otherwise, it can not stand on its own without falling. In order to balance the model, we add voxels inside the hollowed and voxelized model to move the center of mass into the support region.

We compute the center of mass of the model by calculating the average coordinates of all the voxels. The projection along the gravity direction will be the $x, y$ coordinates while $z$ represents the height of the center of mass. The support region is the convex hull of all the voxels at the lowest layer that touches the ground. For example, in Fig 4.2, we get the support region by computing the convex hull of all the voxels at the lowest layer.

In order to move the center of mass into the support region, we need to add extra voxels inside the hollowed model. As there are multiple ways to add voxels, we try to find a way that can balance the model more quickly while using as few voxels as possible.

As inspired by [8], when given the projection of the center of mass and the convex hull of support region, we can get a few cut planes that could accelerate our balancing process. As shown in Fig 4.3, for the plane $p_1$ that goes through the edge $e_1$ of the support region

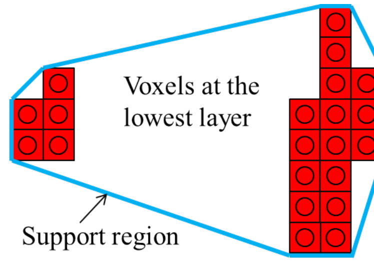



Figure 4.2: The support region is the convex hull of the lowest layer of a model

and is parallel to the direction of gravity, the center of mass and the support region are on different sides of the plane. Even if we add a voxel on the side of the plane that contains the center of mass, it will not help move the center of mass into the support region. However, if we add voxels on the other half-space that contains the support region, it could potentially move the center of mass closer to the support region. Therefore, we define the plane $p_1$ as the cut plane and we only consider adding voxels on certain sides of the cut plane. Using the relative positions of the center of mass and support region, we can get all the cut planes. For example, in Fig 4.4, the potential cut planes are $p_0, p_1, p_2, p_3, p_4, p_5$ which correspond to all the edges of the support region. Since the center of mass is on the side of $p_0, p_1, p_2$ that do not contain the support region, we define $p_0, p_1, p_2$ as the cut planes. If the center of mass were on the same side of the support region for all the potential cut planes, then we would know that the center of mass must be inside the support region.

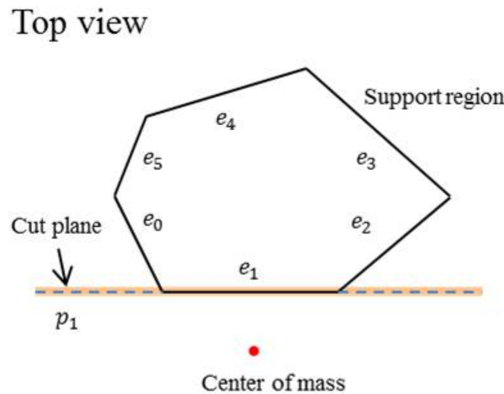



Figure 4.3: Example of cut plane at top view

In order to accelerate the balancing process, we choose the most significant voxel to add every time. When there is only one cut plane, we choose the voxel that has the largest distance to the cut plane such as in Fig 4.5, as it will generate the largest movement of moving the center of mass closer to the cut plane. When there are a few cut planes, we choose the voxel with the largest composed distance. In Fig 4.6, the composed distance is

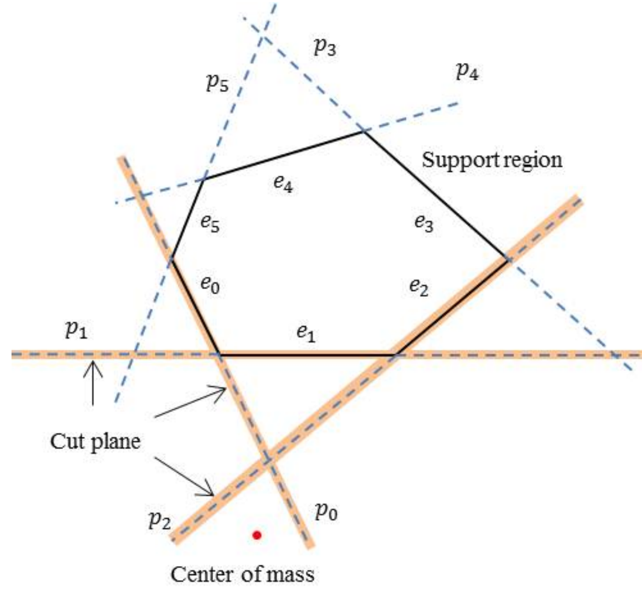



Figure 4.4: According to the relative position of center of mass and support region, we obtain all the cut planes

$d_c = r_0 \times d_0 + r_1 \times d_1 + r_2 \times d_2$, where $d_i$ is the distance from the voxel to the cut plane $p_i$ and $r_i = c_i/(c_0 + c_1 + c_2)$ where $c_i$ is the distance from the center of mass to the cut plane $p_i$. In this composed distance, we consider the distances from the voxel to all the cut planes as our goal is moving the center of mass to the other side of all the cut planes. In this way, we add up all the distances $d_i$ to form the composed distance $d_c$. In addition, as whether the center of mass is inside the support region or not depends on the largest distance from the center of mass to all the cut planes, we give more priority to the cut plane that has a larger distance to the center of mass. Since the larger $c_i$ is, the larger $r_i$ is as well, we use $r_i$ to weight each distance $d_i$. In fact, the distance in the case of one cut plane is included in the composed distance. Therefore, we can generate a general equation to represent the composed distance $d_c$ as follows:

$$d_c = \sum_{i=0}^{N-1} r_i \times d_i \tag{4.1}$$

where $N$ is the number of cut planes, $d_i$ is the distance from the voxel to the cut plane and

$$r_i = \frac{c_i}{\sum_{i=0}^{N-1} c_i} \tag{4.2}$$

where $c_i$ is the distance from the center of mass to the cut plane.

As stated previously, we use the cut planes to limit the voxel-adding space so that we do not add unnecessary voxels. However, in order to generate more efficient movement of the

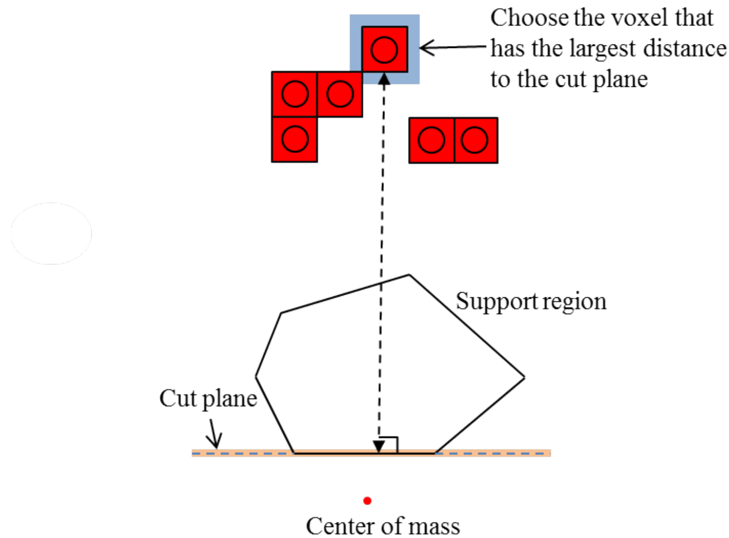



Figure 4.5: We choose the most significant voxel that has the largest distance to the cut plane to add.

center of mass, we add another two lines to limit the voxel-adding space in the first stage. As in Fig 4.7, we find the leftmost and rightmost endpoints of edges which correspond to the cut planes of the support region and create two lines by connecting the center of mass with these two endpoints separately, which generates a V-shaped area. Finally, we get the voxel-adding space by combining the sides of the cut planes that contain the support region and the V-shaped area. The voxel-adding space is the light blue area in Fig 4.7. For example, in Fig 4.7, with the V-shape limitation, adding the series of green voxels in the light blue area could move the center of mass along the green dotted lines. This generates more efficient movement. However, if we add voxels outside the V-shape such as the purple voxels, the movement might be like the purple dotted path which is inefficient.

Therefore, the balancing process is as follows:

- 1: We compute the support region and the original center of mass of the hollowed voxelized model.
- 2: According to the relative position of support region and center of mass, we estimate whether the model is balanced. If it is balanced, we stop. Otherwise, we find the voxel-adding space by computing the cut planes and the V-shaped area or the extended area.
- 3: We find the voxel with the largest composited distance to add and recompute the center of mass and go to step2. However, if we do not find any voxels to add in this step, then we go to step 4.

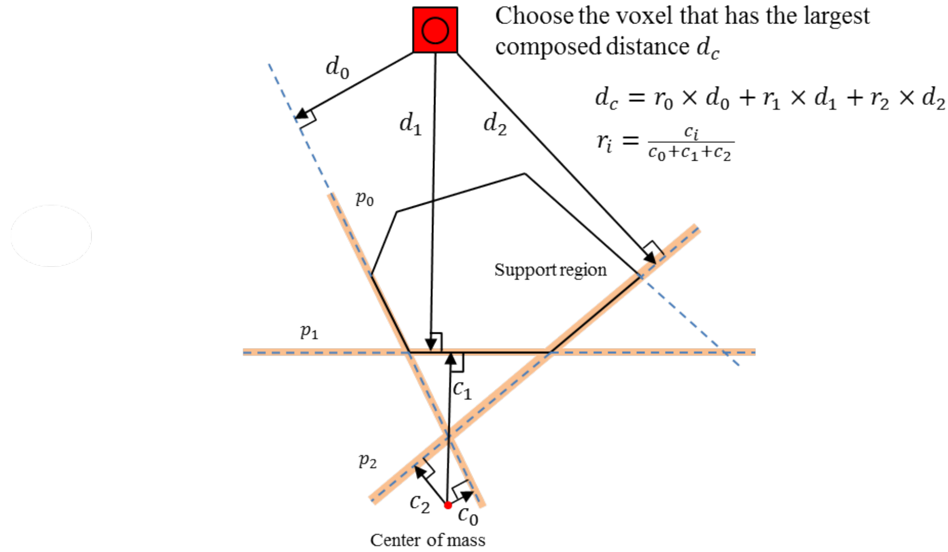



Figure 4.6: We get the composed distance by combining the distances from the voxel and the center of mass to all the cut planes

- 4: If the recent voxel-adding space is generated by the V-shaped area, we extend it as shown in Fig 4.8 and go to step 2. We find the left and rightmost cut planes. Starting from the center of mass, we draw two lines that are parallel to these planes to form the extended area. It is also possible that there is only one cut plane, in which case we only draw a line containing the center of mass that is parallel to this cut plane. In both cases, the extended area is on the side that contains the support region. If the recent voxel-adding space has already been extended, then we know that we have already added all the possible voxels to balance the model and estimated that the model can not be balanced, since if we add voxels outside the extended area, the center of mass can only get farther away from the support region.

## 4.4 Step 3:Merging

After voxelization and balancing, the next step is to merge the voxels into larger bricks. We use the randomized greedy merge algorithm [5] to merge the bricks with their neighbours into the legal brick sets shown in Figure 4.9. In order to reduce the brick count and increase connectivity, the algorithm merges bricks into larger bricks until no further merge is possible. The idea of merging in this algorithm is actually inspired by the work of [4]. Their algorithm is as follows [5]:

- 1: Choose a brick in the model at random.

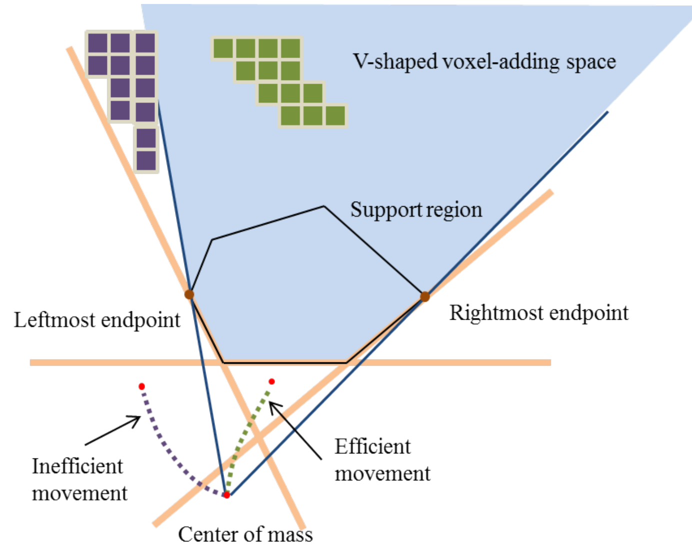



Figure 4.7: We get the voxel-adding space(light blue area) by considering the V-shape limitation. We give an example of generating inefficient move path by adding voxels(in purple) outside the V-shaped voxel-adding space. When we add voxels(in green) that are inside the area, the center of mass would only move inside the V-shape.

- 2: Find the legal set of neighbours with which the brick can be merged.
- 3: Select the neighbour with the lowest cost value and merge.
- 4: Go to Step 2 until there are no more mergeable neighbours.
- 5: Go to Step 1 until no bricks can merge.

The legal set of bricks we use in Step 2 is shown in Figure 4.9. But arbitrary sets could also work here if they have the same height. For Step 3, we prefer the neighbouring brick that, when merged with the current brick, would create the most connections. Two bricks are connected when they are on adjacent layers and have at least one knob overlapping. If two merges have the same connections, we just choose one of them randomly. Figure 4.10 shows one result of the algorithm for a $10 \times 10$ grid(we do not show the upper and lower layers).

## 4.5 Step 4:Improving connections

As mentioned previously, we prefer to choose a construction with large connectivity so that we can generate a constructable and structurally strong LEGO model. This means that the more bricks are connected to each other, the structurally stronger it would be. In this way, we use a graph where each brick is represented by a vertex and each connection between two bricks is represented by an edge to represent the LEGO brick model[5]. With this mapping, we

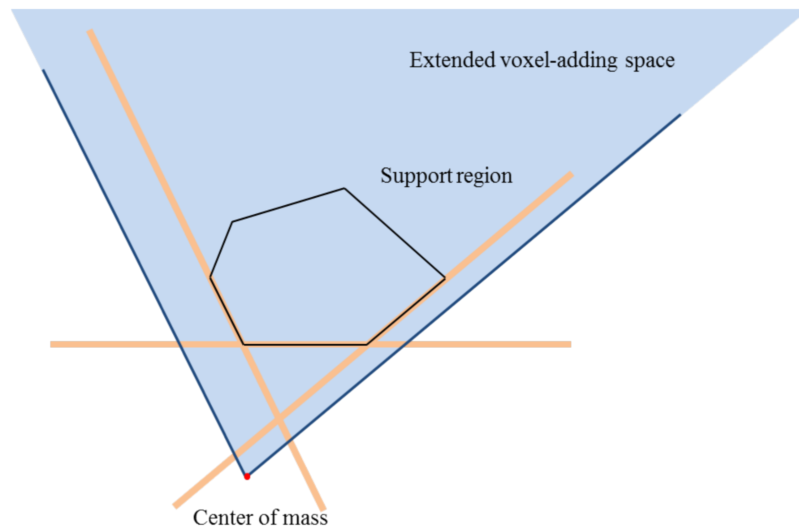



Figure 4.8: We get the extended voxel-adding space by generating planes that go through the center of mass and are parallel to the leftmost and rightmost cut planes. This area must contain the support region.

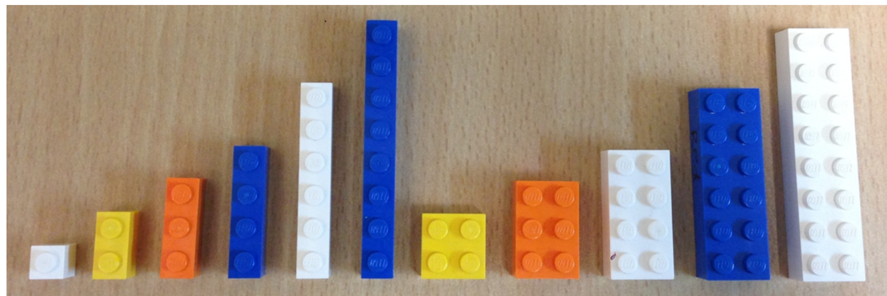


Figure 4.9: The set of legal LEGO bricks

can analyse the connections of a LEGO model by directly analysing its graph representation. Figure 4.11 shows the example of the LEGO brick layouts and their corresponding graph representations.

We can easily check whether the model is completely connected by observing the graph. If the number of connected components in the graph is larger than expected, then we can predict that some bricks in the LEGO model will fall off as they are not connected to the rest. Since if two non-trivial subgraphs of size greater than 2 are only connected to each other by one brick, the LEGO model can be broken easily, we define the brick that connects these two subgraphs as a weak articulation point[5] or we call it a weak point. In Figure 4.12, we give an example for weak points which are marked by red circles under our definition. If we find disconnected components or weak points in the graph, we need to remove it to improve the connections.

In [5], they tried to improve connectivity by changing the brick layout. They first split

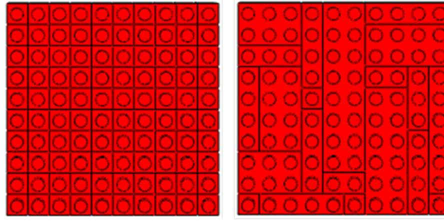


Figure 4.10: The result of merging for a $10 \times 10$ grid

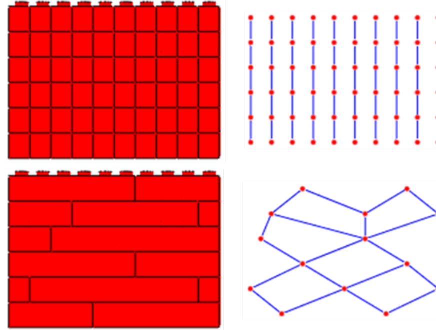


Figure 4.11: Two LEGO bricks layouts(left) and their corresponding graph representations

each of the bricks that were weak points or neighbouring a weak point into $1 \times 1$ bricks. After that they ran the merge algorithm again but changed the cost function to a random cost function. They preformed the split-and-merge process iteratively until the number of the connected components and weak points no longer reduce. However, such a process might not solve connection problems. It is possible that there are thin parts where weak points can not be removed(such as the connection between the tail of a dog and its body), or disconnected components for specific voxelizations. In these cases, we will need to change the original model by adding extra bricks to generate a constructable and structurally strong LEGO model. Since we can improve the structural strength of the model by adding as many extra bricks as needed, we need to find a balance between improving connections and keeping the original model. In this thesis, we will find a way that only needs to add necessary bricks to solve the connection problems. In this way, the added volume for the LEGO model will be as little as possible so that we can keep the original model as much as possible.

We will first run the split-and-merge algorithm. If there are still connection problems, we use the following steps to solve them. We will connect disconnected components first and then remove weak points.

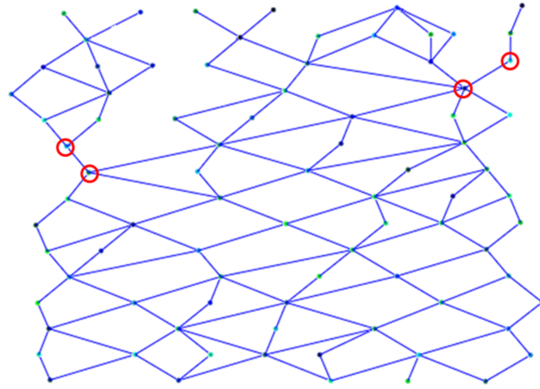
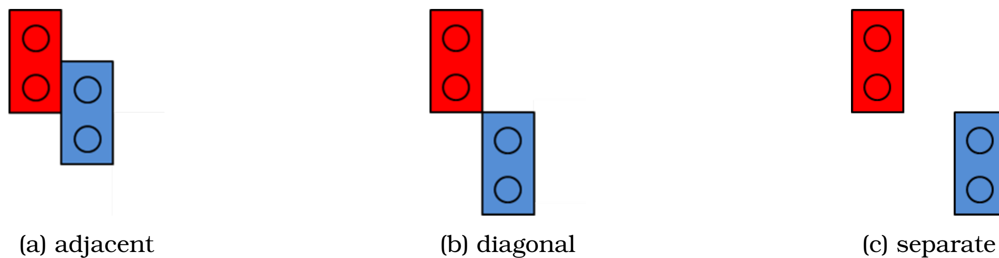


Figure 4.12: An example for weak points



(a) adjacent (b) diagonal (c) separate

Figure 4.13: The relative positions of bricks coming from different components at the same layer. We use different color to represent different components.

### 4.5.1 Connect disconnected components

After getting the graph representation of the LEGO model, we calculate how many connected components the LEGO model has after the merge step by directly checking how many connected subgraphs the graph has. However, before we start to connect different components together, we need to decide which components should be connected and which should be left disconnected. In Figure 4.13, we show the possible relative positions of bricks coming from different components on the same layer. We use different colors to represent different components. In the case of Fig 4.13 a), we can see that these two bricks should belong to the same connected component in the original 3D model whereas in Fig 4.13 b) and 4.13 c), the two bricks may or may not belong to the same component. Therefore, in order to decide whether two disconnected components should be connected, we check every pair of bricks that each of them belongs to one component and they are on the same layer. If any such pair of bricks is in the relative position of case 4.13 a) the two components should be connected, otherwise they should not be connected.

We observe that disconnections usually occur when two components only touch each other at a single layer. For example, in Fig 4.14, there are five different disconnected components(in different colours). As we can see, the two components in green and in mauve only touch each

other with bricks A and B. In other words, the disconnections usually appear at very thin regions. Therefore, disconnections should not appear if there are enough voxels around the

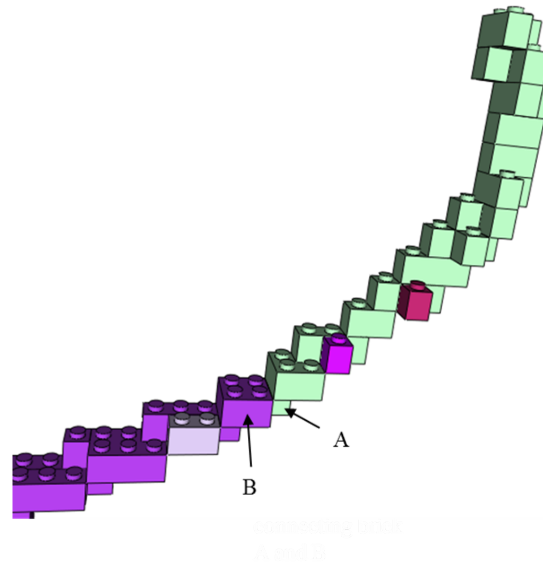
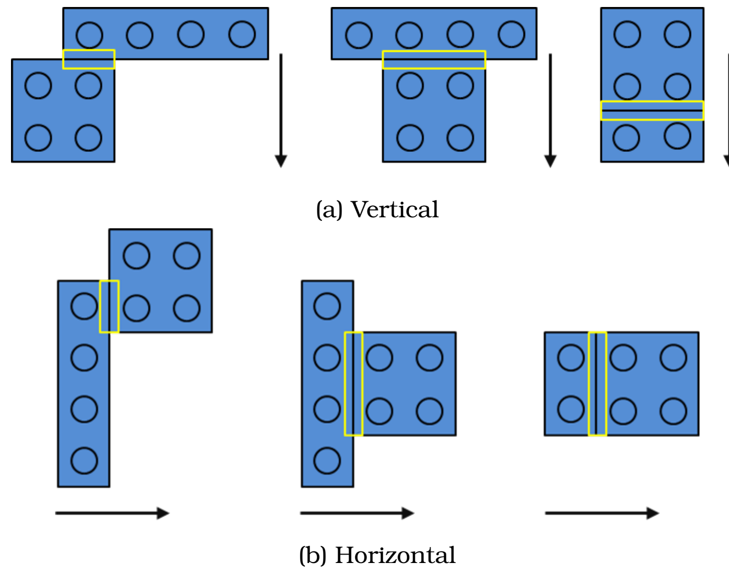



Figure 4.14: There are five disconnected components(in different colours) in the figure. The two components in green and in mauve only touch each other with bricks A and B.

part. Under this observation, we connect these disconnected components by adding extra voxels around the disconnected thin parts. The solution is following, for each components:

- 1: Find bricks in which at least one of their neighbors belongs to other components and record these bricks and their neighbors which belong to other components.
- 2: Add extra voxels above and below such bricks and their neighbors.
  - 2.1: Find the connection direction according to the relative position of the brick and its neighbor.
  - 2.2: Add a "band" of voxels along the connection direction above and below the brick and its neighbor.
- 3: Run the merge algorithm again.

We implement step 1 by searching all the neighbors for each brick and checking whether the neighbor belongs to other components. For step 2.1, we set the connection direction to be vertical or horizontal as in Fig 4.15 which is perpendicular to the direction of the adjacent edges. For step 2.2, the width of the band depends on the number of adjacent voxels between the brick and its neighbor. As we want to connect the brick and its neighbor, we need at least one voxel at each side of them both and we only add at most two voxels for each side in order



(a) Vertical

(b) Horizontal

Figure 4.15: The connect directions according to the relative positions

to keep the original model. Whether to add one or two voxels depends on the width of the brick and its neighbor.In this way, the length of the band is at least two and at most four. In Fig 4.16, we show some examples of the added voxels which are shadowed in purple. After adding the extra voxels, we run the merge algorithm again and repeat this process until we connect all the disconnected components.

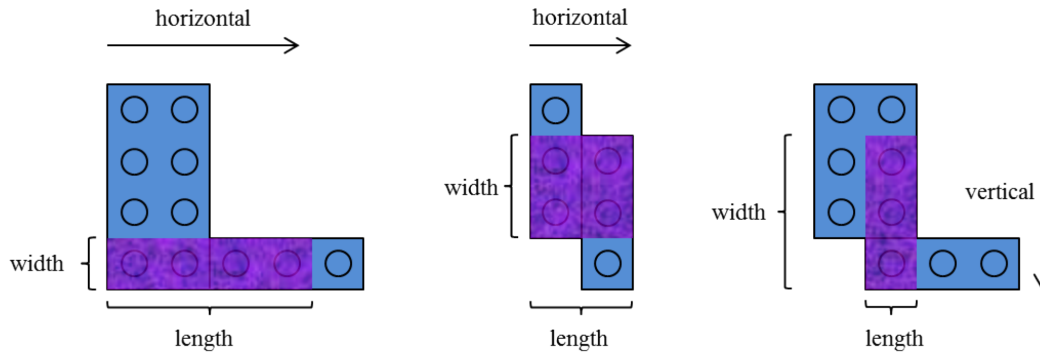



Figure 4.16: The added voxels (in purple shadow) at different situations of the brick and its neighbor

### 4.5.2 Remove weak points

As mentioned previously, the definition of a weak point is that: if two non-trivial subgraphs of size greater than 2 are only connected to each other by one brick, we define the brick that connects these two subgraphs as a weak point. According to this definition, we can predict that if we delete the weak point at the model, we will generate at least two disconnected

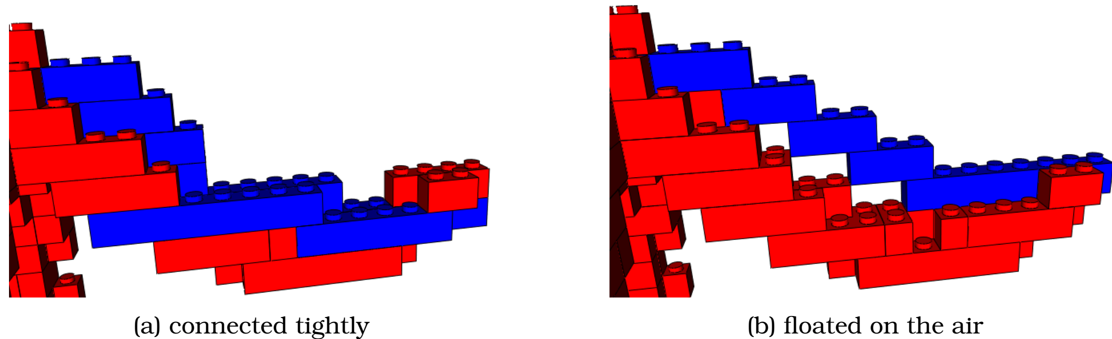
subgraphs. Therefore, in order to fix the weak point, we need to add new edges to connect these subgraphs into one connected graph. As there are multiple possible ways to add an edge to connect two subgraphs, we choose the way that adds the fewest number of voxels.

According to the mapping from the LEGO model to the graph representation, we know that we can only get an edge between two bricks that are on adjacent layers. This means that if we want to connect two bricks on layer $N$and $N+2$, we will need to add a brick on layer $N+1$ to connect these two bricks, which results in adding at least two edges in the graph. It indicates that if we connect two bricks at different layers, the less the layer difference is, the fewer the number of bricks we need to add between them. Although we need the fewest number of voxels, not bricks; the fewer the bricks, the fewer the voxels. In this way, we first find a solution to connect two vertices in different subgraphs with the lowest layer difference(which is one), and then increase the difference until we find a solution.

As stated previously, two bricks are connected when they are on adjacent layers and have at least have one knob overlapping. In this way, when we try to connect two bricks, we consider connecting a pair of knobs where each knob come from different bricks. So the solution can be described as: given two knobs, one on the $Nth$ layer and another on the $Mth$ layer, where $N < M$, we start from the source knob at the $Nth$ layer and add one brick at layer $N+1$ where the brick must overlap with the source knob. Then, we add another brick at layer $N+2$ which must have at least one knob overlapping with the brick at layer $N+1$, and repeat this until layer $M-1$, if the brick at layer $M-1$ overlaps with the target knob on layer $M$ then it is a solution to connect the two knobs. We choose the solution that only needs to add the fewest number of voxels. An implicit constraint is that we can only add bricks in a space that is not occupied by the original existing bricks. In addition, in order to generate a stronger model, we prefer the added bricks to be connected to the original model rather than floating in the air. For example, we will prefer the solution of Fig 4.17a) over 4.17b).

In addition, in order to connect two bricks as well as possible, we extend the original brick to other legal sizes of LEGO bricks so that we can try all the possible knobs to connect two bricks. As there are multiple ways to extend a brick, every time we generate a new position of a possible knob by extending the original brick, we will choose the extension that only needs to add the fewest number of voxels. We also need to consider the constraints of space limitation and tight connection when we extend the bricks. Fig 4.18 gives an example of extending a brick. In order to ensure a tight connection, we allow at most one voxel to float in the air as the last possible knob in Fig 4.18.

When we only consider the amount of voxels without the constraints of space limitation



(a) connected tightly

(b) floated on the air

Figure 4.17: Two different solutions of connecting a pair of knobs. The bricks in blue are the added bricks. We choose the solution of a), because b) will weaken the model.

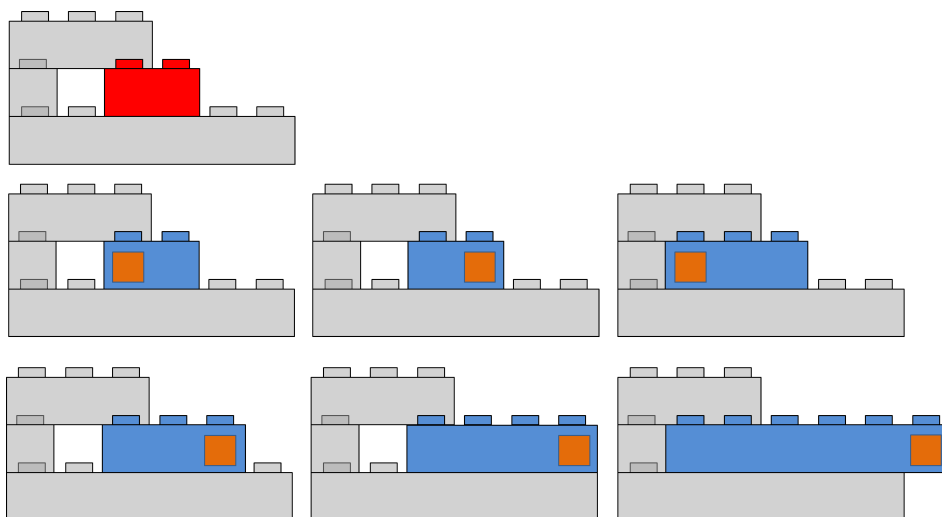


Figure 4.18: An example of extending a brick. We will extend the red brick and the grey bricks which are the exist bricks. we get 6 possible knobs by extending the original red brick. The yellow rectangle represents the position of the knob and the blue brick is the brick that contains the knob and requires the fewest number of voxels.

and tight connection, we could get the optimal solutions directly in the following way:

- Firstly, We get the coordinates of the source knob and target knob in 3D space such as $(s_x, s_y, s_l)$ and $(t_x, t_y, t_l)$ where $s_l < t_l$. The first two coordinates represent the $x, y$ position at each layer and the third coordinate represent the layer number. Our goal is to make the end position of the last added brick to be $(t_x, t_y, t_l - 1)$ while adding the fewest number of voxels. In other words, we try to get closer to the position of $(t_x, t_y)$ every time we add a new brick. In addition, in order to add the fewest number of voxels, we only use the bricks with sizes of $1 \times L$ where $L$ is the legal length of bricks, which is 1, 2, 3, 4, 6, 8 in our implementation. For example, in Fig 4.19, when the start position is $(0, 0)$ and end position is $(3, 1)$, we would need to add 8 voxels if we use a $2 \times 4$ brick while we would only need 6 voxels if we add a $1 \times 4$ brick and a $1 \times 2$ brick. However, when we can only

add one brick instead of two because of the limitation of level difference, we can not end at the target position by adding a $1 \times L$ brick. We will fix this problem later.

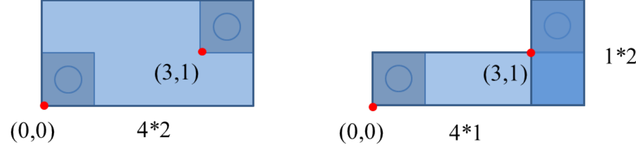



Figure 4.19: Two ways to add new bricks

- Secondly, we set $(s_x, s_y)$ to be our starting position $(b_x, b_y)$ of the brick-adding process at level $s_l + 1$. We choose a legal brick with a size of $1 \times L$ and put it in a proper direction so that the end position will get closer to $(t_x, t_y)$. In other words, when the end position is $(e_x, e_y)$ which is equal to $(b_x + l_x, b_y + l_y)$ where the values of $l_x$ and $l_y$ depend on the size and direction of the brick, we need the following conditions to be satisfied: $|e_x - t_x| \leq |b_x - t_x|$, $|e_y - t_y| \leq |b_y - t_y|$, $(e_x - t_x) \times (b_x - t_x) \geq 0$ and $(e_y - t_y) \times (b_y - t_y) \geq 0$.
- Finally, we set the end position of the previous added brick to be the new starting position of the brick-adding process on the next level and repeat this process until we add a brick on level $t_l - 1$. If the end position of the last added brick is $(t_x, t_y, t_l - 1)$ then we have found an optimal solution.

We claim that we can get all the possible optimal solutions by using the method described above and the number of added voxels will be $|t_x - s_x| + |t_y - s_y| + |t_l - s_l| - 1$. We give an example in 2D space where we only use one coordinate for the position of each layer and one for the layer number. In Fig 4.20, the source position is $(0, 0)$ and target position is $(5, 3)$. As we can see, it has 4 optimal solutions that connect the source knob and target knob in which the number of added voxels is 7. We need to notice that 5 and 7 are not legal lengths of bricks.

As can be seen, the above process is recursive in structure. When given a pair of knobs such that the source knob is $(s_x, s_y, s_l)$ and the target knob is $(t_x, t_y, t_l)$ where $s_l < t_l$ and their level difference is $L_1 = t_l - s_l$, we first add a new brick with end position at $(e_x, e_y, s_l + 1)$ and then recursively call the method to deal with the subproblem where the source knob becomes $(e_x, e_y, s_l + 1)$ and the level difference becomes 1 smaller than the original $L_1$. Finally, we combine the newly added bricks and the solutions of subproblems together and choose the optimal solution from all the possible combinations. However, we can also find subproblems that share subsubproblems. If we were to directly use the above method, we would repeatedly solve the common subsubproblems. Therefore, in order to improve efficiency, we will use

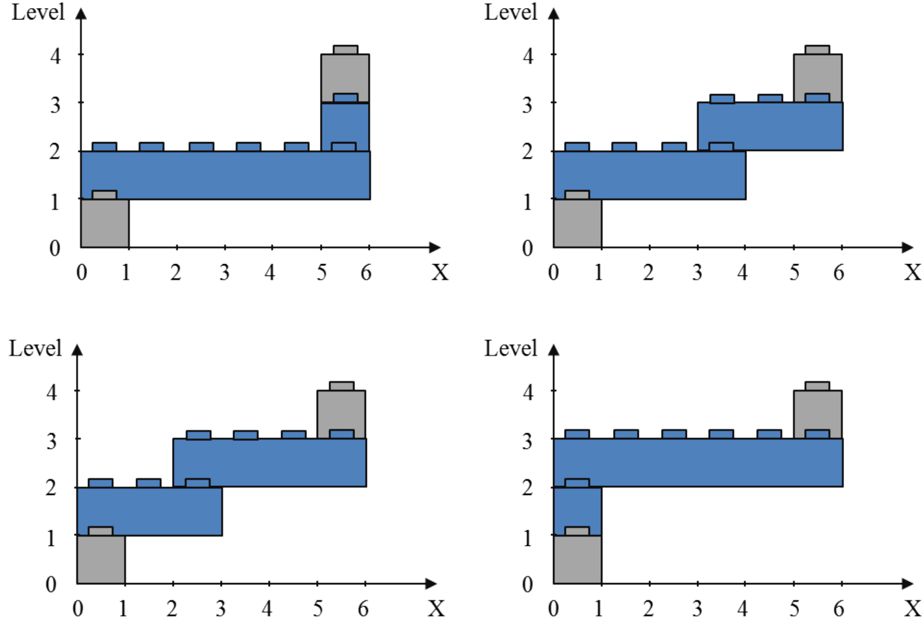



Figure 4.20: An example of all the possible optimal solutions to connect a pair of knobs. The grey bricks are the knobs to connect and the blue bricks are the added bricks. The number of added voxels is 7.

dynamic-programming technique to solve the optimization problem. This means that we will only solve every subsubproblem once and store its answer. In this way, we will not need to recompute the answer every time when we encounter the subsubproblem. Therefore, starting from the highest level $t_l - 1$, we get the optimal solution from $(p_x, p_y, t_l - 1)$ to $(t_x, t_y, t_l)$ for all the possible $p_x, p_y$ where $min(s_x, t_x) \leq p_x \leq max(s_x, t_x)$ and $min(s_y, t_y) \leq p_y \leq max(s_y, t_y)$. And when we compute the solution at next level below $t_l - 2$, we only need to compute the cost from the starting position to all the positions above level $t_l - 1$ and choose the optimal solution. We repeat the process until level $s_l + 1$. Finally, we get the optimal solution to connect the source knob and target knob. For example, in Fig 4.21, we will compute optimal solutions at all the possible positions and finally get the solution of connecting the source knob to the target knob at the bottom-left position. We will record the number of added voxels if there are any optimal solution or write -1 if there are no solution.

However, we still need to consider other constraints. For space limitation, we need to ensure that the needed space is not occupied by an existing brick every time we put a new brick in a certain position. In addition, in order to guarantee tight connection, we need to ensure all the knobs of the newly added brick except the starting and ending position are connected to existing bricks. If any newly added brick do not satisfy these two constraints we will not choose this solution. Moreover, in order to encourage tight connection, we count the

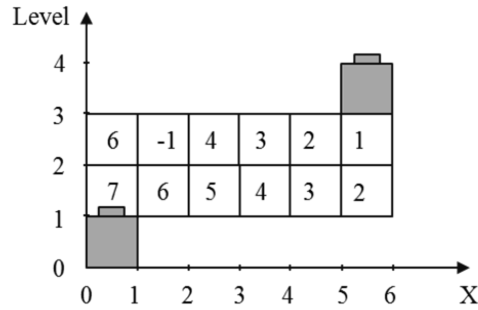
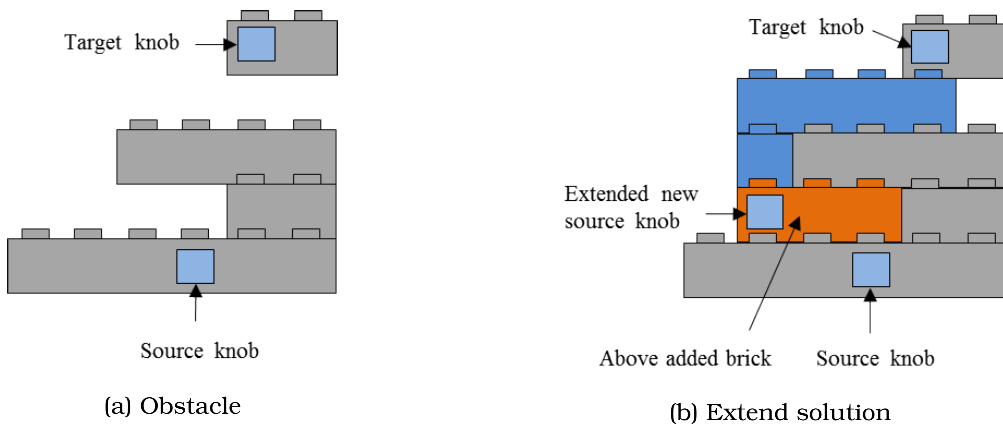



Figure 4.21: An example of the possible optimal solutions at all the position by using the dynamic programming technique. The positive number in each position represents the amount of added voxels of the optimal solution and -1 represents no solution in this position. We get the optimal solution of connecting the source knob to the target knob at the bottom-left position which is 7.

number of existing knobs that are connected by the newly added bricks. For two solutions in which the number of added voxels are the same, we will prefer the one with the larger number of connected knobs. If there are still multiple optimal solutions in which the number of added voxels and connected knobs are the same, we will choose one solution randomly as our optimal solution.

Since we need to consider other constraints, we may not find a solution by using the above algorithm. For example, in Fig 4.22a), because of the existence of original bricks we can not get an optimal solution by using the above algorithm. However, if we were to add a new brick on the layer above of the source knob in such a position that its end position is far away enough from the target knob, we would be able to get pass the original bricks. Then, we could reapply the above algorithm to the pair of knobs where the end position of the added brick is the new source knob. In other words, we can extend the position of source knob by adding a brick above the source knob so that we can cross the obstacle. For example, in Fig 4.22b). We can extend the position of the source knob until we get a solution or go to level $t_l - 1$.

Finally, if we still can not find a solution to connect a pair of knobs after all the above processes, then we need to try other pairs of knobs until we find a solution to connect two bricks.




(a) Obstacle

(b) Extend solution

Figure 4.22: At a), we show an example that could not generate the optimal solution because of the existence of original bricks. At b), we extend the source knob by adding a brick above it so that we could across the obstacle and then generate an optimal solution.

# Chapter 5

# Experimental Results

## 5.1 Balancing Results

In Fig 5.1, we provide an example of our algorithm for balancing models. Fig 5.1 a) gives the voxelization result of the dinosaur model and the resolution is 32. However, the voxelization model is not balanced. In Fig 5.1 d), the black polygon is the support region, the blue dot is the projection of the original centre of mass onto the plane of support region and the blue line is the projection line. As can be seen, the original centre of mass is outside of the support region and it is unbalanced. We try to balance this model and get the result in Fig 5.1 d). The red dot is the final projection of the centre of mass and it is inside the support region. Fig 5.1 b) shows the original layout of the model at layer 18 and Fig 5.1 c) shows the result after balancing and the blue voxels are the added voxels. Fig 5.2 gives another example of balancing a three-overlaid-spheres model. For this example we voxelized the model in a higher resolution of 64.

In Fig 5.3 and 5.4, we use our algorithm to balance these models after placing the models in certain positions and build the actual model to show the correction of our algorithm. The Fig 5.3 a) gives the original model of the example. Since the model would stand if we put it in the position shown in the Fig 5.3 a), then in order to make use of our algorithm, we would put the original model in the position shown in Fig 5.3 b) so that it is unbalanced. The Fig 5.3 c) gives the voxelization result of the bear model in this position at a resolution of 16. As we can see, the base layer is quite small and only has two voxels. Fig 5.3 d) and Fig 5.3 e) give the layouts of the model at layer 7 before and after balancing. Fig 5.3 f) shows the movement of the centre of mass of the model. We built the actual model of the bear after optimization.

Fig 5.3 g) shows the original result before balancing and Fig 5.3 h) shows the balanced model which could stand without falling.

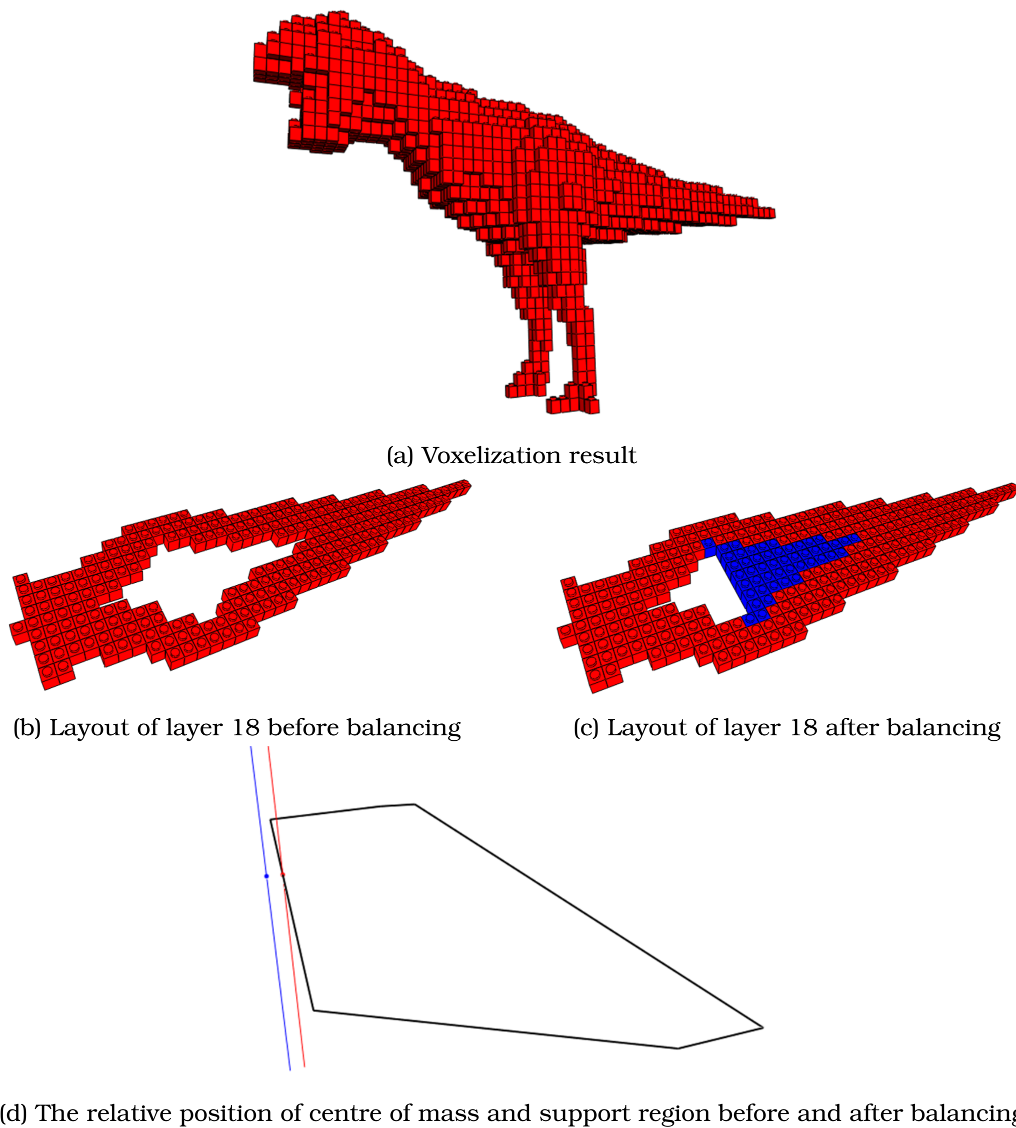
## 5.2 Improving connections Results

### 5.2.1 Connect disconnected components

In Fig 5.5, we show an example of connecting multiple components together. Fig 5.5 a) shows the original disconnected components with different colours for each after we run the merge algorithm. The figure shows that the disconnected components appear at the very thin part and we can add extra voxels to connect them together. For example, in Fig 5.5 a), bricks A and B are the neighbouring bricks of the green component and the purple component. We would add extra voxels above and below A and B according to our algorithm as shown in Fig 5.5 b). After we have added all the needed voxels(in blue) as shown in Fig 5.5 b), we run the merge algorithm again and get a connected component as shown in Fig 5.5 c). Fig 5.5 d) shows the graph for the connected component.

### 5.2.2 Remove weak points

In Fig 5.6 and Fig 5.7, we give examples of weak points removal. The weak points in both examples are at the thin parts of the models where Fig 5.6 shows the tail part of a dog and Fig 5.7 shows the tail part of a cat. In Fig 5.6 a), there are 7 weak points which are the red bricks and we marked the nodes of weak points with the red circles at the corresponding graph representation in Fig 5.6 c). Fig 5.6 b) shows the result of our algorithm where red bricks are the original bricks and blue bricks are the newly added or extended bricks. From our observation, a few weak points are at the adjacent layer and it is possible for us to remove them with the same removal process. For example, the three weak points which are marked by blue circles in Fig 5.6 d) can be removed simultaneously by connecting the two nodes that are marked by the blue triangles. That removal process is quite efficient since we only need to add one extra voxel to remove three weak points which are the bricks pointed by the blue arrows in Fig 5.6 b). For another example, the four weak points marked by the yellow circles are removed simultaneously by dealing with the nodes marked by yellow triangles which only add eight voxels. In addition, we observe that the newly added bricks are connected tightly to the original existing bricks. We also generate an efficient and tightly connected result in the example shown in Fig 5.6.



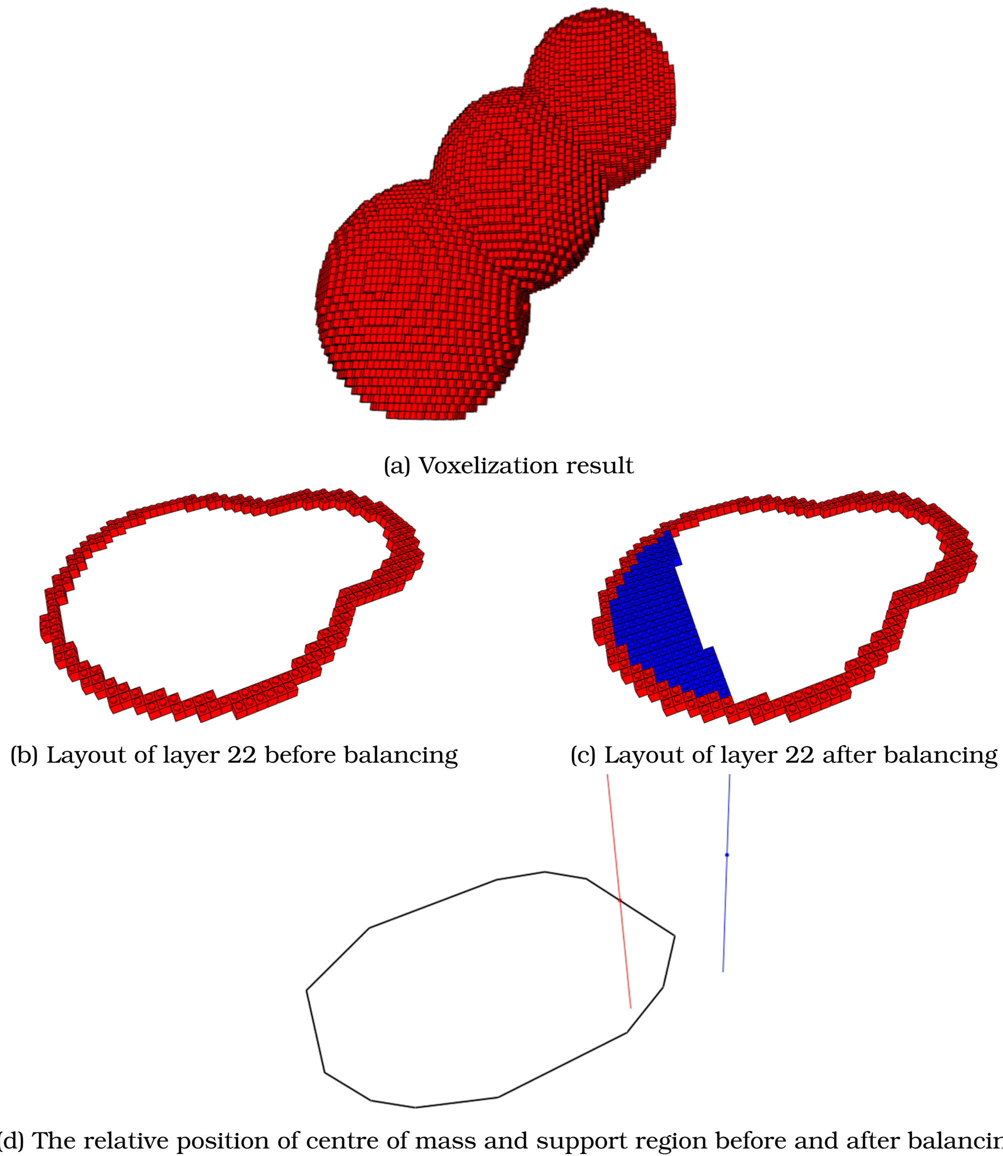
(a) Voxelization result

(b) Layout of layer 18 before balancing

(c) Layout of layer 18 after balancing

(d) The relative position of centre of mass and support region before and after balancing

Figure 5.1: We give an example of balancing a dinosaur model. The resolution of a) is 32. The blue voxels is the newly added voxels after balancing in c). In d), the black polygon is the support region and the blue dot is the projection of the original centre of mass onto the plane of support region in gravity direction as in blue line and the red dot is the result after balancing.



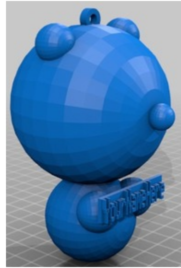
(a) Voxelization result

(b) Layout of layer 22 before balancing

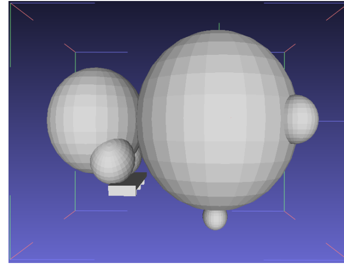
(c) Layout of layer 22 after balancing

(d) The relative position of centre of mass and support region before and after balancing

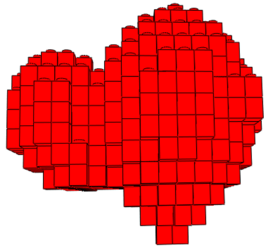
Figure 5.2: We give an example of balancing a three-overlaid-spheres model. The resolution of a) is 64. The blue voxels is the newly added voxels after balancing in c). In d), the black polygon is the support region and the blue dot is the projection of the original centre of mass onto the plane of support region in gravity direction as in blue line and the red dot is the result after balancing.



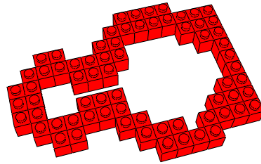
(a) Digital model



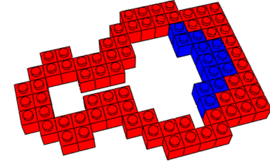
(b) Rotated model



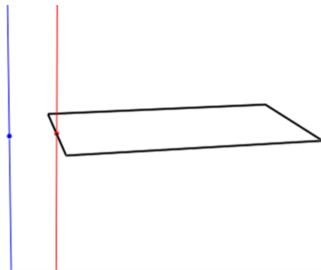
(c) Voxelization result



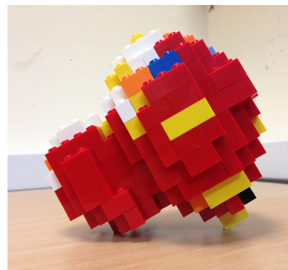
(d) Layer 7 before balance



(e) Layer 7 after balance



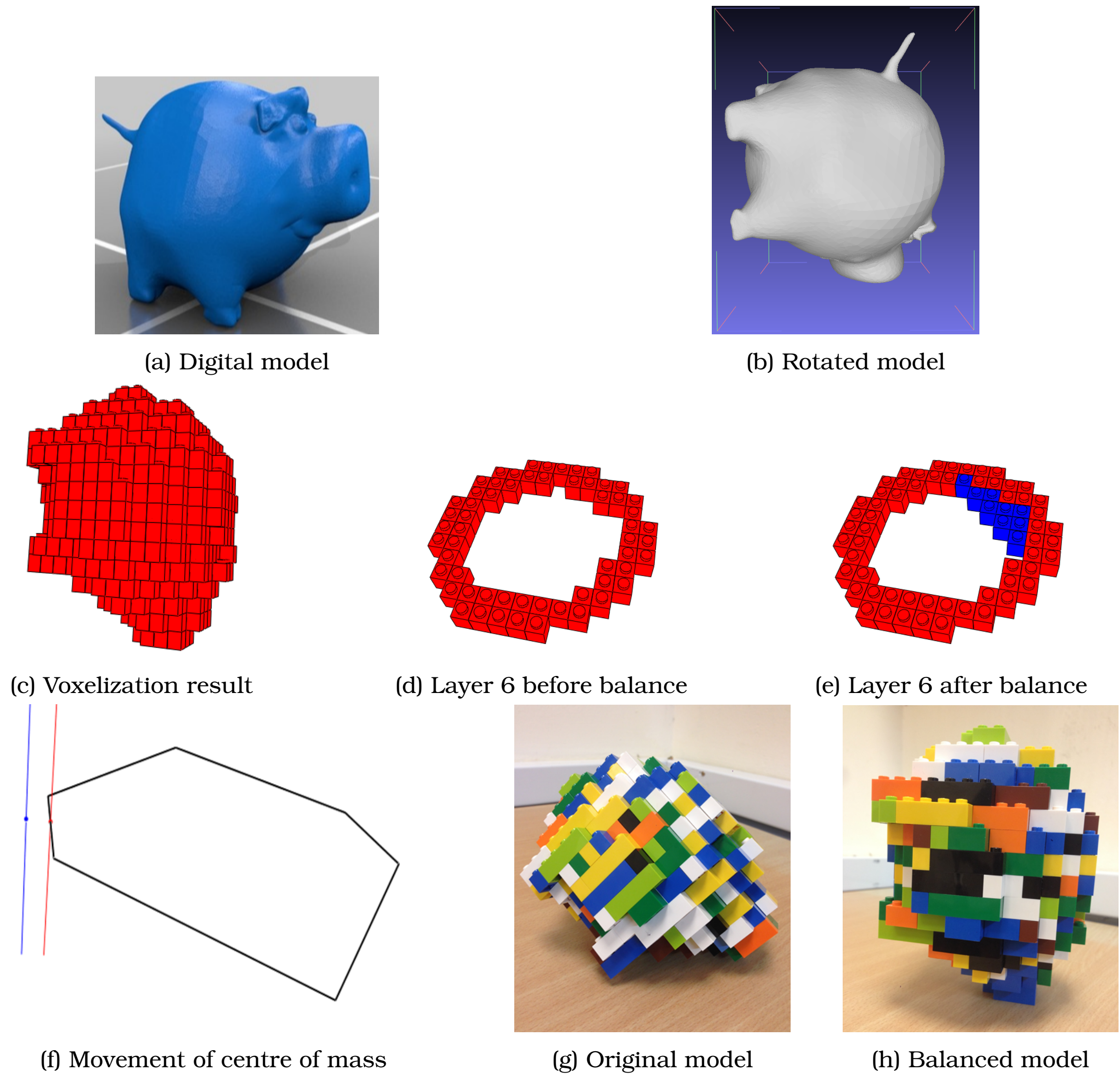
(f) Movement of centre of mass



(g) Original model

(h) Balanced model

Figure 5.3: We give an example of balancing a bear model. The resolution of c) is 16. In order to state the correctness of our algorithm, we build the actual LEGO model before and after balancing.



(a) Digital model

(b) Rotated model

(c) Voxelization result

(d) Layer 6 before balance

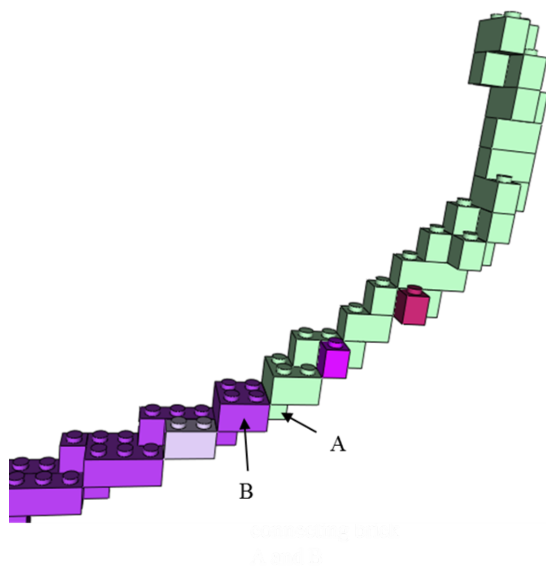
(e) Layer 6 after balance

(f) Movement of centre of mass

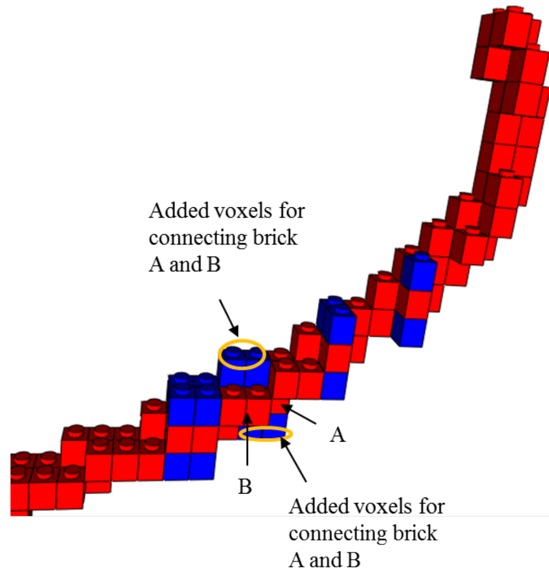
(g) Original model

(h) Balanced model

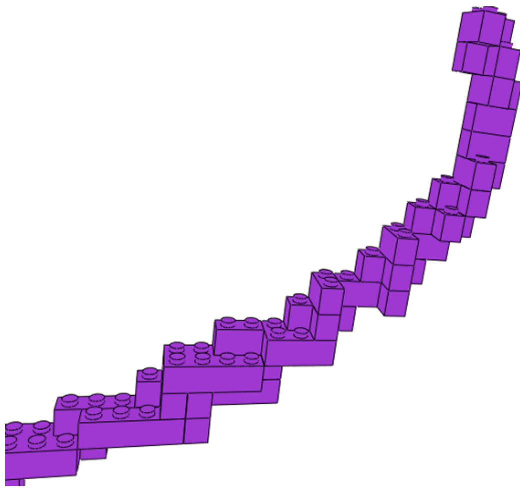
Figure 5.4: We give an example of balancing a pig model. The resolution of c) is 16. In order to state the correctness of our algorithm, we build the actual LEGO model before and after balancing.



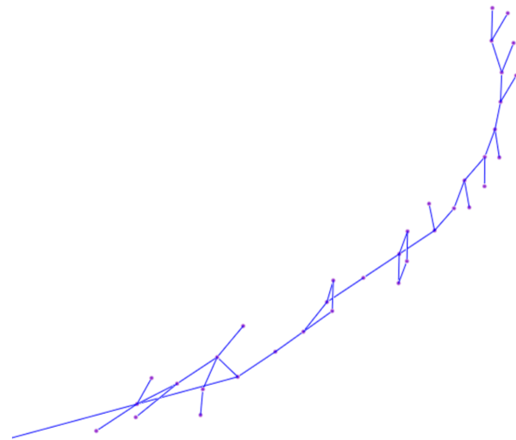

(a) Original disconnected solution




(b) Added voxels

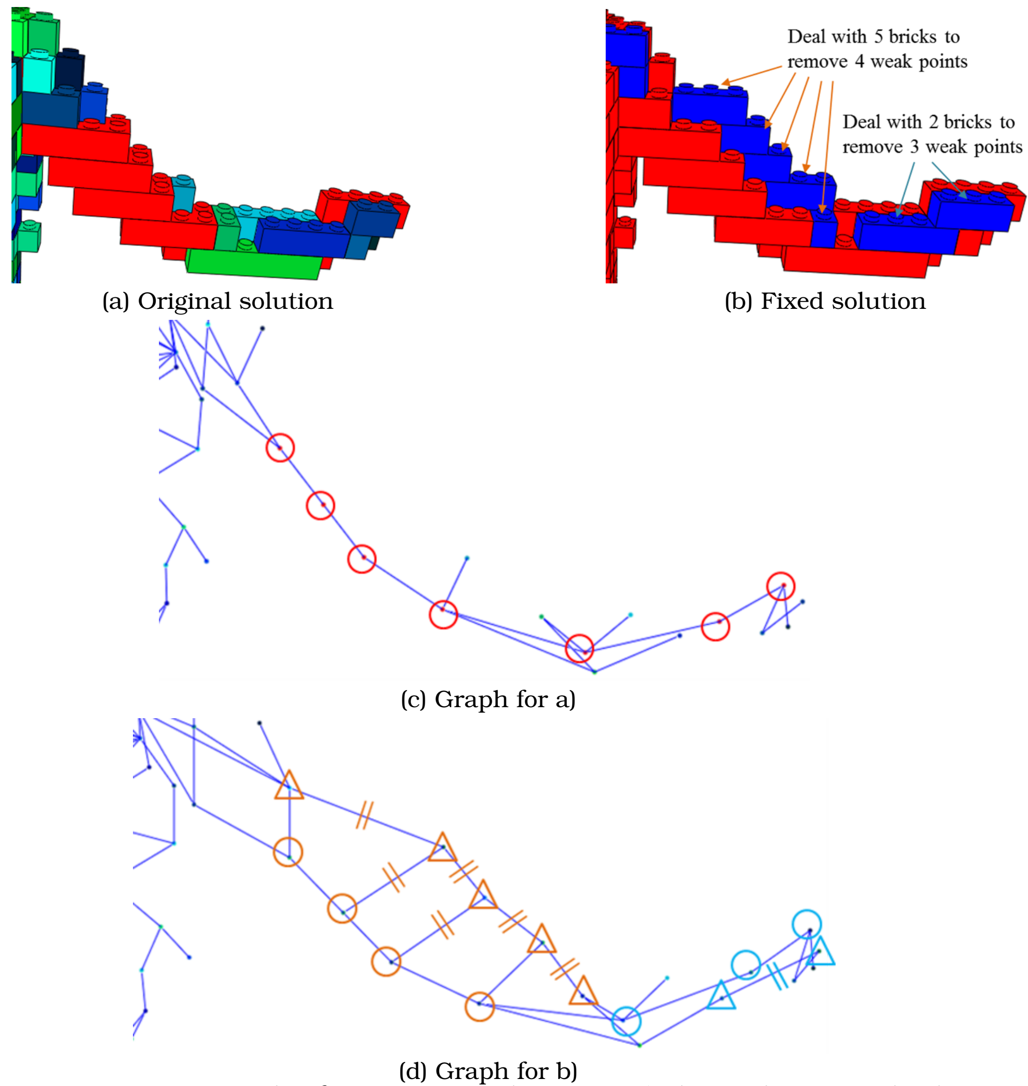


(c) Fixed connected solution



(d) Graph for c)

Figure 5.5: We give an example of connecting multiple components together. a) shows the original disconnected solution which has 4 disconnected components where each has a different color. In b), the red voxels are the original existing voxels and the blue voxels are the added voxels to connect different components. c) is our fixed connected solution after we have added the needed voxels and rerun the merge algorithm. d) is the graph for the connected solution.



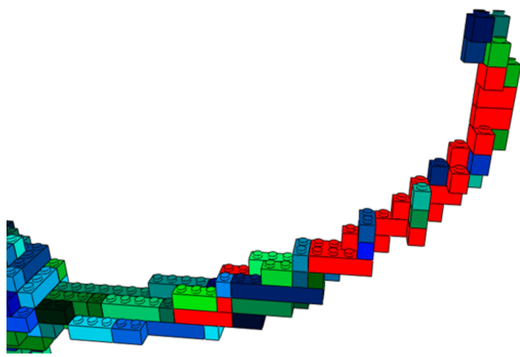

(a) Original solution

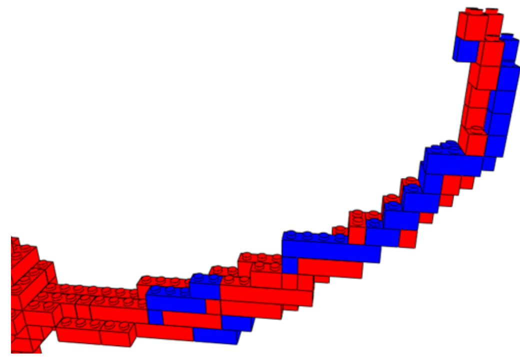
(b) Fixed solution

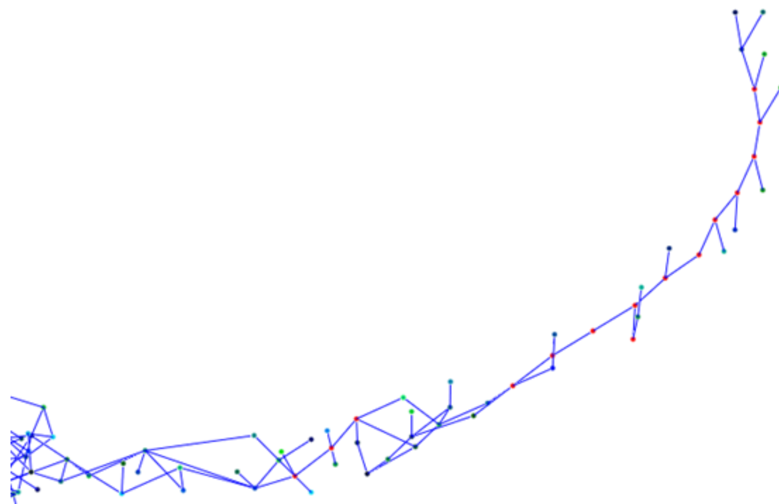
(c) Graph for a)

(d) Graph for b)

Figure 5.6: We give an example of removing weak points. a) shows the original solution which has 7 weak points(red bricks). c) shows the corresponding graph for the original solution and the nodes marked by red circles are the weak points. b) is our solution to remove these weak points. We could remove a few weak points which are in adjacent layers at the same time. As shown in d), when we deal with the nodes marked by blue(yellow) triangles, we can remove these weak points marked by blue(yellow) circles simultaneously.


(a) Original solution


(b) Fixed solution


(c) Graph for a)

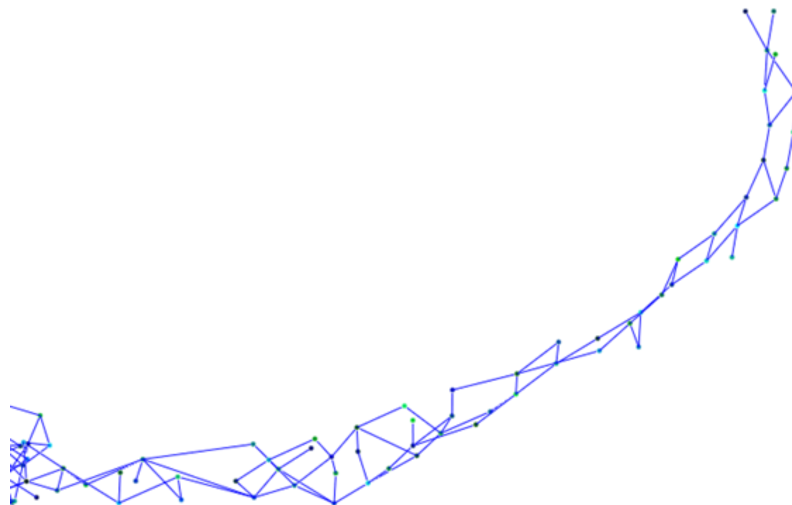

(d) Graph for b)

Figure 5.7: We give another example of removing weak points.

# Chapter 6

# Limitations

In our balancing algorithm, we gave the same weight for all the voxels of the voxelized model and computed the centre of mass by calculating the average coordinates of all the voxels. However, when we built the actual LEGO model, the unit weight per voxel for each size of the bricks were different. That meant that the actual centre of mass was different from what we computed. In order to make sure that the actual LEGO model is actually balanced, we need to compute the accurate centre of mass or give more tolerance for the balancing condition. As every time when we ran the merge algorithm we would generate different results of building a LEGO model, the actual centre of mass would also be different. This meant that we could not calculate the actual centre of mass until we got the result. Since we would like to balance the model no matter what the merge result was, we could solve the problem by giving more tolerance for the balance condition rather than computing the accurate centre of mass.

As we tried to balance the model by only adding extra voxels inside the hollowed model so that we could keep the appearance of original model, it was possible that there were not enough potential voxels to add to make the model balance. We could try other methods to balance the model such as changing the support region, adding struts or even adding other material with different density inside the hollowed model.

During the balancing process, we treated all the bricks of the model as a whole and we only considered placing the centre of mass inside the convex hull of the lowest layer. However, since the force between bricks are limited, the actual LEGO model might collapse at the thin regions. For example, in Fig 6.1, although the centre of mass is inside the support region of the lowest layer, the model might still collapse at the thin leg part such as the $12th$ layer. Since the centre of mass for all the bricks above that layer is outside the convex hull of the

bricks on the $12th$ layer, the total weight of all these bricks might be large enough to cause the model to break at the $12th$ layer. We wouldn't know whether the model would fall down at the thin part or not unless we built the actual LEGO model or preformed force and stress analysis.

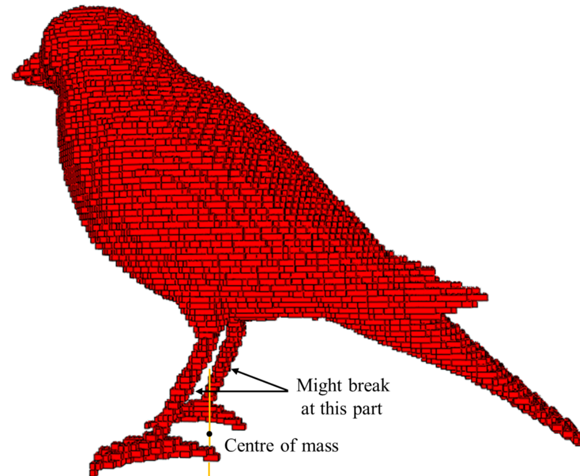



Figure 6.1: Although the centre of mass of the bird is inside the support region, it might break at the thin leg part.

The process of improving connections was one way to improve the strength of the model at the necessary parts. Since we only analysed the structure of the LEGO model in graph representation and ignored the actual size of the bricks and the actual force and stress between bricks, we could not ensure that the model could stand without collapsing.

# Chapter 7

# Conclusion

In this thesis, we have described a method to automatically generate the LEGO model from a given 3D digital model. The basic idea is to merge all the voxels of the voxelized model into larger LEGO bricks. We focus on improving the structural strength of the LEGO model. The first thing we do is to balance the model by adding necessary extra voxels inside the voxelized and hollowed model. In addition, we improve the connections by using a graph to represent the actual connections between bricks and analysing these structures to find the badly connected parts and then repairing them by adding extra bricks.

For the future work, we can take three different aspects into account. We could use other methods to balance the model such as changing the support region, adding struts or even adding other material with different density inside the hollowed model. We could also consider improving the structure strength by analysing the force and stress between bricks. Finally, we could use more specific bricks such as slope bricks to make the surface of the model smoother.

# References


[1] R Gower, A Heydtmann, and H Petersen. “LEGO: Automated Model Construction”. In: (1998).

[2] Pavel Petrovic. “Solving LEGO brick layout problem using Evolutionary Algorithms”. In: *Proceedings to Norwegian Conference on Computer Science*. 2001.

[3] D Winkler. *Automated brick layout*. 2005.

[4] Lynette van Zijl and Eugene Smal. “Cellular automata with cell clustering”. In: *Automata*. Citeseer. 2008, pp. 425–441.

[5] Romain Testuz, Yuliy Schwartzburg, and Mark Pauly. “Automatic generation of constructable brick sculptures”. In: *Proc. Eurographics*. Vol. 13. 2013.

[6] Greg Saul et al. “SketchChair: an all-in-one chair design system for end users”. In: *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*. ACM. 2011, pp. 73–80.

[7] Nobuyuki Umetani, Takeo Igarashi, and Niloy J Mitra. “Guided exploration of physically valid shapes for furniture design.” In: *ACM Trans. Graph.* 31.4 (2012), p. 86.

[8] Romain Prévost et al. “Make it stand: balancing shapes for 3d fabrication”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 81.

[9] Ondrej Stava et al. “Stress relief: improving structural strength of 3D printable objects”. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 48.

[10] Grégoire Allaire, François Jouve, and Anca-Maria Toader. “A level-set method for shape optimization”. In: *Comptes Rendus Mathematique* 334.12 (2002), pp. 1125–1130.

[11] John C Hart, Brent Baker, and Jeyprakash Michaelraj. “Structural simulation of tree growth and response”. In: *The Visual Computer* 19.2 (2003), pp. 151–163.

[12] Alexandru Telea and Andrei Jalba. “Voxel-based assessment of printability of 3D shapes”. In: *Mathematical Morphology and Its Applications to Image and Signal Processing*. Springer, 2011, pp. 393–404.

[13] Christopher M Bishop et al. “Neural networks for pattern recognition”. In: (1995).

[14] Eugene Smal. “Automated brick sculpture construction”. PhD thesis. Stellenbosch: Stellenbosch University, 2008.

[15] Patrick Min. *binvox 3D mesh voxelizer, keywords: voxelization, voxelisation, 3D model.* 2014. URL: `http://www.cs.princeton.edu/~min/binvox/`.

[16] Fakir S. Nooruddin and Greg Turk. “Simplification and repair of polygonal models using volumetric techniques”. In: *Visualization and Computer Graphics, IEEE Transactions on* 9.2 (2003), pp. 191–205.