Title	An Information-theoretic Approach to Origami Folding Sequence Generation from 3D Shape Models
Author(s)	Bui, Duong Ha
Citation	
Issue Date	2017-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/14163
Rights	
Description	Supervisor:丁　洛榮，情報科学研究科，修士

An Information-theoretic Approach to Origami Folding Sequence Generation from 3D Shape Models

Bui Ha Duong

School of Information Science
Japan Advanced Institute of Science and Technology
March, 2017

Master's Thesis

An Information-theoretic Approach to Origami Folding Sequence Generation from 3D Shape Models

1510050 Bui Ha Duong

Supervisor :	Professor Chong Nak-Young
Main Examiner :	Professor Chong Nak-Young
Examiners :	Professor Ryuuhei Uehara
	Associate Professor Kokolo Ikeda

School of Information Science
Japan Advanced Institute of Science and Technology

February, 2017

Abstract

Folding presents in many fundamental aspects of life. Many things in universe simulate and construct based on this operation. The wave of light and sound repeatedly fold and unfold in space to broadcast information. We are all born with a DNA folding form. Inheriting the properties of folding, origami - the art of paper folding, also contribute to several technological products. With the main characteristic is creative, many works in space technology, automobiles, medicine, robotics, and programmable matter, which based on origami, are considered outstanding.

This thesis considers an information-theoretic approach to modeling and answering the origami folding sequence generation from 3d shape models problem. The algorithm, as input, receives an origami paper (flat sheet square of paper), a 3d model from a real life object or a creative work, and a predefined crease pattern. It, then, generates some of the possible folding sequences that will result in the desired model. By this definition, our task is properly described as a combinatorial optimization problem (COP). A feasible solution is a list of folding actions to create creases which are included in the input crease pattern. The set of feasible solutions is called the search space. The objective function is finding an optimal solution in the search space that has the minimum Hausdorff distance with the input objective model. We present a framework to tackle this COP using particle swarm optimization (PSO). With the observation that the problem is an NP-hard problem, and the solutions are in discrete space, we proposed a modified discrete PSO (DPSO) method that can be suitable for our requirements. The characteristics of the proposed algorithm are carefully discussed. First, we introduced a discrete search space. In this space, the positions of particles (or feasible solutions) and its velocities are vectors with integer elements. Second, the behavior of the particles in the swarm is adjusted. We redefined all arithmetic operators to customize the formulae in the standard PSO that are used to move particles and change position. Based on that, the modified DPSO version can take the advantages of the standard PSO's characters, as well as, efficiently search in our discrete space. Besides, a folding simulation to convert a feasible solution (or a folding sequence) into a flat model is also adapted and developed in our work.

With this approach, some experiments are conducted for evaluation. Our system shows that it is promising. Folding sequences of input objective models have been showed with high precision in acceptable running time. The DPSO algorithm always keeps track of minimizing the Hausdorff distance between an input and feasible solutions. In someway, this work revealed the contribution to the field of origami simulator and folding multiple objects model from a single paper sheet.

Acknowledgment

First and foremost, I am deeply grateful to my supervisor, Professor Chong Nak-Young for his insightful encouragement and continuous support. His creative ideas, critical thinkings, and professional working style are important factors that affect directly to the completion of my master thesis.

Then, I would like to express my gratitude to Professor Yuto Lim and Professor Sungmoon Jeong for their valuable suggestions and useful advice in my research.

Thirdly, my sincere thanks goes to my thesis committee: Professor Ryuuhei Uehara, Professor Kokolo Ikeda and my second supervisor Professor Fumihiko Asano for their helpful comments, interesting questions, and considerations.

Moreover, special thanks also to the teachers and staffs in Japan Advanced Institute of Science and Technology. With their hard work and dedication, I have my chance to achieve necessary knowledge as well as to learn in a high-level learning environment.

Last but not least, my deepest heartfelt appreciation goes to my family and my friends for always believing in me. Like the North Star, they are the guiding light for my life's adventures.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Folding presents in many fundamental aspects of life. Many things in universe simulate and construct based on this operation. The wave of light and sound repeatedly fold and unfold in space to broadcast information. We are all born with a DNA folding form. Inherited the properties of folding, origami - the art of paper folding - also contribute to several technological products. With the main characteristic is creative, many works in space technology, automobiles, medicine, robotics, programmable matter, etc. based on origami are considered outstanding. Naturally, people use mathematics to describe the features of origami. From the 1930s, problems and solutions about paper folding and unfolding have been constructed in computational ways [1].

However, until now, it is widely thought that the creation of an origami model is work of art and most people are familiar with folding origami models which have been created by other artists. Very few information research articles have addressed the question of autonomous creating origami model. Furthermore, within the next few years, origami is likely to become an important component in programmable matter.

Therefore, this research will follow the state-of-the-art origami research to design an algorithm that, in the future, can support the autonomous folding systems. The algorithm, as input, receives an origami paper (flat sheet square of paper), a 3d model from a real life object or a creative work, and a predefined crease pattern. It, then, generates some of the possible folding sequences that will result in the desired model.

1.1 Literature Review

Our research has a strong connection with mathematics and technique origami. Specially, three main subfields that directly related to our work are,

- Origami simulator [2–6]
- Folding sequence generation [7, 8]
- Folding multiple objects from a single sheet [9]

In this section, the state-of-the-art overviews of these subjects are introduced.

Freeform Origami [5, 10–12]

This software is a well-known folding simulator in origami. By combining the functionals of 2 tools *Origamizer* and *Rigid Origami Simulator*, it is developed by Tomohiro Tachi from 2010 until 2016. These programs provide to users an environment to interact with a virtual paper. The common tasks in origami such as folding paper, modifying crease pattern are included. The author also helps people easily use his software by implemented the animations of folding actions.

However this is not only an origami simulator, but it is also able to generate a crease pattern of a polyhedron. By using the quadrilateral mesh information of the input, it is first unfolding the 3D shape to get the candidate crease pattern. Then, this crease pattern is simultaneously folded and controlled by apply affine transformation. This process is repeated until the objective model is reached.

Generating Folding Sequences from Crease Patterns of Flat-Foldable Origami [8]

This is an interesting research from Hugo A. Akitaya et al. The primary purpose of this work is find a way to generate the folding sequence of a particular flat-foldable origami shape. To accomplish this goal, the frameworks builds a new graph-like data structure called extended crease pattern. The researchers construct this data by using the input crease pattern information, then with obtained graph, they unfold the input model. The folding sequence can quickly present by invert the unfolding process. In this system, sometimes, users are requested to decide which is the next step in the unfolding sequence because many outcomes from the extended crease pattern are possible. So it is considered as a semi-autonomous system.

Planning to Fold Multiple Objects from a Single Self-Folding Sheet [9]

This is a study about folding multiple objects from a single origami paper. An et al. considers how to transform between 3D shapes by using programmable matter. In this article, they properly defined a programmable sheet with the set of hinges. A list of objective models as input is tried to construct and convert between these by using the predefined-origami paper. With the support of another origami methods, they can find the folding actions for a specific shape. From this information, an efficient plan to construct the input object are provided.

1.2 Research Motivation

The important role of folding has been discussed. Today, with the broadening in many research subjects, folding proved that it is suitable with many new ideas. Erik Demaine, who is a professor has many studies in origami, listed some types of folding, as well as their applications [13],

- Linkage folding: protein folding, hydraulic tube bending, robotics
- Paper folding: packaging, airbag folding, sheet-metal bending
- Unfolding polyhedrons: sheet-metal bending, manufacturing

Among the active development of science and technology, particularly in robotics, automation and materials science, this research was proposed in the hope to work with self-folding systems and programmable matters to become a useful application [14–17]. We can create a screen that can change the size, a mobile phone that folds to transform into a tablet or a tv, a bag that folds to hold any amount from small size to big size or a robot that folds to adapt to the environment. Besides, this system can support the origami artists in design and build the origami models, or maybe help people who are inexperienced in folding. Furthermore, we have a chance to study about new origami techniques, mathematical origami problems such as fold and cut, unfolding and folding polyhedrons.

1.3 Research Goal

In those approaches mentioned in Section 1.1, researchers have tended to focus on the mathematical aspects of origami problems. Consequently, the solutions are also derived mainly from mathematical techniques. Besides that, although many types of research in origami have been processed, but because of the difficulty of the task, a well planned, fully-autonomous way to generate the folding sequence of an origami model is still required.

Therefore, in this thesis, we aim to present a new generic way to find answers for folding-related issues. An information-theoretic approach will be elaborate proposed in the next chapters. In our research, the inputs are a predefined crease pattern and an objective model. With this information, our proposed algorithm tries to build an object model from the predefined crease pattern that maximizes the similarity with the input shape. With this techniques, we model our problem as a combinatorial optimization problem and from that, we can take advantage of efficient computational methods.

1.4 Thesis Organization

We divided this thesis into five chapters: Introduction, Background, Methodologies, Evaluation and Conclusion.

In Chapter 2, we introduce various background knowledge related to our research. At first, basic concepts about Origami and its application are mentioned. Next, combinatorial optimization is covered to help us understanding about this field of optimization as well as the role of this definition in many real life problems. Then an efficient method to solve the optimization problem is presented.

In Chapter 3, we describe in detail all the proposed methods that we use in this research.

In Chapter 4, first, we prepare all the necessary information to construct experiments. Then, all the experimental results are figured out. Based on this, we analyze our proposed approach.

In Chapter 5, we summarize the current results in our study and mention some future works.

Chapter 2

Background

2.1 Origami

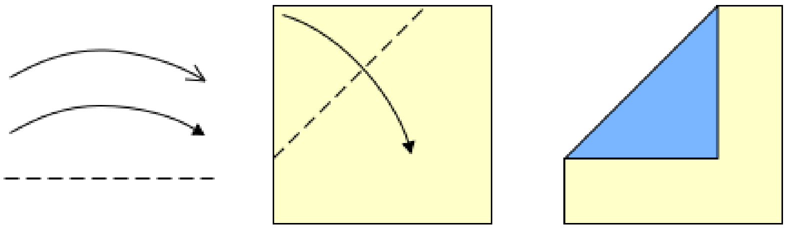
The origin of origami is still covered by unanswered questions. Where does it come from? Who is the creator? History facts about this topic have not been recorded [18]. In the early of 1900s, the very first works and innovations in this field are beginning properly constructing and documenting by Akira Yoshizawa, Kosho Uchiyama, and others [19]. From the 1980s, with the development of mathematics and computing, researchers have created many complexity origami models [20]. Today, when talking about *origami* - a word in Japanese, people usually reminded about Japan, as well as the art of paper folding. In daily communication, this word is also indicated all folding practices. An origami model is built by apply many single folds. These actions are usually simple, but their combinations are capable of making great structures. Basic origami folds - valley fold and mountain fold are showed in Figure 2.1 [21]. In Figure 2.2 are some famous origami models.

Technical origami is an interesting subject in origami. The main object that is researched in this field is engineered crease pattern. The image of all the creases which is obtained when unfolding an origami shape is the definition of crease pattern. When the simple step-by-step instructions are inefficient presenting the models, then crease pattern shows its meanings. Traditional crane crease pattern is given in Figure 2.3 [22].

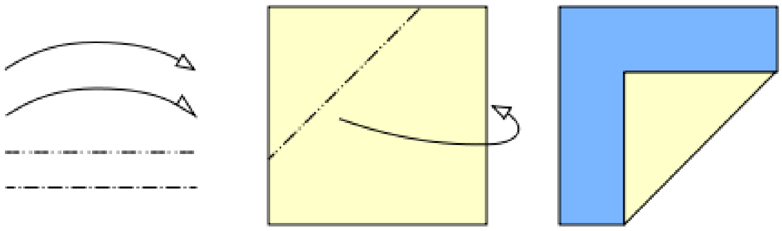
Origami also has many applications in industrial and impact to many aspects of life. In space, BYU-designed solar arrays can be compressed for launch and then set up in space to perform works (Figure 2.4a [23]); In health care, An origami inspired device can be transferred into a diseased artery and repair it (Figure 2.4b [24]), etc.

2.2 Combinatorial Optimization

Combinatorial optimization is a concept in mathematics and computing. In this problem, the task is finding an optimal solution from a finite set of candidates [25]. Many famous optimization problems are categorized in this topic. Some examples are knapsack prob-



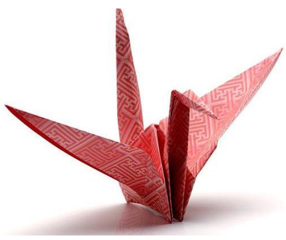
(a) Valley fold



(b) Mountain fold

Figure 2.1: Origami valley fold and mountain fold

(a) Origami Dragon (Designed by Jo Nakashima)



(b) Origami Crane

(c) Origami T-REX (Designed by Jo Nakashima)

Figure 2.2: Some origami models

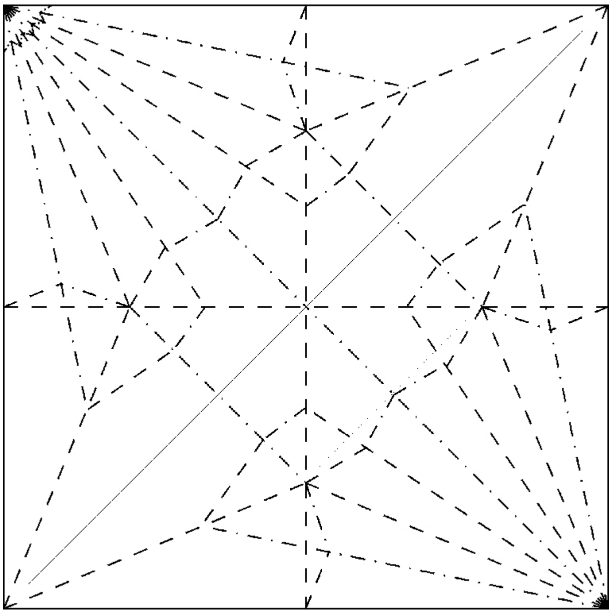
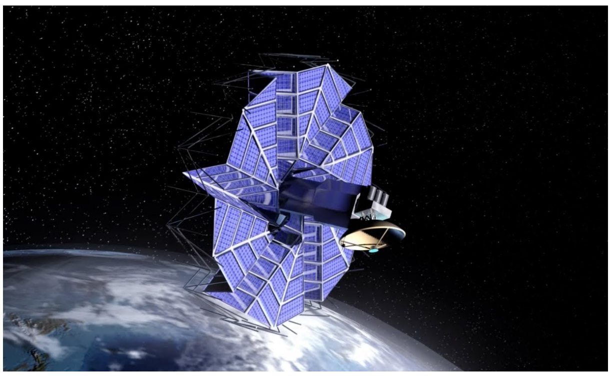


Figure 2.3: Traditional Crane crease pattern



(a)

(b)

Figure 2.4: Applications of origami in industrial

lem, traveling salesman problem, machine scheduling problem, sudoku problem, minimum spanning tree problem, etc. The computational complexity of tasks in this field is usually NP-complete and NP-hard. Therefore, brute-force approaches are not suitable in these case.

Many studies and theories in computational complexity, algorithm, computational methods related to combinatorial optimization. It has important applications in several fields, including mathematics, computational science, and engineering, etc.

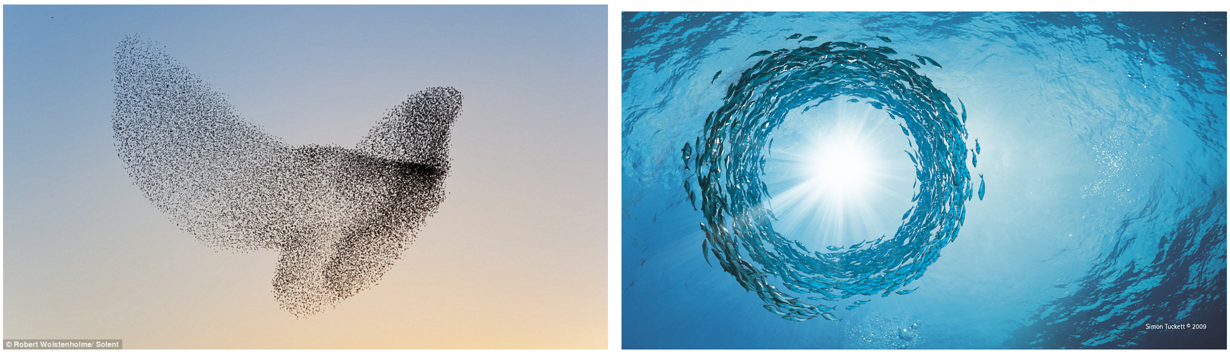
2.3 Particle Swarm Optimization (PSO)

As we defined our problem as a combinatorial optimization problem, we can apply many possible solutions to solve the problem. Some instances are,

- Exhaustive search algorithm. In real-life combinatorial optimization, the problems usually are exponential growth with the size of inputs. In this case, the exhaustive search algorithm is only suitable for the small data. It will take days, years or forever to find out a solution for large size issues.
- Simple heuristics (greedy algorithm, randomized algorithm). The simple searching strategies are indicated in the name of this type of algorithm. With complex systems, this method can not guarantee an acceptable optimal solution because of the lack of elaborate in the underlying heuristic function.
- Classical optimization methods (optimization algorithms, iterative methods). With this approach, our problem is required to be differentiable. But sometimes, It is too difficult to differentiate a complicated function.
- Meta-heuristics
- Etc,

From these analyses, in this thesis, we choose particle swarm optimization as a meta-heuristics algorithm that can provide an approximate solution to our problem.

In the period 1995 - 1998, from the ideas and the workings of Kennedy and Eberhart (1995) [26–30], the Particle Swarm Optimization algorithm has been presented. Until now, this biologically inspired algorithm contributed an efficient way to optimize and search optimal solutions in computational science. Communication is the most important skill in life. Members of groups or societies (bird flock and fish school in Figure 2.5, group of people, etc.) broadcast their information, their experiences. They gain knowledge, skills by talking and listening. In this way, each in this society can help itself, also help others to optimize the performing specific tasks. By considering the way nature works, PSO simulates the communication between individuals in society, and solve a particular



(a) Bird flock (Credit to Robert Wolstenholme) (b) Fish school (Credit to Simon Tuckett)

Figure 2.5: A bird flock and a fish school in real life

optimization problem.

Research has been reported to explain and evaluate PSO algorithm. The most popular book about PSO is *Swarm Intelligence* by Kennedy and Eberhart [31]. They portray numerous philosophical part of PSO and swarm intelligence. A broad overview of PSO applications is made by Poli [32]. Recently, PSO has been researched in theoretical and experimental aspects on an in-depth analysis which has been published by Bonyadi and Michalewicz [33].

In computer science, with a given cost function, PSO iteratively trying to find a position in the search space which optimizes the objective function. To achieve this task, this computational method defines two main concepts are a swarm and list members of swarm called particles. Particles, at first, are provided its position in the search space as well as its velocity. With velocity, these objects are able to move toward between positions based on a simple mathematical formula. In the searching process, the swarm usually find a way to access the best position as fast as possible in search space by using information of best-known particle and communicate with neighbors. Finally, the solution proposed by the swarm is the position of the best particle in this set.

In optimization problems, PSO with its characteristics, have many advantages. First, it is a problem-independent algorithm; we can easily adapt this method to any types of task. Next, a vast search space can be analyzed with a small effort by using PSO. Compare with simple heuristics and classical optimization problems, the number of calculation task in PSO is lower. Furthermore, the requirement that input problem needs to be differentiable is not necessary for this approach.

Today, PSO has proved the practical and the suitable in many continuous optimization problems [30, 34–38]. One of the most popular computational technique for optimization is PSO.

2.3.1 Standard PSO (SPSO)

As discussed in Section 2.3, a basic variant of the PSO algorithm contains a list of feasible solutions, called particles, and are members of a swarm. These particles know their positions and velocities. Each particle applies the objective function to evaluate its position. It also uses its velocity to moved to new positions, particle's moving function usually is a user-defined function which depends on the optimization problem. And to know the direction of the movements, the particle's current direction information, the particle's best-known position information, as well as the swarm's best-known position information, are combined to guided the particle. When a particle discovers a new best position, it will communicate and update to the entire swarm. The process of updating and moving of the swarm is repeated and by doing so it is hoped, but not guaranteed that an optimal solution will eventually be explored.

Formally,

- Let x is a feasible solution as well as a particle of the swarm.
- Let A is a set of all feasible solutions (or search space).
- Let $f(x) : R^n \to R$ is the objective function. It indicates that this function evaluates particles' position. The gradient of $f(x)$ is not known.
- PSO algorithm need to find a feasible solution x_0 such that $f(x_0) \leq f(x)$ (*"minimization"*) or $f(x_0) \geq f(x)$ (*"maximization"*) where $x, x_0 \in A$. x_0 is an optimal solution.

Let S be the number of particles in the swarm, each having a position $x_i \in R^n$ in the search-space and a velocity $v_i \in R^n$. Let p_i be the best-known position of particle i and let g be the best-known position of the entire swarm. A basic PSO algorithm is introduced in Algorithm 1 [39].

The performance of optimization algorithm depends on the selection of PSO parameters. How to decide and choose suitable PSO parameters are reported in many articles [35, 40–43]. Some PSO parameters tuning techniques are using another overlaying optimizer, a concept known as meta-optimization [44–46], or even fine-tuned during the optimization, e.g., utilizing fuzzy logic [47].

As described, obviously, we can see that the most importance things in PSO algorithm are particles. And the core parts of particles are velocity and position. The behavior of particles is directly effected by velocity. To moving, three information are processed as the following (Figure 2.6 [48]),

- particle's current direction
- particle's previous best position

Algorithm 1 Standard particle swarm optimization

Randomize the position of particles
Randomize the velocity of particles
while Termination condition not reached **do**
    for Each particle i **do**
        for Each dimension d **do**
            Pick random numbers $r_p, r_g \sim U(0, 1)$
            Update the particle's velocity $v_{i,d} \leftarrow \omega v_{i,d} + \varphi_p r_p(p_{i,d} - x_{i,d}) + \varphi_g r_g(g_d - x_{i,d})$
        end for
        Update the position of the particle $x_i \leftarrow x_i + v_i$
        Evaluate the fitness $f(x_i)$
        if $f(x_i) > f(p_i)$ **then**
            Update the particle's best known position $p_i \leftarrow x_i$
            if $f(p_i) > f(g)$ **then**
                Update the swarm's best known position $g \leftarrow p_i$
            end if
        end if
    end for
end while
g is the proposed solution of the algorithm
The parameters ω, φ_p and φ_g are defined by the properties of the problem. And they control the efficient and behavior of the algorithm.

- swarm's best known position

Formalized by the following equations,

$$\begin{cases} v_i^{t+1} \leftarrow \omega v_i^t + \varphi_p r_p (p_i^t - x_i^t) + \varphi_g r_g (g^t - x_i^t) \\ x_i^{t+1} \leftarrow x_i^t + v_i^{t+1} \end{cases} \tag{2.1}$$

Where

- v_i^t velocity at time step t of particle i
- x_i^t position at time step t of particle i
- p_i^t particle i's best known position at time step t
- g^t swarm's best known position at time step t
- $\omega, \varphi_p r_p, \varphi_g r_g \sim U(0,1)$ social/cognitive confidence coefficients

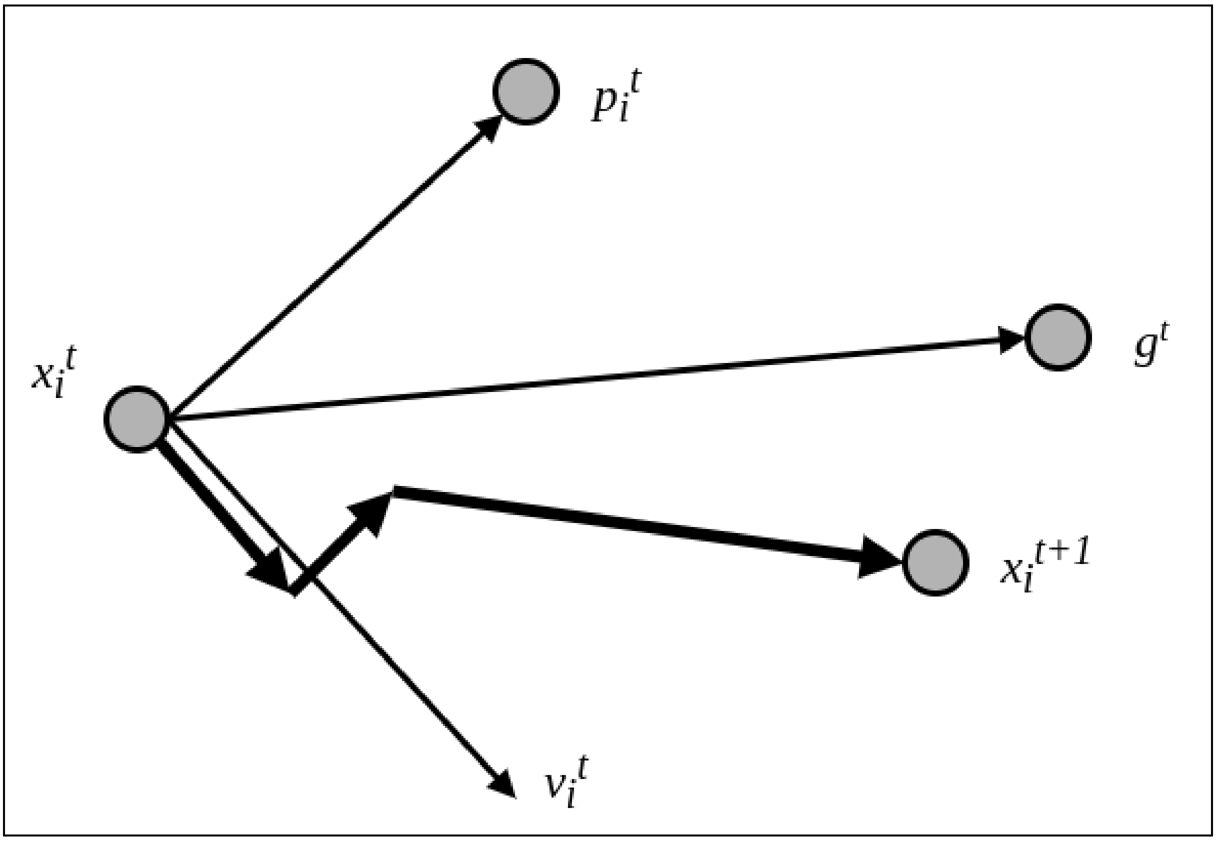


Figure 2.6: Particle motion

Basically, we can modify PSO algorithm, make it more suitable, flexible and efficient with various types of optimization problem by defining the concepts and operators in

Equation 2.1. Some PSO variants are Hybridization PSO; Alleviate premature PSO; Simplifications PSO; Multi-objective Optimization PSO; Binary, discrete, and combinatorial PSO.

Chapter 3

Methodologies

In order to solve our problem, we proposed a workflow process as in Figure 3.1. In this chapter, the first three parts in this process will be discussed. The final part is talked in Chapter 4. The way we define our subject is in Section 3.1. Also, in this section, how to model our problem as a combinatorial optimization problem is suggested. The proposed approach to solving the modeled problem is presented in Section 3.2. Finally, the folding simulation that is used in this research is covered in Section 3.3. Examples are provided through all the sections in this chapter.

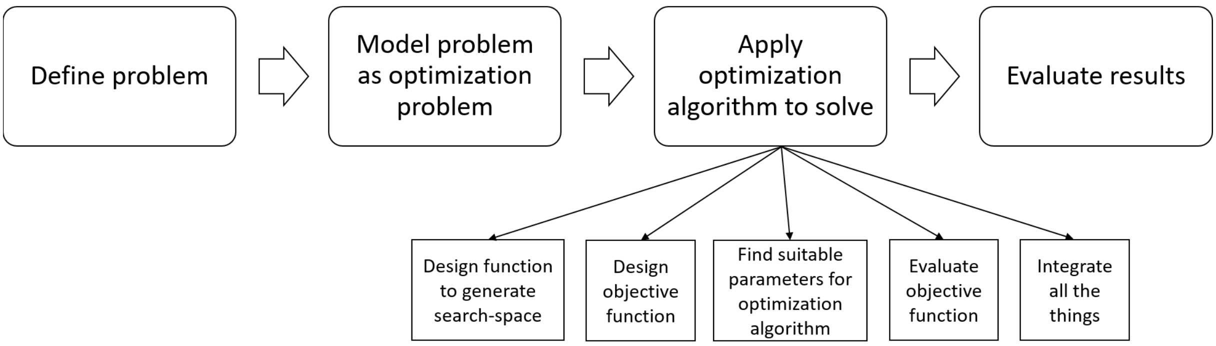


Figure 3.1: Process to solve problem

3.1 Modeling as Combinatorial Optimization Problem

In this research, we provided an approach to model our problem as a combinatorial optimization problem. The details of this method will be analyzed in this section. Personally, this is the most creative work in our entire research process. Firstly, the definition of the problem will be presented in Section 3.1.1. Secondly, the core parts of a combinatorial optimization, are the feasible solutions, the search space, and the objective function will be properly defined in Section 3.1.2, Section 3.1.3 and Section 3.1.4. Finally, to help readers understanding this section and respond to it more fully, an example will be showed in Section 3.1.5.

3.1.1 Problem Definition

We consider the problem as follows (Figure 3.2):

Input

- An origami paper (flat sheet square of paper) $OriginalPaper$
- A set of n predefined creases $ActionSet = \{Action_1, Action_2, ..., Action_n\}$
- An object model from real life or creative work as objective model $InputObject$

Output

- A folding sequence from the predefined crease pattern x_{best}
- An object model $OutputObject$ is built by applied the folding sequence x_{best} to the $OriginalPaper$
- The similarity of the objective model $InputObject$ and the object model $OutputObject$ must be maximize

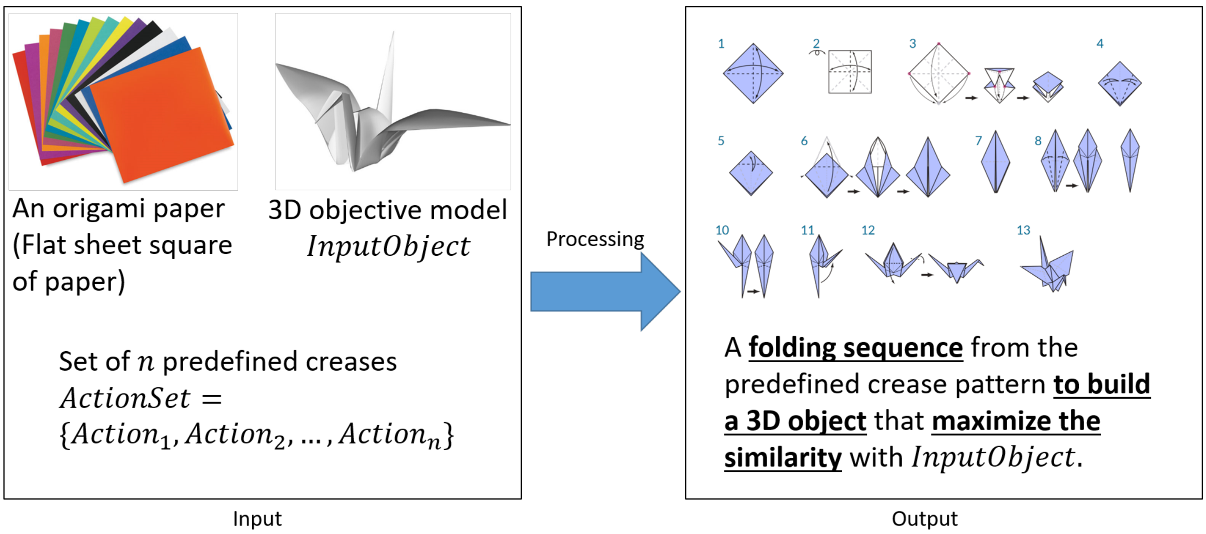


Figure 3.2: Problem definition

3.1.2 Feasible Solution

Action$_0$

We define a special folding action is called $Action_0$. When we use the $Action_0$ to fold the origami paper, the current shape of this paper will not be changed. After adding the $Action_0$ to the $ActionSet$, our new $ActionSet$ is

$$ActionSet = \{Action_0, Action_1, Action_2, ..., Action_n\}$$

Folding process

A folding process is a list of actions we get from $ActionSet$. The meaning of a folding process is that we will apply these actions in the list to the origami paper one by one to make a specific shape of origami.

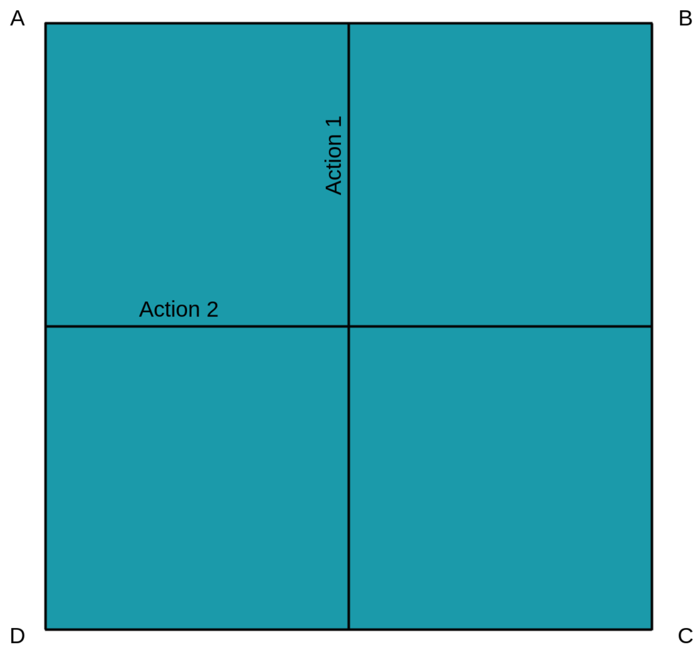


Figure 3.3: Example of origami with 2 folding actions

For example, in Figure 3.3, we have an origami with 2 folding actions. When we apply the $Action_1$, the edge AD will be moved to the edge BC. Similarly, the edge AB will be moved to the edge DC when the $Action_2$ be processed. And the $ActionSet$ in this case is,

$$ActionSet = \{Action_0, Action_1, Action_2\}$$

Some folding processes are,

- $Action_0$
- $Action_1 \rightarrow Action_2$
- $Action_1 \rightarrow Action_1$
- $Action_1 \rightarrow Action_2 \rightarrow Action_1 \rightarrow Action_2$
- $Action_0 \rightarrow Action_0 \rightarrow Action_0 \rightarrow Action_0$
- Etc,

But,

- $Action_1 \rightarrow Action_2 \rightarrow Action_3$ is not a folding process ($Action_3 \notin ActionSet$)
- Etc,

Feasible solution

And from these description, **feasible solution** x (or candidate solution) is a folding process with exact length n ($n - tuples$ of integer).

$$\begin{cases} x = (x_1, x_2, ..., x_n) \\ x_i \in ActionSet \\ 1 \leq i \leq n \end{cases}$$

Consider the example in Figure 3.3, in this case $n = 2$. We have some feasible solutions are,

- $x = Action_0 \rightarrow Action_0$
- $x = Action_0 \rightarrow Action_1$
- $x = Action_1 \rightarrow Action_2$
- Etc,

But

- $x = Action_0$ is not a feasible solution ($length = 1 < 2$)
- $x = Action_1 \rightarrow Action_2 \rightarrow Action_1 \rightarrow Action_2$ is not a feasible solution ($length = 4 > 2$)
- $x = Action_1 \rightarrow Action_3$ is not a folding process ($Action_3 \notin ActionSet$)
- Etc,

3.1.3 Search Space

Define $search - space$ A (or choice set) is a set of all feasible solutions. Typically, A is $n - dimensional$ integer lattice, denoted Z^n, is the lattice in the Euclidean space R^n. Because each candidate solution x is a vector with n elements, and we also have $n + 1$ candidate for each element, then the set A has a cardinality (the total number of feasible solutions) of $(n + 1)^n$. The size of the $search - space$ A increases exponentially with the number of folding actions. In Table 3.1 is the size of the $search - space$ A with n from 1 to 10.

With the example in Figure 3.3, we have $ActionSet = \{Action_0, Action_1, Action_2\}$ and $n = 2$. The $search - space$ A has a cardinality of 9. All the feasible solutions are,

- $x_1 = Action_0 \rightarrow Action_0$
- $x_2 = Action_0 \rightarrow Action_1$
- $x_3 = Action_0 \rightarrow Action_2$
- $x_4 = Action_1 \rightarrow Action_0$
- $x_5 = Action_1 \rightarrow Action_1$
- $x_6 = Action_1 \rightarrow Action_2$
- $x_7 = Action_2 \rightarrow Action_0$
- $x_8 = Action_2 \rightarrow Action_1$
- $x_9 = Action_2 \rightarrow Action_2$

Figure 3.4 is the demonstration of the $search - space$ A and the feasible solutions with $n = 2$ and $n = 3$. A green point is a feasible solution.

3.1.4 Objective Function

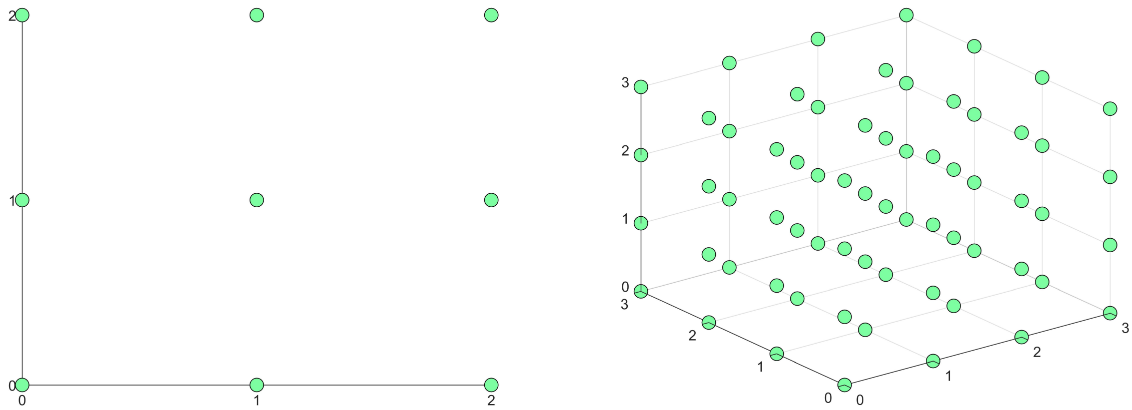
Let $g(x)$ is a function that converts the feasible solution x into an object. With the $ActionSet$ as in Figure 3.3, we have some examples of $g(x)$ in Figure 3.5. In Figure 3.5a is the origami paper after we call $g(x = Action_0 \rightarrow Action_1)$. And when $g(x = Action_1 \rightarrow Action_2)$ processed, the object in Figure 3.5b is the result.

Let $f(x) : Z^n \rightarrow R$ is a function that compares the similarity between $g(x)$ and $InputObject$. The lower value means more similarity and the higher value means more difference. The function f is called the objective function (or cost function).

We can accurately define our problem as an optimization problem in the following way:
Given function $f : A \rightarrow R$ from some set A to the real numbers
Sought an element x_{best} in A such that $f(x_{best}) \leq f(x)$ for all x in A("minimization").

n	Size of A
1	2
2	9
3	64
4	625
5	7,776
6	117,649
7	2,097,152
8	43,046,721
9	1,000,000,000
10	25,937,424,601

Table 3.1: The size of the *search space* A with $n \in [1, 10], n \in Z$



(a) With $n = 2$, we have 9 feasible solutions (b) With $n = 3$, we have 64 feasible solutions

Figure 3.4: Examples of search space with $n = 2$, $n = 3$

(a) $g(x = Action_0 \rightarrow Action_1)$ (b) $g(x = Action_1 \rightarrow Action_2)$

Figure 3.5: Examples of function $g(x)$

3.1.5 Example

In this section, a full example of our problem will be discussed. We will present an original paper, a set of folding actions, and some objective models. From these information, we will provided the feasible solutions, as well as the search space. The origami model after applied a specific feasible solution also will be showed. And finally, we optimize the objective functions to find the optimal solutions.

- In Figure 3.6 is the back and front of the original paper. The paper has the size 60x60.
- We use the same $ActionSet$ as in Figure 3.3. $ActionSet = \{Action_0, Action_1, Action_2\}$ and the number of folding actions is $n = 2$.
- In Figure 3.7 are 4 examples of objective models.

As discussed in Section 3.1.3, we can enumerate all the feasible solutions as the following,

- $x_1 = Action_0 \rightarrow Action_0$
- $x_2 = Action_0 \rightarrow Action_1$
- $x_3 = Action_0 \rightarrow Action_2$
- $x_4 = Action_1 \rightarrow Action_0$
- $x_5 = Action_1 \rightarrow Action_1$
- $x_6 = Action_1 \rightarrow Action_2$
- $x_7 = Action_2 \rightarrow Action_0$
- $x_8 = Action_2 \rightarrow Action_1$
- $x_9 = Action_2 \rightarrow Action_2$

We apply the function $g(x)$ for each feasible solution, and the origami paper after applied a specific feasible solution is presented in Figure 3.8.

To obtain the best candidate, for each objective model, we compare this model with all object models of feasible solutions. And the object model with the most similarity indicates the optimal solution.

- For the objective model in Figure 3.7a, the optimal solutions are x_3, x_7, x_9.
- For the objective model in Figure 3.7b, the optimal solutions are x_2, x_4, x_5.
- For the objective model in Figure 3.7c, the optimal solutions are x_6, x_8.
- For the objective model in Figure 3.7d, the optimal solutions are x_2, x_4, x_5.

(a) Front of the original paper (b) Back of the original paper

Figure 3.6: The original paper with the size 60x60

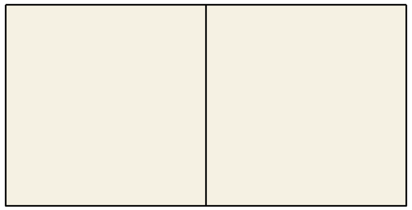
3.2 Discrete PSO (DPSO)

We defined our problem as a combinatorial optimization problem in Section 3.1. In this section, an approach to use DPSO to find an answer to our modeled problem is discussed. At first, we provide some basic information about DPSO as well as discretization methods. From the knowledge of SPSO (Section 2.3.1) and DPSO, our proposed DPSO algorithm is presented in Section 3.2.1. In Section 3.2.2, a support function to generate particles, that is used in our algorithm, is talked. Examples are also provided.

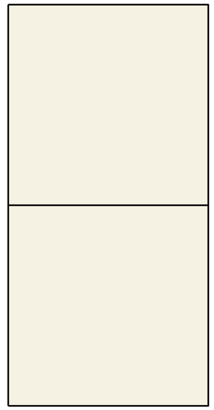
In many continuous state-space, original PSO algorithm proved its advantages are straightforward and efficient technique. In fact, optimization problems included continuous optimization problem and discrete optimization problem. Many important problems are set in a space featuring discrete. Consequently, requesting to modify, apply PSO algorithm to solve combinatorial optimization problem has been raised. It has become an attractive subject for years. In 1997, the authors of original PSO, Kennedy and Eberhart, proposed a discrete binary version of PSO that is called binary PSO (BPSO) [49]. From that day, several approaches about BPSO and DPSO have been developed in the literature [48, 50–53]. These modified PSO algorithms have demonstrated promising execution on benchmark examples.

Obviously, to use PSO for discrete problems, we need a method to represent the position of a particle in discrete space. Krause [54] characterize the codification of candidate solutions in three encoding schemes:

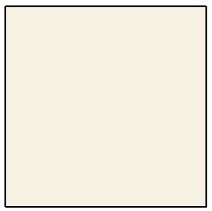
- Binary codification (BC) for candidate solutions.
- Integer codification (IC) for candidate solutions.
- Using transformation methods to transform real values into a BC (real-to-binary: RTB) or an IC (real-to-integer: RTI), where RTI represents a combination of integer values. These transformations have to be done at each iteration loop.



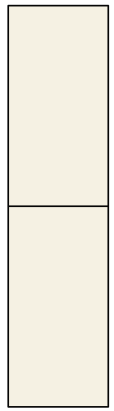
(a) Objective model 1



(b) Objective model 2

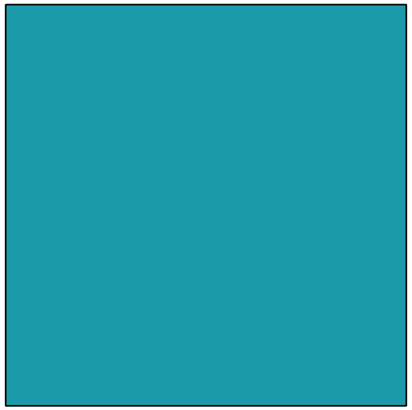


(c) Objective model 3

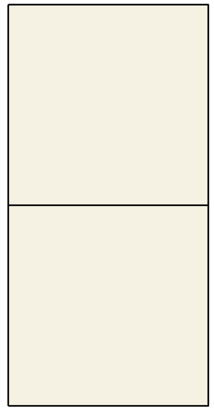


(d) Objective model 4

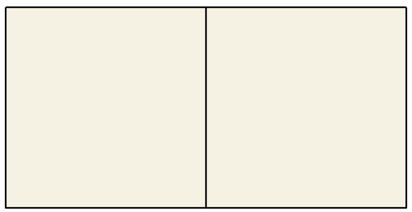
Figure 3.7: Examples of objective models



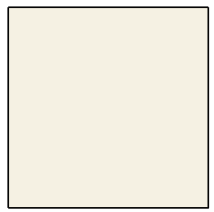
(a) Object model of x_1



(b) Object model of x_2, x_4, x_5



(c) Object model of x_3, x_7, x_9



(d) Object model of x_6, x_8

Figure 3.8: Origami paper after applied a specific feasible solution

They also categorize the discretization methods which are used in literature as follow:

- Sigmoid Function [49]
- Random-key [55, 56]
- Smallest Position Value [57]
- Modified Position Equation [58, 59]
- Great Value Priority [60]
- Nearest Integer [61]

3.2.1 Proposed Algorithm

As discussed SPSO (Section 2.3.1) and DPSO (Section 3.2), we proposed a DPSO algorithm to solve our problem in this section. We talked about moving function of a particle in Section 2.3.1. Now, we will consider the equation of motion Equation 2.1 to adapt to our problem. The important things is to be able to define the following objects and mathematical operations,

- *position* of particles
- *velocity* of particles
- Subtraction operator $position \ominus position \to velocity$
- External multiplication operator $\mu \otimes velocity \to velocity$
- Addition operator $velocity \oplus velocty \to velocity$
- Displacement operator $position \oplus velocity \to position$

Assumed we have an instances of problem with a set of n predefined creases

$$ActionSet = \{Action_1, Action_2, ..., Action_n\}$$

By modeling method in Section 3.1.2, we add $Action_0$ to $ActionSet$, now

$$ActionSet = \{Action_0, Action_1, Action_2, ..., Action_n\}$$

Let fs is a feasible solution,

$$\begin{cases} fs = (fs_1, fs_2, ..., fs_n) \\ fs_i \in ActionSet \\ 1 \le i \le n \end{cases}$$

Using $ActionSet$ and fs, we define these concepts one by one as the followings,

position of particles

Let x is the position of particle fs. Then x is a $1-by-n$ vector. Each element i of x is an integer in $[0, n]$ that is the index of fs_i in $ActionSet$. Formally,

$$\begin{cases} x = (x_1, x_2, ..., x_n) \\ x_i \in [0, n] \\ x_i \in Z \\ 1 \leq i \leq n \end{cases}$$

For examples,

- $fs = Action_0 \rightarrow Action_1 \Rightarrow x = (0, 1)$
- $fs = Action_0 \rightarrow Action_0 \Rightarrow x = (0, 0)$
- $fs = Action_1 \rightarrow Action_1 \Rightarrow x = (1, 1)$
- $fs = Action_1 \rightarrow Action_2 \Rightarrow x = (1, 2)$

velocity of particles

Because *velocity* is applied to particles after each time step to change the position so *velocity* in this case is instantaneous velocity. Let v is the velocity of particle fs. Then v is a $1-by-n$ vector. Each element i of v is an integer in $[V_{min}, V_{max}]$ that represents the rate of change of element x_i in dimension i. Obviously, in our problem, the farthest distance in each dimension is n, and the nearest distance is 0. So, $V_{min} = 0$ and $V_{max} = n$. Formally,

$$\begin{cases} v = (v_1, v_2, ..., v_n) \\ v_i \in [0, n] \\ v_i \in Z \\ 1 \leq i \leq n \end{cases}$$

Subtraction operator $position \ominus position \rightarrow velocity$

Let $v^{ab} = x^a \ominus x^b$ where x^a, x^b are particle's positions and v is *velocity*. We define,

$$\begin{cases} v^{ab} = (v_1^{ab}, v_2^{ab}, ..., v_n^{ab}) \\ v_i^{ab} = (x_i^a - x_i^b) \mod (n+1) \\ 1 \leq i \leq n \end{cases}$$

For examples,

- $(1, 2, 3) \ominus (1, 2, 3) = (0, 0, 0)$
- $(1, 2, 3) \ominus (3, 2, 1) = (2, 0, 2)$
- $(1, 2, 3) \ominus (0, 0, 0) = (1, 2, 3)$

External multiplication operator $\mu \otimes velocity \rightarrow velocity$

Let $v' = \mu \otimes v$ where $\mu \in R$, v' and v are *velocities*. We have,

$$\begin{cases} v' = (v'_1, v'_2, ..., v'_n) \\ v'_i = \lceil \mu v_i \rceil \mod (n+1) \\ 1 \leq i \leq n \end{cases}$$

For examples,

- $0.5 \otimes (1, 2, 3) = (1, 1, 2)$
- $2.7 \otimes (1, 2, 3) = (3, 2, 1)$
- $-0.5 \otimes (1, 2, 3) = (0, 3, 3)$

Addition operator $velocity \oplus velocty \rightarrow velocity$

Let $v^{ab} = v^a \oplus v^b$ where v^a, v^b, v^{ab} are *velocities*. We define,

$$\begin{cases} v^{ab} = (v^{ab}_1, v^{ab}_2, ..., v^{ab}_n) \\ v^{ab}_i = (v^a_i + v^b_i) \mod (n+1) \\ 1 \leq i \leq n \end{cases}$$

For examples,

- $(0, 0, 0) \oplus (0, 0, 1) = (0, 0, 1)$
- $(3, 3, 3) \oplus (3, 3, 3) = (2, 2, 2)$
- $(1, 2, 3) \oplus (3, 2, 1) = (0, 0, 0)$

Displacement operator $position \oplus velocity \rightarrow position$

Let $x' = x \oplus v$ where x, x' are positions and v is velocity. We define,

$$\begin{cases} x' = (x'_1, x'_2, ..., x'_n) \\ x'_i = (x_i + v_i) \mod (n+1) \\ 1 \leq i \leq n \end{cases}$$

For examples,

- $(1, 2, 3) \oplus (0, 0, 0) = (1, 2, 3)$
- $(3, 3, 3) \oplus (3, 3, 3) = (2, 2, 2)$
- $(1, 2, 3) \oplus (3, 2, 1) = (0, 0, 0)$

From these definitions, the equation of motion for our problem is properly modified as in Equation 3.1,

$$\begin{cases} v_i^{t+1} \leftarrow \omega \otimes v_i^t \oplus \varphi_p r_p \otimes (p_i^t \ominus x_i^t) \oplus \varphi_g r_g \otimes (g^t \ominus x_i^t) \\ x_i^{t+1} \leftarrow x_i^t \oplus v_i^{t+1} \end{cases} \tag{3.1}$$

And from modified equation, we proposed the DPSO algorithm as in Algorithm 2.

Algorithm 2 Proposed discrete particle swarm optimization

$V_{min} \leftarrow 0$
$V_{max} \leftarrow$ Number of dimensions
for Each particle i **do**
    $x_i \leftarrow InitializeParticle$(Number of dimensions) (Algorithm 3)
    for Each dimension d **do**
        $v_{i,d} \sim U\{V_{min}, V_{max}\}$
    end for
end for
while Termination condition not reached **do**
    for Each particle i **do**
        for Each dimension d **do**
            Pick random numbers $r_p, r_g \sim U(0, 1)$
            Update the particle's velocity
            $v_{i,d} \leftarrow \omega \otimes v_{i,d} \oplus \varphi_p r_p \otimes (p_{i,d} \ominus x_{i,d}) \oplus \varphi_g r_g \otimes (g_d \ominus x_{i,d})$
        end for
        Update the position of the particle $x_i \leftarrow x_i \oplus v_i$
        Evaluate the fitness $f(x_i)$
        if $f(x_i) > f(p_i)$ **then**
            Update the particle's best known position $p_i \leftarrow x_i$
            if $f(p_i) > f(g)$ **then**
                Update the swarm's best known position $g \leftarrow p_i$
            end if
        end if
    end for
end while
g is the proposed solution of the algorithm
The parameters ω, φ_p and φ_g are defined by the properties of the problem. And they control the efficient and behavior of the algorithm.

3.2.2 Initializing Particles

A technique for initialize the particles are also designed. We will apply this algorithm in the very first part of DPSO algorithm. The idea is quite simple. We first randomize a list of folding actions. After that, for each randomized action, we continue random a index for it in range $[0, n]$. We use Fisher–Yates shuffle method in our process. The detail are presented in Algorithm 3.

Algorithm 3 Initialize the particles

Input: $n \leftarrow$ Number of dimensions
Output: $x \leftarrow 1 - by - n$ vector represents a random position of particle

$RandomLength \leftarrow U\{0, n\}$
$RandomAction \leftarrow 1 - by - n$ vector
$RandomIndex \leftarrow 1 - by - n$ vector
for Each dimension d **do**
    $x_d \leftarrow 0$
    $RandomAction_d \leftarrow d$
    $RandomIndex_d \leftarrow d$
end for
Shuffle $RandomAction$ (Fisher–Yates shuffle algorithm)
Shuffle $RandomIndex$ (Fisher–Yates shuffle algorithm)
for Each $d \in [1, RandomLength], d \in Z$ **do**
    $k \leftarrow RandomIndex_d$
    $x_k \leftarrow RandomAction_d$
end for
return x

3.3 Converting Feasible Solution into Origami Object

As introduced in Section 3.1.4, we need a function $g(x)$ to convert a feasible solution x into a corresponding object model. This object model will be used to compare with the objective model $InputObject$, from that we are able to optimize the objective function and find the optimal solution.

In this section, one possible method to construct the function $g(x)$ will be presented. Section 3.3.1 covers the overall algorithm. The procedure to calculate the similarity between object models is studied in Section 3.3.2. Throughout the sections, examples of concepts are demonstrated to support the definitions.

3.3.1 Applied Algorithm

Algorithm 4 Construct object model from feasible solution

Input: Feasible solution $x = (x_1, x_2, ..., x_n)$
Output: Origami paper after applied x

$BinaryTree \leftarrow$ Initialize a binary tree
Make the $OriginalPaper$ into the root of $BinaryTree$
$i \leftarrow 0$
while $i \leq n$ **do**
    $i \leftarrow i + 1$
    for Each $Leaf \leftarrow$ leaf of $BinaryTree$ **do**
        if $Leaf$ contains x_i **then**
            Make $Leaf$ into a new parent node - $NewParent$
            x_i divides $Leaf$ into 2 new faces
            Make 2 new faces into 2 new children of $NewParent$
        end if
    end for
end while
Combine all the leaves of $BinaryTree$ to make the object model
return Object model

In this research, we developed a virtual manipulation system for origami to build the function $g(x)$. This system was originally presented by Miyazaki et al. in 1996 [2]. We can construct simple flat folding origami models with this system.

We use faces, edges and vertexes to describe the state of the folded origami paper. Each flat of paper is presented by a face. A face contains multiple edges. Moreover, each edge has two vertexes. The folded origami paper is a list of faces.

As we build the object model, the algorithm to convert the feasible solution into an origami object are discussed in Algorithm 4. Based on the idea that each crease will separate the origami plane into two new planes. We will construct a binary tree structure. The root is the $OriginalPaper$. Each node of the tree is a face and stores the information about the relative position of the plane with the original plane. When apply a folding action, we will traverse from the root and find all the leaves that contain the crease, make these leaves into parent nodes, create two new children nodes (as two new faces which are created by folding action). Obviously, two nodes with the same parent will share one common edge. Finally, we combine all the leaves of the binary tree to get the final shape.

We have some demonstrations in Figure 3.9 and Figure 3.10. In the first example, we apply folding process $Action_1 \rightarrow Action_2 \rightarrow Action_3$. E1, E2, E3, E4, E5 are these common edges between faces. When we combine all the leaves F5, F6, F7, F8, F9, and F10 we can get the object model. The states of the origami model after each folding action are showed in Table 3.2.

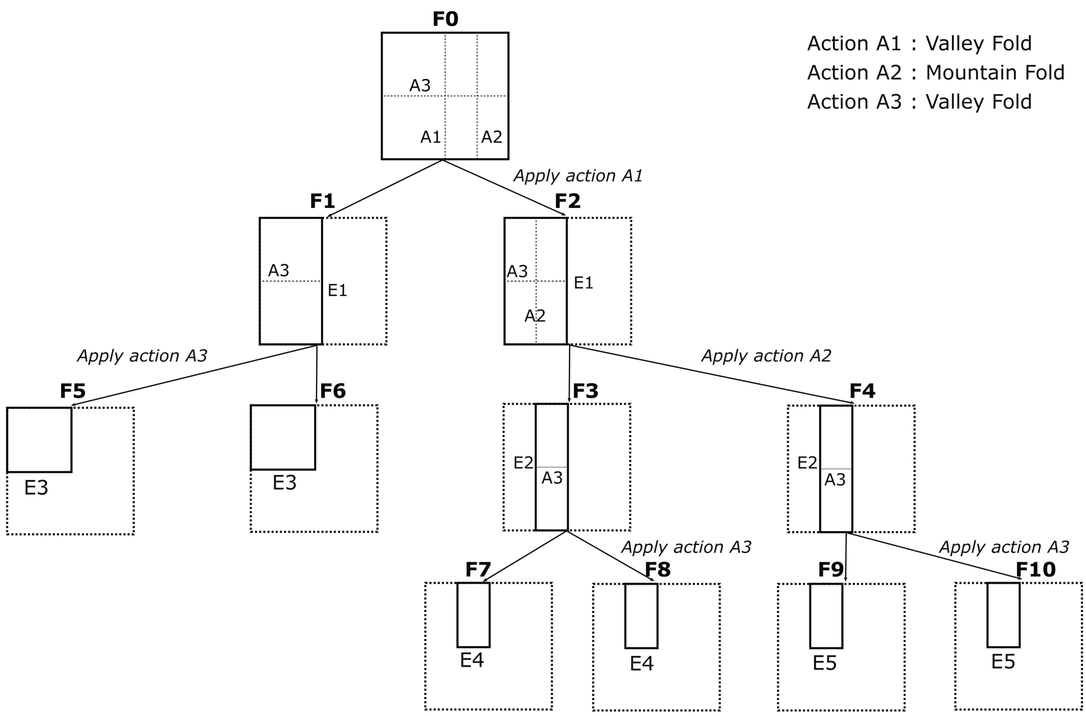


Figure 3.9: Example of how to convert feasible solutions into object model (1)

In the second example, we apply folding process $Action_1 \rightarrow Action_2 \rightarrow Action_3$. E1, E2, E3, E4, E5 are these common edges between faces. We can get the object model after combine the leaves F4, F6, F7, F8, F9, and F10. The states of the origami model after each folding action are showed in Table 3.2.

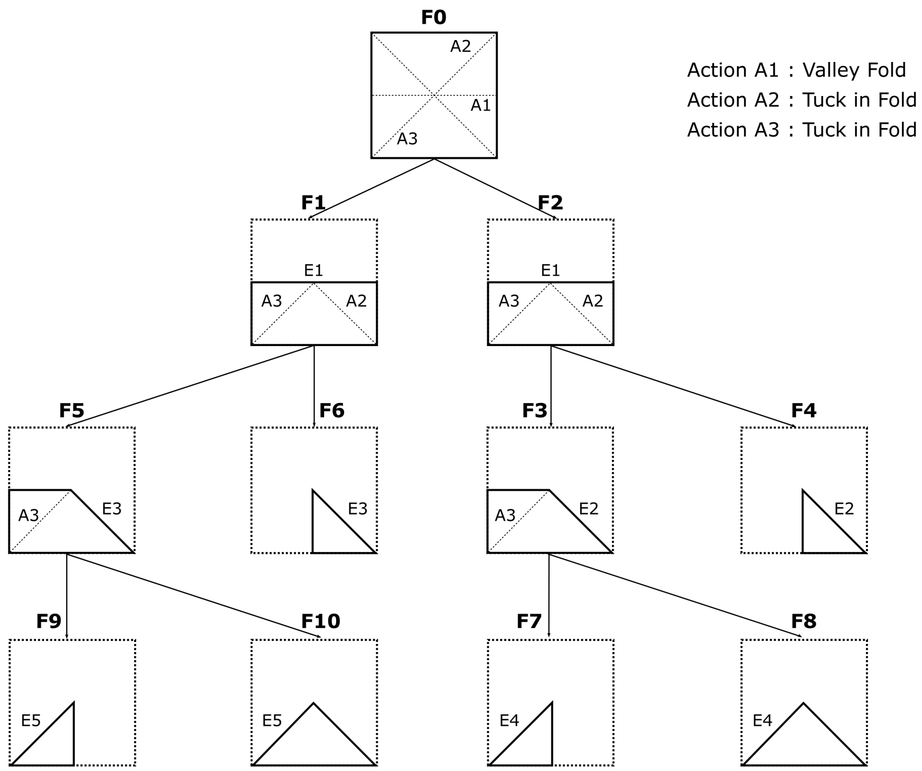


Figure 3.10: Example of how to convert feasible solutions into object model (2)

Applied action	Leaf nodes	Parent nodes	Common edges
Begin	$\{F_0\}$	$\emptyset$	
$Action_1$	$\{F_1, F_2\}$	$\{F_0\}$	$\{F_1 \cap F_2 = E_1,$
$Action_2$	$\{F_3, F_4, F_5, F_6\}$	$\{F_0, F_1, F_2\}$	$F_3 \cap F_4 = E_2, F_5 \cap F_6 = E_3,$
$Action_3$	$\{F_4, F_6, F_7, F_8, F_9, F_{10}\}$	$\{F_0, F_1, F_2, F_3, F_5\}$	$F_7 \cap F_8 = E_4, F_9 \cap F_{10} = E_5\}$

Table 3.2: The states of the origami model after each folding action in Figure 3.9

Applied action	Leaf nodes	Parent nodes	Common edges
Begin	$\{F_0\}$	$\emptyset$	
$Action_1$	$\{F_1, F_2\}$	$\{F_0\}$	$\{F_1 \cap F_2 = E_1,$
$Action_2$	$\{F_3, F_4, F_5, F_6\}$	$\{F_0, F_1, F_2\}$	$F_3 \cap F_4 = E_2, F_5 \cap F_6 = E_3,$
$Action_3$	$\{F_5, F_6, F_7, F_8, F_9, F_{10}\}$	$\{F_0, F_1, F_2, F_3, F_4\}$	$F_7 \cap F_8 = E_4, F_9 \cap F_{10} = E_5\}$

Table 3.3: The states of the origami model after each folding action in Figure 3.10

3.3.2 Calculate Similarity between Object Models

Algorithm 5 Calculate the similarity between object models

Input: 2 object models $Object_1, Object_2$
Output: Similarity of $Object_1$ and $Object_2$

Normalize $Object_1$, $Object_2$
$ObjectPCL_1 \leftarrow$ Convert $Object_1$ into point cloud data
$ObjectPCL_2 \leftarrow$ Convert $Object_2$ into point cloud data
$ResultSimilarity \leftarrow$ Hausdorff distance of $ObjectPCL_1$ and $ObjectPCL_2$
return $ResultSimilarity$ as the similarity of $Object_1$ and $Object_2$

We implemented the function to calculate the similarity between folded origami papers using point cloud data (Algorithm 5). With PCL library [62], we convert the object models into point cloud data by using the information about faces, edges, and vertexes of models. The Hausdorff distance [63,64] between 2 object models is calculated. The lower Hausdorff distance means more similar between objects and higher distance means more different. We can normalize the point cloud data in sizes and density. The similarity between object models is computed with high precision. The running time of the algorithm is also acceptable.

We have some demonstrations in Figure 3.11, Figure 3.12 and Figure 3.13. In Figure 3.11, we compare the similarity between 2 apples, because they are the same objects then the Hausdorff distance is 0.
In Figure 3.12, an apple and a little different object - a tennis ball are compared, the Hausdorff distance is 21.3.
In Figure 3.13, the Hausdorff distance between an apple and a tea cup is 39.6.

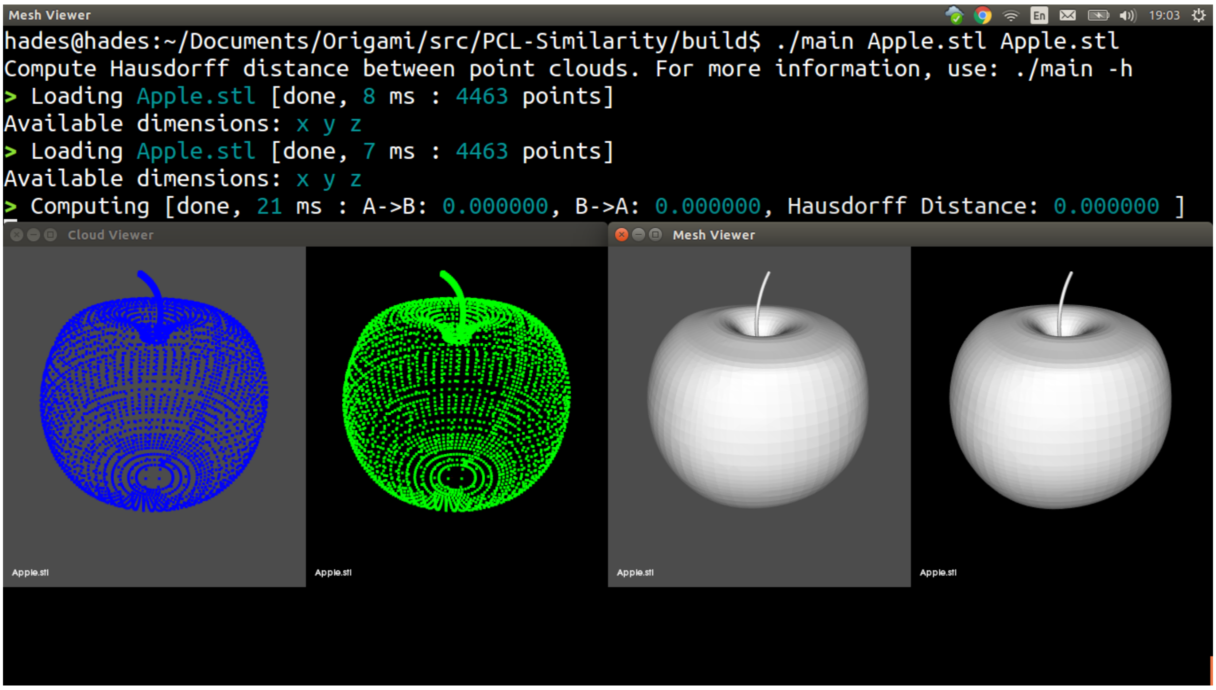


Figure 3.11: Hausdorff distance between 2 apples = 0.0

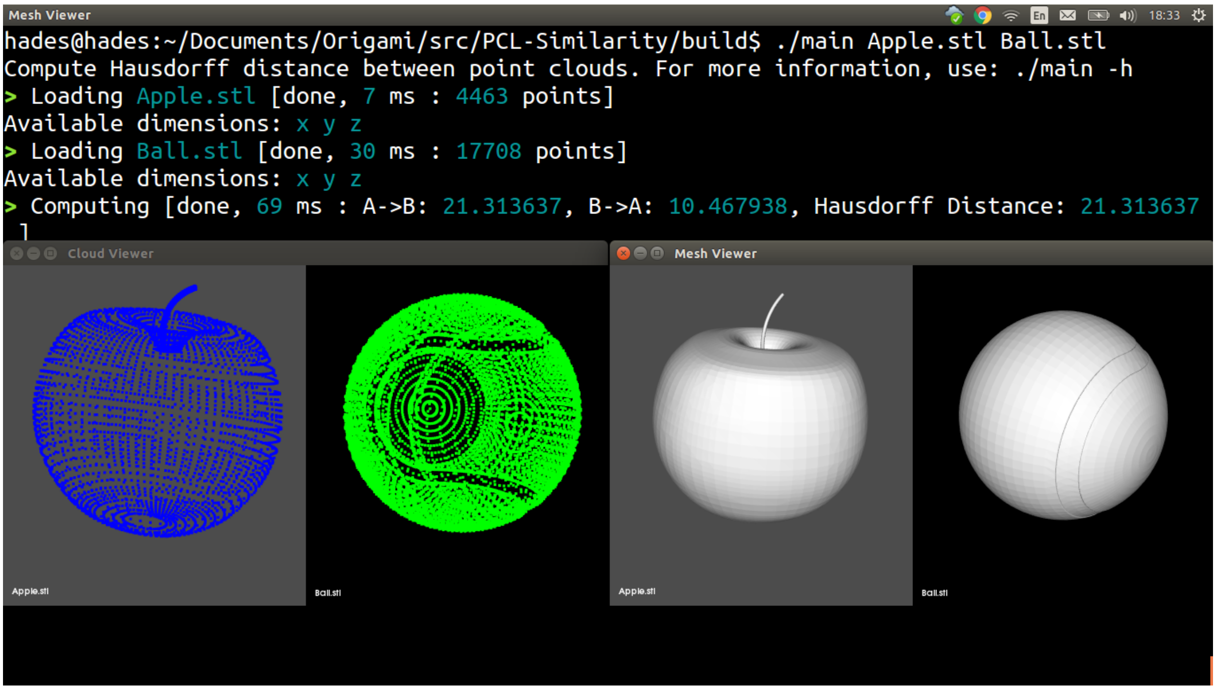


Figure 3.12: Hausdorff distance between an apple and a ball ≈ 21.31

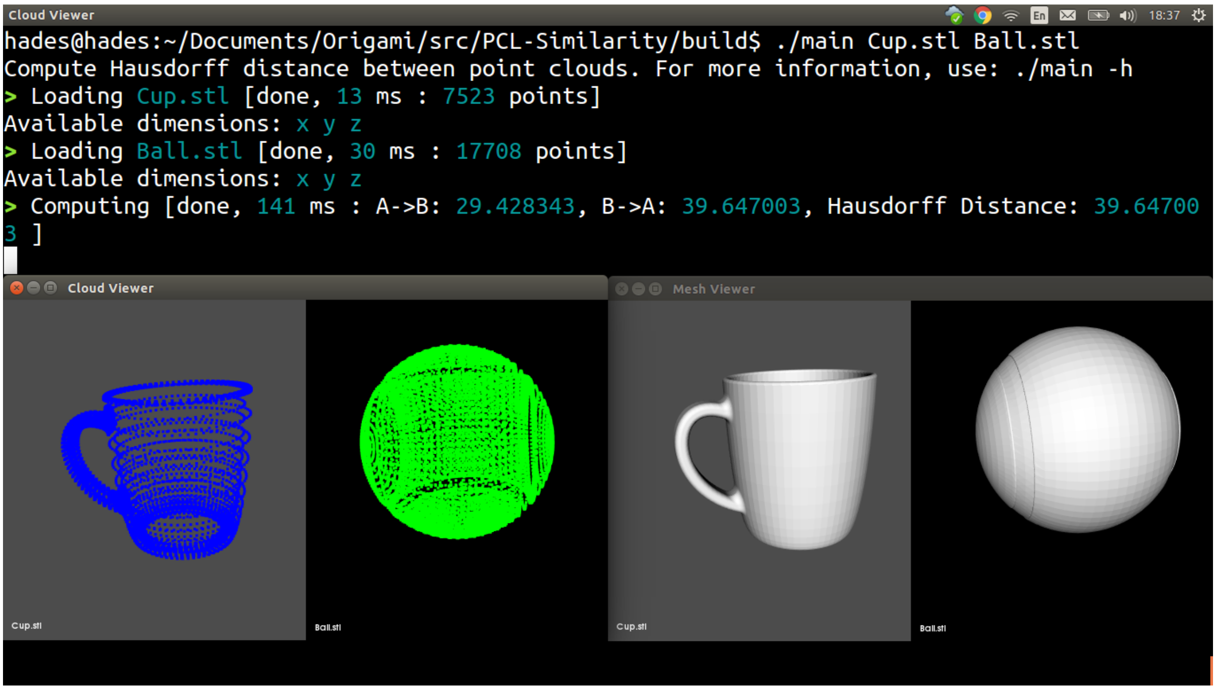


Figure 3.13: Hausdorff distance between a cup and a ball ≈ 39.64

Chapter 4

Evaluation

In this section, we will evaluate our proposed approach that has been used to solve our problem. First, we will cover some information about the original origami paper, the input predefined crease patterns, as well as the objective models, will be used in the experiments. The selection of DPSO parameters is clarified. The system for constructing and running experiments is also an important factor. This information is in Section 4.1. Next, the current results of our model will be shown and analyzed in Section 4.2.

4.1 Experimental Details

Experimental Original Origami Paper

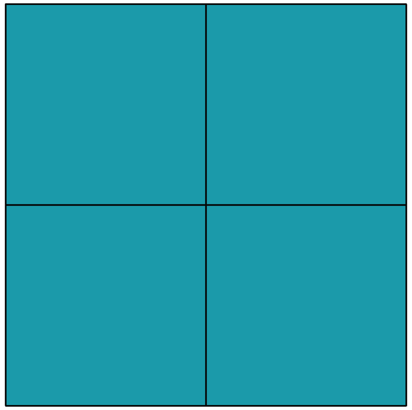
The original origami paper using in the experiments is a flat square paper with size 60x60 (same as Figure 3.6). We use this paper for all the experiments.

Experimental Predefine Crease Patterns

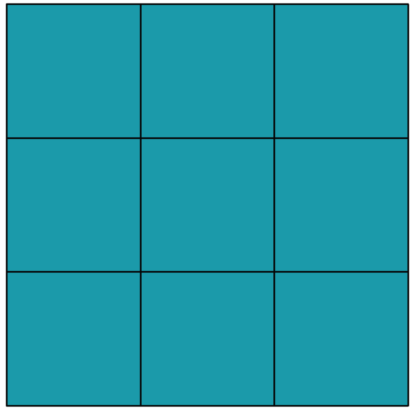
In this research, we use the crease patterns with vertical and horizontal creases. We define $StandardCP_i$ is a crease pattern with i vertical creases from $crease_1$ to $crease_i$, and i horizontal creases from $crease_{i+1}$ to $crease_{2i}$. Furthermore, the distance between these creases is equal in horizontal and vertical corresponding. They also divide the original paper into $(i + 1)$ equal parts in horizontal and vertical. Each crease represents a valley fold or a mountain fold. All the $StandardCP$ are showed in Figure 4.1 and Figure 4.2.

Experimental Objective Models

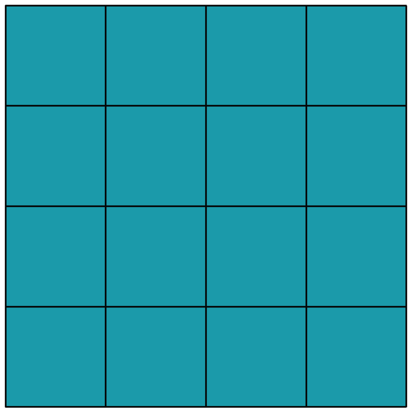
By using the same folding simulation was introduced in Section 3.3, we make some objective models to using in experiments. They are listed in Figure 4.3, Figure 4.4 and Figure 4.5.



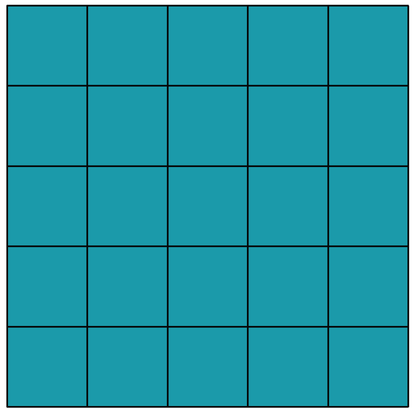
(a) $StandardCP_1$



(b) $StandardCP_2$

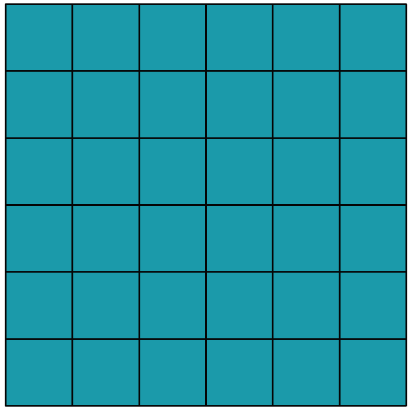


(c) $StandardCP_3$

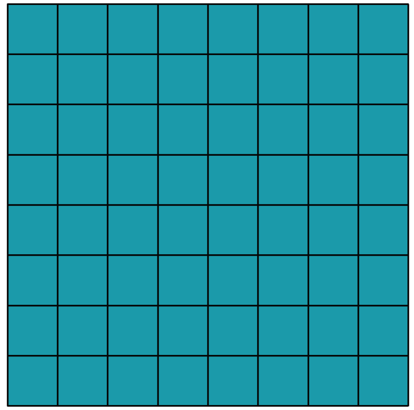


(d) $StandardCP_4$

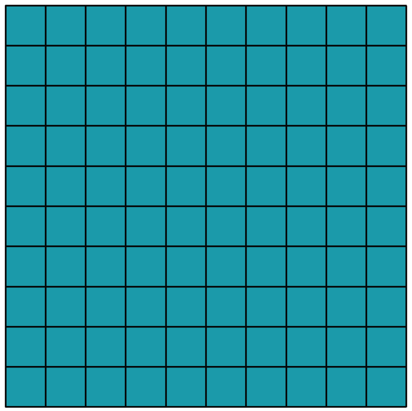
Figure 4.1: Crease patterns using in experiments (1)



(a) $StandardCP_5$

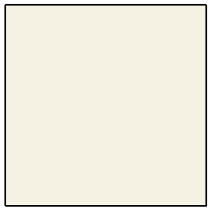


(b) $StandardCP_7$

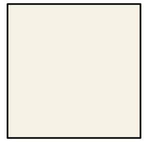


(c) $StandardCP_9$

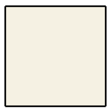
Figure 4.2: Crease patterns using in experiments (2)



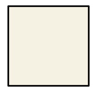
(a) Objective Model 1



(b) Objective Model 2

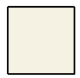


(c) Objective Model 3

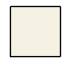


(d) Objective Model 4

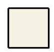
Figure 4.3: Objective models using in experiments (1)



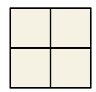
(a) Objective Model 5



(b) Objective Model 6

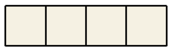


(c) Objective Model 7

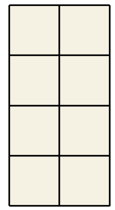


(d) Objective Model 8

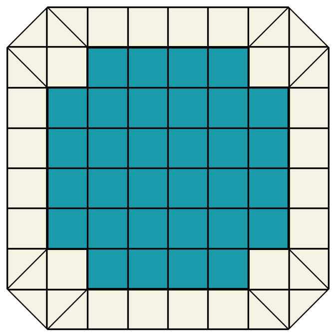
Figure 4.4: Objective models using in experiments (2)



(a) Objective Model 9



(b) Objective Model 10



(c) Objective Model 11

Figure 4.5: Objective models using in experiments (3)

DPSO Parameters

Using knowledge have been studied by Clerc in *Standard Particle Swarm Optimization* [39], we choose the parameters of DPSO algorithm as following,

- The swarm size $S = 35 + B(10, 0.5)$. $B(10, 0.5)$ is a binomial distribution with 10 is the number of trials and 0.5 is success probability in each trial.
- The maximum number of iterations is 100
- $\omega \simeq 0.721$
- $\varphi_p = 1$
- $\varphi_g = 1$

Experimental running system

- Text editor *Visual Studio Code 1.9.0*
- Programming language *C++11*
- Compiler *g++ with CMake*
- Operating system *Ubuntu 14.04 LTS 64-bit*
- Computer *Intel Core i7 CPU 3.20GHz x 8 with 12 GB of RAM*

4.2 Results and Analyses

We report the data of our approach on 12 test sets in Table 4.1.

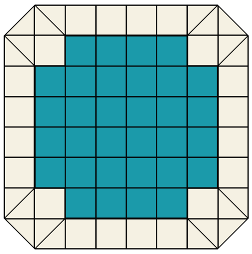
From experiment E1-E7, we try to test how the algorithm folds the crease patterns and make the smallest possible square that can make from this crease pattern. More specifically, the method needs to fold a square with size $\frac{1}{(i+1)^2} OriginalPaper$ from $StandardCP_i$. The results show that our system is capable of providing high precision outputs. All the solutions proposed by our method are also the optimal solution in the test case. These analyses confirm if the objective model is built from a crease pattern that is a subset of the input predefined crease pattern then our technique can find out the optimal solution.

To evaluate the cases where the objective models are not built from folding actions of the input *ActionSet*, experiments E11 and E12 have been used. In experiment E11, the objective model in Figure 4.3a is a square with size $\frac{1}{4} OriginalPaper$. Obviously, we can not construct any shape that the same as the objective model from the $StandardCP_4$. In Figure 4.6, the proposed solution of our algorithm for experiment E11 is showed. Because attempting to get 0.0 in Hausdorff distance is impossible, the shapes that are most similar to the objective model are presented. Correspondingly, in experiment E12, the objective model is a shape that is constructed from the $StandardCP_9$ (Figure 4.5c). This shape is also not a rectangular or a square but an octagon. We need to find a folding sequence to build it from the $StandardCP_7$. As usual, our approach has been succeeded to find the minimum Hausdorff distance between the objective model and the feasible solutions. The shape of the optimal solution has been obtained and introduced in Figure 4.7.

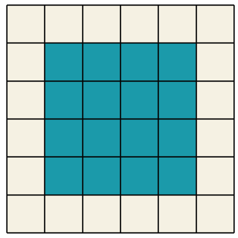
(a) Objective Model 1 (b) The output shape of program

Figure 4.6: Result of experiment E11

Furthermore, not only the objective models with a square shape but also many interesting shapes are considered. In demonstrations, E8, E9 and E10, other rectangular and square shapes are applied to evaluate our system. As expected, our experiments prove that with a suitable predefined crease pattern, we can create many useful forms. Moreover, with the support of our system, efficient ways to transform between shapes can be



(a) Objective Model 11



(b) The output shape of program

Figure 4.7: Result of experiment E12

found easily.

From the test cases, we see that the running time is increase with the number of creases. Two reasons cause it. First, the calculation time of the folding simulation is dependent on the size of the *ActionSet*. The more folding action we apply to the origami paper, the more time the program need to build the object model. Second, as we discussed in Section 3.1.3, the search space is exponential growth with the input size. Then the correlation between swarm's search area and the search space is understandable. The smaller of the rate ($swarm's\ search\ area/search\ space$) implies the harder the particles in the swarm need to perform its job in order to find out the best solution. Besides that, the numbers of the optimal solution of input shapes are different from each other. Some object models have more ways to construct it than another model. It is also a factor that affects directly to the running time in the experiments.

	Crease Pattern	Objective model	Haursdoff distance			Iteration			Avg. runtime
			Min	Max	Avg	Min	Max	Avg	(ms)
E1	$StandardCP_1$	Objective Model 1	0.0	0.0	0.0	1	1	1.0	635.12
E2	$StandardCP_2$	Objective Model 2	0.0	0.0	0.0	1	1	1.0	699.96
E3	$StandardCP_3$	Objective Model 3	0.0	0.0	0.0	1	1	1.0	719.08
E4	$StandardCP_4$	Objective Model 4	0.0	0.0	0.0	1	4	1.06	838.24
E5	$StandardCP_5$	Objective Model 5	0.0	0.0	0.0	1	7	1.68	1724.32
E6	$StandardCP_7$	Objective Model 6	0.0	0.0	0.0	1	90	18.33	35064.85
E7	$StandardCP_9$	Objective Model 7	0.0	0.0	0.0	1	44	14.58	46526.12
E8	$StandardCP_9$	Objective Model 8	0.0	0.0	0.0	1	16	5.06	12827.00
E9	$StandardCP_9$	Objective Model 9	0.0	6.0	0.72	4	100	46.62	111413.44
E10	$StandardCP_7$	Objective Model 10	0.0	3.75	0.07	1	100	20.5	33893.63
E11	$StandardCP_4$	Objective Model 1	8.5	8.5	8.5	100	100	100	78594.77
E12	$StandardCP_7$	Objective Model 11	3.3	16.37	6.0	100	100	100	164159.70

Table 4.1: Results for experiments

As seen in Figure 4.8, the particle swarm converged quickly on the optimum for objective function in experiment E9. The Hausdorff distance got the value 0.6 from the trial 17. The algorithm optimized the problem from iteration 84 and got the cost function equal 0. The proposed solution $x = (15, 16, 7, 18, 14, 7, 13, 18, 16, 7, 3, 0, 0, 17, 6, 16, 13, 17)$ is also an optimal solution.

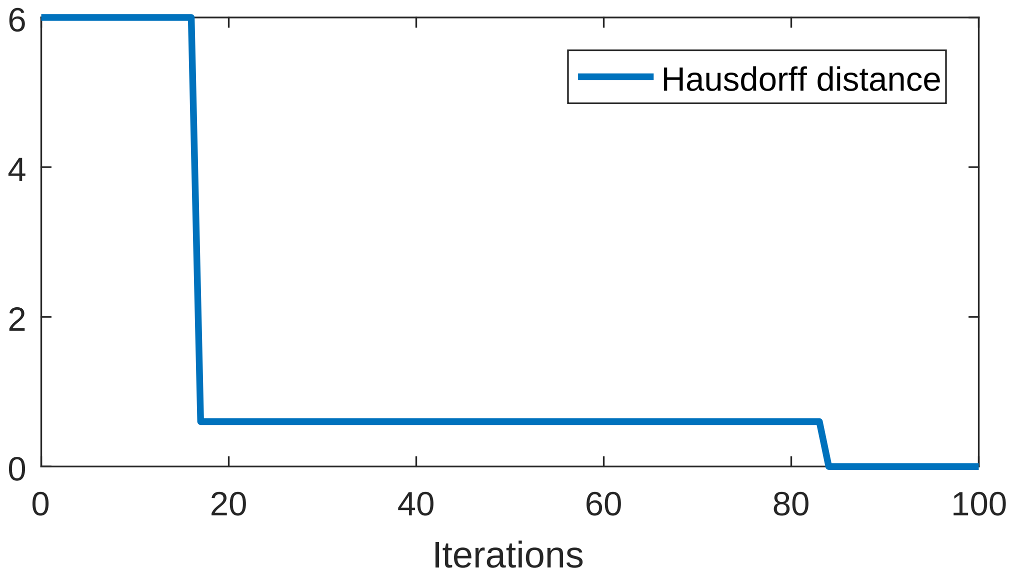


Figure 4.8: Hausdorff distance through iterations of an instance running of experiment E9

Besides advantages of our method, we aware that our research may have some limitations. The first is the folding action we can apply is too simple, only horizontal and vertical valley fold. Consequently, the feasible solutions are only the square or rectangular shapes. The objective models and the object model of candidate solutions are in 2D. The lack of knowledge in origami as well as not able to find a suitable folding simulation are the reasons why this problem happened.

The second is the maximum number the creases can be processed is restricted to 18 creases. When we apply an *ActionSet* with larger than 18 members, the consumed resources of CPU and the running times are unacceptable. This is caused by the properties of the combinatorial optimization problem and our approach, the constraints of origami, as well as the limitation of the folding simulation algorithm.

Chapter 5

Conclusion

In conclusion, we presented a new generic information-theoretical approach to solving the folding sequence generation problem. This problem was modeled as a combinatorial optimization problem. The modified DPSO algorithm is proposed as an efficient method to apply to the defined issue. An origami simulator has been adapted and developed to use in our process. From these work and information in Section 4.2, our method proved that it is an efficient technique. A high precision folding sequence can be found by using our research. We have confirmed that, with a well-prepared predefined crease pattern, many useful origami shapes can be obtained by call our method. Furthermore, this system not only discuss finding suitable crease pattern but also somehow this work revealed the contribution to the field of origami simulator and folding multiple objects model from a single paper sheet.

Our work clearly has some limitations. The most significant limitation is a result of the fact that the objective models are too simple. We are just able to handle the 2D square and rectangular origami. The folding simulation only can calculate the orthogonal predefined crease pattern is a disadvantage. The most common weakness of the combinatorial optimization problem - the large search space - still exists. A way to reduce the dimension of the search space is not proposed. Therefore, the size of the predefined crease pattern is restricted to not larger than 18 creases. Consequently, still, some aspects of the problem are not covered.

5.1 Future Work

We are currently in the process of investigating how to build or apply a better folding simulator. More study on proposed DPSO algorithm is necessary. Another approach to solve the combinatorial optimization problem will need to be undertaken. Further study of the problem would be of interest. For example, we should explore how to apply the convolutional neural network (CNN) to find the origami base of the input object and reduce the dimension of the search space. Alternatively, finding a predefined crease pattern, that can construct into useful origami models, is required. Another work to be done in the future is solving the problem with multiple origami papers, constructing into object models and combining together to form the input object.

Bibliography

[1] C. H. Yi, "Origami research and applications."

[2] S. Miyazaki, T. Yasuda, S. Yokoi, and J.-i. Toriwaki, "An origami playing simulator in the virtual space," *Journal of Visualization and Computer Animation*, vol. 7, no. 1, pp. 25–42, 1996.

[3] J. Mitani, "Development of origami pattern editor (oripa) and a method for estimating a folded configuration of origami from the crease pattern," *IPSJ Journal*, vol. 48, no. 9, pp. 3309–3317, 2007.

[4] J. Mitani, "The folded shape restoration and the rendering method of origami from the crease pattern," in *Proc. Int. Conf. on Geometry and Graphics*, pp. 1–7, 2008.

[5] T. Tachi, "Freeform origami." `www.tsg.ne.jp/TT/software/`, 2010–2016.

[6] Z. Xi, "Rigid origami folder." `http://masc.cs.gmu.edu/wiki/RigidOrigamiFolder`, 2014.

[7] E. M. Arkin, M. A. Bender, E. D. Demaine, M. L. Demaine, J. S. Mitchell, S. Sethia, and S. S. Skiena, "When can you fold a map?," in *Workshop on Algorithms and Data Structures*, pp. 401–413, Springer, 2001.

[8] H. A. Akitaya, J. Mitani, Y. Kanamori, and Y. Fukui, "Generating folding sequences from crease patterns of flat-foldable origami," in *ACM SIGGRAPH 2013 Posters*, SIGGRAPH '13, (New York, NY, USA), pp. 20:1–20:1, ACM, 2013.

[9] B. An, N. Benbernou, E. d. Demaine, and D. Rus, "Planning to fold multiple objects from a single self-folding sheet," *Robotica*, vol. 29, pp. 87–102, Jan. 2011.

[10] T. Tachi, "Generalization of rigid foldable quadrilateral mesh origami," in *Symposium of the International Association for Shell and Spatial Structures (50th. 2009. Valencia). Evolution and Trends in Design, Analysis and Construction of Shell and Spatial Structures: Proceedings*, Editorial Universitat Politècnica de València, 2009.

[11] T. Tachi, "Freeform variations of origami," *J. Geom. Graph*, vol. 14, no. 2, pp. 203–215, 2010.

[12] T. Tachi, "Freeform rigid-foldable structure using bidirectionally flat-foldable planar quadrilateral mesh," *Advances in architectural geometry 2010*, pp. 87–102, 2010.

[13] E. Demaine, "Erik demaine's folding and unfolding page."

[14] D. J. Balkcom and M. T. Mason, "Robotic origami folding," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 613–627, 2008.

[15] E. Hawkes, B. An, N. Benbernou, H. Tanaka, S. Kim, E. Demaine, D. Rus, and R. Wood, "Programmable matter by folding," *Proceedings of the National Academy of Sciences*, vol. 107, no. 28, pp. 12441–12445, 2010.

[16] S. Felton, M. Tolley, E. Demaine, D. Rus, and R. Wood, "A method for building self-folding machines," *Science*, vol. 345, no. 6197, pp. 644–646, 2014.

[17] S. Miyashita, S. Guitron, M. Ludersdorfer, C. R. Sung, and D. Rus, "An untethered miniature origami robot that self-folds, walks, swims, and degrades," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 1490–1496, IEEE, 2015.

[18] H. Koshiro, "History of origami." http://origami.ousaan.com/library/historye.html.

[19] F. Margalit, "Akira yoshizawa, 94, modern origami master," *The New York Times*, Apr 2005.

[20] L. Robert J, *Origami Design Secrets*. Dover Publications, 2003.

[21] Wikipedia, "Yoshizawa–Randlett system — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Yoshizawa%E2%80%93Randlett%20system&oldid=730827960, 2017. [Online; accessed 08-February-2017].

[22] CREASED, "Crease pattern corner." http://creased.com/corners/CreasePatternCorner/creasepatternCorner.html. [Online; accessed 08-February-2017].

[23] B. Y. University, "Origami in space: Byu-designed solar arrays inspired by origami." https://www.youtube.com/watch?v=3E12uju1vgQ, 2013.

[24] Z. You and K. Kuribayashi.

[25] A. Schrijver, "A course in combinatorial optimization," 2006.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, pp. 1942–1948 vol.4, Nov 1995.

[27] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pp. 39–43, Oct 1995.

[28] R. Eberhart, P. Simpson, and R. Dobbins, *Computational Intelligence PC Tools*. San Diego, CA, USA: Academic Press Professional, Inc., 1996.

[29] J. Kennedy, "The particle swarm: social adaptation of knowledge," in *Evolutionary Computation, 1997., IEEE International Conference on*, pp. 303–308, Apr 1997.

[30] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 69–73, May 1998.

[31] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[32] R. Poli, "Analysis of the publications on the applications of particle swarm optimisation," *J. Artif. Evol. App.*, vol. 2008, pp. 4:1–4:10, Jan. 2008.

[33] M. R. Bonyadi and Z. Michalewicz, "Particle swarm optimization for single objective continuous space problems: a review," *Evolutionary computation*, 2016.

[34] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, vol. 1, pp. 101–106 vol. 1, 2001.

[35] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58–73, Feb 2002.

[36] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 225–239, June 2004.

[37] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 281–295, June 2006.

[38] J. Kennedy, *Particle Swarm Optimization*, pp. 760–766. Boston, MA: Springer US, 2010.

[39] M. Clerc, "Standard Particle Swarm Optimisation." 15 pages, Sept. 2012.

[40] M. Taherkhani and R. Safabakhsh, "A novel stability-based adaptive inertia weight for particle swarm optimization," *Appl. Soft Comput.*, vol. 38, pp. 281–295, Jan. 2016.

[41] Y. Shi and R. C. Eberhart, *Parameter selection in particle swarm optimization*, pp. 591–600. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.

[42] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, pp. 84–88 vol.1, 2000.

[43] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317 – 325, 2003.

[44] M. Meissner, M. Schmuker, and G. Schneider, "Optimized particle swarm optimization (opso) and its application to artificial neural network training," *BMC Bioinformatics*, vol. 7, no. 1, p. 125, 2006.

[45] M. E. H. Pedersen, *Tuning & simplifying heuristical optimization.* PhD thesis, University of Southampton, 2010.

[46] M. Pedersen and A. Chipperfield, "Simplifying particle swarm optimization," *Applied Soft Computing*, vol. 10, no. 2, pp. 618 – 628, 2010.

[47] M. S. Nobile, G. Pasi, P. Cazzaniga, D. Besozzi, R. Colombo, and G. Mauri, "Proactive particles in swarm optimization: A self-tuning algorithm based on fuzzy logic," in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8, Aug 2015.

[48] M. Clerc, *Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem*, pp. 219–239. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[49] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5, pp. 4104–4108 vol.5, Oct 1997.

[50] B. Al-kazemi and C. Mohan, *Discrete Multi-Phase Particle Swarm Optimization*, pp. 305–327. London: Springer London, 2005.

[51] G. Pampara, N. Franken, and A. P. Engelbrecht, "Combining particle swarm optimisation with angle modulation to solve binary problems," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1, pp. 89–96 Vol.1, Sept 2005.

[52] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Machine Learning and Cybernetics, 2003 International Conference on*, vol. 3, pp. 1583–1585, IEEE, 2003.

[53] W. N. Chen, J. Zhang, H. S. H. Chung, W. L. Zhong, W. G. Wu, and Y. h. Shi, "A novel set-based particle swarm optimization method for discrete optimization

problems," *IEEE Transactions on Evolutionary Computation*, vol. 14, pp. 278–300, April 2010.

[54] J. Krause, J. Cordeiro, R. S. Parpinelli, and H. S. Lopes, "A survey of swarm algorithms applied to discrete optimization problems," *Swarm Intelligence and Bio-Inspired Computation*, vol. 4, no. 9, pp. 169–191, 2013.

[55] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[56] M. E. Kurz and R. G. Askin, "Scheduling flexible flow lines with sequence-dependent setup times," *European Journal of Operational Research*, vol. 159, no. 1, pp. 66 – 82, 2004.

[57] M. F. Tasgetiren, M. Sevkli, Y.-C. Liang, and G. Gencyilmaz, "Particle swarm optimization algorithm for single machine total weighted tardiness problem," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 2, pp. 1412–1419 Vol.2, June 2004.

[58] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem," *Computers & Operations Research*, vol. 35, no. 9, pp. 2807 – 2839, 2008. Part Special Issue: Bio-inspired Methods in Combinatorial Optimization.

[59] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1930 – 1947, 2007.

[60] L. Congying, Z. Huanping, and Y. Xinfeng, "Particle swarm optimization algorithm for quadratic assignment problem," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, vol. 3, pp. 1728–1731, Dec 2011.

[61] S. Burnwal and S. Deb, "Scheduling optimization of flexible manufacturing system using cuckoo search-based approach," *The International Journal of Advanced Manufacturing Technology*, vol. 64, no. 5, pp. 951–959, 2013.

[62] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[63] M. M. Deza and E. Deza, "Encyclopedia of distances," in *Encyclopedia of Distances*, pp. 1–583, Springer, 2009.

[64] R. T. Rockafellar and R. J.-B. Wets, *Variational analysis*, vol. 317. Springer Science & Business Media, 2009.