Generating Goal Configurations for Scalable Shape Formation in Robotic Swarms

Hanlin Wang and Michael Rubenstein

Northwestern University, Evanston, IL, 60201, USA
h.w@u.northwestern.edu
rubenstein@northwestern.edu

Abstract. In this paper, we present an algorithm that automatically encodes a user-defined complex 2D shape to a set of cells on a grid each characterizing a robot currently in the swarm. The algorithm is validated via up to 100 simulated robots as well as up to 100 physical robots. The results show that the goal configurations generated by the algorithm for the swarms with any size are consistent with the input shapes, moreover, it allows the swarm to adapt to the swarm size change quickly and robustly. The supplementary materials for this paper can be found at: https://tinyurl.com/2huc42t6

Keywords: Swarm systems, Shape Formation

1 Introduction

Shape formation is an important and well studied problem in the robotic swarm systems. Here, the task is to move a group of robots to form a user-defined shape. In the past, the problem has received a lot of attentions due to its extensive real-world applications such as automated warehouse [1], entertainment applications [2], and more [3].

Many past efforts have concentrated on a vanilla version of the problem. In the vanilla shape formation problem, the swarm size is assumed to stay the same all the times, and the representation of the desired shape is often pre-computed and given to the swarm as an input. Many methods to represent the desired shapes has been presented in the past, including curves or regions explicitly described by a mathematical formula [4, 5], potential fields [6], masked grid [7, 8, 9], and more [10, 11]. The mathematical formula-based representations [4, 5, 12, 6] can help to derive the formation control laws when the robot's kinematic or dynamic constraints need to be considered. However, when the desired shape is complex, it is time-consuming (sometimes even impossible) to encode the desired shape to a mathematical formula. Masked grid, also known as "binary image" in 2D case [7] or "binary volumes" in 3D case [8], is a grid where each cell is labeled with either a 1 or a 0, indicating whether the cell is in the shape or not, and the desired shape is described as the set of in-shape cells on the grid [7, 9, 8]. Masked grid is a convenient way to encode the complex shape, in addition,

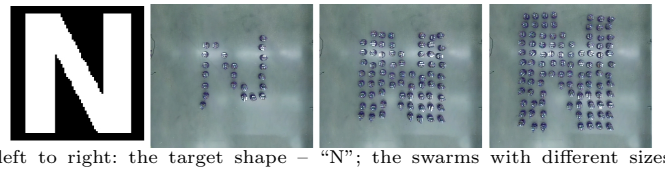

Fig. 1: From left to right: the target shape – "N"; the swarms with different sizes forming the configurations generated by the proposed algorithm

for the swarm of modular robots that are with discrete attachment locations [13, 14], the masked grid is a natural way to describe the collective's configuration.

Beyond the vanilla shape formation, the other version of the problem is so called *scalable* (or *scale-independent*) shape formation [7, 8]. In the *scalable* shape formation problem, robots can be removed from / added to the swarm in real-time. When the removal or addition of the robots occurs, there are two strategies for the swarm to adapt. One option is to keep the scale of the desired shape fixed, and change the density of the robots [15]. One drawback of this method is that: when the robot's physical size is finite, the size of swarm to display the shape will be limited, as one can fit only finite amount of robots in a unit of space. On the contrary, the other option is to keep the density of the robot fixed and change the size of the desired shape [7, 8]. When using the masked grid to describe the target shape, there are two options to scale the goal configuration: change the number of robots fitted in each cell and fix the number of in-shape cells [7], or, change the number of in-shape cells and fit exactly one robot to each cell [8]. As shown in [7, 8], both of these two methods can offer the swarm the capability of self-healing, making the system resilient to the removal and addition of robots. On the other hand, for the algorithm proposed in [7], when the swarm size changes, it takes the swarm a long time to adapt, as the swarm needs to wait "long enough" to sense the change of the swarm size. Moreover, the algorithm presented in the [8] only works for certain types of shapes, and the generated configurations can be perfectly formed only by the swarms with certain sizes.

In this paper, we present an algorithm that automatically encodes an input 2D shape (given by an binary image) to a masked square grid where each in-shape cell characterizes a robot current in the swarm. Given an input shape and the swarm size n, the algorithm will first use naive binary image scaling methods to generate two reference grids, in which one has slightly more than n in-shape cells and the other has slightly less than n in-shape cells, then use a second subroutine, called *interpolation*, to refine those two reference grids so as to obtain the final output – a masked grid with exactly n in-shape cells. The algorithm is validated via both the simulated and physical experiments, the results show that the goal configurations generated by the algorithm are consistent with the original input shapes, moreover, when the swarm size changes, it allows the swarm to adapt quickly and robustly.

2 Preliminaries

In this section, we will formally state the problem, and introduce the notations frequently used in the rest of the paper.

2.1 Goal Configuration Generation: Problem Statement

The proposed algorithm takes two inputs: an binary image describing the desired shape, and the size of the swarm to display the desired shape. Note that a binary image is essentially a 2D masked grid, therefore, for the sake of description, in the rest of the paper, we use the word "pixel" and the word "cell" interchangeably. The output of the algorithm is a masked grid such that: the number of in-shape cells must equal to the input swarm size.

When designing the algorithm, there are two factors to be considered: first, the generated goal configurations should be consistent with the input shape; second, in order to allow the swarm to quickly adapt to the removal and addition of the robots, the goal configurations generated for different swarm sizes should be similar to each other as well.

2.2 Notations

Let $\mathcal{G}_i$ be a 2D $m \times m$ masked square grid i, we use $c_i^{(x,y)} \in \{0, 1\}$ to denote the label of the cell in the x-th row and y-th column from the left-top corner, and we use $\mathcal{S}(\mathcal{G}_i) = \{(x, y) \mid c_i^{(x,y)} = 1\}$ to denote the set of the coordinates of all in-shape cells on $\mathcal{G}_i$. Given a set $\mathcal{A}$, we use $|\mathcal{A}|$ to denote its cardinality. For a pair of sets $\mathcal{A}$ and $\mathcal{B}$: $\mathcal{A} \cup \mathcal{B}$ denotes the union of set $\mathcal{A}$ and set $\mathcal{B}$; $\mathcal{A} \cap \mathcal{B}$ denotes the intersection of set $\mathcal{A}$ and set $\mathcal{B}$; $\mathcal{A} - \mathcal{B}$ denotes the set of all the elements that are in set $\mathcal{A}$ but not in set $\mathcal{B}$. For $n \geq 3$ sets $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{n-1}$, their union is denoted as $\bigcup_{i=0}^{n-1} \mathcal{A}_i$.

3 Approach

We assume the desired shape is given to the swarm in the format of a 100×100 binary image, however this approach can be generalized to any size binary image. The proposed algorithm consists of two subroutines – *scaling* and *interpolation*. Given the desired shape $\mathcal{G}_{in}$ and the swarm size n, the algorithm will first use the *scaling* subroutine to find two reference masked grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ such that: $\mathcal{G}_o^h$ has slightly more in-shapes than n and $\mathcal{G}_o^l$ has slightly less in-shapes than n, then apply the *interpolation* subroutine to $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ so as to obtain an output that is with exactly n in-shape cells. A graphical illustration of the overall pipeline is shown in Fig. 2 and a detailed description of algorithm's overall pipeline is shown in Alg. 1.

Algorithm 1: Pipeline for proposed algorithm

Input: Input shape $\mathcal{G}_{in}$, swarm size n
Output: Configuration for the swarm $\mathcal{G}_o$

$\mathcal{G}_o^l, \mathcal{G}_o^h \leftarrow scaling(\mathcal{G}_{in}, n)$
if $|\mathcal{S}(\mathcal{G}_o^l)|$ *is* n **then**
  $\mathcal{G}_o \leftarrow \mathcal{G}_o^l$
else
  $\mathcal{G}_o \leftarrow interpolation(\mathcal{G}_o^l, \mathcal{G}_o^h, n)$
return $\mathcal{G}_o$

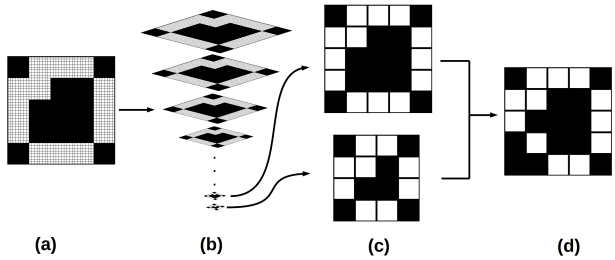

Fig. 2: The graphical illustration of the the overall pipeline of the presented algorithm. From left to right: (a) The input shape, which is given in the format of a binary image. In this example, the task is to find a goal configuration for a swarm of 12 robots; (b) We apply *scaling* subroutine to the input masked grid so as to find two reference grids with approximately 12 in-shape cells; (c) Two reference masked grids with each pixel enlarged for the visualization purpose; (d) A configuration with 12 in-shape cells is constructed by the *interpolation* subroutine using those 2 reference grids in (c).

Algorithm 2: *scaling* subroutine

Input: Input shape $\mathcal{G}_{in}$, swarm size n
Output: Two reference masked grids $\mathcal{G}_o^l$, $\mathcal{G}_o^h$

1 $m \leftarrow 100$ // inialize m to the size of $\mathcal{G}_{in}$
2 $iter \leftarrow 1$
3 $last \leftarrow \mathcal{G}_{in}$ // variable to store the last scaled grid
4 **if** $|\mathcal{S}(\mathcal{G}_{in})|$ *is* n **then**
5 $\mathcal{G}_o^l \leftarrow \mathcal{G}_{in}$, $\mathcal{G}_o^h \leftarrow \mathcal{G}_{in}$
6 **if** $|\mathcal{S}(\mathcal{G}_{in})| < n$ **then**
7 **while** *1* **do**
8 $m \leftarrow m + 1$ // gradually increase scaled grid size
9 $cur \leftarrow$ scale the grid $\mathcal{G}_{in}$ to size $m \times m$
10 **if** $|\mathcal{S}(cur)| \geq n$ **then**
11 $\mathcal{G}_o^l \leftarrow last$, $\mathcal{G}_o^h \leftarrow cur$
12 **break**
13 **else**
14 $last \leftarrow cur$
15 $iter \leftarrow iter + 1$
16 **if** $|\mathcal{S}(\mathcal{G}_{in})| > n$ **then**
17 **while** *1* **do**
18 $m \leftarrow m - 1$ // gradually decrease scaled grid size
19 **if** $m < 15$ **then** // switch to skeletonization
20 $cur \leftarrow$ skeletonize the grid $last$
21 **if** $|\mathcal{S}(cur)| \leq n$ **then**
22 $\mathcal{G}_o^l \leftarrow cur$, $\mathcal{G}_o^h \leftarrow last$
23 **break**
24 **else**
25 Error: the input n is too small.
26 $cur \leftarrow$ scale the grid $\mathcal{G}_{in}$ to size $m \times m$
27 **if** $|\mathcal{S}(cur)| \leq n$ **then**
28 $\mathcal{G}_o^l \leftarrow cur$, $\mathcal{G}_o^h \leftarrow last$
29 **break**
30 **else**
31 $last \leftarrow cur$
32 $iter \leftarrow iter + 1$
33 **return** $\mathcal{G}_o^l$, $\mathcal{G}_o^h$

3.1 Scaling

In the *scaling* subroutine, we first use the *image scaling* to change the number of in-shape pixels. Image scaling is a well studied topic [16, 17], here, the task is to create a new version of the image with a different width and/or height in pixels. Many strategies to scale a binary image have been proposed in the past, in this paper, we use the *nearest neighbor interpolation* [16] as our *image scaling* method.

Given an input shape $\mathcal{G}_{in}$ and swarm size n, there are three possible cases: If the number of in-shape cells on the input grid $|\mathcal{S}(\mathcal{G}_{in})|$ equals to n, the algorithm will return $\mathcal{G}_{in}$ directly (Alg. 2, Line 4-5). If the $|\mathcal{S}(\mathcal{G}_{in})| < n$, the algorithm will first keep upscaling the $\mathcal{G}_{in}$ (Alg. 2, Line 6-15) until a grid that has more than n in-shape cells is found (Alg. 2, Line 10 - 12), then return the two scaled images obtained most lately, and exit this subroutine (Alg. 2, Line 11-12). Similarly, if $|\mathcal{S}(\mathcal{G}_{in})| > n$, the algorithm will keep downscaling the $\mathcal{G}_{in}$ until finding a grid that contains less than n in-shape cells (Alg. 2, Line 16-32).

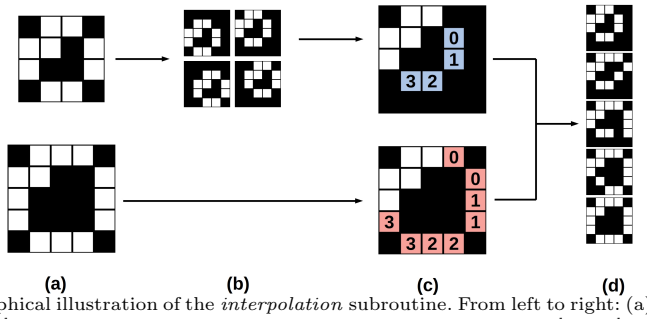

Fig. 3: The graphical illustration of the *interpolation* subroutine. From left to right: (a) Two reference grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ with 9 and 13 in-shape cells, respectively; (b) We align $\mathcal{G}_o^l$ to $\mathcal{G}_o^h$ (Alg. 3, Line 1). There are 4 possible locations to place $\mathcal{G}_o^l$ on a 5×5 gird, and we choose the one in the right-up corner because the *difference score* between this grid and $\mathcal{G}_o^h$ is the lowest; (c) We calculate the set $\mathcal{D}_{l-h}$, which is the set of cells filled with blue color, and the set $\mathcal{D}_{h-l}$, which is the set of cells filled with red color, and then split these two sets into two sets of 4 subsets $\{d_{l-h}^0, \ldots, d_{l-h}^3\}$ and $\{d_{h-l}^0, \ldots, d_{h-l}^3\}$, the number on each cells indicates the subset that it belongs to (Alg. 3, Line 2-24); (d) 5 configurations generated using $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ with different input swarm size n s. From top to bottom: the configuration generated for the swarm with a size of $9, 10, 11, 12, 13$, respectively (Alg. 3, Line 25-27).

One issue for using the *image scaling* to reduce the number of in-shape cells is that: When the size of the scaled image is too small ($\leq 15 \times 15$ according to our experiments), it often fails to preserve the main structure of the input shape. Therefore, to prevent the main structure of the shape in scaled image being distorted by over-downsampling, the size of the scaled image cannot be smaller than a threshold (Alg. 2, Line 20). This limits the minimal number of in-shape cells in the outputs that can be generated.

Besides the *image scaling*, an alternative to reduce the pixels required to display a shape is the operation *skeletonization* [18]. The operation *skeletonization* generates a "thinner" version of the input shape that emphasizes shape's geometrical and topological properties. In the *scaling* subroutine, we use the operation *skeletonization* to extend the range of swarm sizes for which our method can work: if the image scaled with the minimal size still has more than n in-shape cells, we then apply the operation *skeletonization* to this scaled image so as to obtain the shape's skeleton, which is a masked grid with fewer in-shape cells (Alg. 2, Line 20). It is possible that the number of in-shape cell on the skeleton is still more than n, if that happens, algorithm will raise an error to tell the user that the input swarm size n is too small for displaying the desired shape $\mathcal{G}_{in}$ (Alg. 2, Line 25). See Alg. 2 for the detailed pseudo code for the *scaling* subroutine.

3.2 Interpolation

The high-level idea behind the *interpolation* subroutine can be described as follows: Say we have a $l \times l$ binary image $\mathcal{G}_o^l$ and a $h \times h$ binary image $\mathcal{G}_o^h$ with a and b amount of in-shape cells, respectively. Assume $h > l$ and $b > a$, we want to generate a sequence of $h \times h$ binary images $\mathcal{G}_a, \ldots, \mathcal{G}_b$, in which each generated image $\mathcal{G}_i$ has exactly i amount of in-shape cells. To do so, we first place the input grid $\mathcal{G}_o^l$ on a empty $h \times h$ grid at a location such that the overlapping between the newly formed binary image and the $\mathcal{G}_o^h$ is maximized.

Then, we calculate the difference between the newly formed $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$, which can be characterized by the cells that are with different labels on those two grids. The sequence of the binary images $\mathcal{G}_a, \ldots, \mathcal{G}_b$ can be constructed by gradually toggling the labels of those difference cells on the grid $\mathcal{G}_o^l$. To be more specific, for the aligned $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$, their difference can be characterized by two sets: $\mathcal{D}_{l-h} = \mathcal{S}(\mathcal{G}_o^l) - \mathcal{S}(\mathcal{G}_o^h)$, which is the set of cells that are in the shape on $\mathcal{G}_o^l$ but off the shape on $\mathcal{G}_o^h$, and $\mathcal{D}_{h-l} = \mathcal{S}(\mathcal{G}_o^h) - \mathcal{S}(\mathcal{G}_o^l)$, which is the set of cells that are in the shape on $\mathcal{G}_o^h$ but off the shape on $\mathcal{G}_o^l$. (Alg. 3, Line 3-4). Next, let $k = |\mathcal{S}(\mathcal{G}_o^h)| - |\mathcal{S}(\mathcal{G}_o^l)|$ be the difference between the numbers of in-shape cells on $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ (Alg. 3, Line 2), we split those two sets into two groups of k disjointed subsets $d_{l-h}^0, \ldots, d_{l-h}^{k-1}$ and $d_{h-l}^0, \ldots, d_{h-l}^{k-1}$ such that: $\bigcup_{i=0}^{k-1} d_{l-h}^i = \mathcal{D}_{l-h}$ and $\bigcup_{i=0}^{k-1} d_{h-l}^i = \mathcal{D}_{h-l}$. In addition, for a pair of subsets d_{h-l}^i and d_{l-h}^i, we enforce their sizes to be such that: $|d_{h-l}^i| = 1 + |d_{l-h}^i|$ (Alg. 3, Line 5-24). With this constraint on each subset's cardinality, given an swarm size n, its corresponding configuration can be constructed as follows: first, set the labels the cells $\bigcup_{i=0}^{n-|\mathcal{S}(\mathcal{G}_o^l)|} d_{l-h}^i$ to 0 on $\mathcal{G}_o^l$, then, set the labels of the cells $\bigcup_{i=0}^{n-|\mathcal{S}(\mathcal{G}_o^l)|} d_{h-l}^i$ to 1 on $\mathcal{G}_o^l$ (Alg. 3, Line 25-27). It is straight forward to examine that the masked grid constructed via the procedure above will have exactly n in-shape cells. The pseudo code for the *interpolation* subroutine is shown in Alg. 3. A graphical illustration of the *interpolation* subroutine is shown in Fig. 3.

Assume that the reference grid $\mathcal{G}_o^l$ has a size of $l \times l$ and $\mathcal{G}_o^h$ has a size of $h \times h$, where $h \geq l$ according to Alg. 2. The *interpolation* subroutine will first draw the $\mathcal{G}_o^l$ on an empty $h \times h$ grid (Alg. 3, Line 1). Before drawing $\mathcal{G}_o^l$ on this empty $h \times h$ grid, we need to determine the location to place the $\mathcal{G}_o^l$ on this $h \times h$ grid, as there might be multiple choices since $h \geq l$. To do so, we first define a metric called *difference score* as follows:

Definition 1. *Given two masked grids $\mathcal{A}$ and $\mathcal{B}$, the difference score between $\mathcal{A}$ and $\mathcal{B}$ is given by $|\mathcal{S}(\mathcal{A}) - \mathcal{S}(\mathcal{B})| + |\mathcal{S}(\mathcal{B}) - \mathcal{S}(\mathcal{A})|$, i.e, the number of cells that are in shape on $\mathcal{A}$ but not in shape on $\mathcal{B}$ plus the number of cells that are on in shape on $\mathcal{B}$ but not in shape on $\mathcal{A}$.*

With this metric, the location to place the grid $\mathcal{G}_o^l$ on the new $h \times h$ grid can be determined as follows: we exhaustively search all possible translations, and choose the translation that gives the minimal *difference score* between the grid $\mathcal{G}_o^h$ and the newly generated $\mathcal{G}_o^l$ (Alg. 3, Line 1).

After aligning $\mathcal{G}_o^l$ to $\mathcal{G}_o^h$, we first calculate the difference between $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ (Alg. 3, Line 2-4), then pack the set $\mathcal{D}_{l-h}$ into k subsets (Alg. 3, Line 5-16). sz^i denotes the size of the subset d_{l-h}^i of $\mathcal{D}_{l-h}^i$, we first calculate the size of each subset d_{l-h}^i (Alg. 3, Line 5-10). After determining the size of each subset d_{l-h}^i, we then start to determine the contents of each subset d_{l-h}^i. When removing the cells from the shape, we want to remove the cells following the order such that: the cells closer to shape's boundary will be removed first. This order helps to avoid generating "holes" in the remaining shape. To address this design consideration, in the algorithm, for all the cells in $\mathcal{D}_{l-h}$, we first calculate each cell's Manhattan distance to the boundary using the operation *distance transform* [19] (Alg. 3,

Algorithm 3: *interpolation* subroutine

Input: Reference grids $\mathcal{G}_o^l, \mathcal{G}_o^h$, swarm size n
Output: The generated masked grid $\mathcal{G}_o$

1 $\mathcal{G}_o^l \leftarrow$ align $\mathcal{G}_o^l$ to $\mathcal{G}_o^h$
2 $k \leftarrow |\mathcal{S}(\mathcal{G}_o^h)| - |\mathcal{S}(\mathcal{G}_o^l)|$ // calculate the difference of in-shape numbers on two reference grid
3 $\mathcal{D}_{l-h} \leftarrow \mathcal{S}(\mathcal{G}_o^l) - \mathcal{S}(\mathcal{G}_o^h)$ // the set of cells that are in the shape on $\mathcal{G}_o^l$ but off the shape on $\mathcal{G}_o^h$
4 $\mathcal{D}_{h-l} \leftarrow \mathcal{S}(\mathcal{G}_o^h) - \mathcal{S}(\mathcal{G}_o^l)$ // the set of cells that are in the shape on $\mathcal{G}_o^h$ but off the shape on $\mathcal{G}_o^l$
5 Initialize $sz^0, \ldots, sz^{k-1}$ to be all 0s // the size of each subset used in interpolation
6 **for** $i \leftarrow 0, \ldots, k-1$ **do** // determine each subset's size
7 **if** $i \leq |\mathcal{D}_{l-h}| \ mod \ k$ **then**
8 $sz^i \leftarrow \lfloor \frac{|\mathcal{D}_{l-h}|}{k} \rfloor + 1$
9 **else**
10 $sz^i \leftarrow \lfloor \frac{|\mathcal{D}_{l-h}|}{k} \rfloor$
11 $dt(\mathcal{G}_o^l) \leftarrow$ apply *distance transform* to $\mathcal{G}_o^l$ // calculate each in-shape cell's distance to the boundary
12 *buf_sub*$\leftarrow$use each cell's value in $dt(\mathcal{G}_o^l)$ as the key to sort $\mathcal{S}(\mathcal{G}_o^l)$ in ascending order
13 Initialize $d_{l-h}^0, \ldots, d_{l-h}^{k-1}$ to be all $\varnothing$s // the subsets of cells to be turned off on $\mathcal{G}_o^l$
14 **for** $i \leftarrow 0, \ldots, k-1$ **do** // assemble the subsets of cells to be turned off on $\mathcal{G}_o^l$
15 $d_{l-h}^i \leftarrow d_{l-h}^i \cup \{$the first sz^i cells in *buf_sub*$\}$
16 *buf_sub*$\leftarrow$*buf_sub* - d_{l-h}^i
17 Initialize $d_{h-l}^0, \ldots, d_{h-l}^{k-1}$ to be all $\varnothing$s // the subsets of cells to be turned on on $\mathcal{G}_o^l$
18 *buf_add*$\leftarrow \mathcal{D}_{h-l}$
19 **for** $i \leftarrow 0, \ldots, k-1$ **do**
20 $d_{h-l}^i \leftarrow d_{h-l}^i \cup \{sz^i$ amount of cells in *buf_add* that are closest to the cells in $d_{l-h}^i\}$
21 *buf_add*$\leftarrow$*buf_add* - d_{h-l}^i
22 **for** $i \leftarrow 0, \ldots, k-1$ **do**
23 $d_{h-l}^i \leftarrow d_{h-l}^i \cup \{$the first cell in *buf_add*$\}$
24 *buf_add*$\leftarrow$*buf_add* - d_{h-l}^i
25 $\mathcal{G}_o \leftarrow$a copy of $\mathcal{G}_o^l$ // make a copy of $\mathcal{G}_o^l$ and toggle the labels of cells on it so as to construct the output
26 set labels of all the cells in $\bigcup_{i=0}^{n-|\mathcal{G}_o^l|} d_{l-h}^i$ to 0 on $\mathcal{G}_o$
27 set labels of all the cells in $\bigcup_{i=0}^{n-|\mathcal{G}_o^l|} d_{h-l}^i$ to 1 on $\mathcal{G}_o$
28 **return** $\mathcal{G}_o$

Line 11), and then use each cell's distance to boundary as the key to sort all the cells in $\mathcal{D}_{l-h}$ in ascending order (Alg. 3, Line 12). The sorted $\mathcal{D}_{l-h}$ is stored in the variable *buf_sub*. Next, we start to assemble each subset d_{l-h}^i according to the determined pack size sz^i (Alg. 3, Line 14-16): for each subset d_{l-h}^i, we pack the first sz^i cells in *buf_sub* into it (Alg. 3, Line 15), and then delete those sz^i cells from the *buf_sub* right after so as to avoid the case where the same cell shows up in two difference subsets (Alg. 3, Line 16).

Next, we start to assemble the subsets $d_{h-l}^0, \ldots, d_{h-l}^{k-1}$ (Alg. 3, Line 18 - 24). We first make a copy of $\mathcal{D}_{h-l}$ and store it to variable *buf_add* (Alg. 3, Line 18). Note that when removing a subset d_{l-h}^i of cells from the shape, we will damage the structure of the shape. To reduce the effect of the removal of d_{l-h}^i, when packing each subset d_{h-l}^i, which are the sets to be added to the remaining shape, we want the cells in d_{h-l}^i to be as "close" to cells in d_{l-h}^i as possible (Alg. 3, Line 20). To be more specific, given a subset d_{l-h}^i and the set *buf_add*, we treat each cell in d_{l-h}^i as a "job" and each cell currently in *buf_add* as a "worker", and the cost for each "worker" doing each "job" is given by the Manhattan distance between those two cells. We use the Hungarian algorithm [20] to assign exactly 1 "worker" to each "job" in each subset d_{h-l}^i such that the total cost is minimized,

and these assigned "workers" will be packed into d^i_{h-l} (Alg. 3, Line 20). Recall that as stated in the overall description of the *interpolation* subroutine, we have a constraint on the each pair of subsets' cardinalities that: $|d^i_{h-l}| = 1+|d^i_{l-h}|$. On the other hand, it is straight forward to examine that after Alg. 3, Line 19-21, each pair of subsets d^i_{h-l} and d^i_{l-h} have the same cardinality. In order to satisfy the cardinality constraint above, in Alg. 3, Line 22-24, we add one extra cell to each of those subsets $d^0_{h-l}, \ldots, d^{k-1}_{h-l}$.

So far, both the subsets $d^0_{l-h}, \ldots, d^{k-1}_{l-h}$ and the subsets $d^0_{h-l}, \ldots, d^{k-1}_{h-l}$ have been assembled, we then construct the desired masked grid $\mathcal{G}_o$ following the procedure described at the beginning of the Section 3.2 (Alg. 3, Line 25-27). See Fig. 3 for a graphical illustration of the *interpolation* subroutine.

In *interpolation* subroutine, one key element is the way to determine sz^i, which is the size of each subset d^i_{h-l} (Alg. 3, Line 6-10). Given two masked grids $\mathcal{G}^l_o$ and $\mathcal{G}^h_o$, there might be multiple feasible combinations of each subset's size. One can consider a case where $|\mathcal{D}_{l-h}| = 2, |\mathcal{D}_{h-l}| = 4, k = 2$, one way to split $\mathcal{D}_{l-h}$ and $\mathcal{D}_{h-l}$ is: $|d^0_{l-h}| = 1, |d^1_{l-h}| = 1, |d^0_{h-l}| = 2, |d^1_{h-l}| = 2$, and the other way is: $|d^0_{l-h}| = 2, |d^1_{l-h}| = 0, |d^0_{h-l}| = 3, |d^1_{h-l}| = 1$. According to the overall pipeline of the algorithm (Alg. 1), for the same pair of reference grids $\mathcal{G}^l_o$ and $\mathcal{G}^h_o$, they could be used to construct $|\mathcal{S}(\mathcal{G}^h_o)| - |\mathcal{S}(\mathcal{G}^l_o)| + 1$ different configurations with $n = |\mathcal{S}(\mathcal{G}^l_o)|, |\mathcal{S}(\mathcal{G}^l_o)| + 1, \ldots, |\mathcal{S}(\mathcal{G}^h_o)|$ amount of in-shape cells, respectively. It is trivial to see that different ways to determine each subset's size will result in different *difference score*s among these generated masked grids. Recall that as stated in the Section 2.1, to allow the swarm to quickly adapt to the swarm size change, one of our design considerations is: the configurations generated for different swarm sizes should be similar to each other. Responding to this design consideration, given a pair of reference grids $\mathcal{G}^l_o$ and $\mathcal{G}^h_o$, for the configurations generated from them, a desirable way to determine each subset's size should make the *difference score* between any pair of masked grids with adjacent number of in-shape cells as small as possible. In the following, we show that: given two reference grids $\mathcal{G}^l_o$ and $\mathcal{G}^h_o$, the way that we determine each subset's size (Alg. 3, Line 6-10) is actually the optimal way that can minimize the maximal *difference score* between any pair of generated masked grids whose difference of in-shape cell number is one.

Problem 1 *(Fair packing) Given a set $\mathcal{A}$ and an integer k, $\mathcal{P}_k(\mathcal{A})$ denotes a k-partition of the set $\mathcal{A}$, which is a set of k subsets $\{a_0, \ldots, a_{k-1}\}$ such that: (i) $\bigcup_{i=0}^{k-1} a_i = \mathcal{A}$, and (ii) $\forall i \neq j, a_i \cap a_j = \varnothing$. The task is to find a $\mathcal{P}_k(\mathcal{A})$ that minimizes the maximal cardinality among all those k subsets $a_i \in \mathcal{P}_k(\mathcal{A})$. That is, given a set $\mathcal{A}$ and an integer k, find a partition $\mathcal{P}^*_k(\mathcal{A})$ such that:*

$$\mathcal{P}^*_k(\mathcal{A}) = \mathit{argmin} \max_{a_i \in \mathcal{P}^*_k(\mathcal{A})} |a_i|$$

To interpret Problem 1, one can consider a simple instance of it: Say we have 10 balls and we are tasked to put those 10 balls into 3 bins. We want to find a way to assign those 10 balls to those 3 bins such that the maximal number of balls among all 3 bins is minimized. Next, in the Lemma 1, we shows an sufficient condition for a solution to be optimal to Problem 1.

Lemma 1. *Given a set $\mathcal{A}$ and an integer k, let $\mathcal{P}_k^{'}(\mathcal{A})$ be a k-partition of the set $\mathcal{A}$, if the $\mathcal{P}_k^{'}(\mathcal{A})$ is made such that:*

$$\max_{a_i \in \mathcal{P}_k^{'}(\mathcal{A})} |a_i| - \min_{a_i \in \mathcal{P}_k^{'}(\mathcal{A})} |a_i| \leq 1 \tag{1}$$

Then $\mathcal{P}_k^{'}(\mathcal{A})$ is an optimal solution to Problem 1.

Proof. See Section 1 in [21] ■

Theorem 1. *(Smooth transition) Given two reference masked grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ with a and b amount of in-shape cells, respectively, let $\mathcal{G}_a, \mathcal{G}_{a+1}, \ldots, \mathcal{G}_b$ be the masked grids generated for the swarms with a size of $a, a+1, \ldots, b$. Among all the ways to determine the size of each subset used in interpolation subroutine, Alg. 3 Line 6-10 is the optimal way for minimizing the following objective:*

$$\max_{a \leq i \leq b-1} |\mathcal{S}(\mathcal{G}_i) - \mathcal{S}(\mathcal{G}_{i+1})| + |\mathcal{S}(\mathcal{G}_{i+1}) - \mathcal{S}(\mathcal{G}_i)| \tag{2}$$

Proof. See Section 2 in [21] ■

4 Performance Evaluation

In this section, we empirically study the performance of algorithm proposed in this paper. Given a goal shape, in Section 4.1, we first study the quality of configurations generated for the swarms with different sizes, then, in Section 4.2 and Section 4.3, the generated configurations were formed by a swarm of simulated robots as well as a swarm of physical swarms using the shape formation algorithm proposed in [9], and the results show that the proposed algorithm can indeed make the swarm adapt to the swarm size change quickly and robustly.

In the experiments, we use four complex shapes as our goal shapes: the "N", the "star", the "wrench", and the "circle". These four goal shapes are shown in the Fig. 4.

4.1 Experiments on Generated Configurations

First, given each goal shape, we use the proposed algorithm to generate the goal configurations with in-shape cell number ranging from around 20 to 1024. Recall that as stated in the Section 2.1, we have two design considerations: the similarity between each generated configuration and the goal shape, and the similarity between the configurations generated for difference swarm sizes. The videos of

Fig. 4: Goal shapes used in the experiments. From left to right: the shape "letter N", the shape "star", the shape "wrench", and the shape "circle".

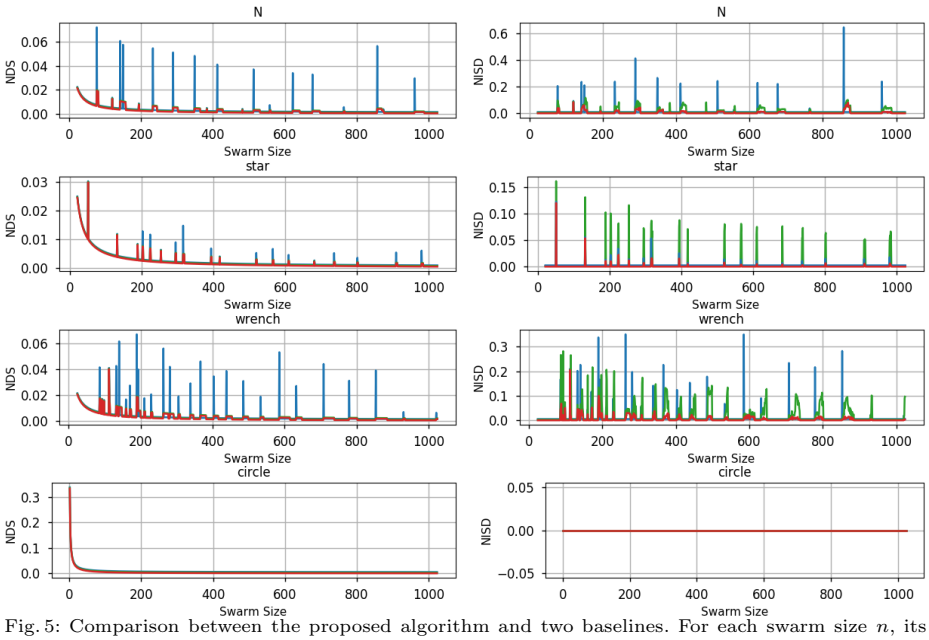

Fig. 5: Comparison between the proposed algorithm and two baselines. For each swarm size n, its corresponding data points on the plots are the NDS and NISD between the generated masked grids with n and $n+1$ in-shape cells, respectively. The red plots are the results for the proposed algorithm, the blue plots are the results for the baseline 1, and the green plots, which overlap with the red plots in all NDS plots, are the results for the baseline 2.

the configurations generated for different swarm sizes can be found in [21]. In addition, we introduce two metrics to qualitatively evaluate the similarity between each pair of the binary images whose difference of in-shape cell number is one: the *normalized difference score* (NDS), and the *normalized inter-shape distance* (NISD). The NDS is the ratio between those two configurations' *difference score* and the sum of two configurations' in-shape cell numbers. The NISD is defined as follows: given two masked grid $\mathcal{A}$ and $\mathcal{B}$ where $|\mathcal{S}(\mathcal{A})| \leq |\mathcal{S}(\mathcal{B})|$, we assign cells in $\mathcal{S}(\mathcal{B})$ to the cells in $\mathcal{S}(\mathcal{A})$ in a way such that: (i) for each in-shape cell on $\mathcal{A}$, we assign exactly one in-shape cell on $\mathcal{B}$ to it, in addition, (ii) each in-shape cell on $\mathcal{B}$ can be assigned to no more than one cell on $\mathcal{A}$. The cost of each pair of in-shape cell on $\mathcal{A}$ and its assigned in-shape cell from $\mathcal{B}$ is given by the Manhattan distance between them in cells. The NISD is the ratio between the minimal total cost that any feasible assignment can achieve and sum of two configurations' in-shape cell numbers. Intuitively, the NDS shows the "mismatch" between two configurations, and NIDS essentially characterizes the minimal average distance traveled by the swarm to transform from one configuration to the other. The plots showing these two metrics over difference in-shape cell numbers for each goal shape are shown in Fig. 5. In addition, we also compare the proposed algorithm with two baselines. Both of those two baselines use the same pipeline as our algorithm does. The difference between our algorithm and the baseline 1 is that: when executing the *interpolation* subroutine, instead of using Alg. 3 Line 6-10 to determine each subset's size, the baseline 1 will aggressively set sz^0 to be $|\mathcal{D}_{l-h}|$ and set $sz^i \dots sz^{k-1}$ to be 0. The difference between the our algorithm

and the baseline 2 is that: when executing the *interpolation* subroutine, instead of using Alg. 3 Line 11-24 to assemble each subset, the baseline 2 will naively fit the cells into each subset by the lexical order of each cell's coordinate. In the *interpolation* subroutine, there are two subproblems to be solved: how to determine each subset's size, and how to determine the content of each subset. These two baselines are essentially two naive solutions to those two subproblems.

As expected, in these plots, we can see that our algorithm outperforms two baselines with respect to both NDS and NISD for all four goal shapes, confirming that our way to determine the size and content of each subset in *interpolation* subroutine (Alg. 3 Line 6-10, Line 11-24) can indeed make the transition between goal configurations generated for different swarm sizes more "smooth". Note that our algorithm and baseline 2 uses the same way to determine each subset's size in *interpolation* subroutine, as a result, in all NDS plots, our algorithm (black) overlaps with baseline 2 (green). The other counter-intuitive observation here is that: the NISD for the shape "circle" is 0 for all the swarm sizes, this is because: using the pipeline presented in the paper, for any swarm size n, the "circle" generated for the swarm size n will always be "inside" the "circle" generated for the swarm size $n+1$, therefore, the "circle's" NISD is by definition 0 for all the swarm sizes.

4.2 Experiments on Simulated Robotic Swarm

In the simulation, a swarm of up to 100 simulated *Coachbot* robots were tasked to use the shape formation algorithm proposed in [9] to form the configurations generated from our algorithm. The simulation consists two main components: a world engine written in C that simulates robot's on-board hardware resources, and a user-code loader written in python that executes the user's code. The world engine simulates the *Coachbot* robot's motion, sensing and communication in a very realistic way, that is, the specifications of all the simulated hardware, including the maximal speed of robot's wheel, sampling rate of robot's positioning sensor, throughput of the inter-robot communication channel, etc, are made to be consistent with the real robot. In addition, the user-code loader is designed in a way that: the code used to operate the simulated robot can be used to operate the actual *Coachbot* robot without any modification. In the simulation, the communication rate is $20hz$, the maximal speed of robot's wheel is $0.1m/s$, and each edge on the grid has a length of $0.3m$. The demonstration videos of the simulation can be found in [21].

In the first experiment, we study the how the addition of the robots will affect the swarm's behavior. In this experiment, the swarm size is initialized to be around 20. Every time when robots currently in the swarm complete forming the shape, we add one more robot in a random location near the swarm and broadcast the new swarm size to the swarm. The robots will update their goal configuration according to the new swarm size, and then start to form the new goal configuration. This process will be repeated until the swarm size gets to 100. For each goal shape, we repeat this experiment 50 times, and in each trial, we study two metrics: the response time (RT), which is time between the swarm size

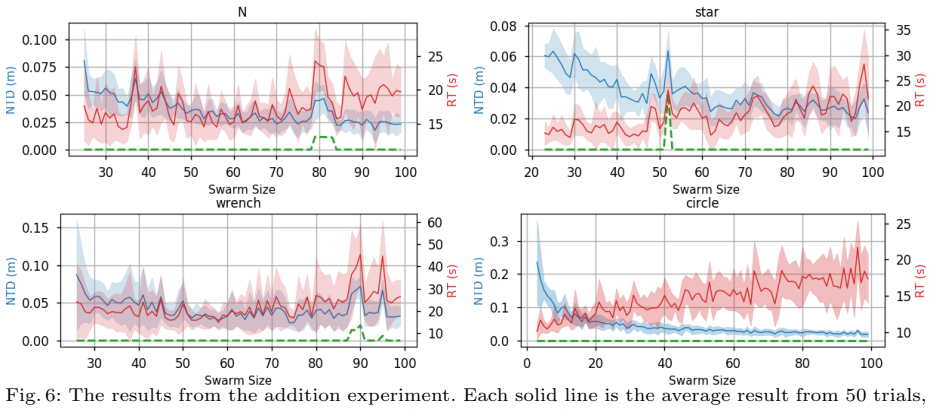

Fig. 6: The results from the addition experiment. Each solid line is the average result from 50 trials, and the colored shade areas show the confidence intervals for NTD and RT over swarm size at a confidence level of two σ. Each green dotted line is the shape's NISD obtained from the previous section.

change and the swarm forming the shape at the new scale, and the normalized travel distance (NTD), which the average distance traveled by the swarm to form the shape at the new scale. The results from 50 trials are shown in Fig. 6.

In addition, besides the addition of the robots, we are also interested in effect of the removal of robots on the swarm's behavior. In the second experiment, the swarm size is initialized to 100, and similar to the first experiment, every time when the current swarm complete forming the shape, we randomly choose one robot currently in the swarm, remove it from the swarm, and broadcast the new swarm size to the robots. The robots will update their goal configuration according to the new swarm size, and then start to form the new goal configuration. This process will be repeated until the swarm size gets to minimal swarm size required to display the shape. For each goal shape, we repeat this experiment 50 times, and in each trial, the metric RT and NTS are investigated, see Fig. 7 for the results from all 50 trials. As we can see in the plots, in both addition and subtraction experiments, every time when the swarm size change occurs, the swarm is able to adapt quickly, within 40 s to be more specific. Moreover, one can observe that for each shape, at some certain swarm sizes, the RT and NTD change sharply in both subtraction and addition experiments. For example, for the shape "wrench", the RT and NTD spike at the swarm size 52. To investigate the cause of these spikes, we compare the NISD obtained from previous section (green dotted line) with the NTD and RT obtained from the simulation. Unsurprisingly, the results show that the swarm sizes where the NTD and RT spike are consistent with the swarm sizes where the shape's NISD spikes, in the other words, the swarm sizes where the RT and NTD spike are the swarm sizes where the generated goal configurations change greatly.

4.3 Experiments on Physical Robotic Swarm

Beyond the simulations, we also experiment on a swarm of up to 100 real *Coachbot* robots. In the experiment, the robots are tasked to form the shape "N". The swarm size starts to be 100, every time when the current swarm complete forming the shape, we remove a batch of robots from the swarm and then broadcast

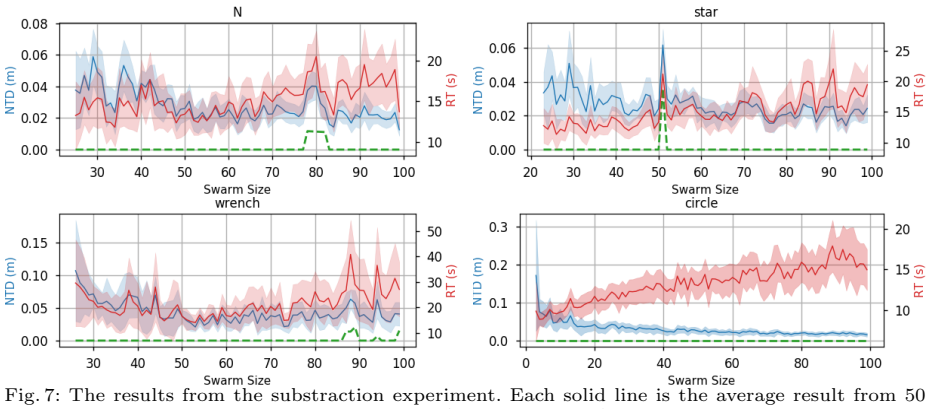

Fig. 7: The results from the substraction experiment. Each solid line is the average result from 50 trials, and the colored shade areas show the confidence intervals for NTD and RT over swarm size at a confidence level of two σ. Each green dotted line is the shape's NISD obtained from the previous section.

the new swarm size to the robots. The robots use the presented algorithm to update their goal configuration according to the new swarm size, and then start to form the new goal configuration. This process is repeated until the swarm size gets to 23. See Fig. 1 for the still images from the experiment, and the video for this experiment can be found in [21]. As we can see in the video, when the swarm size changes, the robots adapt quickly and robustly.

5 Conclusion

In this paper, we present an algorithm that encodes a 2D shape to a set of cells on a grid each characterizing a robot currently in the swarm. The performance of the algorithm is thoroughly evaluated via both the simulated swarm and physical swarms. The experiments show that the generated goal configuration for the swarm is consistent with the input shape, in addition, when the swarm size changes, it allows robots to adapt quickly and robustly.

References

[1] Guy Scher et al. "Warehouse Automation in a Day: From Model to Implementation with Provable Guarantees". In: *IEEE 16th International Conference on Automation Science and Engineering*. IEEE. 2020, pp. 280–287.

[2] Mathieu Le Goc et al. "Zooids: Building blocks for swarm user interfaces". In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 2016, pp. 97–109.

[3] Sebastian Trüg et al. "Applying automatic planning systems to airport ground-traffic control–a feasibility study". In: *Annual Conference on Artificial Intelligence*. Springer. 2004, pp. 183–197.

[4] A Hsieh Mong-ying et al. "Pattern generation with multiple robots". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006.* IEEE. 2006, pp. 2442–2447.

[5] Chien Chern Cheah et al. "Region-based shape control for a swarm of robots". In: *Automatica* 45.10 (2009), pp. 2406–2411.
[6] M. ani Hsieh et al. "Decentralized Controllers for Shape Generation with Robotic Swarms". In: *Robotica* 26.5 (Sept. 2008), pp. 691–701. ISSN: 0263-5747.
[7] Michael Rubenstein et al. "Scalable self-assembly and self-repair in a collective of robots". In: *2009 IEEE/RSJ international conference on Intelligent robots and systems*. IEEE. 2009, pp. 1484–1489.
[8] Kasper Stoy et al. "Self-repair through scale independent self-reconfiguration". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2. IEEE. 2004, pp. 2062–2067.
[9] Hanlin Wang et al. "Shape Formation in Homogeneous Swarms Using Local Task Swapping". In: *IEEE Transactions on Robotics* (2020).
[10] Michael Rubenstein et al. "A scalable and distributed approach for self-assembly and self-healing of a differentiated shape". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 1397–1402.
[11] Melvin Gauci et al. "Programmable self-disassembly for shape formation in large-scale robot collectives". In: *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 573–586.
[12] Javier Alonso-Mora et al. "Multi-robot system for artistic pattern formation". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4512–4517.
[13] John W Romanishin et al. "3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions". In: *2015 IEEE International Conference on Robotics and Automation*. IEEE. 2015, pp. 1925–1932.
[14] Mark Yim et al. "Modular self-reconfigurable robot systems [grand challenges of robotics]". In: *IEEE Robotics & Automation Magazine* 14.1 (2007), pp. 43–52.
[15] Jimming Cheng et al. "Robust and self-repairing formation control for swarms of mobile agents". In: *AAAI*. Vol. 5. 2005. 2005.
[16] Rukundo Olivier et al. "Nearest neighbor value interpolation". In: *Int. J. Adv. Comput. Sci. Appl* 3.4 (2012), pp. 25–30.
[17] Daniel Vaquero et al. "A survey of image retargeting techniques". In: *Applications of Digital Image Processing XXXIII*. Vol. 7798. International Society for Optics and Photonics. 2010, p. 779814.
[18] YY Zhang et al. "A parallel thinning algorithm with two-subiteration that generates one-pixel-wide skeletons". In: *Proceedings of 13th International Conference on Pattern Recognition*. Vol. 4. IEEE. 1996, pp. 457–461.
[19] Robert Fisher. Distance transform. URL: https://tinyurl.com/22uacjz2.
[20] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.
[21] Hanlin Wang et al. Supplementary Materials. URL: https://tinyurl.com/2huc42t6.