On Genetic Algorithms for the Edge Unfolding of Polytopes

Dr. Saleha Raza
Associate Professor, CS
Habib University
Karachi, Pakistan
saleha.raza@sse.habib.edu.pk

Musab Kasbati
Computer Science
Habib University
Karachi, Pakistan
mk07811@st.habib.edu.pk

Raahim Hashmi
Computer Science
Habib University
Karachi, Pakistan
rh08461@st.habib.edu.pk

***Abstract*—Dürer's conjecture states that every polytope has a non-overlapping edge unfolding. It was first proposed in 1975 by Shephard who named it after Albrecht Dürer, a renowned German artist and mathematician. While there exists algorithms that have been successful at finding non-overlapping edge unfoldings (*nets*) of polytopes, there exists no proof of their correctness. As the possible unfoldings is exponential in the features of the polytope, such as its number of faces or number of vertices, we found that it was worth investigating a meta-heuristic approach that explores the vast search space and outputs an unfolding with minimal, and ideally zero, overlaps.**

***Index Terms*—polytopes, unfolding, computational geometry, genetic algorithms, dürer's conjecture**

I. Introduction

The term polytope can be ambiguous due to different meanings across literature. For this report, we define polytopes as three-dimensional bounded convex polyhedrons. An unfolding, or specifically, an edge-unfolding of a polytope can be thought of as cutting the polytope along its edges and laying its faces out flat in a two-dimensional plane such that no faces are disconnected (see Figure 1). A *net* is an edge-unfolding of a polytope such that the faces do not overlap. Whether or not every polytope has a net is an open problem in computational geometry known as Dürer's Conjecture, first proposed by Shephard in 1975 and named after Albrecht Dürer, a renowned German artist and mathematician whose book "Underweysung der Messung" (Instruction in Measurement with Compass and Straightedge) contributed greatly to the understanding of polyhedra and their two-dimensional unfoldings.

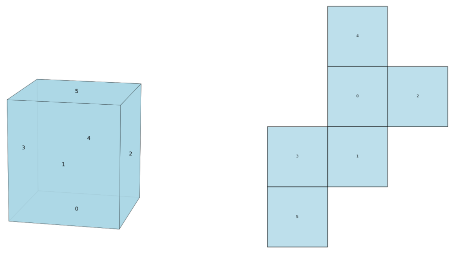

Fig. 1. An example of a cube and one of its possible nets.

The natural approach of cutting along the edges to obtain an unfolding of a polytope can be formally defined as obtaining a *cut-graph* $G_c = (V_c, E_c)$ such that G_c is a spanning tree of the polytope's vertex graph $G_p = (V, E)$ where V is the set of vertices of the polytope and E is the set of edges between the vertices. The edge set E_c are the edges to cut along to obtain an unfolding. It can be proven that cutting the edges in any spanning tree of G_p gives an unfolding for the polytope (Lemma 22.1.1 of [1]). It can also be proven that for every cut-graph G_c, there exists a *join-graph* $G_j = (V_j, E_j)$ that is a spanning tree of the polytope's face/dual graph $G_f = (V_f, E_f)$, where V_f is the set of faces of the polytope and $(u, v) \in E_f$ if $u, v \in V_f$ and there exists a common edge between the faces u and v (Lemma 2.2 of [2]). While a cut-graph presents a natural way of obtaining an unfolding, that is by cutting along its edges, the join-graph turns out to be computationally easier to deal with and is a simpler way to view the unfolding, e.g. the unfolding in figure 1 can be viewed as a join-tree of the dual graph of the cube.

It is important to note that not all spanning trees result in unfoldings that are nets, and as such, the problem of finding nets of a polytope reduces to finding a cut-graph or a join-graph that results in a net. As the number of spanning trees for a given graph can be exponential in the number of vertices, it is not possible to check all spanning trees to find a net. Work has been done on designing algorithms that attempt to produce a net for a given polytope by generating a spanning tree according to some heuristic; however, no algorithm has been proven to always find a net, despite algorithms existing that have always succeeded to find a net for given polytopes. In this report, we present, to the best of our knowledge, the first metaheuristic approach for finding nets of polytopes. We use a genetic algorithm to obtain a spanning tree of the polytope's dual graph and compare the results with existing algorithms.

II. Related Work

The most substantial resource on algorithms for finding nets of polytopes is the thesis of Schlickenrieder [2]. In his thesis, Schlickenrieder explores 24 algorithms that take in a vertex-graph or a face-graph of a polytope and output a spanning tree. The algorithms include common graph algorithms such as BREADTH-FIRST-SEARCH and DEPTH-FIRST-SEARCH, algorithms to test his or other's conjectures such as MIN-PERIMETER which produces a minimum spanning tree of the vertex graph to to test a conjecture that the unfolding with minimal perimeter is always a net, as well as algorithms that are combine aspects or heuristics of other algorithms. He tested the algorithms on a dataset of about 10000 polytopes and measured the percentage of produced unfoldings that had overlaps as the metric of the algorithm's performance. Among the algorithms he tested, the STEEPEST-EDGE-UNFOLD algorithm was the most successful, with only 1.2% of produced unfoldings having overlaps. The algorithm is based on his conjecture that cutting along the edges that are the "straightest" according to some metric can produce a net. For this purpose, the algorithm works by randomly generating a unit vector and taking the vertex that is least in the direction of the vector as the root. Then, from each vertex, it selects edges that are the most in the direction of the vector. Due to the random nature of the algorithm, running it multiple times can produce different unfoldings. Schlickenrieder found that STEEPEST-EDGE-UNFOLD always managed to produce a net within 7 attempts for all the polytopes in his dataset.

III. METHODOLOGY

A. Genetic Algorithms

Genetic algorithms (GAs) are a class of metaheuristic optimization algorithms used for combinatorial problems. They are fundamentally inspired by biological evolution and natural selection. GAs work by maintaining a generation of *candidates* which are possible solutions to the problem, iteratively improving upon them through operations that mimic biological processes such as genetic recombination and mutation, and selecting the fittest candidates to survive to the next generation. A GA is typically composed of the following steps:

(1) **Initialization:** A population of candidates is randomly generated.
(2) **Evaluation:** Each candidate is evaluated using a fitness function that measures how well it solves the problem.
(3) **Parent Selection:** Candidates are selected to be parents based on some selection scheme, such as rank-based selection or binary tournament selection.
(4) **Crossover:** The information of parent solutions is combined to create offspring candidates. A *crossover operator*, such as uniform crossover or two-point crossover, is used for this purpose.
(5) **Mutation:** Based on some small probability, the offsprings undergo a small mutation that alters their information, potentially introducing new information to the population. A *mutation operator* is used for this purpose.
(6) **Survivor Selection:** The next generation of candidates is selected from the current generation and the offspring based on some selection scheme, such as truncation or fitness-proportionate selection.
(7) **Termination:** Steps 2-6 are repeated until a termination condition is met, such as a maximum number of generations or the population converging to a solution.

To facilitate this proceedure, a representation of the candidate solution, called a *chromosome*, a fitness function, and crossover and mutation operators are required. The size of the population, parent and survivor selection schemes, and the mutation probability are also typical hyperparameters of a GA that can be tuned to improve the its performance.

B. GA for Unfolding Polytopes

Our GA for unfolding polytopes takes in a polytope's dual graph and maintains a population of candidate solutions that can be converted to join-graphs. It aims to minimize the number of overlaps in the produced unfoldings.

1) Chromosome Representation: For each candidate, we maintained an array of length $|E_f|$ of integers between 0 and $|E_f| - 1$ such that the i-th index represented the weight of the i-th edge in the dual graph. To convert the chromosome to a join-graph, we simply used Prim's Algorithm to obtain the minimum spanning tree of the dual graph with the weights of the edges set to the values in the chromosome. This representation allowed us to easily initialized a random population of candidates, as well as apply crossover and mutation operators with little consideration regarding the validity of the resulting candidates.

2) Fitness Function: The fitness of a candidate was taken to be the number of overlaps in the unfolding produced by the join-graph of the candidate. To check for the number of overlaps, we relied on the Separating Axis Theorem (SAT) which states that two convex shapes do not overlap if and only if there exists an axis such that the projections of the two shapes on the axis do not overlap. Additionally, we can significantly reduce the number of axes we need to check by only checking the axes formed by the normal vectors of the faces of the polytope. Furthermore, instead of comparing every pair of faces, we can do what is called a *broad-phase check* by first comparing the x coordinates of the bounding boxes of the faces. If the bounding boxes do not overlap, we can skip checking the pair of faces as they cannot possibly overlap. Once we have the pair of faces for whom we do need to check for overlap, we can use the SAT for a *narrow-phase check*. Once we have the number of overlaps, we can simply use that as a fitness value for the candidate.

3) Crossover and Mutation Operators: We used a uniform crossover operator which builds the offsprings by tossing a coin for each index in the parents' chromosomes and taking the value from one of the parents. This was mostly unproblematic; however, it did leave room for multiple edges to have the same weight which isn't an issue immediately but may have caused issues if allowed to persist. To deal with this, we permuted the edges that had the same weights so that their order with respect to each other was not preserved because - ?

C. Generating Polytopes

In order to generate polytopes for testing our GA, we identified a few categories of polytopes from Schlickenrieder's thesis [2] that were among the hardest to unfold. The categories we used were:

- **Uniform Polytopes:** These are polytopes that were generated by taking the convex hull of n points in the bounding box defined by the vertices in $\{-n, n\} \times \{-n, n\} \times \{-n, n\}$.

- **Flat Polytopes:** The generation of these polytopes was similar to the uniform polytopes, except that with a 0.7 probability, the i-th point would have the same x and y coordinates as the $(i-1)$-th point with its z coordinate equal to $-n$ where n is the total number of points. Note that the first point was generated randomly in the same bounding box as the uniform polytopes. This generation gives a polytope with a higher number of faces that have more than 3 vertices.
- **Spherical Polytopes:** These polytopes were generated by taking the convex hull of n points on the surface of a unit sphere. The points were generated by taking random values of θ in the range $[0, 2\pi]$ and ϕ in the range $[0, \pi]$ respectively and inputting them into the parametrized equation of the sphere: $\langle \sin(\phi)\cos(\theta), \sin(\phi)\sin(\theta), \cos(\phi)\rangle$.
- **Half-spherical Polytopes:** These polytopes were generated in the same way as the spherical polytopes, with the range of ϕ being $[0, \pi/2]$ instead of $[0, \pi]$. The resultant polytope has a relatively flat base and points concentrated to the top of the polytope.
- **Turtle Polytopes:** The vertices for these polytopes were points in the set $\{(x, y, x^2 + y^2) \mid x \in \{-i, -i+1, \ldots, i\}, y \in \{-j, -j+1, \ldots, j\}\}$ where i and j were randomly chosen integers in the range $[1, 7]$.

D. Visualization

Once the points of the polytopes were generated, we obtained their convex hull and the simplices (triangulations of faces) of the resulting polytope using functions from Python's `scipy` library. Then, we merged those simplices by obtaining their plane equations and taking the union of the points of the simplices that had the same plane equation. This allowed us to obtain the faces of the resulting polytopes. Furthermore, we sorted the points with respect to their angle with the centroid of the face in order to figure out the edges between the points of the face; with this ordering, there was an edge between two consecutive points and the first and last point. To obtain the dual graph of the polytope, we initialized a graph with the faces of the polytope as its nodes and an empty edge set. We then iterated through the faces and added an edge between two faces if they had an edge in common.

After obtaining the dual graph, we could proceed with obtaining unfoldings of the polytope. However, we still needed to visualize the unfolding. For this, we first fixed the orientation of the vertices in the array defining a face. As two adjacent faces must share an edge, that is they have two vertices in common and in both faces they appear consecutively, we made sure that the two vertices were in the same order in both faces. This required simply reversing the order of the vertices in one of the faces if they were not in the same order as the other face.

E. Abbreviations and Acronyms

F. Units

G. Equations

H. LaTeX-Specific Advice

I. Some Common Mistakes

J. Authors and Affiliations

Acknowledgment

References

References

[1] E. D. Demaine and J. O'Rourke, *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge: Cambridge university press, 2007.
[2] W. Schlickenrieder, "Nets of Polyhedra."