Maple's Graph theory package extension

MITACS Summer School 2007

Raul Aliaga, Andrew Arnold, Nikolas Karalis, Wendy Wu

August 17, 2007

Contents

Contents

Contents

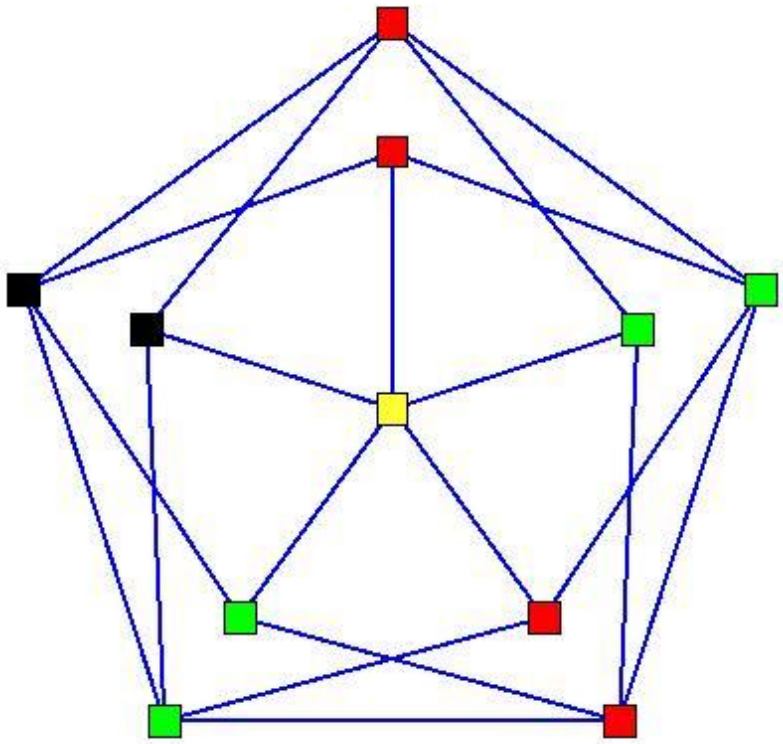
Contents

Contents

- Important problem, many applications.
- NP-complete, a difficult problem.

Definitions

- Graph Coloring.



- Heuristic approach's:
  - Binary LP problem
  - Degree's and Greedy Colorings
- Intensive tests.

Contents

The model

$$\min \sum_{k=1\ldots MaxColor} \sum_{i1\ldots|V|} X_{i,k}k$$

s.t: The "proper coloring" condition

$$A_{i,j}(X_{i,k} + X_{j,k}) \leq 1 \quad \forall k = 1 \ldots MaxColor \quad \forall i,j = 1 \ldots |V|$$

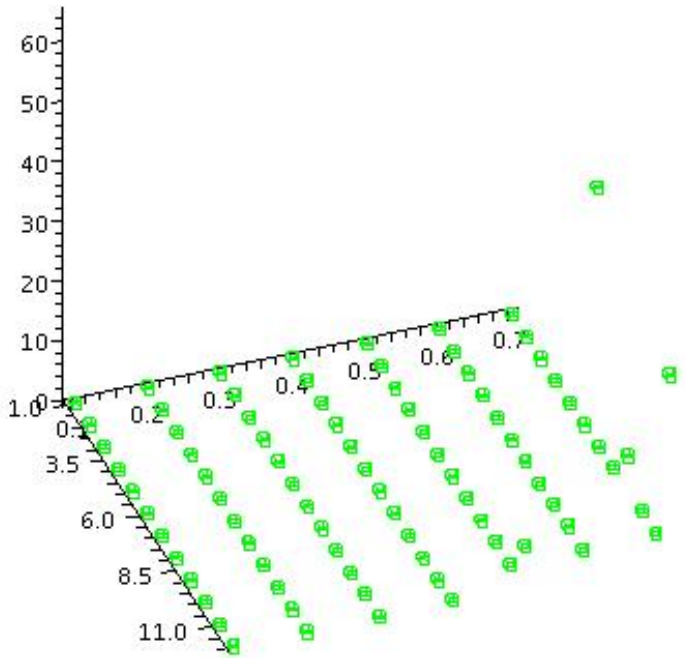
and that every vertex uses at least one color -and no more-

$$\sum_{k=1\ldots MaxColor} X_{i,k} = 1 \quad \forall i = 1 \ldots |V|$$

Results

:

- Branch and Bound procedure, time expensive!.



Contents

Greedy General

1. Define a degree for every vertex.
2. Choose an uncolored vertex with highest degree value given by the previous definition
3. Repeat until every vertex is colored.

Degree's definitions

- **"Traditional" degree**, defined as:

$$d(v) = \sum_{e=uv, e\in E(G), u\in V(g)} \quad \forall v \in V(G)$$

i.e., the amount of incident vertices to a vertex v by edges in $E(G)$.

- **Color Degree (CD)**:

$$d_{color}(v) = \sum_{e=uv, e\in E(G), u\in V(g), C(v)\neq 0} \quad \forall v \in V(G)$$

i.e., the already colored vertices incident to v.

- **Different Color Degree (CDD)**:

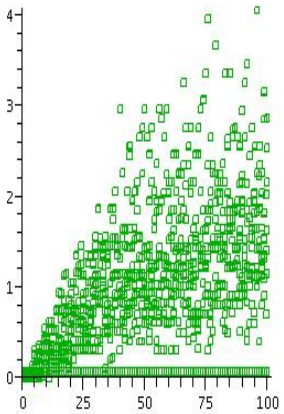
$$d_{dcolor}(v) = \sum_{\substack{e = uv, e \in E(G), u \in V(g), \\ C(v) \neq 0, C(v) \neq C(w) \forall u, w \in N(v)}} \quad \forall v \in V(G)$$

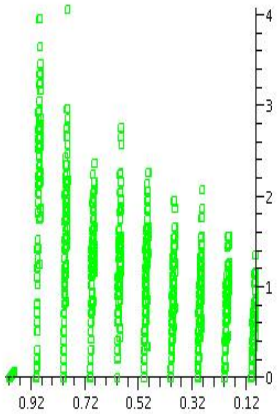
i.e., the amount of different colors that the already colored neighbours of v have.

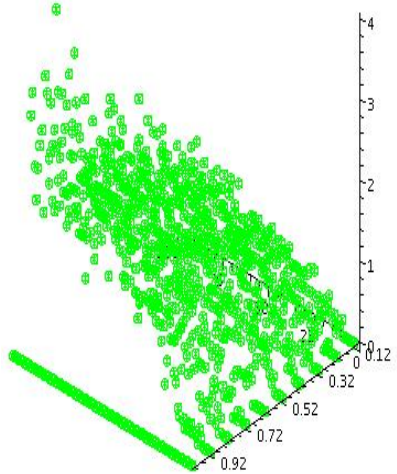
Tests

- Tests for
  - Degree Permutation
  - Already-colored degree
  - Already-colored-with-different colors degree
- Difference of numbers of colors compared with greedy algorithm with traditional degree

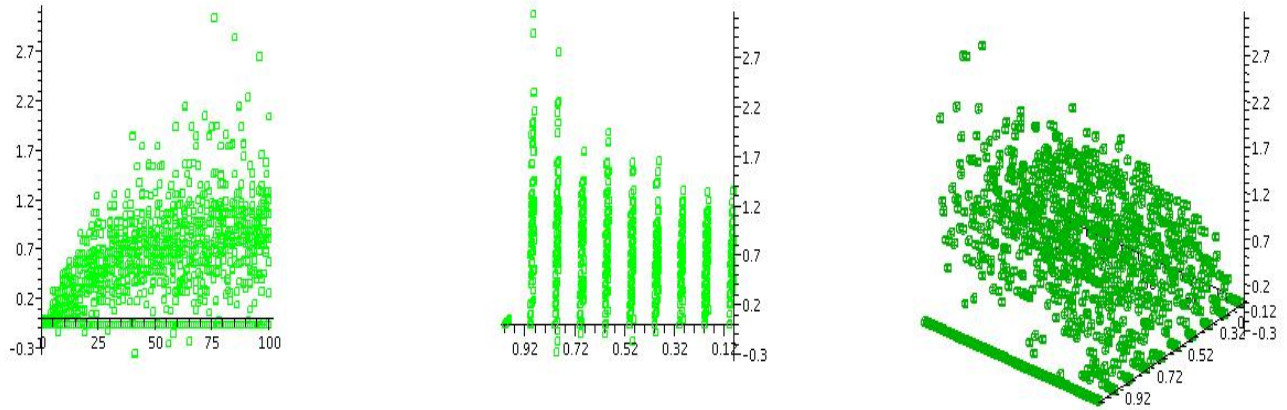
Results: Greedy with Permutation



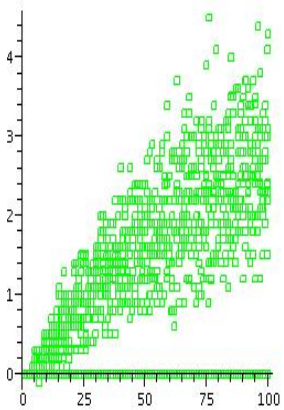


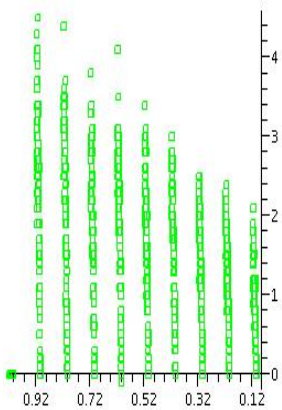


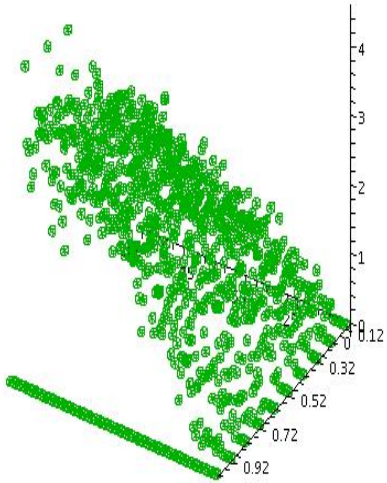
Results: Greedy with Already-colored Degree



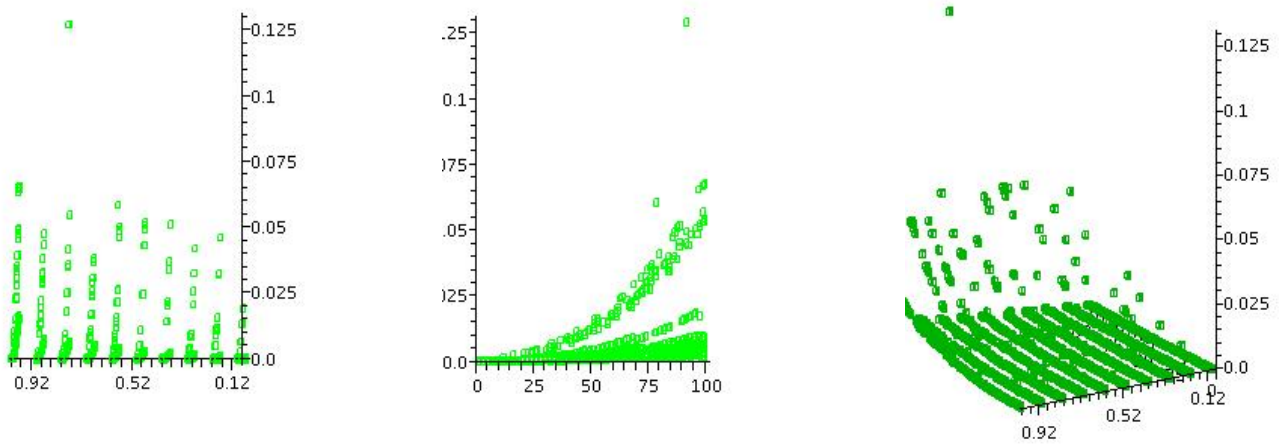
Results: Greedy with visible distinct colors



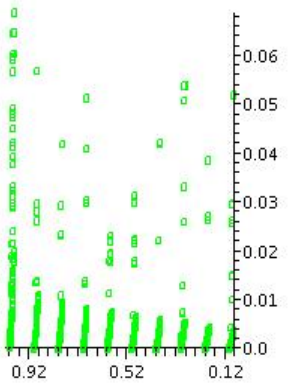


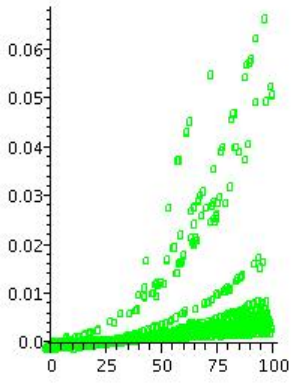


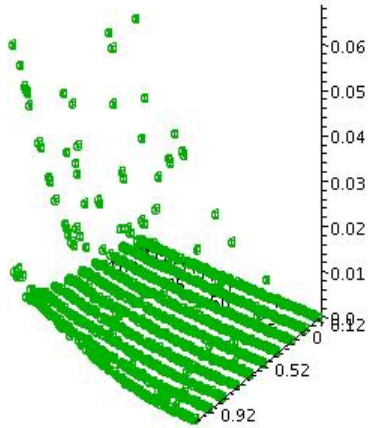
Results: Greedy coloring time



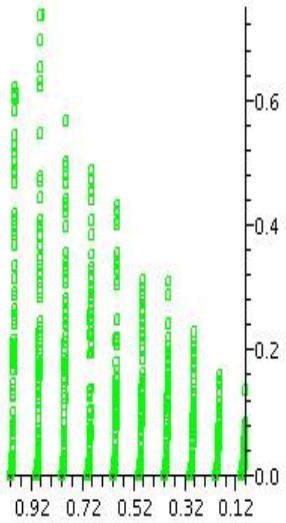
Results: Degree permutation time

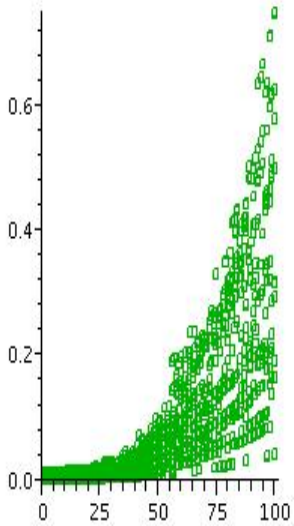


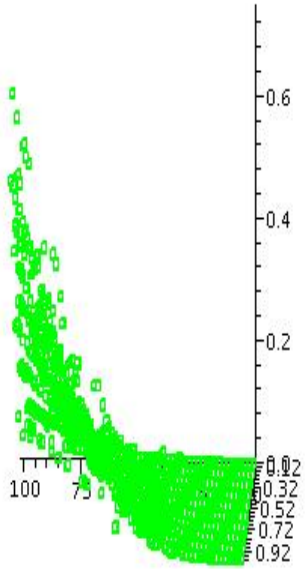




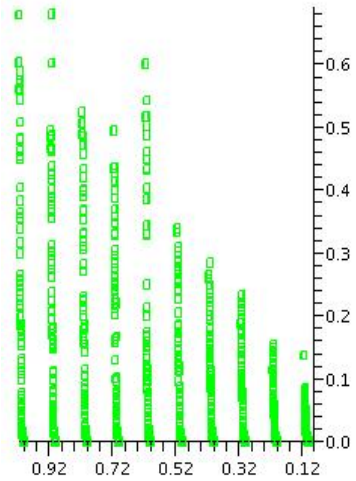
Results: Already colored time

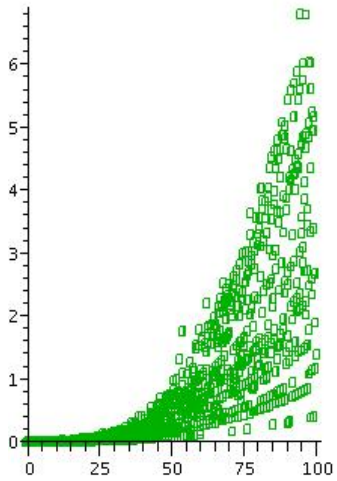


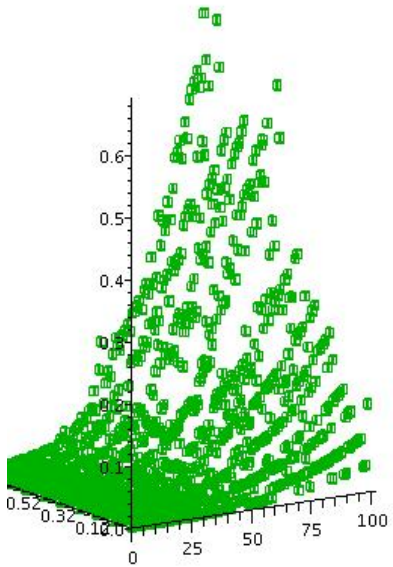




Results: Distinct colors time







Looking for improvements

- Heap Data structure
- Randomization of the vertex selection

Conclusions

- Good Heuristic, but with increasing costs
- Possible better asymptotic perfomance

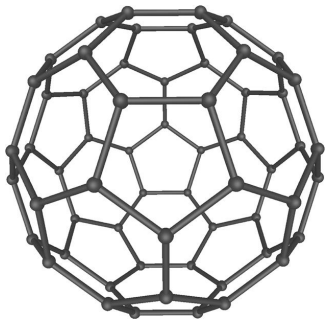
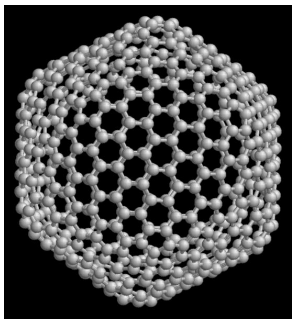
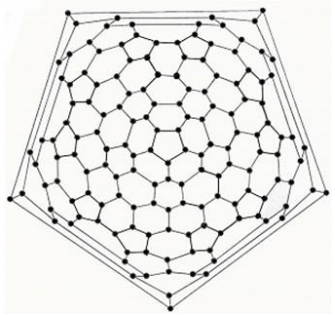
Future work

- Possible chance of combinatorial structures of the LP solution polyhedron
- More intensive and large tests

Contents

Constructing Random Fullerene Isomers

Fullerenes are carbon molecules that form a closed surface.

In graph theory, fullerenes refer to any 3-regular, planar graphs comprised of pentagonal and hexagonal faces.

The problem:

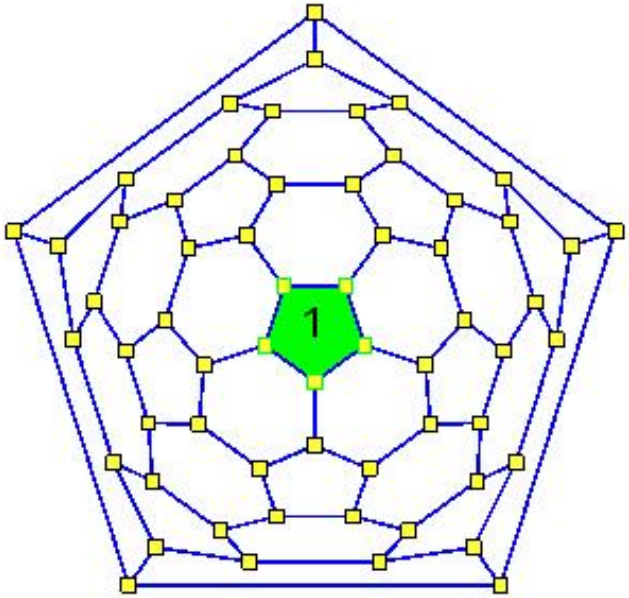
Implement a fast algorithm in Maple that, given a specified number of vertices, generates a random fullerene isomer.

Contents

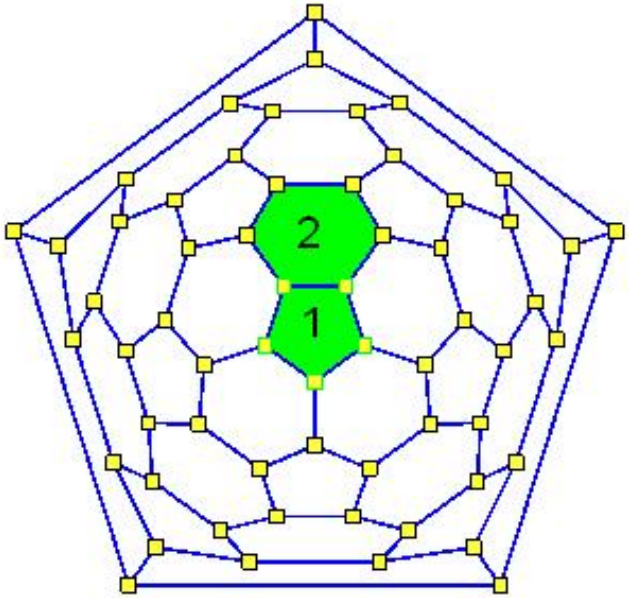
The Spiral Code Algorithm

In practise, most fullerenes can be encoded as a spiral code.

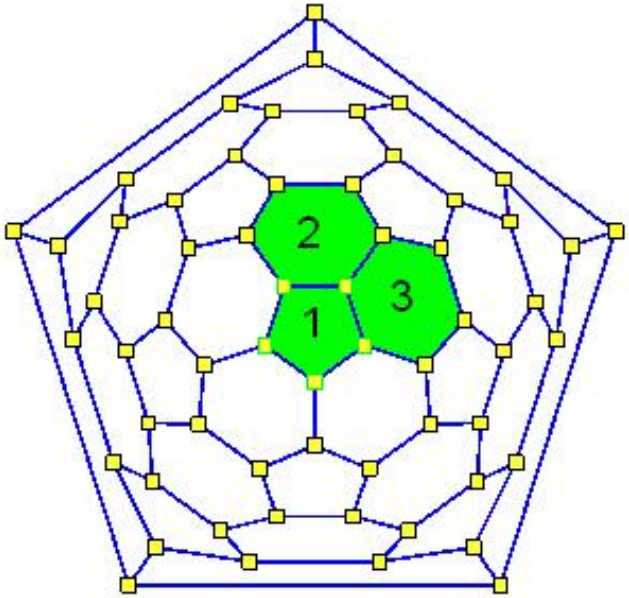
- To generate a spiral code for a fullerene, pick a starting face, then spiral outward, passing every face once, until you've traversed them all.
- The spiral code is the number of edges per face, ordered as in the spiral.



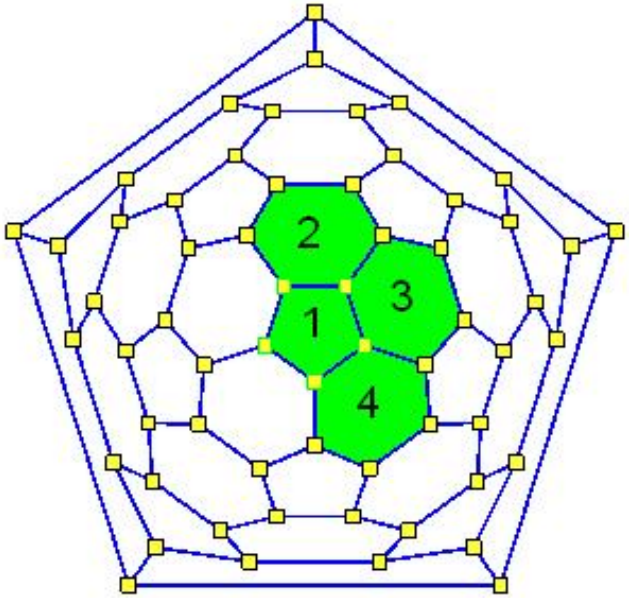
5



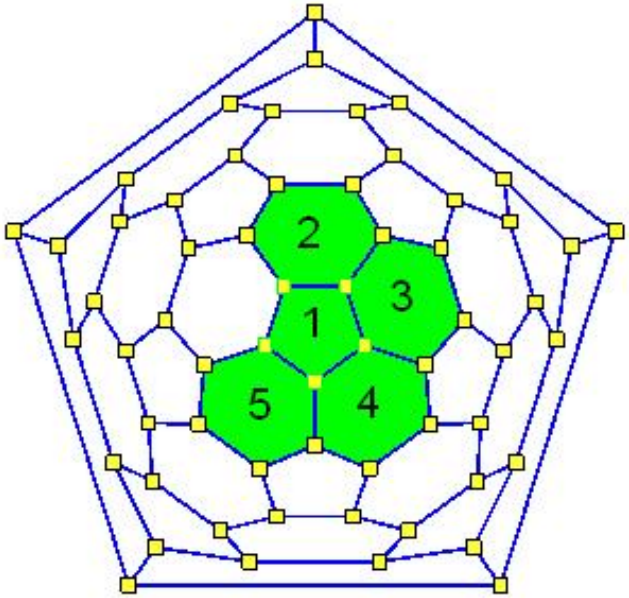
56



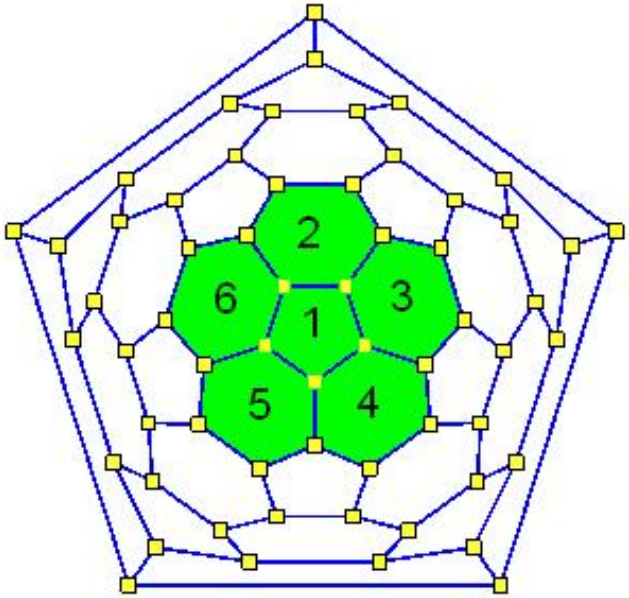
566



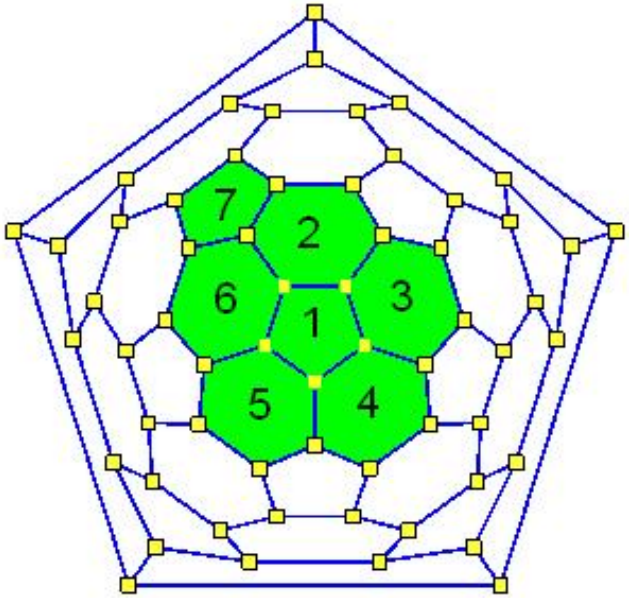
5666



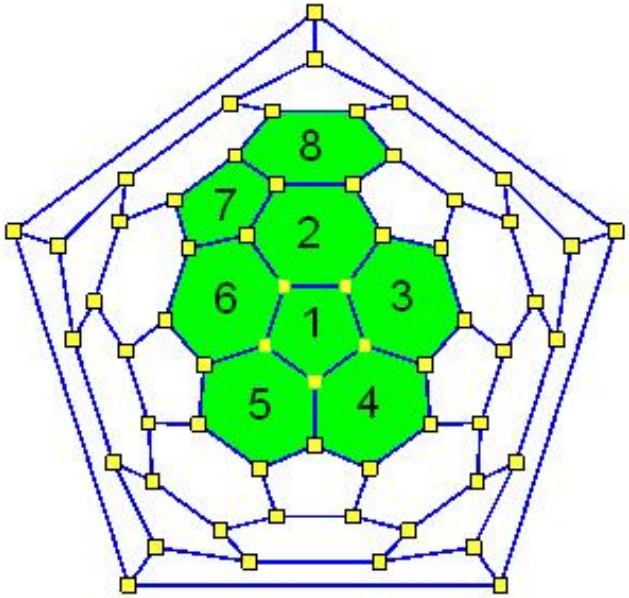
56666



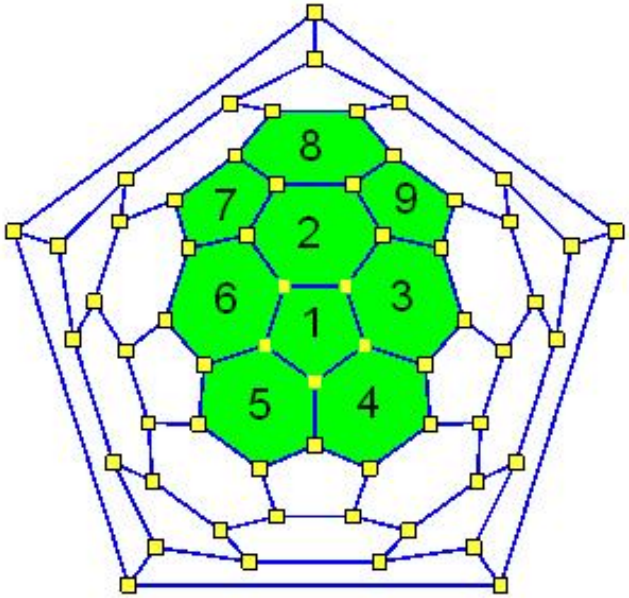
$$566666$$



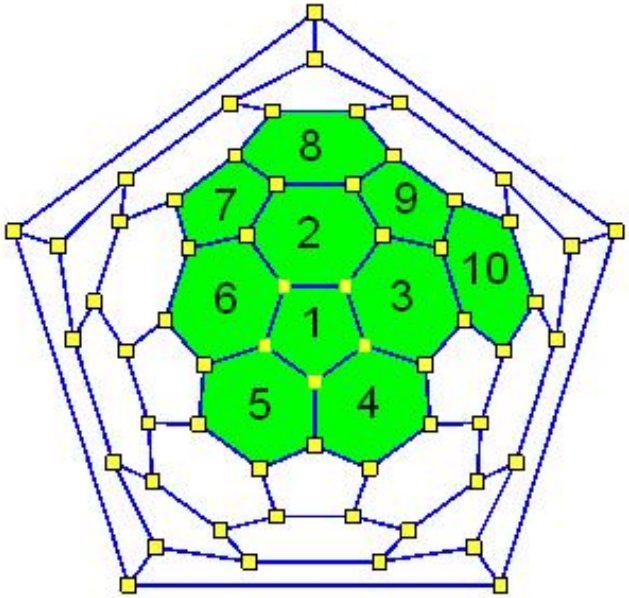
5666665



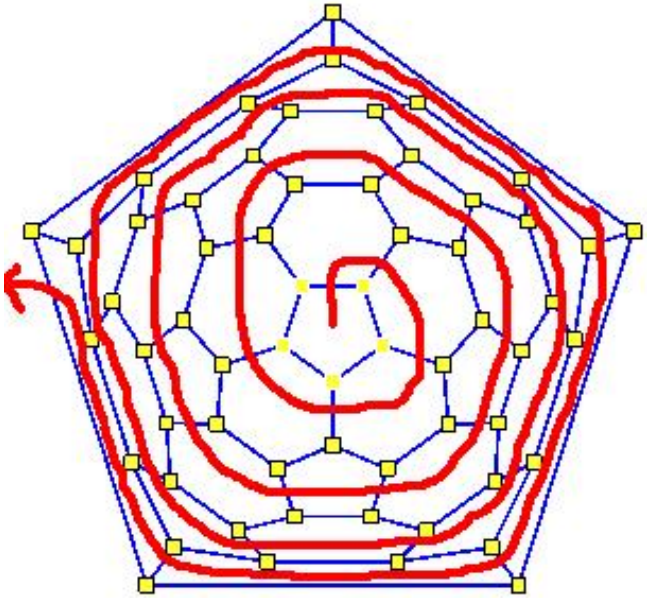
56666656



566666565



5666665656



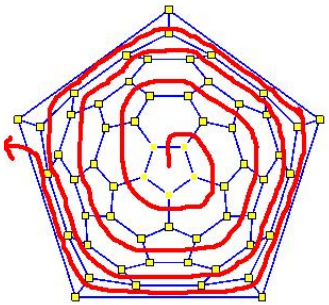
$$5666665656565656565656565666665$$

Manolopolous and Fowler proposed an algorithm to ennumerate fullerene isomers which can be encoded by a spiral code:

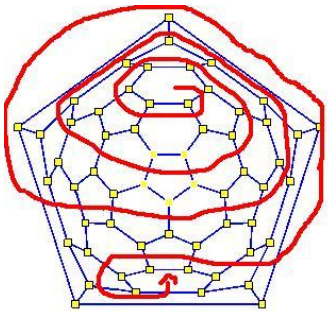
- A fullerene with n faces has $p = 12$ pentagonal faces and $h = \frac{n}{2} + 2$ hexagonal faces.
- So, generate, all possible spiral codes with p 5's and h 6's and test if each encodes a fullerene
- For each spiral code, see if it constructs a fullerene

- A given fullerene could conceivably have upwards of $6n$ spiral codes
- To remove duplicates, we construct all possible spiral codes for our fullerene, and check if our spiral code was the one of smallest value.

Spiral Code Algorithm



code A = 566666565656566565656565666665



code B = 656565666566566566566565656566

code A < code B

We call the smallest spiral code for a given fullerene the **canonical form** for that fullerene.

Constructing random fullernes using spiral codes

Idea: Generate random spiral codes until one such code generates a fullerene.

This algorithm works reasonably well given small numbers of vertices; however...

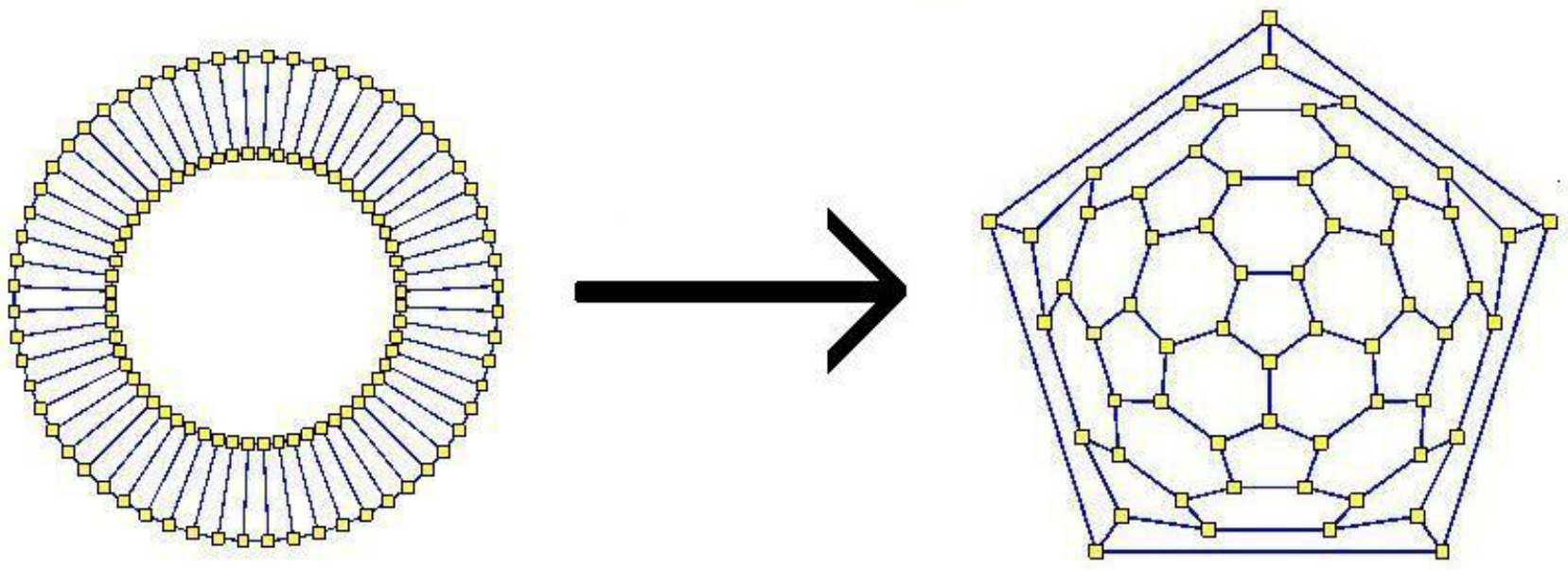
- A fullerene with n vertices has at most $6n$ spiral codes
- There are $O(n^9)$ fullerene isomers with n vertices
- So, there are $O(n^{10})$ spiral codes that may generate a fullerene with n vertices.
- There are $\binom{\frac{n}{2}+2}{12}$ spiral codes which we can test for an n-vertex fullerene.

The proportion of good spiral codes grows exponentially small, so choosing spiral codes at random is a poor idea.

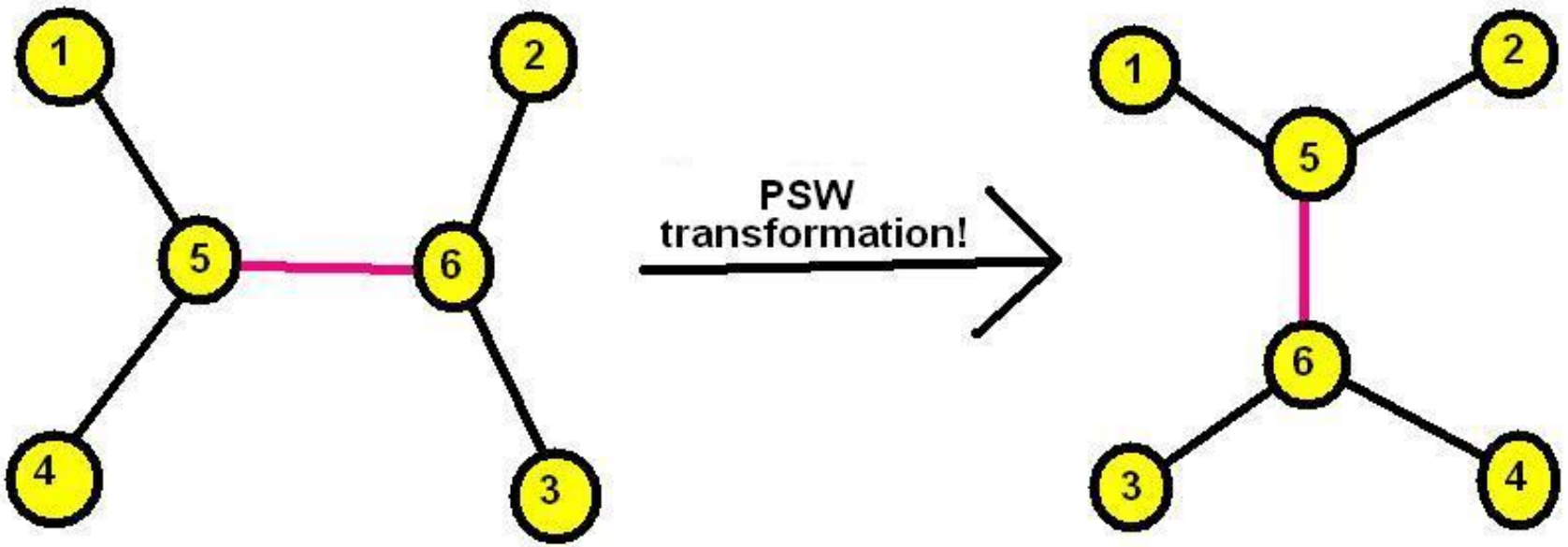
In addition, there exist fullerenes for which there is no spiral code. In such fullerenes, any spiral will reach a dead end before traversing every face.

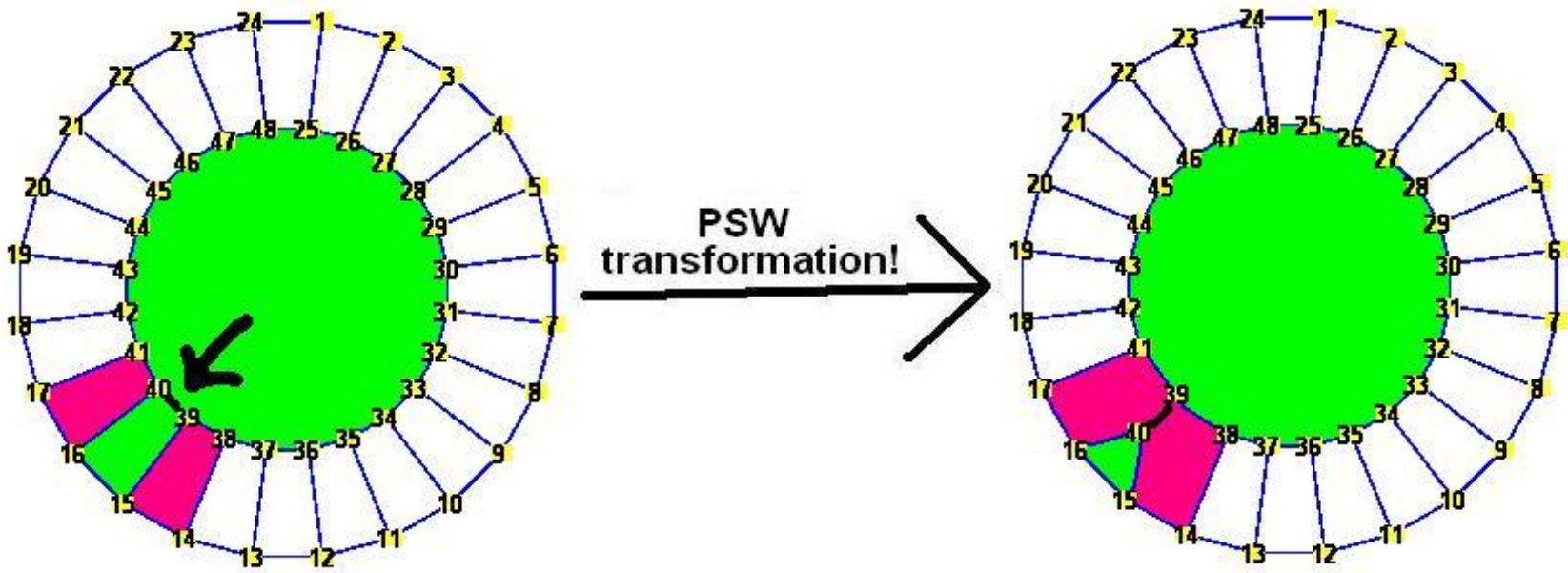
Contents

Plestenjak, Pisanki, and Graovac describe an algorithm that takes a prism and rearranges it into a fullerene graph.



They achieve this by doing transformations on edges.


1
2
5
6
4
3
PSW
transformation!
1
2
5
6
3
4



Notice that the pink faces gain an edge, whereas the green faces lose an edge.

A polyhedron with n vertices has:

- $\frac{3n}{2}$ edges
- $\frac{n}{2} + 2$ faces

So, if we let $\bar{f}$ be the average number of edges in an n-vertex polyhedron, then:
$\bar{f} = \frac{3n}{4 \cdot \left(12 + \frac{20-n}{2}\right)}$

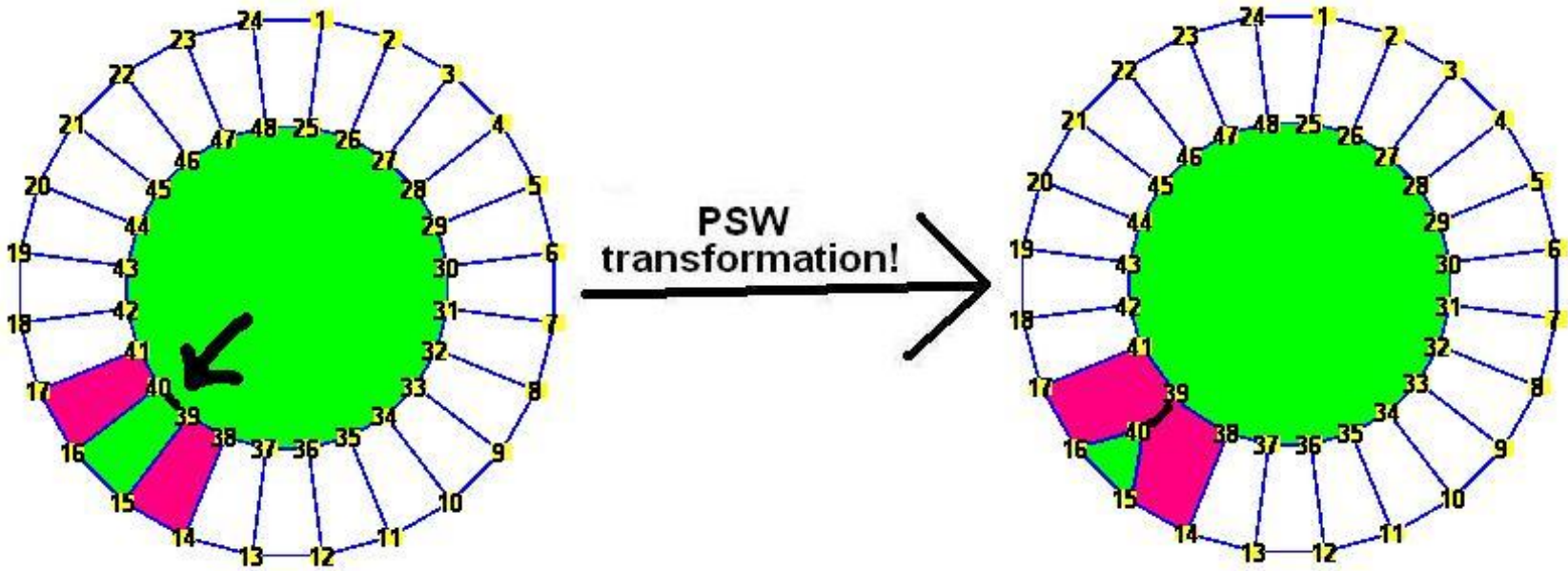
Let f_i denote the number of edges in the i_{th} face of our graph.
We define a function on our graph G:
$E(G) = \left(\sum_{i=1}^{\frac{n}{2}+2}(|f_i - \bar{f}|)\right) - \left(12(\bar{f} - 5) + (\frac{n}{2} - 10)(6 - \bar{f})\right)$
E(G) has two notable properties:

- $E(G) \geq 0$ for any polyhedral graph
- $E(G) = 0$ if and only if G is a fullerene

We want to choose an edge for which the PSW transformation minimizes E(G)! Let be the change in E(G) that would result from a PSW transformation on edge e. We treat $\Delta_e E(G)$ as the edge weight of edge e. We say an edge is a best edge if it has minimal weight.

- if there is a unique best edge with negative weight, we select it.
- if not, we pick a random subset of 5 edges from G, and take the edge of minimal weight from that subset.

Every time we perform a PSW transformation, we have first find a minimum weight edge and then update the edge weights for any edge with a vertex in one of the four affected faces.



One step in our algorithm takes $O(n)$ operations.

- Having implemented the algorithm in Maple, we found that, on average, the number of PSW transformations it took to get a fullerene is roughly linear with respect to the number of vertices.
- The average-case time complexity is roughly $O(n^2)$
- Additionally, there was a large amount of variation in the algorithm running-time.

Future work

Given that the running-time for the algorithm varies greatly, it would be useful if there were a better edge-selection rule. We currently do not know of a fast edge-selection algorithm which will minimize the number of PSW-transformations required to obtain a fullerene graph.

Contents

- A **planar graph** is a graph that can be drawn so that no edges intersect in the plane.
- A **plane graph** is a planar graph already drawn in the plane without edge intersections.
- A graph G is said to be **3-connected** if there does not exist a set of 2 vertices whose removal disconnects the graph.

Contents

the method

RubberDraw is a plane drawing method for 3-connected planar graphs.

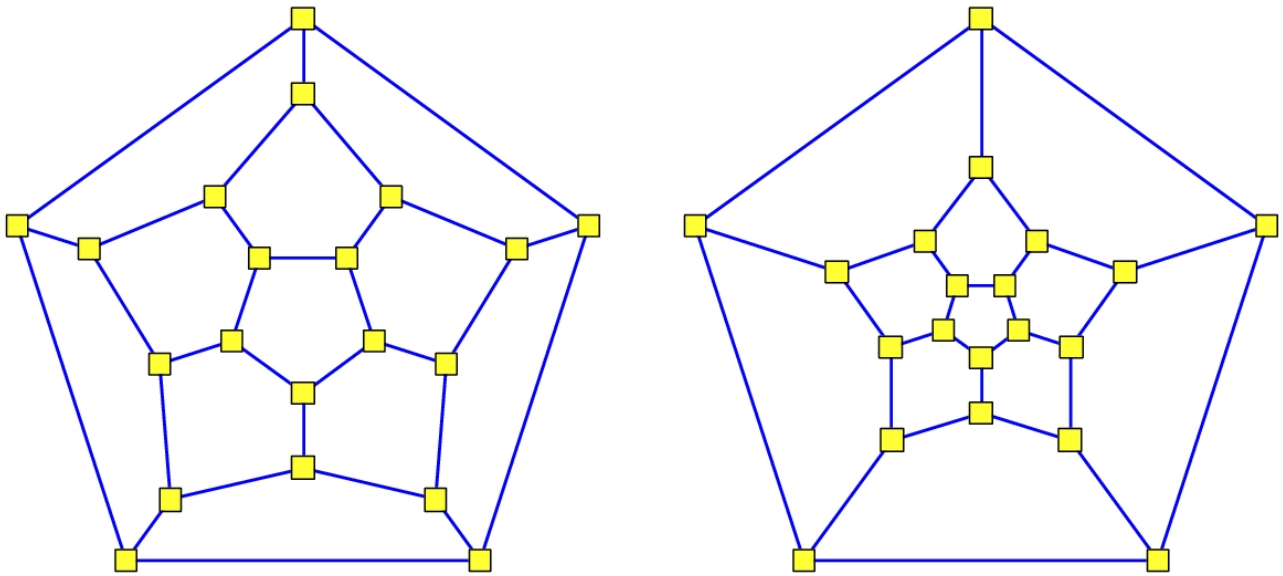


Figure: G:=DodecahedronGraph(); RubberDraw(G);

the rescaling problem

We have a cluster of vertices at the center of the graph.

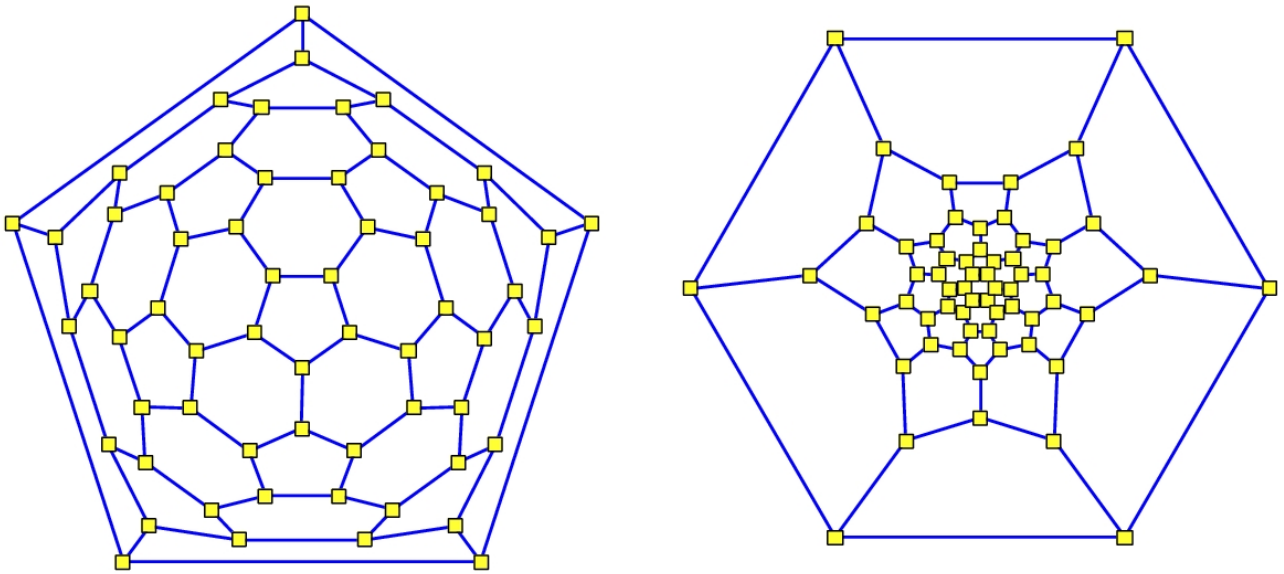


Figure: S:=SoccerBallGraph(); RubberDraw(S);

Contents

A Good Drawing

The largest cycle of the GrinbergGraph is a enneagon and has 9 edges connected to the inner vertices.

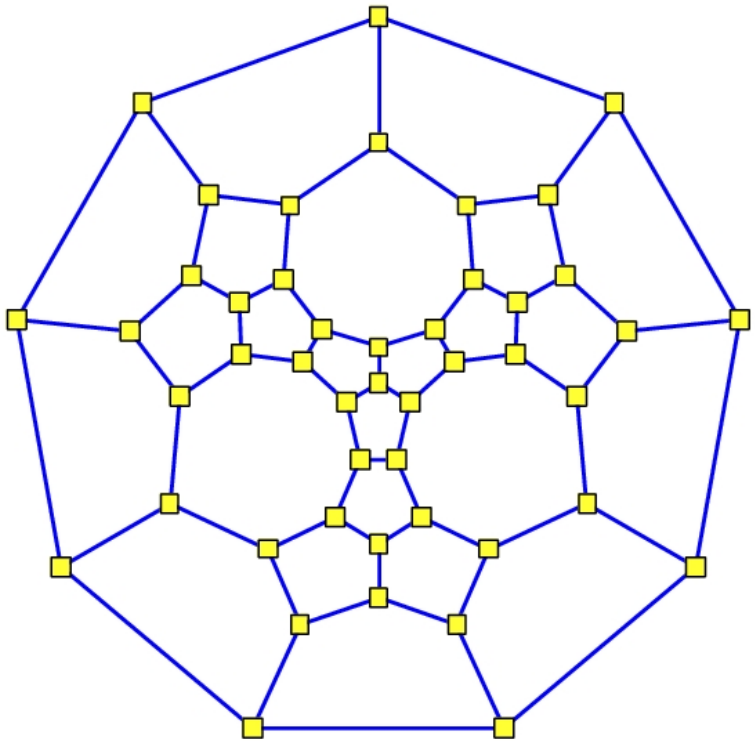
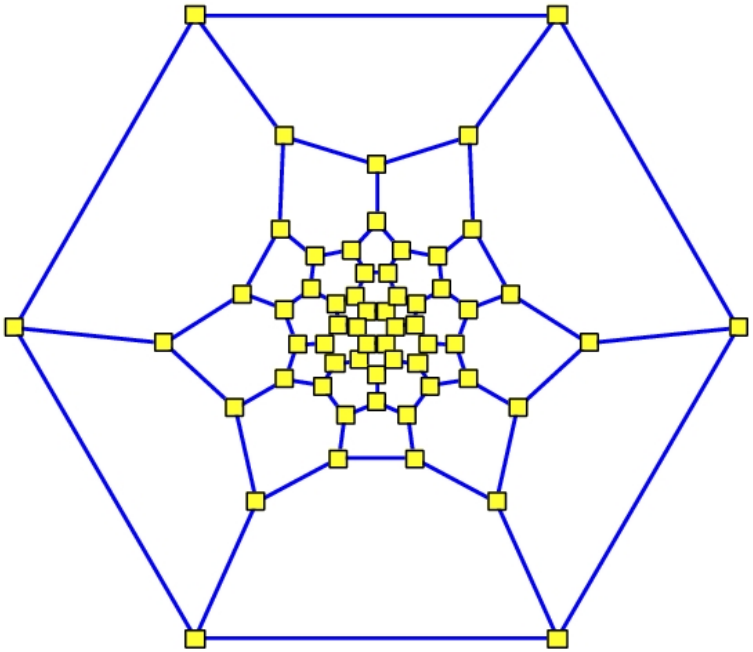


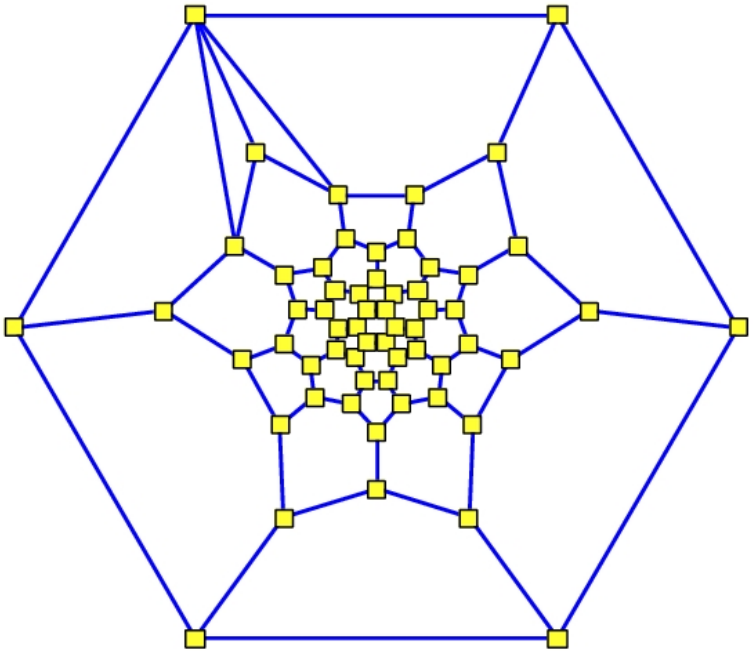
Figure: G:=GrinbergGraph(); RubberDraw(G);

The First Trial



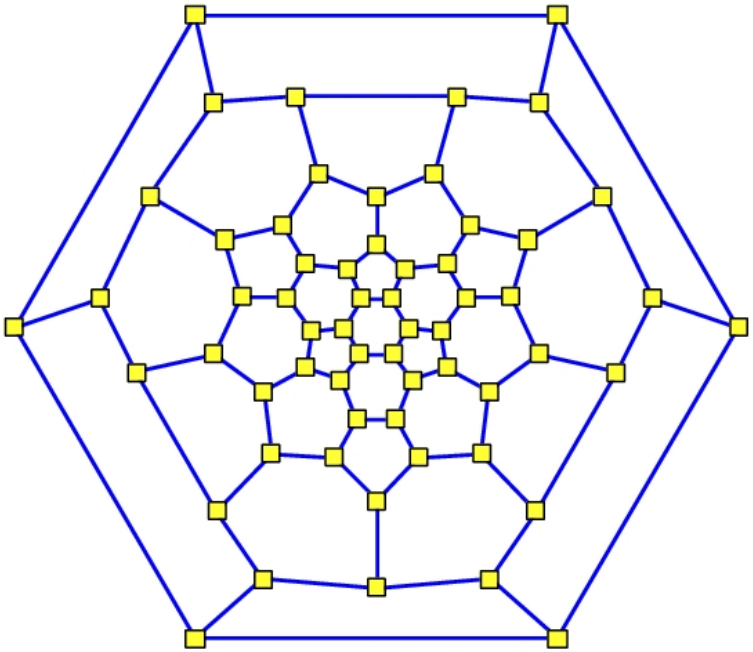
this is the RubberDraw of the SoccerBallGraph

The First Trial



add the additional edges

The First Trial

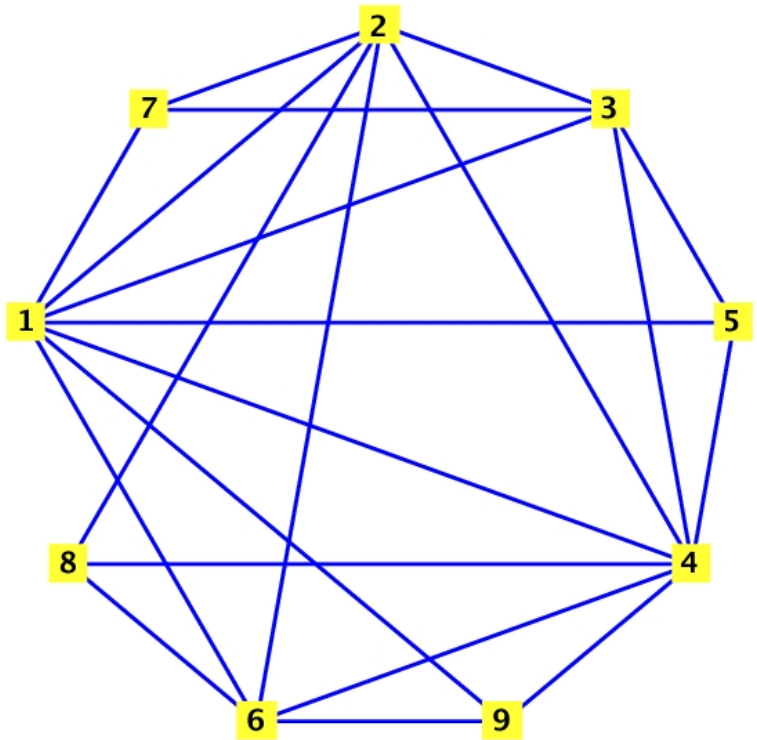


the resulting graph, additional edges have been removed

Random Planar Triangulations

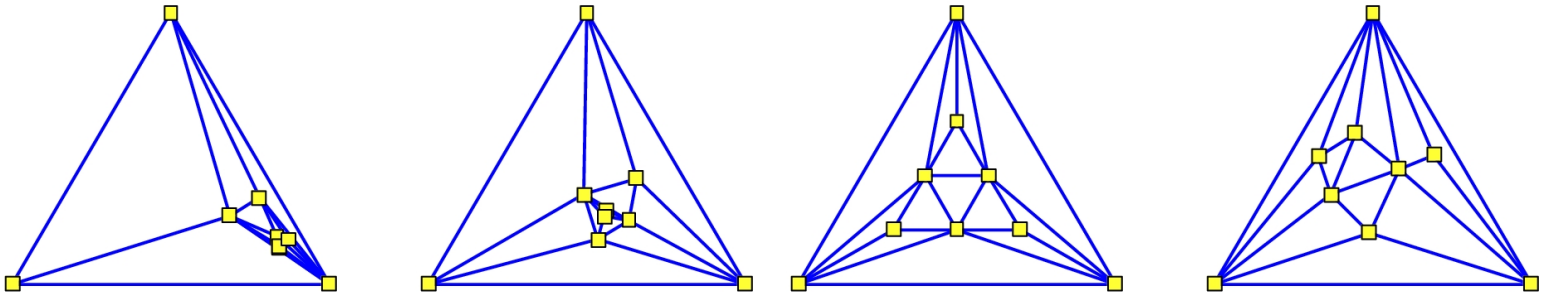
A **planar triangulation** is a planar graph whose faces are all triangular.

G:= RandomTriangulation(9); DrawGraph(G);



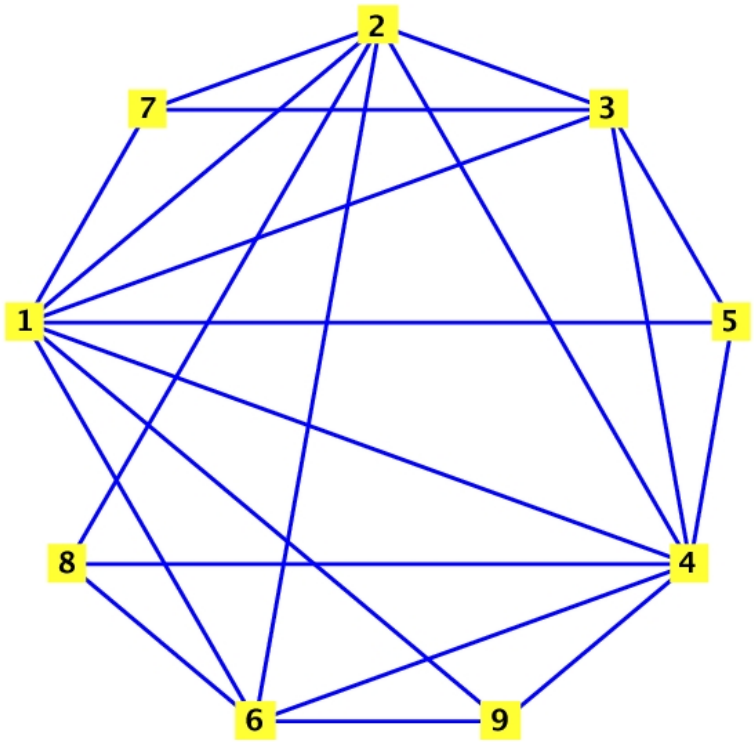
Plane Drawing of Planar Triangulations

Most of the outcoming graphs from RubberDraw fall into the following four patterns.



Contents

Choosing the Right Outer Face



Let F be the list of faces of the planar triangulation graph G,
F := [[3, 2, 7], [7, 2, 1], [4, 3, 5], [4, 2, 3], [8, 4, 6], [4, 9, 6], [6, 2, 8], [5, 3, 1], [1, 9, 4], [4, 5, 1], [2, 4, 8], [1, 3, 7], [6, 9, 1], [1, 2, 6]]

Choosing the Right Outer Face

- dsof := [[5, 6, 3], [3, 6, 7], [7, 5, 3], [7, 6, 5], [3, 7, 5], [7, 3, 5], [5, 6, 3], [3, 5, 7], [7, 3, 7], [7, 3, 7], [6, 7, 3], [7, 5, 3], [5, 3, 7], [7, 6, 5]]
- mdev := [1.111111111, 1.555555556, 1.333333333, .6666666667, 1.333333333, 1.333333333, 1.111111111, 1.333333333, 1.777777778, 1.777777778, 1.555555556, 1.333333333, 1.333333333, .6666666667]
- dssum := [14, 16, 15, 18, 15, 15, 14, 15, 17, 17, 16, 15, 15, 18]

Choosing the Right Outer Face

Take the ratio:

$$\frac{mdev[i] + NumberOfVertices(G)}{dssum[i]}$$

dv := [1.507936508, 1.347222222, 1.422222222, 1.148148148, 1.422222222, 1.422222222, 1.507936508, 1.422222222, 1.281045752, 1.281045752, 1.347222222, 1.422222222, 1.422222222, 1.148148148]

Choosing the Right Outer Face

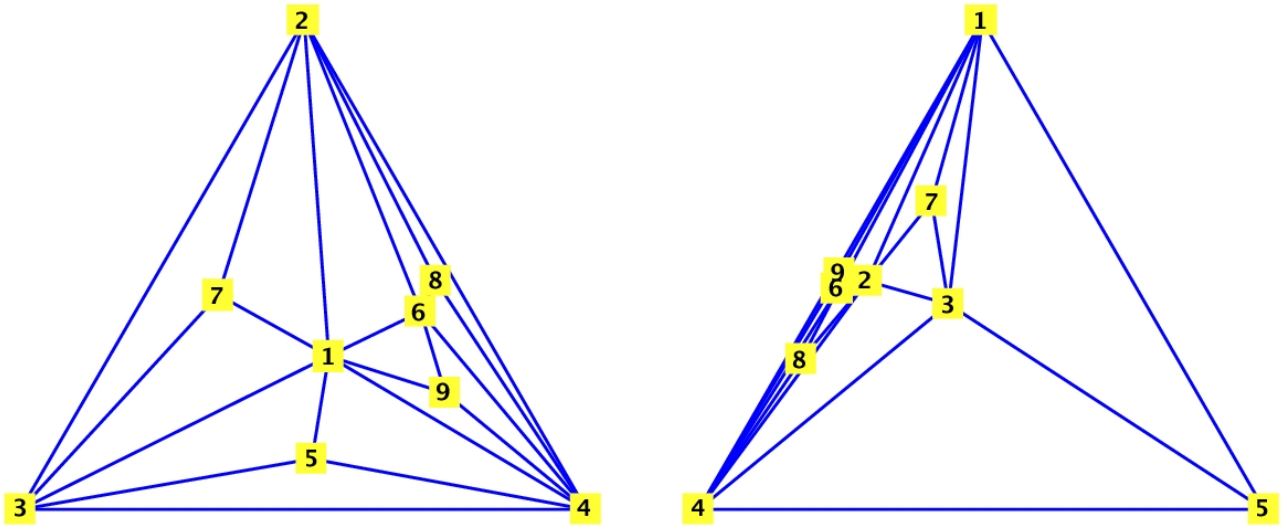
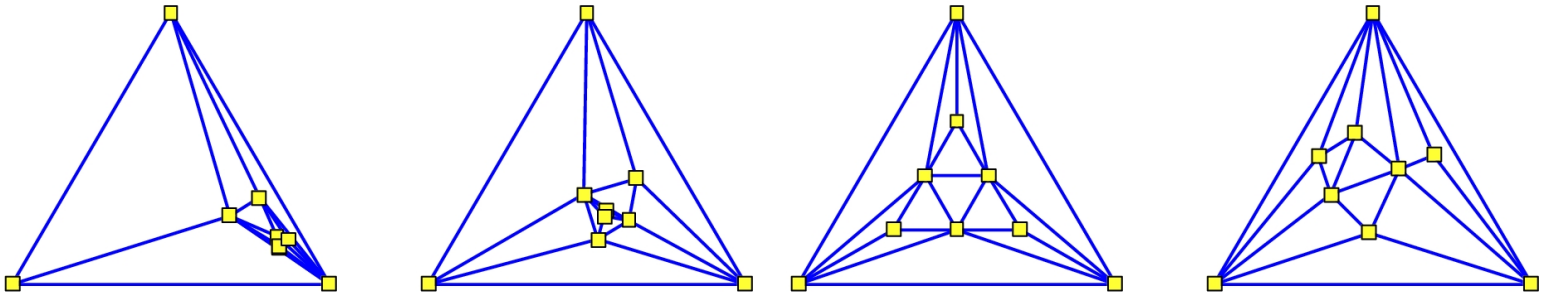
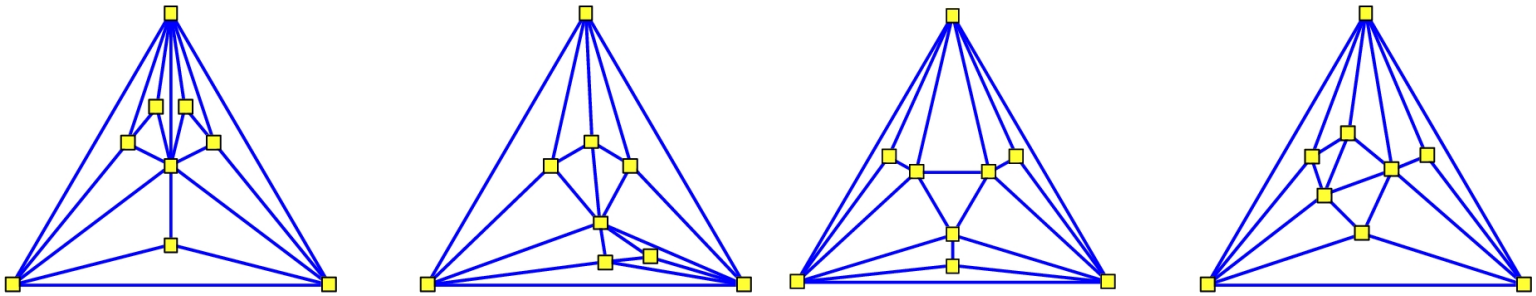


Figure: The updated RubberDraw picks a better outer face.

Improved Drawing of Planar Triangulations



Improved Drawing of Planar Triangulations

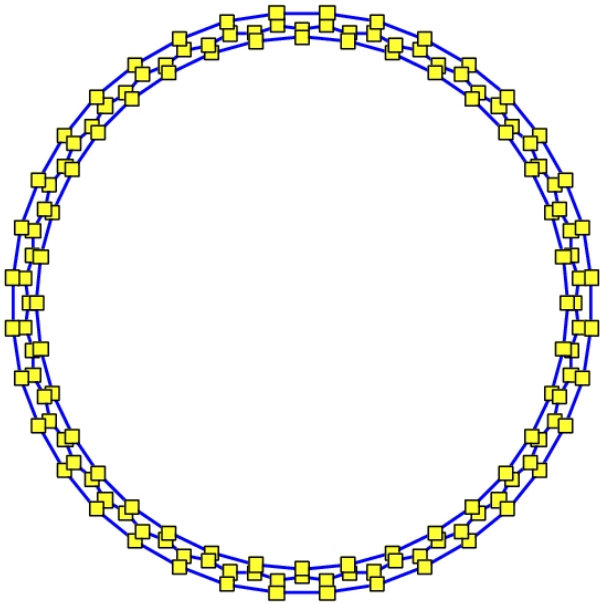


the same four graphs redrawn by the updated RubberDraw

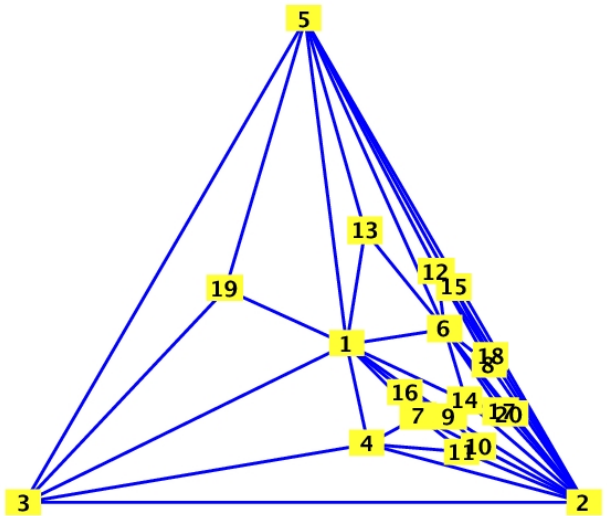
Contents

Not all problems are solved!

GeneralizedPetersenGraph(72, 2);



Graphs with more vertices



The Last Experiment

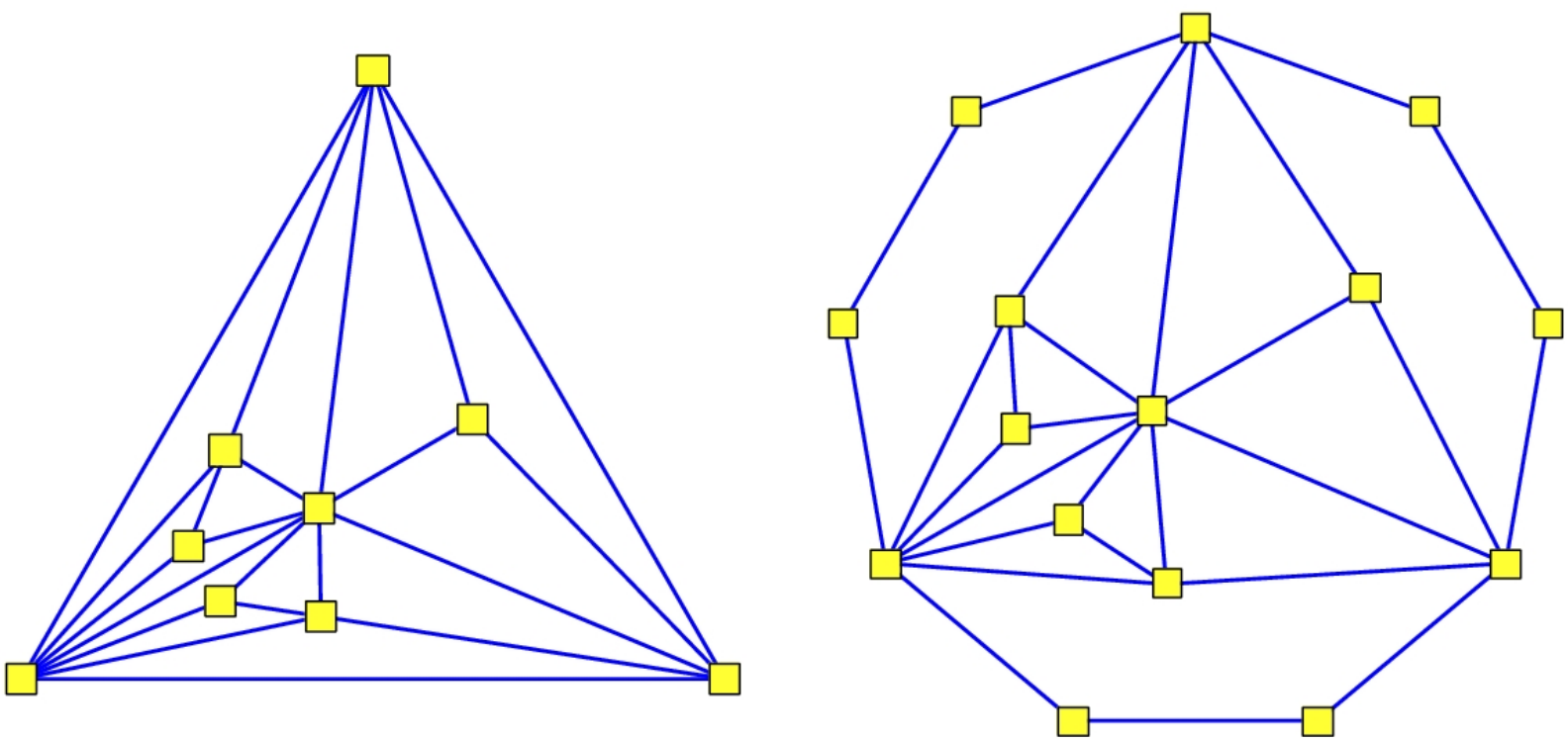


Figure: the last experiment

Contents

Problem Description

Graph Isomorphism

A graph isomorphism is a bijection (a one-to-one and onto mapping) between the vertices of two graphs G and H $f : V(G) \rightarrow V(H)$ with the property that any two vertices u and v from G are adjacent if and only if f(u) and f(v) are adjacent in H.

The Problem

The graph isomorphism problem is whether two graphs are isomorphic (there exists a graph isomorphism between them) or not. This problem is characterized as NP, but it is not know yet if it belongs in the polynomial time or the NP-complete class.

Definitions

The **Adjacency Matrix** of a finite directed or undirected graph G on n vertices is the n x n matrix where A_{ij} is the number of edges from vertex i to vertex j. In the cases we are interested, the graphs have no loops, so all the A_{ii} are 0.
The **Degree** of a vertex is the number of edges incident to the vertex.
The **Degree Matrix** is a diagonal matrix, where the entry $D_{i,i}$ is the degree of the vertex i.
The **Laplacian Matrix** is given by D-A and the **Absolute Laplacian Matrix** is given by D+A.

Contents

Backtracking

The backtracking method gives a quaranteed correct answer to the question whether 2 graphs are isomorphic or not.

It is a brute force method, which makes it really slow.

Partition

Backtracking is based on the initial partition of the vertices, which is done by using the degree of each vertex.

Table 1 : Results from the partitioning method

N	Total	Partition	Non Isomorphic	Connected	Bipartite	Tree
1	1	1	1	1	1	1
2	2	2	2	1	2	1
3	8	4	4	2	3	1
4	64	16	11	6	7	2
5	1024	84	34	21	13	3
6	32768	936	156	112	35	6
7	2097152	16758	1044	853	88	11

However, this can be improved by using a dynamic partition table. (Future work)

Contents

Already Used

- Adjacency Matrix A
- Laplacian Matrix L = D-A
- Absolute Laplacian Matrix L = D + A

Already Used

- Adjacency Matrix A
- Laplacian Matrix L = D-A
- Absolute Laplacian Matrix L = D + A

Already Used

- Adjacency Matrix A
- Laplacian Matrix L = D-A
- Absolute Laplacian Matrix L = D + A

New ones

- $E = A + D^2$
- $M = L \bullet L' \quad , \quad L = A + D \quad , \quad L' = 2A + D$
- $M = L \bullet L' \quad , \quad L = A + D^2 \quad , \quad L' = 2A + D^2$

New ones

- $E = A + D^2$
- $M = L \bullet L' \quad , \quad L = A + D \quad , \quad L' = 2A + D$
- $M = L \bullet L' \quad , \quad L = A + D^2 \quad , \quad L' = 2A + D^2$

New ones

- $E = A + D^2$
- $M = L \bullet L' \quad , \quad L = A + D \quad , \quad L' = 2A + D$
- $M = L \bullet L' \quad , \quad L = A + D^2 \quad , \quad L' = 2A + D^2$

Similar Matrices

- Rank
- Determinant
- Trace
- Eigenvalues
- Characteristic Polynomial
- Minimal Polynomial

More new ones

A' = All Pairs Distance Matrix

- $A' + D$
- $A' + D^2$
- $M = L \bullet L'$, $L = A' + D^2$, $L' = 2A' + D^2$

More new ones

A' = All Pairs Distance Matrix

- $A' + D$
- $A' + D^2$
- $M = L \bullet L'$, $L = A' + D^2$, $L' = 2A' + D^2$

More new ones

A' = All Pairs Distance Matrix

- $A' + D$
- $A' + D^2$
- $M = L \bullet L' \quad , \quad L = A' + D^2 \quad , \quad L' = 2A' + D^2$

Contents

Table 2 : Number of graphs with cospectral mates

Vertices	Non Isomorphic	A	A+$\bar{A}$	L	$\lvert L \rvert$	E	M
2	2	0	0	0	0	0	0
3	4	0	0	0	0	0	0
4	11	0	0	0	2	0	0
5	34	2	0	0	4	0	0
6	156	10	0	4	16	0	0
7	1.044	110	40	130	102	0	0
8	12.346	1.722	1.166	1.767	1.201	0	0
9	274.668	51.038	43.811	42.595	19.001	0	0
10	12.005.168	2.560.516	2.418.152	1.412.438	636.607		

Table 3 : Graphs on 10 vertices with non-isomorphic cospectral mate

Edges	Non Isomorphic	A	A+$\bar{A}$	L	$\|L\|$	E	M	M'
2	2	0	0	0	0	0	0	0
3	5	0	0	0	2	0	0	0
4	11	4	0	0	2	0	0	0
5	26	5	0	0	4	0	0	0
6	66	26	2	7	11	0	0	0
7	165	62	6	21	31	0	0	0
8	428	191	22	75	80	0	0	0
9	1.103	412	86	237	155	0	0	0
10	2.769	1.068	278	568	338	0	0	0
11	6.579	1.994	831	1.279	681	0	0	0
12	15.772	4.843	2.178	2.722	1.307	0	0	0
13	34.663	8.874	5.380	5.455	2.344	0	0	0
14	71.318	18.747	11.811	10.428	4.362	8	2	0
15	136.433	31.852	24.094	18.826	8.069	26	14	0
16	241.577	56.827	44.229	31.373	13.909	58	52	0
17	395.166	87.986	75.358	47.972	21.814	166	150	
18	596.191	133.350	116.870	68.692	31.495	364	332	
19	828.728	181.236	166.403	92.350	42.534	624	600	
20	1.061.159	233.250	217.639	119.163	54.427	987	931	
21	1.251.389	273.336	260.561	145.233	65.430		1243	
22	1.358.852	294.399	283.328	161.818	72.165		1424	
23	1.358.852	291.391	283.328	161.818	72.181		1424	

Thank you