communications in mathematical
match
and in computer chemistry

no. 38, October 1998

ISSN 0340-6253

MATCDY (38) 99-109 (1998)

# Evolutionary Algorithms for Cluster Geometry*

Drago Bokal†
Institute of Mathematics, Physics and Mechanics,
University of Ljubljana,
Ljubljana, Slovenia

**Abstract**

The paper presents a genetic algorithm for maximizing the Isoperimetric Quotient of polyhedral clusters. This algorithm helped extending the definition of Isoperimetric Quotient to clusters with nonplanar faces. Detailed description of the genetic algorithm is followed by the performance analysis. Some other observations of genetic algorithm's behavior are also presented.

## Introduction

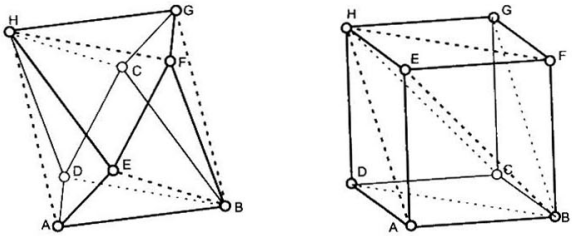
Evolutionary algorithms are becoming widely used in different fields of technology and science [5, 6]. They are inspired by modeling natural selection, which gives them certain properties not common in classic algorithmic approach. One of these properties is the fact that they can adapt to the problem they are solving and this can help discovering some unpredicted points of view of that problem.

This paper describes how one can use evolutionary algorithm to determine the position of cluster's vertices in 3D space. The idea is to define a *fitness function*, which describes the acceptability of particular arrangement of points and then search for the optimum of this function. This approach was already used by some graph-drawing algorithms, for example NiceGraph algorithm [10] that minimizes the strain in the bonds among vertices using simulated annealing. In the experiment described in this paper, the so-called Polyhedral Isoperimetric Quotient [2, 8, 11] was used as fitness function and genetic algorithm was used to find its maximum.

Polyhedral Isoperimetric Quotient is basically a measure for how spherical a given polyhedron is. It is a dimensionless quantity which ranges from 0 for flat, planar objects to 1 which is obtained only for sphere. A detailed definition is given later.

*All correspondence regarding this paper should be sent to e-mail drago.bokal@fmf.uni-lj.si
†Currently student of mathemathics at University of Ljubljana.

Figure 1: This coordinate placement for a cube was obtained by genetic algorithm that maximized the IQ of a cube using arbitrary triangulation for calculating volume. The computed structure is actually a hexagonal bipyramid. Regular cube with added edges (dotted lines) is displayed for comparison.




## Definition of Polyhedral Isoperimetric Quotient

Polya [11] introduced Isoperimetric Quotient (IQ) as a measure of how spherical a given polyhedron $M$ is. IQ is defined as a normalized ratio of the square of polyhedron's volume and the cube of its surface:

$$IQ(M) := 36\pi \frac{V(M)^2}{S(M)^3}$$

More about IQ can be read in [2, 8]

The upper definition is suitable for polyhedra with planar faces. In such cases surface and volume of the polyhedron can easily be calculated using arbitrary triangulation of faces. When extending the definition of IQ to polyhedral clusters with (possibly) nonplanar faces, special care has to be taken. Using triangulation as in the case of planar faces actually introduces new edges and privileges certain vertices. When maximizing IQ the newly introduced edges give the GA more freedom for choosing cluster's geometry and the algorithm takes advantage of these new edges. In the case of the cube the coordinate placement shown on figure 1 was obtained. After adding the invisible edges (dotted lines) this structure becomes a hexagonal bipyramid (its top vertices on figure 1 are B and H). Its IQ is 0.69813 and is bigger than maximal possible IQ for cube [2], which is $\frac{\pi}{6} \approx 0.52360$. A solution to this problem was proposed in [8]. A new vertex is introduced in the barycenter of each face of the polyhedron $M$. The face is triangulated by connecting the barycenter with all the vertices of that face. Doing this for all faces of $M$ the so-called *two-dimensional subdivision* $S_2(M)$ is obtained. It is also known as *omnicapping*. The IQ of polyhedral cluster $M$ with nonplanar faces is then defined as the IQ of $S_2(M)$. This method is stable, since $IQ(S_2(S_2(M))) = IQ(S_2(M))$ and it is aligned with original definition, since if $M$ has all faces planar we have $IQ(M) = IQ(S_2(M))$. Experiments also show that maximizing the IQ of clusters using this definition produces (almost) planar faces, although it sometimes contracts vertices (discussed later).

# Description of the Genetic Algorithm

Here a generic description of *genetic algorithm* (GA) is given. The terms *individual* and *fitness function* are discussed first, since they connect the generic description with actual implementation. Any type of individuals could be used for GA, as long as the operators *crossover* and *mutation* are adjusted so that they fit the data type of that individual. Also, any *fitness function* can be maximized (or with slight modifications, minimized) using this algorithm. More about genetic algorithms can be found for example in [1, 5, 6].
Solutions to the problem of maximizing cluster's IQ are represented by positions of its coordinates in the 3D space. Thus each *individual* in the GA's population will represent one placement of cluster's points in space. Each point has three coordinates, thus the solution space for cluster with $n$ points is $3n$-dimensional. The connectivity of the points is common to all individuals in a single run of GA, therefore it can be stored globally for the whole population.
The *fitness function* used in the GA was the IQ of the given individual. The purpose of the algorithm was to find the coordinate placement with maximal IQ, therefore algorithm had to maximize the fitness function.
At the beginning of a single run of GA, a population of $N$ individuals is created. Usually the individuals are created randomly, but they could also be generated in some more sophisticated way or they could also be read from a file.
The individuals in the population are then sorted according to the value of the fitness function. If the fitness function is being maximized, they are sorted in descending order and in the case of minimization in ascending order. Using this order, the population is divided into *good* and *bad* part. The good part is the first $k$ individuals of the population. In each generation the bad part of the population is discarded and individuals from the good part are randomly selected for recombination with crossover to form new generation. This sort of selection classifies the described GA among *elitist GA* with *ranking selection* [5]. Elitist GA means that some of the best individuals are preserved among generation changes (the other possibility is to create whole population with recombination) and ranking selection means that the probability of choosing an individual for selection for crossover depends on its rank in the current population, not on the actual value of the fitness function. Other types of GA and other selection methods are described in [5, 6].
There are many possible crossover strategies [5, 6]. Three of them, which fit best to the problem of IQ maximization, are described here. The first is *arithmetical crossover*. By this method the coordinates of the new individual are calculated as the average of corresponding parents' coordinates:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $1^{st}$ parent chromosome: | $a_1[x]$ | $a_1[y]$ | $a_1[z]$ | $a_2[x]$ | $a_2[y]$ | $a_2[z]$ | ... | $a_n[x]$ | $a_n[y]$ | $a_n[z]$ |
| $2^{nd}$ parent chromosome: | $b_1[x]$ | $b_1[y]$ | $b_1[z]$ | $b_2[x]$ | $b_2[y]$ | $b_2[z]$ | ... | $b_n[x]$ | $b_n[y]$ | $b_n[z]$ |
| child chromosome: | $c_1[x]$ | $c_1[y]$ | $c_1[z]$ | $c_2[x]$ | $c_2[y]$ | $c_2[z]$ | ... | $c_n[x]$ | $c_n[y]$ | $c_n[z]$ |

Arithmetical crossover algorithm:

**begin**
  **for** $i := 1$ to $n$ do **begin**
    $c_i[x] := \frac{a_i[x]+b_i[x]}{2}$
    $c_i[y] := \frac{a_i[y]+b_i[y]}{2}$

```
    c_i[z] := (a_i[z]+b_i[z])/2
  end for
end
```

This type of crossover is natural in the cases, where individuals are vectors of real numbers. Such cases often arise in geometric problems.
The other suitable crossover strategy is *single point crossover*. It produces new individuals by breaking parent's coordinate vector at certain point and takes one part of child's vector from the first and the other from the second parent. Let $a_i$, $b_i$ and $c_i$ be the points from the first parent, the second parent and the child, respectively. Single point crossover is then given by the following scheme:

Single point crossover algorithm:

```
begin
  k:=random(n)+1;
  for i := 1 to n do begin
    if i < k then begin
      c_i[x] := a_i[x]; c_i[y] := a_i[y]; c_i[z] := a_i[z];
    end if
    else begin
      c_i[x] := b_i[x]; c_i[y] := b_i[y]; c_i[z] := b_i[z];
    end else
  end for
end
```

*Uniform crossover* is the third suitable crossover strategy. When producing new individual, it randomly takes each coordinate from one of the parents. Again, let $a_i$, $b_i$ and $c_i$ be the points from the first parent, the second parent and the child, respectively and let $p$ be the probability that a child's point will be taken from the first parent. The following scheme then describes uniform crossover strategy:

Uniform crossover algorithm:

```
begin
  for i := 1 to n do begin
    k:=random(1)
    if k < p then begin
      c_i[x] := a_i[x]; c_i[y] := a_i[y]; c_i[z] := a_i[z];
    end if
    else begin
      c_i[x] := b_i[x]; c_i[y] := b_i[y]; c_i[z] := b_i[z];
    end else
  end for
end
```

Among these three crossover types the arithmetical crossover performed best (it obtained highest maxima of the fitness function - IQ) and was therefore chosen for application.

Table 1: The following table lists all the parameters of the GA described in this paper

| Symbol | Value | Meaning |
|---|---|---|
| $N$ | 100 | Size of the population |
| $g$ | 30 | Size of the good part of the population |
| $N-g$ | 70 | Size of the bad part of the population |
| $p$ | 0.01 | Mutation probability |
| $d$ | variable | Mutation size |
| $d_0$ | 0.1 | Initial mutation size |
| $q$ | 0.988 | Mutation size decreasing factor - $d := qd$ if there was no improvement |
| $r$ | 0.97 | Mutation size increasing factor - $d := d + r\delta$ if $\delta$ is the improvement |
| $s$ | 1000 | Algorithm stops after $s$ generations of no improvement |
| $T$ | 5 | Number of GA threads |
| $d_{min}$ | 0.0001 | Minimal mutation size, reached by single GA thread |
| $c$ | 100 | Number of generations between two competitions among threads |

After an individual is created it undergoes the *mutation* operator. Coordinates of some of its points are randomly moved in any direction for a small amount ranging from 0 (no change) to number $\frac{d}{2}$ using the following formula:

$$x = x + (d * (0.5 - random(1)))$$
$$y = y + (d * (0.5 - random(1)))$$
$$z = z + (d * (0.5 - random(1)))$$

The parameter $d$ is not constant all the time; it is alerted at each generation. If there was no improvement in the value of the fitness function of the best individual in the last generation, new $d$ is decreased with the formula

$$d := qd$$

for some $q$ slightly less than 1. But if new best individual is discovered in the generation, $d$ is increased using the formula

$$d := d + r\delta$$

where $\delta$ means the improvement of the best value of fitness function from the previous generation. The constant $r$ should be chosen so that $d$ does not get too high at early stages of evolution when there is a lot of improvement.

This strategy introduces some aspects of simulated annealing [1, 5] into GA. Decreasing $d$ helps in fine-tuning the solutions at later stages of GA, when large mutations move the individual far from the discovered optimum. On the other side $d$ is increased each time a better solution is found to help the GA explore the promising neighborhood of that solution. Of course this slows down the convergence of GA, but the goal was not to find good solutions fast, but to take more time and find as good solutions as possible.

After all the bad part of the population is replaced by new individuals, the population is sorted again and the cycle is repeated. The algorithm is stopped when there is no improvement of the best individual in previous $s$ generations.

When the algorithm was run with different random seed, slightly different local optima was obtained (up to 1% difference in IQ). In order to obtain the highest possible local

optimum, algorithm has to be ran many times and the best individual of all the runs should be selected as the optimum.

Different GA runs can be ran parallel. In this case they are named *threads*. There is an advantage of running different threads: at late stages, when the GA almost converges (and just the fine-tuning of the optimum is being done) one can compare the approximate optima discovered by different threads and discard those that are not promising the best solution. Thus the processor time for fine-tuning is spent just for the best thread.

The value of mutation amount $d$ was considered to be the upper bound for fitness function improvement of a thread. (This is obviously not true at the early stages of the evolution, but when the thread almost converges to an local maximum, it was empirically shown as true, at least in the case of IQ.) After all the threads converge to certain optimum and $d$ falls under some predefined value $d_{min}$, the threads are compared every $c$ generations. Let $m$ be the maximal value of the fitness function in population of all the threads, $i$ be the maximal value of fitness function of population in particular thread and $d$ the mutation amount of the same thread. If the condition

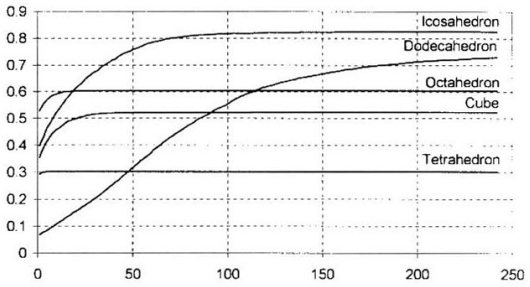
$$m - i > d$$

holds for this thread, the thread is discarded, since it (most probably) won't converge to a local optimum, higher than that of the thread with currently maximal value of fitness function.

An pseudo-code implementation of the described GA is shown in the following listing:

```
begin
    read initial data
    for each thread do begin
      randomly generate thread's population of N individuals
      sort the population by decreasing value of the fitness function
      d[thread] := d_0
      max[thread] := maximal value of the fitness function in thread's population
    end for
    generation:=0
    repeat
      generation:=generation+1
      for each thread do begin
        discard worst N - g individuals
        repeat
          randomly select two individuals P_1 and P_2 from the best g individuals
          i :=crossover(P_1, P_2)
          mutate points of the individual i with probability p
          add i to the population
        until N - k individuals are created
        sort the population
        max' := maximal value of the fitness function in new population
        δ := max' - max[thread] calculate the improvement
        max[thread] := max'
        if δ == 0 then d[thread] := q · d[thread]
        else d[thread] := d[thread] + rδ
```

Figure 2: Maximal IQ in the GA's population versus time for platonic polyhedra.




**end** *for*
**if** generation **mod** $c == 0$ **then begin**
$m :=$ best value of fitness function obtained by the best thread
**for** each thread **do**
**if** ($d$[thread] $< d_{min}$) **and** ($m -$ max[thread] $> d$[thread]) **then**
discard this thread
**end** *if*
**until** no improvement in last $s$ generations
result:=best value of the fitness function obtained by the best thread
store the individual with best value of the fitness function
**end**

Table 2: Testing performance of genetic algorithm on platonic polyhedra

| Name | Max IQ | % of max. IQ after $k$ gen. | | | gen. to reach $k$% of max IQ | | |
|---|---|---|---|---|---|---|---|
| | | 10 g. | 50 g. | 250 g. | 95% | 97.5% | 100% |
| Tetrahedron | 0.302 | 100% | 100% | 100% | 1 g. | 2 g. | 6 g. |
| Octahedron | 0.605* | 97.6% | 99.8% | 100% | 6 g. | 10 g. | 588 g. |
| Cube | 0.524 | 87.3% | 99.5% | 99.8% | 20 g. | 27 g. | 720 g. |
| Icosahedron | 0.829* | 63.1% | 91.4% | 99.7% | 63 g. | 80 g. | 1010 g. |
| Dodecahedron | 0.755 | 14.3% | 42.0% | 96.9% | 208 g. | 268 g. | 1512 g. |

*For Octahedron and icosahedron the upper limit given in [2] is not an exact upper limit. Since for other platonic polyhedra the GA converges to the global optimum, which is obtained when the vertices are placed in the shape of regular platonic polyhedron, the known IQ of regular platonic polyhedron is assumed as the exact upper limit also for octahedron and icosahedron.

Table 3: Results of GA's performance test on trivalent polyhedral clusters. Max IQ is the IQ to which the GA's population converged.

| Name | Max IQ | % of max. IQ after $k$ gen. | | | gen. to reach $k$% of max IQ | | |
|---|---|---|---|---|---|---|---|
| | | 10 g. | 50 g. | 250 g. | 95% | 97.5% | 100% |
| TPC 70 | 0.6690 | 47.5% | 86.2% | 99.4% | 75 g. | 91 g. | 1333 g. |
| TPC 72 | 0.6302 | 41.2% | 75.2% | 97.9% | 124 g. | 212 g. | 2161 g. |
| TPC 55 | 0.6509 | 37.3% | 69.5% | 98.5% | 131 g. | 171 g. | 2647 g. |
| TPC 42 | 0.6403 | 38.4% | 74.3% | 98.0% | 132 g. | 202 g. | 2025 g. |
| TPC 31 | 0.6354 | 32.2% | 75.0% | 95.8% | 187 g. | 510 g. | 2084 g. |
| TPC 45 | 0.5500 | 42.9% | 74.0% | 95.7% | 223 g. | 525 g. | 2876 g. |
| TPC 61 | 0.6209 | 39.2% | 75.1% | 94.9% | 256 g. | 564 g. | 2136 g. |
| TPC 58 | 0.5602 | 41.1% | 77.4% | 94.5% | 276 g. | 627 g. | 2143 g. |
| TPC 47 | 0.5420 | 41.4% | 73.6% | 93.3% | 380 g. | 594 g. | 2992 g. |
| TPC 48 | 0.5654 | 30.3% | 69.7% | 93.0% | 410 g. | 712 g. | 3349 g. |
| TPC 50 | 0.5547 | 43.0% | 74.8% | 90.6% | 501 g. | 758 g. | 2932 g. |
| TPC 30 | 0.5406 | 47.0% | 76.0% | 91.0% | 783 g. | 1172 g. | 2729 g. |

## Results and Observations

The algorithm described above was used to maximize the IQ of platonic polyhedra and some trivalent polyhedral clusters. With running GA on platonic polyhedra the performance of the algorithm was most objectively tested, since exact upper bound of IQ for some platonic polyhedra is known [2]. The results are displayed in table 2. First the exact upper limit for the IQ is given and then the percentage of that maximal IQ that was on average obtained at 10, 50 and 250 generations of running IQ. The next three columns give the number of generations that had to be passed so that average IQ reached 95%, 97.5% and 100% of the maximal IQ. On figure 2 the chart of IQ versus time for platonic polyhedra is shown.

Table 2 shows that the GA converged to the global optimum for all platonic polyhedra. Of course more complex polyhedra take more time for GA to converge to the optimum. The fastest is convergence for Tetrahedron, followed by Octahedron, Cube, Icosahedron and Dodecahedron. The same order for polyhedra is obtained with sorting them by number of vertices. Therefore it may be concluded that the complexity for maximizing IQ depends mostly on the number of vertices. This can be explained observing that the dimension of the search space grows linearly with number of vertices.

Trivalent polyhedral clusters (TPC) [4] were used to analyze performance of GA among clusters with the same number of vertices. All of the chosen TPCs have 14 vertices, 21 edges and 9 faces, just the connectivity is different (Schlegel diagrams for the TPCs used here are shown on figure 5). Table 3 shows how the GA performed on these clusters.

The results show that GA performed best on symmetric TPCs with smaller faces (TPC 70 - TPC 42) and that asymmetrical polyhedra with one big and several small faces are hard to optimize. GA's convergence was the fastest on TPC 70 and slowest on the TPC 30. TPC 70 has only faces with 4 and 5 vertices and several axes of symmetry and TPC 30 has several faces with 3, 4, 5 and one with 8 vertices and it has only one axis of symmetry.

Figure 3: Vertice distances through the run of GA for TPC 45.

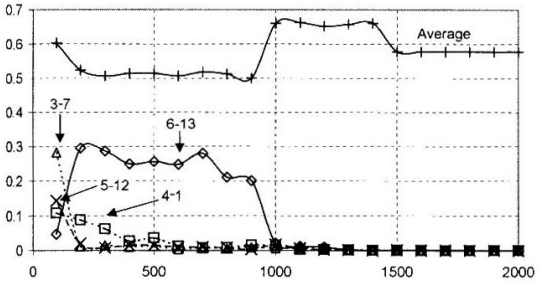
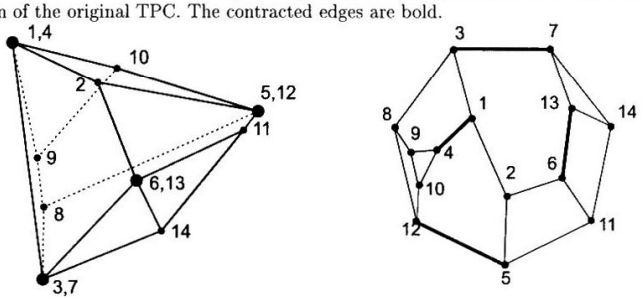
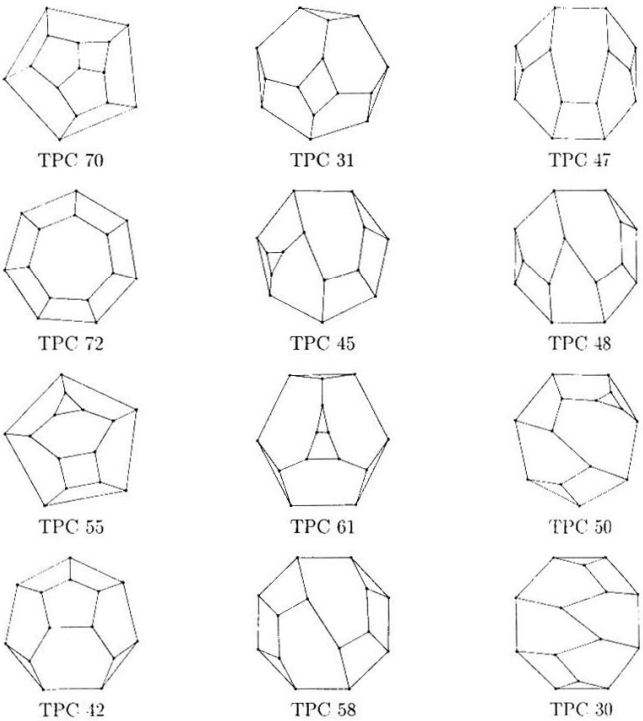



Figure 4: The resulting TPC 45 geometry with contracted vertices and the Schlegel diagram of the original TPC. The contracted edges are bold.




This is also the difference among TPC 30 and other used TPCs. It might be concluded that GA converges fast when the cluster is symmetric and has simple faces. On contrary the GA does not perform so well on highly asymmetric clusters with large faces.
An interesting phenomena was observed when maximizing IQ for TPC. The maximization of IQ contracts vertices in order to obtain more symmetric structures with smaller faces. An example of this is TPC 45. Figure 3 shows the distance among certain vertex pairs versus generation number. After 1500 generations the pairs (1,4), (3,7), (5,12) and (6,13) were at average distance less than 0.001 and at the end of GA run the distance was 0. The original TPC 45 has the following faces: 1 heptagon, 2 hexagons, 1 pentagon, 3 quadrangles and 2 triangles. The structure obtained after maximizing the IQ has only one pentagon, other faces are triangles and quadrangles. Similar behavior was observed with other clusters. This shows that structures with large IQ usually have faces with small number of vertices.

Figure 5: The connectivities of trivalent polyhedral clusters, used for analyzing GA performance on clusters with the same number of vertices. The GA performed best on the first column and worst on the last column.



## Acknowledgments

I would like to express gratitude to Dr. Tomaž Pisanski, whose comments helped me to improve the paper. I would also like to thank all the contributors to the Vega [7, 9] project; their work saved quite a lot of my time.

## References

[1] Emilie Aarts, Jan Karel Lenstra (ed.): Local Search in Combinatorial Optimization, John Willey & Sons, Chichester (1997)

[2] A. Deza, M. Deza, and W. Grishukhin, Embeddings into Half-Cubes of Fullerenes, Virus Capsids, Geodesic Domes and Coordinations Polyhedra, Submitted.

[3] P.W. Fowler, T. Pisanski and J. Shawe-Taylor in: R. Tamassia and I.G. Tollis (Eds.), Graph Drawing, Lecture Notes in Computer Science (Springer Verlag, Berlin) 894 (1995) 282.

[4] Y. Jiang, Y. Shao, E. Kirby: Topology and Stability of Trivalent Polyhedral Clusters, Fullerene Science and Technology 2 (1994) 481-497.

[5] Z. Michalewicz: Genetic Algorithms + Data structures = Evolution Programs (third, revised edition), Springer-Verlag, Berlin (1996)

[6] M. Mitchell: An Introduction to Genetic Algorithms, MIT Press (1996)

[7] T. Pisanski (Ed.), Vega Version 0.2 Quick Reference Manual and Vega Graph Gallery (Commission for the DMFA publications, Jadranska 19, SI-1111 Ljubljana, Slovenia 1995).

[8] T. Pisanski et al.: Isoperimetric Quotient for Fullerenes and Other Polyhedral Cages, J. Chemical Information and Computer Sciences 37 (1997) 1028-1032.

[9] T. Pisanski et al., Vega: System for manipulating discrete mathematical structures. Available on `http://vega.ijp.si/` .

[10] T. Pisanski, B. Plestenjak and A. Graovac: NiceGraph Program and its application in Chemistry, Croatica Chemica Acta, 68 (1995) 283 - 292.

[11] G. Pólya: Induction and analogy in mathematics, Princeton University Press, Princeton (1954)