Combinatorics and Graph Theory with Mathematica

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.sunysb.edu/∼skiena

Joint work with Sriram Pemmaraju.

Motivation

In combinatorics and graph theory, theorems get developed by formulating conjectures and then seeking counter-examples or experimental support.
Combinatorica is a system for exploring discrete mathematics. It enhances Mathematica by over 450 functions to

- Construct combinatorial objects.
- Determine their properties.
- Display them to reflect their structure.

These objects include permutations, partitions, Young tableaux, and particularly graphs.

Software for Discrete Mathematics

Most competing programs from the time of the original *Combinatorica* do not exist anymore!

- *Networks Package (Colborn, Waterloo)* - Graph package focusing on network reliability.
- *Combinatorica (Pemmaraju and Skiena)* - Mathematica package with invariants, graph database, and Java-based editor.
- *Leda (Mehlhorn and Naher)* - C++ library of data structures and algorithms.
- *Boost Graph Library (Siek)* - Graph Template Library for C++

Outline of Talk

- Description of the new *Combinatorica* as an extension of Mathematica.
- Some demos of the new *Combinatorica*.
- Interesting Problems in Discrete Mathematics approached via *Combinatorica*.

Research Applications

Graph theorists as well as engineers and scientists from a variety of other fields:

- Monkey social networks (Northwestern)
- Vertex enumeration of convex hulls (McGill)
- Physics of automata universes (Linz)
- Partitions of atomic nuclei (Rutgers)

Unfolding Polyhedra

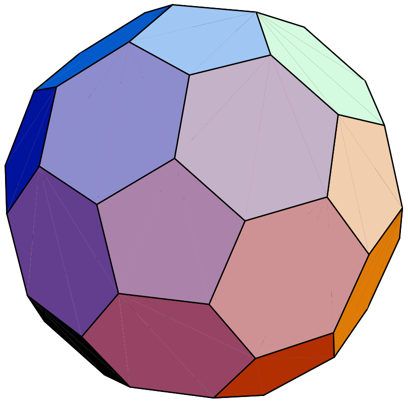
Everyone has seen polyhedral models which can be cut from paper and folded to make the object.

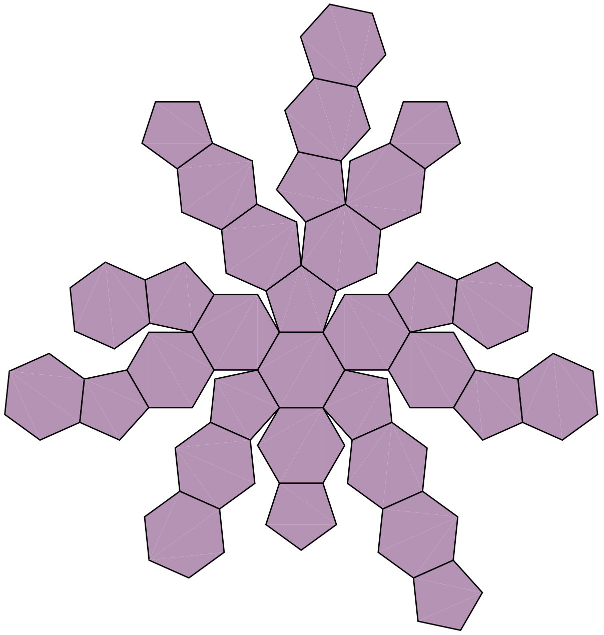
But can every convex polyhedra be so unfolded without self-intersecting on the paper?

This problem has been open for hundreds of years!

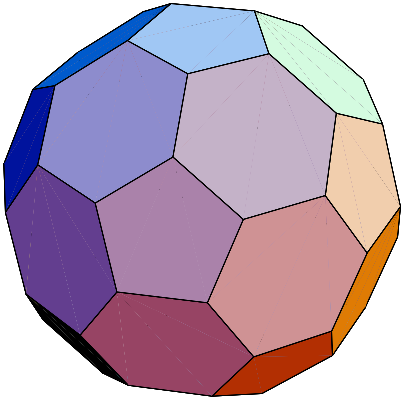
Makoto Namiki and Komei Fukuda built a package to find unfoldings on top of *Combinatorica*, exploiting that every possible unfolding is a spanning tree of the dual graph of the polyhedra.

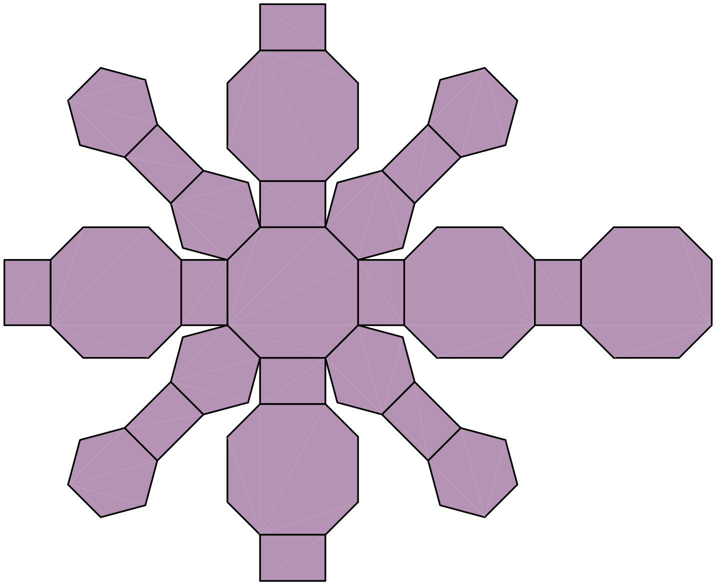
Truncated Isosahedron and Unfolding





Great Rhombicuboctahedron 4.6.8 and Unfolding





Graph Grammars

Graph grammars are a computational model using rules which rewrite graphs based on the existence of given subgraphs.

$$\begin{tikzcd}
\begin{matrix}0&0&1\\0&0&0\\0&1&0\end{matrix} \arrow[d, "\begin{smallmatrix}0&0&1&0&0\\0&0&0&1&0\\1&0&0&0&0\end{smallmatrix}"'] & \begin{matrix}0&0\\0&0\end{matrix} \arrow[l, "\begin{smallmatrix}1&0&0\\0&1&0\end{smallmatrix}"'] \arrow[r, "\begin{smallmatrix}1&0\\0&1\end{smallmatrix}"] \arrow[d, phantom, "\begin{smallmatrix}0&1&0&0\\0&0&1&0\end{smallmatrix}"] & \begin{matrix}0&1\\0&0\end{matrix} \arrow[d, "\begin{smallmatrix}1&0&0&0\\0&1&0&0\end{smallmatrix}"] \\
\begin{matrix}0&0&1&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&1&1&0&1\\0&0&1&1&0\end{matrix} & \begin{matrix}0&1&1&0\\1&0&1&1\\1&1&0&1\\0&1&1&0\end{matrix} \arrow[l, "\begin{smallmatrix}0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{smallmatrix}"] \arrow[r, "\begin{smallmatrix}0&0&1&0\\1&0&0&0\\0&1&0&0\\0&0&0&1\end{smallmatrix}"'] & \begin{matrix}0&1&1&1\\1&0&1&1\\1&1&0&0\\1&1&0&0\end{matrix}
\end{tikzcd}$$

Gabriel Valiente of UPC, Barcelona has used *Combinatorica* to build *Grammatica* a package for manipulating graph grammars.

http://www.lsi.upc.es/∼valiente/grammatica/

Education

Used for courses in the United States as well as internationally.

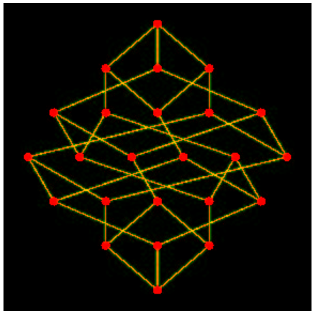
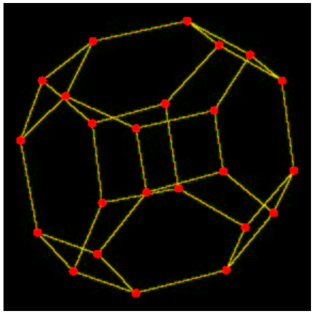
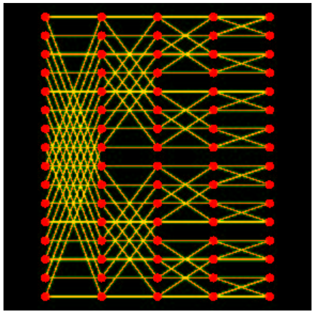
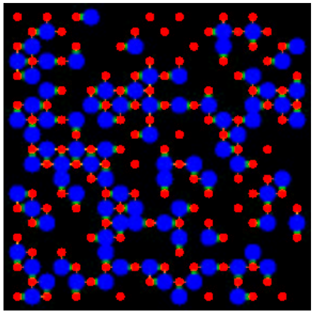
Winner: EDUCOM Distinguished Mathematics Award

Student projects include modeling Albuquerque's road network (U. NM) and studying stable marriages (Philadelphia H.S.).

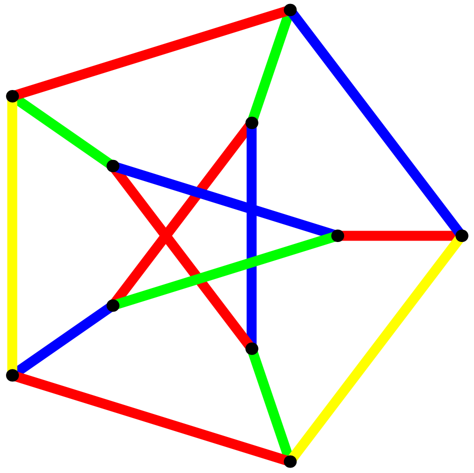
Our new book *Computational Discrete Mathematics* is designed to be suitable as a text or supplement in combinatorics and graph theory courses.

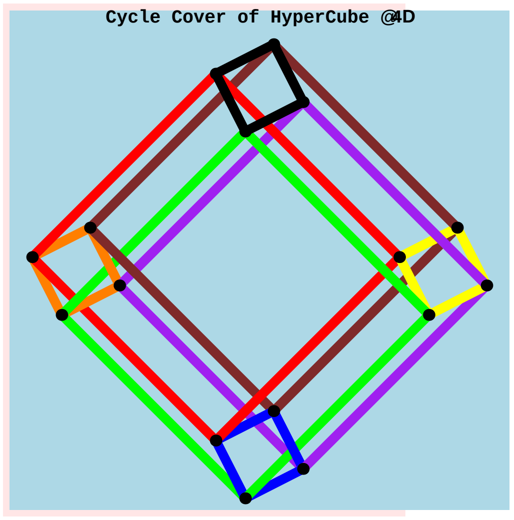
New *Mathematica* licensing policies from WRI make educational use much more practical.

Improved Graphics and Animation

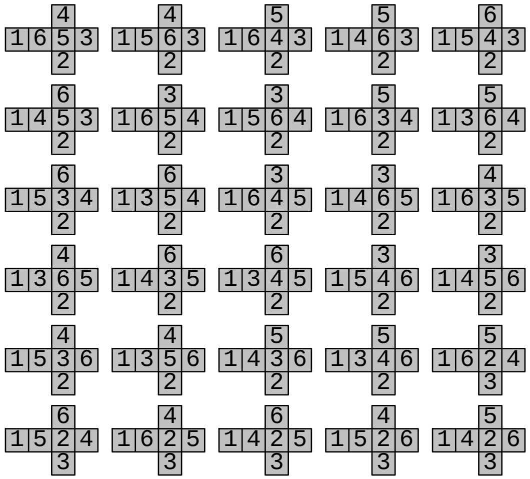





Color to Illustrate Structure



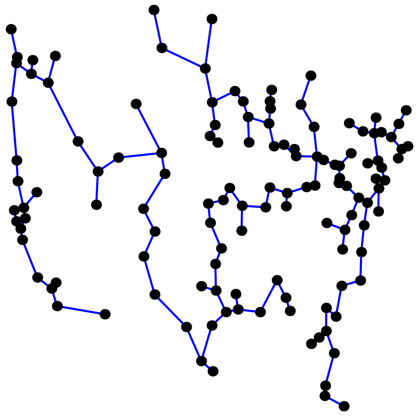


New Functionality



New features include set partitions, Polya’s theory, and countless functional improvements.

Improved Algorithmics



Algorithms for minimum spanning tree, shortest path, and network flow are now much more efficient.

Why Mathematica Instead of C?

Portability – across a wide variety of machines.

Dissemination – built-in user communities and distribution channels.

High-Level Language - 2500 lines of code sufficed for over 230 documented functions, now 7500 for over 450 functions. The functions are short enough to be understandable.

Facilities – symbolic computation, arbitrary precision arithmetic, graphics, and the rest of Mathematica makes Combinatorica more powerful.

Efficiency

The drawback is *efficiency*, although even the *old* Combinatorica now runs 100 times faster than it used to:

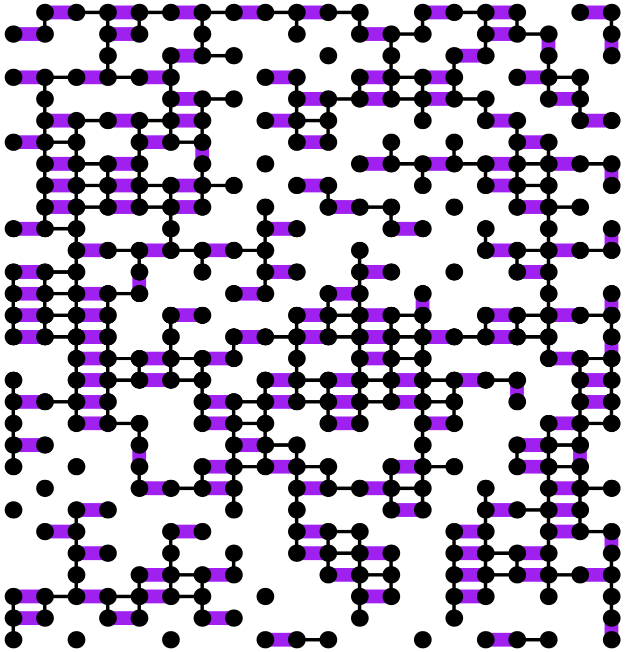
Task	Sun-3 (1990)	This Computer (2003)
PlanarQ[GridGraph[4,4]]	234.10	0.33
Length[Partitions[30]]	289.85	1.11
C VertexConnectivity[GridGraph[3,3]]	239.67	0.68
RandomPartition[1000]	831.68	0.89

Even bigger speed improvements occur in the new *Combinatorica* on many graph functions.

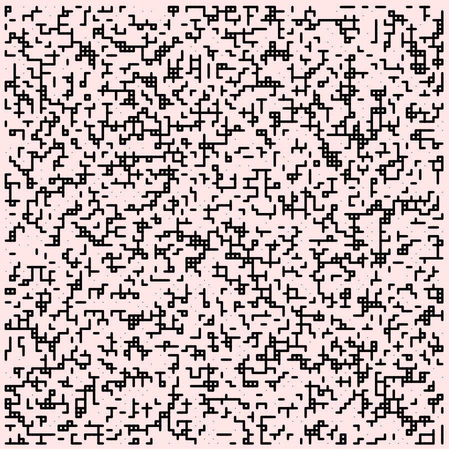
Experiments with tens or even hundreds of thousands of vertices are now doable, where 50 once seemed the limit!

The Wolf-RAM model of computation requires non-traditional programming techniques to achieve efficiency.

Much of the improved performance results from using sparse graph data structures and selective compilation.



Improved Combinatorica Performance



This random subgraph of a 100×100 grid graph, has 672 connected components, found in 2.5 seconds.

```
g = GridGraph[100, 100];
h = InduceSubgraph[g, RandomSubset[Range[10000]]];
Timing[c = ConnectedComponents[h];]
```

How Long is a Run?

We can break any permutation of numbers in *runs*, where a run is defined as a maximal sequence of increasing, left-to-right consecutive elements. The permutation $(5, 2, 3, 1, 4)$ defines the runs $(5), (2, 3), (1, 4)$.

1. Which permutations have the most and least number of runs?
2. What is the expected length of the first run in a random permutation of n element, for large n?
3. Is the expected length of the second run in a random permutation longer, shorter, or the same as that of the first run? (Be careful!)

Cuanto se demora una corrida?

Partimos una permutación de números en *corridas*, donde corrida se define como la secuencia maximal ascendente, de izq. a der., de elementos consecutivos. La permutación $(5, 2, 3, 1, 4)$ define las corridas $(5), (2, 3), (1, 4)$.

1. Cual permutación tiene el mayor y menor número de corridas?
2. Cual es la longitud esperada de la primera corrida en una permutación aleatoria de n elementos, para n grandes?
3. Será la longitud esperada de la segunda corrida en una permutación aleatoria mas larga, mas corta o del mismo tamaño que la primera corrida? (Sea cuidadoso!)

Computing Run Statistics

```
Average[l_List] := N[ Apply[ Plus, l] / Length[l] ]

RunAverage[l_List, i_Integer] :=
   Average[
      Map[ (If [Length[Runs[#]] >= i,
                  Length[Runs[#][[i]]], 0])&,
            l
      ]
]

StartElementAverage[l_List, i_Integer] :=
   Average[
      Map[ (If [Length[Runs[#]] >= i,
                  First[Runs[#][[i]]], 0])&,
           l
      ]
] / Length[First[l]]
```

Let’s Compute the Average Run Length for All Permutations!

```
In[91]:= RunAverage[Permutations[{1,2,3,4,5}],1]

Out[91]= 1.71667

In[92]:= RunAverage[Permutations[{1,2,3,4,5}],2]

Out[92]= 1.84167

In[93]:= RunAverage[Permutations[{1,2,3,4,5,6}], 1]

Out[93]= 1.71806

In[94]:= RunAverage[Permutations[{1,2,3,4,5,6}], 2]

Out[94]= 1.92083

In[95]:= Timing[RunAverage[Permutations[{1,2,3,4,5,6,7}],1]]

Out[95]= {1.832 Second, 1.71825}
```

```
In[96]:= Timing[RunAverage[Permutations[{1,2,3,4,5,6,7}],2]]

Out[96]= {1.813 Second, 1.94464}
```

Why is it getting so slow? The combinatorial explosion!

Let’s try random sampling!

```
In[102]:= RandomPermutation[10]
Out[102]= {6, 4, 3, 8, 9, 5, 7, 1, 10, 2}

In[103]:= t = Table[RandomPermutation[100],{100}];

In[104]:= RunAverage[ t, 1]
Out[104]= 1.69

In[105]:= RunAverage[ t, 2]
Out[105]= 1.89

In[106]:=  t = Table[RandomPermutation[100],{100}];

In[107]:=  RunAverage[ t, 1]
Out[107]= 1.67

In[108]:= RunAverage[ t, 2]
Out[108]= 2.13

In[109]:= t = Table[RandomPermutation[20], {1000}];

In[110]:=  RunAverage[ t, 1]
Out[110]= 1.776

In[111]:= RunAverage[ t, 2]
Out[111]= 1.972
```

Why do you believe what you believe? Students discover random sampling and the need for confidence intervals!

What is the average value of the first element of a run?

```
In[117]:= t = Table[RandomPermutation[20], {1000}];

In[118]:= StartElementAverage[t,1]

Out[118]= 0.5164

In[119]:= StartElementAverage[t,2]

Out[119]= 0.3716
```

Aha! The second run is longer than the first because the starting element is on average smaller. By definition, it must be smaller than the last element of the first run, so it must be smaller than the average element!

The first value is 0.5 on average, as a random element should be.

How Long is the First Run?

For the first run to be of length i, there $i - 1$ arrangements of the first i items to make it one run. Thus:

```
In[127]:= N[ Sum[iˆ2/(i+1)!, {i,1,20}] ]

Out[127]= 1.71828

In[129]:= <<Algebra‘SymbolicSum‘

In[134]:= SymbolicSum[ iˆ2/(i+1)!, {i,1,Infinity}]

Out[134]= -1 + E
```

Animations

Mathematica animations are not applets, but a series of canned images which can be converted to animated gifs.
See our collection of algorithm animations on www.combinatorica.com
WebMathematica makes it possible to live demos on the web if you are willing to dedicate the appropriate resources.

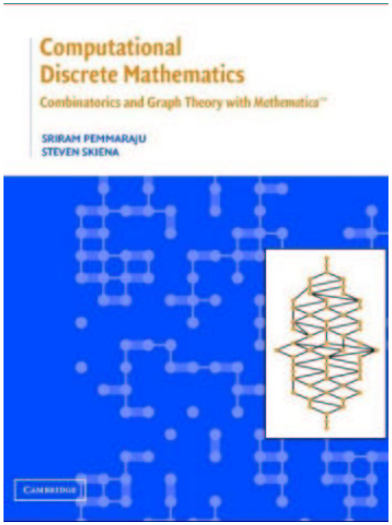
Combinatorica Demo

Availability

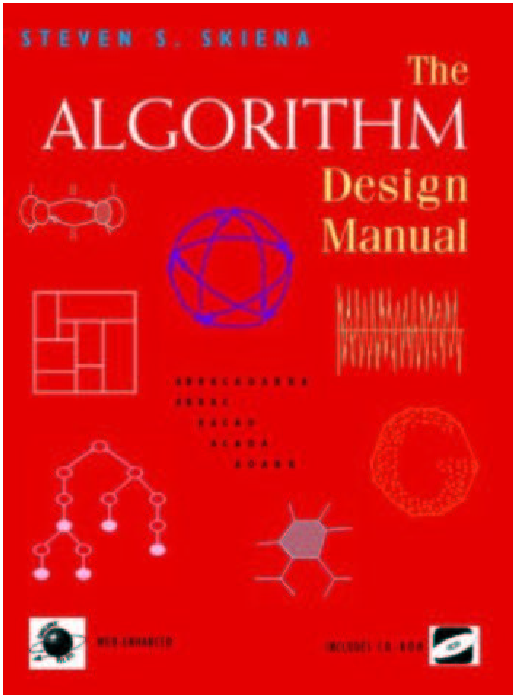
Combinatorica is included with the distribution of Mathematica in the Packages/DiscreteMath directory.
It is also available from www.combinatorica.com
Documentation is provided online and in the Mathematica Packages Guide, but the best source is:

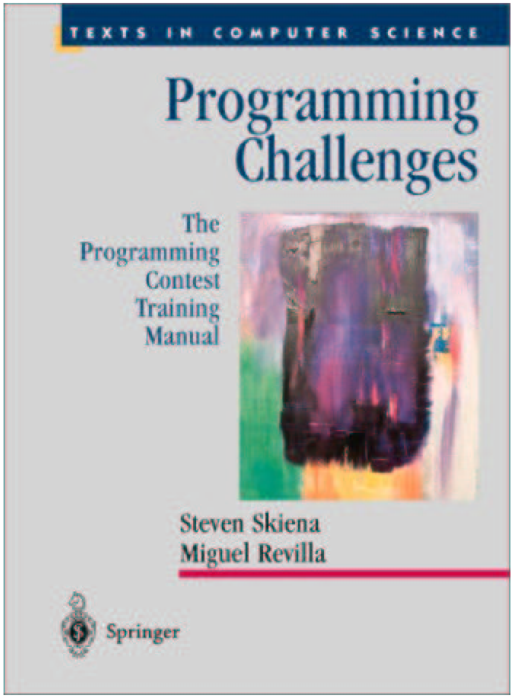
The Book

S. Pemmaraju and S. Skiena, *Computational Discrete Mathematics: Combinatorics and Graph Theory in Mathematica*, Cambridge University Press, 2003.



For Further Reading




TEXTS IN COMPUTER SCIENCE
Programming Challenges
The Programming Contest Training Manual
Steven Skiena
Miguel Revilla
Springer