Combinatorial Generation in the Presence of Symmetry

Derrick Stolee

Iowa State University
dstolee@iastate.edu
http://www.math.iastate.edu/dstolee/

December 2, 2013
ISU MECS Seminar
Application: Generating Chemical Molecules
Application: Generating Chemical Molecules

α-CD

β-CD

γ-CD
Application: Generating Chemical Molecules

Chirality?
Application: Compiling Software

1. \( r = c \div a_3 \)
2. \( p = -b \div a_3 \)
3. \( a_3 = 3 \times a \)
4. \( ad_3 = a_3 \times d \)
5. \( psq = p \times p \)
6. \( psq = psq \times psq \)
7. \( pcu = psq \times psq \)
8. \( q = pcu - (bc - thread) \div sixasq \)
9. \( rmq = r - psq \)
10. \( qsq = q \times q \)
11. \( t = rmq \times rmq \times rmq \)
12. \( z = \sqrt{qsq - t} \)
13. \( w = \sqrt[3]{q + z} \)
14. \( y = \sqrt[3]{q - z} \)
15. \( x = w + y + p \)
Exponential Behavior is Unavoidable

When dealing with NP-hard problems (or worse!), exponential behavior is unavoidable.
Exponential Behavior is Unavoidable

When dealing with NP-hard problems (or worse!), exponential behavior is unavoidable.

All we can do is delay or diminish that exponential behavior.
Shifting the Exponent

\[ 1.5^N \]

- Feasible for 1000 nodes
- Feasible for 1 node
Shifting the Exponent

$T$

$N$

feasible for 1000 nodes

feasible for 1 node

$1.5^N$

$1.25^N$
Shifting the Exponent

The diagram illustrates the relationship between $T$, $N$, and $(1.5^N)$, $(1.25^N)$, and $(1.25^{N-N_0})$. The curves represent different scenarios:

- The blue region above $1.5^N$ is feasible for 1000 nodes.
- The green region above $1.25^N$ is feasible for 1 node.
- The yellow region below $1.25^{N-N_0}$ also indicates feasibility for 1 node.

The diagram helps visualize how shifting the exponent affects the feasibility of certain conditions.
Graphs
Graphs
An **isomorphism** between $G_1$ and $G_2$ is a bijection from $V(G_1)$ to $V(G_2)$ that induces a bijection from $E(G_1)$ to $E(G_2)$. 
An **isomorphism** between $G_1$ and $G_2$ is a bijection from $V(G_1)$ to $V(G_2)$ that induces a bijection from $E(G_1)$ to $E(G_2)$.

An **automorphism** of $G$ is a bijection from $V(G)$ to $V(G)$ that induces a bijection from $E(G)$ to $E(G)$. 

---

**Graphs**

![Graph Diagrams]
Graphs: Automorphisms

The set of automorphisms form a group.
Graphs: Automorphisms

The set of **automorphisms** form a **group**.
Permutations

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Permutations
Permutations

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>
Permutations
Permutations
Graphs: Orbits

An **orbit** is a maximal set of objects such that every object is sent to every other object by some automorphism.
Graphs: Orbits

An **orbit** is a maximal set of objects such that every object is sent to every other object by some automorphism.
An **orbit** is a maximal set of objects such that every object is sent to every other object by some automorphism.
An **orbit** is a maximal set of objects such that every object is sent to every other object by some automorphism.
Other Objects
Colored (Partitioned) Graphs
Other Objects

Latin Squares

A B P C O D N E M F L G K H J I
B C A D P E O F N G M H L I K J
C D B E A F P G O H N I M J L K
D E C F B G A H P I O J N K M L
E F D G C H B I A J P K O L N M
F G E H D I C J B K A L P M O N
G H F I E J D K C L B M A N P O
H I G J F K E L D M C N B O A P
I J H K G L F M E N D O C P B A
J K I L H M G N F O E P D A C B
K L J M I N H O G P F A E B D C
L M K N J O I P H A G B F C E D
M N L O K P J A I B H C G D F E
N O M P L A K B J C I D H E G F
O P N A M B L C K D J E I F H G
P A O B N C M D L E K F J G I H

This 16 × 16 latin square assists in the construction of a Williams Design.
Subobjects
Independent Sets
Subobjects

(Induced) Subgraphs
Subobjects

(Induced) Subgraphs
Subobjects

(Induced) Subgraphs
Subobjects

(Induced) Subgraphs
Subobjects

(Proper) Colorings
**Goal:** Generate all *unlabeled* objects that satisfy the constraints.
Symmetry Breaking

1. Reduces isomorphic duplicates.
2. Does not allow for dynamic symmetry updates.
3. Removes symmetry, then uses standard symmetry-unaware algorithms.
Orbital Branching

1. Reduces isomorphic duplicates.
2. Allows for dynamic symmetry updates.
3. Branching method can be customized to the given problem.
4. Integrates well with branch-and-bound methods and constraint propagation.

(Ostrowski talked about this, also my CS Colloquium)
Canonical Deletion

1. Eliminates isomorphic duplicates*.
2. Allows for dynamic symmetry updates.
3. Augmentation method can be customized to the given problem.
4. Does not integrate well with branch-and-bound methods or constraint propagation.

Canonical Deletion

2. Define a *canonical construction path* to every unlabeled object.
3. Only follow paths that agree with the canonical construction path.
Let’s generate all graphs of order $n$ by adding vertices one-by-one.

**Augmentation:** Add a vertex adjacent to a set $S \subset V(G)$.

**Deletion:** Select a vertex $v \in V(G)$ to delete, $G' = G - v$. 
Generating with Vertex Augmentations
Generating with Vertex Augmentations
Generating with Vertex Augmentations
Generating with Vertex Augmentations
Generating with Vertex Augmentations
Generating with Vertex Augmentations
Example: Generating Graphs by Vertex Additions

Let’s generate all graphs of order $n$ by adding vertices one-by-one.

**Augmentation:** Add a vertex adjacent to a set $S \subseteq V(G)$.

**IMPORTANT:** Only one augmentation per orbit!

**Deletion:** Select a vertex $v \in V(G)$ to delete, $G' = G - v$. 
Canonical Labeling

A canonical labeling takes a labeled graph $G$. Canonical labels can be computed by McKay's nauty software.
Canonical Labeling

A canonical labeling takes a labeled graph $G$ and applies labels $\sigma_G(v)$ to each $v \in V(G)$. Canonically labeled graphs are isomorphic if and only if they have the same label sequence. Canonical labels can be computed by McKay's nauty software.
A **canonical labeling** takes a labeled graph $G$ and applies labels $\sigma_G(v)$ to each $v \in V(G)$ so that any $H \cong G$ with labels $\sigma_H(v)$ has an isomorphism $\sigma^{-1}_H(\sigma_G(v))$ from $G$ to $H$. Canonical labels can be computed by McKay's *nauty* software.
A canonical labeling takes a labeled graph $G$ and applies labels $\sigma_G(v)$ to each $v \in V(G)$ so that any $H \cong G$ with labels $\sigma_H(v)$ has an isomorphism $\sigma_H^{-1}(\sigma_G(v))$ from $G$ to $H$. Canonical labels can be computed by McKay's nauty software.
Canonical Labeling

A canonical labeling takes a labeled graph $G$ and applies labels $\sigma_G(v)$ to each $v \in V(G)$ so that any $H \cong G$ with labels $\sigma_H(v)$ has an isomorphism $\sigma_H^{-1}(\sigma_G(v))$ from $G$ to $H$. 

![Diagram showing graphs $G$, $[n]$, and $H$ with an isomorphism between $G$ and $H$.]
Canonical Labeling

A canonical labeling takes a labeled graph $G$ and applies labels $\sigma_G(v)$ to each $v \in V(G)$ so that any $H \cong G$ with labels $\sigma_H(v)$ has an isomorphism $\sigma_H^{-1}(\sigma_G(v))$ from $G$ to $H$. 

 Canonical labels can be computed by McKay's nauty software.
Canonical Labeling

A **canonical labeling** takes a labeled graph $G$ and applies labels $\sigma_G(v)$ to each $v \in V(G)$ so that any $H \cong G$ with labels $\sigma_H(v)$ has an isomorphism $\sigma_H^{-1}(\sigma_G(v))$ from $G$ to $H$.

Canonical labels can be computed by McKay’s *nauty* software.
Canonical Deletion by Filtering

Let $S = V(G)$. Filter $S$ until $|S| = 1$ by the following conditions:

1. Remove cut vertices from $S$.
2. Let $d = \min\{\deg(v) : v \in S\}$. Set $S \leftarrow \{v \in S : \deg(v) = d\}$.
3. (Include other, more complicated invariants here.)
4. Compute a canonical labeling $\ell$, and set $v = \arg\min_{v \in S} \ell(v)$.

The vertex $v$ is the canonical deletion.
Canonical Deletion by Filtering

Let $S = V(G)$. Filter $S$ until $|S| = 1$ by the following conditions:

1. Remove cut vertices from $S$. 

Let $S = V(G)$. Filter $S$ until $|S| = 1$ by the following conditions:

1. Remove cut vertices from $S$.
2. Let $d = \min\{\deg(v) : v \in S\}$. Set
   
   $$S \leftarrow \{v \in S : \deg(v) = d\}.$$
Let $S = V(G)$. Filter $S$ until $|S| = 1$ by the following conditions:

1. Remove cut vertices from $S$.
2. Let $d = \min\{\deg(v) : v \in S\}$. Set

   $$S \leftarrow \{v \in S : \deg(v) = d\}.$$

3. (Include other, more complicated invariants here.)
Let $S = V(G)$. Filter $S$ until $|S| = 1$ by the following conditions:

1. Remove cut vertices from $S$.
2. Let $d = \min\{\deg(v) : v \in S\}$. Set
   $$S \leftarrow \{v \in S : \deg(v) = d\}.$$ 
3. (Include other, more complicated invariants here.)
4. Compute a canonical labeling $\ell$, and set
   $$v = \arg\min_{v \in S} \ell(v).$$
Canonical Deletion by Filtering

Let $S = V(G)$. Filter $S$ until $|S| = 1$ by the following conditions:

1. Remove cut vertices from $S$.
2. Let $d = \min\{\deg(v) : v \in S\}$. Set
   \[ S \leftarrow \{ v \in S : \deg(v) = d \}. \]
3. (Include other, more complicated invariants here.)
4. Compute a canonical labeling $\ell$, and set
   \[ v = \arg\min_{v \in S} \ell(v). \]

The vertex $v$ is the canonical deletion.
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
By thinking of our filtering mechanism for the canonical deletion, we can avoid making augmentations that will not be canonical deletions:
Using Deletion To Minimize Augmentations

By thinking of our filtering mechanism for the canonical deletion, we can avoid making augmentations that will not be canonical deletions:

1. If minimizing degree, do not add anything of degree more than \(\delta(G) + 1\).
Using Deletion To Minimize Augmentations

By thinking of our filtering mechanism for the canonical deletion, we can avoid making augmentations that will not be canonical deletions:

1. If minimizing degree, do not add anything of degree more than $\delta(G) + 1$.
2. If not deleting cut-vertices, everything has degree at least one.
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Canonical Vertex Deletions
Effectiveness of Canonical Deletion

Every unlabeled object is expanded exactly once.
Effectiveness of Canonical Deletion

Every unlabeled object is **expanded** exactly once.

Every unlabeled object is **reached** at most *once per possible deletion*.
Effectiveness of Canonical Deletion

Every unlabeled graph is **expanded** exactly once.

Every unlabeled graph is **reached** at most $n$ times.
Effectiveness of Canonical Deletion

Every unlabeled graph is **expanded** exactly once.

Every unlabeled graph is **reached** at most $n$ times.

Most unlabeled graphs have $n!$ different labelings.

So the resulting computation time is about
$$\sum_{n=1}^{N} n \cdot f(n) \approx 2N^2 - N \log N$$
where $f(n)$ is the average time to compute canonical labels and automorphisms.
Effectiveness of Canonical Deletion

Every unlabeled graph is **expanded** exactly once.

Every unlabeled graph is **reached** at most \( n \) times.

Most unlabeled graphs have \( n! \) different labelings.

So the resulting computation time is about

\[
\sum_{n=1}^{N} 2^{\binom{n}{2}} \cdot \frac{nf(n)}{n!} \approx 2^{N^2 - N \log N}
\]

where \( f(n) \) is the average time to compute canonical labels and automorphisms.
<table>
<thead>
<tr>
<th>$n$</th>
<th>Labeled graphs of order $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>32,768</td>
</tr>
<tr>
<td>7</td>
<td>2,097,152</td>
</tr>
<tr>
<td>8</td>
<td>268,435,456</td>
</tr>
<tr>
<td>9</td>
<td>68,719,476,736</td>
</tr>
<tr>
<td>10</td>
<td>35,184,372,088,832</td>
</tr>
<tr>
<td>11</td>
<td>36,028,797,018,963,968</td>
</tr>
<tr>
<td>12</td>
<td>73,786,976,294,838,206,464</td>
</tr>
<tr>
<td>13</td>
<td>302,231,454,903,657,293,676,544</td>
</tr>
<tr>
<td>14</td>
<td>2,475,880,078,570,760,549,798,248,448</td>
</tr>
<tr>
<td>15</td>
<td>40,564,819,207,303,340,847,894,502,572,032</td>
</tr>
</tbody>
</table>

$$2^{\binom{n}{2}} \approx 2^{\theta(n^2)}$$
<table>
<thead>
<tr>
<th>$n$</th>
<th>Unlabeled connected graphs of order $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>509</td>
</tr>
<tr>
<td>8</td>
<td>4,060</td>
</tr>
<tr>
<td>9</td>
<td>41,301</td>
</tr>
<tr>
<td>10</td>
<td>510,489</td>
</tr>
<tr>
<td>11</td>
<td>7,319,447</td>
</tr>
<tr>
<td>12</td>
<td>117,940,535</td>
</tr>
<tr>
<td>13</td>
<td>2,094,480,864</td>
</tr>
<tr>
<td>14</td>
<td>40,497,138,011</td>
</tr>
<tr>
<td>15</td>
<td>845,480,228,069</td>
</tr>
</tbody>
</table>

OEIS Sequence A002851 Grows $2^{\Omega(n^2)}$. 
<table>
<thead>
<tr>
<th>$n$</th>
<th>Unlabeled connected graphs of order $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>509</td>
</tr>
<tr>
<td>8</td>
<td>4,060</td>
</tr>
<tr>
<td>9</td>
<td>41,301</td>
</tr>
<tr>
<td>10</td>
<td>510,489</td>
</tr>
<tr>
<td>11</td>
<td>7,319,447</td>
</tr>
<tr>
<td><strong>12</strong></td>
<td><strong>117,940,535</strong></td>
</tr>
<tr>
<td>13</td>
<td>2,094,480,864</td>
</tr>
<tr>
<td>14</td>
<td>40,497,138,011</td>
</tr>
<tr>
<td>15</td>
<td>845,480,228,069</td>
</tr>
</tbody>
</table>

Requires about 1 day of CPU Time.
<table>
<thead>
<tr>
<th>$n$</th>
<th>Unlabeled connected graphs of order $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>509</td>
</tr>
<tr>
<td>8</td>
<td>4,060</td>
</tr>
<tr>
<td>9</td>
<td>41,301</td>
</tr>
<tr>
<td>10</td>
<td>510,489</td>
</tr>
<tr>
<td>11</td>
<td>7,319,447</td>
</tr>
<tr>
<td>12</td>
<td>117,940,535</td>
</tr>
<tr>
<td>13</td>
<td>2,094,480,864</td>
</tr>
<tr>
<td>14</td>
<td>40,497,138,011</td>
</tr>
<tr>
<td>15</td>
<td>845,480,228,069</td>
</tr>
</tbody>
</table>

Requires over **1 year** of CPU Time.
Implementation

My **TreeSearch** library enables parallelization in the Condor scheduler.

Executes on the **Open Science Grid**, a collection of supercomputers around the country.
Q: How can we integrate constraint propagation with canonical deletion?
Let $f(n, p)$ be the maximum number of edges in a graph of order $n$ with exactly $p$ perfect matchings.
Let $f(n, p)$ be the maximum number of edges in a graph of order $n$ with exactly $p$ perfect matchings.

We determine this value and characterize all graphs achieving this bound for all $n$ (for small $p$).
Let \( f(n, p) \) be the **maximum number of edges** in a graph of order \( n \) with **exactly** \( p \) **perfect matchings**.

We determine this value and characterize all graphs achieving this bound for all \( n \) (for small \( p \)).

Requires building a canonical deletion that has the number of perfect matchings be **monotonic**!
To learn more...

- B. D. McKay. Isomorph-free exhaustive generation.
- B. D. McKay. Small graphs are reconstructible.
- F. Margot. Pruning by isomorphism in branch-and-cut.
- B. D. McKay, A. Meynert. Small latin squares, quasigroups, and loops.
- G. Brinkmann, B. D. McKay. Posets on up to 16 points.
- P. Kaski, P. R. J. Östergard. The Steiner triple systems of order 19.
- D. Stolee. Isomorph-free generation of 2-connected graphs with applications.
- D. Stolee. Generating $p$-extremal graphs.
Combinatorial Generation in the Presence of Symmetry

Derrick Stolee

Iowa State University
dstolee@iastate.edu
http://www.math.iastate.edu/dstolee/

December 2, 2013
ISU MECS Seminar