The canonical augmentation method

Derrick Stolee[1]
University of Nebraska–Lincoln
s-dstolee1@math.unl.edu
http://www.math.unl.edu/~s-dstolee1/

May 13, 2011

[1]Supported by NSF grants CCF-0916525 and DMS-0914815.

“Generating” means...

“Generating” means...

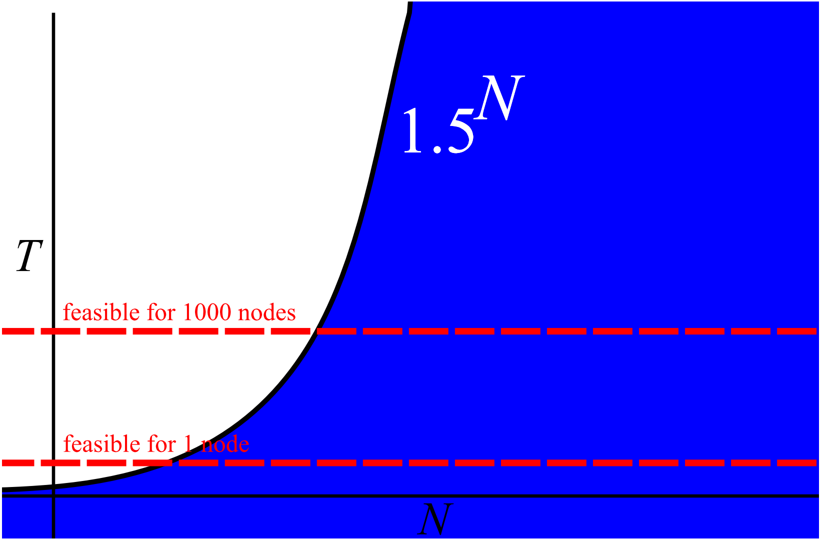
Searching (looking for objects)

“Generating” means...

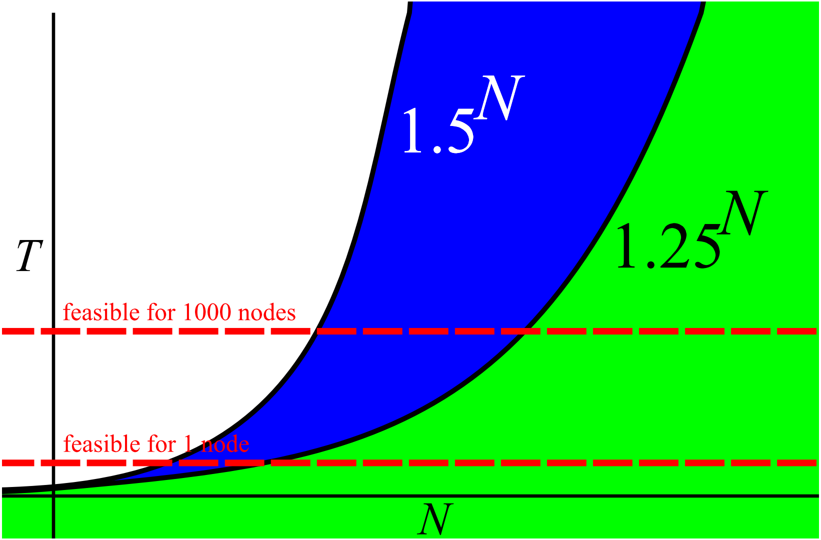
Searching (looking for objects)

Exhaustively (not missing any)

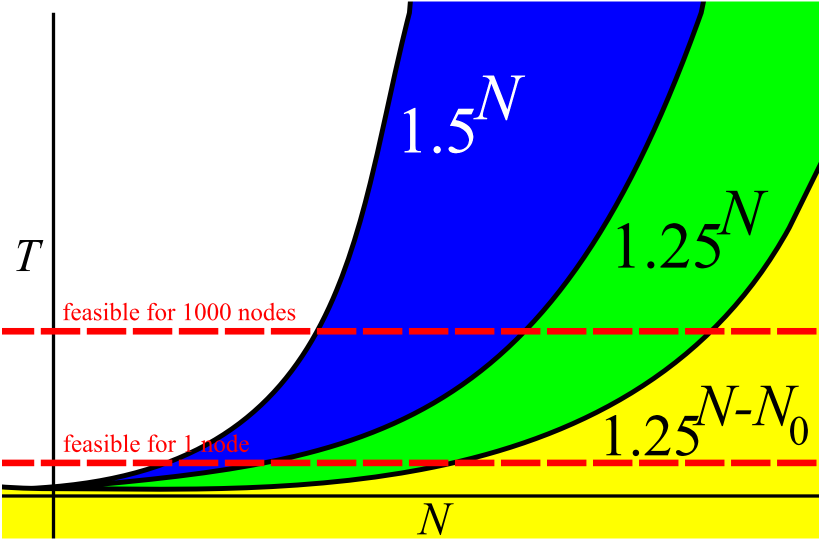
Shifting the Exponent



Shifting the Exponent



Shifting the Exponent



Search by Augmentations

While generating **[combinatorial object]**

Search by Augmentations

While generating **[combinatorial object]** we start at **[base object]**

Search by Augmentations

While generating **[combinatorial object]** we start at **[base object]** and augment by all possible **[augmentations]**

Search by Augmentations

While generating **[combinatorial object]** we start at **[base object]** and augment by all possible **[augmentations]** but keep in mind **[symmetries]**.

Search by Augmentations

While generating **[combinatorial object]** we start at **[base object]** and augment by all possible **[augmentations]** but keep in mind **[symmetries]**.

This technique leads to an isomorphism class appearing once for every possible augmentation sequence that generates that unlabeled object.

Search by Augmentations

While generating **[combinatorial object]** we start at **[base object]** and augment by all possible **[augmentations]** but keep in mind **[symmetries]**.

This technique leads to an isomorphism class appearing once for every possible augmentation sequence that generates that unlabeled object.

So, we define a **canonical deletion** which is an invariant reversal of the augmentation.

Search by Augmentations

While generating **[combinatorial object]** we start at **[base object]** and augment by all possible **[augmentations]** but keep in mind **[symmetries]**.

This technique leads to an isomorphism class appearing once for every possible augmentation sequence that generates that unlabeled object.

So, we define a **canonical deletion** which is an invariant reversal of the augmentation.

Stick in results from **[your favorite combinatorics]**, and you may have an efficient algorithm!

Search by Augmentations

While generating **triangle free graphs** we start at **an isolated vertex** and augment by all possible **vertices (with independent neighbors)** but keep in mind **set orbits.**

This technique leads to an isomorphism class appearing once for every possible augmentation sequence that generates that unlabeled object.

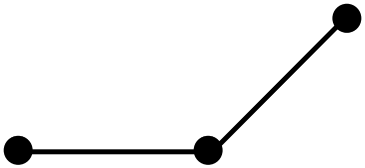
So, we define a **canonical deletion** which is an invariant reversal of the augmentation.

Stick in results from **graph theory**, and you may have an efficient algorithm!

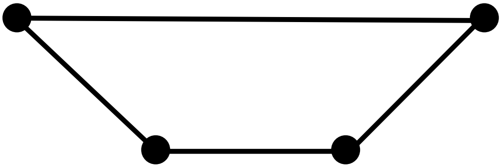
Generating with Vertex Augmentations

Generating with Vertex Augmentations

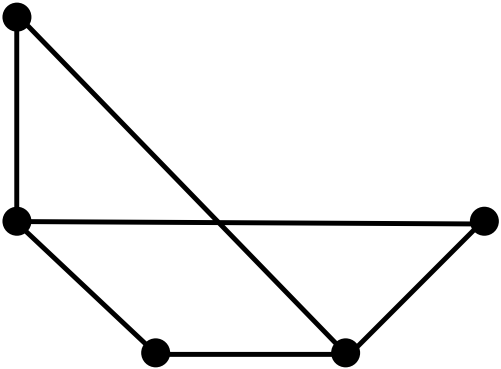
Generating with Vertex Augmentations



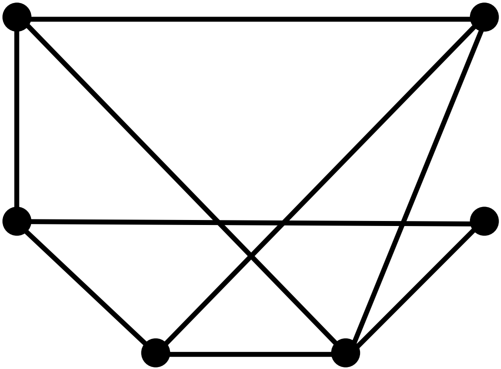
Generating with Vertex Augmentations



Generating with Vertex Augmentations



Generating with Vertex Augmentations

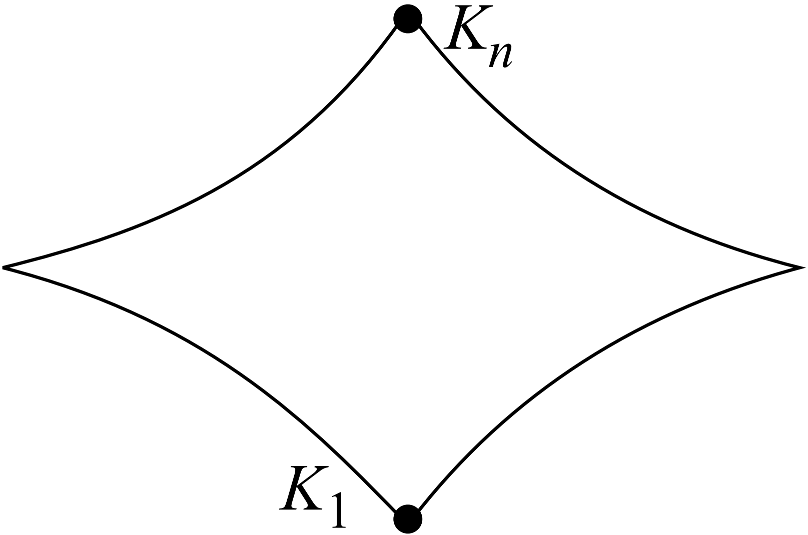


Search as a Poset

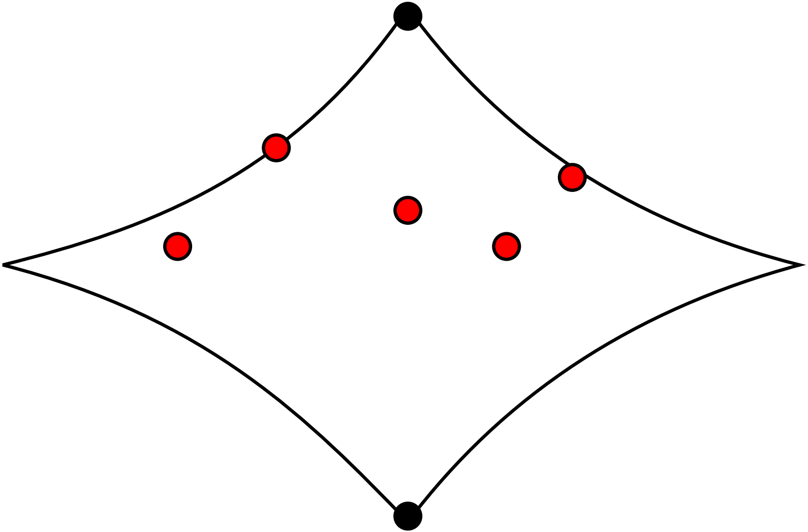
Say $H \preceq G$ if G is reachable from H via a sequence of augmentations.

This defines a partial order on **partial objects**.

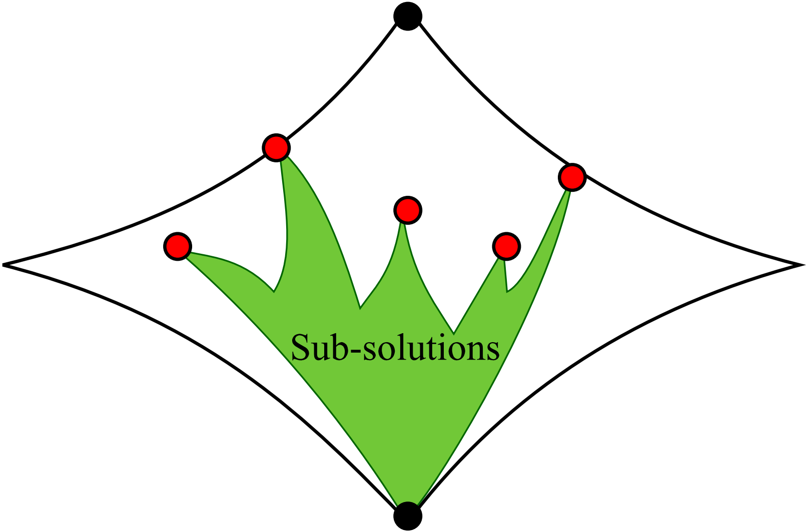
Search as a Poset



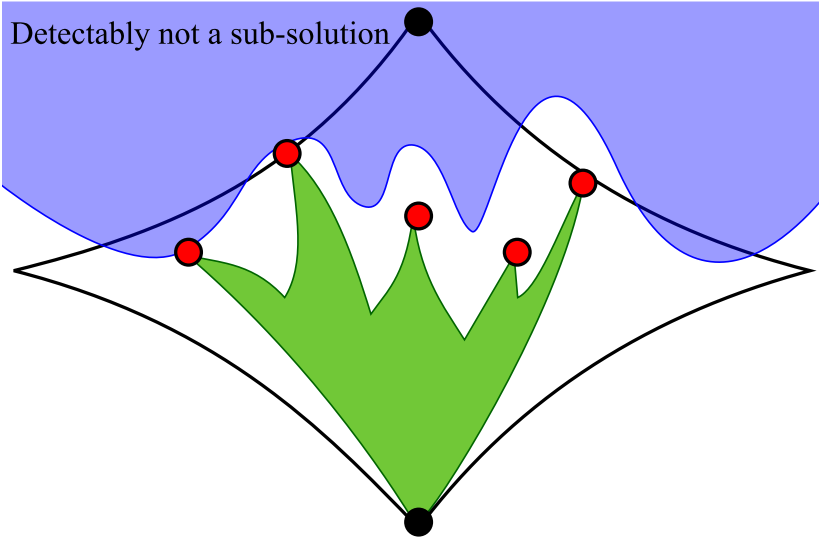
Search as a Poset



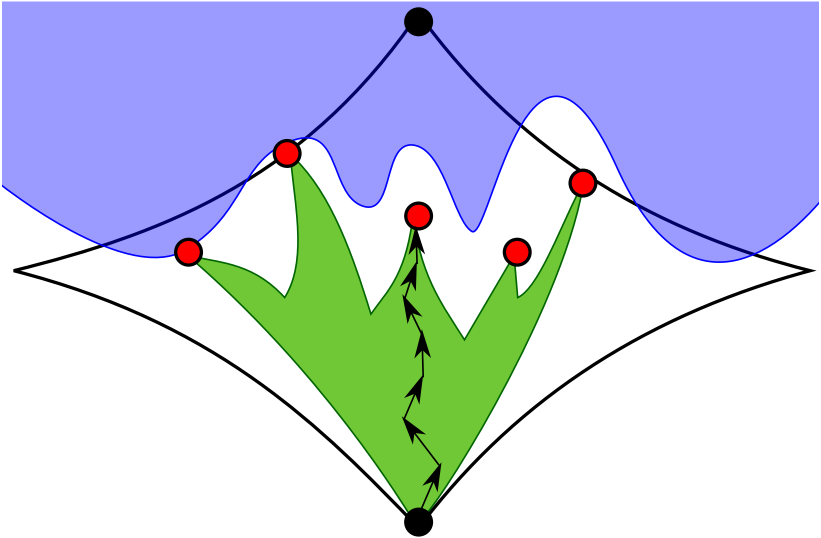
Search as a Poset



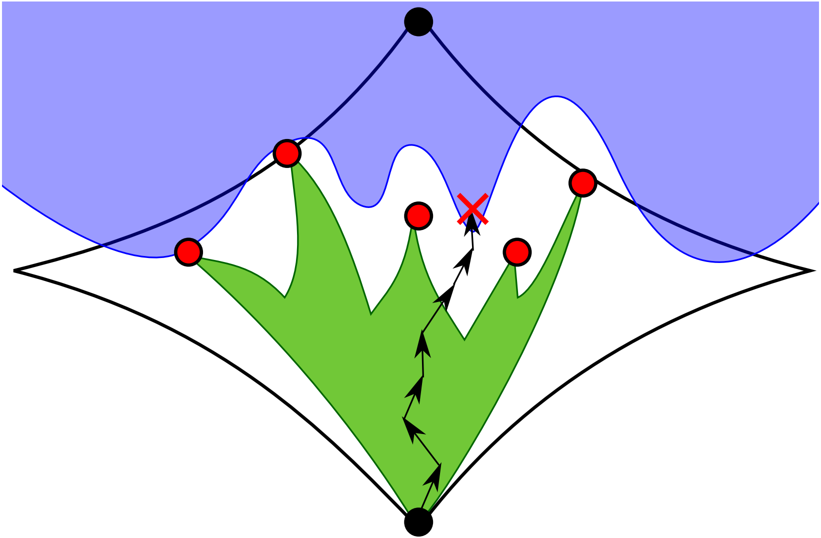
Search as a Poset



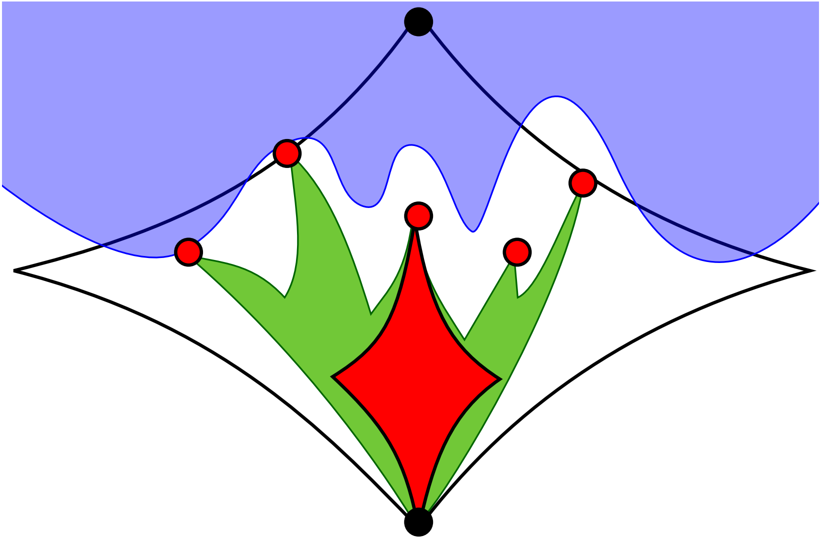
Search as a Poset



Search as a Poset



Search as a Poset



Canonical Labeling

A **canonical labeling** takes a labeled graph G

Canonical Labeling

A **canonical labeling** takes a labeled graph G and applies labels $\sigma_G(v)$ to each $v \in V(G)$

Canonical Labeling

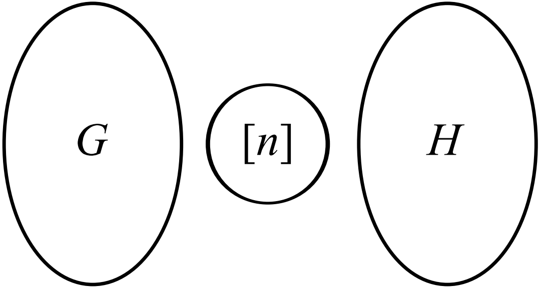
A **canonical labeling** takes a labeled graph G and applies labels $\sigma_G(v)$ to each $v \in V(G)$ so that any $H \cong G$ with labels $\sigma_H(v)$

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.

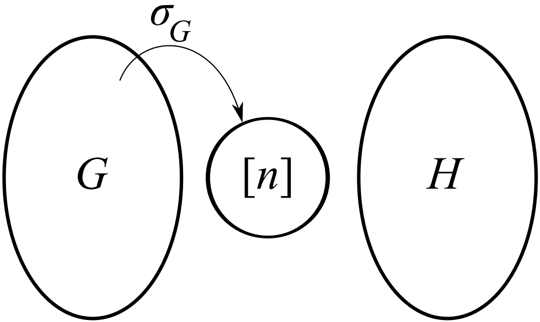
Canonical Labeling

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.



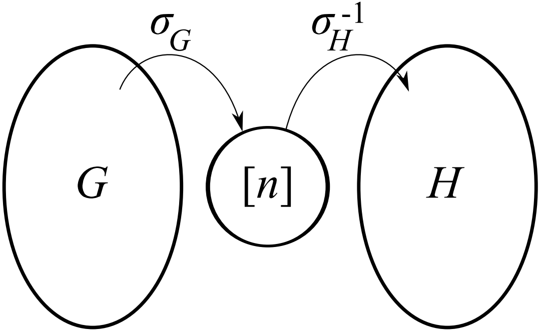
Canonical Labeling

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 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.

Brendan McKay

“Isomorph-free exhaustive generation”

258 citations on Google Scholar.

How it works

Remove *isomorphs* by:

How it works

Remove *isomorphs* by:

1. Define a **canonical deletion** (the augmentation that "should" have generated this graph).

How it works

Remove *isomorphs* by:

1. Define a **canonical deletion** (the augmentation that "should" have generated this graph).
2. The canonical deletion must be *invariant*.

How it works

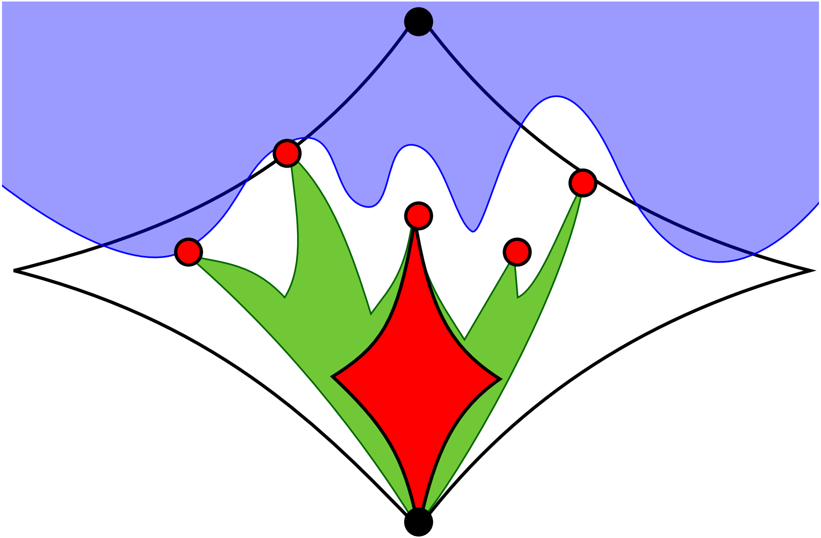
Remove *isomorphs* by:

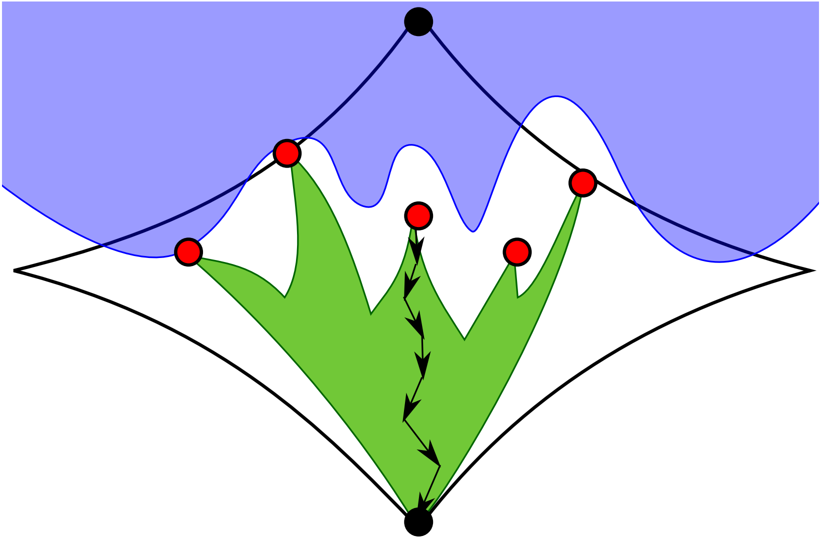
1. Define a **canonical deletion** (the augmentation that "should" have generated this graph).
2. The canonical deletion must be *invariant*.
3. Reject any augmentations not isomorphic to canonical deletion.

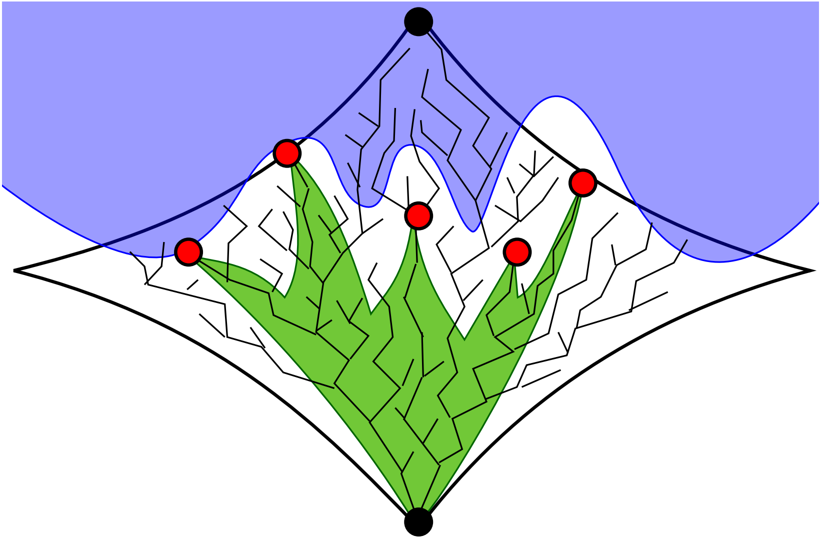
How it works

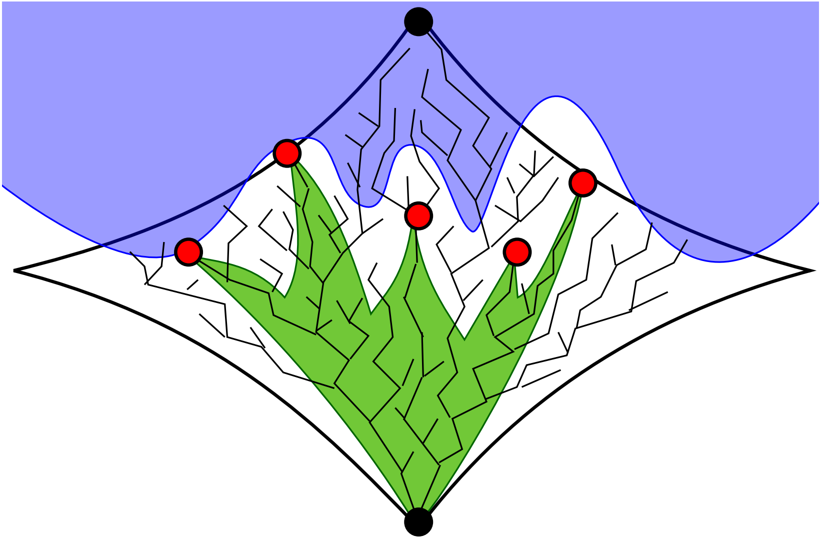
Remove *isomorphs* by:

1. Define a **canonical deletion** (the augmentation that "should" have generated this graph).
2. The canonical deletion must be *invariant*.
3. Reject any augmentations not isomorphic to canonical deletion.
4. **Optimization:** Use deletion rule to reduce number of attempted augmentations (e.g. minimum degree).

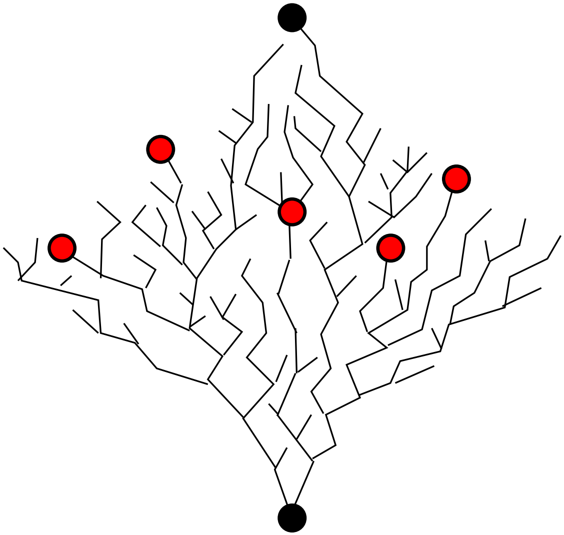


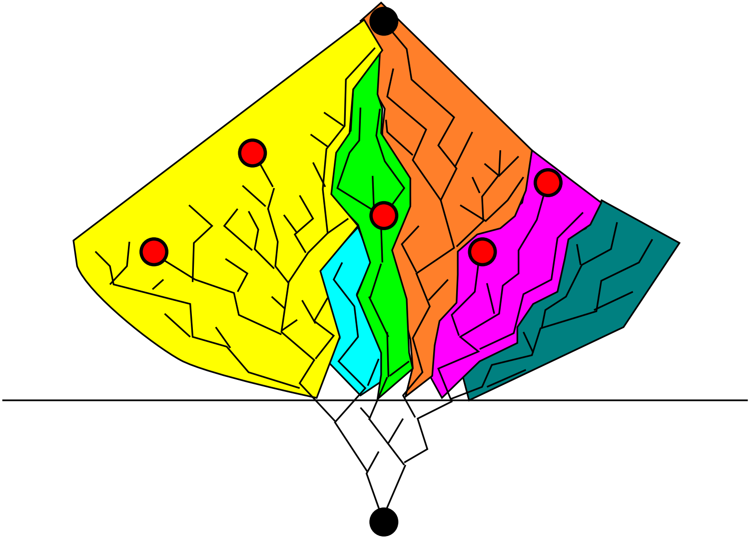






(take a drink)





Isomorph-free Generation Examples

1. Triangle-Free Graphs

Isomorph-free Generation Examples

1. Triangle-Free Graphs
2. Posets (up to order 16)

Isomorph-free Generation Examples

1. Triangle-Free Graphs
2. Posets (up to order 16)
3. Latin Squares

Isomorph-free Generation Examples

1. Triangle-Free Graphs
2. Posets (up to order 16)
3. Latin Squares
4. Steiner Triple Systems

Isomorph-free Generation Examples

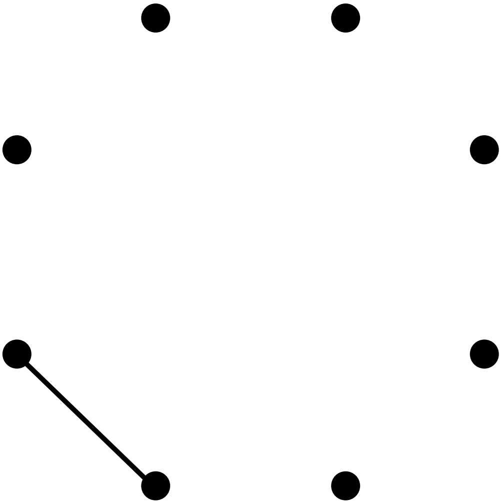
1. Triangle-Free Graphs
2. Posets (up to order 16)
3. Latin Squares
4. Steiner Triple Systems
5. Verify Reconstruction Conjecture (up to 11 vertices).

Examples in Software

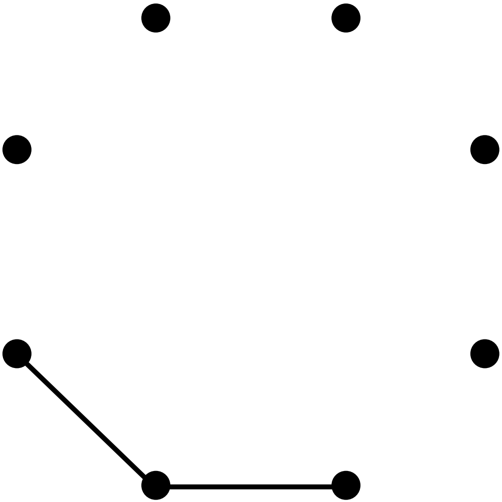
Canonical augmentation appears in the following software:

1. McKay's `geng` and `genbg` programs.
2. Sage.org's Graph library: `graphs()` and `digraphs()` methods.

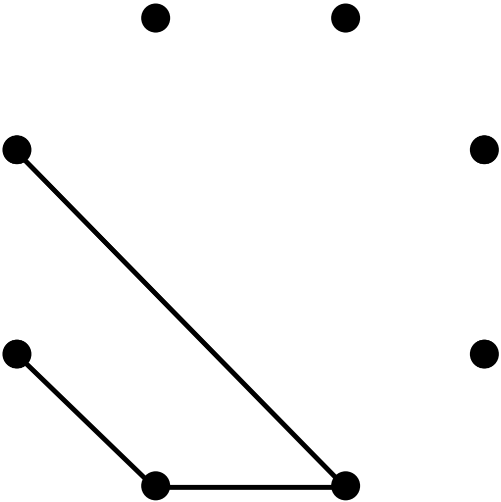
Augmentations by Edges



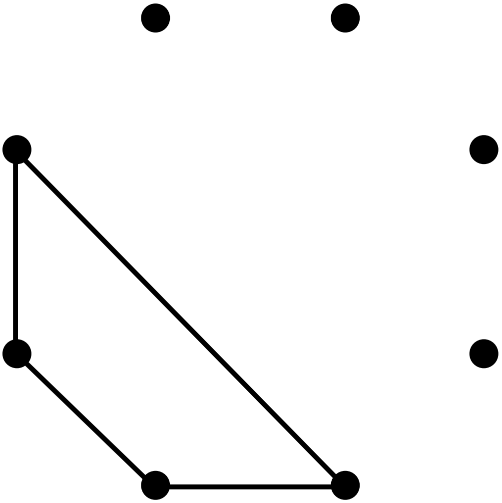
Augmentations by Edges



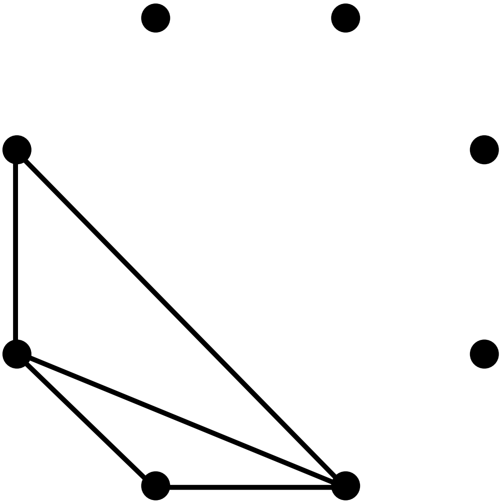
Augmentations by Edges



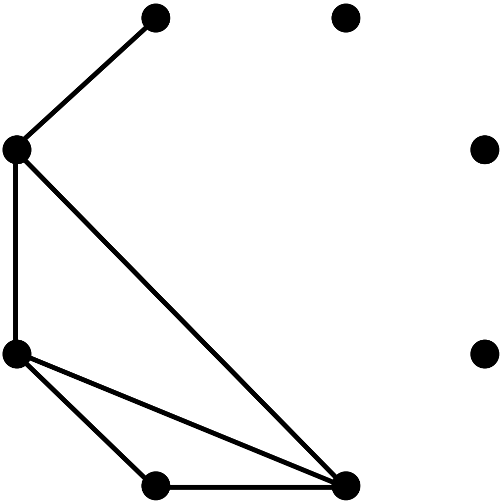
Augmentations by Edges



Augmentations by Edges



Augmentations by Edges



Examples in Software

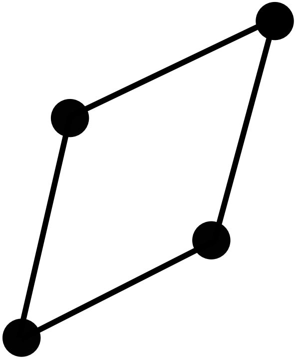
Canonical augmentation by ears appears in the following software:

1. Sage.org's Graph library: `graphs()` and `digraphs()` methods.

   (Use the flag `augment='edges'`)

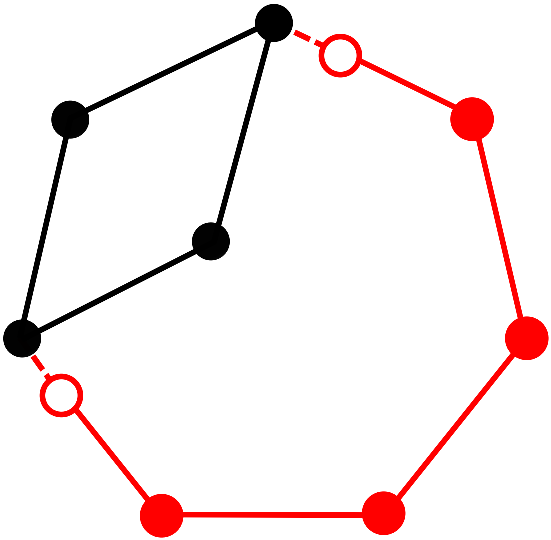
Augmentations by Ears

All 2-connected graphs have an ear decomposition.



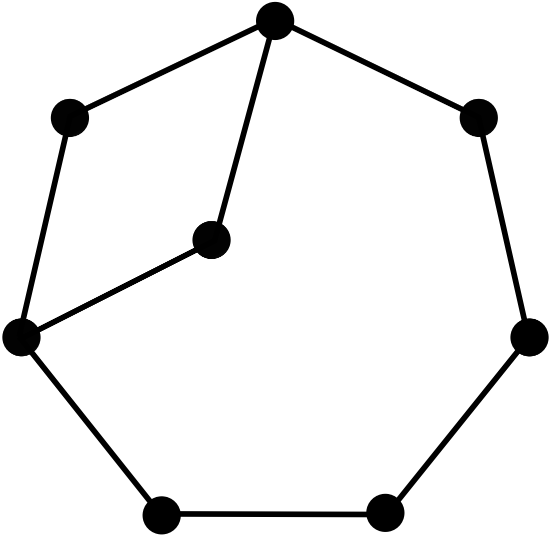
Augmentations by Ears

All 2-connected graphs have an ear decomposition.



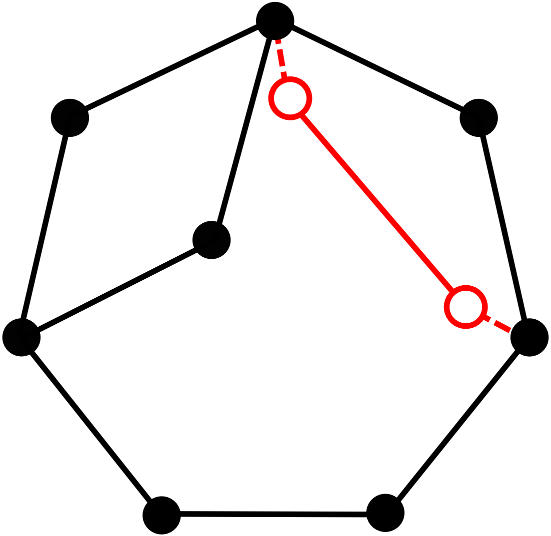
Augmentations by Ears

All 2-connected graphs have an ear decomposition.



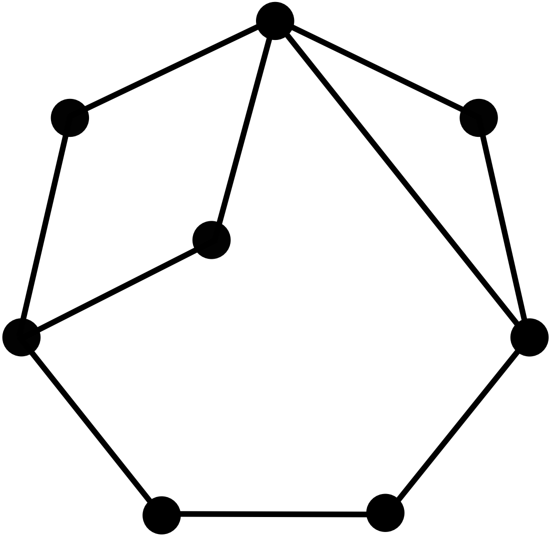
Augmentations by Ears

All 2-connected graphs have an ear decomposition.



Augmentations by Ears

All 2-connected graphs have an ear decomposition.



Ear Augmentation Applications

Ear Augmentation Applications

1. Edge-Reconstruction Conjecture.

Ear Augmentation Applications

1. Edge-Reconstruction Conjecture.

2. Extremal graphs with a fixed number of perfect matchings.

Ear Augmentation Applications

1. Edge-Reconstruction Conjecture.
2. Extremal graphs with a fixed number of perfect matchings.
3. **Uniquely K_r-saturated graphs.**

Uniquely K_r-Saturated Graphs

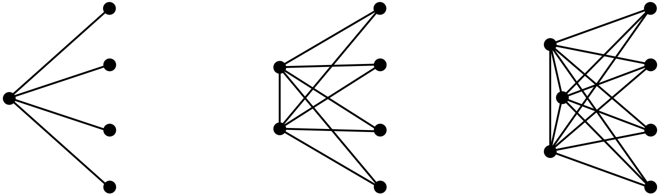
Definition

A graph G is *uniquely K_r-saturated* if G contains no K_r and for every edge $e \in \overline{G}$ admits exactly one copy of K_r in $G + e$.

Uniquely K_r-Saturated Graphs

Definition

A graph G is *uniquely K_r-saturated* if G contains no K_r and for every edge $e \in \overline{G}$ admits exactly one copy of K_r in $G + e$.



(a) 1-book (b) 2-book (c) 3-book

Figure: The $(r-2)$-books are uniquely K_r saturated.

Dominating Vertices

Adding a dominating vertex to a uniquely K_r-saturated graph creates a uniquely K_{r+1}-saturated graph.

Removing a dominating vertex from a uniquely K_r-saturated graph creates a uniquely K_{r-1}-saturated graph.

Dominating Vertices

Adding a dominating vertex to a uniquely K_r-saturated graph creates a uniquely K_{r+1}-saturated graph.

Removing a dominating vertex from a uniquely K_r-saturated graph creates a uniquely K_{r-1}-saturated graph.

Q: Which uniquely K_r-saturated graphs have no dominating vertex?

Dominating Vertices

Adding a dominating vertex to a uniquely K_r-saturated graph creates a uniquely K_{r+1}-saturated graph.

Removing a dominating vertex from a uniquely K_r-saturated graph creates a uniquely K_{r-1}-saturated graph.

Q: Which uniquely K_r-saturated graphs have no dominating vertex?

A: Known for $r \in \{2, 3\}$.

Joshua Cooper Paul Wenger

Two Conjectures:

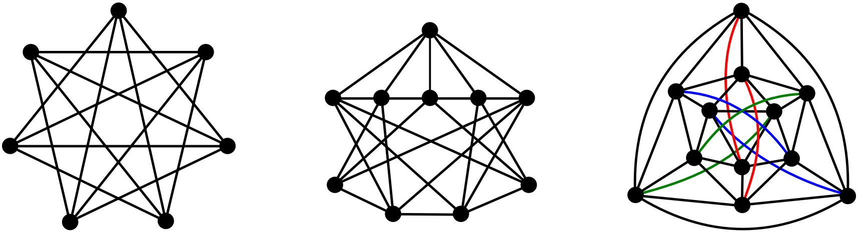
1. For each r, there are a finite number of uniquely K_r-saturated graphs with no dominating vertex.

2. For each r, every uniquely K_r-saturated graph with no dominating vertex is regular.

Previously verified to 9 vertices.

Uniquely K_r-Saturated Graphs

1. Uniquely K_r-saturated graphs have diameter 2 (and are 2-connected).
2. Strength: K_4-free is a sparse, monotone property.
3. Verified for $r = 4$ and $n \leq 12$.
4. Verified for $r \in \{5, 6\}$ and $n \leq 11$.



(a) (b) (c)

The only uniquely K_4-saturated graphs on up to 12 vertices with no dominating vertex.

Coupled Augmentations

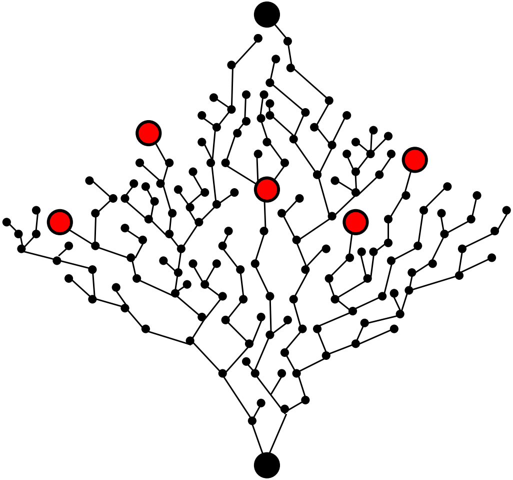
Idea: Let the problem constraints dictate the augmentation type.

Caveat Programmer

Balance: *Number of Nodes* vs. *Computation per Node*

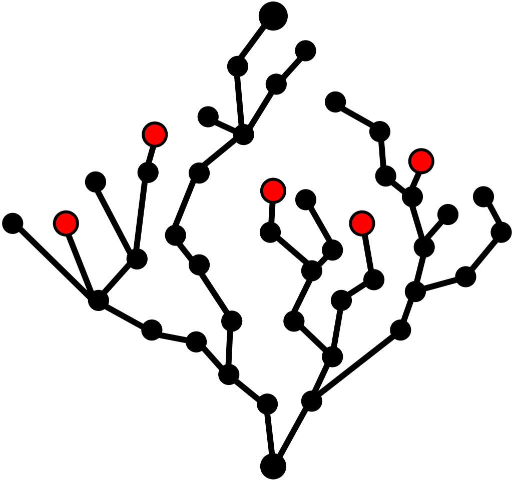
Caveat Programmer

Balance: *Number of Nodes* vs. *Computation per Node*



Caveat Programmer

Balance: *Number of Nodes* vs. *Computation per Node*



K_r-completions

Consider searching for uniquely K_r-saturated graphs.

K_r-completions

Consider searching for uniquely K_r-saturated graphs.

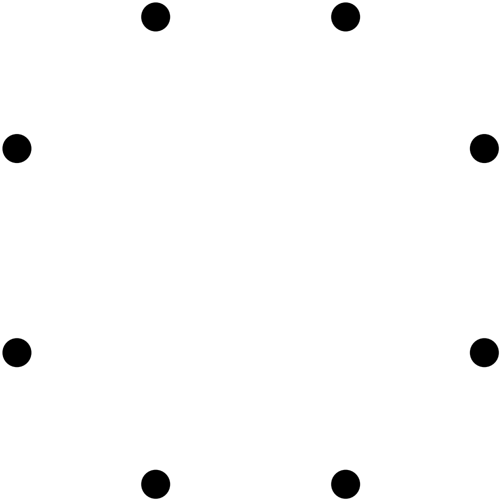
Every non-edge requires a K_r^- *completion*.

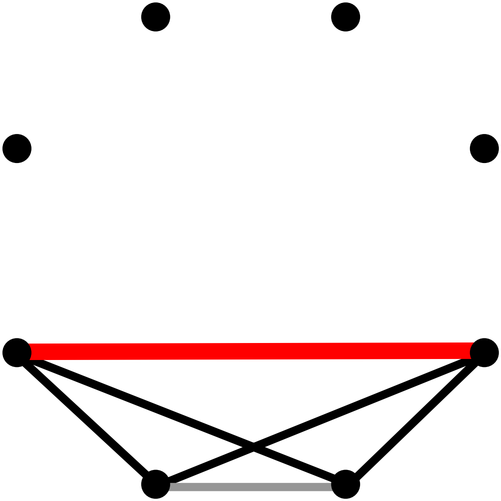
K_r-completions

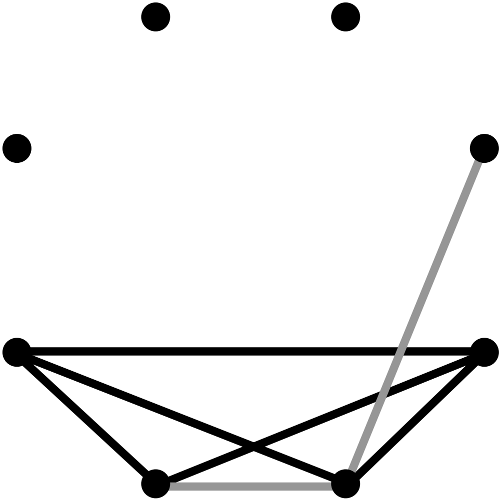
Consider searching for uniquely K_r-saturated graphs.

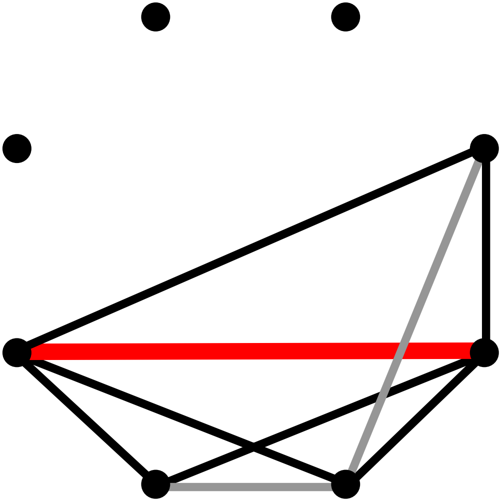
Every non-edge requires a K_r^- *completion*.

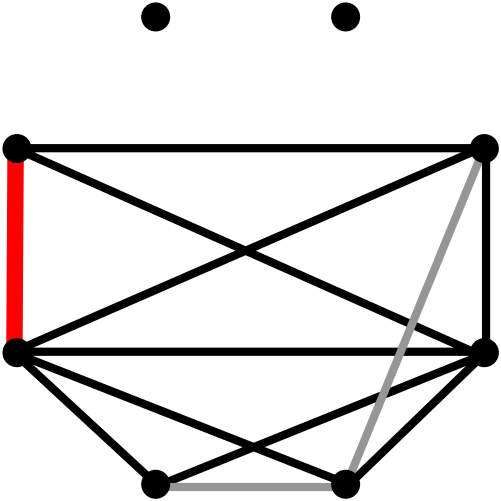
Augmentation: Pick a vertex pair to be "completed" non-edge, also select where to place the K_r^- completion.

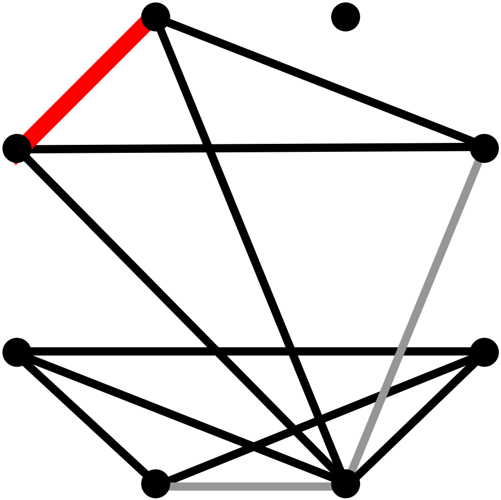












Canonical Non-Edge

A “deletion” requires picking a canonical completed non-edge.

Canonical Non-Edge

A “deletion” requires picking a canonical completed non-edge.

Remove all edges which came from that edge’s completion.

Canonical Non-Edge

A "deletion" requires picking a canonical completed non-edge.

Remove all edges which came from that edge's completion.

But only if they don't appear in another completion!

Canonical Non-Edge

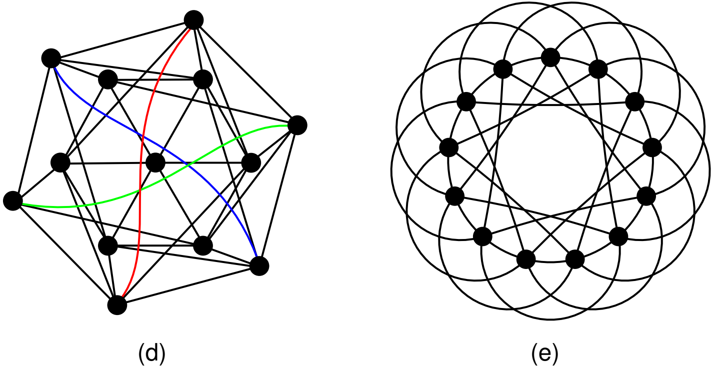
A “deletion” requires picking a canonical completed non-edge.

Remove all edges which came from that edge’s completion.

But only if they don’t appear in another completion!

Extra Step: Try filling all open pairs with edges.

Results



Joint with Stephen G. Hartke ($N = 13$)

The only uniquely K_4-saturated graph on 14 vertices is the 2-book with 12 pages.

To learn more...

- B. D. McKay. Isomorph-free exhaustive generation.
- B. D. McKay. Small graphs are reconstructible.
- D. Stolee. Isomorph-free generation of 2-connected graphs with applications.
- D. Stolee. Generating *p*-extremal graphs.
- 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.

The canonical augmentation method

Derrick Stolee
University of Nebraska–Lincoln
s-dstolee1@math.unl.edu
http://www.math.unl.edu/∼s-dstolee1/

May 13, 2011

[1]Supported by NSF grants CCF-0916525 and DMS-0914815.