An Exact Branch and Bound Algorithm

with

Symmetry Breaking

for the

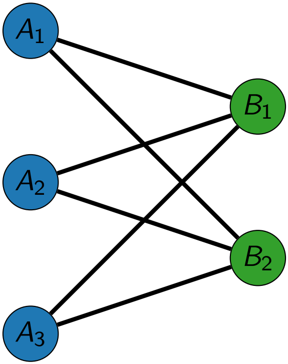
Maximum Balanced Induced Biclique Problem

Ciaran McCreesh   Patrick Prosser

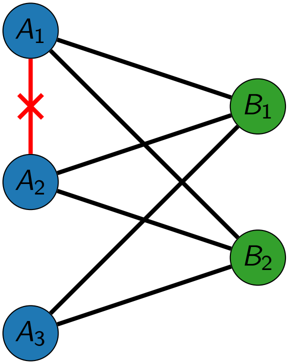
February 11, 2014

Maximum Balanced Induced Bicliques

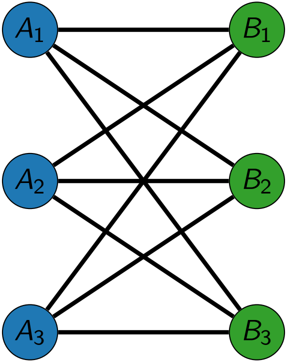
Bicliques



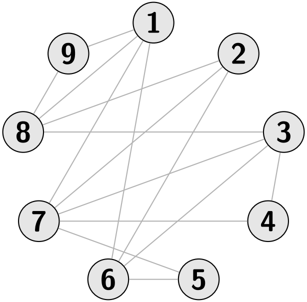
Induced Bicliques



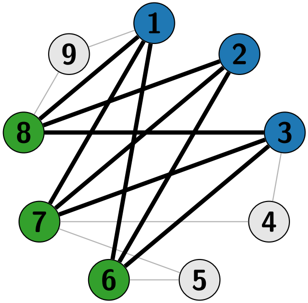
Balanced Bicliques



The Maximum Balanced Induced Biclique Problem



The Maximum Balanced Induced Biclique Problem



Existing Results

Complexity

- NP-hard, even in a bipartite graph (Garey and Johnson).

Other Biclique Variants

- Maximum vertex non-induced biclique:
  - Trivially useless.
- Maximum vertex biclique in a bipartite graph:
  - Easy (König's theorem and bipartite matching).
  - Corollary: maximum clique for a union of two cliques is easy.
- Maximum vertex induced biclique in an arbitrary graph:
  - NP-hard
  - Applications in data mining.
- Maximum edge induced biclique in a bipartite graph:
  - NP-hard
  - Applications in data mining.

Applications

?

Why Care?

- Interesting algorithmic properties:
  - Non-hereditary, but still reasonably well-behaved.
  - We have a good bound.
  - One simple symmetry.

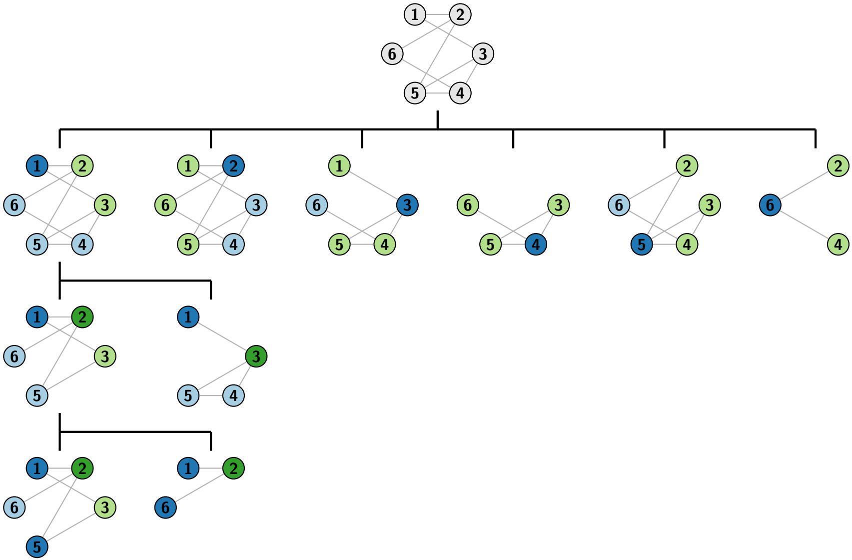
Our Algorithm

Inspiration

- Maximum clique algorithms by Tomita et al.
- Bitset encodings by San Segundo et al.
    - A speedup of between two and twenty for maximum clique.

Branch...

- Recursively grow two compatible independent sets, A and B.
- Have two candidate sets, P_a and P_b.
- Recursively expand:
  - Pick a vertex v from P_a, add it to A.
  - So we must remove adjacent vertices from P_a, and non-adjacent vertices from P_b.
  - Now recurse, swapping the roles of A and B.
  - Then consider removing v from A and P_a.


1
2
6
3
5
4

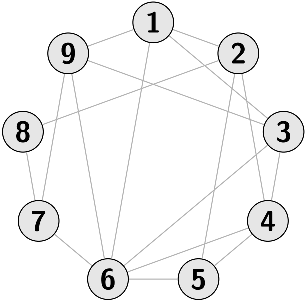
. . . and Bound

- Keep track of the best solution found so far, (A_{max}, B_{max}). We call this the *incumbent*.
- Careful! The balance condition means feasibility is not quite hereditary. At leaf nodes, either $|A| = |B|$ or $|A| = |B| + 1$.
- If $|A| + |P_a| \leq |A_{max}|$, or $|B| + |P_b| \leq |B_{max}|$, then we cannot unseat the incumbent, so we backtrack.
- A much better bound can be found using clique covers.

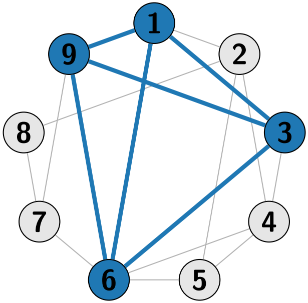
A Bound using Clique Covers

- If we can colour a graph using k colours, it cannot contain a clique with more than k vertices (each vertex in a clique must be given a different colour).
- Dually, if we can cover a graph using k cliques, its independence number is at most k.

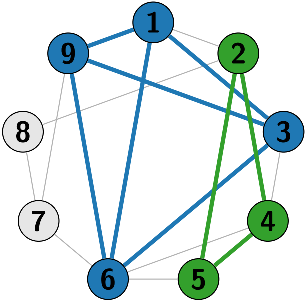
A Bound using Clique Covers



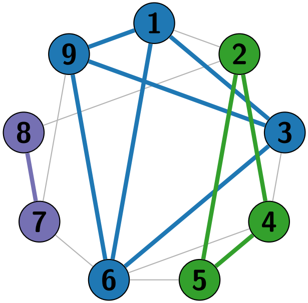
A Bound using Clique Covers



A Bound using Clique Covers



A Bound using Clique Covers



A Bound using Clique Covers

- We use a greedy clique cover.
- Vertices are permuted at the top of search, for a static variable ordering.
- We only need to perform one clique cover per recursive call, not one per vertex selection.

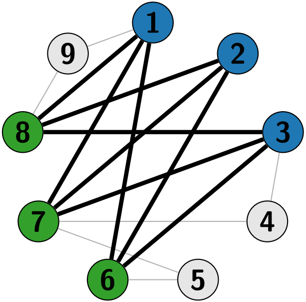
Dealing with Bipartite Graphs

- This bound knows about independent sets on each side, but not about compatibility.
- This bound is useless if the graph is bipartite, or becomes bipartite during search.
- We can detect this: a greedy clique cover uses k cliques iff the input is an independent set.
- Open problem: find a bound for this case which is both useful and quick to compute.

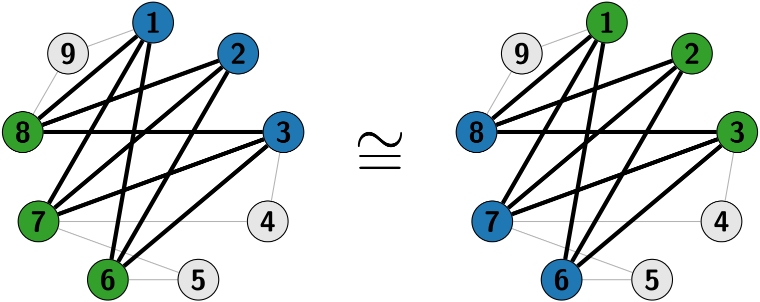
Symmetries

$$(A, B) \cong (B, A)$$

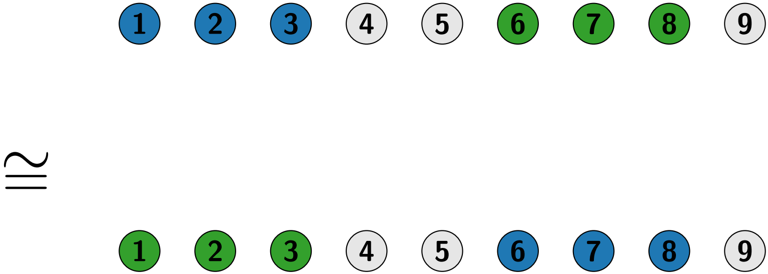
Symmetries



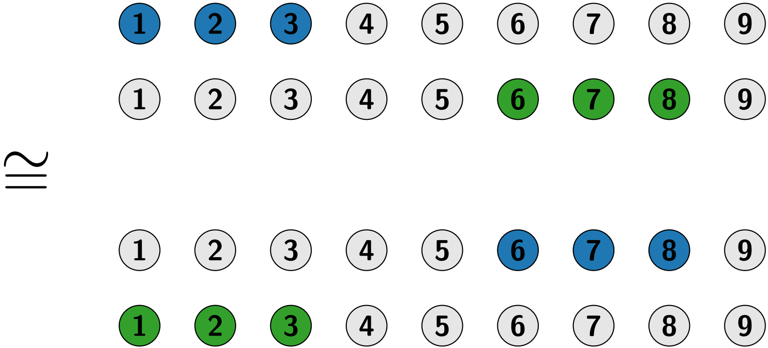
Symmetries



Excluding Symmetries



Excluding Symmetries



Excluding Symmetries

1	1	1	0	0	0	0	0	0
0	0	0	0	0	1	1	1	0

$\cong$

0	0	0	0	0	1	1	1	0
1	1	1	0	0	0	0	0	0

Excluding Symmetries

$$\begin{array}{ccccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$\geq_{lex}$

$$\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}$$

$\cong$

$$\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}$$

$\leq_{lex}$

$$\begin{array}{ccccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Excluding Symmetries using Lex

- Idea: only find solutions (A, B) where $A \geq_{lex} B$.
  - Don't swap the roles of A and B when recursing for the purposes of this test.
- We remove half of the solutions (not half of the search space).
- If we can prove that $B \geq_{lex} A$ must hold based upon the decisions made so far, backtrack.
- The most significant set bit in A must be more significant than the most significant bit set in B.
- If the first k bits of A are zero, then the first k bits of B must be zero.

Excluding Symmetries: What Could Possibly Go Wrong?

- We may have to explore deep into the search tree before the rule kicks in: so long as the most significant bit is undecided, we can't filter anything.
- Worse, we may exclude a solution which we would otherwise find quickly.

Excluding Symmetries, Second Attempt

- We have fixed an arbitrary order for the bits. This order may not be the same as the decision order.
- Idea: allow the algorithm to select the arbitrary order for the lex comparison.
- So we select the most significant bit first.
- When we reject a vertex v from A, if B is empty, then reject v from P_b.

Excluding Symmetries, with Two Lines of Code

`expand` :: (Graph G, Set A, Set B, Set P_a, Set P_b, Set A_{max}, Set B_{max})

begin
  $(bounds, order) \leftarrow$ `cliqueSort`(G, P_a)
  for $i \leftarrow |P_a|$ **downto** 1 **do**
    if $bounds[i] + |A| > |A_{max}|$ **and** $|P_b| + |B| > |B_{max}|$ **then**
      $v \leftarrow order[i]$
      $A \leftarrow A \cup \{v\}$ // Consider $v \in A$
      $P_a \leftarrow P_a \setminus \{v\}$
      $P'_a \leftarrow P_a \cap \overline{\mathrm{N_G}(v)}$ // Remove vertices adjacent to v
      $P'_b \leftarrow P_b \cap \mathrm{N_G}(v)$ // Remove vertices not adjacent to v
      if $|A| = |B|$ **and** $|A| > |A_{max}|$ **then**
        $(A_{max}, B_{max}) \leftarrow (A, B)$ // We've found a better solution
      if $P'_b \neq \emptyset$ **then**
        `expand`$(G, B, A, P'_b, P'_a, B_{max}, A_{max})$ // Swap and recurse
      $A \leftarrow A \setminus \{v\}$ // Now consider $v \notin A$
      if $B = \emptyset$ **then**
        $P_b \leftarrow P_b \setminus \{v\}$ // Avoid symmetric solutions

Excluding Symmetries, with Two Lines of Code

```
expand :: (Graph G, Set A, Set B, Set Pa, Set Pb, Set Amax, Set Bmax)
begin
    (bounds, order) ← cliqueSort(G, Pa)
    for i ← |Pa| downto 1 do
        if bounds[i] + |A| > |Amax| and |Pb| + |B| > |Bmax| then
            v ← order[i]
            A ← A ∪ {v}                                  // Consider v ∈ A
            Pa ← Pa \ {v}
            P'a ← Pa ∩ ‾N_G(v)‾                          // Remove vertices adjacent to v
            P'b ← Pb ∩ N_G(v)                            // Remove vertices not adjacent to v
            if |A| = |B| and |A| > |Amax| then
                (Amax, Bmax) ← (A, B)                    // We've found a better solution
            if P'b ≠ ∅ then
                expand(G, B, A, P'b, P'a, Bmax, Amax)    // Swap and recurse
            A ← A \ {v}                                  // Now consider v ∉ A
            if B = ∅ then
                Pb ← Pb \ {v}                            // Avoid symmetric solutions
```

Excluding Symmetries, with Two Lines of Code

expand :: (Graph G, Set A, Set B, Set P_a, Set P_b, Set A_{max}, Set B_{max})

begin
  $(bounds, order) \leftarrow$ cliqueSort(G, P_a)
  for $i \leftarrow |P_a|$ **downto** 1 **do**
    if $bounds[i] + |A| > |A_{max}|$ **and** $|P_b| + |B| > |B_{max}|$ **then**
      $v \leftarrow order[i]$
      $A \leftarrow A \cup \{v\}$ // Consider $v \in A$
      $P_a \leftarrow P_a \setminus \{v\}$
      $P'_a \leftarrow P_a \cap \overline{N_G(v)}$ // Remove vertices adjacent to v
      $P'_b \leftarrow P_b \cap N_G(v)$ // Remove vertices not adjacent to v
      if $|A| = |B|$ **and** $|A| > |A_{max}|$ **then**
        $(A_{max}, B_{max}) \leftarrow (A, B)$ // We've found a better solution
      if $P'_b \neq \emptyset$ **then**
        expand$(G, B, A, P'_b, P'_a, B_{max}, A_{max})$ // Swap and recurse
      $A \leftarrow A \setminus \{v\}$ // Now consider $v \notin A$
      if $B = \emptyset$ **then**
        $P_b \leftarrow P_b \setminus \{v\}$ // Avoid symmetric solutions

Excluding Symmetries, with Two Lines of Code

expand :: (Graph G, Set A, Set B, Set P_a, Set P_b, Set A_{max}, Set B_{max})

begin
  $(bounds, order) \leftarrow$ cliqueSort(G, P_a)
  for $i \leftarrow |P_a|$ **downto** 1 **do**
    if $bounds[i] + |A| > |A_{max}|$ **and** $|P_b| + |B| > |B_{max}|$ **then**
      $v \leftarrow order[i]$
      $A \leftarrow A \cup \{v\}$ // Consider $v \in A$
      $P_a \leftarrow P_a \setminus \{v\}$
      $P'_a \leftarrow P_a \cap \overline{N_G(v)}$ // Remove vertices adjacent to v
      $P'_b \leftarrow P_b \cap N_G(v)$ // Remove vertices not adjacent to v
      if $|A| = |B|$ **and** $|A| > |A_{max}|$ **then**
        $(A_{max}, B_{max}) \leftarrow (A, B)$ // We've found a better solution
      if $P'_b \neq \emptyset$ **then**
        expand$(G, B, A, P'_b, P'_a, B_{max}, A_{max})$ // Swap and recurse
      $A \leftarrow A \setminus \{v\}$ // Now consider $v \notin A$
      if $B = \emptyset$ **then**
        $P_b \leftarrow P_b \setminus \{v\}$ // Avoid symmetric solutions

Excluding Symmetries, with Two Lines of Code

expand :: (Graph G, Set A, Set B, Set P_a, Set P_b, Set A_{max}, Set B_{max})

begin
- $(bounds, order) \leftarrow$ `cliqueSort`(G, P_a)
- **for** $i \leftarrow |P_a|$ **downto** 1 **do**
  - **if** $bounds[i] + |A| > |A_{max}|$ **and** $|P_b| + |B| > |B_{max}|$ **then**
    - $v \leftarrow order[i]$
    - $A \leftarrow A \cup \{v\}$ — `// Consider` $v \in A$
    - $P_a \leftarrow P_a \setminus \{v\}$
    - $P'_a \leftarrow P_a \cap \overline{N_G(v)}$ — `// Remove vertices adjacent to` v
    - $P'_b \leftarrow P_b \cap N_G(v)$ — `// Remove vertices not adjacent to` v
    - **if** $|A| = |B|$ **and** $|A| > |A_{max}|$ **then**
      - $(A_{max}, B_{max}) \leftarrow (A, B)$ — `// We've found a better solution`
    - **if** $P'_b \neq \emptyset$ **then**
      - `expand`$(G, B, A, P'_b, P'_a, B_{max}, A_{max})$ — `// Swap and recurse`
    - $A \leftarrow A \setminus \{v\}$ — `// Now consider` $v \notin A$
    - **if** $B = \emptyset$ **then**
      - $P_b \leftarrow P_b \setminus \{v\}$ — `// Avoid symmetric solutions`

Excluding Symmetries, with Two Lines of Code

```
expand :: (Graph G, Set A, Set B, Set P_a, Set P_b, Set A_max, Set B_max)
begin
    (bounds, order) ← cliqueSort(G, P_a)
    for i ← |P_a| downto 1 do
        if bounds[i] + |A| > |A_max| and |P_b| + |B| > |B_max| then
            v ← order[i]
            A ← A ∪ {v}                                  // Consider v ∈ A
            P_a ← P_a \ {v}
            P'_a ← P_a ∩ overline(N_G(v))                // Remove vertices adjacent to v
            P'_b ← P_b ∩ N_G(v)                          // Remove vertices not adjacent to v
            if |A| = |B| and |A| > |A_max| then
                (A_max, B_max) ← (A, B)                  // We've found a better solution
            if P'_b ≠ ∅ then
                expand(G, B, A, P'_b, P'_a, B_max, A_max) // Swap and recurse
            A ← A \ {v}                                  // Now consider v ∉ A
            if B = ∅ then
                P_b ← P_b \ {v}                          // Avoid symmetric solutions
```

Excluding Symmetries, with Two Lines of Code

expand :: (Graph G, Set A, Set B, Set P_a, Set P_b, Set A_{max}, Set B_{max})

begin
- $(bounds, order) \leftarrow \texttt{cliqueSort}(G, P_a)$
- **for** $i \leftarrow |P_a|$ **downto** 1 **do**
  - **if** $bounds[i] + |A| > |A_{max}|$ **and** $|P_b| + |B| > |B_{max}|$ **then**
    - $v \leftarrow order[i]$
    - $A \leftarrow A \cup \{v\}$ — // Consider $v \in A$
    - $P_a \leftarrow P_a \setminus \{v\}$
    - $P'_a \leftarrow P_a \cap \overline{\mathrm{N}_G(v)}$ — // Remove vertices adjacent to v
    - $P'_b \leftarrow P_b \cap \mathrm{N}_G(v)$ — // Remove vertices not adjacent to v
    - **if** $|A| = |B|$ **and** $|A| > |A_{max}|$ **then**
      - $(A_{max}, B_{max}) \leftarrow (A, B)$ — // We've found a better solution
    - **if** $P'_b \neq \emptyset$ **then**
      - $\texttt{expand}(G, B, A, P'_b, P'_a, B_{max}, A_{max})$ — // Swap and recurse
    - $A \leftarrow A \setminus \{v\}$ — // Now consider $v \notin A$
    - **if** $B = \emptyset$ **then**
      - $P_b \leftarrow P_b \setminus \{v\}$ — // Avoid symmetric solutions

The Rest of the Algorithm

`improvedBiclique` :: (Graph G) $\rightarrow$ (Set **of** Integer, Set **of** Integer)
begin
    $(A_{max}, B_{max}) \leftarrow (\emptyset, \emptyset)$
    permute G so that the vertices are in non-increasing degree order
    `expand`$(G, \emptyset, \emptyset, \mathrm{V}(G), \mathrm{V}(G), A_{max}, B_{max})$
    return (A_{max}, B_{max}) (unpermuted)

`cliqueSort` :: (Graph G, Set P) $\rightarrow$ (Array **of** Integer, Array **of** Integer)
begin
    bounds $\leftarrow$ an Array **of** Integer
    order $\leftarrow$ an Array **of** Integer
    $P' \leftarrow P$ `// vertices yet to be allocated`
    $k \leftarrow 1$ `// current clique number`
    while $P' \neq \emptyset$ **do**
        $Q \leftarrow P'$ `// vertices to consider for the current clique`
        while $Q \neq \emptyset$ **do**
            $v \leftarrow$ the first element of Q `// get next vertex to allocate`
            $P' \leftarrow P' \setminus \{v\}$
            $Q \leftarrow Q \cap \mathrm{N}(G, v)$ `// remove non-adjacent vertices`
            append k to *bounds*
            append v to *order*
        $k \leftarrow k + 1$ `// start a new clique`
    return (*bounds*, *order*)

The Rest of the Algorithm

```
improvedBiclique :: (Graph G) → (Set of Integer, Set of Integer)
begin
    (A_max, B_max) ← (∅, ∅)
    permute G so that the vertices are in non-increasing degree order
    expand(G, ∅, ∅, V(G), V(G), A_max, B_max)
    return (A_max, B_max) (unpermuted)

cliqueSort :: (Graph G, Set P) → (Array of Integer, Array of Integer)
begin
    bounds ← an Array of Integer
    order ← an Array of Integer
    P′ ← P                                  // vertices yet to be allocated
    k ← 1                                   // current clique number
    while P′ ≠ ∅ do
        Q ← P′                              // vertices to consider for the current clique
        while Q ≠ ∅ do
            v ← the first element of Q      // get next vertex to allocate
            P′ ← P′ \ {v}
            Q ← Q ∩ N(G, v)                 // remove non-adjacent vertices
            append k to bounds
            append v to order
        k ← k + 1                           // start a new clique
    return (bounds, order)
```

The Rest of the Algorithm

```
improvedBiclique :: (Graph G) → (Set of Integer, Set of Integer)
begin
    (A_max, B_max) ← (∅, ∅)
    permute G so that the vertices are in non-increasing degree order
    expand(G, ∅, ∅, V(G), V(G), A_max, B_max)
    return (A_max, B_max) (unpermuted)

cliqueSort :: (Graph G, Set P) → (Array of Integer, Array of Integer)
begin
    bounds ← an Array of Integer
    order ← an Array of Integer
    P′ ← P                                   // vertices yet to be allocated
    k ← 1                                    // current clique number
    while P′ ≠ ∅ do
        Q ← P′                               // vertices to consider for the current clique
        while Q ≠ ∅ do
            v ← the first element of Q       // get next vertex to allocate
            P′ ← P′ \ {v}
            Q ← Q ∩ N(G, v)                  // remove non-adjacent vertices
            append k to bounds
            append v to order
        k ← k + 1                            // start a new clique
    return (bounds, order)
```

Results

Results

- We can solve all but four DIMACS problems in under a day.
- Usually, but not always, easier than maximum clique.
- Usually, but not always, easier than maximum independent set.
- Large sparse graphs with $|V| > 15,000$ and $|E| > 250,000$ take under 20 seconds.
- Excluding symmetries gains us between 0% and 50%.

Results

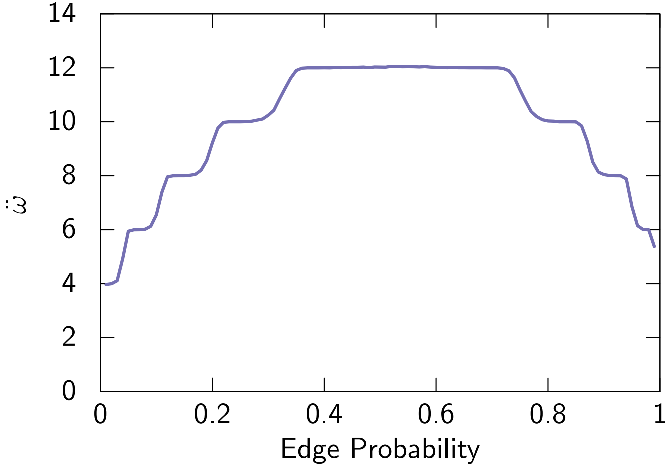
- We can solve all but four DIMACS problems in under a day.
- Usually, but not always, easier than maximum clique.
- Usually, but not always, easier than maximum independent set.
- Large sparse graphs with $|V| > 15,000$ and $|E| > 250,000$ take under 20 seconds.
- Excluding symmetries gains us between 0% and 50%.
    - Unless you look closely...
    - The bound function can get worse for subproblems ("misleading"), and is not invariant under isomorphism ("evil"). Occasionally this gives wild results.

Future Work

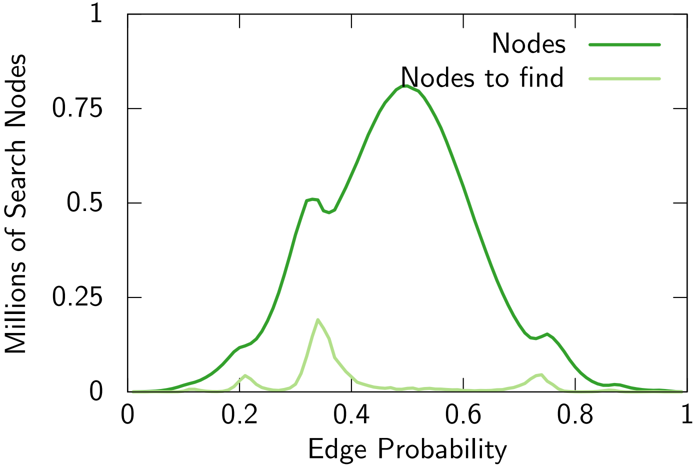
Is an Algorithm Worth It?

- A naïve constraint programming model is easy, but slow.
- What about a better constraint programming model?
- What about MIP?

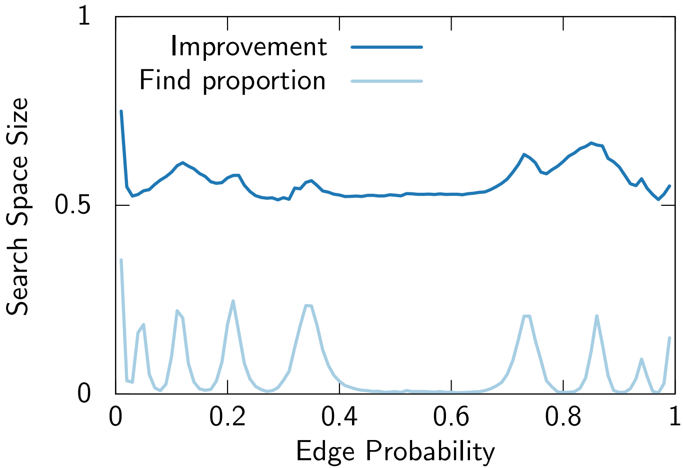
Random Graphs G(250, x)



Difficulty of G(250, x)



Effects of Symmetry Exclusion in G(250, x)



Parallel Branch and Bound

- For maximum clique, parallel branch and bound typically gives us close to linear speedups, and sometimes much better.
- We do the same here. But how do symmetries interact with parallelism?

http://dcs.gla.ac.uk/~ciaran
c.mccreesh.1@research.gla.ac.uk