Almost-Symmetries of Graphs

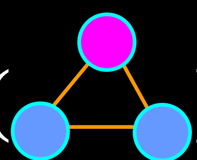
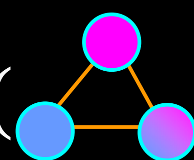
Igor Markov, University of Michigan

Motivation 1: Want More Symmetries

- Symmetries in CP are well-defined
- We know
  - How to represent symmetries
  - How to find them
    (or at least expect them to be given)
  - How to use them
- We know they help & want more of them
- Idea: relax the notion of symmetry
  - "Slightly defective" symmetries

Motivation 2: Need More Than Symmetries

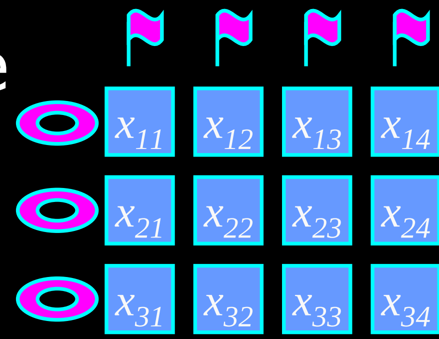
- Existing symmetry-based techniques do not handle special-casing well
  - They map variables to variables, values to values <u>in all cases</u>
- Obstacle to extensions: loss of transitivity

$Sym($  $) = S_2$ $Sym($  $) = ?$

Desiderata for Almost-symmetries

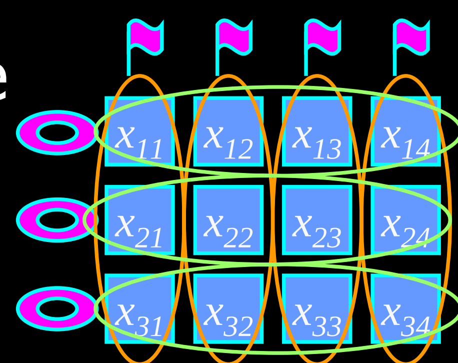
- Conceptually similar to symmetries
- As easy to capture
- As easy to find
  (or get someone write them down for you!)
- As easy to use in symmetry-breaking
- As helpful (computationally)
- More numerous than symmetries

The Pigeonhole Principle



- Cannot place *N* objects into *N-1* slots
- Boolean formulation
  - a matrix of *N(N-1)* 0-1 variables
  - must assign each object to 1+ slot
  - no two objects can be in the same slot

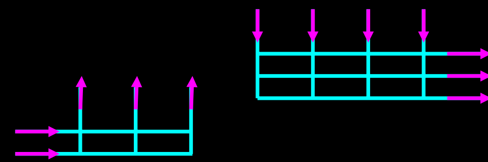
The Pigeonhole Principle



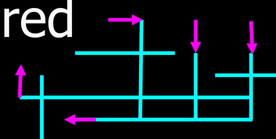
- Cannot place *N* objects into *N-1* slots
- Boolean formulation
  - a matrix of *N(N-1)* 0-1 variables
  - must assign each object to 1+ slot
  - no two objects can be in the same slot
- Symmetries help (1985, 1996, 2003)
- A fundamental challenging problem

  … or a solved template ?

PHP in Wire Routing



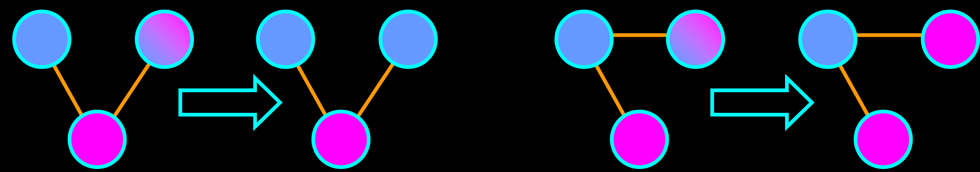
- Problem instances are formulated essentially in Boolean terms
  - each wire must be routed in some way
  - no two wires can use the same track
- Using symmetry is critical
- Many instances of PHP appear as sub-instances of larger satisfiable instances
  - Syntactic symmetry is often obscured
  - Conditional, local symmetries, etc



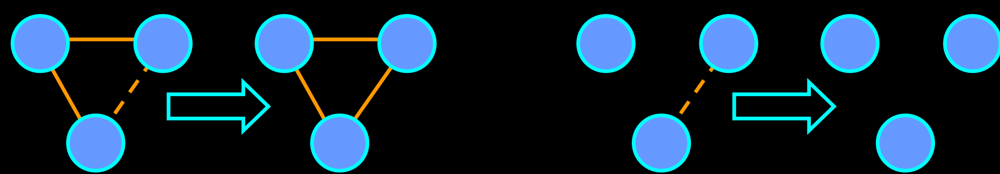
Almost-automorphisms of Graphs

- Automorphism: *a vertex permutation that preserves edges and vertex colors*
- Try to "almost preserve" edges & colors
  - Some edges can map to non-edges
  - Some vertices can map to wrong colors
  - How do we quantify, limit "some" ?
- Use a "chameleon color" (variables) for vertices
  - Just like an * in regular expressions
  - Just like don't-cares in Boolean functions & circuits
- Similarly for edges

Chameleon Vertices and Edges

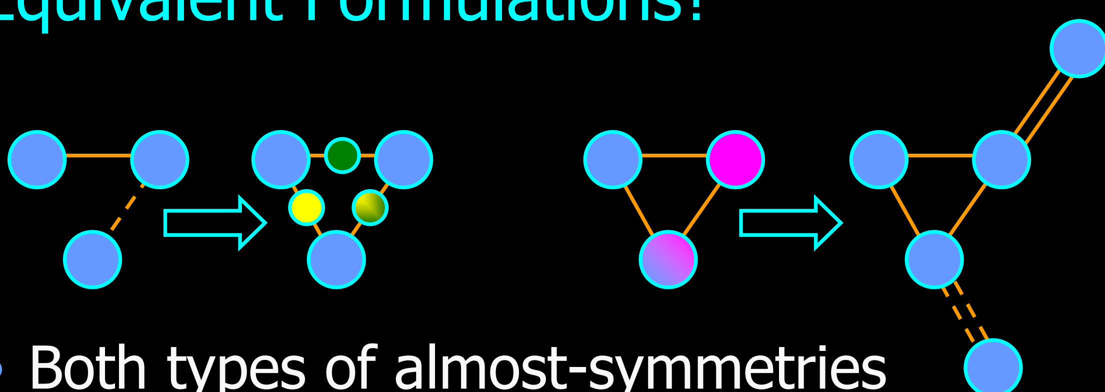


- Chameleon vertices must assume specific colors so as to enable symmetries



- Chameleon edges must decide whether they exist or not, so as to enable symms

Equivalent Formulations?

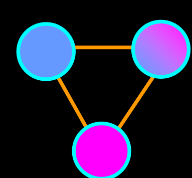


- Both types of almost-symmetries must carry the same algebraic structure
- Conversion overhead: $O(V^3+E^3)$
  - Seems impractical
- Further analysis suggests developing two independent (but compatible) techniques

Structure in Almost-Symmetries

- To deal with graph automorphisms one uses their *group structure*
  - Provably exponential compression: N automorphisms always captured by $\leq log_2 N$ generators
  - Efficient set-like operations ($\cap$, $\in$, etc)
  - Stabilizer-chain algorithms
  - Very fast graph-automorphism algorithms
- Almost-automorphisms do not form
  - Groups, semi-groups, monoids, grouppoids

Structure in Almost-Automorphisms

$Aut($  $)=S_2 \cup S_2$

- Clear all colors
  - The resulting Aut( ) contains all almost-autos
- Paint chameleons using a new/unique color
  - The resulting Aut( ) is contained in every G_i
- Consider all combos of chameleon colorings
  - For each coloring, can find Aut( )
  - Almost-autos form a union of subgroups $\cup G_i$
- Can often find more compact $\cup G_i$ expressions
  - Many G_i can be trivial, equal, or subgroups of G_k
  - <u>Worst case is exponential</u> (see Appendix A)

Capturing Almost-Automorphisms

- To capture $\cup G_i$
  - Capture each group by a list of generators
- An algorithm for finding almost-symmetries should produce
  - Lists of lists of group generators
    (all lists are unordered)
- Simplified problem formulations
  - Find *a largest* subgroup G_i
    (can always express it compactly)
  - Detect cases when almost-autos form a group

Finding Almost-automorphisms

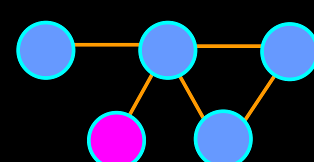
- Naïve algorithm
  - Iterate over all colorings of chameleon vertices
  - Call SAUCY for every coloring to find G_i
  - Discard redundant G_i
- May need to use GAP to compare groups
  - The same subgroup may be captured
    with different generating sets
    (can't just match lists)
  - For G and H, compute generators$(G \cap H)$, reuse them
- Observation
  - If the colorless graph has no symmetries,
    no need to branch on colors

Finding Almost-automorphisms (2)

- Research challenge:
  - Extend McKay's algorithm (NAUTY) or its derivatives (SAUCY) to solve the graph almost-automorphism problem
  - Preserve its performance in the traditional case (no chameleons)
- Seems doable!
  - Vertex-based case
  - Edge-based case

Basic Ideas For GAA Algorithms

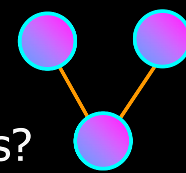
- McKay's GA algorithm interleaves branching with pruning (partition refinement)
  - Vertices with different degrees can't be mapped to each other
  - Ditto for different colors
  - At some point, we just have to try mapping similar vertices to each other
  - After such branching, we may be able to prune some more (partition refinement)



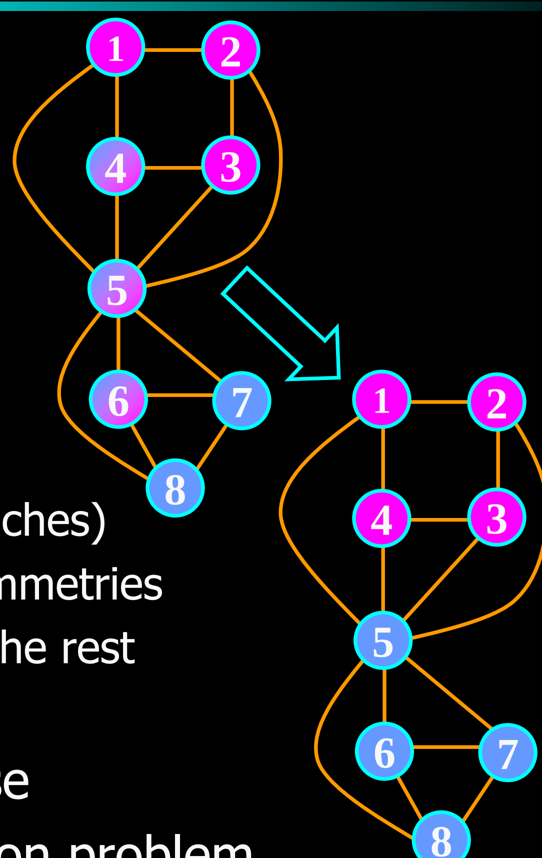
- Need new steps for color instantiation

Designing GAA Algorithms

- Naïve algorithm: tries all color instantiations first, then calls McKay
- Key idea 1: delay branching on colors
  - E.g., what if vertices have different degrees?
  - When branching, minimize further branching factor, apply partition refinement immediately
- Key idea 2: propagate colors early
  - If a class of potentially equivalent vertices only contains chameleons and pink vertices, make all chameleons pink (subgroup containment)
- Key idea 3: avoid branching via dominance
  - If a class of potentially equivalent vertices contains only chameleons, color all of them blue

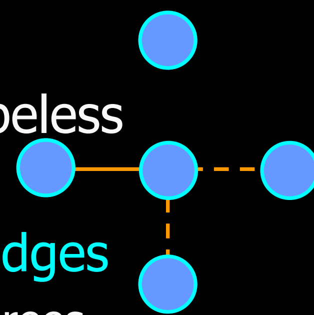


Example



- Vertex degrees: 3 & 7
  - Color vertex 5 blue
- Vertices 4 and 6 are in the same class, with blue and pink vertices
  - Branch on colors (4 branches)
  - Three branches yield symmetries
  - One of them subsumes the rest
- Almost-symmetries form a group in this case
- The symmetry-restoration problem

What about edge-based almost-syms?



- The same naïve algorithm works (branch on edges first), but is hopeless
  - We must delay branching
- Major problems with chameleon edges
  - Vertices have ranges of possible degrees and cannot always be split into classes
  - Partition refinement does not work anymore
- Solved in Appendix B
  - Vertex-range graph, two-step partition refinement

Static Almost-Symmetry-Breaking

- Almost-symmetries can be viewed as conditional symmetries
  - Symmetries with identical preconditions can be composed
  - Other symmetries may not be composable
- A. S.-B. Predicates must now include preconditions ($\Pi \Rightarrow \Sigma$)
  - Can now localize symmetries to sub-instances via *boundary conditions*

Conclusions

- Almost-symmetries can be understood and studied through graph modeling
- Vertex-based and edge-based cases
- Algebraic structure: union of subgroups $\cup G_i$
  - Represented by lists of lists of generators
- Computational challenge: finding compact hierarchies of generators for $\cup G_i$ when possible
  - Seems doable in both vertex- and edge-based cases
- Static almost-symmetry-breaking is fairly straightfoward