A programmable embedder

A staged approach for mapping problems to the Chimera graph

Marcus Daniels
<mdaniels@lanl.gov>

2016-09-29

LA-UR-16-27277

Embedding
Finding a minor in a bipartite graph

C : Calibration
M : Machine
i, j: members of an ordered set of qubit identifiers
σ: {-1,1}

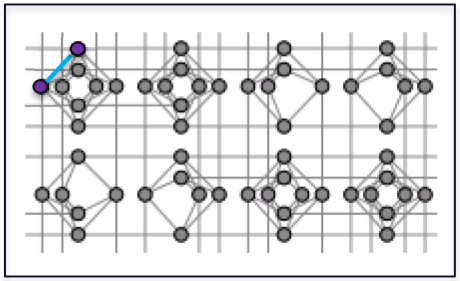
conn(C,i,j): Predicate that is true when a coupler may connect two qubits on a bipartite graph G(C).

Problem specification, e.g.

$$\sum_{i} h_i \sigma_i + \sum_{i<j} J_{ij}\sigma_i\sigma_j$$

$$\sum_{i} h_i \sigma_i + \sum_{conn(C(M),i,j)} J_{ij}\sigma_i\sigma_j$$

G(C)

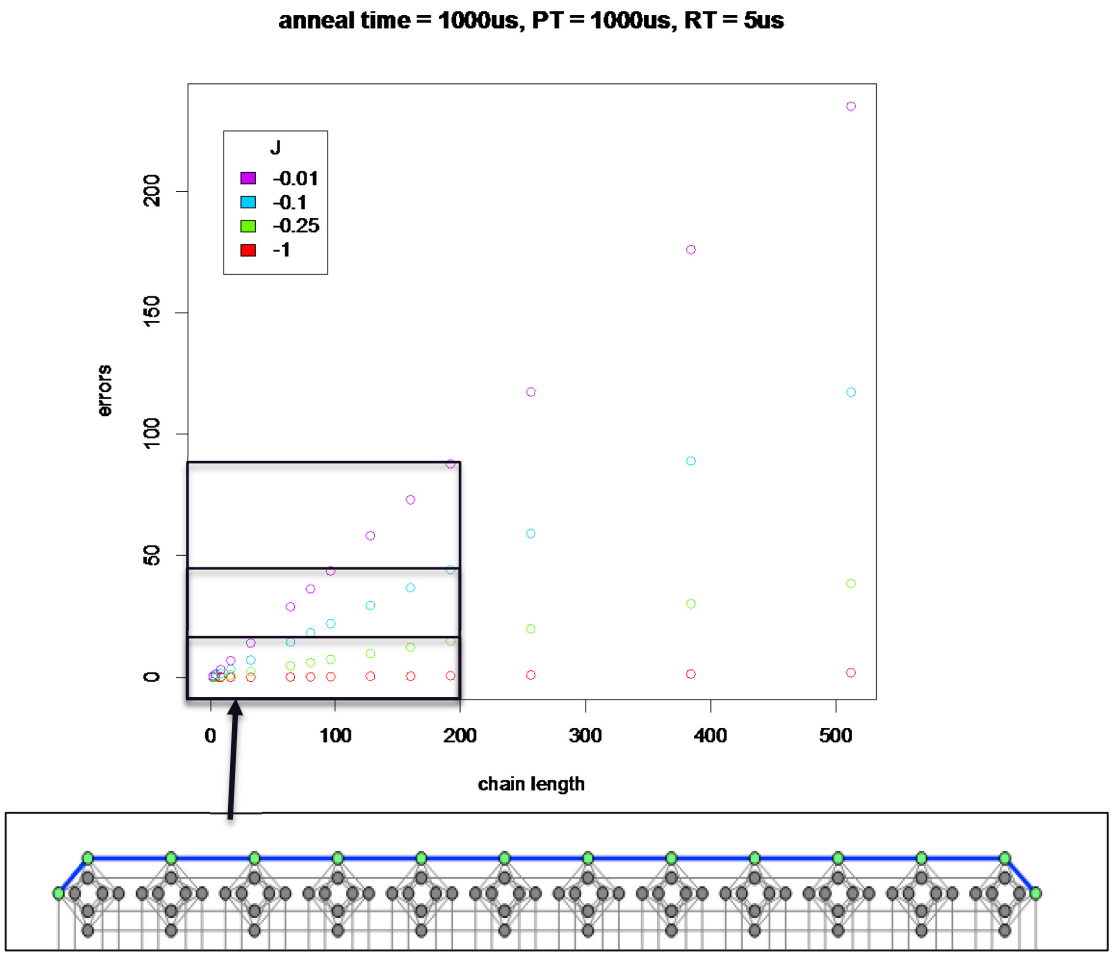


Goals

- **Control**
  - If a black box embedder fails, what then?
  - Understand why
  - Convey additional constraints and embedding tactics so that it won't fail.
- **Composability**
  - Programs or large models usually have different components.
  - Unit and system testing
    - Ground truth of small Ising systems compared to simulators (SA, QMC).
    - Do component embeddings behave the same on the D-Wave annealer when combined?
      - Longer wires
      - Magnetic fields in the vicinity
- **Decomposability**
  - Incrementally take apart a large embedding and put it back together
    - Can you fix your car or do you have to buy a new one?
- **Retrospective analysis**
  - Have a mechanism to generate many logically equivalent embeddings.
    - Evaluate them with D-Wave annealer, figure out why there are differences.

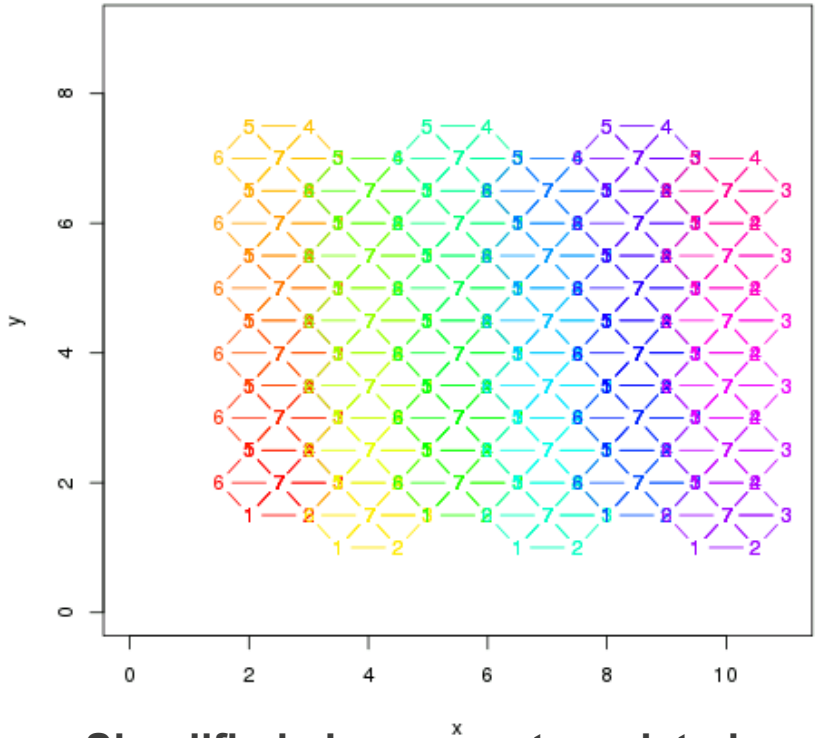
Why this fixation on control?

- **Accuracy is a bigger problem with an analog system**
  - Longer wires of qubits have proportionately more errors.

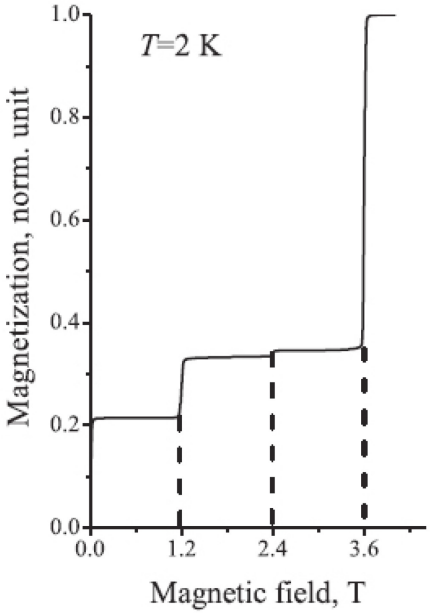


An example of an embedding: Metamagnetic $Ca_3Co_2O_6$

- Chains of face-sharing CoO_6 triangular prisms and octahedra with Ca ions between.



- Simplified above as a tesselated trigonal hexagons.
- Each integer-labeled vertex repels its neighbors.

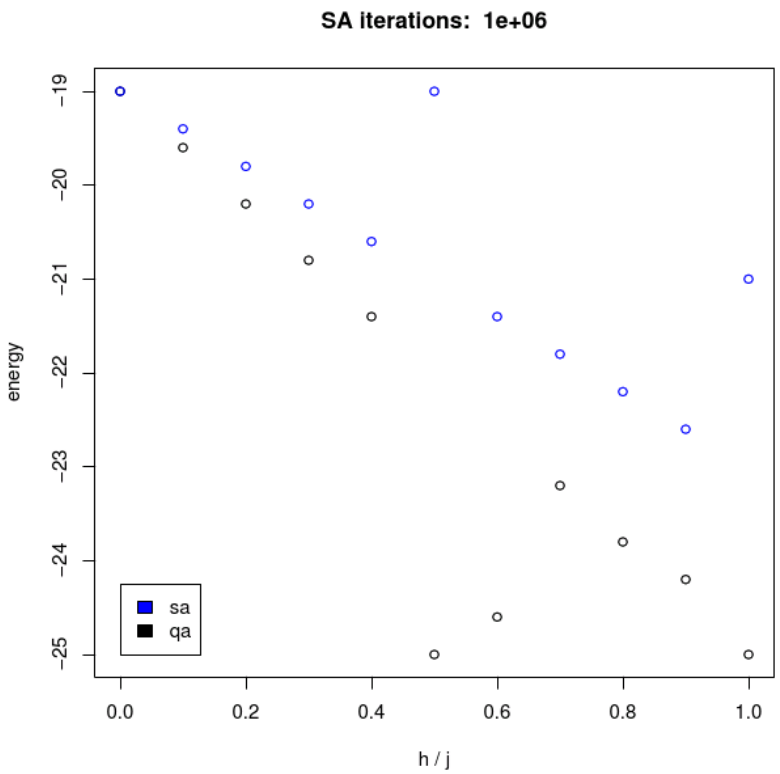


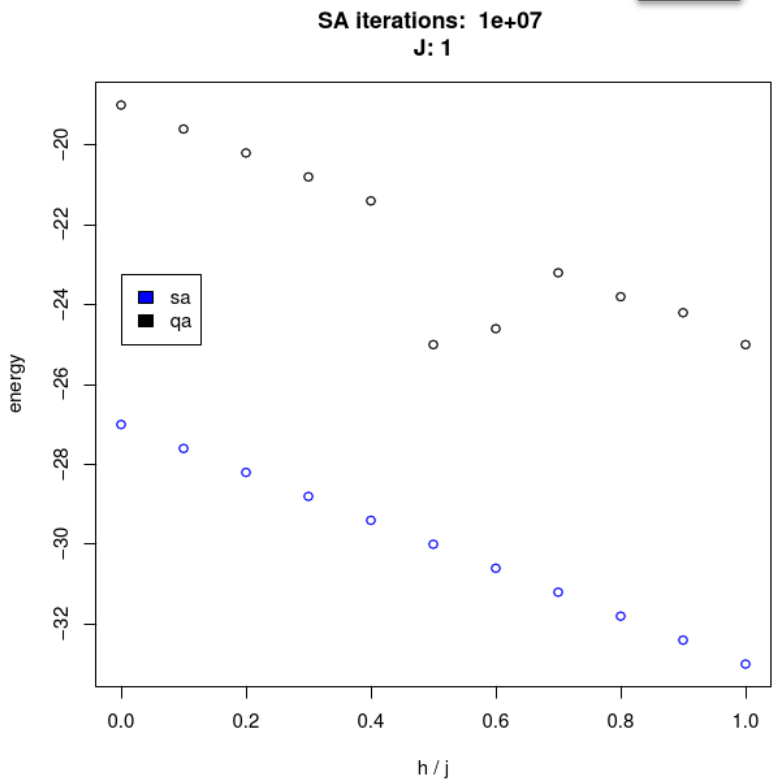
Magnetic moment shows jumps in field as an external field is applied.

Y. Udasov Phys. Rev. Lett., 96 (2006), p. 027212

Quantum annealing vs. simulated annealing

20 millisecond anneals (long)

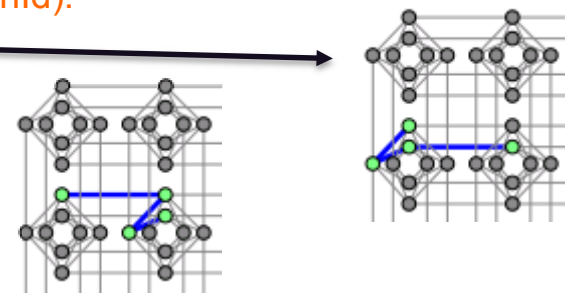




Approach

Multiscale indexing of Chimera graph shortest paths

- **Use shortest paths, period.**
  - Don't revise shortest paths as nodes on the graph are consumed.
- **For small regions on the Chimera graph, find many alternative shortest paths between all possible points up to a path cutoff length.**
  - distance(from, to, path, pathLength, alternativePathId, pathId).
  - distance(100,109,"100 96 101 109",4,0,587402).
  - distance(100,109,"100 97 101 109",4,1,587403).
  - distance(100,109,"100 98 101 109",4,2,587404).
  - distance(100,109,"100 99 101 109",4,3,587405).
  - distance(100,109,"100 108 104 109",4,4,587406).
  - 618,967 shortest paths



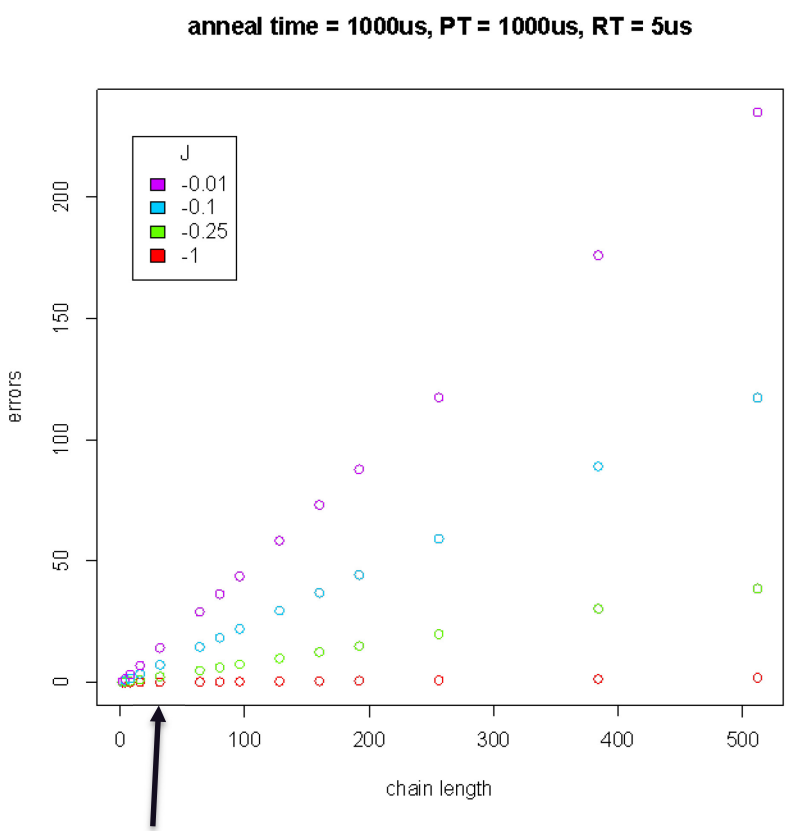
- **For larger regions on the Chimera graph, find fewer alternative shortest paths between all possible points up to a path cutoff length.**
  - distance(900,1024,"900 908 916 924 932 928 1024",7,0,5234881).
  - distance(900,1024,"900 896 901 909 917 925 933 928 1024",9,1,5234882).
  - distance(900,1024,"900 897 901 909 917 925 933 928 1024",9,2,5234883).
  - 1,595,355 shortest paths

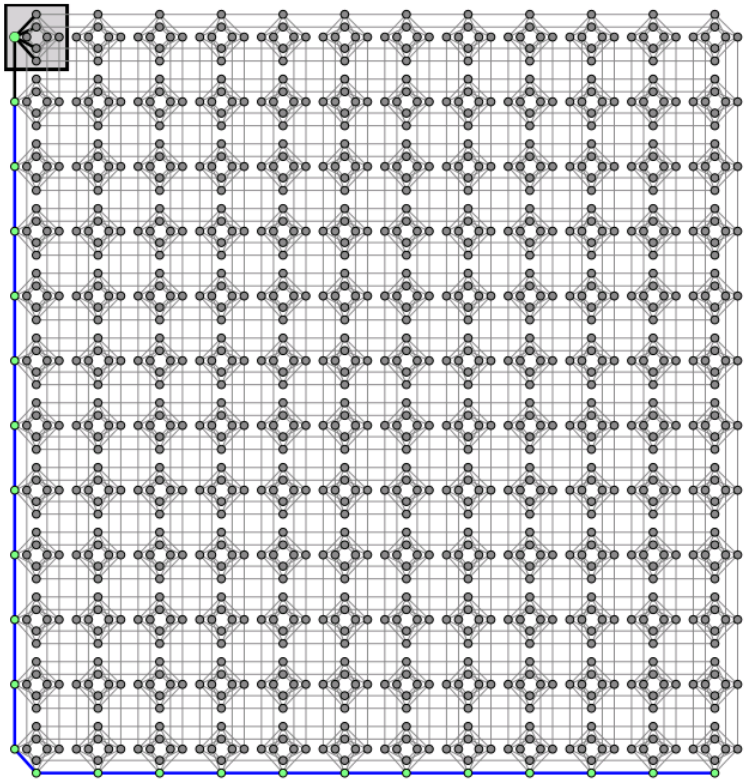
Note: Strings are just for this import format. One way of expressing facts is in fact tables (rather than a database), and lists can't be used there.

Longer paths

For the whole graph, find a shortest path of arbitrary length that connects each possible pair of working qubits.

1,208,900 shortest paths

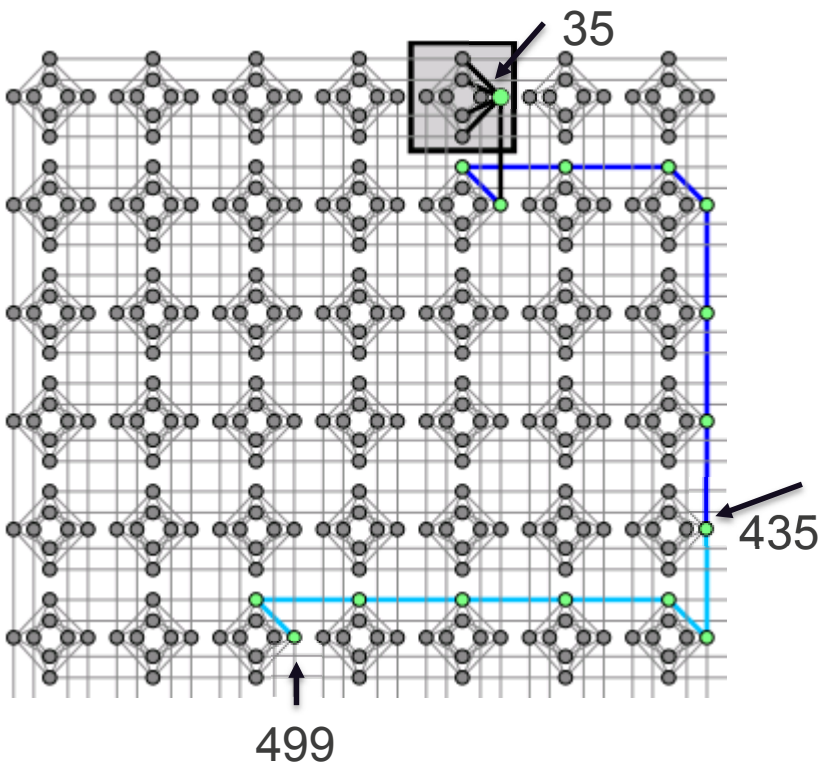




Pre-compute non-conflicting pairs of paths

sharedVertex(35,435,499,
"35 131 132 140 148 147 243 339 435",
"435 531 532 524 516 508 500 499",
196608,
2560255).

- **Meaning:**
  - There exist two **non-overlapping** shortest paths that join at qubit 435.



The first shortest path is 35 to 131 to 140 to 148 to 147 to 243 to 339 and 435.
Name this path with the unique integer 196608.

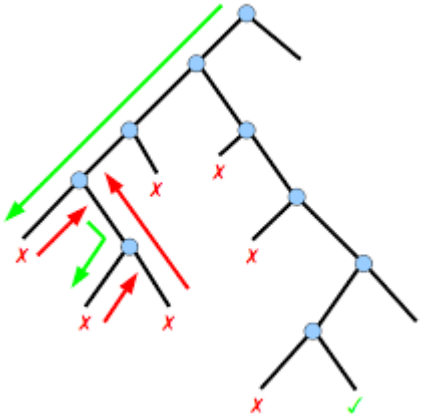
The second path continues from there to 531, 532, 524, 516, 508, 500 and 499.
Name this path with the unique integer 2560255.

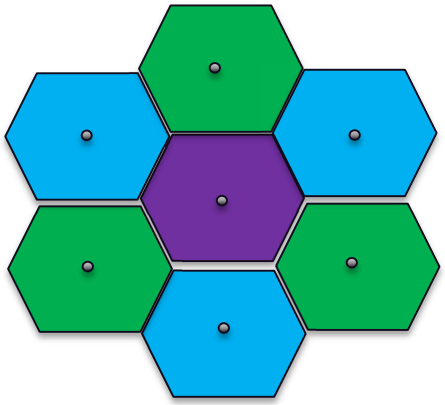
Index paths in a memory-mapped database

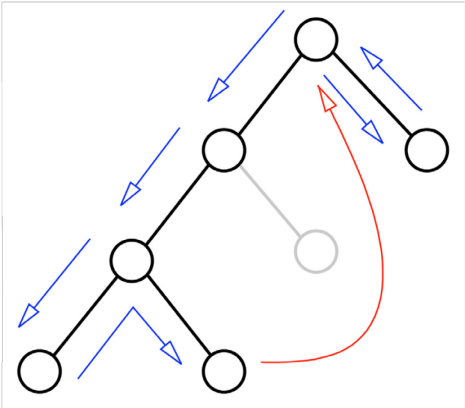
- **Balanced binary tree in a swap file**
  - *No recalculation of shortest path for users.*
  - – Rather than backing store from system swap, a mmap'ed file is used.
  - – Attach (mmap) and go – no loading
  - – File is stored in a tmpfs (a ramdisk) for one-time loading
  - – Database optimized for reads
- **Exhaustive indexing**
  - – Query any combination of qubits and get the appropriate set of records.
    - start/end -> middle qubit, and unique path ids
    - start/middle -> end, and pair of unique path ids
    - start/middle/end -> unique path ids
    - start/middle/end/path id -> unique path ids
    - paths from path identifiers
  - – Yes, it is large!
    - $ du –sh staged
      - 809G staged
    - Calculate on the fly with a GPU? https://www.blazegraph.com/ ?
      - High latency (50us) to query

Declarative [logic & constraint] programming

- **Say what must hold true in a solution, rather than how to accomplish it.**
  - qubit A is coupled to qubit B, B to C, and C to D, without naming what physical qubits A, B, C, or D correspond to.
    - Thus, solutions can take many configurations, e.g. a vertical, horizontal, or zig-zag line on the Chimera graph.
- **In a logic programming language like Mercury, solutions are found by depth-first search with backtracking. All allowed states can be visited.**
- **Backjumping is an elaboration to skip over variables that aren't related through constraints.**
  - Careful ordering of variables minimizes benefits of backjumping.
    - Chen, X.; van Beek, P., Journal Of Artificial Intelligence Research, Volume 14, pages 53-81, 2001







Coupling qubit path database to Mercury

Given
Solve for
Db = Graph database
Bm = Bitmap mask
Dist = Distance cutoff

- **Solvers for single wires**
  - connect2(Db,Bm,Dist,A,B,PathId)

Identify the alternative paths that connect qubit A to B.

- **Solvers for connected wires**
  - ioo_oo(Db,Bm,Dist,A,B,C,PathIdAB,PathIdBC)
  - oio_oo(Db,Bm,Dist,A,B,C,PathIdAB,PathIdBC)
  - ioi_oo(Db,Bm,Dist,A,B,C,PathIdAB,PathIdBC)
  - iio_io(Db,Bm,Dist,A,B,C,PathIdAB,PathIdBC)
  - iii_io(Db,Bm,Dist,A,B,C,PathIdAB,PathIdBC)

"Solvers" are just predicates, and they can fail. The term is meant to suggest that they are finding many answers and will fork the search in many directions, especially when they are minimally constrained like **ioo_oo** or **oio_oo.**

- **Predicates**
  - noOverlap(Db,PathId,QubitList)

Wires and start/end qubits are distinct.

- **Functions:**
  - calcLen(Db,PathId)
  - calcDist(A,B)
  - getPathStr(Db,PathId)

To constrain search, undesirable answers with long wires, or short wires with insufficient connection points can be forbidden.

Expand to the members of a path.

  - Bm = trimDb(Db,Exceptions,PathList)

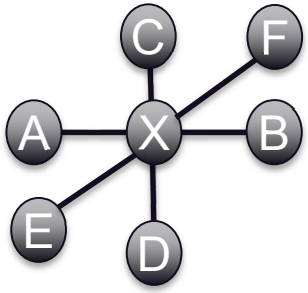
Remove intersections with these paths from consideration in the future.

Trimming paths -- wires must not cross

- **A used path precludes use of other paths. Only one path can own a qubit unless it at a connection point.**

- **AXB precludes use of CXD and EXF if X is not a logical qubit in the user problem.**

- **PathId is an integer associated with each path, e.g.**
  - AXB = 0
  - EXF = 1
  - CXD = 3
  - WYZ = 9 (last path – in practice there are millions)



- **Bitmask formed of union of these:**
  - 0 0 0 0 0 0 1 0 1 1

- **Before following any path, the relevant bit in the bitmask must be zero.**

Embedding a trigonal hexagon predicate “hexagonG6” (like a subroutine)

Given

Solve for

oio_oo(IndDb, Trim_bitmap0, Dist1, A, G, B, AGPathId, BGPathId, Trim_bitmap1),

connect2(IndDb, Trim_bitmap1, Dist1, A, B, ABPathId, Trim_bitmap2),

oio_oo(IndDb, Trim_bitmap2, Dist1, C, G, D, CGPathId, DGPathId, Trim_bitmap3),

connect2(IndDb, Trim_bitmap3, Dist1, C, D, CDPathId, Trim_bitmap4),

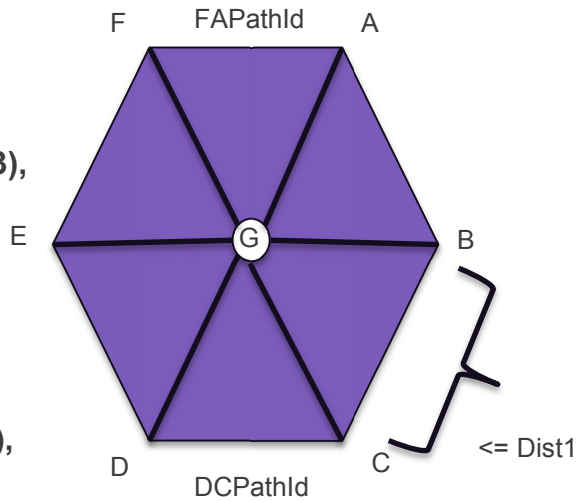
connect2(IndDb, Trim_bitmap4, Dist1, B, C, BCPathId, Trim_bitmap5),

oio_oo(IndDb, Trim_bitmap5, Dist1, E, G, F, EGPathId, FGPathId, Trim_bitmap6),

connect2(IndDb, Trim_bitmap6, Dist1, E, F, EFPathId, Trim_bitmap7),

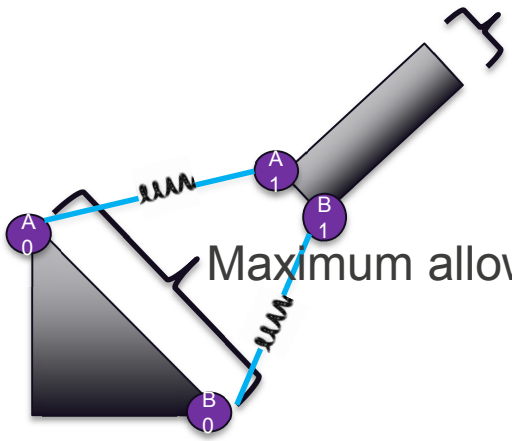
connect2(IndDb, Trim_bitmap7, Dist1, D, E, DEPathId, Trim_bitmap8),

connect2(IndDb, Trim_bitmap8, Dist1, F, A, FAPathId, Trim_bitmap9),



Springs: Compatibility of object sizes

Constraining component sizes speeds embedding and avoids sprawl.



Maximum allowed width of rectangle

Maximum allowed hypotenuse of triangle

A and B cannot be the same qubits. Need springy wires.

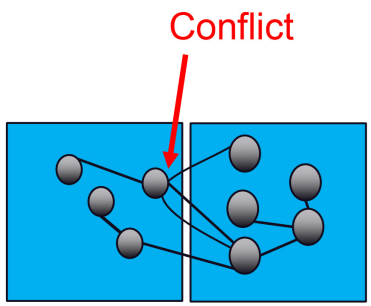
Given

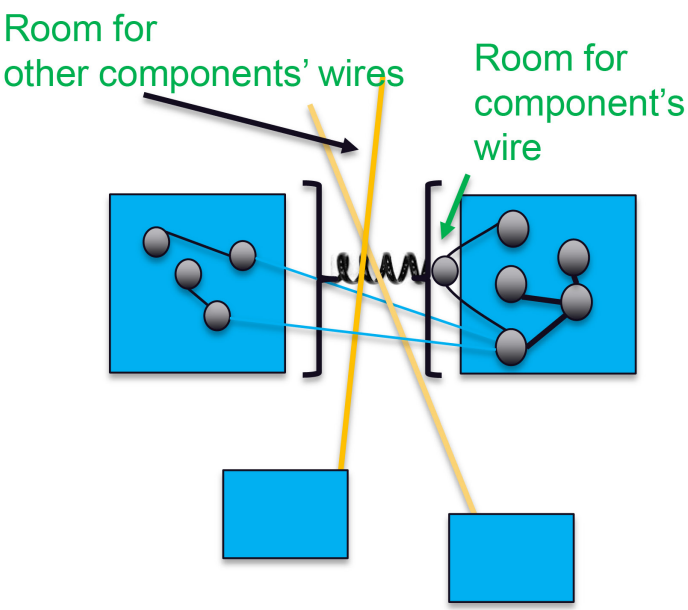
Solve for

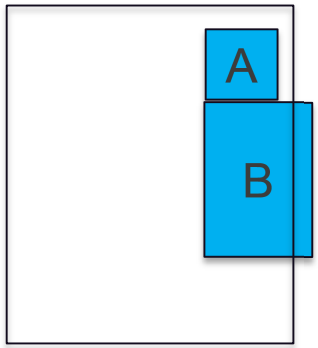
ioo_oo(IndDb, Trim_bitmapAB, 8, A0, A1, B1, A0A1PathId, A1B1PathId, Trim_bitmapAB1),
calcDist(A0,A1) > 0,
calcDist(B0,B1) > 0,
iii_io(IndDb, Trim_bitmapAB1, 8, A1, B1, B0, A1B1PathId, B1B0PathId, Trim_bitmapAB2),

Distance filtering

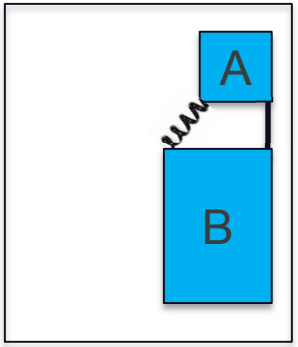
Springs: Getting out of tight spots



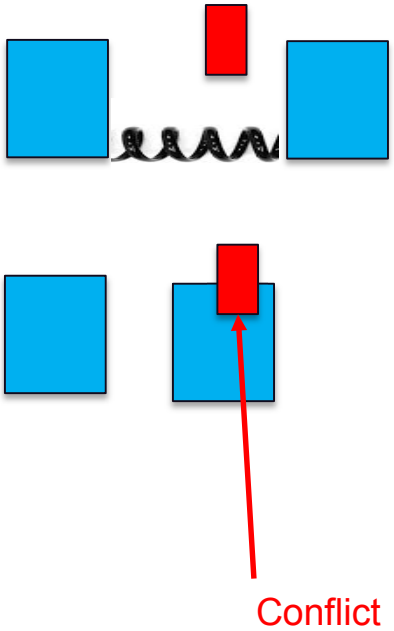




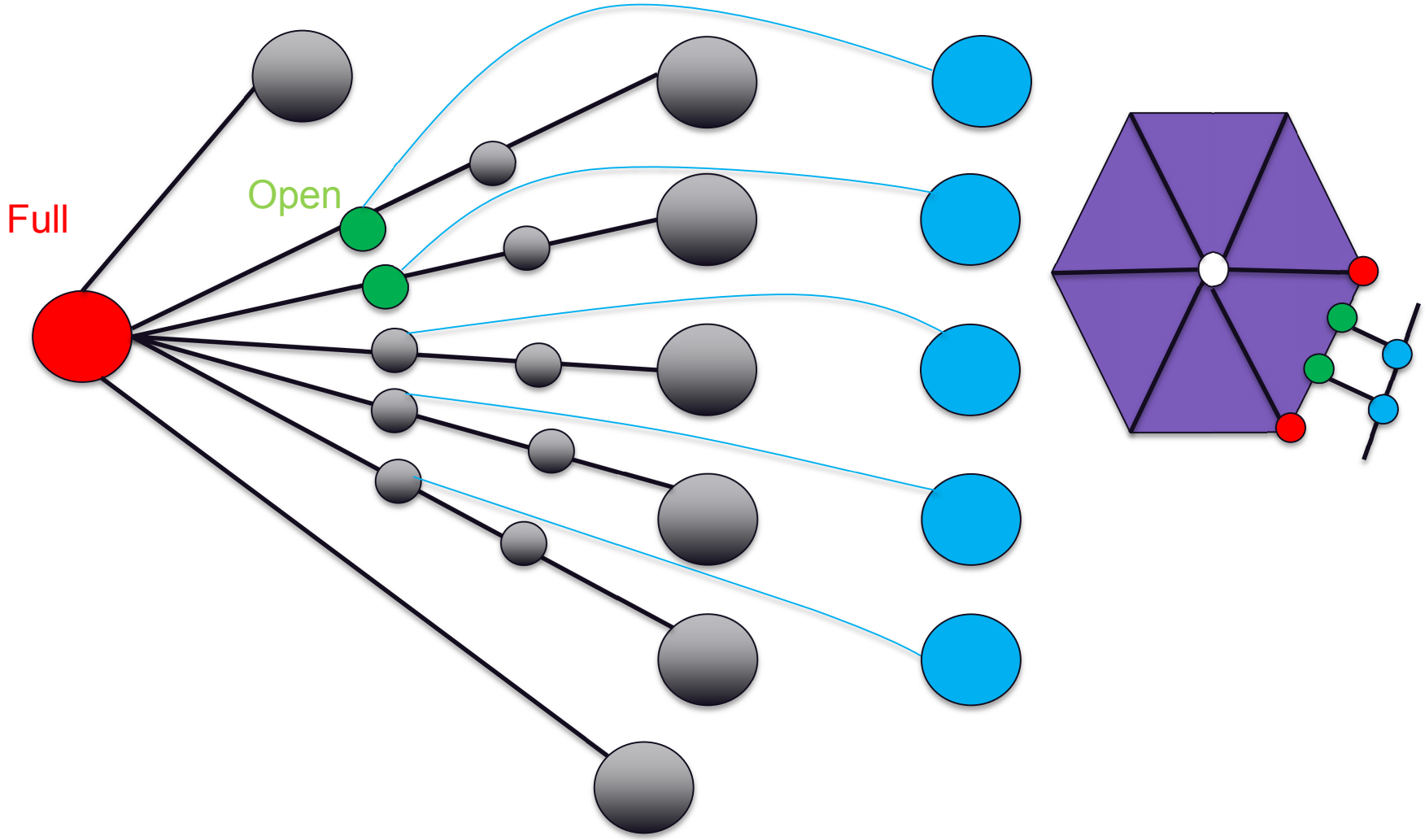
Let a peer move from edge of graph



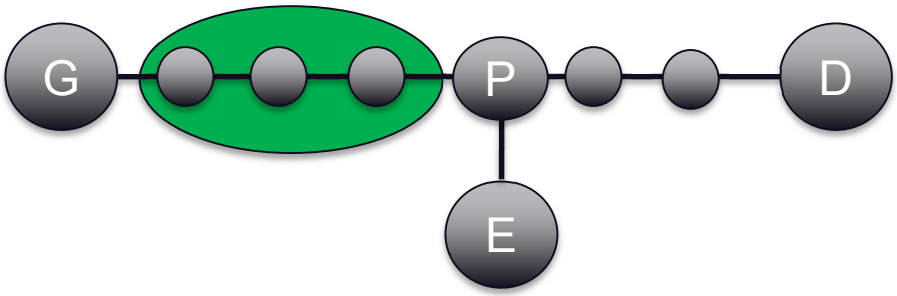
Avoid a disabled qubit that would break up a component



Wires as alternative connection points



Re-use of wire segments



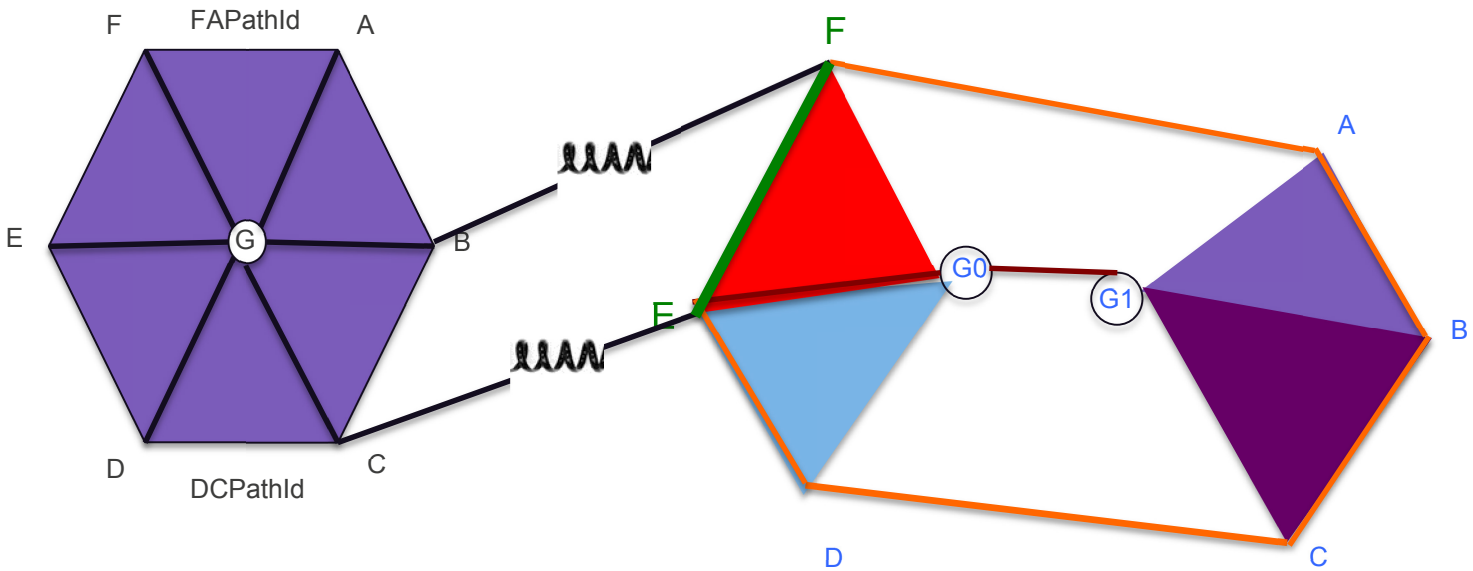
When predicates fail (e.g. DE), introduce P (symbolically) and use GPE and DPE instead of two pair searches GE and DE.

Attaching a split trigonal hexagon to another

Given

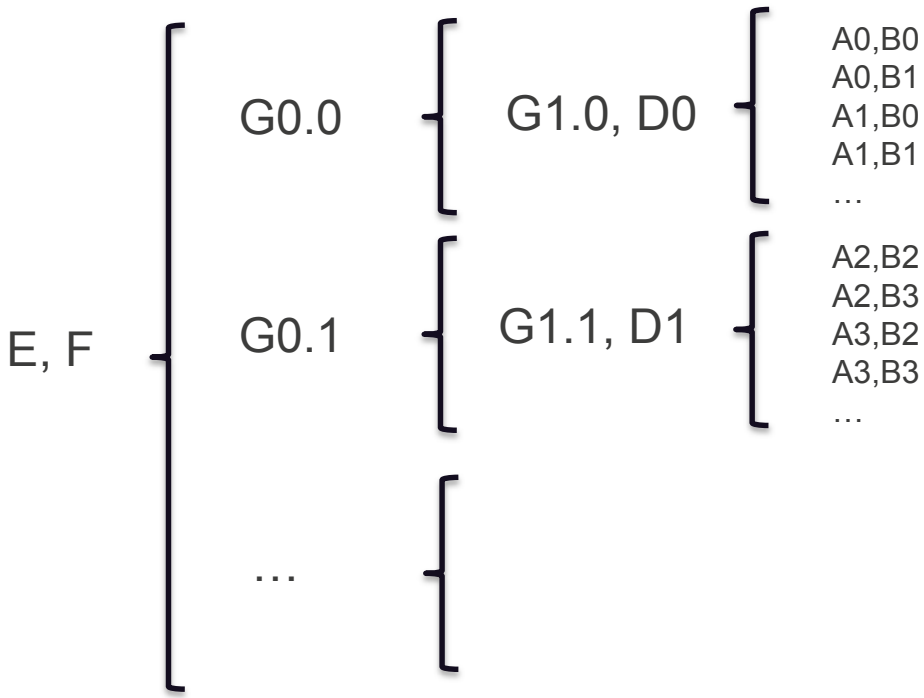
Solve for

```
ioi_oo(IndDb, Trim_bitmap0, Dist, E, G0, F, EGPathId, FGPathId, Trim_bitmap1),
iio_io(IndDb, Trim_bitmap1, Dist, E, G0, G1, EGPathId, G0G1PathId, Trim_bitmap2),
iio_io(IndDb, Trim_bitmap2, Dist, E, G0, D, EGPathId, DGPathId, Trim_bitmap3),
connect2(IndDb, Trim_bitmap3, Dist, D, E, DEPathId, Trim_bitmap4),
oio_oo(IndDb, Trim_bitmap4, Dist, B, G1, A, BGPathId, AGPathId, Trim_bitmap5),
connect2(IndDb, Trim_bitmap5, Dist, A, B, ABPathId, Trim_bitmap6),
iio_io(IndDb, Trim_bitmap6, Dist, B, G1, C, BGPathId, CGPathId, Trim_bitmap7),
connect2(IndDb, Trim_bitmap7, Dist, B, C, BCPathId, Trim_bitmap8),
connect2(IndDb, Trim_bitmap8, Dist, C, D, CDPathId, Trim_bitmap9),
connect2(IndDb, Trim_bitmap9, Dist, F, A, FAPathId, Trim_bitmap10),
```



How the search expands

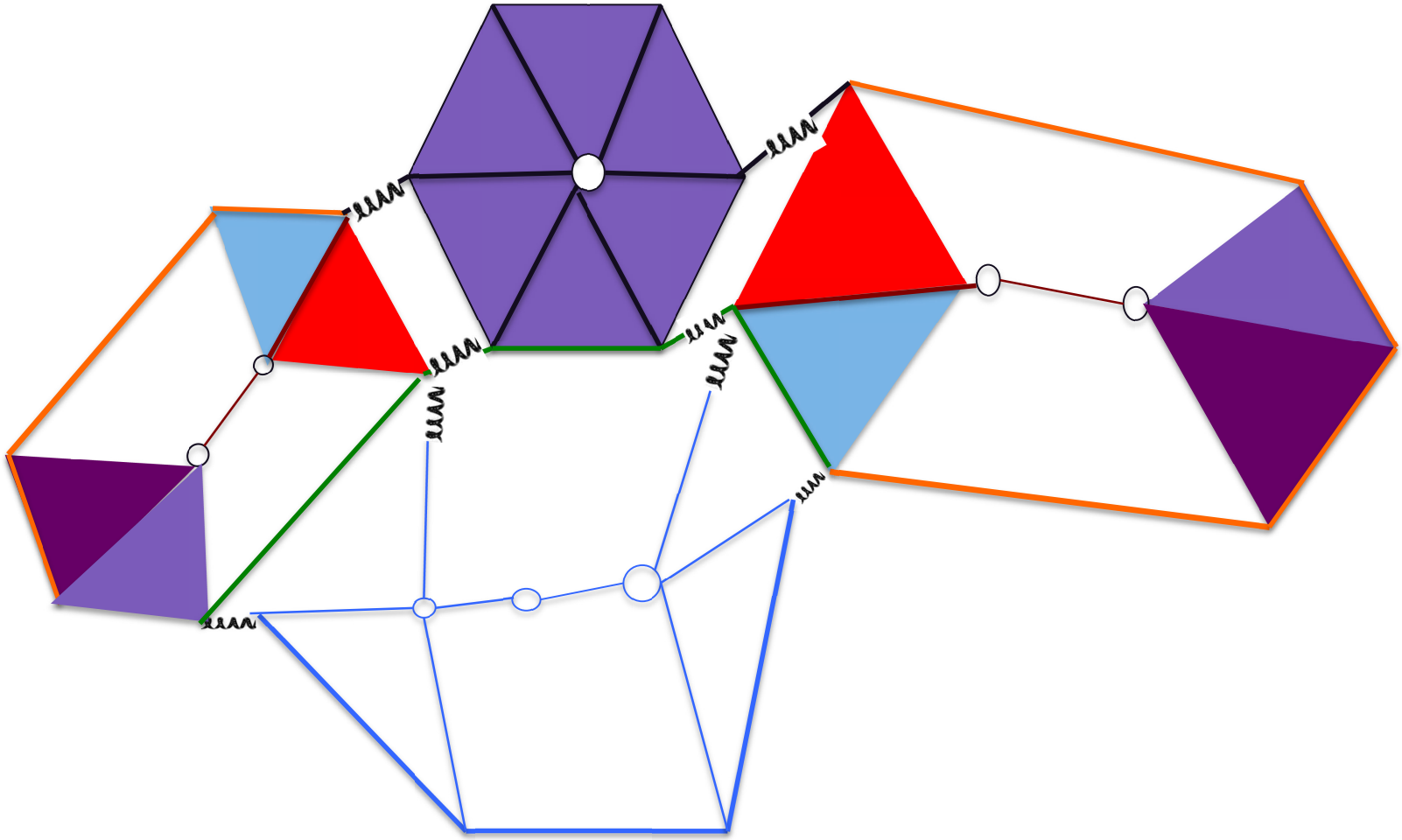
ioi_oo(IndDb, Trim_bitmap0, Dist, E, G0, F, EGPathId, FGPathId, Trim_bitmap1),
iio_io(IndDb, Trim_bitmap1, Dist, E, G0, G1, EGPathId, G0G1PathId, Trim_bitmap2),
iio_io(IndDb, Trim_bitmap2, Dist, E, G0, D, EGPathId, DGPathId, Trim_bitmap3),
oio_oo(IndDb, Trim_bitmap4, Dist, B, G1, A, BGPathId, AGPathId, Trim_bitmap5),



Given three sides, fit another

Given

Solve for



Given three sides, fit another

Given

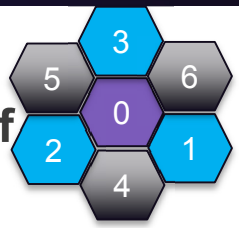
Solve for

```
ioi_oo(IndDb, Trim_bitmap0, Dist, B, GABC, A, BGPathId, AGPathId, Trim_bitmap1),
ioi_oo(IndDb, Trim_bitmap1, Dist, E, GDEF, F, EGPathId, FGPathId, Trim_bitmap2),
ioi_oo(IndDb, Trim_bitmap2, Dist, GABC, G, GDEF, GABC_G_PathId, G_GDEF_PathId, Trim_bitmap3),

iio_io(IndDb, Trim_bitmap3, Dist, B, GABC, C, BGPathId, CGPathId, Trim_bitmap4),
iio_io(IndDb, Trim_bitmap4, Dist, E, GDEF, D, EGPathId, DGPathId, Trim_bitmap5),

connect2(IndDb, Trim_bitmap5, Dist, B, C, BCPathId, Trim_bitmap6),

connect2(IndDb, Trim_bitmap6, Dist, C, D, CDPathId, Trim_bitmap7),
connect2(IndDb, Trim_bitmap7, DistDE, D, E, DEPathId, Trim_bitmap8),
```

Extension qubit G exists because shortest paths sometimes can't be used once the problem is developed this far.

The graph has grown too much and has too many obstacles. This connects two shortest paths.

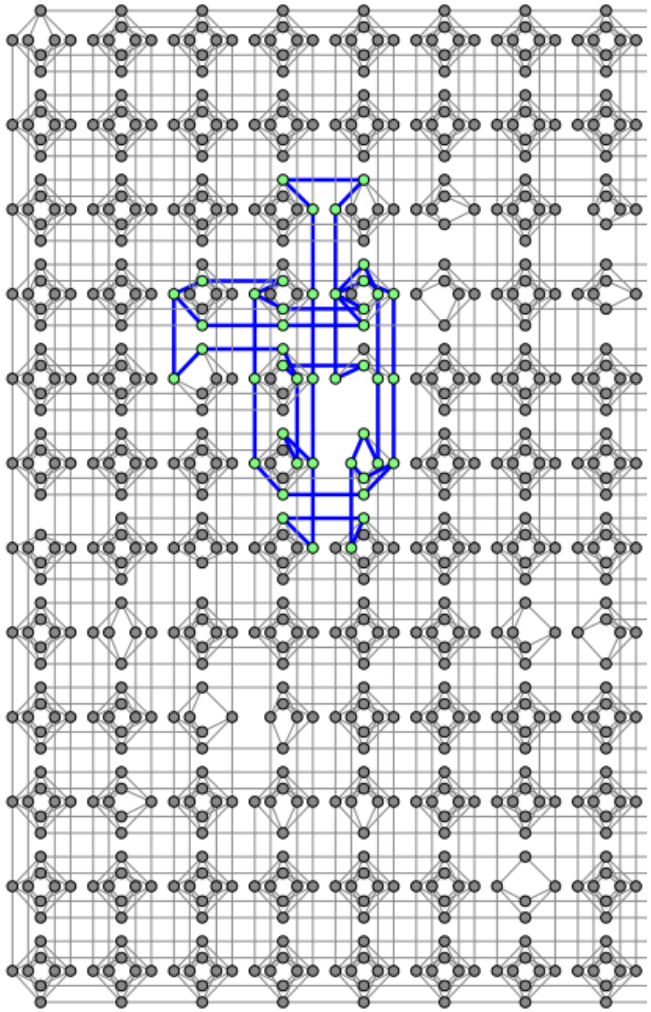
Putting it together



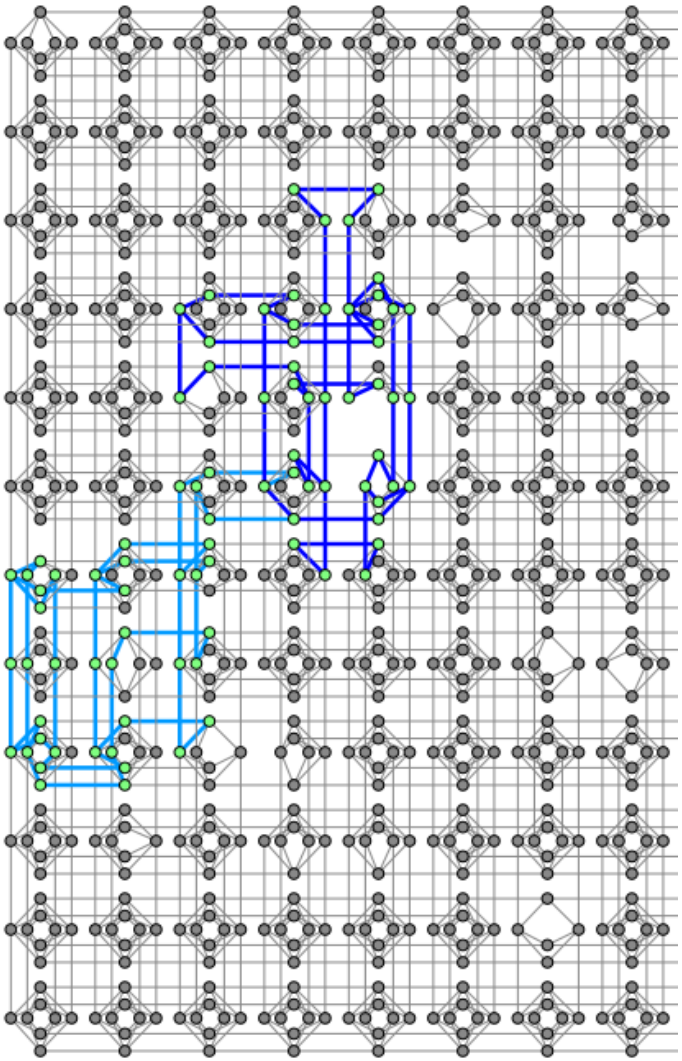
1. Embed center hexagon.
2. Get connection qubits for hexagon. Take one step inward the path of the 0/1 edge so that connections don't run out.
3. Trim out all used paths and qubits except for these 2 qubits.
4. Introduce the springs to hexagon 1, re-trim.
5. Embed hexagon 1
6. Repeat from #2 but for hexagons 2 and 3
7. Get connection qubits for hexagon 4. There are more edge constraints, but there are springs for each fixed qubit.
8. Trim out all used paths and qubits except for the 4 connection qubits.
9. Introduce the springs to hexagon 4, re-trim.
10. Repeat from #7 for hexagons 5 and 6.

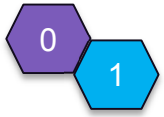
Backtrack when a dead end is hit.

Hexagon 0 (no constraints)

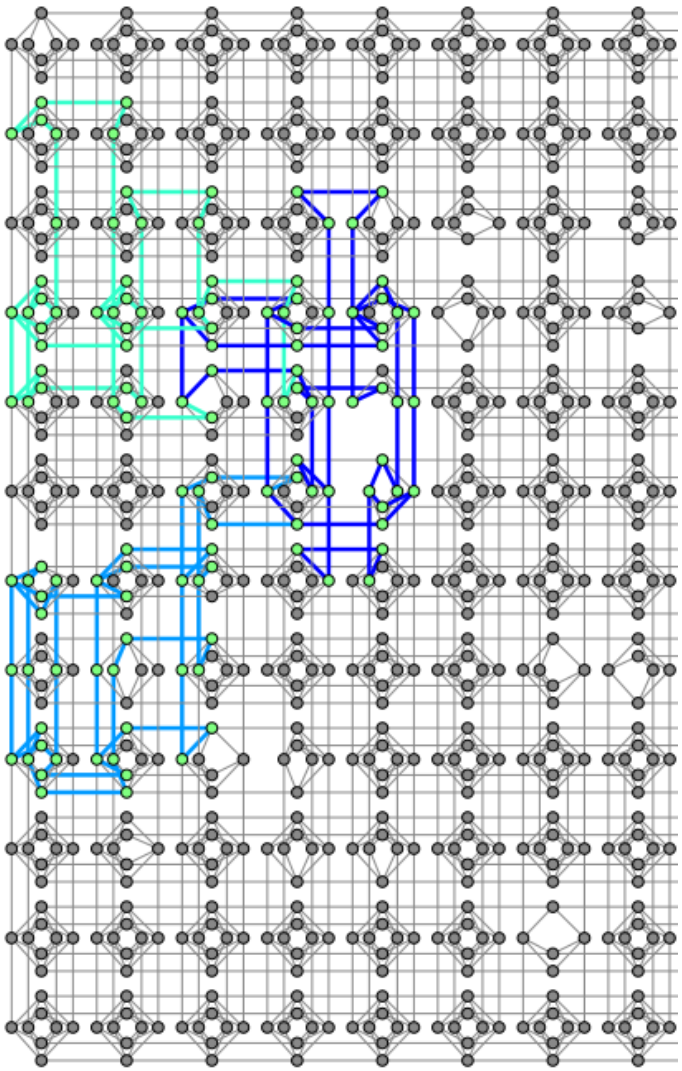


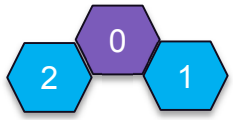
Hexagon 1 (one constraint)



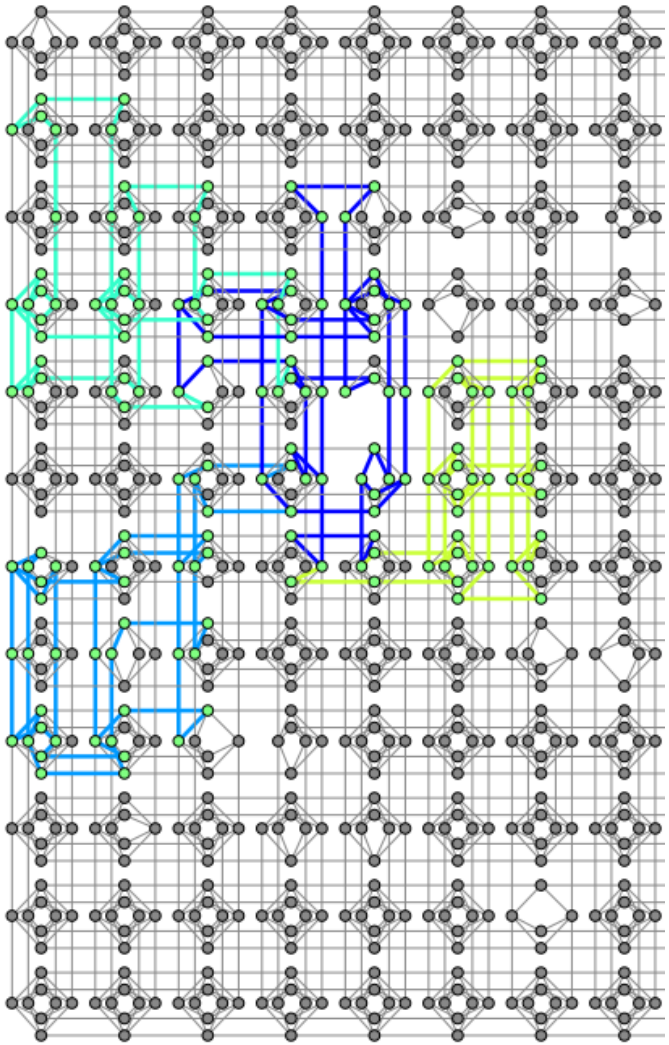


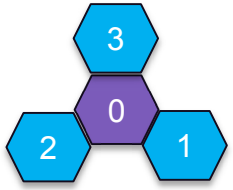
Hexagon 2 (1 constraint)



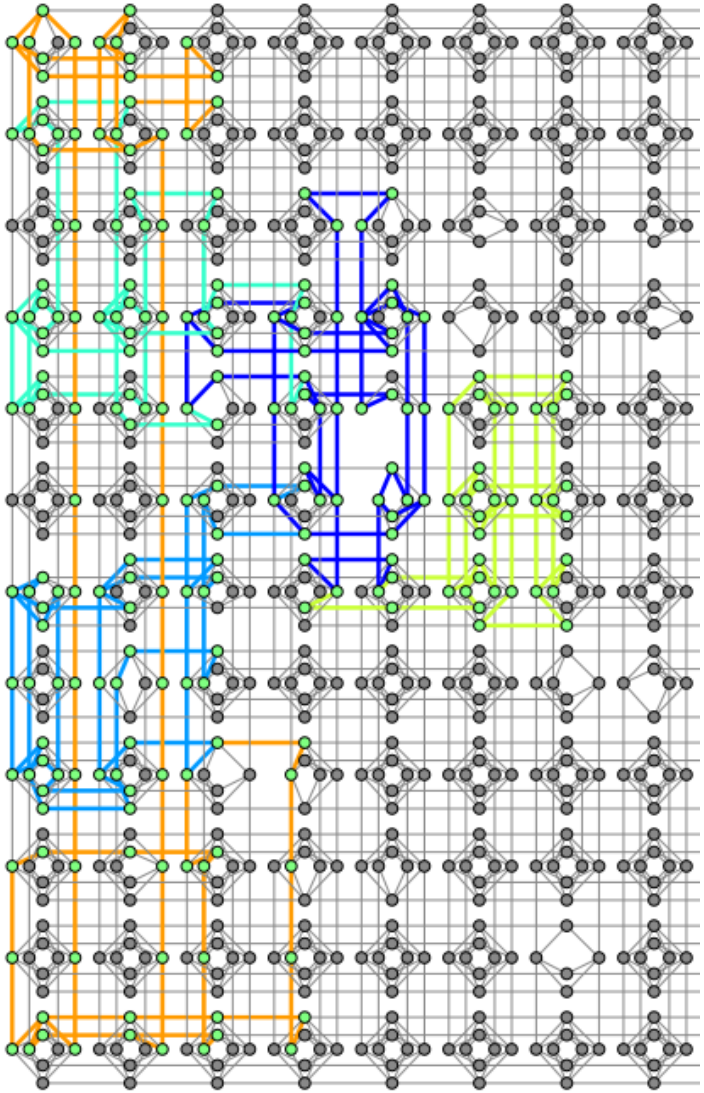


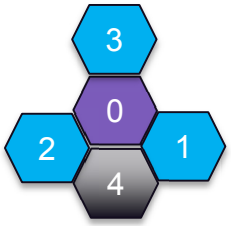
Hexagon 3 (one constraint)



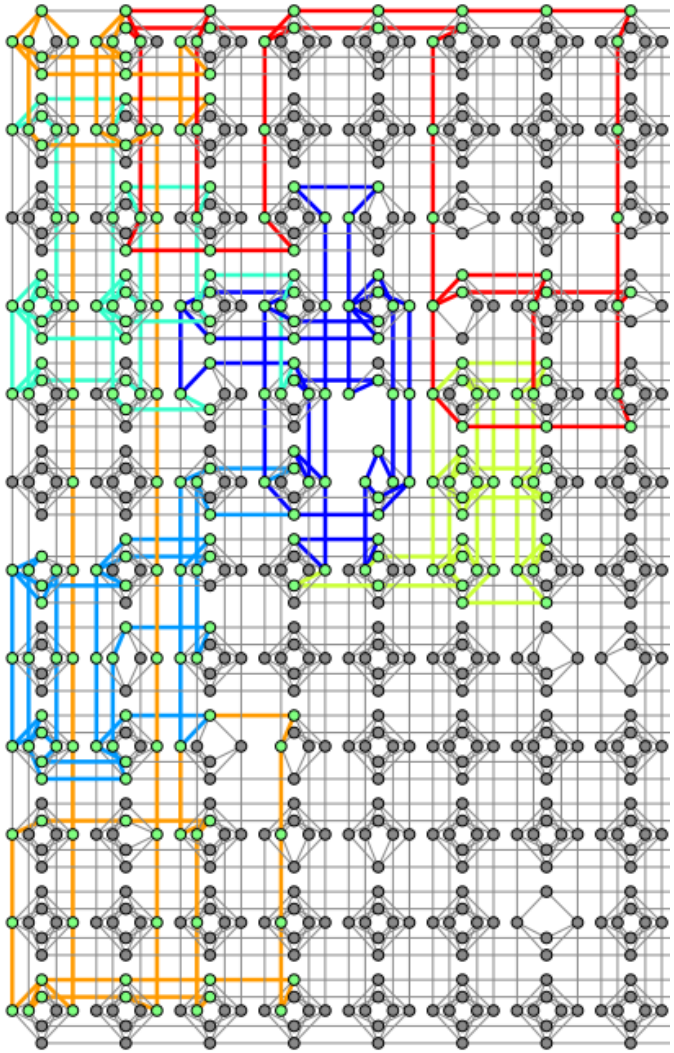


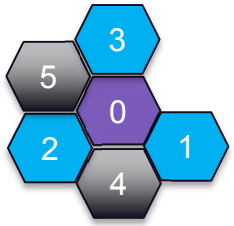
Hexagon 4 (three constraints)



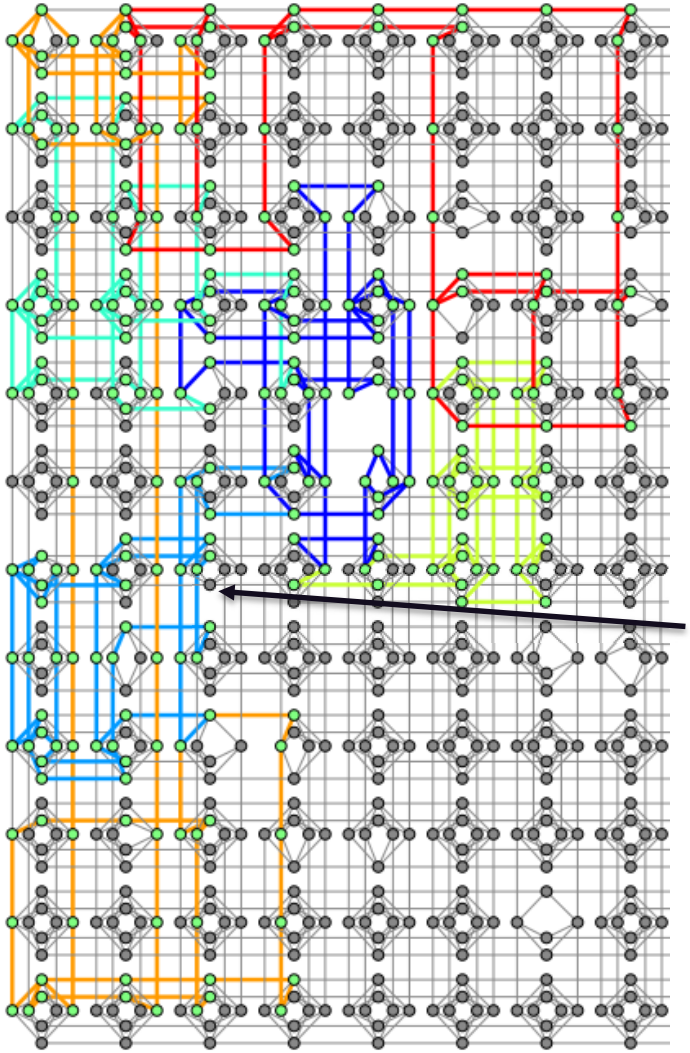


Hexagon 5 (three constraints)





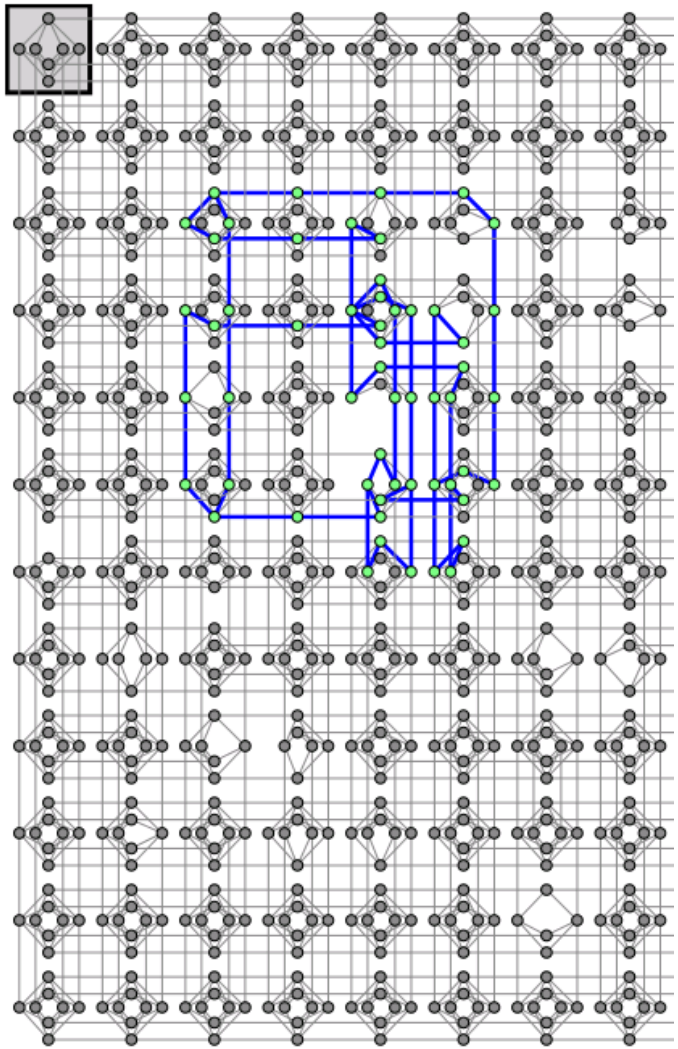
What about the last hexagon?



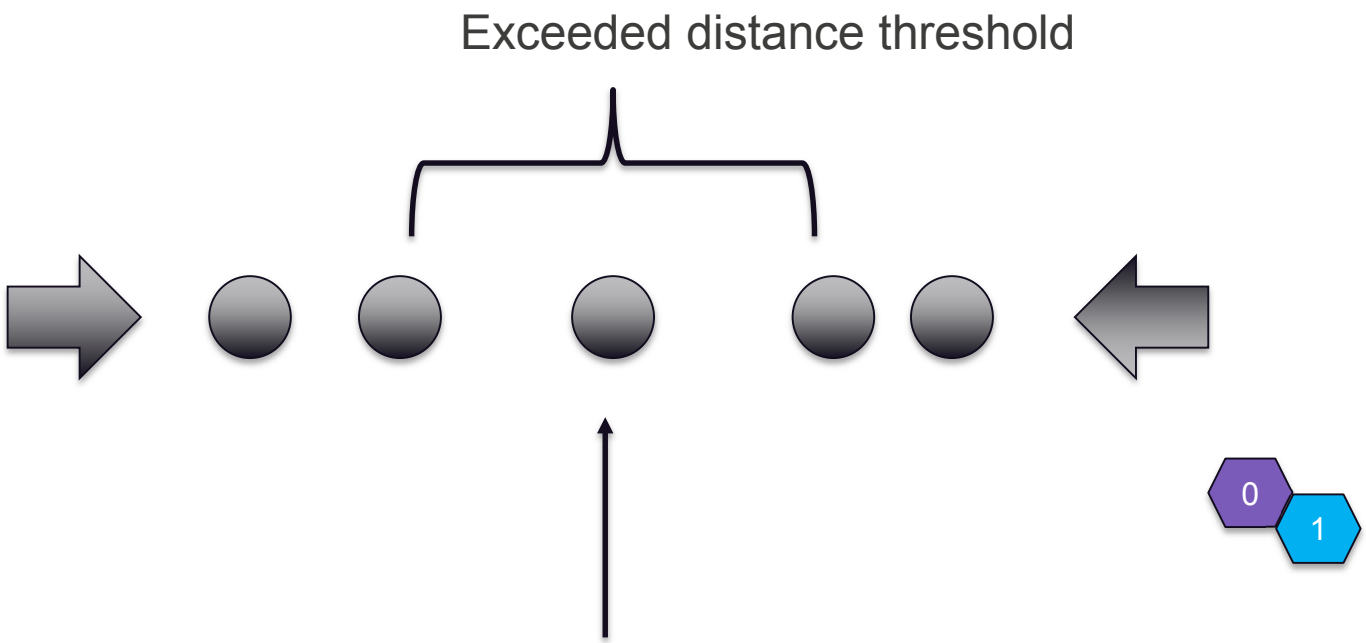
Backtrack, refit hexagon 0 and restart with hexagon 1.

Force 592 & 593 to different unit cells?

Out of connections here

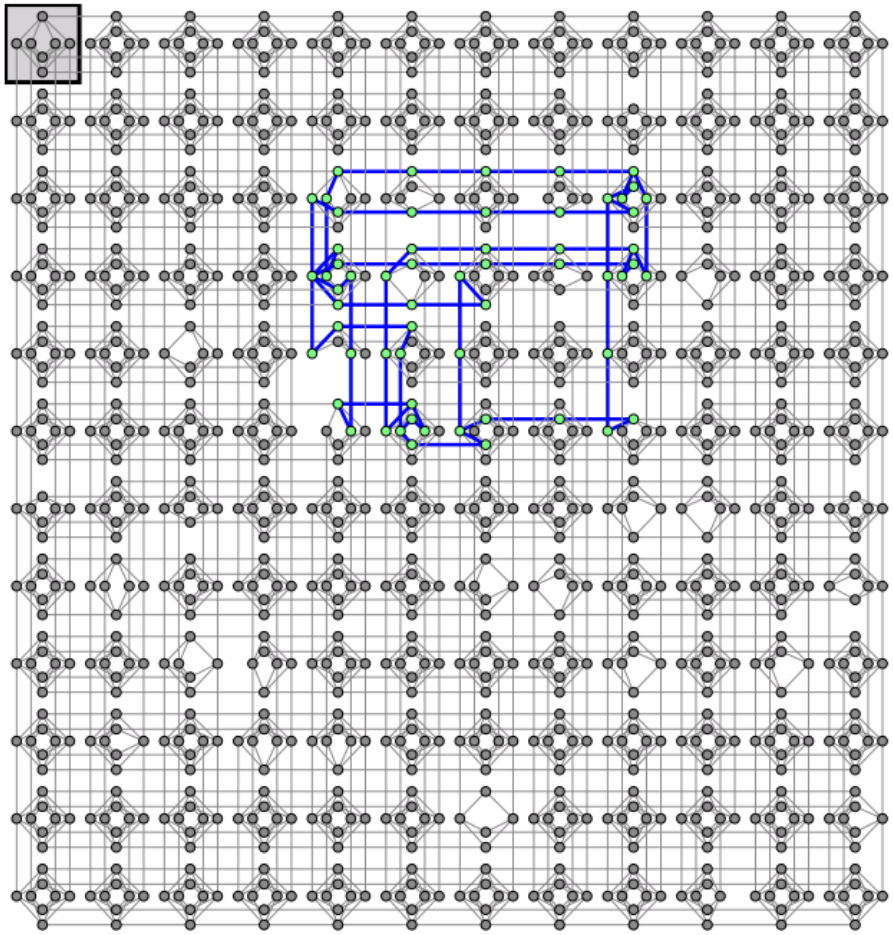


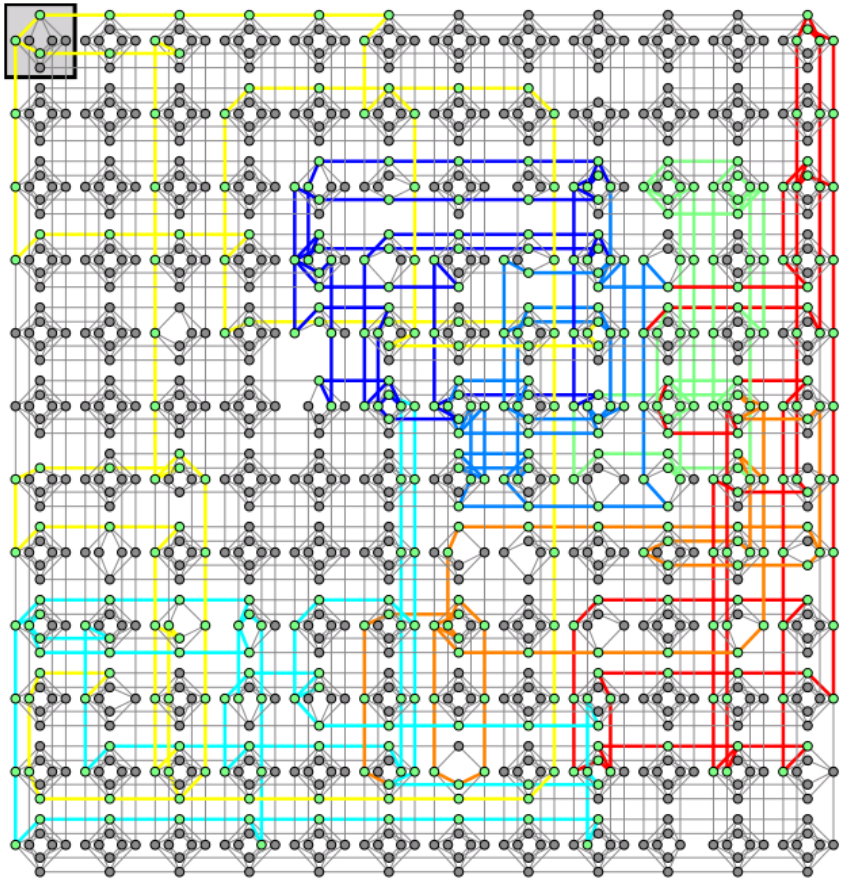
Fix



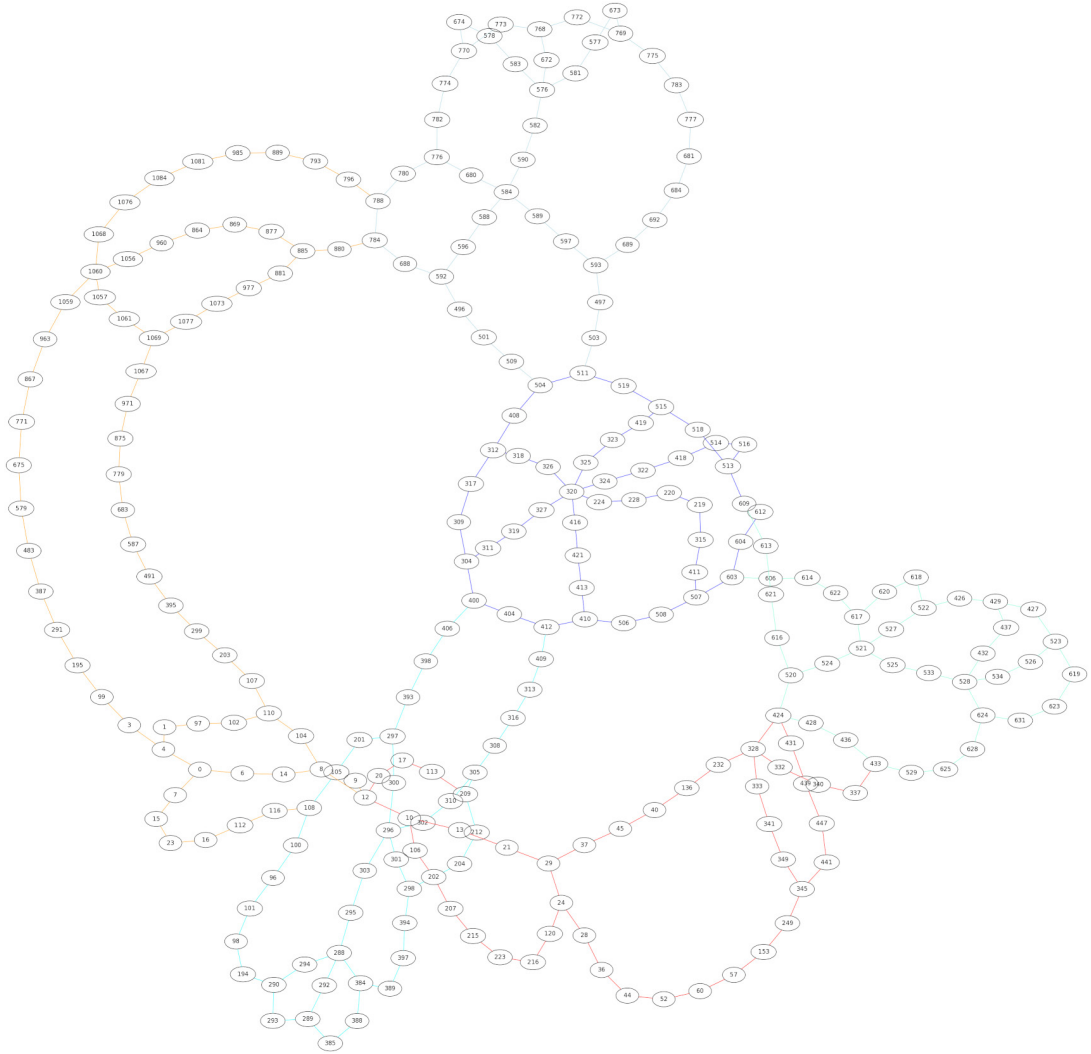
Ended-up in the center, and oversubscribed the qubit coupler connections

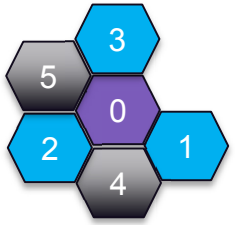
Try #2





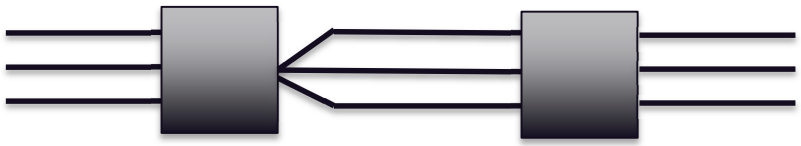
Network visualization





Next steps

Can long connections be reinforced?
Periodically interleave error correcting circuits?



Investigate smarter solver techniques.
- Parallel breadth-first search
- Lazy clause generation

Try new D-Wave algorithm on trigonal hexagon embedding problems:
- Fast clique minor generation in Chimera qubit connectivity graphs
- Tomas Boothby, Andrew D. King, Aidan Roy, July 20, 2015
- https://github.com/dwavesystems/chimera-embedding