Graph Partitioning using the D-Wave for Electronic Structure Problems

ISTI D-Wave Quantum Computing Efforts Debrief

S. M. Mniszewski, smm@lanl.gov
C. F. A. Negre, cnegre@lanl.gov
H. Ushijima-Mwesigwa, hayato@lanl.gov

October 6, 2016

Los Alamos NATIONAL LABORATORY EST. 1943

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

LA-UR-16-27873

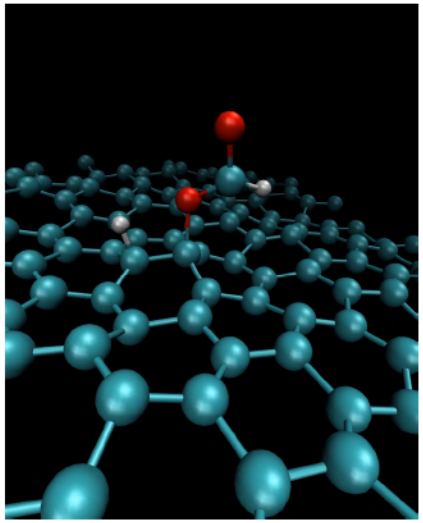
Description

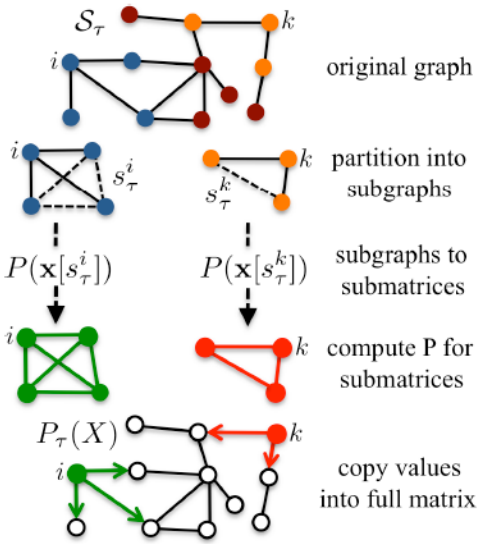
- **Motivated by graph-based methods for quantum molecular dynamics (QMD) simulations**
- **Based on ideas from recursive bisection in METIS and the "map coloring" problem presented in the D-Wave tutorial**
- **Explored graph partitioning/clustering methods and implementations that run on the D-Wave**
- **Approaches:**
  - *Partitioning* of a graph into equal parts minimizing the number of connections between parts.
  - *Clustering* of a graph using modularity or community detection.
  - Extend to recursive, multilevel, and all-at-once methods.
- **Show "proof of principle" results on example graphs and material systems**

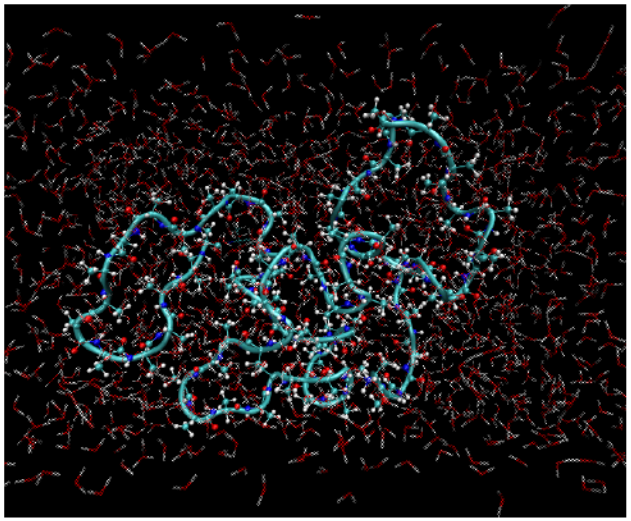
Graph partitioning/clustering implementations on the D-Wave.

Motivation

- **Next Generation Quantum Molecular Dynamics LDRD-DR (PI:AMN Niklasson)**
- **Quantum-based models capture the making and breaking of covalent bonds, charge transfer between species of differing electronegativities, and long-range electrostatic interactions - reactions**
- **Graph-based methods for quantum molecular dynamics (QMD) simulations**
  A. M. N. Niklasson et al, Graph-based linear scaling electronic structure theory, *J. Chem. Phys.* **144**, 234101 (2016).
- **Density matrix generated each timestep from many small sub-matrices (or sub-graphs)**
- **Shown to be equivalent to traditional methods (ex. diagonalization)**







Two-Qubit Objectives from D-Wave Tutorial

q_1	q_2	$O(a_1, a_2, b_{12}; q_1, q_2)$	equal $q_1 = q_2$	not equal $q_1 = \sim q_2$	implies $q_1 \Rightarrow q_2$
0	0	0	0	0	0
0	1	a_2	1	-1	0
1	0	a_1	1	-1	1
1	1	$a_1 + a_2 + b_{12}$	0	0	0

$a_1 = 1$
$a_2 = 1$
$b_{12} = -2$

$a_1 = -1$
$a_2 = -1$
$b_{12} = 2$

$a_1 = 1$
$a_2 = 0$
$b_{12} = -1$

$$Obj(a_i, b_{ij}, q_i) = \sum a_i q_i + \sum b_{ij} q_i q_j$$

Graph Coloring to Equal Coloring – DW example

Graph coloring: two regions, three colors - two-regions.q

param: color = 1, neighbor = 1

Possible colorings for AB and BC
var: BC_red, BC_green, BC_blue
var: AB_red, AB_green, AB_blue

AB and BC get one of three colors:
term: -1 * color * BC_red
term: -1 * color * BC_green
term: -1 * color * BC_blue
term: 2 * color * BC_red * BC_green
term: 2 * color * BC_red * BC_blue
term: 2 * color * BC_green * BC_blue
…

 # PENALIZE colorings in which AB and BC have the same:
term: 1 * neighbor * AB_red * BC_red
term: 1 * neighbor * AB_green * BC_green
term: 1 * neighbor * AB_blue * BC_blue

Check that each region only gets a single color:
assert: color: BC_red + BC_green + BC_blue - 1
assert: color: AB_red + AB_green + AB_blue - 1

 # Check that neighboring regions do NOT get the same color:
assert: neighbor: 1 * AB_red * BC_red
assert: neighbor: 1 * AB_green * BC_green
assert: neighbor: 1 * AB_blue * BC_blue

Equal coloring: two regions, three colors – equal-regions.q

param: color = 1, neighbor = 1

Possible colorings for AB and BC
var: BC_red, BC_green, BC_blue
var: AB_red, AB_green, AB_blue

AB and BC get one of three colors:
term: -1 * color * BC_red
term: -1 * color * BC_green
term: -1 * color * BC_blue
term: 2 * color * BC_red * BC_green➔
term: 2 * color * BC_red * BC_blue
term: 2 * color * BC_green * BC_blue
,,,

 # ENCOURAGE colorings in which AB and BC have the same:
term: -1 * neighbor * AB_red * BC_red
term: -1 * neighbor * AB_green * BC_green
term: -1 * neighbor * AB_blue * BC_blue

Check that each region only gets a single color:
assert: color: BC_red + BC_green + BC_blue - 1
assert: color: AB_red + AB_green + AB_blue – 1

 # Check that neighboring regions get the SAME color:
assert: neighbor: -1 * AB_red * BC_red
assert: neighbor: -1 * AB_green * BC_green
assert: neighbor: -1 * AB_blue * BC_blue

Three Partitioning – DW Example

Graph coloring: two regions, three colors – *two-regions.q*

```
param: color = 1, neighbor = 1

# Possible colorings for AB and BC
var: BC_red, BC_green, BC_blue
var: AB_red,  AB_green, AB_blue

# AB and BC get one of three colors:
term: -1 * color * BC_red
term: -1 * color * BC_green
term: -1 * color * BC_blue
term: 2 * color * BC_red * BC_green
term: 2 * color * BC_red * BC_blue
term: 2 * color * BC_green * BC_blue
…
# PENALIZE colorings in which AB and BC have the same:
term: 1 * neighbor * AB_red * BC_red
term: 1 * neighbor * AB_green * BC_green
term: 1 * neighbor * AB_blue * BC_blue

# Check that each region only gets a single color:
assert: color: BC_red + BC_green + BC_blue - 1
assert: color: AB_red + AB_green + AB_blue - 1

# Check that neighboring regions do NOT get the same color:
assert: neighbor: 1 * AB_red * BC_red
assert: neighbor: 1 * AB_green * BC_green
assert: neighbor: 1 * AB_blue * BC_blue
```

Three parts: three regions, three colors – *part3-regions.q*

```
param: color = 1, neighbor = 0.5

# Possible colorings for AB and BC
var: BC_red, BC_green, BC_blue
var: AB_red,  AB_green, AB_blue
Var: AC_red, AC_green, AC_blue

# AB, BC,and  AC get one of three colors:
term: -1 * color * BC_red
term: -1 * color * BC_green
term: -1 * color * BC_blue
term: 2 * color * BC_red * BC_green
term: 2 * color * BC_red * BC_blue
term: 2 * color * BC_green * BC_blue
...
# ENCOURAGE colorings in which AB-BC, BC-AC, AB-AC are same:
term: -1 * neighbor * AB_red * BC_red
term: -1 * neighbor * AB_green * BC_green
term: -1 * neighbor * AB_blue * BC_blue
…
# PENALIZE colorings in which AB-BC, BC-AC, AB-AC have the same
term: 1 * neighbor * AB_red * BC_red
term: 1 * neighbor * AB_green * BC_green
term: 1 * neighbor * AB_blue * BC_blue
…
# Check that each region only gets a single color:
assert: color: BC_red + BC_green + BC_blue - 1
assert: color: AB_red + AB_green + AB_blue - 1
assert: color: AC_red + AC_green + AC_blue - 1
```

Graph Clustering - Description

- **Using modularity or community network analysis for natural clusters**
  - M. E. J. Newman, Modularity and community structure in networks, 2006, PNAS, vol. 103, no. 23, 8577-8582.
- **For example, identifying secondary structures in proteins**
- **Maximize modularity**
- **Fits naturally on D-Wave, no reformulation required**

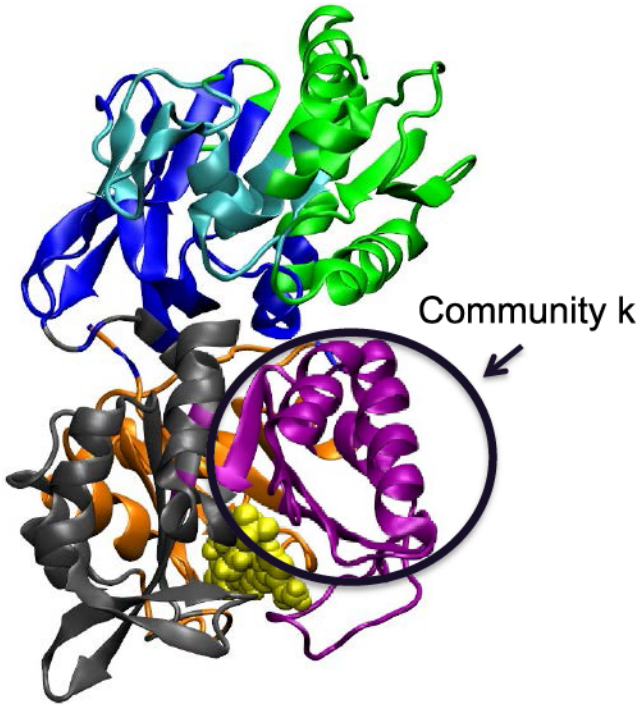
Correlation matrix (for CNA)

$$\mathbf{r}_{ij}^{MI} = g(\mathbf{I}[\mathbf{x}_i, \mathbf{x}_j])$$

Girvan-Newman

$\longrightarrow$

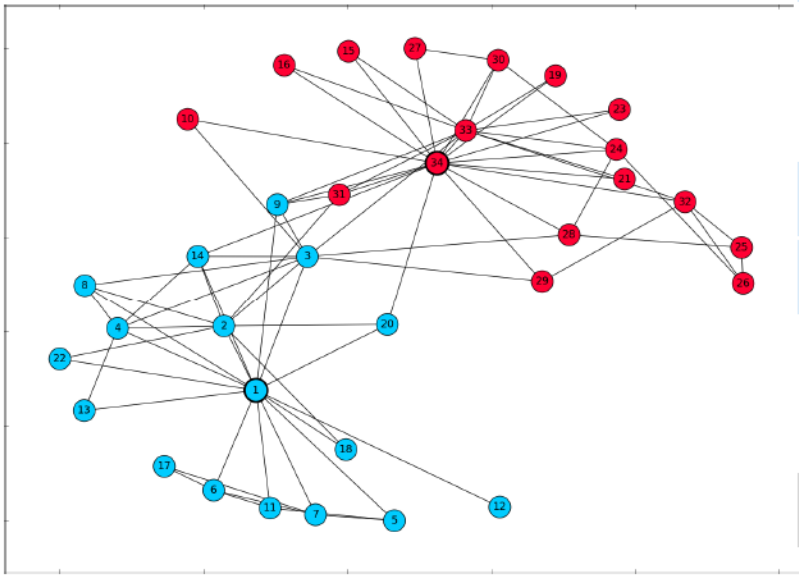
Modularity Matrix



IGPS Protein Structure

Graph Clustering – example with thresholding

- Using modularity or community detection for natural clusters
- Compare for 2 clusters
- Effect of thresholds on weights
- Reduction in couplers used



Karate club graph (n = 34) using qbsolv

Threshold	# couplers	modularity
0	561	0.371794871795
0.02	544	0.371794871795
0.05	411	0.371466140697
0.07	300	0.371466140697
0.08	244	0.271449704142
0.1	227	0.271449704142
0.11	212	0.255095332018
0.12	194	0.255095332018
0.13	169	0

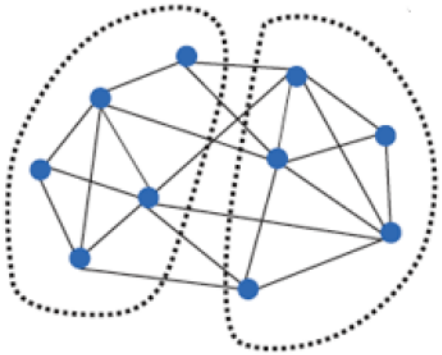
Graph Clustering – recursive, 4 clusters

- NetworkX powerlaw_cluster_graph() graph generator, n = 100
- Random graph with powerlaw degree distribution and approximate average clustering
- modularity = 0.225261961874 using NetworkX community module
- Thresholding reduces D-Wave couplers needed

Threshold	# couplers	modularity
0	4950	0.226555990405
0.06	4593	0.226514960232
0.08	3799	0.187831713168
0.1	3091	0.066793334175

Graph Partitioning - Description

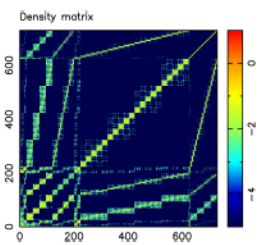
- **Definition:**
  - Given a graph G = (V, E)
    - V ~ nodes
    - E ~ edges (possibly weighted)
- **Goal:** Partition V into k equal parts minimizing the number of cut edges between parts



- **Applications:**
  - Graph-based QMD simulations - Used as initial solution for core-halo partitioning
  - Physical network design
  - VLSI design
  - Telephone network design (original application with algorithm due to Kernighan)
  - Load balancing - minimize total communication between processors
  - Sparse matrix-vector multiplication - Partition the rows of a matrix to minimize communication during matrix-vector multiplication

Graph Partitioning: Formulation

- **Exact Formulation**
  - min $x^T L x$, such that $\Sigma\, x_i = n/2$, x_i in {0,1}
- **QUBO**
  - min $x^T (\beta L + \alpha 1_{nxn})\, x - \alpha n\, (\Sigma x_i) + \alpha n^2/4$, x_i *in* $\{0,1\}$
- **where**
- L ~ Laplacian matrix; 1_{nxn} ~ all ones matrix
- α, β ~ penalization constants

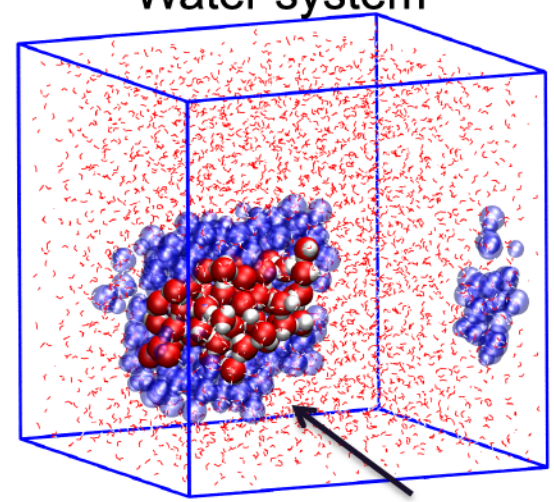


Density matrix (for QMD)

$$\rho_{ij} = \langle i | f(H) | j \rangle$$

METIS →

10,000 atoms
Water system



Partition k

Graph Partitioning – Benchmarks and Random Graphs

- **Graphs partitioned: 2 parts**
  - Using **sapi**, up to ~70 vertices
  - Using **qbsolv**, up to ~8000 vertices
- **Results comparison: minimize connections**
  - Quality
    - METIS
    - KaHIP (winner 10th DIMACS challenge)
    - Best known solution
  - Data –
    - Walshaw benchmark archive (http://chriswalshaw.co.uk/partition/)
    - Random graph models, e. g. Erdos-Renyi, PowerLaw graphs
- **Comparable to existing methods and sometimes better!**

Graph Partitioning Results using sapi

Dense graph with penalty constant $\alpha = \beta = 1$, Graph(n, 0.9).

n	METIS	KaHIP	sapi
			simulator
20	82	82	82
30	183	182	182
40	326	324	330
			D-Wave 2X
40	334	334	334
60	766	765	768
70	1039	1042	1045

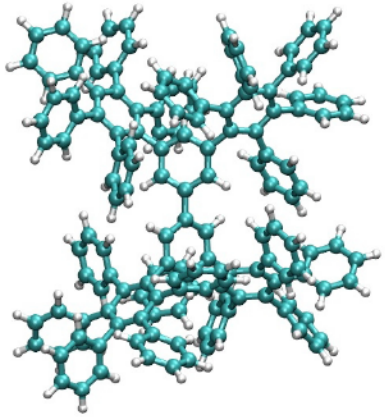
Graph Partitioning Results using qbsolv

Walshaw GP benchmark graphs:

Graph	n	best	METIS	KaHIP	qbsolv
add20	2395	596	723	760	647
data	2851	189	225	221	191
3elt	4720	90	91*	92	90
bcsstk33	8738	10171	10244*	10175	10171

*: not 0% imbalance

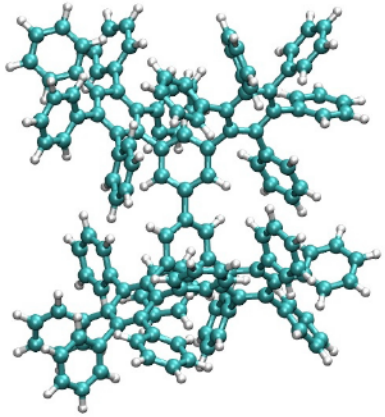
Graph Partitioning Results using qbsolv



QMD protein system density matrix graphs:

Graph	n	METIS	KaHIP	qbsolv
Phenyl dendrimer	730	705	705	705
Peptide 1aft	384	11	6	3

Graph Partitioning – Recursive Bisection w/o refinement



Partitioning into k parts:

Graph	k	METIS	KaHIP	qbsolv
Phenyl dendrimer n = 730	2	705	705	705
	4	2636	2638	2654
	8	12621	12677	15769
Peptide 1aft n = 384	2	11	6	3
	4	37	35	22
	8	72	78	66
	16	193	198	604

Graph Partitioning – Multiple parts, all at once

- Partition into k parts
- Unary embedding
- k qubits per vertex
- Turn on one of k part qubits for each vertex
- Similar to graph coloring problem
- Using Hayato's formulation

Example: partition into 4 parts
graph has 250 nodes and 28049 edges

Dwave part sizes [63, 58, 63, 66]
Dwave cost 20618

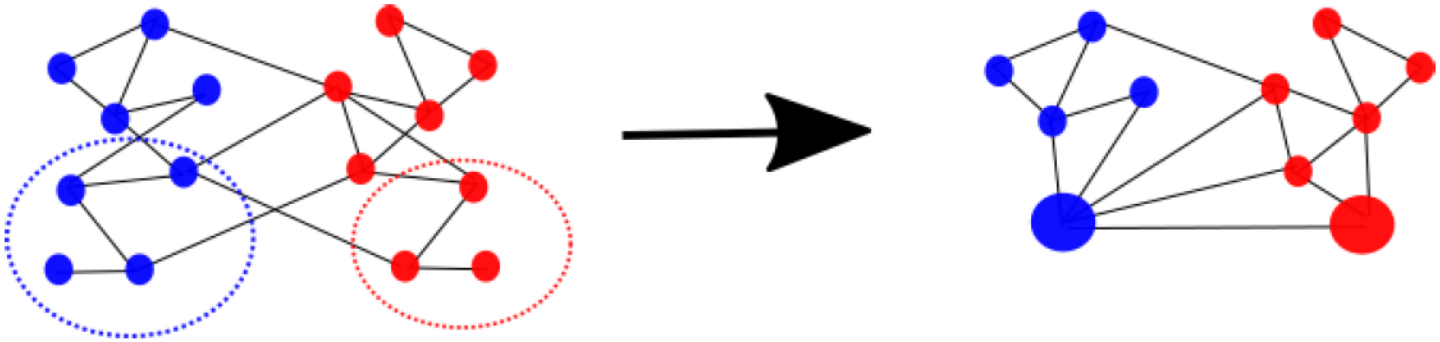
partitioning with metis...
Metis part sizes [64, 64, 62, 60]
METIS cost 20710

Graph with penalty constant $\alpha = \beta = 1$.

n	k	Metis	qbsolv
250	2	13773	13740
	4	20754	20686
	8	24173	24131
	16	26084	25920
500	2	55473	54885
	4	83207	82989
	8	97578	97369
	16	104438	104361
1000	2	223833	221643
	4	334939	334228
	8	391182	390684
	16	419539	418453

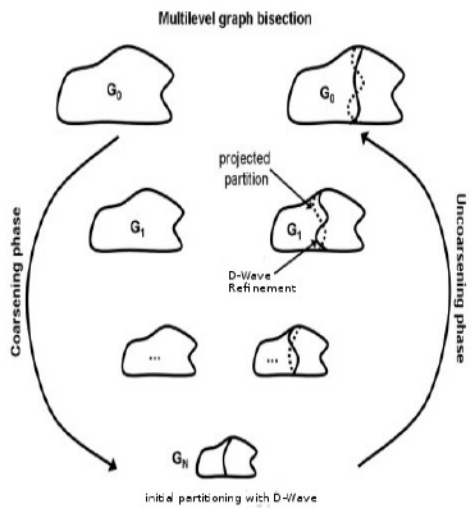
Refining a Partitioning using D-Wave

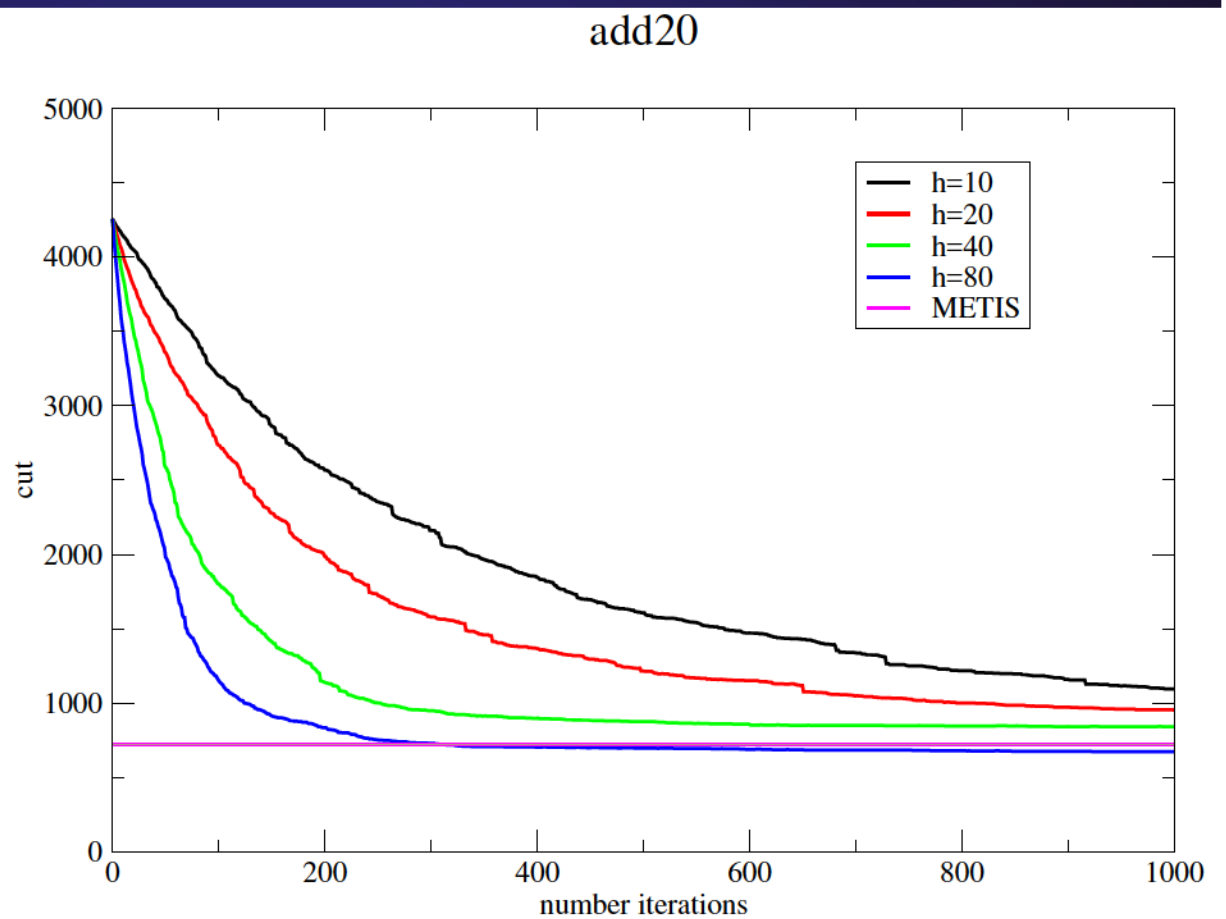
- **Qn:** Largest graph with sapi?
  - 40 nodes for an arbitrary graph on D-Wave 2X
- **Challenge:** How to partition a 1M node graph or larger?
- **Solution:** Use D-Wave as a refinement/improvement method
  - **Goal:** Design an iterative scheme for partitioning with D-Wave
  - **Input:** Initial partitioning
  - **Output:** Improved partitioning
  - **Procedure:** At each iteration partition a subgraph of G



D-Wave Multilevel Refinement Results

- Randomly partition h vertices at each iteration
- Walshaw benchmark graph
  - add20
    - 2395 vertices, 7462 edges





Future Plans

- **Further work on D-Wave multilevel refinement Graph Partitioning**
- **Develop "proof of principle" integration of D-Wave graph clustering/ partitioning into a QMD code as an accelerator**
- **Making codes and tutorials available**
- **Publish results**

  Graph Partitioning Using Quantum Annealing on the D-Wave

  H. Ushijima-Mwesigwa, C. F. A. Negre, S. M. Mniszewski

The End

Thank You!