Programming Interfaces
Overview

• **Framework for solving high-value applications as QUBOs**
  - Limitations imposed by hardware, and how to circumvent them
  - General purpose interfaces:
    • Integer programming
    • Problem descriptions in first order logic
  - Problem specific solutions
  - Accessibility
Compiling to QUBOs

• Many discrete optimization problems are not QUBOs, and not in a form to be executed on our hardware

• A compilation step which translates a high-level problem description down to the hardware h/J specification

• Issues addressed by compilation:
  — Multivariable interactions: reduce to 2-local
  — Connectivity mismatch: bring 2-local interaction graph to Chimera format
  — Decompose problems larger than available hardware
  — Resolve precision conflicts
Overcoming limitations: 2-local constraint

• Some problems naturally have interactions that couple many variables
  — Common example: alldiff(X,Y,Z) constraint
    • The values assumed by variables X,Y,Z must all be different

• Many ways to simplify higher-order interactions to 2-local, though all require the addition of extra variables (qubits)

• Often times the most qubit-efficient translation is to introduce variables representing products
  — The product z=x₁x₂ can be represented as global minima of the quadratic penalty
    H(x₁,x₂,z) = x₁x₂-2(x₁+x₂)z+3z
  — Thus, the 3-local term x₁x₂x₃ can be represented as minₗ {zx₃+λH(x₁,x₂,z)}
Overcoming limitations: connectivity

- The interactions between variables in any given problem will likely not match the connectivity of qubits.

- Current connectivity of qubits driven by hardware constraints.
- By ferromagnetically slaving qubits we can create graph minors of Chimera with additional connectivity.
- Largest complete graph that can be embedded in $C_n$ is $K_{4n+1}$.
- Can dynamically map a problem’s connectivity to Chimera – we have elected not to do so due to the overhead this introduces.
- We fix connectivity beforehand on a class of problems.
- In future we can design chips with problem-specific connectivity.

$C_3$: chimera connectivity
Overcoming limitations: small qubit numbers

• How do we solve a problem that has many more variables than we have qubits?

• Lots of alternatives, both problem specific and general purpose

• General purpose possibilities:
  – Large-neighbourhood local search
    • Hill-climb with a large neighbourhood
  – Cut-set conditioning:
    • Fix a subset of variables and condition on the values they can assume
  – Branch-and bound
    • Terminate search higher in tree by solving problem below a given node in hardware
Overcoming limitations: precision

• Given that a variable may be represented by more than 1 qubit the h/J parameters may be distributed across qubits and their interactions with other qubits
General programming models: integer programming

\[
\max_x \langle c, x \rangle \text{ such that } Ax \leq a, Bx = b, x \text{ integral}
\]

- Represent domain of \( x \) in binary
- Equality constraints: add as penalties
  \[ \lambda \| Bx - b \|^2 \]
- Inequality constraints: add slack variables to express as equality constraints
  \[ \mu \| Ax - a + s \|^2 \]
- QUBO objective
  \[ E(x, s) = \langle c, x \rangle + \lambda \| Bx - b \|^2 + \mu \| Ax - a + s \|^2 \]
- Penalty weights can be updated iteratively according to constraint violations
General programming models: constraint programming

• Given a set of n constraints \{C_1(x),...,C_n(x)\} find a feasible solution x that satisfies all constraints
  
  3-satisfiability:
  \[ or(x_1, x_3, x_7) \land or(x_2, x_4, x_7) \land or(x_5, x_6) \]

• QUBO representations of many constraints:

<table>
<thead>
<tr>
<th>(x + y \leq 1)</th>
<th>(xy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x + y \geq 1)</td>
<td>(1-x-y+xy)</td>
</tr>
<tr>
<td>(x + y = 1)</td>
<td>(1-x-y+2xy)</td>
</tr>
<tr>
<td>(x \leq y)</td>
<td>(x-xy)</td>
</tr>
<tr>
<td>(_\ldots)</td>
<td>(_\ldots)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(x + y + z \geq 1)</th>
<th>(1-x-y-z+xy+zx+yz-xyz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x + y + (1-z) \geq 1)</td>
<td>(z-xz-yz+xyz)</td>
</tr>
<tr>
<td>(x + (1-y) + (1-z) \geq 1)</td>
<td>(yz-xyz)</td>
</tr>
<tr>
<td>((1-x) + (1-y) + (1-z) \geq 1)</td>
<td>(xyz)</td>
</tr>
</tbody>
</table>

• We have an automated procedure to automatically generate constraints in QUBO form given any feasible set
  
  Often times it is necessary to introduce ancillary variables
  
  Automated procedure minimizes precision requirements, and respects connectivity constraints

• Constraint satisfaction problems have minimal precision constraints, and provide excellent test cases for early versions of hardware
Declarative Constraint Programming

• It is convenient to allow the description of constraint satisfaction problems in first order logic

• Graph Coloring

  – Instance data: Vertex(v), Edge(v₁, v₂)
  – Solution predicate: Color(v, c)
  – Problem specification:

    \[
    \forall v₁, v₂ \ (Edge(v₁, v₂) \rightarrow Color(v₁, c₁) \land Color(v₂, c₂) \land c₁ \neq c₂) \land \\
    \forall v \exists c \ Color(v, c) \land \\
    \forall v, c₁, c₂ \ \neg (Color(v, c₁) \land Color(v, c₂) \land c₁ \neq c₂)
    \]

• Description in FO logic allows

  – Separation between problem specification and problem data
  – Automated problem transformations
  – Problem optimizations
    • Variable elimination
    • Decomposition strategies
    • Inferences based on FO unit propagation to simplify problem
  – Efficient compilation to SAT
    • We compile to a slightly more general problem: SAT + cardinality constraints
  – Captures precisely the class NP
SQL interface

- Most users uncomfortable programming in first order logic
- Therefore, wrap first order logic in a familiar veneer → SQL

- SQL: database query language
  - Precise translation to first order logic
  - Close to data sources defining problem instances

- Working implementation @ http://sql.dwavesys.com/
  - Compiles an extended version of SQL to SAT + CC
    - Slightly more general semantics that conservatively extends SQL
    - Solves SAT+CC with modified state-of-the-art public domain solvers

- Adapt to hardware as hardware scales up
Problem-specific strategies

• **Machine learning**
  - Supervised learning: classification

• **Demo application**
  - Factoring of a biprime
Factoring

• An example of the kinds of optimizations possible on a specific problem
• Given a biprime \( p \) finds its factors \( a \) and \( b \)
• Naive approach: minimize \((p-ab)^2\)
  
  − Problems:
    • 4th order terms, lots of ancillary variables to reduce to quadratic
    • High connectivity between variables
    • High degree of precision required on parameter values

• A resolution
  
  − Write down the multiplication circuit realizing the product of \( a \) and \( b \)
  − Encode the circuit as minima of a suitably defined QUBO
  − Fix the circuit output to \( p \) and minimize the QUBO to find the inputs
  − Extremely sparse connectivity (fixed architecture)
  − Only 5 h/J values needed \{-2,-1,0,1,2\}
  − \( O(n^2) \) variables to factor a biprime of \( 2^n \) bits
  − Suggests annealing schedules
Accessibility: web services

Operational now
- State-of-the-art software solvers
- Same API as hardware comes online
- Submit problems from Java, Matlab, Python, C++, ...

http://apps.dwavesys.com/orionui