
# Quantum Computing

# Index

# QC Introduction (I)

- The power of QC comes from encoding information in a non-classical way (qubits)
- Enable computations to take advantage of purely quantum effects, such as quantum tunneling, quantum interference, and quantum entanglement, that are not available classically
- Quantum annealers  allows users to program them without needing to know about the underlying quantum mechanical effects
- Quantum annealing (QA) is a metaheuristic optimization algorithm that makes use of quantum effects
- QA are designed to minimize Quadratic Unconstrained Binary Optimization (QUBO) problems

# QC Introduction (II)

- QA should not be confused with adiabatic QC (known to support universal QC)
  - The Hamiltonian problem in QA is a classical Hamiltonian
  - Adiabatic QC also interpolates between an initial and final Hamiltonian, the final Hamiltonian can be highly non-classical with no analogous classical cost function, thus enabling much more general sorts of quantum computations
- Hamiltonian is the operator corresponding to the total energy of the system (H) = Kinetic (T) and Potential (V) energies

$$\hat{H} = \hat{T} + \hat{V}$$

# QC Introduction (III)

- What is Quantum Annealing?
- How the AQ process works?
- Physics of QA
- Measuring Q Physics to solve problems

# Index

# QC basics (I)

- Model: A collection of variables with associated linear and quadratic biases
- Sampler: A process that samples from low energy states of a problem's objective function
- A binary quadratic model (BQM) sampler samples from low energy states in models such as those defined by an Ising equation or a QUBO problem and returns an iterable of samples, in order of increasing energy
- A dimod sampler provides 'simple qubo' and 'sample_ising' methods as well as the generic BQM sampler method

# QC basis (II)

- Steps for a Programming a Q-annelaer
  1. Model the problem as a binary CSP (not in all the cases)
  2. Map the problem into a QUBO
  3. Embed the QUBOs using solvers such as:
     - Qbsolv
     - Minus-Minus
  4. Solve the different subproblems (sub-graphs) by mapping them to the D-wafe chip, or instead use classical QUBO algorithms

# 1. Binary CSP (I)

- A constraint satisfaction problem (CSP) is binary if all constraints are either unary or binary relations (wrt the decision variables)
- The cardinality of the scope of the constraint is either 1 or 2
- For example, x > 2 (unary) or : x !=y (binary)
- D-wave implements a CSP class (see the manual reference)

# 1. Binary CSP (II)

- D-wave implementation:
  - **classConstraintSatisfactionProblem**(***kwargs*)
- This example creates a binary-valued constraint satisfaction problem, adds two constraints, $a$ = $b$ and $b$ != $c$, and tests $a,b,c$ = 1,1,0

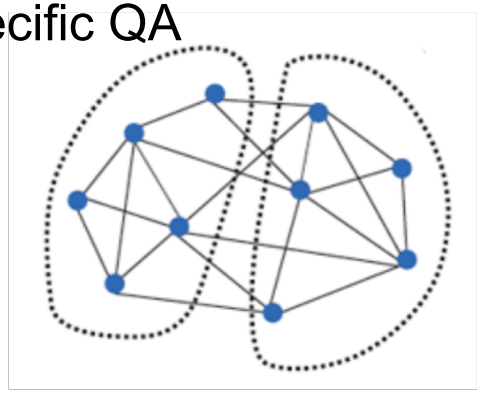
```
>>> import dwavebinarycsp
>>> import operator
>>> csp = dwavebinarycsp.ConstraintSatisfactionProblem('BINARY')
>>> csp.add_constraint(operator.eq, ['a', 'b'])
>>> csp.add_constraint(operator.ne, ['b', 'c'])
>>> csp.check({'a': 1, 'b': 1, 'c': 0})
True
```

# 2. Mapping: QUBO

- Choose binary variables, formulate constraints, and embed the violation of constraints as energy penalties
- A QUBO is a minimization function that incorporates:
  - Constraints: the rules that we have to follow
  - Objective: what we want to minimize
- Steps for a QUBO process
  1. Write out the objective and constraints in the problem
  2. Convert the objective and constraints to math statements with binary variables
  3. Make the objective and constraints “QUBO appropriate”
     - Objective is a minimization function
     - Constraints are satisfied at their minimum values
  4. Combine the objective and constraints.
- QUBO = min (**objective**) + y(**constraints**)

# 3. Embedding

- Hw has low connectivity, no direct one-to-one mapping between the QUBO binary variables and the physical qubits
- Once the QUBO problem is formulated, takes these hardware-independent QUBOs to other QUBOs that match the specific QA hardware that will be used

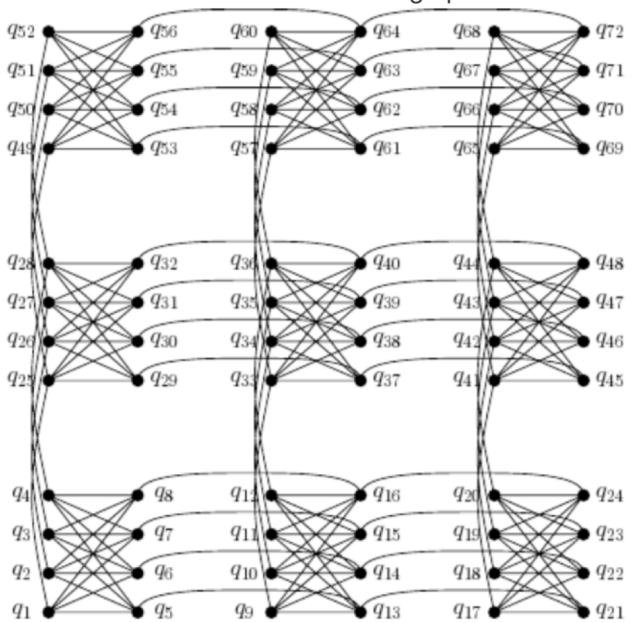


- Quantum solutions:
  - Qbsolv: handles large problems by automatically breaking them down into smaller segments that can run individually on D-Wave's
  - Minor-Minor: is a heuristic tool for minor embedding: given a minor and target graph, it tries to find a mapping that embeds the minor into the target

# 4. Solve the problem

- The layout of the qubits and couplers of a D-Wave quantum annealer is a n × n lattice of unit cells called a Chimera graph
- Each unit cell is composed of a bi-partite graph of 8 qubits
- D-Wave machine @ NASA 12 × 12 such units and a total of 1152 qubits, of which 1097 are working
- Each qubit is coupled to at most 6 other qubits, 4 within its own unit cell and 2 to qubits in its neighboring cells
- To embed a generic QUBO of N variables, $N^2$ qubits and couplers are needed in the worst case so that each binary variable can be represented by N physical qubits and effectively couple to all other binary variables

9 units cells in a chimera graph

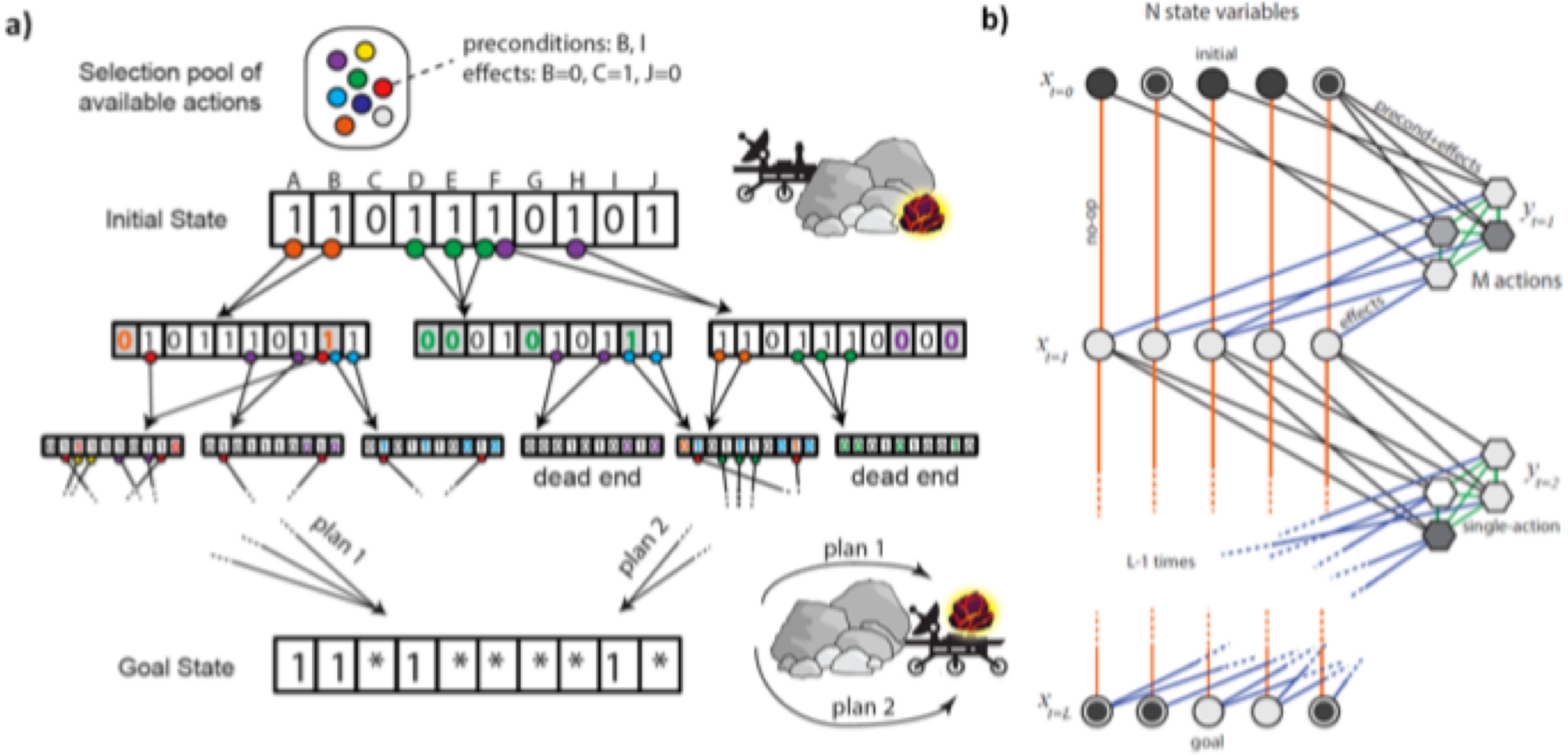


- In case HW does not provide a solution for a high-size QUBO problem, use instead QUBO classical resolution algorithms
  - HFS, etc

# Index

# NASA Applications

- Quantum annealing for P & S
  - The initial state (e.g., Rover behind the rocks without sample) is specified by assigning True (1) False (0)
  - Path represents a sequence (with possible repetitions) of actions selected from a pool (colors).



# NASA Applications

- Fault detection for electrical power-distribution
  - Diagnosing the minimal number of faults capable of explaining a set of given observations, e.g., from sensor readouts