PLANNING AND GAMES

ENCODING IN SAT AND QBF

… AND QUANTUM CIRCUIT COMPILATION

IRFANSHA SHAIK

JACO VAN DE POL

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

PLAN

1. **Motivating example: Quantum Circuit Layout Synthesis**

2. Classical Planning / Bounded Model Checking (SAT)

3. Concise Encoding of Planning in QBF (Quantified Boolean Formulas)
    - Path Compression
    - Lifted Planning (almost first-order)

4. Concise Encoding of 2-player board games in QBF
    - Tic-tac-toe, Hex, Breakthrough, Domineering

5. Validation of encodings with QBF certificates

INTRODUCTION – LAYOUT SYNTHESIS

- *Optimal Layout Synthesis for Quantum Circuits as Classical Planning,*
  Irfansha Shaik and Jaco van de Pol. ICCAD'23 IEEE/ACM, San Francisco, 2023

Quantum algorithms can solve some problems faster than classical algorithms

Quantum Computers exist today … NISQ era

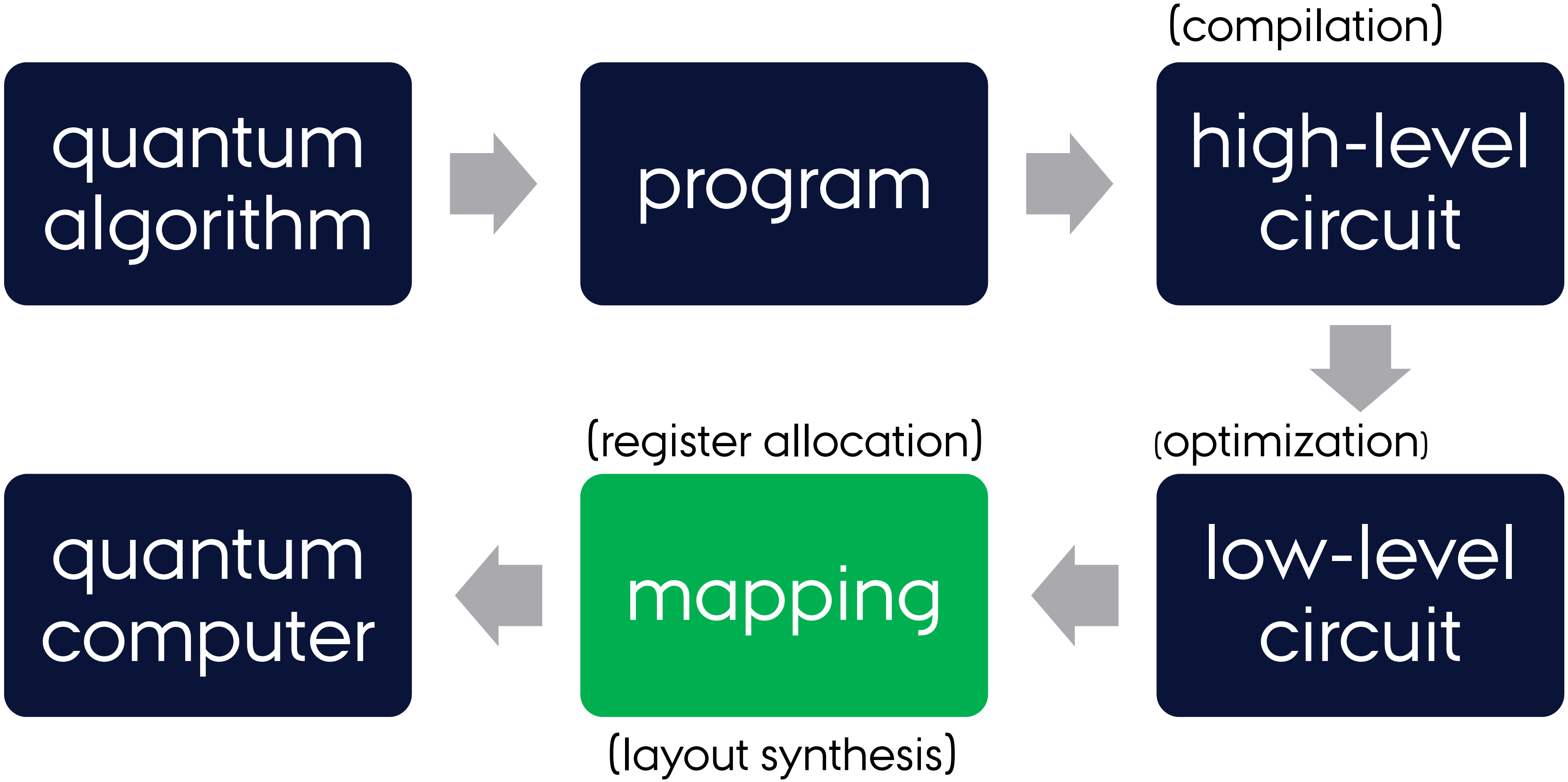
- Intermediate scale (limited number of qubits, limited connectivity)
- Noisy (decoherence, interference)

Circuit Optimization: minimize the number of gates / depth of circuit by rewriting the circuit

Layout Synthesis: Map "logical quantum circuit" to a "physical platform"

- Swap qubits around, to obey connectivity restrictions
- Every swap increases the noise, so minimize this!

A SMALL STEP IN THE "QUANTUM PIPELINE"



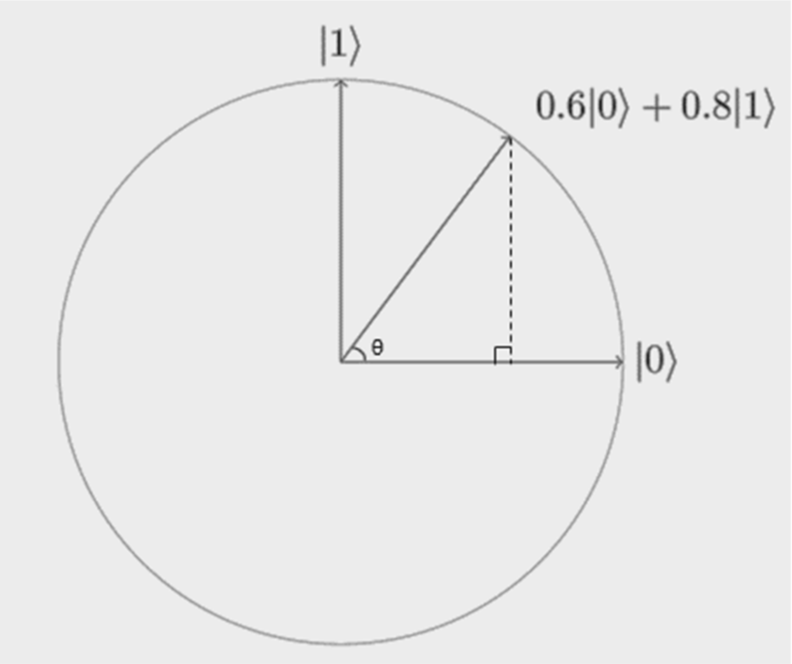
quantum algorithm → program → (compilation) high-level circuit → (optimization) low-level circuit → (register allocation) mapping (layout synthesis) → quantum computer

QUBITS AND QUANTUM GATES (HEAVILY SIMPLIFIED)

Qubits:

Basic vectors: $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$

Arbitrary qubit: $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ with $|\alpha|^2 + |\beta|^2 = 1$



1 qubit Quantum Gates: $I, X, Y, Z, H, S, T, T^\dagger$, …

Identity = I = $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\quad I(x) = x$

NOT = X = $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $\quad NOT(x) = \neg x$

MULTI-QUBIT QUANTUM GATES (CLASSICAL VIEW)

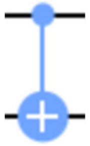
Multiple qubits: $|\phi\psi\rangle = |\phi\rangle \otimes |\psi\rangle$ (tensor product)

$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$, $|01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}$, $|10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}$ and $|11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$

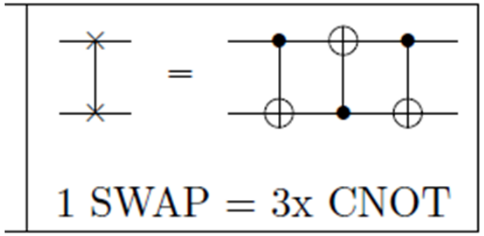
n qubits: 2^n-dimensional Hilbert space

Binary operators:

$$\text{CNOT} = \text{CX} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$



$$CNOT(x,y) = (x, x \oplus y)$$



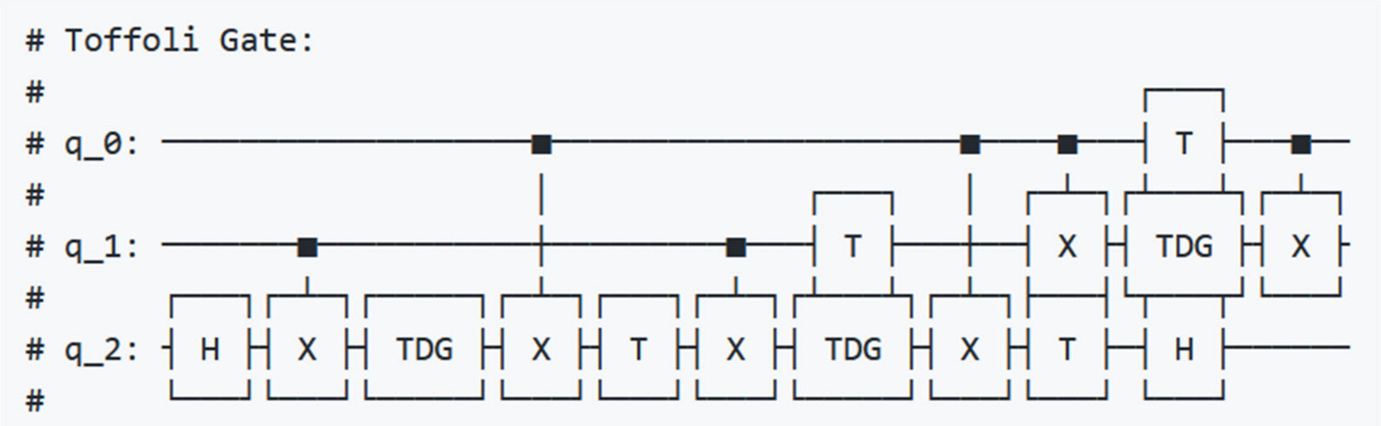
1 SWAP = 3x CNOT

$$\text{SWAP} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

$$\text{SWAP}(x,y) = (y,x)$$

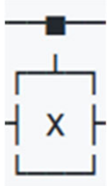
LAYOUT SYNTHESIS

(FROM GITHUB.COM/UCLA-VAST/OLSQ)



Toffoli Gate:

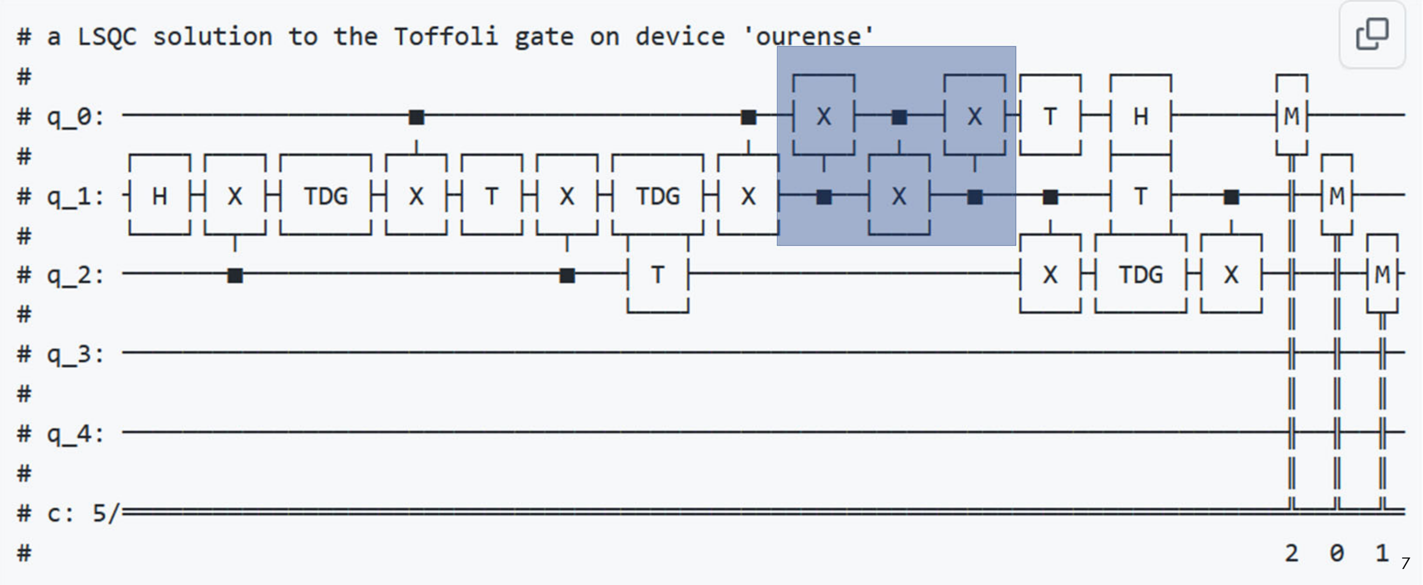
All gates (H,X,T) are unary except binary CNOT gate:



Requires full connectivity $(q_0, q_1), (q_1, q_2), (q_0, q_2)$

What if the target only has $(q_0, q_1), (q_1, q_2)$?

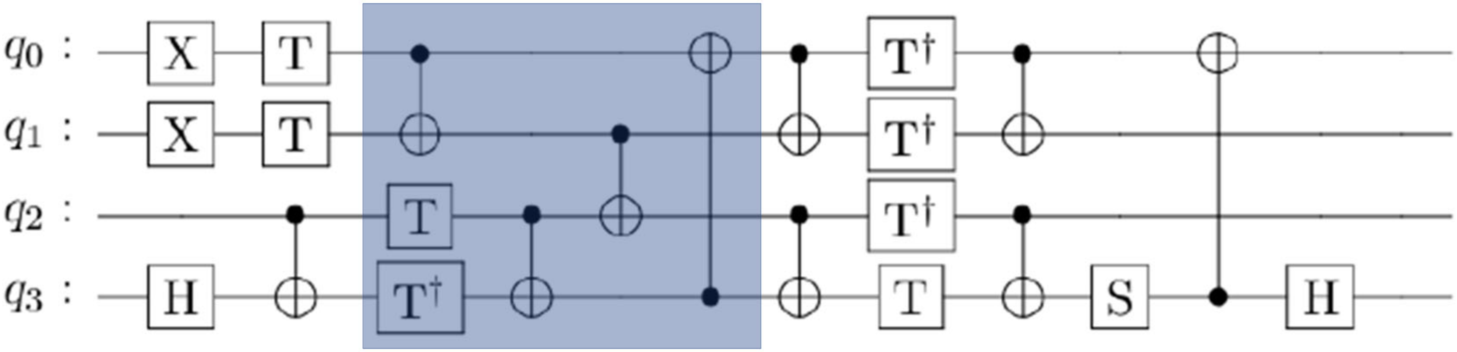
1 SWAP by 3 CNOT gates



a LSQC solution to the Toffoli gate on device 'ourense'

LAYOUT SYNTHESIS: ADDER CIRCUIT

First input: a (logical) quantum circuit

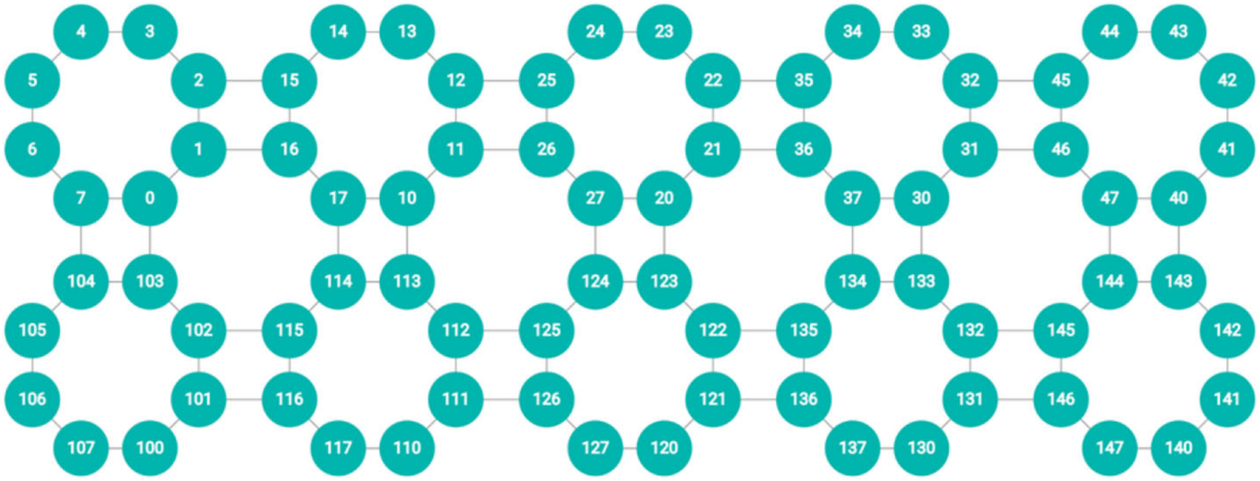


Note: Need a square: $(q_0, q_1), (q_1, q_2), (q_2, q_3), (q_3, q_1)$

What if the physical platform only has $(q_0, q_1), (q_1, q_3), (q_2, q_3)$?

PHYSICAL PLATFORMS

Second input: a (directed) Coupling Map



Rigetti Aspen-3, 80 qubits (source: aws.amazon.com)

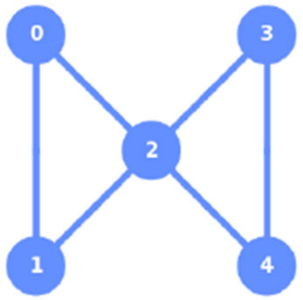


Fig. 3. Coupling map for IBM-QX2 (Tenerife)

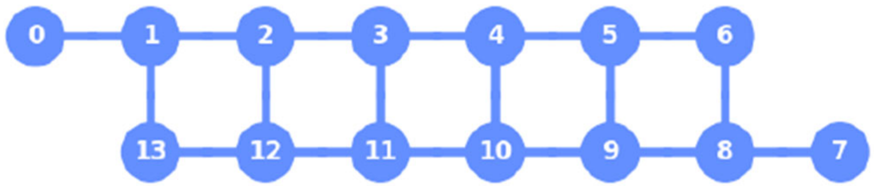
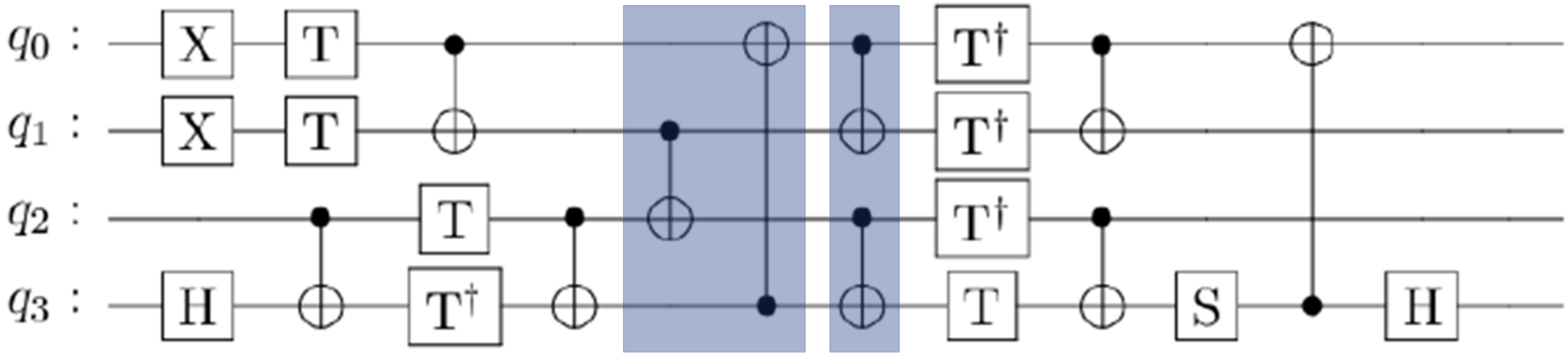
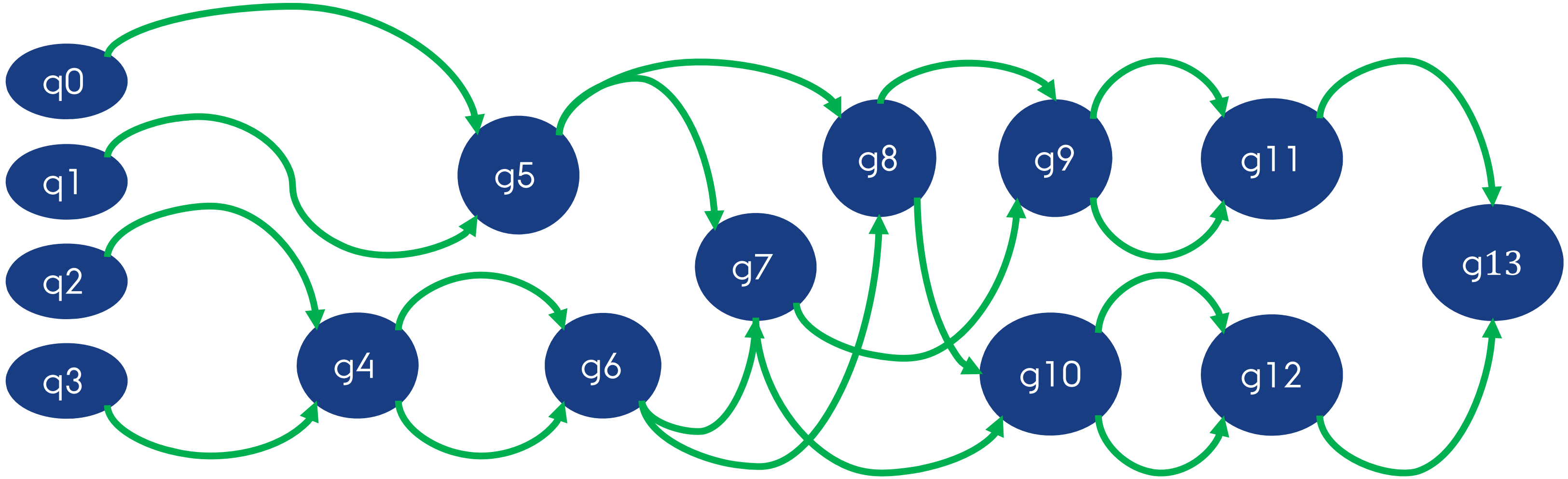


Fig. 4. Coupling map for IBM Melbourne

DEPENDENCY GRAPH (DAG)

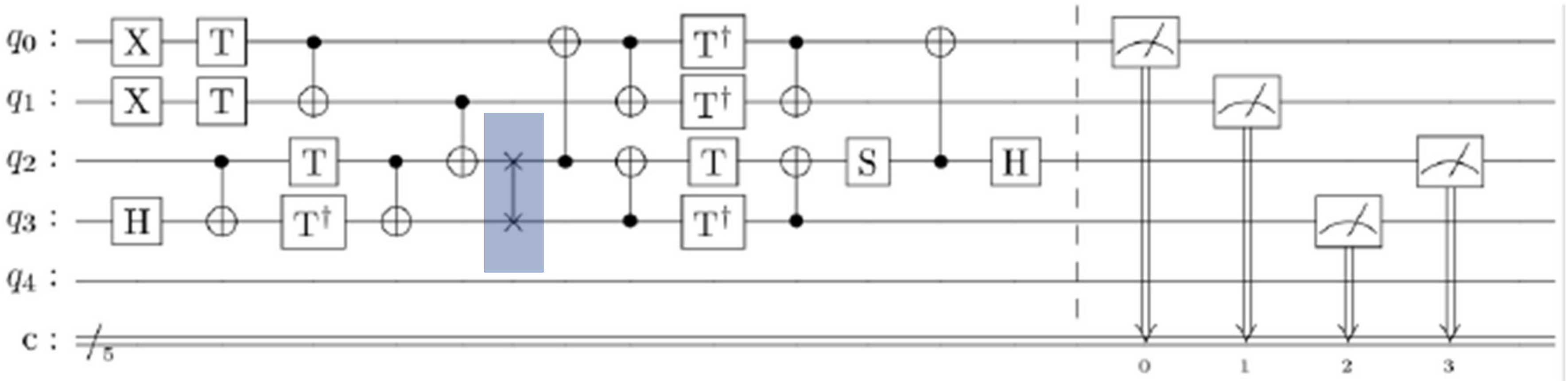


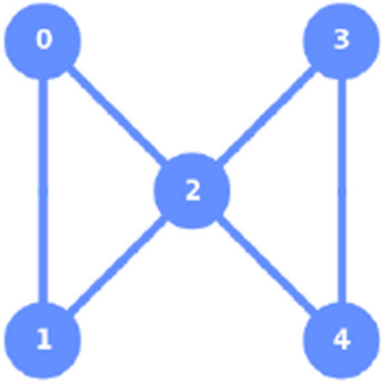
For layout mapping, we can focus on the CNOT gates and ignore all unary gates



ADDER – ON 5-QUBIT PLATFORM

Result: a new quantum circuit, observing "neighbours", inserting minimal #SWAP gates





Apparently, the optimal solution for this example uses only 1 SWAP

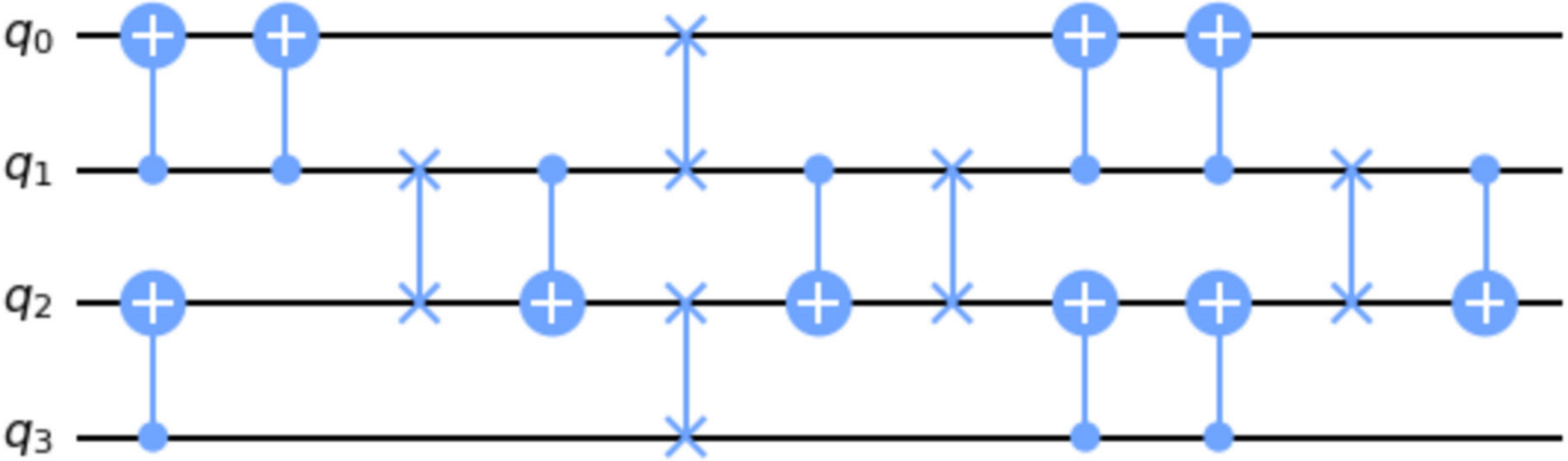
Note: after the swap, the "logical qubits" are on different "physical qubits"

(indicated by the "measurements")

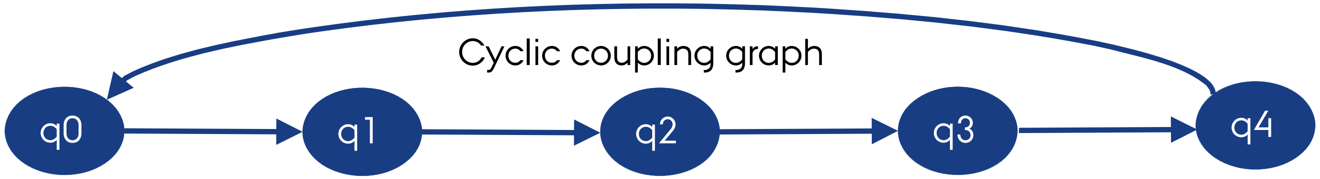
Complicated factor: in general, one may need extra "ancillary" qubits! (q4)

MORE CHALLENGING – RESTRICT PLATFORM

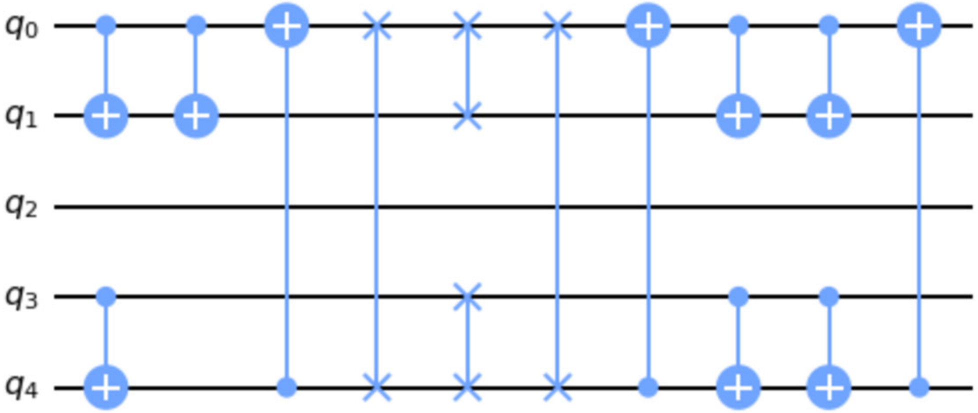
Optimal number of swaps: 5



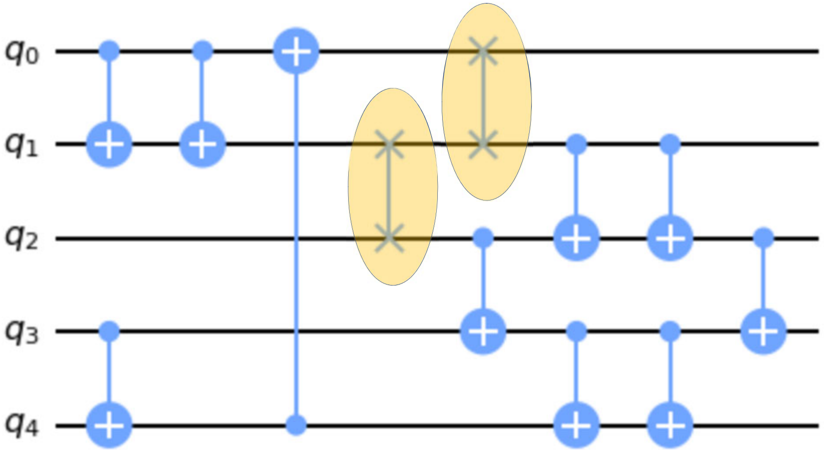
NEED FOR ANCILLARY QUBITS



Without using ancillary bits (4 swaps)



With using ancillary bits (2 swaps)



LESSONS INSPIRED BY CONCURRENCY

The notion of equivalence determines "optimality"

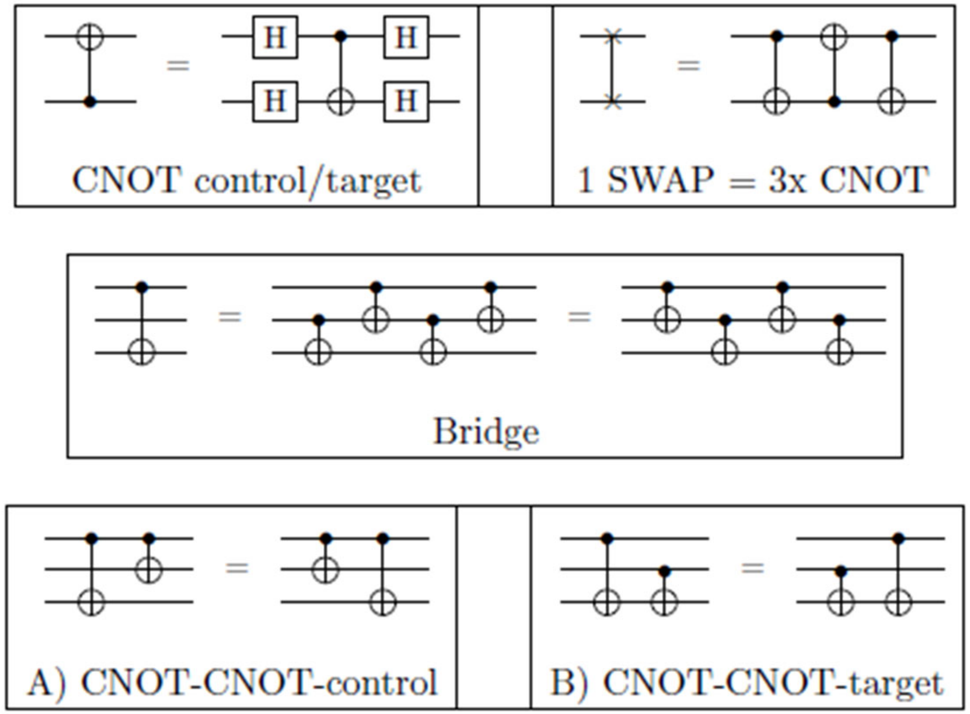
Should we preserve

- the same list of gates?
- only "local" dependency DAG? (POR)
- "layers" in a topological sorting? (parallelism)

Adder example		
On cycle-platform	With ancillary	2 swaps
	Without ancillary	4 swaps
On star-platform	Total topological order	8 swaps
	Partial order preserved	6 swaps

Can we take advantage of "semantic" equalities?

Where is the limit? Complete re-synthesis?



[Itoko et al., in: Integration, 2020]

PLAN

1. Motivating example: Quantum Circuit Layout Synthesis
2. **Classical Planning / Bounded Model Checking (SAT)**
3. Concise Encoding of Planning in QBF
    - Path Compression
    - Lifted Planning (almost first-order)
4. Concise Encoding of 2-player board games in QBF
    - Tic-tac-toe, Hex, Breakthrough, Domineering
5. Validation of encodings with QBF certificates

CLASSICAL PLANNING (PDDL)

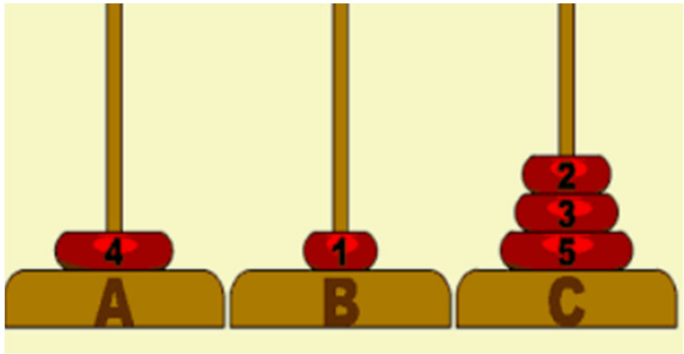
Domain description:

States:

- Described by predicates
- Can be tested and updated

Actions: (parametric)

- Described by pre-conditions and effects
- Both are conjunctions of predicates

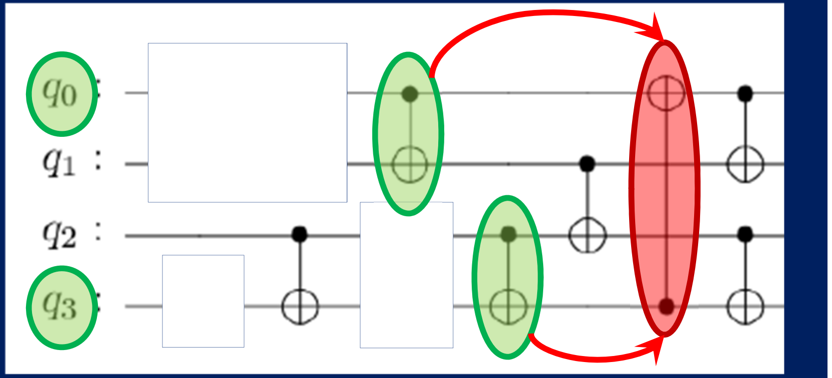


Problem instance:

- Concrete objects
- Initial state, Goal state(s)

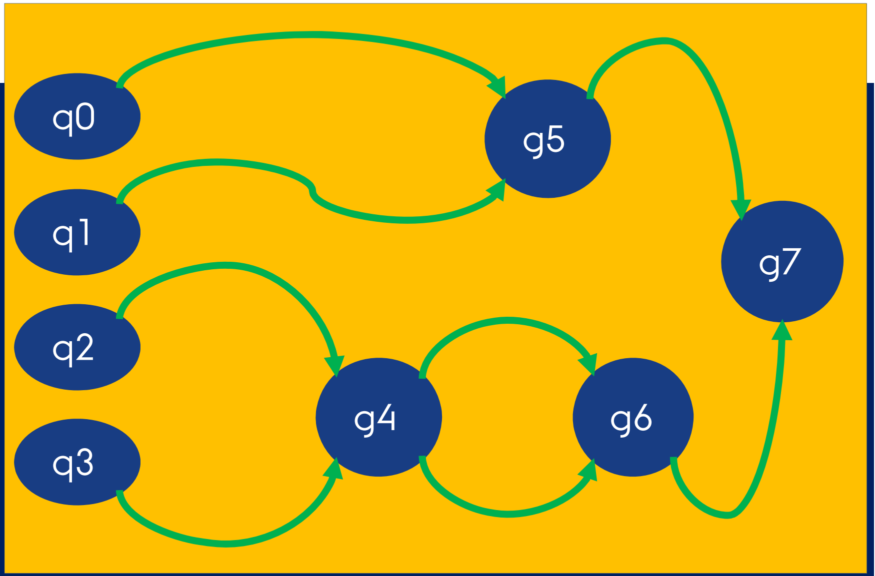
LAYOUT SYNTHESIS → PLANNING

(OUTSTANDING DOMAIN SUBMISSION AWARD IPC-2023)



```
(define (domain quantum)
(:types
    gate - object   ; binary or input gate
    pbit - object   ; physical qubit
    lbit - gate     ; logical qubit
)
(:predicates
     (mapped ?l - lbit  ?p - pbit)                   ; l is mapped on p
     (used ?p - pbit)                                ; p is in use
     (done ?g - gate)                                ; gate g is done
     (neighbour ?p1 ?p2 - pbit)                      ; static connections
     (cnot ?l1 ?l2 - lbit  ?g0 ?g1 ?g2 - gate)      ; gate dependencies
) … )
```

LAYOUT SYNTHESIS: PROBLEM FILE



```
(define (problem adder) (:domain quantum)
(:objects
    l0 l1 l2 l3 - lbit
    p0 p1 p2 p3 p4 - pbit
    g4 g5 g6 g7 g8 g9 g10 g11 g12 g13 - gate
)
(:init ; static predicates: platform & circuit
    (neighbour p0 p1) (neighbour p1 p0) (neighbour p3 p4) ...
    (cnot l2 l3 g4 l2 l3) (cnot l0 l1 g5 l0 l1) (cnot l1 l2 g7 g5 g6) ...
)
(:goal
    (and (done g13))
))
```

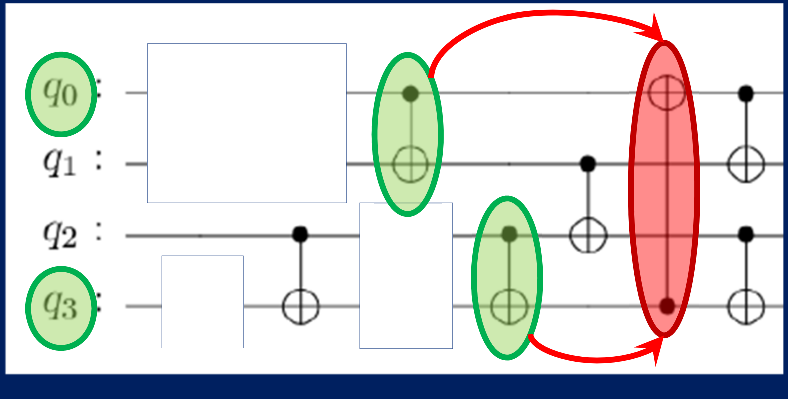
ACTIONS: MAP INITIAL + SWAP

```
(:action map_init
    :parameters
        (?l1 - lbit ?p1 - pbit)
    :precondition (and
        (not (used ?p1))
        (not (done ?l1))
    )
    :effect (and
        (mapped ?l1 ?p1)
        (used ?p1)
        (done ?l1)
    )
)
```

```
(:action swap
    :parameters
        (?l1 ?l2 - lbit ?p1 ?p2 - pbit)
    :precondition (and
        (neighbour ?p1 ?p2)
        (mapped ?l1 ?p1)
        (mapped ?l2 ?p2)
    )
    :effect (and
        (not (mapped ?l1 ?p1))
        (not (mapped ?l2 ?p2))
        (mapped ?l1 ?p2)
        (mapped ?l2 ?p1)
    )
)
```

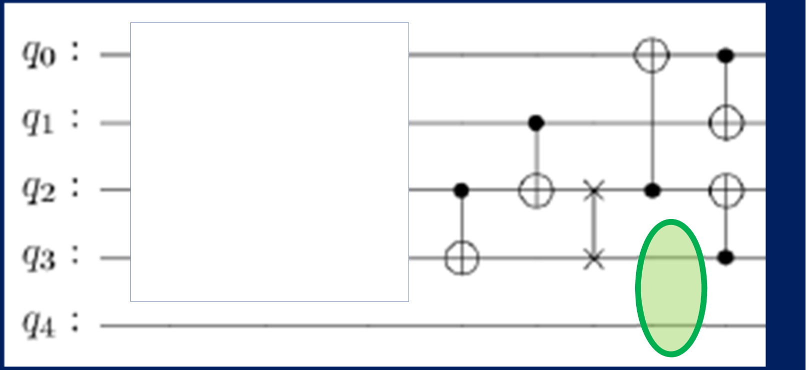
ACTIONS: APPLY A CNOT GATE

```
(:action apply_cnot
    :parameters
        (?l1 ?l2 - lbit ?p1 ?p2 - pbit ?g0 ?g1 ?g2 - gate)
    :precondition (and
        (cnot ?l1 ?l2 ?g0 ?g1 ?g2)
        (neighbour ?p1 ?p2)
        (mapped ?l1 ?p1) (mapped ?l2 ?p2)
        (done ?g1) (done ?g2) (not (done ?g0))
    )
    :effect (and
        (done ?g0)
    )
)
```



COMPLETENESS: USE ANCILLARY QUBIT

```
(:action use_ancillary
    :parameters
        (?l1 - lbit ?p1 ?p2 - pbit)
    :precondition (and
        (neighbour ?p1 ?p2)
        (mapped ?l1 ?p1)
        (not (used ?p2))
    )
    :effect (and
        (not (mapped ?l1 ?p1)) (not (used ?p1))
        (mapped ?l1 ?p2) (used ?p2)
)   )
```



we also need the symmetric one with (neighbor ?p2 ?p1)

EFFICIENCY: COMBINE INIT_MAP IN CNOT

```
(:action apply_cnot_input_input
    :parameters (?l1 ?l2 - lbit ?p1 ?p2 - pbit ?g0 - gate)
    :precondition (and
        (cnot ?l1 ?l2 ?g0 ?l1 ?l2)
        (neighbour ?p1 ?p2)
        (not (used ?p1)) (not (used ?p2))
        (not (done ?g0)) (not (done ?l1)) (not (done ?l2))
    )
    :effect (and
        (done ?g0) (done ?l1) (done ?l2)
        (mapped ?l1 ?p1) (used ?p1)
        (mapped ?l2 ?p2) (used ?p2)
)   )
```

we also need two variants with one input and one gate

EFFICIENCY 2: GROUNDING ALL GATES

```
(:action apply_cnot_g7        ; g7 depends on CNOT gates g5, g6
    :parameters (?p1 ?p2 - pbit)
    :precondition (and
        (neighbor ?p1 ?p2)
        (mapped l1 ?p1) (mapped l2 ?p2)
        (done g5) (done g6) (not (done g7)))
    :effect (and (done g7)))
(:action apply_cnot_g4        ; g4 depends on input gates l2, l3
    :parameters?p1 ?p2 - pbit)
    :precondition (and (neighbor ?p1 ?p2)
       (not (used ?p1)) (not (used ?p2)) (not (done g4)))
    :effect (and (done g4)
        (mapped l2 ?p1) (used ?p1)
        (mapped l3 ?p2) (used ?p2)))
```

IMPLEMENTATION: Q-SYNTH

Our tool Q-SYNTH

Implemented in Python:

- Input: quantum circuit + coupling graph → Output: PDDL planning instance
- Uses QISKIT for I/O and computing the dependency graph

We need a "planning tool", running in "optimal mode"

→ take winners from IPC – international planning competitions

- Madagascar – SAT based (similar to BMC)
- Fast Downward Soup

Shortest Plan → quantum circuit: re-insert the unary gates

COMPARING TO RELATED WORK

SABRE (QISKIT): Tackling the qubit mapping problem for NISQ-era quantum devices
G. Li, Y. Ding, and Y. Xie (ASPLOS 2019)

- Heuristic, fast, but not-optimal

QMAP: Mapping Quantum Circuits to IBM QX Architectures
Robert Wille, Lukas Burgholzer and Alwin Zulehner (DAC-2019)

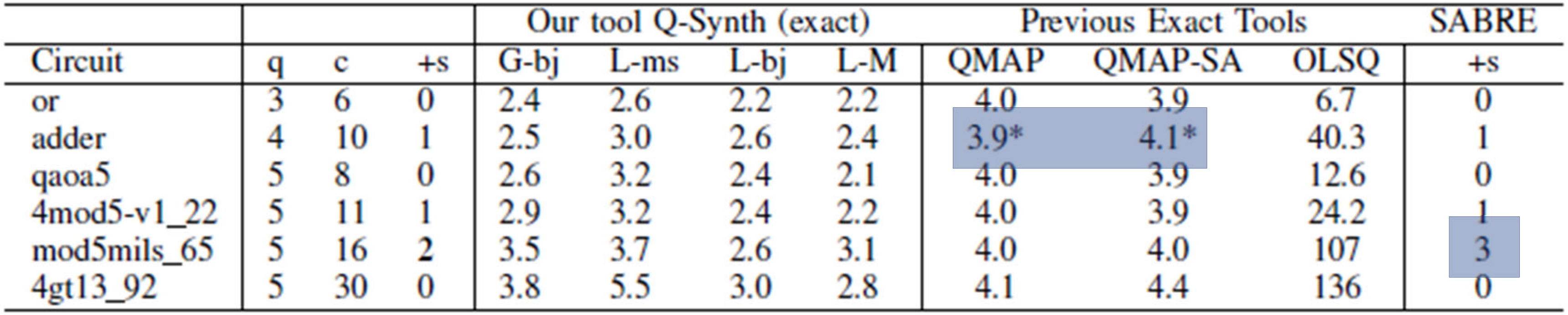
- Used **SAT + SMT** technology
- Considers all permutations of qubits

OLSQ: Optimal Layout Synthesis for Quantum Computing,
Bochen Tan and Jason Cong (ICCAP 2020)

- Also **SAT + SMT**, exponentially less variables, avoids permutations

SMALL PLATFORM – 5 QUBITS

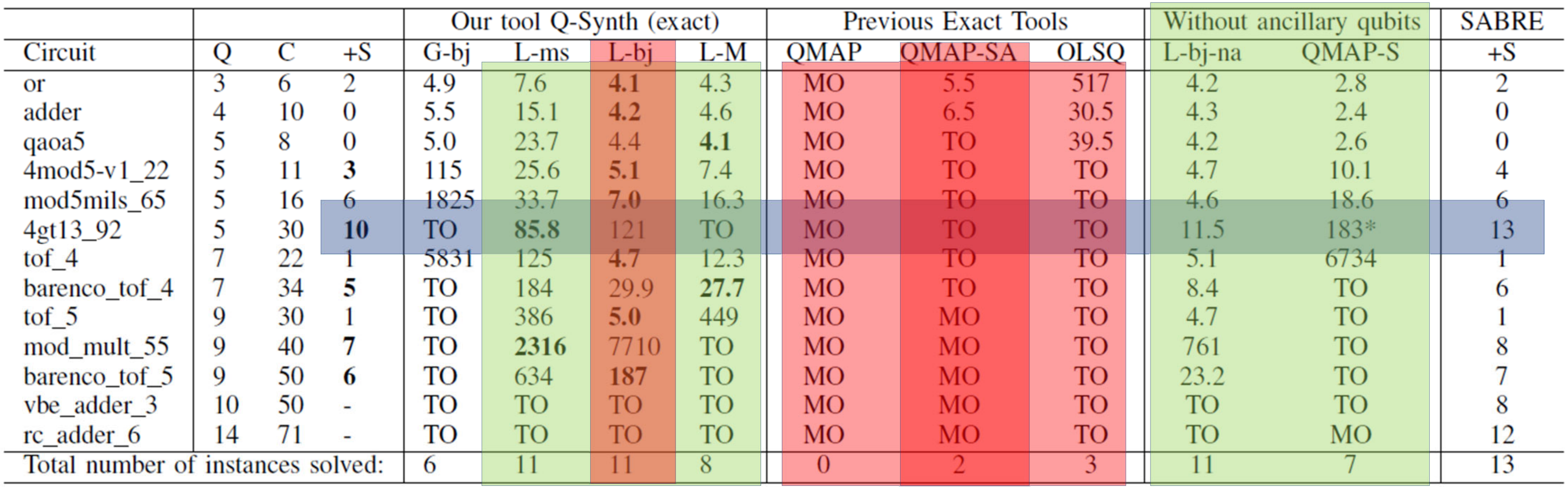
(TIME IN SEC)



				Our tool Q-Synth (exact)				Previous Exact Tools			SABRE
Circuit	q	c	+s	G-bj	L-ms	L-bj	L-M	QMAP	QMAP-SA	OLSQ	+s
or	3	6	0	2.4	2.6	2.2	2.2	4.0	3.9	6.7	0
adder	4	10	1	2.5	3.0	2.6	2.4	3.9*	4.1*	40.3	1
qaoa5	5	8	0	2.6	3.2	2.4	2.1	4.0	3.9	12.6	0
4mod5-v1_22	5	11	1	2.9	3.2	2.4	2.2	4.0	3.9	24.2	1
mod5mils_65	5	16	2	3.5	3.7	2.6	3.1	4.0	4.0	107	3
4gt13_92	5	30	0	3.8	5.5	3.0	2.8	4.1	4.4	136	0

- Note: all tools work, Q-synth is fast and always optimal
  - G-bj: Global Encoding (lifted, separate init)
  - L-* : Local Encoding (grounded, integrated init)
- SABRE is non-optimal (by design)
- QMAP is non-optimal in a few cases (total order)
- OLSQ might also be non-optimal (fixed horizon)

LARGER PLATFORM – 14 QUBITS (3HR TIMEOUT, 48 GB MEM)



				Our tool Q-Synth (exact)				Previous Exact Tools			Without ancillary qubits		SABRE
Circuit	Q	C	+S	G-bj	L-ms	L-bj	L-M	QMAP	QMAP-SA	OLSQ	L-bj-na	QMAP-S	+S
or	3	6	2	4.9	7.6	**4.1**	4.3	MO	5.5	517	4.2	2.8	2
adder	4	10	0	5.5	15.1	**4.2**	4.6	MO	6.5	30.5	4.3	2.4	0
qaoa5	5	8	0	5.0	23.7	4.4	**4.1**	MO	TO	39.5	4.2	2.6	0
4mod5-v1_22	5	11	**3**	115	25.6	**5.1**	7.4	MO	TO	TO	4.7	10.1	4
mod5mils_65	5	16	6	1825	33.7	**7.0**	16.3	MO	TO	TO	4.6	18.6	6
4gt13_92	5	30	**10**	TO	**85.8**	121	TO	MO	TO	TO	11.5	183*	13
tof_4	7	22	1	5831	125	**4.7**	12.3	MO	TO	TO	5.1	6734	1
barenco_tof_4	7	34	**5**	TO	184	29.9	**27.7**	MO	TO	TO	8.4	TO	6
tof_5	9	30	1	TO	386	**5.0**	449	MO	MO	TO	4.7	TO	1
mod_mult_55	9	40	**7**	TO	**2316**	7710	TO	MO	MO	TO	761	TO	8
barenco_tof_5	9	50	**6**	TO	634	**187**	TO	MO	MO	TO	23.2	TO	7
vbe_adder_3	10	50	-	TO	TO	TO	TO	MO	MO	TO	TO	TO	8
rc_adder_6	14	71	-	TO	TO	TO	TO	MO	MO	TO	TO	MO	12
Total number of instances solved:				6	11	11	8	0	2	3	11	7	13

- SABRE is non-optimal: can use 13 Swaps instead of 10 = 9 extra CNOT gates (!)
- Q-SYNTH is the only exact tool that scales: *9 logical qubits on 14 physical qubits*
- The planner backends are somewhat complementary
- Planning without using ancillary bits is considerably easier

CONCLUSION, PERSPECTIVES

- Classical Planning for optimal layout synthesis gives superior results
- Classical Planning could also make use of heuristic planners (suboptimal plans)

Even better encodings are possible, to limit the search space

- The sub-architecture technique from QMAP can also be applied in Q-SYNTH
- Parallel plans, exploit symmetries, relaxed dependencies

- ***Current/Future work:*** *Direct SAT encoding, …*
- ***Future work:*** *Other costs (swap depth, noise reduction, …)*
- ***Future work:*** *Quantum Circuit Optimization seems a much harder problem*

PLAN

1. Motivating example: Quantum Circuit Layout Synthesis

2. **Classical Planning / Bounded Model Checking (SAT)**

3. Concise Encoding of Planning in QBF
   - Path Compression
   - Lifted Planning (almost first-order)

4. Concise Encoding of 2-player board games in QBF
   - Tic-tac-toe, Hex, Breakthrough, Domineering

5. Validation of encodings with QBF certificates

PLANNING: BOUNDED MODEL CHECKING

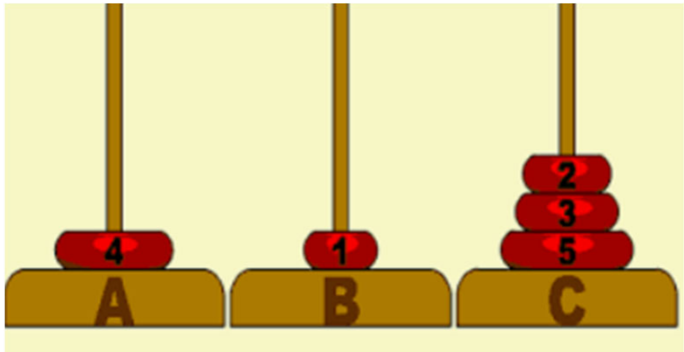
Domain description:

States (S): Described by predicates

Actions (A): Described by pre-conditions and effects

Problem instance:

- Concrete objects
- Initial state (I), Goal state (G)



Bounded Model Checking / Planning as Satisfiability [Kautz, Selman 1992]

- State variables S_i per time step
- Transition relation A captures actions
- Reduce to SAT solver (plan of 2 actions)

$$\exists S_0, S_1, S_2 : I(S_0) \land A(S_0, S_1) \land A(S_1, S_2) \land G(S_2)$$

LIMITATION OF BMC WITH SAT IN PLANNING

The encoding may become very large:

- The transition relation can become quite large
  - **Grounding**: Instantiate actions for all **object combinations**

$$\boldsymbol{apply\_cnot(l_1, l_2, p_1, p_2, g_1, g_2, g_3)}$$

  - **Copying:** The transition relation is copied for each step
- Existing solution (in planning and bounded model checking):
  - “Iterative squaring”, also known as “path compression”
  - Encode in QBF: only one copy of transition relation A

QBF AND CONCISE ENCODING

QBF extends Propositional Logic with quantifiers over propositions

Only one of the following formulas is true:

1. $\forall x\, \exists y : x \land y$
2. $\exists y\, \forall x : x \land y$
3. $\forall x\, \exists y : x \leftrightarrow y$
4. $\exists y\, \forall x : x \leftrightarrow y$

- Let ϕ be a very large formula: how can we encode $\phi(A) \land \phi(B)$ concisely?

$$\forall x : (x = A \lor x = B) \to \phi(x)$$

- Here we use only one copy of ϕ

QBF SOLVERS

SAT is a special case of QBF – only ∃-quantifiers – NP-hard

In general, solving QBF is PSPACE-complete

The **quantifier alternation depth** is a crucial factor for complexity

Practical QBF solvers exist, there is also a yearly QBF-EVAL competition

(we contributed our planning and game encodings to QBF-EVAL)

Example QBF solvers:

CAQE	[Rabe, Tentrup]
DepQBF	[Lonsing, Biere]
…	

We proposed a translator from PDDL to QBF (ICAPS 2022)

PLAN

1. Motivating example: Quantum Circuit Layout Synthesis

2. Classical Planning / Bounded Model Checking (SAT)

3. **Concise Encoding of Planning in QBF**
    - **Path Compression**
    - Lifted Planning (almost first-order)

4. Concise Encoding of 2-player board games in QBF
    - Tic-tac-toe, Hex, Breakthrough, Domineering

5. Validation of encodings with QBF certificates

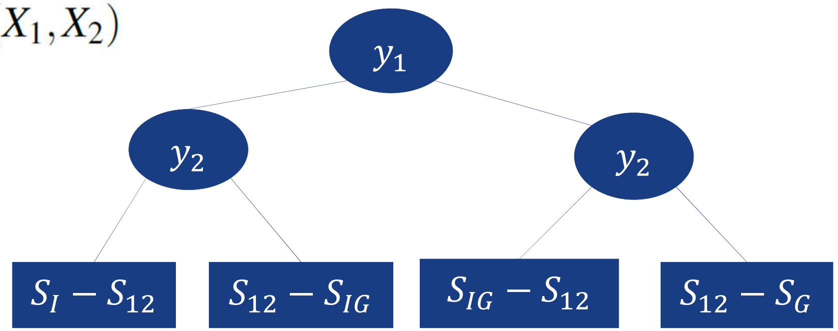
FLAT AND COMPACT TREE ENCODING

Flat Encoding

- Only one copy of transition relation
- $\log(k) + 2$ quantifier alternations

[Rintanen, 2003]
[Dershowitz etal, 2005]
[Jussila, Biere, 2007]

$$\begin{aligned}
&\exists S_I, S_G\\
&\exists S_{IG} \forall y_1 \exists S_1, S_2\\
&\exists S_{12} \forall y_2 \exists X_1, X_2\\
&(\neg y_1 \rightarrow (S_1 \leftrightarrow S_I \wedge S_2 \leftrightarrow S_{IG})) \wedge (y_1 \rightarrow (S_1 \leftrightarrow S_{IG} \wedge S_2 \leftrightarrow S_G)) \wedge\\
&(\neg y_2 \rightarrow (X_1 \leftrightarrow S_1 \wedge X_2 \leftrightarrow S_{12})) \wedge (y_2 \rightarrow (X_1 \leftrightarrow S_{12} \wedge X_2 \leftrightarrow S_2)) \wedge\\
&I(S_I) \wedge G(S_G) \wedge T(X_1, X_2)
\end{aligned}$$



Compact Tree Encoding

- Log copies of transition relation
- $\log(k) + 1$ quantifier alternations

[Cashmore et al, 2012]

$$\begin{aligned}
&\exists S_I, S_G\\
&\exists X_2 \forall y_2\\
&\exists X_1 \forall y_1 \exists X_0\\
&((\neg y_1 \wedge \neg y_2) \rightarrow T(S_I, X_0)) \wedge ((y_1 \wedge y_2) \rightarrow T(X_0, S_G)) \wedge\\
&(\neg y_1 \rightarrow T(X_0, X_1)) \wedge (y_1 \rightarrow T(X_1, X_0)) \wedge\\
&((\neg y_2 \wedge y_1) \rightarrow T(X_0, X_2)) \wedge ((y_2 \wedge \neg y_1) \rightarrow T(X_2, X_0)) \wedge\\
&I(S_I) \wedge G(S_G)
\end{aligned}$$

PLAN

1. Motivating example: Quantum Circuit Layout Synthesis
2. Classical Planning / Bounded Model Checking (SAT)
3. **Concise Encoding of Planning in QBF**
   - Path Compression
   - **Lifted Planning (almost first-order)**
4. Concise Encoding of 2-player board games in QBF
   - Tic-tac-toe, Hex, Breakthrough, Domineering
5. Validation of encodings with QBF certificates

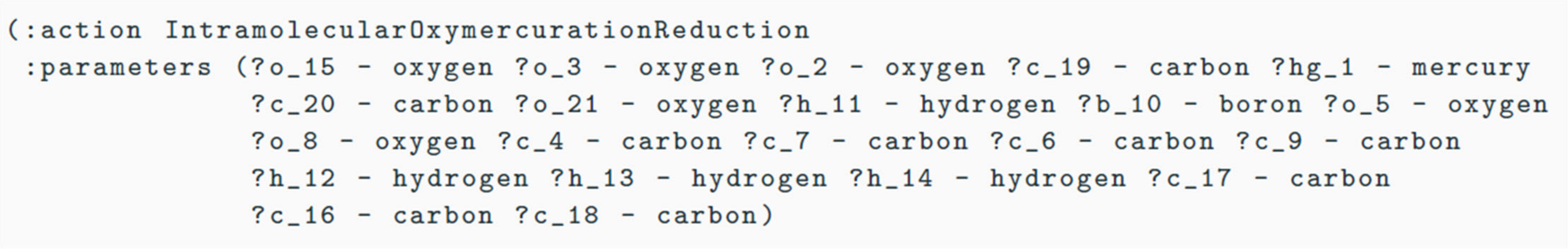
LIFTED CLASSICAL PLANNING AS QBF (1)

- Irfansha Shaik and Jaco van de Pol, *Classical planning as QBF without grounding*. In: ICAPS 2022 (International Conference on Automated Planning and Scheduling)
- **Grounding**: Instantiate actions for all **object combinations**

  apply_cnot(l1, l2, p1, p2, g1, g2, g3)

  $\mathbf{4 \times 4 \times 5 \times 5 \times 10 \times 10 \times 10 \;= 400,000}$ **potential instances!**

- **Main motivating example: Organic Synthesis (MIT exams organic chemistry)** [Masoumi et al, 2015], outstanding domain submission award IPC-2018



```
(:action IntramolecularOxymercurationReduction
 :parameters (?o_15 - oxygen ?o_3 - oxygen ?o_2 - oxygen ?c_19 - carbon ?hg_1 - mercury
             ?c_20 - carbon ?o_21 - oxygen ?h_11 - hydrogen ?b_10 - boron ?o_5 - oxygen
             ?o_8 - oxygen ?c_4 - carbon ?c_7 - carbon ?c_6 - carbon ?c_9 - carbon
             ?h_12 - hydrogen ?h_13 - hydrogen ?h_14 - hydrogen ?c_17 - carbon
             ?c_16 - carbon ?c_18 - carbon)
```

- **Note: in this case, the direct SAT encoding cannot even be generated!**

LIFTED CLASSICAL PLANNING AS QBF (2)

- We use Universal Quantifiers to enumerate all object combinations symbolically
- Result: QBF encoding remains small, even for organic synthesis
- QBF solver CAQE can handle this encoding reasonably well
  - Beats SAT-based planning by a large margin (Madagascar)
  - Competitive with the winning planning tools from IPC 2018 (FDSS, PowerLifted)
- We are only competitive if “grounding” is the bottleneck

$\exists \mathrm{A}^0, \mathrm{PM}^0, \ldots, \mathrm{A}^{k-1}, \mathrm{PM}^{k-1}$ …. *There exist Actions and Parameters*

$\forall \mathrm{OC}$ …. *Such that for all object combinations*

$\exists \mathrm{P}^0, \ldots, \mathrm{P}^k$ …. *There exists a plan of k steps*

$$\mathrm{I_u}(\mathrm{P}^0, \mathrm{OC}) \wedge \mathrm{G_u}(\mathrm{P}^k, \mathrm{OC}) \wedge \bigwedge_{i=0}^{k-1} \mathrm{T}_\mathrm{u}^i(\mathrm{P}^i, \mathrm{P}^{i+1}, \mathrm{OC}, \mathrm{A}^i, \mathrm{PM}^i)$$

PLAN

1. Motivating example: Quantum Circuit Layout Synthesis

2. Classical Planning / Bounded Model Checking (SAT)

3. Concise Encoding of Planning in QBF
    - Path Compression
    - Lifted Planning (almost first-order)

4. **Concise Encoding of 2-player board games in QBF**
    - Tic-tac-toe, Hex, Breakthrough, Domineering

5. Validation of encodings with QBF certificates

ENCODING 2-PLAYER GAMES IN QBF

We used QBF to encode planning problems

QBF allows arbitrary quantifier alternation

Then why not encode 2-player games?

Player 1 can win the game in 3 moves:

$$\boldsymbol{\exists m_1 \forall m_2 \exists m_3 \forall m_4 \exists m_5} : \exists S_0, S_1, S_2, S_3, S_4, S_5 : \\ start(S_0) \land Move(S_0, m_1, S_1) \land \cdots \land Move(S_4, m_5, S_5) \land Won(S_5)$$

Needed:

- Domain-specific language to specify grid-based board games: BDDL
- Concise encoding from BDDL into QBF “there exists a winning strategy in d moves”

BDDL – BOARD-GAME DESCRIPTION

Irfansha Shaik, Jaco van de Pol, *Concise QBF Encodings for Games on a Grid.*
In: arXiv :2303.16949, 2023

Domain-specific modeling language for Grid-based Board Games

- Assume an $m \times n$ board of positions, and 2 players (Black, White)
- Describe **Black and White moves** for symbolic position (x, y)
  - Can refer to neighbours, like "*open*$(x, y+1)$" or "*white*$(x-2, y+1)$"
  - Implicitly observe the board boundaries
- Describe **Black and White winning conditions**
  - Patterns starting at symbolic position (x, y)
  - Example: black(x, y), black$(x, y+1)$, black$(x, y+2)$

PLAN

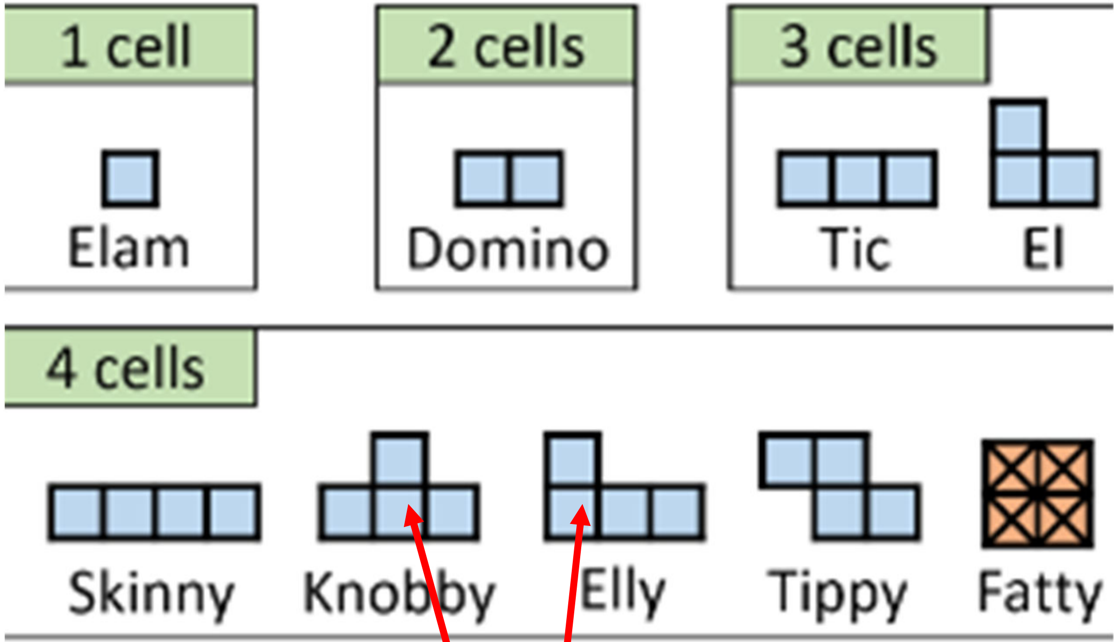
1. Motivating example: Quantum Circuit Layout Synthesis
2. Classical Planning / Bounded Model Checking (SAT)
3. Concise Encoding of Planning in QBF
   - Path Compression
   - Lifted Planning (almost first-order)
4. Concise Encoding of 2-player board games in QBF
   - **Tic-tac-toe, Connect4, Breakthrough, Domineering, Hex**
5. Validation of encodings with QBF certificates

TIC-TAC-TOE (POSITIONAL GAME)

```
#blackactions
:action occupy
:parameters (?x,?y)
:precondition (open(?x,?y))
:effect (black(?x,?y))

#whiteactions
:action occupy
:parameters (?x,?y)
:precondition (open(?x,?y))
:effect (white(?x,?y))
```

GENERALIZED TTT (POSITIONAL GAME)

```
#blackgoals % Elly
( black(?x,?y) black(?x,?y+1)
  black(?x+1,?y) black(?x+2,?y))
( black(?x,?y) black(?x,?y+1)
  black(?x+1,?y) black(?x+1,?y))
...

#whitegoals % Knobby
( white(?x,?y) white(?x,?y+1)
  white(?x-1,?y) white(?x+1,?y)
...
```



[Diptamara et al, QBF @ SAT 2016]

(x,y)

CONNECT4 (NON-POSITIONAL GAME)

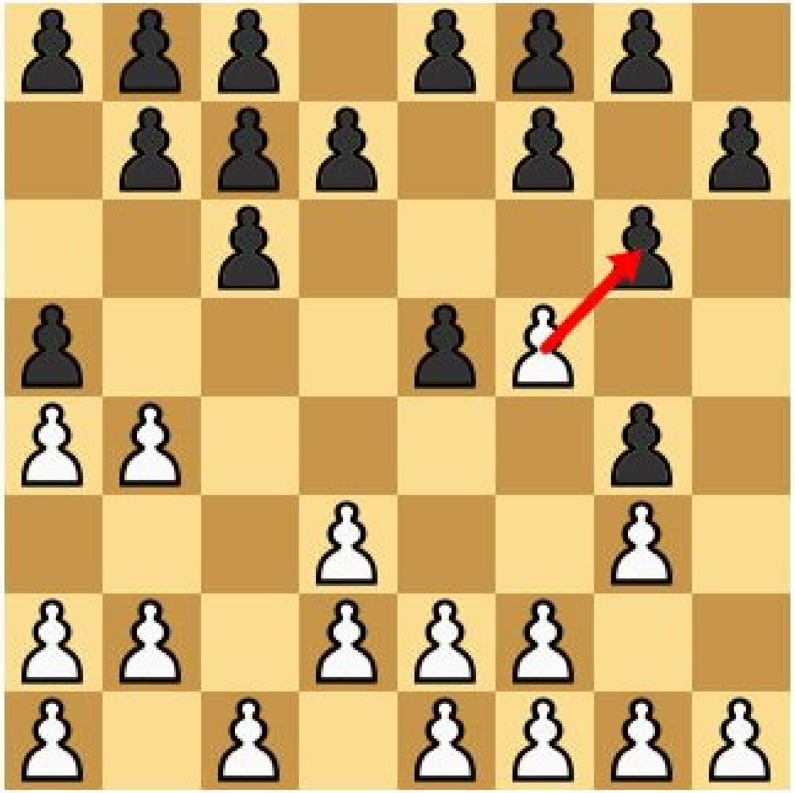
Move on (x, y) depends on other positions

```
#blackactions
:action occupy_on_top
:parameters (?x,?y)
:precondition
   (open(?x,?y) (not(open(?x,?y-1)))
:effect (black(?x,?y))

:action occupy_on_bottom
:parameters (?x,?y)
:precondition (open(?x, ymin))
:effect (black(?x, ymin))
```

BREAK-THROUGH (CAN TAKE PIECES)

"Chess with pawns"

```
#whiteactions
:action take_right
:parameters (?x,?y)
:precondition
  (white(?x,?y) black(?x+1,?y+1))
:effect
  (open(?x,?y) white(?x+1,?y+1))

#whitegoals
  (white(?x, ymax))
```

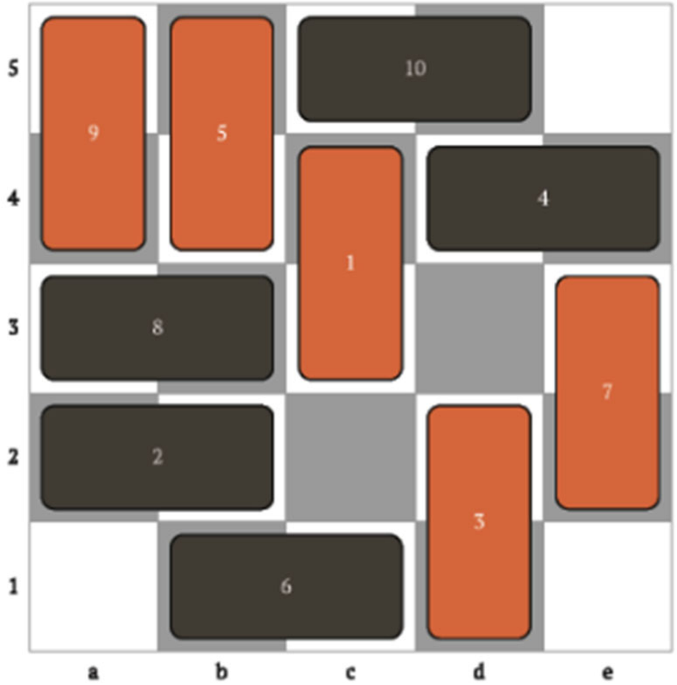


[Stephenson et al., preprint

DOMINEERING (MULTIPLE POSITIONS)

```
#blackactions
:action domino-horizontal
:parameters (?x,?y)
:precondition (open(?x,?y) open(?x+1,?y))
:effect (black(?x,?y) black(?x+1,?y))

#blackgoals
  False
```

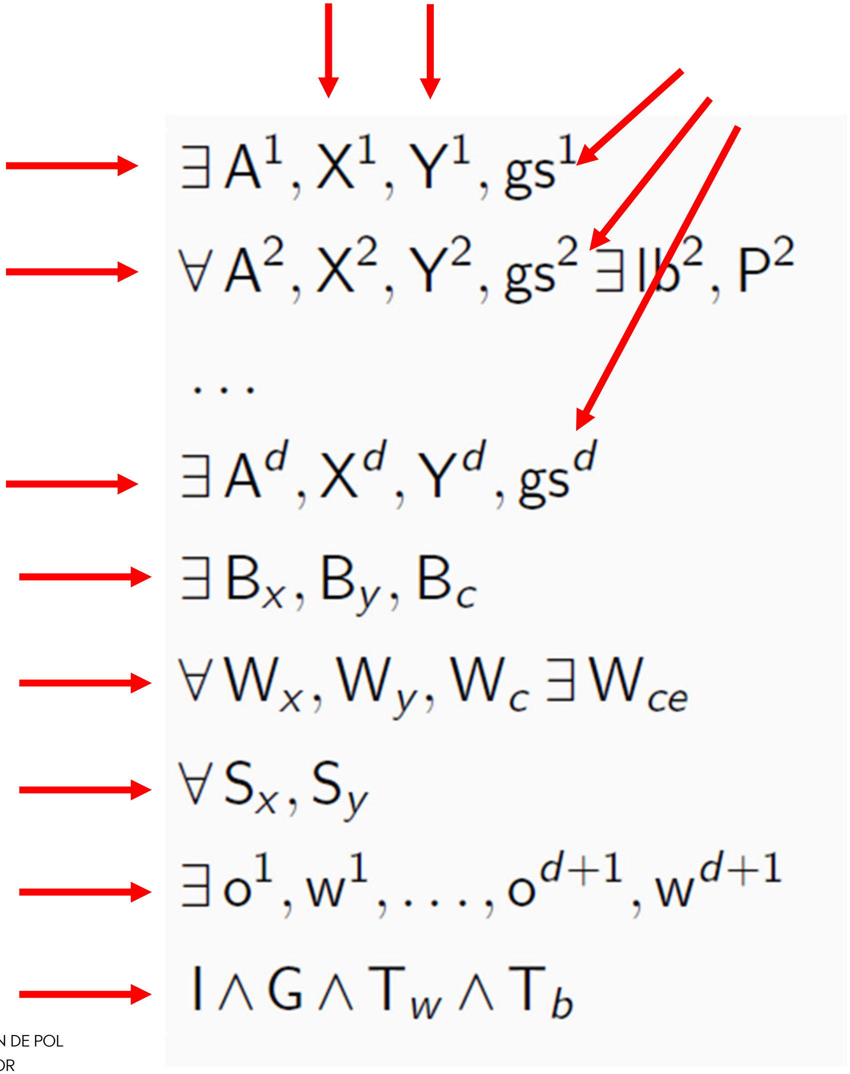
The first player that has no move loses the game



[J. Davies et al., Institute of Mathematics]

LIFTED QBF TRANSLATION: WIN IN $\leq d$ STEPS

- We use quantifier alternations to alternate between moves of the players.
- We encode positions using bitvectors. We introduce “Boolean circuitry” to handle “$?x \pm c$” and “$y_{min} \leq ?y \leq y_{max}$”
- We use “game-stop” variables to check (non)-winning of intermediate positions
- We use bitvectors for Black Winning and White Non-Winning conditions
- We use symbolic position variables to check for all positions symbolically
- We check that the resulting play is a valid winning play, for pos (S_x, S_y)



$$\exists \mathrm{A}^1, \mathrm{X}^1, \mathrm{Y}^1, \mathrm{gs}^1$$
$$\forall \mathrm{A}^2, \mathrm{X}^2, \mathrm{Y}^2, \mathrm{gs}^2 \, \exists \mathrm{lb}^2, \mathrm{P}^2$$
$$\ldots$$
$$\exists \mathrm{A}^d, \mathrm{X}^d, \mathrm{Y}^d, \mathrm{gs}^d$$
$$\exists \mathrm{B}_x, \mathrm{B}_y, \mathrm{B}_c$$
$$\forall \mathrm{W}_x, \mathrm{W}_y, \mathrm{W}_c \, \exists \mathrm{W}_{ce}$$
$$\forall \mathrm{S}_x, \mathrm{S}_y$$
$$\exists \mathrm{o}^1, \mathrm{w}^1, \ldots, \mathrm{o}^{d+1}, \mathrm{w}^{d+1}$$
$$\mathrm{I} \wedge \mathrm{G} \wedge \mathrm{T}_w \wedge \mathrm{T}_b$$

linear in input size of game and d,
$d + 3$ quantifier alternations

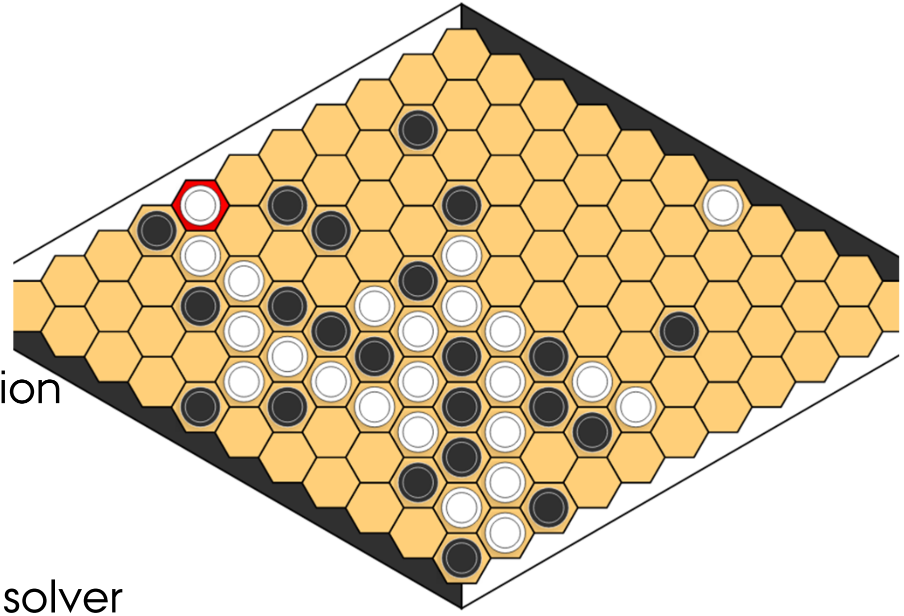
HEX (PIET HEIN)

Irfansha Shaik, V. Mayer-Eichberger, J vd Pol, A. Saffidine,

Implicit State and Goals in QBF Encodings for Positional Games.
In: arXiv 2301.07345, 2023

Hex: Back to positional games

- *But exponentially many winning patterns.*



Solution:

- Lifted encoding of winning path, using $N(x,y)$-relation
- Several encodings: (non)-lifted, (non)-CNF, …

Results:

- Can solve Hein's puzzles on small boards with QBF-solver
- Can encode and solve human-played end-games on large boards

PLAN

1. Motivating example: Quantum Circuit Layout Synthesis

2. Classical Planning / Bounded Model Checking (SAT)

3. Concise Encoding of Planning in QBF
    - Path Compression
    - Lifted Planning (almost first-order)

4. Concise Encoding of 2-player board games in QBF
    - Tic-tac-toe, Connect4, Breakthrough, Domineering, Hex

5. Validation of encodings with QBF certificates

VALIDATION OF ENCODINGS IN QBF

Irfansha Shaik, Maximilian Heisinger, Martina Seidl, and Jaco van de Pol,
Validation of QBF Encodings with Winning Strategies. In: SAT 2023

Encoding in QBF is difficult: low-level, error-prone, resulting strategy is hard to test

Can we use certificates, which are already produced by QBF solvers?

- Validation of QBF solvers
- Validation of QBF encoding

Ideas:

- Can use the certificate to extract winning strategy ➔ interactive play
- Also: basis for automatic testing of invariants, equivalences, etc.

Scalability:

- **Full certificate:** checking is efficient, but certificates are enormous
- **Only witness of first layer:** certificates are small, but need many QBF queries
- **Partial certificates:** certificates are reasonable, checking is efficient

CONCLUSION

- We can solve interesting problems as classical planning problems
  - Optimal Quantum Layout Synthesis
  - Organic Synthesis
- QBF and QBF solvers can be used, based on concise encodings
  - Path compression
  - Lifted encodings
- The QBF encoding can be extended to 2-player games
  - Board games, symbolic positions
  - Hex, symbolic winning conditions
- Validation of encodings
  - Winning strategies can be validated, based on QBF certificates
  - Interactive play, invariant testing, equivalence testing

AARHUS
UNIVERSITY