Quantum computational speed-ups with smaller quantum computers

Vedran Dunjko
v.dunjko@liacs.leidenuniv.nl

based on: J. Math Phys ***61****, 012201 (2020);, Phys. Rev. Lett. 121, 250501 (2018) + works in progress*

with Y. Ge, I. Cirac, M. Rennela, A. Laarman; H. Calandra, C. Moussa

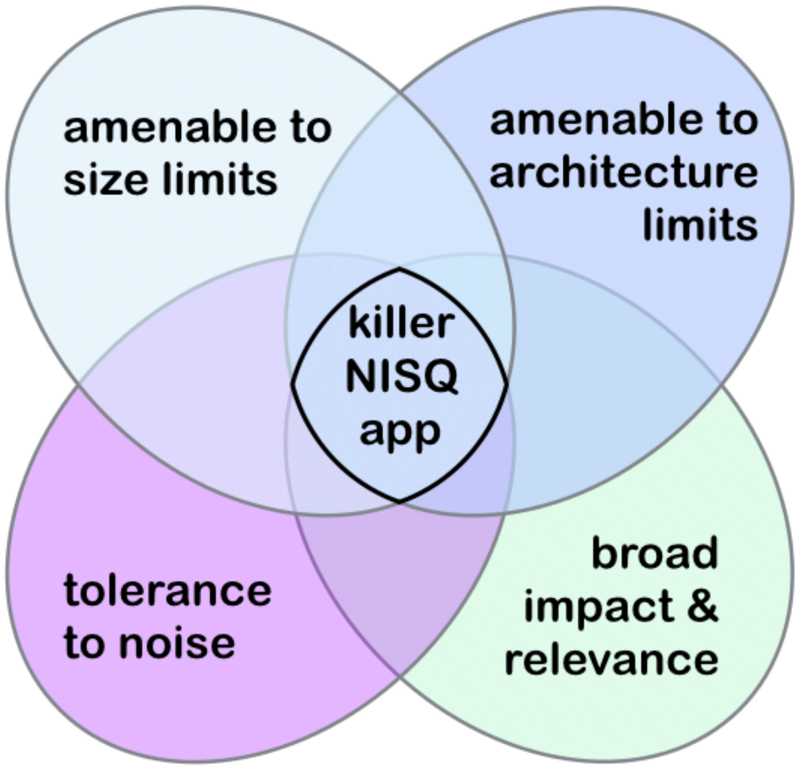
Quantum Software Consortium
Universiteit Leiden
liacs Leiden Institute of Advanced Computer Science

Basic motivational question:

suppose you have a problem instance (say SAT) of ***size*** $\boldsymbol{n}$***,***
and quantum computer handling $\boldsymbol{k} \ll \boldsymbol{n}$ ***qubits.***

Is the QC any good for you?

Motivation



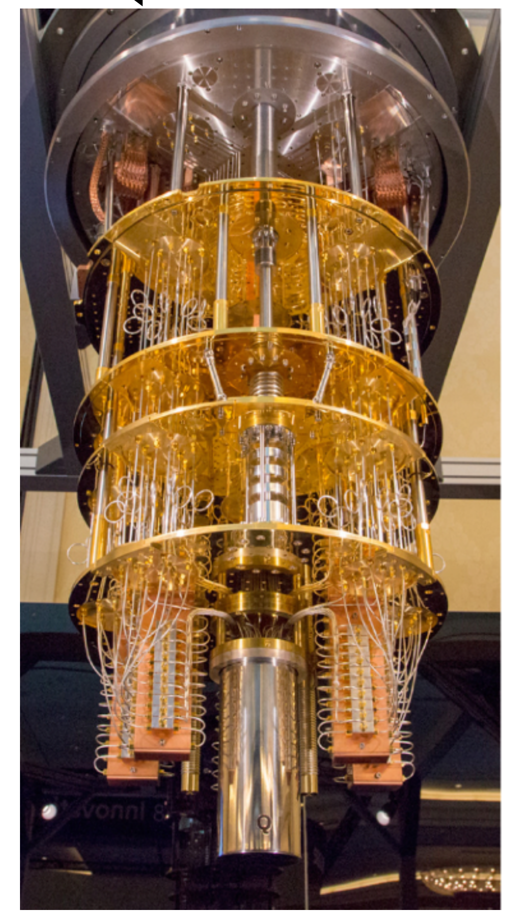
Here:

1) dealing with just size issues - no errors

2) speed-ups/enhancements
not supremacy, common hard probs.

3) basic curiosity - how much can a
smaller device help bigger problems

50 qubit QC… in 3 years 100-1000 qubit QC



Banana for scale

Also one qubit is *much* more
expensive than a banana…

Related work and settings

Question: *suppose you have a (3SAT) problem of size n, and quantum computer handling $k<<n$ qubits.*

What can you do?

(A) ***Hack it.*** Identify all smaller subroutines, speed those up.

A bit more systematic:

(B) ***"Quantum circuit chop"***
Find a suitable quantum algorithm for problem; chop up the circuit.
(Bravyi, Smith, Smolin '16; Peng, Harrow, Ozols, Wu '19)

(C) ***"Classical algorithm chop"***
Find a good classical algorithm which chops well, and can be quantum-enhanced

Related work and settings

Question: *suppose you have a (3SAT) problem of size n, and quantum computer handling $k<<n$ qubits.*

What can you do?

(A) ***Hack it.*** Identify all smaller subroutines, speed those up.

A bit more systematic:

(B) ***"Quantum circuit chop"***
Find a suitable quantum algorithm for problem; chop up the circuit.
(Bravyi, Smith, Smolin '16; Peng, Harrow, Ozols, Wu '19)

(C) ***"Classical algorithm chop"***
Find a good classical algorithm which chops well, and can be quantum-enhanced

Trade-offs in applicability, and which approach gives better results

Our setting - more detail

- have *some* classical algorithm for a problem in mind :
  Schöning, PPSZ, DPLL for boolean satisfiability (SAT),
  Eppstein for Hamilton cycles

- there is a faster quantum algorithm (few options)

- have n-sized instance of a problem ("solution space size")

- but only a $k<n$-sized QC

When does the QC "*genuinely* help"?
def. here: polynomial speed-up, asymptotically

$$O^*\left(2^{\gamma_c n}\right) \to O^*\left(2^{\gamma_q n}\right)$$

$$\gamma_c \leq \gamma_q$$

Our setting - more detail

- have *some* classical algorithm for a problem in mind :
  Schöning, PPSZ, DPLL for boolean satisfiability (SAT),
  Eppstein for Hamilton cycles

- there is a faster quantum algorithm (few options)

- have n-sized instance of a problem (“solution space size”)

- but only a $k<n$-sized QC

When does the QC “*genuinely* help”?
def. here: polynomial speed-up, asymptotically

k cannot be constant

most interesting case:
$k = \alpha n; \alpha \in (0,1)$

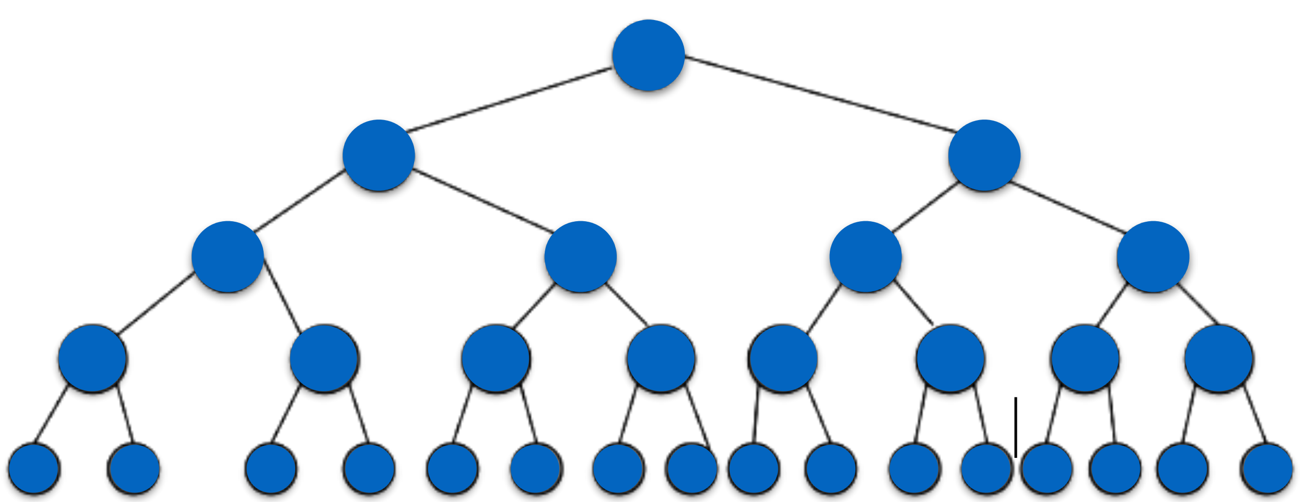
The method: hybrid divide-and-conquer

"Naturally" chop-uppable algorithms: divide-conquer, recursive, backtracking

$$f : \{0,1\}^n \to \{0,1\}$$

$$f(x_1, \ldots, x_n) = (x_1 \vee x_{10} \vee \bar{x}_{51}) \wedge (\bar{x}_3 \vee \bar{x}_{10} \vee \bar{x}_{11}) \wedge (\bar{x}_{11} \vee \bar{x}_{44} \vee \bar{x}_{51}) \cdots$$

E.g. backtracking algorithms exploring trees of possible (partial) solutions



```
procedure ALG(P)
    if TRIVIAL(P) = 1
        return f(P)
    else
        return g(ALG(R_1(P)), ..., ALG(R_l(P)))
end procedure
```

But nodes could be more general "subproblems"

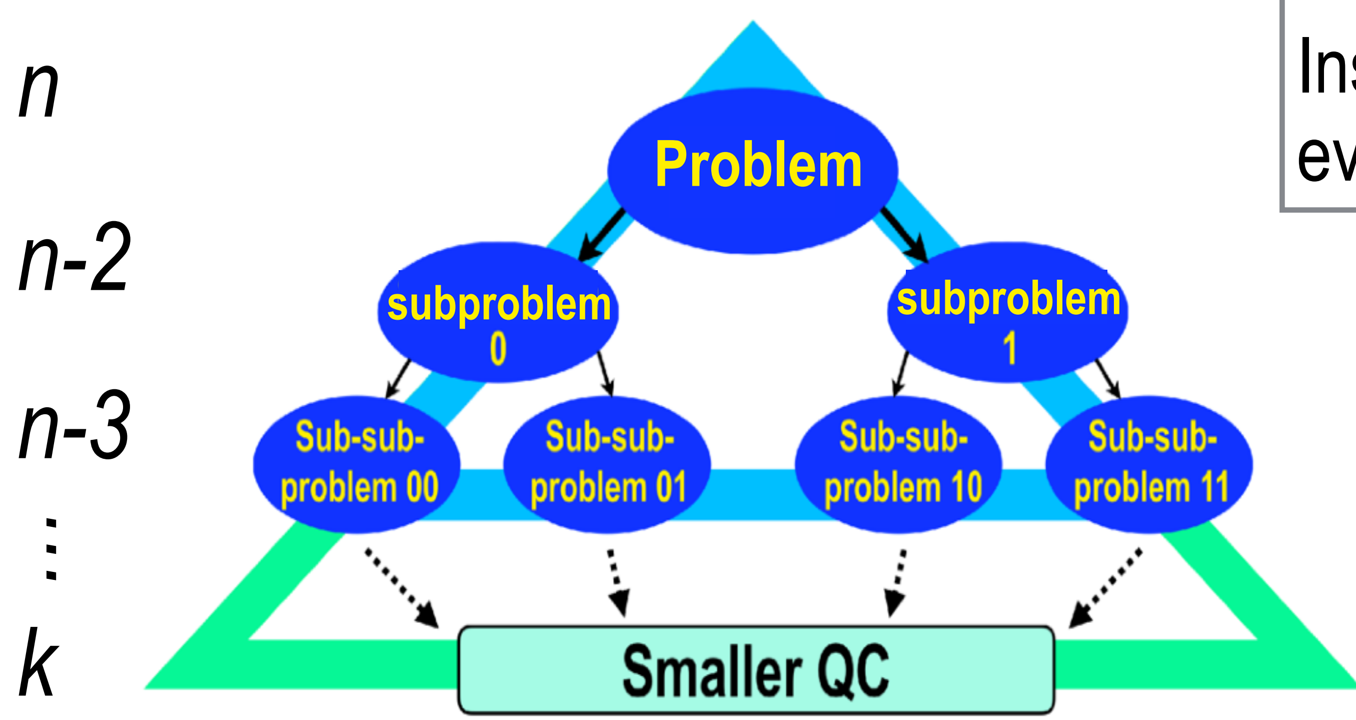
The method: hybrid divide-and-conquer

"Naturally" chop-uppable algorithms: divide-conquer, recursive, backtracking

Obvious hybrid method:

1) pick a suitable classical and quantum algorithm

2) do:

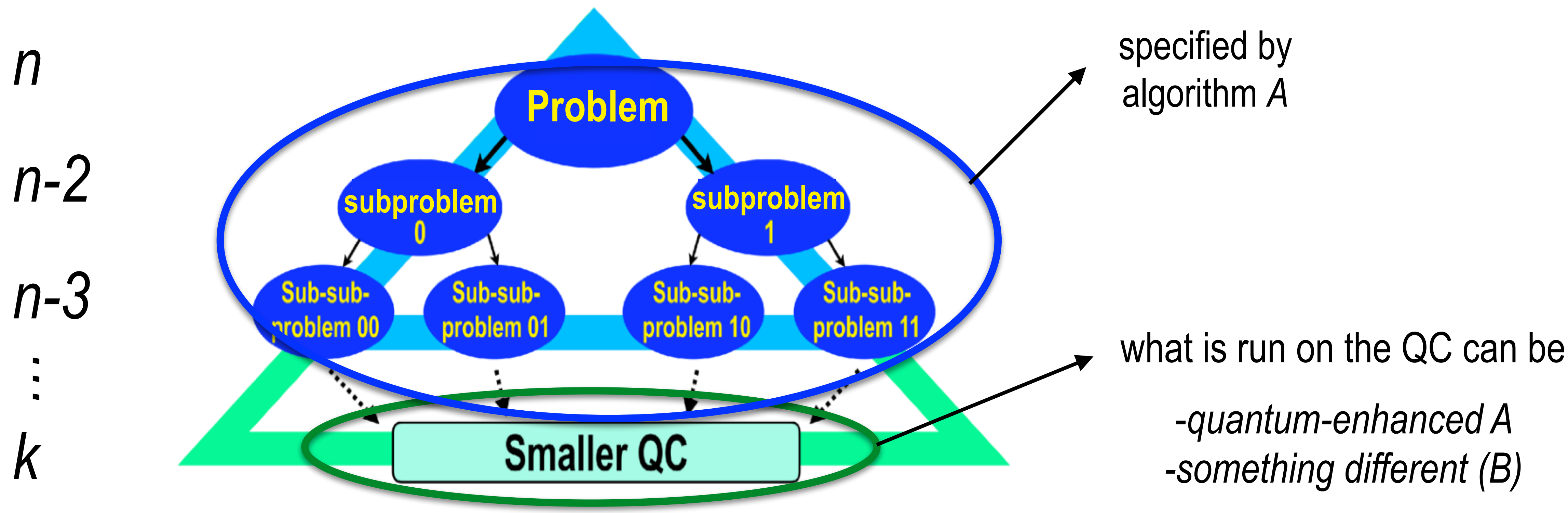


Intuition

Instance naturally shrinks, eventually fits on QC.

The method: hybrid divide-and-conquer

The process need not be fully homogeneous



Goal: express complexity as function of k, n and complexities of A and B

A few catches: we can fail get asymptotic speed-ups for a few reasons.

What can "B" be?
Here focus on quantum algorithms for NP-hard problems

- NP probably not in BQP (1996 / 1993?)

<u>Worst case exact</u>
- "Goverization" (Schöning), quantum walks (cubic Hamilton cycles), q. dynamic programming (graph problems)

<u>Approximation algorithms</u>
- QAOA

<u>Heuristics</u>
- **natural**: annealers & adiabatic QC
- enhancements: Grover, backtracking (tree exploration)
- via linear algebra approaches (in prep.)

- Enhancements v.s. "genuinely new" algorithms

What can “B” be?
Here focus on quantum algorithms for NP-hard problems

- NP probably not in BQP (1996 / 1993?)

Worst case exact
- “Goverization” (Schöning), quantum walks (cubic Hamilton cycles), q. dynamic programming (graph problems)

Approximation algorithms
- QAOA

Heuristics
- **natural**: annealers & adiabatic QC
- enhancements: Grover, backtracking (tree exploration)
- via linear algebra approaches (in prep.)

- Enhancements v.s. “genuinely new” algorithms

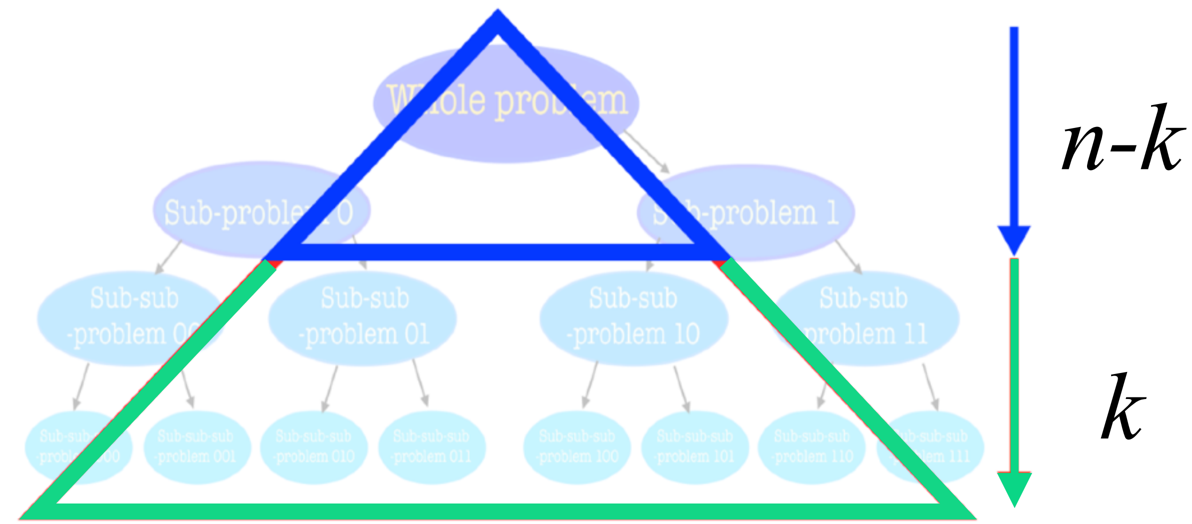
Mostly here algorithms *A* = *B,*
easier analysis,
also a *smart* way to do
heuristics

How it would work ideally: *e.g. tree search for exponential binary trees*

Setting

Algorithm A: Brute force search (BFS)
Algorithm B: Grover (Q. BFS)

Problem: SAT (=full, balanced binary tree)



Complexities

Classical: $2^n \quad (= 2^{\gamma_c n}, \gamma_c = 1)$

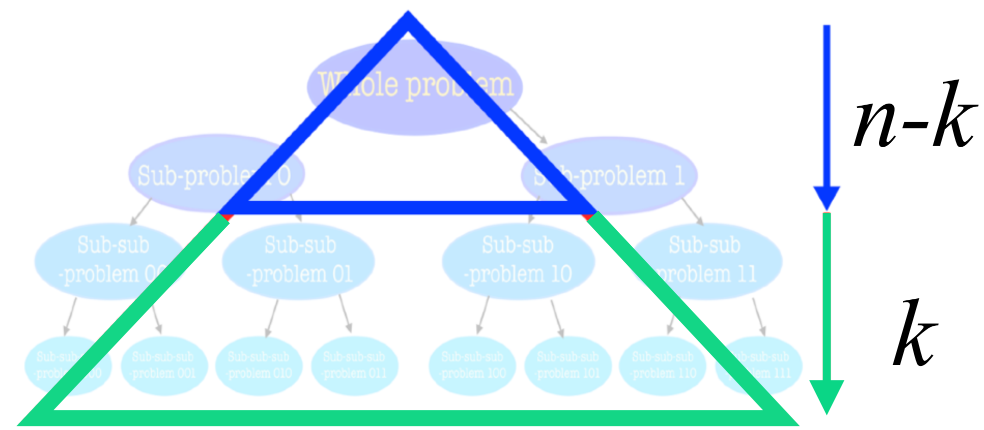
Quantum: $2^{n/2} \quad (= 2^{\gamma_q n}, \gamma_q = 1/2)$

Hybrid: $2^{(n-k)} \times 2^{k/2} = 2^{((n-k)/n + k/n)n} = 2^{((1-\alpha)\gamma_c + \alpha\gamma_q)n}$

Interpolates between ***classical and quantum run-times:*** $\alpha = k/n$

How it would work ideally

Can be done more generally e.g. for algos with run-times given in terms of standard recurrence relations



classical run-time $\sim \exp(\gamma_c n)$

quantum run-time $\sim \exp(\gamma_q n)$

Quantum backtracking algorithms can achieve $\gamma_q \approx \frac{\gamma_c}{2}$

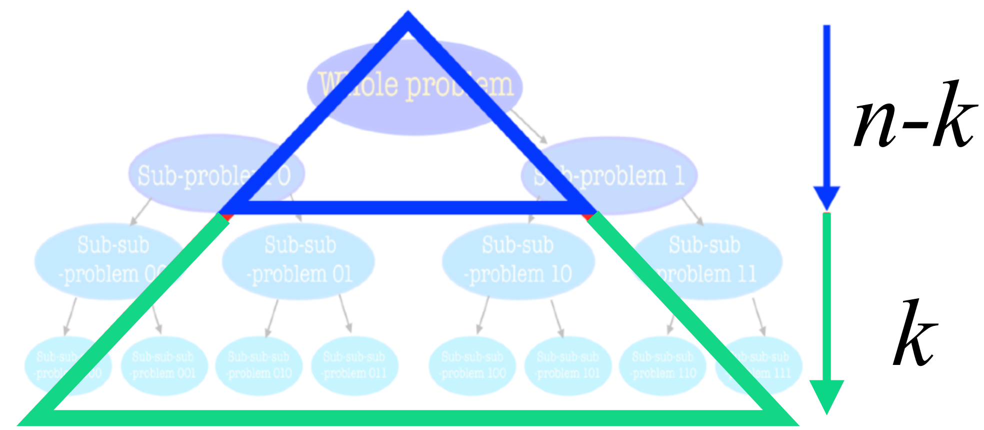
$$\text{hybrid run-time} \sim \exp\left[\left(\frac{n-k}{n}\gamma_c + \frac{k}{n}\gamma_q\right) n\right]$$

$$= exp\left[\left(\underbrace{(1-\alpha)\gamma_c + \alpha\gamma_q}_{\gamma_{hybrid}}\right) n\right]$$

$$\gamma_{hybrid} = (1-\alpha)\gamma_c + (\alpha)\gamma_q$$
$$\alpha = k/n \in O(1)$$

How it would work ideally

Can be done more generally e.g. for algos with run-times given in terms of standard recurrence relations



classical run-time $\sim \exp(\gamma_c n)$
quantum run-time $\sim \exp(\gamma_q n)$

For all constant $\alpha = k/n$,

$$\exp(\gamma_q n) < \exp(\gamma_{hybrid}\, n) < \exp(\gamma_c n)$$

threshold-free: speed-ups attained for all α!

In other words: no matter how small your QC is relative to the instance, there is a poly-speed-up attainable. *When is this possible?*

Freely control a trade-off between problem size(es) and speed-up attainable

How it fails

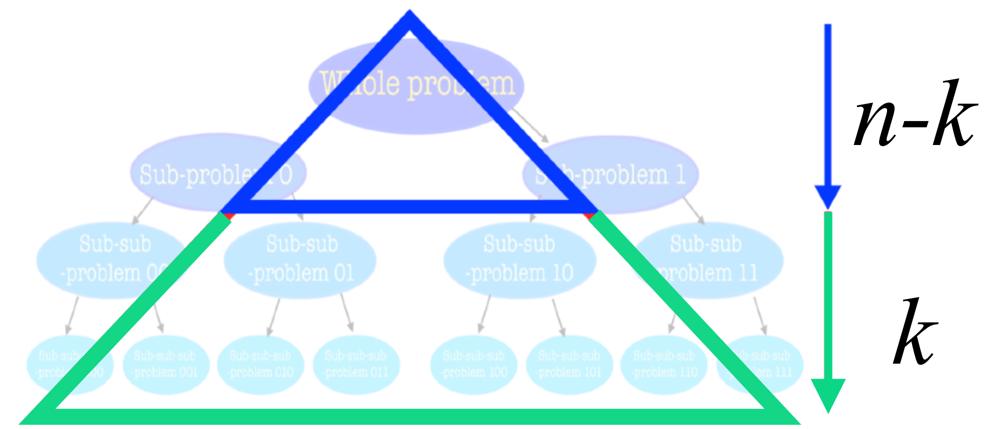
-“fat” algorithms (space complexity) ➡ no *real* speed-up (in limit)

-bad trees ➡ speed-up attained only for high α (not threshold-free)

Fat algorithm example

subtlety: what is “size” matters…

$$\text{hybrid run-time} \sim exp\left[\left((1-\alpha)\gamma_c + \alpha\gamma_q\right)n\right]$$



depth = #variables in subinstance

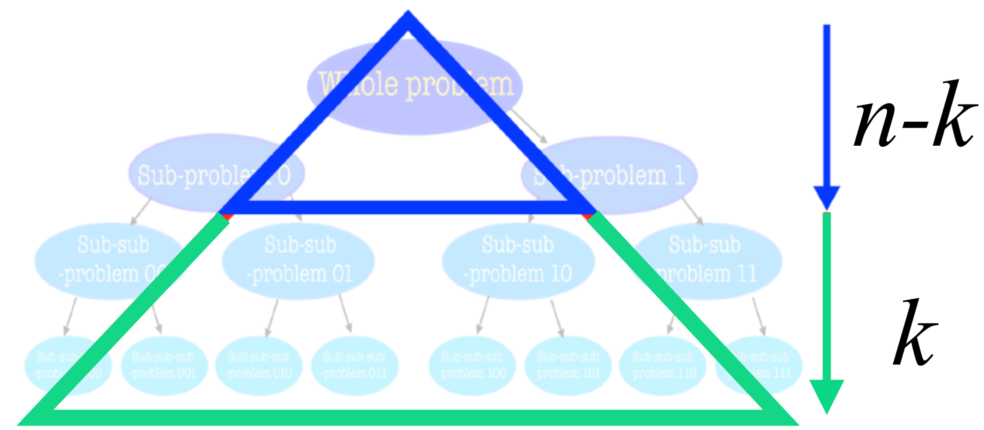
$\alpha \times n = k$ - in expression above, size of formula *we can handle*

but in general “size of formula I can handle” < number of qubits I have

Fat algorithm example

subtlety: what is “size” matters…

$$\text{hybrid run-time} \sim exp\left[\left((1-\alpha)\gamma_c + \alpha\gamma_q\right)n\right]$$



$\alpha \times n = k$ - in expression above, size of formula *we can handle*

Space-complexity* of q. algorithm $f(n)$ ➡ can handle $f^{-1}(\alpha \times n)$ sized-formula, not αn

E.g.: if $f(n) = n^2$

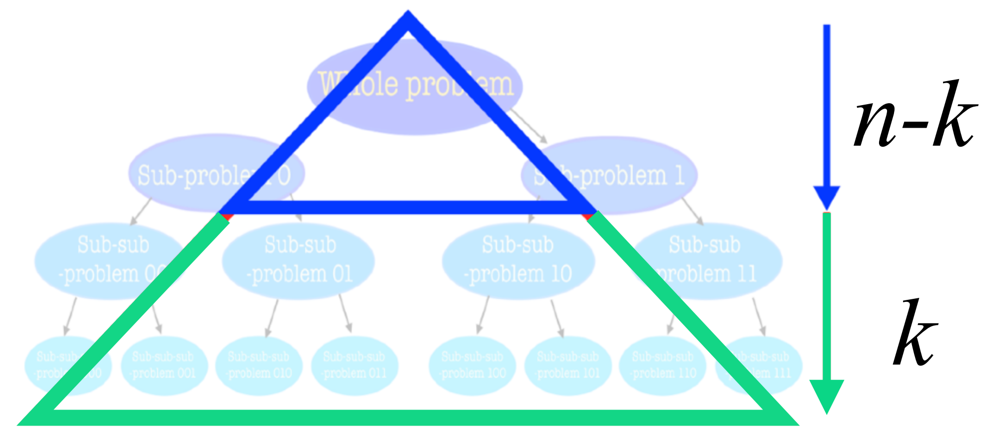
effective size handleable: $k = \sqrt{\alpha n}$

$$exp\left[\left(\left(\left(1-\sqrt{\frac{\alpha}{n}}\right)\gamma_c + \underbrace{\sqrt{\frac{\alpha}{n}}\gamma_q}_{\text{not poly speed-up}}\right)n\right]$$

Fat algorithm example

subtlety: what is “size” matters…

$$\text{hybrid run-time} \sim exp\left[\left((1-\alpha)\,\gamma_c + \alpha\gamma_q\right)n\right]$$



$\alpha \times n = k$ - in expression above, size of formula *we can handle*

Space-complexity* of q. algorithm $f(n)$ ➡ can handle $f^{-1}(\alpha \times n)$ sized-formula, not αn

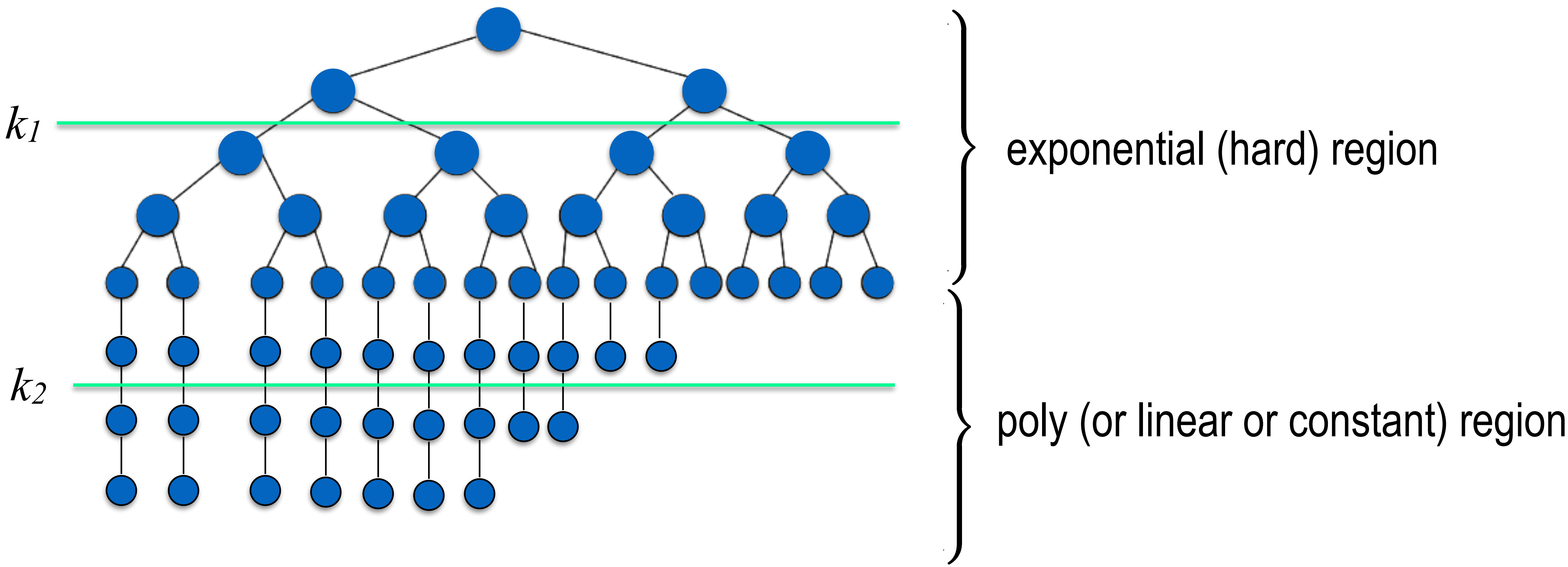
E.g.: if $f(n) = n^2$

effective size handleable: $k = \sqrt{\alpha n}$

$$exp\left[\left(\left(1-\sqrt{\frac{\alpha}{n}}\right)\gamma_c + \underbrace{\sqrt{\frac{\alpha}{n}}\gamma_q}_{\text{not poly speed-up}}\right)n\right]$$

Space complexity of Q.A. needs to be (essentially) linear…

Bad tree example



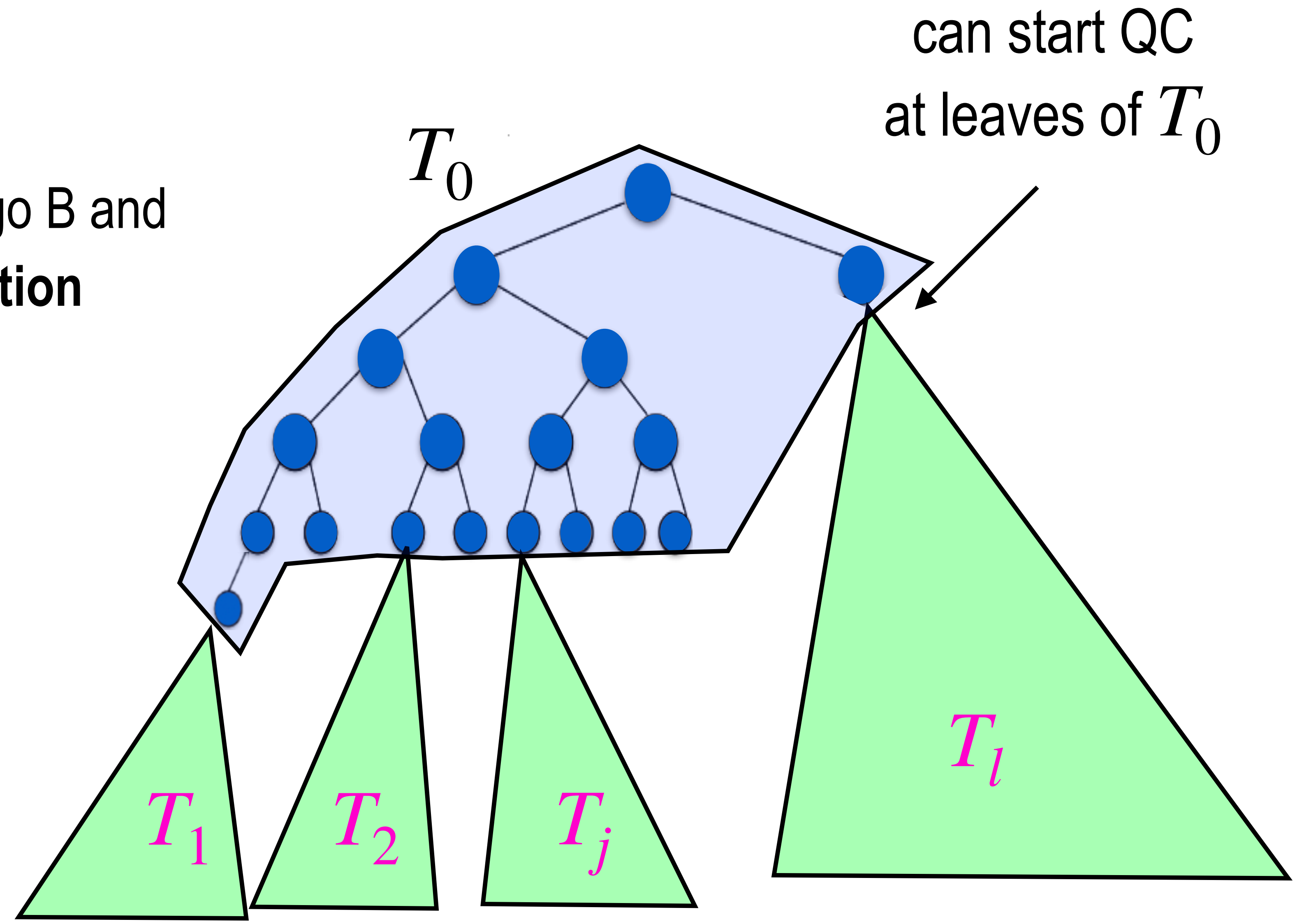
k_1 QC does some hard work — poly speed-up

k_1 QC speeds-up easy work — no speed-up (worse: quantum overheads & cost of reversibility…)

Speed-up is *not threshold-free*

So when does it work?

Algo A induces tree; and with quantum algo B and size limit (αn) a **search-tree decomposition**



Search tree decomposition:

$$T = T_0 + \sum_j T_j$$

So when does it work?

Main theorem

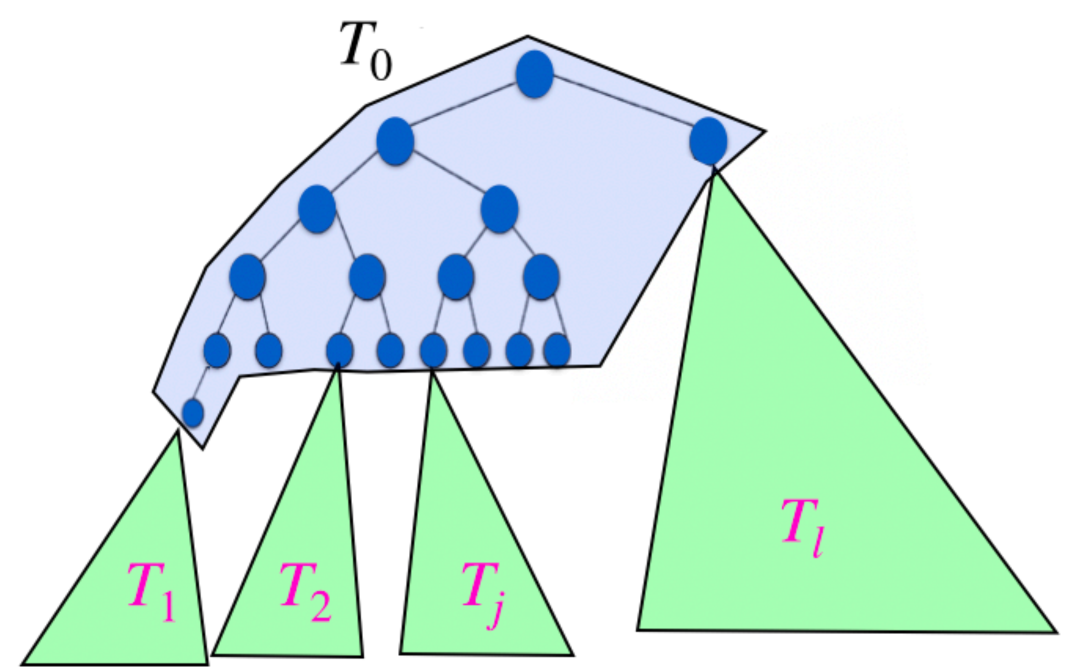
Given classical algo A, quantum algo B, a size limit (αn), consider induced search tree and its decomposition.

Assume T is exponential (can be relaxed)

The hybrid approach achieves poly speed-up if:

(1) $\mathbb{E}_j$ *Time*(A, T_j) *is poly slower than* $\mathbb{E}_j$ *Time*(B, T_j)

(2) Constant fraction of T_j *is exponentially sized*



Search tree decomposition:

$$T = T_0 + \sum_j T_j$$

So when does it work?

Main theorem

Given classical algo A, quantum algo B, a size limit (αn), consider induced search tree and its decomposition.

Assume T is exponential (can be relaxed) → *Don't have to care about overheads and fine-grained complexities*

The hybrid approach achieves poly speed-up if:

(1) $\mathbb{E}_j$ *Time*(A, T_j) *is poly slower than* $\mathbb{E}_j$ *Time*(B, T_j) → *QC is actually faster when used*

(2) Constant fraction of T_j *is exponentially sized* → *There is still sufficient work for QC to do (subtle)*

Intuition:

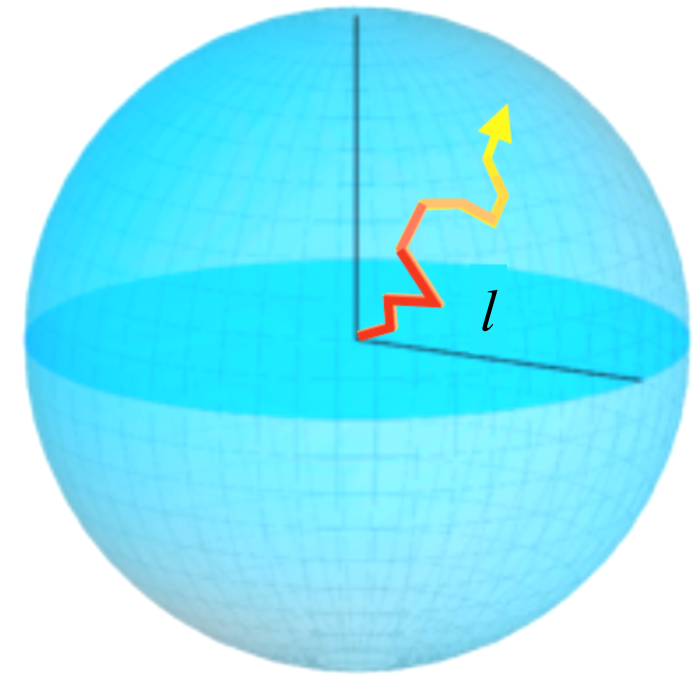
Both *(1)* and *(2)* can be contingent on α. If not, threshold-free speed-up

Now, when does it *actually work?*

Examples where quantum algos. could be made
sufficiently space efficient… without sacrificing speed… while reversible

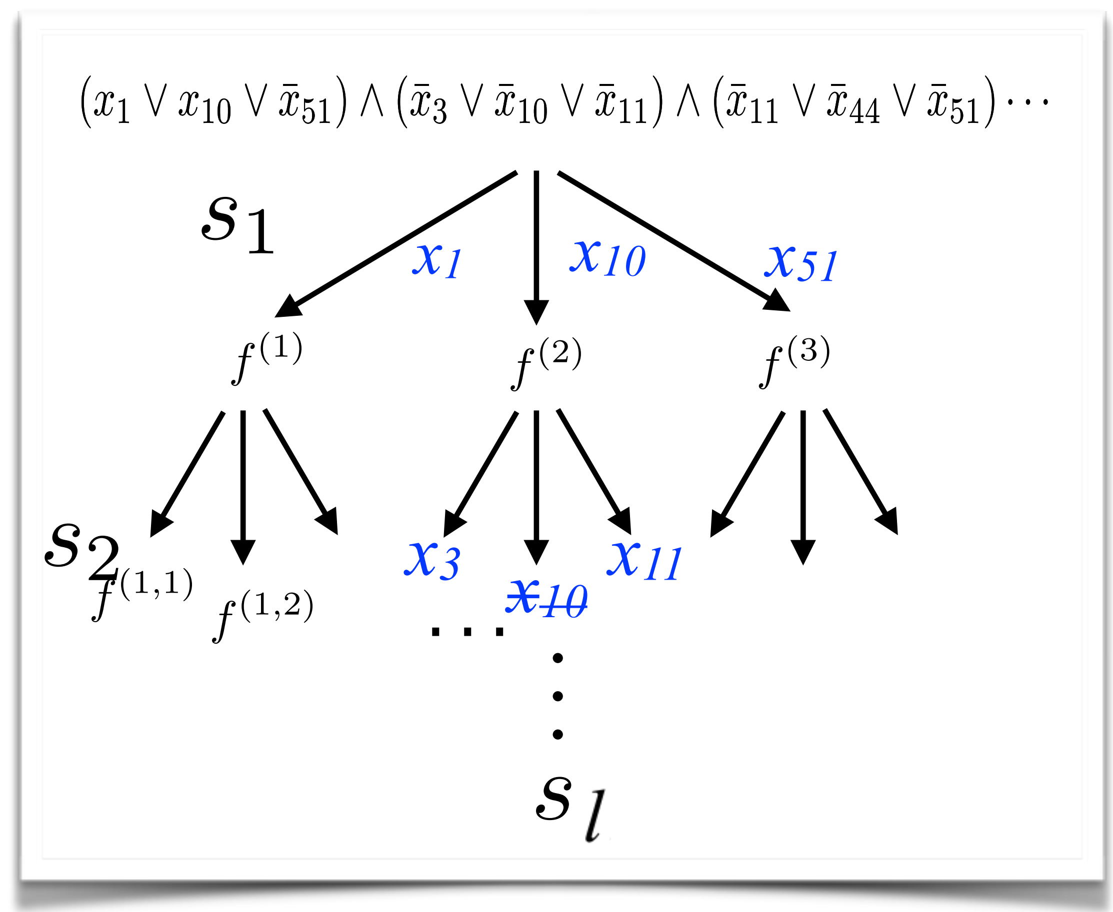
Example 1: Derandomized Schöning for 3-SAT & PromiseBallSAT

$\mathtt{PromiseBallSat}(\vec{x}, l)$



Given assignment $\vec{x}$ does there exist a satisfying assignment within hamming distance l?

Yields a ternary search tree



However! Naïve space complexity:

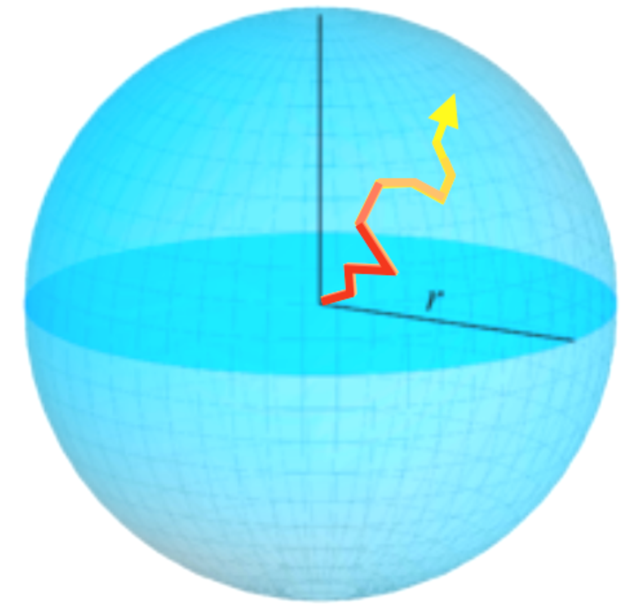
$$O(l \times log(n))$$

Too much…

E. Dantsin et al, Theoretical Computer Science 289, 69 (2002).
R. A. Moser and D. Scheder, in STOC 2011

Example 1: Derandomized Schöning for 3-SAT & PromiseBallSAT

Obstacles



Storing <u>sets *v.s. lists*</u>

list = $\Omega(k \times log(n))$

set = $O(k \times log(n/k)); \; n/k = 1/\alpha \in O(1)$

Problem: *set* update is *not* reversible (which element did I add last?)
direct reversibilization exponentially slow

Solution: special memory structure.
$k \times log(n/k)$ space complexity, and poly-time updates

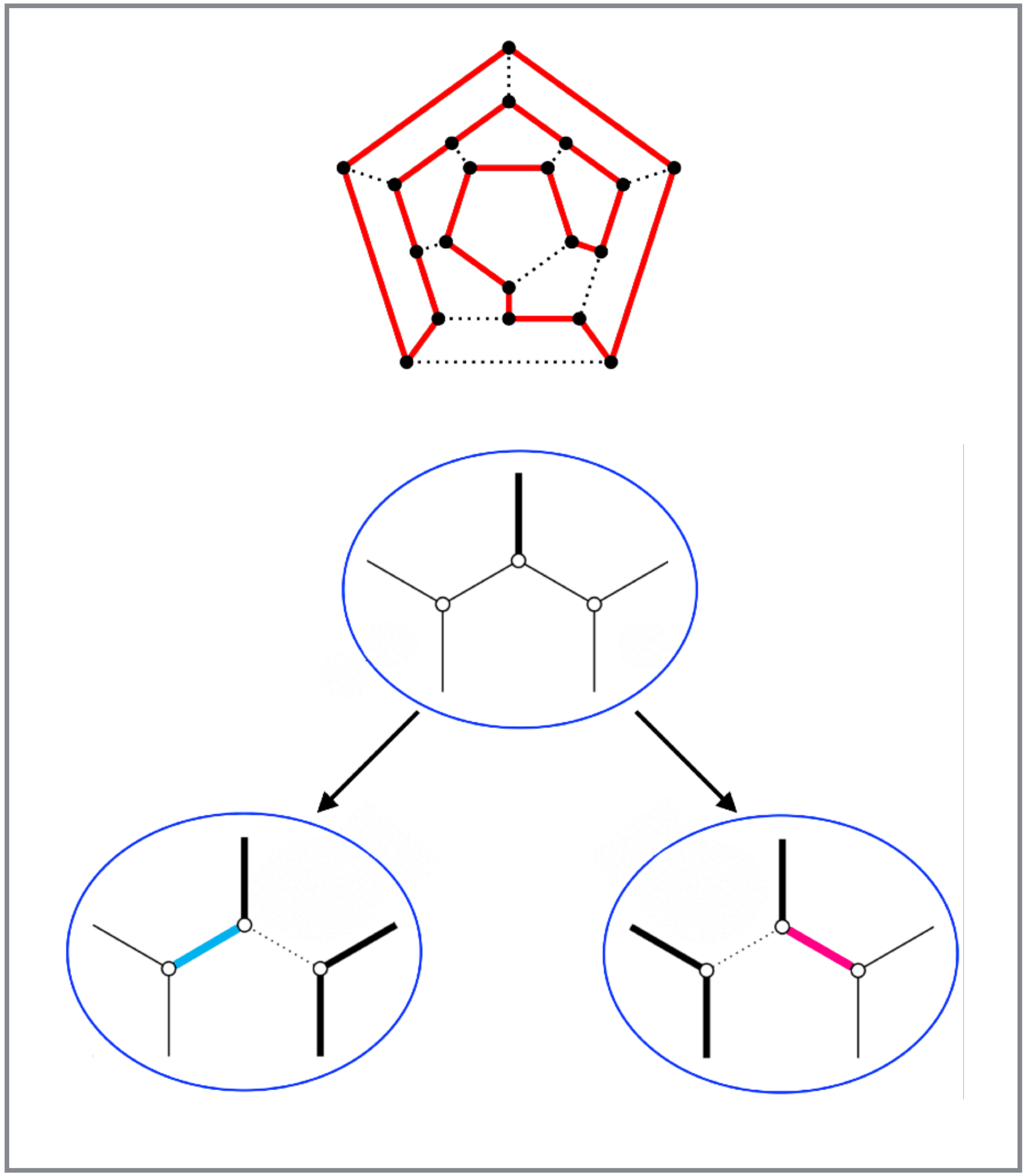
Phys. Rev. Lett. 121, 250501 (2018)

Example 2: Eppstein's algorithm for Hamilton cycles cubic graphs

Developed a number of space-efficient subroutines for dealing with sets

- application to Eppstein's algorithm for cubic graphs

- can "carry" sets of edges
- can identify terminating conditions
- can perform *simplification of graphs*

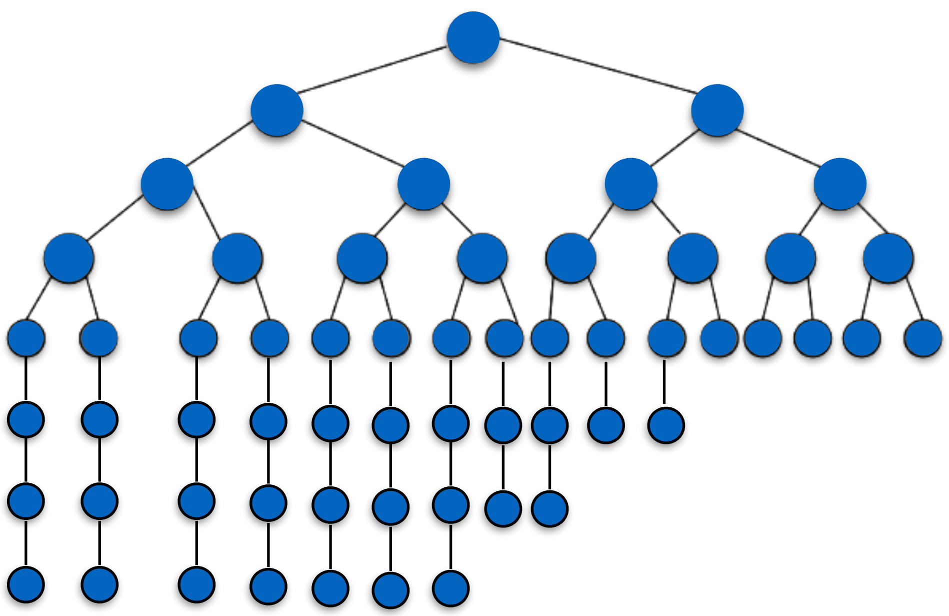


In these cases
- polynomial speed up relative to *best classical upper bound*
- the speed up is ***threshold free***
- ***full search trees (for bounds):*** based on Grover

J. Math Phys ***61****, 012201 (2020)*

Backtracking cases for boolean satisfiability

$$f(x_1,\ldots,x_n) = (x_1 \vee x_{10} \vee \bar{x}_{51}) \wedge (\bar{x}_3 \vee \bar{x}_{10} \vee \bar{x}_{11}) \wedge (\bar{x}_{11} \vee \bar{x}_{44} \vee \bar{x}_{51}) \cdots$$



- node = partial assignment = subformula
- tree depends on *ordering,* and *simplification method*
- can we *infer* the value of a given variable?
- E.g. unit resolution & pure literal rule (DPLL* algorithm), s-implication (PPSZ algorithm)
- Grover: no guaranteed speed-up; ***Quantum backtracking*****:** speed-up in queries

**DPLL:* Davis–Putnam–Logemann–Loveland

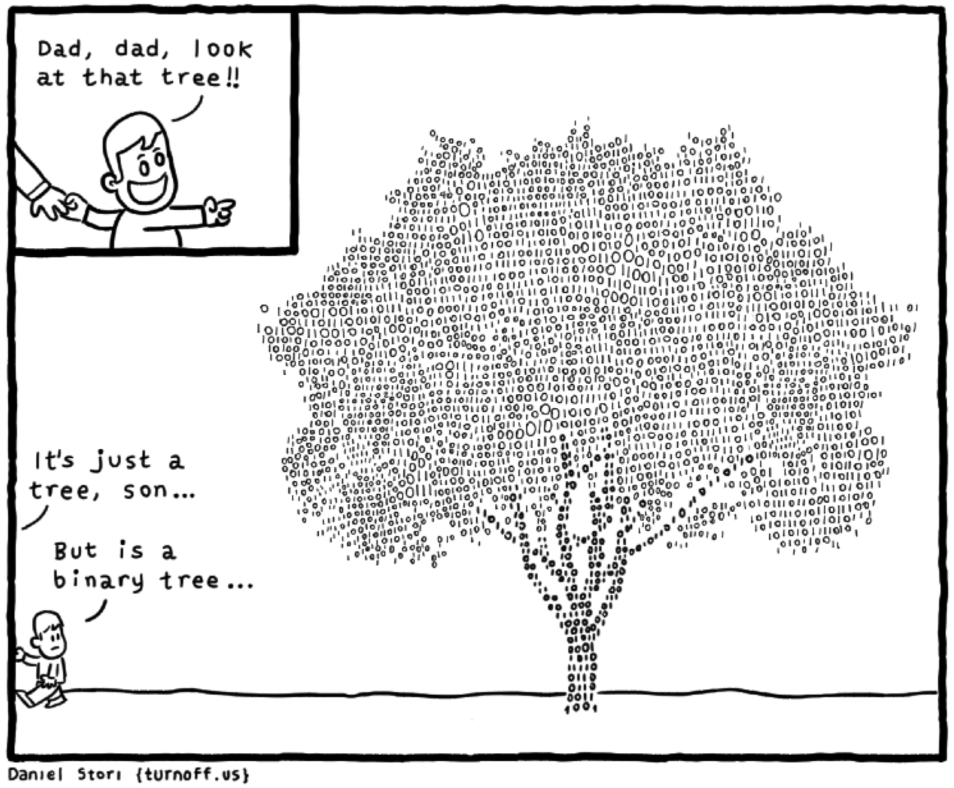
Montanaro ToC '18
Ambainis, Kokainis, STOC '17

Example class 3: uniformly dense trees and SETH

Lm: In backtracking, assume the search tree is s.t. every subtree of depth larger than κn *is of size* $\Omega(2^{\kappa\lambda n})$

Then poly-speed ups whenever QC can handle κn size instances (using space eff. quantum backtracking)

Quantum backtracking can be done **space-efficiently if search heuristics can be done cleverly (space efficiently)**



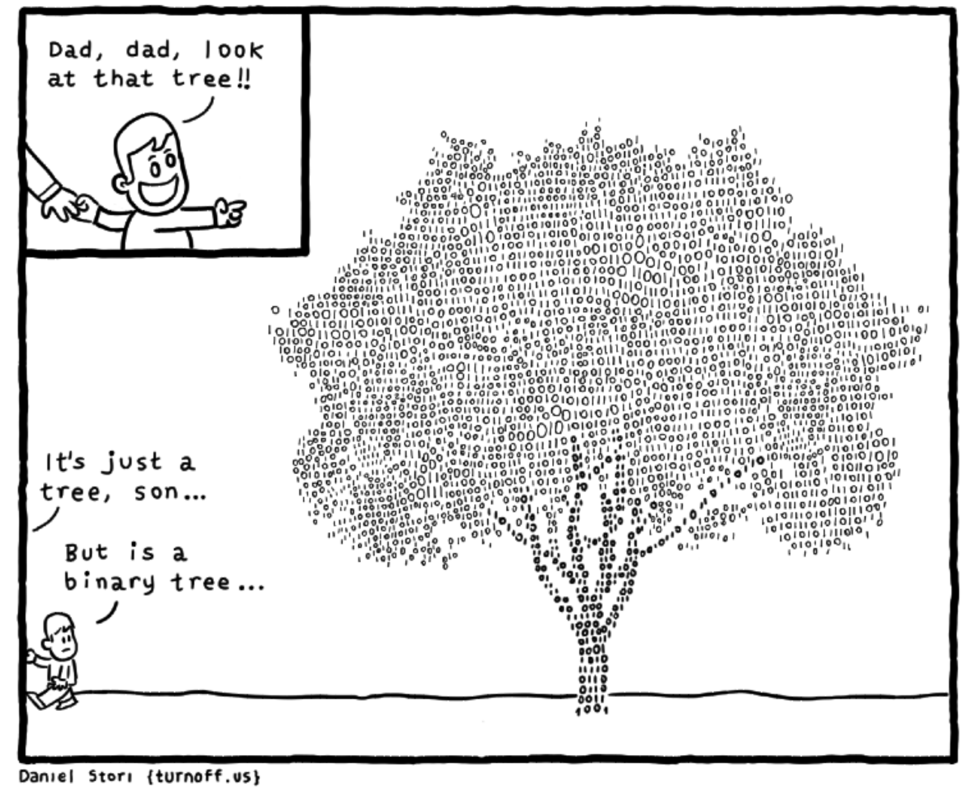
http://turnoff.us/geek/binary-tree/

Example class 3: uniformly dense trees and SETH

Lm: In backtracking, assume the search tree is s.t. every subtree of depth larger than κn *is of size* $\Omega(2^{\kappa\lambda n})$

Then poly-speed ups whenever QC can handle κn size instances (using space eff. quantum backtracking)

Quantum backtracking can be done **space-efficiently if search heuristics can be done cleverly (space efficiently)**



<u>If problem is hard enough, no need to be clever.</u>

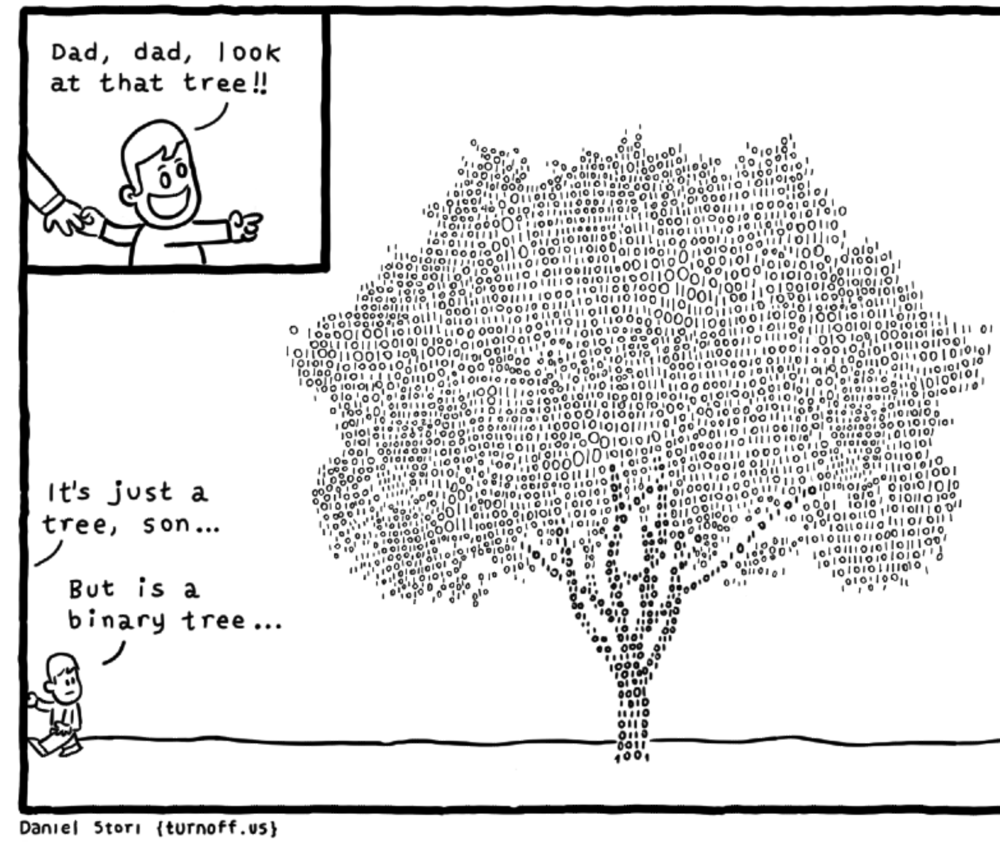
Under strong exponential time hypothesis, for every $\alpha =$ QC-size$/n$ there exists a SAT family for which the hybrid approach is poly faster than *any classical algorithm*

http://turnoff.us/geek/binary-tree/

Example class 3: uniformly dense trees and SETH

Lm: In backtracking, assume the search tree is s.t. every subtree of depth larger than κn *is of size* $\Omega(2^{\kappa\lambda n})$

Then poly-speed ups whenever QC can handle κn size instances



BTW:

$\Omega(2^{\kappa\lambda n})$ - λ is a measure of density

if $\lambda > 1/2$ Grover suffices for speed-up

if $1/2 \geq \lambda > 0$ need q. backtracking for speed-up
(condition 1 of main th. violated for Grover)

http://turnoff.us/geek/binary-tree/

Example 4 (ongoing): PPSZ special cases

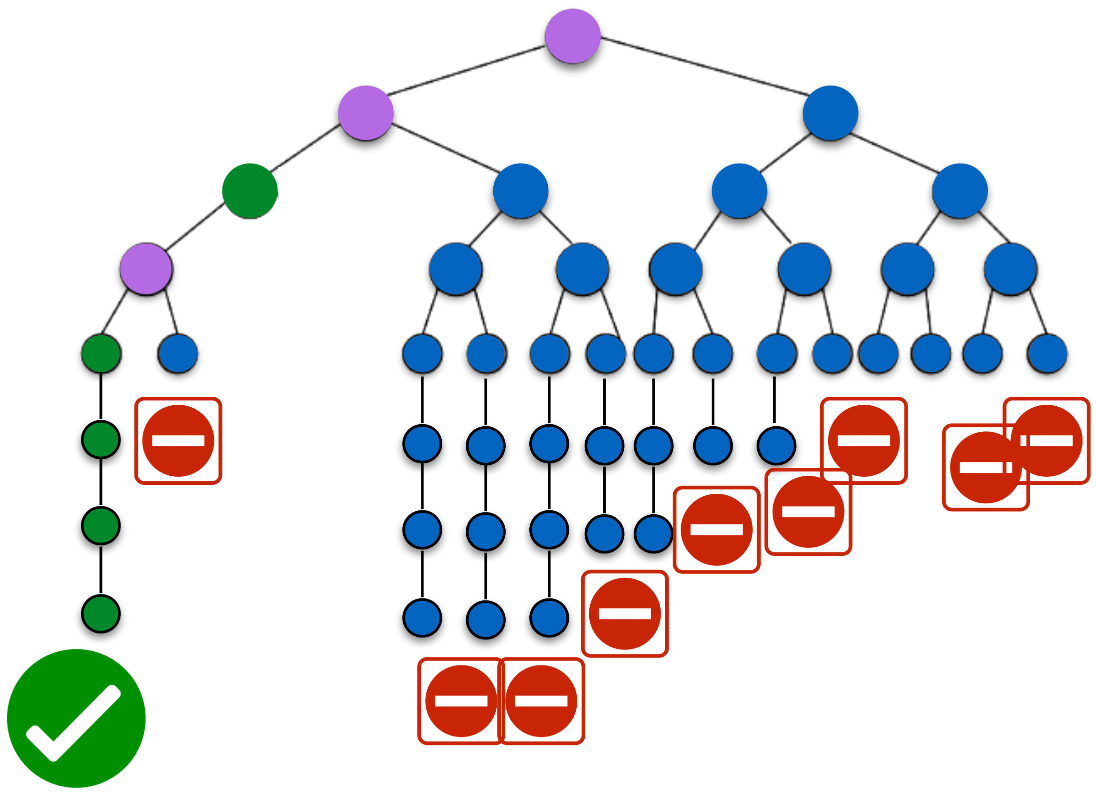
PPSZ = Paturi, Pudlak, Saks and Zane

Basis for fastest exact SAT solver

Order fixed, variable is ***s-implied*** or ***guessed***

PPSZ Theorem: finding solution needs no more than $\approx 0.38n$ guesses (for many orderings)

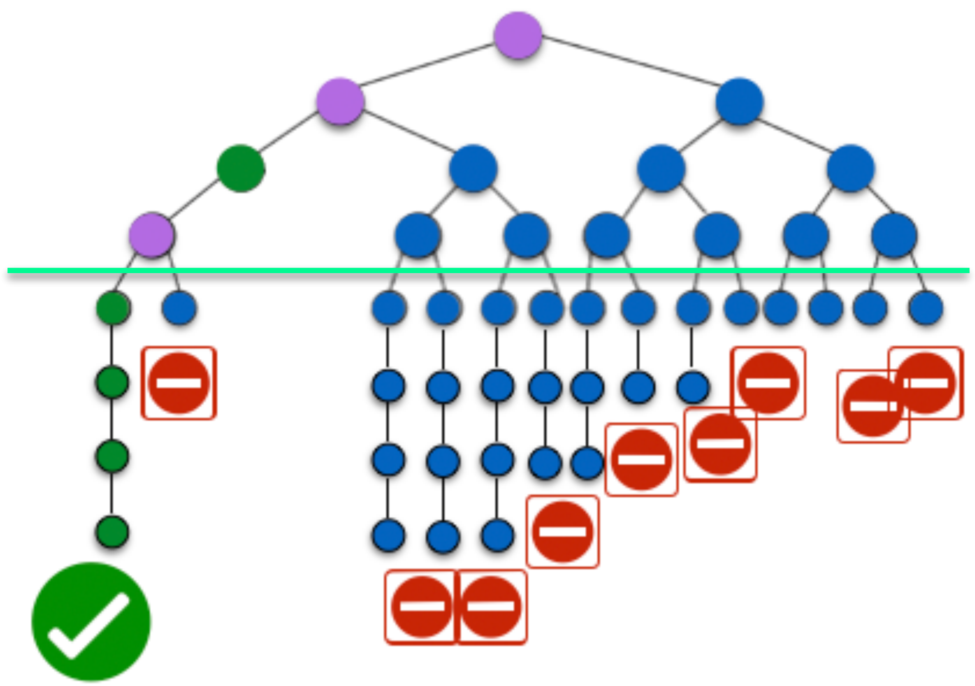
Runtime: $O^*(2^{\gamma n}); \gamma \approx 0.38$



Would have been cool to speed up the best algorithm threshold free…
alas…šmrc (sniff)…

Example 4 (ongoing): PPSZ special cases

PPSZ = Paturi, Pudlak, Saks and Zane



- Bad trees: early guesses → QC no hard work

- Requires *very* space efficient implementation ($o(n)$) of s-implication-based resolution

- Works for some special cases of formulas *(s-implication is efficient!)*

- In general no; it seems it would imply *P-complete* problems can be resolved in *sublinear space*, with *subexponential time*….

<u>Some **theory** and some practice…</u>

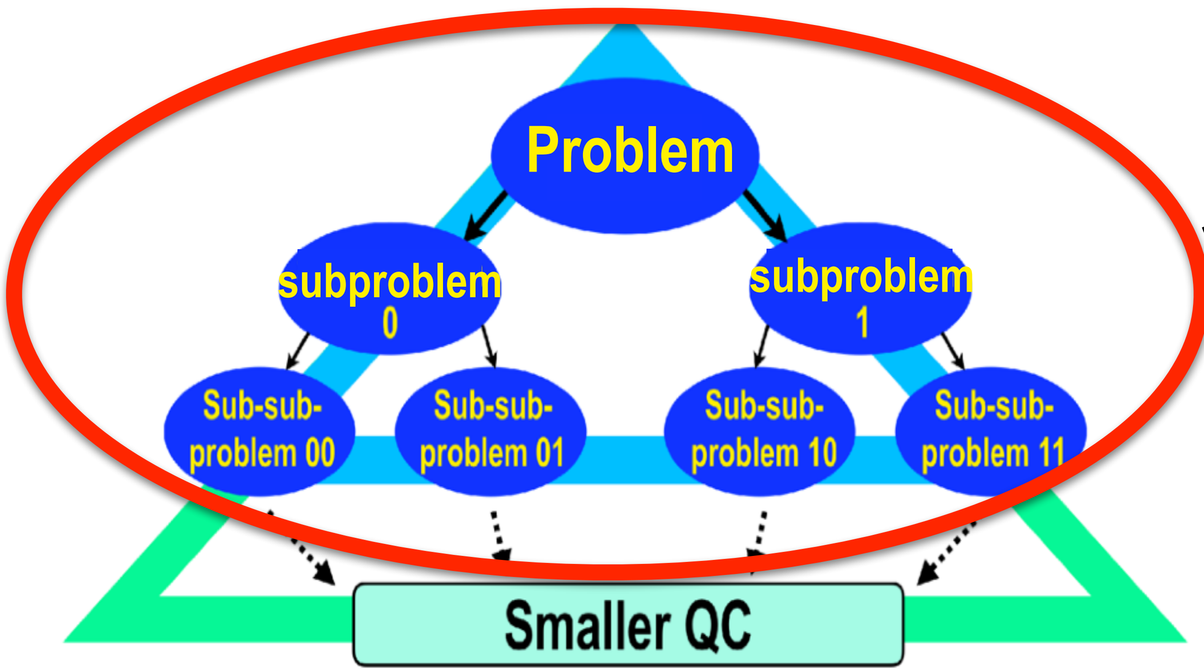
Limits of speed-ups of *all* hybrid methods?

Some **theory** and some practice…

Limits of speed-ups of *all* hybrid methods?

Low hanging fruit results:

For simple hybrid approach, generically poly speed-ups at best (*even if quantum algo is exponentially faster)*



Already this costs
$exp((1-\alpha')\gamma_c n) = exp(\gamma_{opt} n)$

Some **theory** and some practice…

Low hanging fruit results:

We can ofc go beyond "vanilla hybrid". To an (likely exponential) point.

Assume (a specific) $\mathtt{BQP}$ (say) decision problem take $\Omega(2^{\lambda n})$ classically

Then for every α, there exist problem families where $Time(A) = \Omega(2^{\lambda\alpha n})$ yet $Time(Hybrid) = poly(n)$

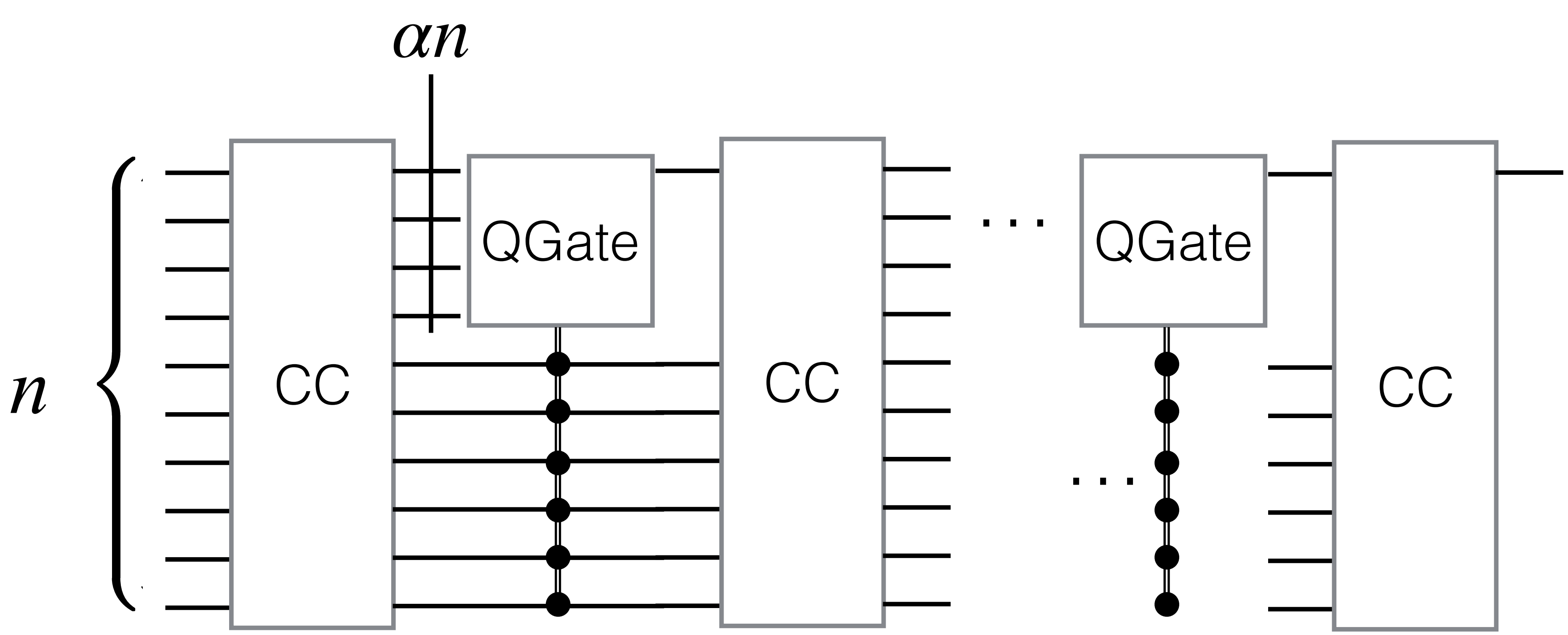
Artificial problem: *QGate(k)-adaptive-boolean-circuit evaluation*

Intuitively "complete" for "what one can do" with a smaller QC

Some **theory** and some practice…

Artificial promise problem: QGate(αn)-adaptive-boolean-circuit evaluation

QGate(k) = quantum poly-sized circuit, k inputs, one output, promise bounded away from 1/2, output the more likely value



QGate(αn, *n*, *Program*), where Program(x), |x| = n, specifies a poly-sized circuit over αn qubits. Kind of “all we can do" in the model. Bounds trivial.

Early days…

Some theory and some **practice**…

- Many other possibilities (e.g. branch-and-bound)

- Speed-ups for heuristics (DPLL & poly-sized trees)

- Real-world speed-ups

  heuristically combining classical with quantum algorithms.
  e.g. QAOA with ML-based algorithm selection used for
  the "chopping up" process (GW v.s. QAOA: arXiv:2001.08271)

  intuition: chop it up such that QC has to do the work as early as possible

Final remarks, loose ends

- why NP-hard? Matters (hard, common (AI, need speedups));
  Because we don't have much better solutions than search, it works

- addressed: can a smaller QC help *asymptotically & provably*

- "smaller"= one possible choice, here most stringent reasonable

  smaller QCs *at least as helpful* as what is shown

- *still a gap between presented model, and what really matters in reality*
- *fine grained analyses…*

- ***pudding:*** *real-world numerical tests needed for heuristics!*

Thank you, and thanks to the co-authors:


Cirac


Ge


Rennela


Laarman


Moussa


Calandra

More on what we do in the neighborhood:

Applied Quantum Algorithms (aQa) Leiden: www.aqa.universiteitleiden.nl

Quantum Software Consortium (QSC): quantumsc.nl