Lecture

Quantum Computing

(CS5070)

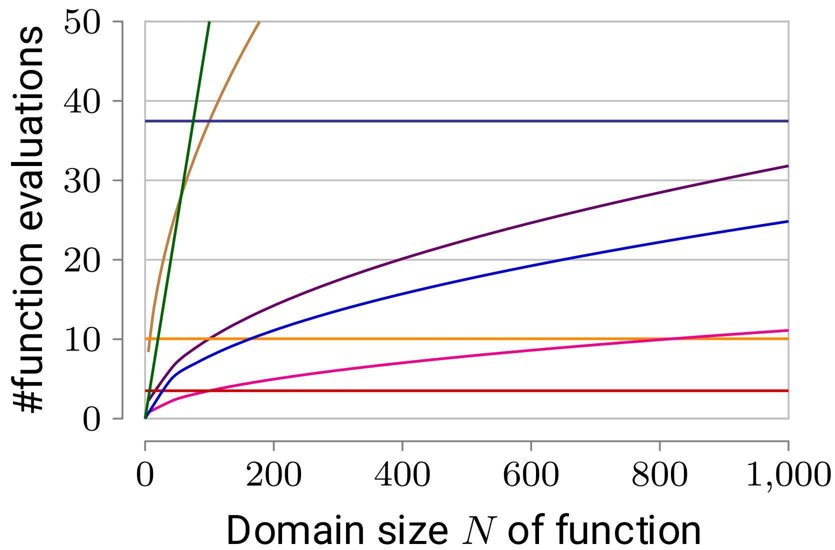
Grover's Search

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Grover's Search Algorithm

- Basis of many other quantum algorithms
- Black box function $f :\{0, \ldots, 2^b - 1\} \mapsto \{true, false\}$ with $N = 2^b$
- Grover's search algorithm finds one $x \in \{0, \ldots, 2^b - 1\}$, such that $f(x) = true$
  - if there is only one solution: $\frac{\pi}{4} \cdot \sqrt{2^b}$ basic steps each of which calls f
    Let $f'(b)$ be runtime complexity of f for testing x to be true:
    $\Rightarrow O(\sqrt{2^b} \cdot f'(b))$
  - if there are k possible solutions: $O(\sqrt{\frac{2^b}{k}} \cdot f'(b))$
- Assuming $f'(b) = O(1)$ on following slides...

Motivation - Quadratic Speedup



- $\frac{N}{2}$ Linear Search

k **known in advance:**

- $\frac{\pi}{4} \cdot \sqrt{N}$ Grover's Search for $k = 1$
- $\frac{\pi}{4} \cdot \sqrt{\frac{N}{k}}$ Grover's Search for $k = \frac{5}{100} \cdot N$
- $\frac{\pi}{4} \cdot \sqrt{\frac{N}{k}}$ Grover's Search for $k = 5$

$k = \frac{5}{100} \cdot N$ **unknown in advance:**

- $\frac{9}{4} \cdot \sqrt{\frac{N}{k}}$ randomized Grover's Search
- $\frac{8 \cdot \pi}{3} \cdot \sqrt{\frac{N}{k}}$ deterministic Grover's Search

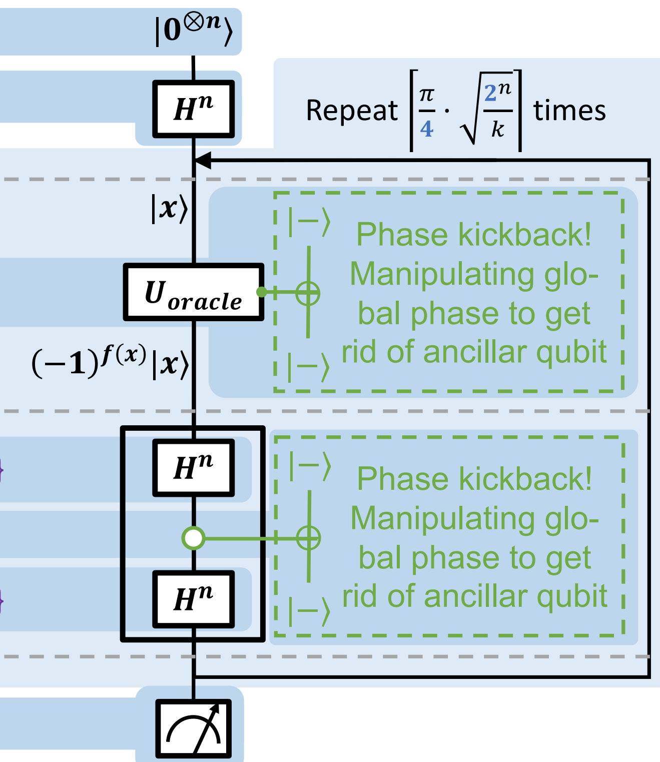
$k = 5$ **unknown in advance:**

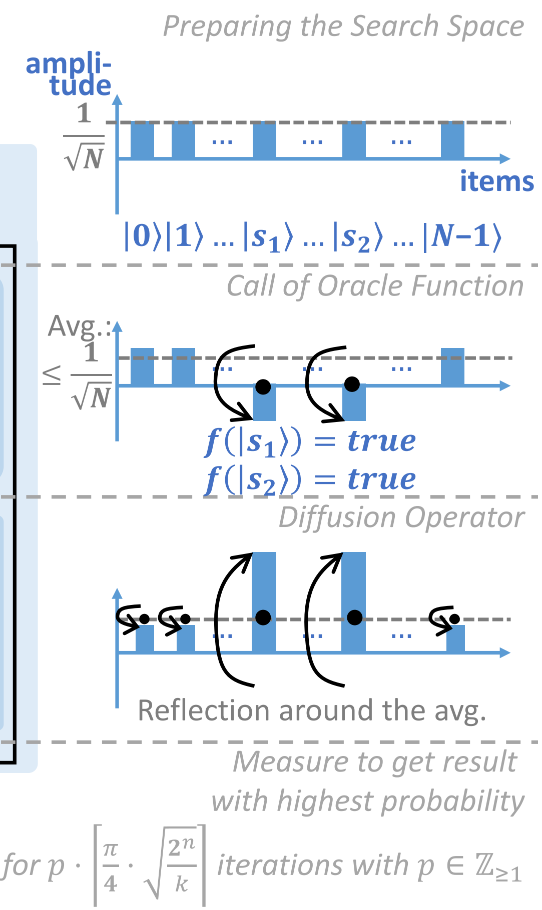
- $\frac{9}{4} \cdot \sqrt{\frac{N}{k}}$ randomized Grover's Search
- $\frac{8 \cdot \pi}{3} \cdot \sqrt{\frac{N}{k}}$ deterministic Grover's Search

N	Linear Search $k = 1$	Grover $k = 1$	$k = \frac{5}{100} \cdot N$			$k = 5$		
			known k	unknown k		known k	unknown k	
				randomized	deterministic		randomized	deterministic
10	5	2.48	3.51	10.06	37.46	1.11	3.81	11.85
100	50	7.85				3.51	10.06	37.47
1000	500	24.84				11.11	31.82	118.48
1000 000	500 000	785.40				351.24	1006.23	3746.57

Grover's Search for k Solutions

```
def grover_k [n:!ℕ, k:!ℕ] (oracle: const uint[n]!→ lifted 𝔹):!ℕ {
  nIterations := ⌊π/4 · √(2^n/k)⌋;
  cand := 0:uint[n];
  for k in [0..n){ cand[k] := H(cand[k]); }
  for k in [0..nIterations) {
    if oracle(cand) { phase(π); }
    for m in [0..n) { cand[m] := H(cand[m]); }
    if cand!=0 { phase(π); }
    for m in [0..n) { cand[m] := H(cand[m]); }
  }
  return measure(cand) as !ℕ;
}
```





Grover's Search: $n = 2$ and oracle is true for $|11\rangle$

1. $cand := \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$
2. $cand := H_2 \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
3. if(oracle(cand)){phase(π);}: $cand := \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^T$
4. Diffusion 1: $cand := H_2 \cdot \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^T = \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^T$
5. Diffusion 2: if(cand$\neq$0){phase(π)}: $cand := \frac{1}{2} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}^T$
6. Diffusion 3: $cand := H_2 \cdot \frac{1}{2} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T = |11\rangle$

We determine the result $|11\rangle$. The 2nd iteration would result in:

3. if(oracle(cand)){phase(π);}: $cand := \begin{bmatrix} 0 & 0 & 0 & -1 \end{bmatrix}^T$
4. Diffusion 1: $cand := H_2 \cdot \begin{bmatrix} 0 & 0 & 0 & -1 \end{bmatrix}^T = \frac{1}{2} \cdot \begin{bmatrix} -1 & 1 & 1 & -1 \end{bmatrix}^T$
5. Diffusion 2: if(cand$\neq$0){phase(π)}: $cand := \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^T$
6. Diffusion 3: $cand := H_2 \cdot \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^T = \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^T$

Grover's Search: $n = 2$ and oracle is true for $|11\rangle$

The 3rd iteration:

3. if(oracle(cand)){phase(π);}: $cand := \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}^T$
4. Diffusion 1: $cand := H_2 \cdot \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}^T = \begin{bmatrix} -1 & 0 & 0 & 0 \end{bmatrix}^T$
5. Diffusion 2: if(cand$\neq$0){phase(π)}: $cand := \begin{bmatrix} -1 & 0 & 0 & 0 \end{bmatrix}^T$
6. Diffusion 3: $cand := H_2 \cdot \begin{bmatrix} -1 & 0 & 0 & 0 \end{bmatrix}^T = \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}^T$

The 4th iteration: Like 1st iteration, but only all $\cdot(-1)$
The 5th iteration: Like 2nd iteration, but only all $\cdot(-1)$
The 6th iteration: Like 3rd iteration, but only all $\cdot(-1)$
The 7th iteration: Like 1st iteration

Takeaways:

- Periodic results
- For more complex examples: Probability of solution increases slowly (and afterward decreases slowly)!

Grover's Search: $n = 2$ and oracle is true for $|11\rangle$

1. iteration **2. iteration** **3. iteration** **4. iteration** **5. iteration** **6. iteration** **7. iteration**

Grover's Search for Unknown Number of Solutions

- If you apply Grover's search with $k \approx$ number of solutions k', then there is a high probability for success
- Several ways for unknown k:
  - Successively applying Grover's search with $k = N, \frac{N}{2}, \frac{N}{4}, \ldots, \frac{N}{2^t}$ until solution is found
    - **Runtime** complexity: With sufficiently high probability, a marked entry will be found by iteration $t = \log_2(\frac{N}{k'}) + c$ for some constant c
      $\Rightarrow$ #iterations $\leq \frac{\pi}{4}\left(1 + \sqrt{2} + \sqrt{4} + \ldots + \sqrt{\frac{N}{k' \cdot 2^c}}\right) = O\left(\sqrt{\frac{N}{k'}}\right)$
  - Next slide: **Randomized** version with $\frac{9}{4}\sqrt{\frac{N}{k'}}$ iterations...
- These methods need to be called several times in case of not finding a solution in preceding calls
  - The probability for success is high, but never 100%

Grover's Search for Unknown Number of Solutions - Silq code

```
// Popular Lehmer generator that uses the prime modulus 2^32-5
def random(state:!ℕ):(!ℕ)×(!ℚ) {
    upperlimit := (2^32 - 5);
    newstate := (state · 279470273) % upperlimit;
    newstateQ := newstate as !ℚ; // for the following division to get a rational number
    return (newstate, newstateQ / upperlimit);
}
// f oracle function, t oracle function as classical function for testing the result!
def grover_unknown_k[n:!ℕ](f: const uint[n] !→ lifted 𝔹, t: !ℕ !→ !𝔹, seed:!ℕ):(!ℤ)×(!ℕ) {
    l := 6/5; // Any value of l strictly between 1 and 4/3 would do...
    m := 1 as !ℝ;
    state := seed; // work with seed to allow for repeated function calls with other results
    while(true){
        (zstate, z) := random(state);
        state = zstate;
        k := floor(z·m) coerce !ℕ;
        result := grover_k[n,k](f);
        if(t(result)){ // call of f would also work, but in this way hybrid approach is more visible
            return (result, state) as (!ℤ)×(!ℕ);
        }
        if(m>=sqrt(2^n)){ return (-1, state) as (!ℤ)×(!ℕ); } // different from cited paper: restart!
        m = min(l·m, sqrt(2^n));
    }
}
```

Optimal Runtime Complexity of Grover's Search

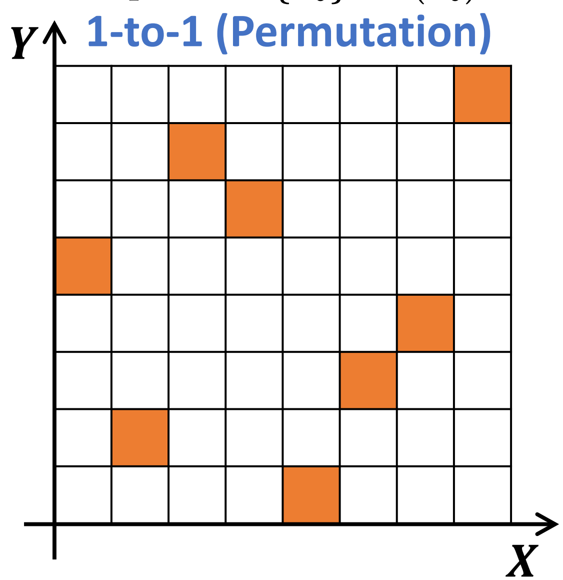
- Grover's algorithm for $k = 1$ and any k: optimal up to sub-constant factors
  - Any algorithm that accesses the database only by using the oracle must apply the oracle at least a $1 - o(1)$ fraction as many times as Grover's algorithm

Grover's Search as Basic Algorithm Used in other Approaches

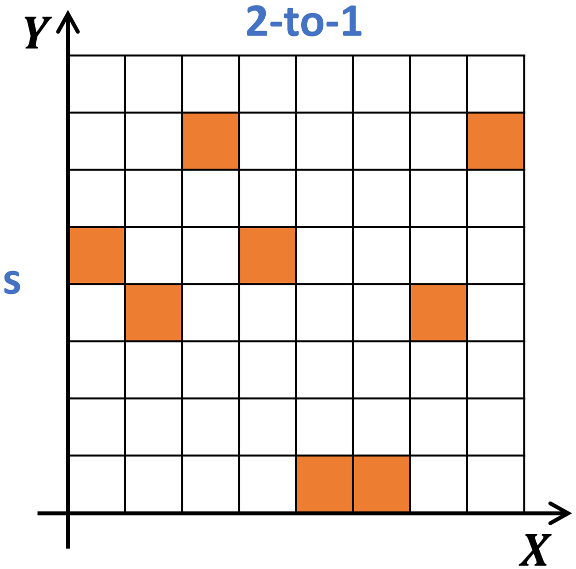
- Solving the collision problem
- Finding Minimum
  - Dynamic Programming
- Solving NP-complete problems by performing exhaustive searches over the set of possible solutions
  - results in quadratic speedup over classical solutions
    - but no polynomial-time solution for NP-complete problems because exponential speedup is **not** reached
  - Example: k-SAT
- ...

Collision Problem

- Given $F : X \to Y$ being either 1-to-1 (permutation) or 2-to-1
  (i.e., $\forall x_0 \in X : \exists x_1 \in X - \{x_0\} : F(x_0) = F(x_1) \wedge \forall x_2 \in X - \{x_0, x_1\} : F(x_2) \neq F(x_0)$)



versus

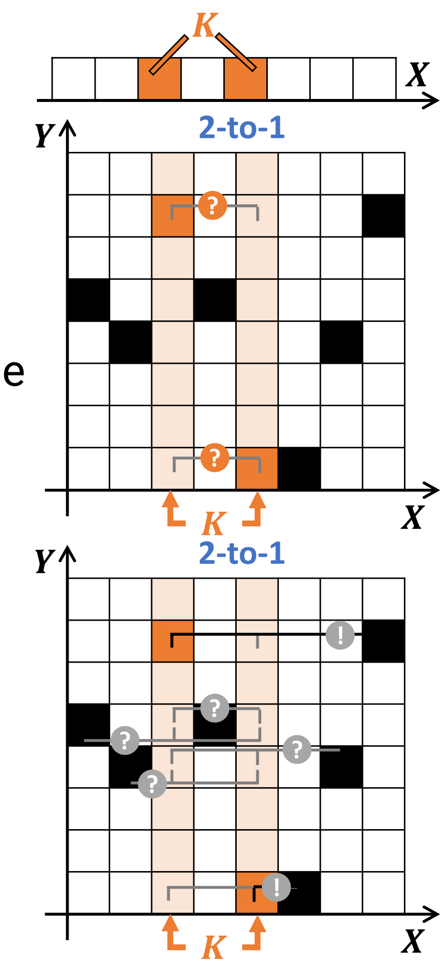


- Classical Randomized Solution
  - Using Birthday Paradox: if we choose (distinct) queries at random, then with high probability we find a collision after $\Theta\left(\sqrt{n}\right)$ queries

Collision Problem

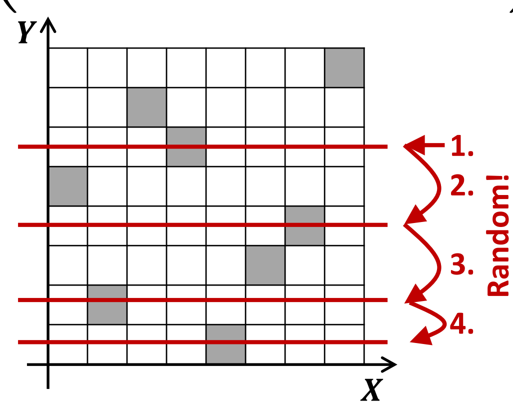
- Quantum Solution
  1. Randomly choose distinct $x_i \in X$ with $i \in \{1, \ldots, k\}$
  2. $K := \{x_1, \ldots, x_k\}$
  3. if($\exists x_a, x_b \in K : x_a \neq x_b \wedge F(x_a) = F(x_b)$)
     then return collision (x_a, x_b)
  4. Build oracle $H : X - K \rightarrow \{true, false\}$
     with $H(x) = true$ if $\exists x_a \in K : F(x) = F(x_a)$, otherwise $H(x) = false$
  5. $x_c :=$ Grover's Search over $X - K$ with oracle H
  6. If $\exists x_a \in K : F(x_c) = F(x_a)$ then return collision (x_a, x_c) else no collision!

  - Runtime $O\left(k + \sqrt{\frac{N-k}{k}} \cdot H'\right)$,
    for $k = \sqrt[3]{N} : O\left(\sqrt[3]{N} \cdot H'\right)$ with H' runtime of H and space $\Theta\left(\sqrt[3]{N}\right)$



Applying Grover - Finding Minimum

- Given function $f : \{0, ..., N-1\} \mapsto \mathbb{R}$
- Algorithm to find minimal $f(x)$:
  1. Choose randomly $x' \in N$
  2. Threshold $t := f(x')$
  3. while(true)
     a) $r :=$ Grover's Search for unknown k with oracle $f(x) < t$
     b) if$(f(r) < t)$ $t := f(r)$ else return t
- Runtime Complexity: $O\left(\frac{45}{4}\sqrt{N} + \frac{7}{10} lg^2(N)\right)$



Applying Grover - Finding Minimum

- Variants:
  - Application-oriented:
    - Estimate good initial threshold t, then continue original algorithm at 3. with threshold t
    - Example (see also next lecture unit):
      - Transactions $T_1, ..., T_p$ to be distributed to m cores of a CPU
      - Total runtime $l = \sum_{i=1}^{p} |T_i|$ of transactions
      - Threshold $t \geq \frac{l}{m}$ for max. runtime on all cores
    - In some applications, the number k of solutions of the oracle can be approximately estimated and used to speed up Grover's search
  - Domain-oriented:

    I) If given domain $[start, end]$ is "small", then continue original algorithm at 3. with $t := end$

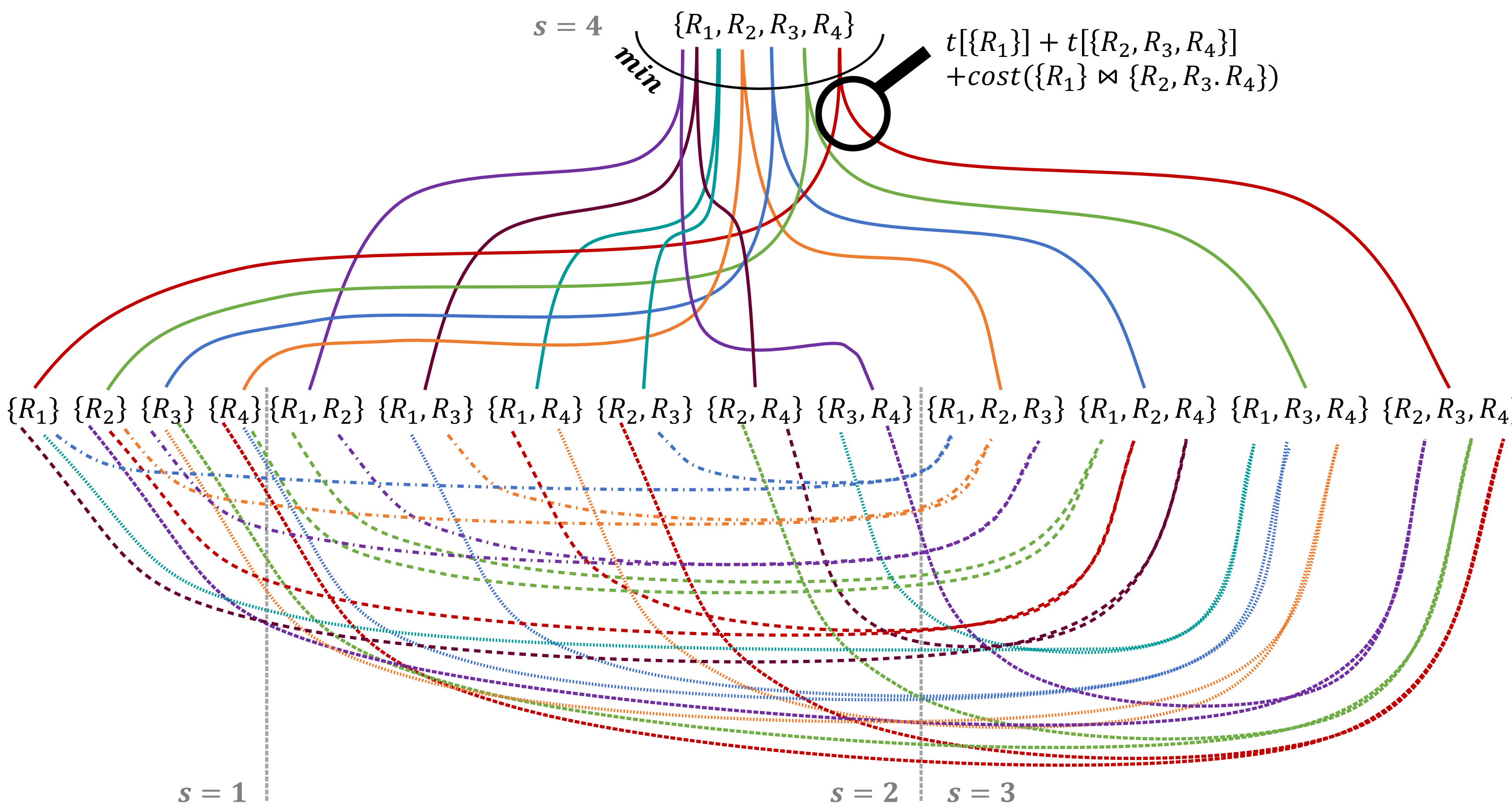
    II) $m := \frac{end - start}{2}$

    III) If Grover's search with oracle $f(x) < m$ finds a solution r, then continue at I) with domain $[start, r]$ else with $[m + 1, end]$

Dynamic Programming - Example: Join Order Optimization

- Find best order (with minimal costs) to join relations $R_1, \ldots, R_n$
- For simplicity of presentation: $cost(R_1 \bowtie R_2) = cost(R_2 \bowtie R_1)$
  - true for many join algorithms, but *not* for e.g. asynchronous hash join
- Classical Algorithm:
  1. Size of subsets to join: $s := 2$
  2. Initialize table: $\forall i \in \{1, \ldots, n\} : t\left[\{R_i\}\right] = cost\left(\{R_i\}\right)$
  3. while($s \leq n$)
     - $\forall S \subseteq \{R_1, \ldots, R_n\}$ with $|S| = s$:
       $t[S] := min\{t[S_1] + t[S_2] + cost(S_1 \bowtie S_2) | S = S_1 \dot{\cup} S_2 \wedge S_1 \neq \varnothing \wedge S_2 \neq \varnothing\}$
     - $s := s + 1$
  4. return $t\left[\{R_1, \ldots, R_n\}\right]$

Dynamic Programming - Example: Join Order Optimization



Combinations from $s = 1$ to $s = 2$ are left out due to simplicity of presentation

Dynamic Programming - Example: Join Order Optimization

- Observations:
  - $\frac{2^s-2}{2} = 2^{s-1} - 1$ combinations for joining a subset of s relations
    - For each relation, put it into the first set S_1 or into the second set S_2 excluding the cases $S_1 = \varnothing \vee S_2 = \varnothing$ as well as the symmetric cases
  - $\Rightarrow \lceil log_2(2^{s-1} - 1) \rceil$ bits for joining a subset of s relations

  $\Rightarrow$ Function for computing total number of bits for all choices during dynamic programming for join order optimization of n relations:

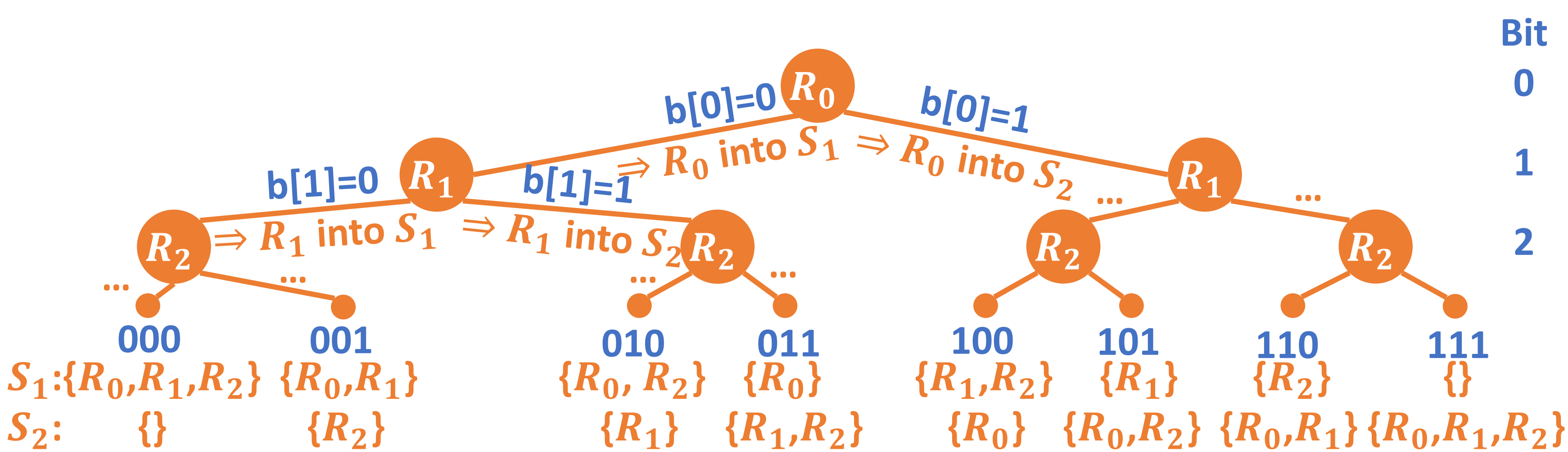
  Algorithm $nrOfBits(s)$

  if $s \leq 2$ then return 0

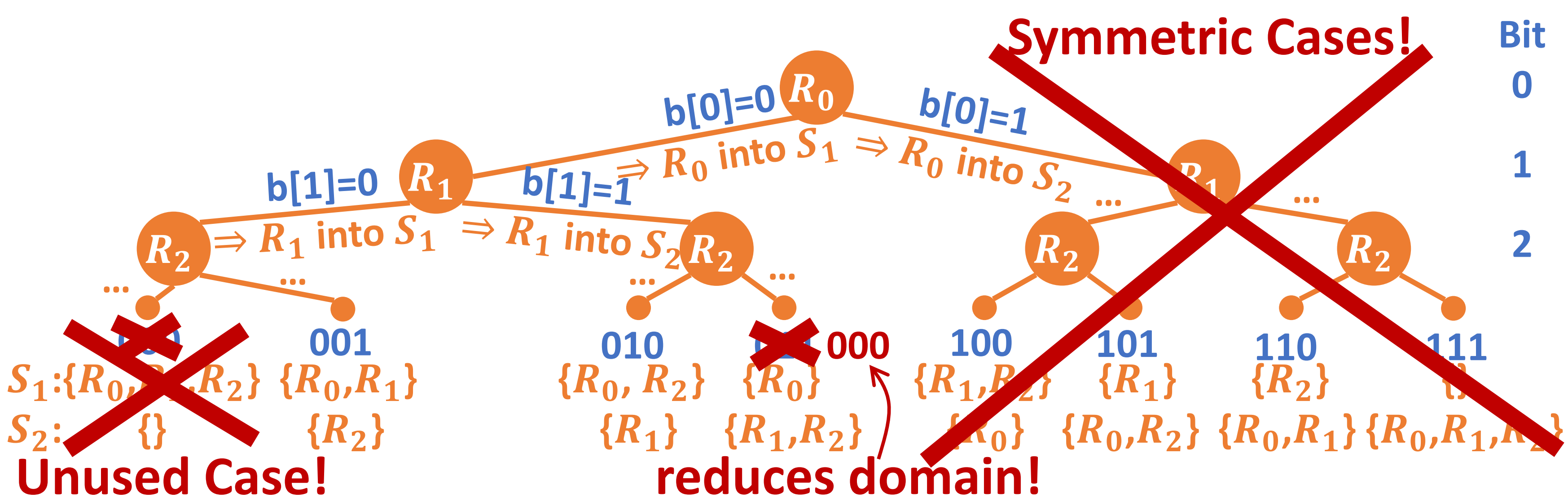
  else return $\lceil log_2(2^{s-1} - 1) \rceil + max\{nrOfBits(i) + nrOfBits(s-i) | i \in 1, ..., \lfloor \frac{s}{2} \rfloor\}$

s	2	3	4	5	6	7	8	9	10
$nrOfBits(s) \equiv \sum_{i=2}^{s} \lceil log_2(2^{s-1} - 1) \rceil$	0	2	5	9	14	20	27	35	44

Quantum Dynamic Programming - Example: Join Order Optimization, Subset Choice



Quantum Dynamic Programming - Example: Join Order Optimization, Subset Choice



Quantum Dynamic Programming - Example: Join Order Optimization

- Algorithm $SubsetChoice(s, b : uint\left[\left\lceil log_2(2^{s-1} - 1)\right\rceil\right], [R_1, ..., R_s])$
  - if $b = 0$ then $b = 2^{s-1} - 2$ // avoiding wasting $b = 0$ for meaningful representation (otherwise it would represent all relations in one subset)
  - $S_1 = \varnothing; S_2 = \varnothing$
  - $\forall i \in \{0, ..., s - 1\}$:
    - if $b[i] = 0$ then $S_1 = S_1 \cup R_i$ else $S_2 = S_2 \cup R_i$
  - return (S_1, S_2)
- Find minimum with following oracle function $f(x)$ in Grover's search:
  - return $recCost([R_1, ..., R_n], x, 0)[1]$ // see next slide!

Quantum Dynamic Programming - Example: Join Order Optimization

- Algorithm $recCost(R, x, b)$
  - $s := |R|$
  - if $s = 1$ then return $(b, cost(R[0]))$
  - if $s = 2$ then return $(b, cost(R[0]) + cost(R[1]) + cost(R[0] \bowtie R[1]))$
  - $bits := \lceil log_2(2^{s-1} - 1) \rceil$
  - $(S_1, S_2) := SubsetChoice(s, x[b, bits - 1], R)$
  - $b := b + bits$
  - $l := recCost(S_1, x, b)$
  - $b := b + l[0]$
  - $r := recCost(S_2, x, b)$
  - return $(b + r[0], l[1] + r[1] + cost(S_1 \bowtie S_2))$

Quantum Dynamic Programming - Example: Join Order Optimization

- Hybrid approaches for reducing number of qubits
  - Quantum dynamic programming for subsets of the problem combined with classical approach
- All costs precomputed versus cost computation of joins within quantum circuit based on histograms of input relations
  - Smaller quantum circuits versus larger speedup

k-SAT-Problem

- Boolean satisfiability problem: Problem of determining if there exists an interpretation that satisfies a given Boolean formula
  - In other words: Do we find an assignment of Boolean variables to the values TRUE or FALSE in such a way that the formula evaluates to TRUE?

    $\Rightarrow$ Formula is satisfiable
- k-SAT-Problem: Boolean satisfiability problem with k variables
  - Every k-SAT-problem can be written as conjunctive normal form (CNF):

    $\bigwedge \bigvee (\neg) x_i$

k-SAT-Problem

- Example for 3SAT:

$$f(v_1, v_2, v_3) = (\neg v_1 \vee \neg v_2 \vee \neg v_3) \wedge (v_1 \vee \neg v_2 \vee v_3) \wedge (v_1 \vee v_2 \vee \neg v_3) \wedge (v_1 \vee \neg v_2 \vee \neg v_3)$$

v_1	v_2	v_3	$f(v_1, v_2, v_3)$
true	true	true	false
true	true	false	true
true	false	true	true
true	false	false	true
false	true	true	false
false	true	false	false
false	false	true	false
false	false	false	true

k-SAT-Problem

- Apply Grover's search with following oracle function:
  - $oracle(x : uint[k])$
  - $\forall i \in 1, ..., k : v_i := x[i-1]$
  - return $f(v_1, ..., v_k)$
- If Grover's search returns a solution,
  then $f(v_1, ..., v_k)$ is satisfiable,
  else it is unsatisfiable (with a high probability)
- Laufzeit $O\left(\sqrt{N}\right) = O\left(1.414^n\right)$ with $n = k$
  - Classical approach for 3SAT: $O\left(1.307^n\right)$
  - Grover beats classical approaches for larger k

Summary and Conclusions

- Grover's search with quadratic speedup
  - for known $k = 1$ and $k > 1$
  - for unknown k
- Applications of Grover's search
  - Collision Problem
  - Finding minimum
  - Quantum dynamic programming with example of join order optimization
  - k-SAT