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

\[ \sim \frac{N}{2} \text{ Linear Search} \]
\[ k \text{ known in advance:} \]
\[ \sim \frac{\pi}{4} \cdot \sqrt{N} \text{ Grover’s Search for } k = 1 \]
\[ \sim \frac{\pi}{4} \cdot \sqrt{\frac{N}{k}} \text{ Grover’s Search for } k = \frac{5}{100} \cdot N \]
\[ \sim \frac{\pi}{4} \cdot \sqrt{\frac{N}{k}} \text{ Grover’s Search for } k = 5 \]
\[ k = \frac{5}{100} \cdot N \text{ unknown in advance:} \]
\[ \sim \frac{9}{4} \cdot \sqrt{\frac{N}{k}} \text{ randomized Grover’s Search} \]
\[ \sim \frac{8 \cdot \pi}{3} \cdot \sqrt{\frac{N}{k}} \text{ deterministic Grover’s Search} \]
\[ k = 5 \text{ unknown in advance:} \]
\[ \sim \frac{9}{4} \cdot \sqrt{\frac{N}{k}} \text{ randomized Grover’s Search} \]
\[ \sim \frac{8 \cdot \pi}{3} \cdot \sqrt{\frac{N}{k}} \text{ deterministic Grover’s Search} \]

<table>
<thead>
<tr>
<th>(N)</th>
<th>Linear Search (k = 1)</th>
<th>Grover Search (k = 1)</th>
<th>(k = \frac{5}{100} \cdot N)</th>
<th>(k = 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(N)</td>
<td>known (k)</td>
<td>unknown (k)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>randomized</td>
<td>deterministic</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>2.48</td>
<td>3.51</td>
<td>10.06</td>
</tr>
<tr>
<td>100</td>
<td>50</td>
<td>7.85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>24.84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>500000</td>
<td>785.40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Grover's Search for $k$ Solutions

```python
def grover_k[n: !N, k: !N] (oracle: const uint[n] -> lifted [B]: !N) {
    n_iter := \left\lceil \frac{\pi}{4} \cdot \sqrt{\frac{2^n}{k}} \right\rceil;
    cand := 0: uint[n];
    |0^\otimes n\rangle
    for k in [0..n] { cand[k] := H(cand[k]); }
    for k in [0..n_iter] {
        if oracle(cand) { phase(\pi); }
        for m in [0..n] { cand[m] := H(cand[m]); }
        if cand != 0 { phase(\pi); }
    }
    return measure(cand) as !N;
}
```

Preparing the Search Space

- $|0, |1, ..., |s_1, |s_2, ..., |N-1\rangle$
- Call of Oracle Function
  - $f(|s_1\rangle) = \text{true}$
  - $f(|s_2\rangle) = \text{true}$
- Diffusion Operator
  - Phase kickback! Manipulating global phase to get rid of ancillary qubit
- Reflection around the avg.
  - Measure to get result with highest probability

$$p \cdot \left[ \frac{\pi}{4} \cdot \sqrt{\frac{2^n}{k}} \right]$$ iterations with $p \in \mathbb{Z}_{\geq 1}$
Grover's Search: $n = 2$ and oracle is true for $|11\rangle$ 

1. $cand := \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$

2. $cand := H_2 \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

3. if (oracle(cand))(phase($\pi$)); $cand := \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^T$

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($\pi$)); $cand := \frac{1}{2} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}^T$

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($\pi$)); $cand := \begin{bmatrix} 0 & 0 & 0 & -1 \end{bmatrix}^T$

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($\pi$)); $cand := \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^T$

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\(\text{oracle(cand)}\}\{\text{phase} (\pi)\};\) $cand := \frac{1}{2} \cdot \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}^T$

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\(\text{cand} \neq 0\}\{\text{phase} (\pi)\};\) $cand := \begin{bmatrix} -1 & 0 & 0 & 0 \end{bmatrix}^T$

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 $|11\rangle$
Grover's Search for Unknown Number of Solutions

- If you apply Grover's search with $k \approx \text{number of solutions } k'$, then there is a high probability for success
- Several ways for unknown $k$:
  - Successively applying Grover's search with $k = N, \frac{N}{2}, \frac{N}{4}, \ldots, \frac{N}{k'}$ until solution is found
    - Runtime complexity: With sufficiently high probability, a marked entry will be found by iteration $t = \log_2 \left( \frac{N}{k'} \right) + c$ for some constant $c$
      \[ \Rightarrow \text{iterations} \leq \frac{\pi}{4} \left( 1 + \sqrt{2} + \sqrt{4} + \ldots + \sqrt{\frac{N}{k'.2^c}} \right) = O \left( \sqrt{\frac{N}{k'}} \right) \]
    - 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

```silq
// 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 ℤ, t: !N !→ !ℤ, 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 $F : X \rightarrow Y$ being either 1-to-1 (permutation) or 2-to-1
  (i.e., $\forall x_0 \in X : \exists x_1 \in X - \{x_0\} : F(x_0) = F(x_1) \land \forall x_2 \in X - \{x_0, x_1\} : F(x_2) \neq F(x_0)$)

- Classical Randomized Solution
  - Using Birthday Paradox: if we choose (distinct) queries at random, then with high probability we find a collision after $\Theta(\sqrt{n})$ 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(\( \exists x_a, x_b \in K : x_a \neq x_b \land F(x_a) = F(x_b) \))
     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. \( x_c := \text{Grover's Search over } X - K \) 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 \( k = 3\sqrt{N} : O \left( 3\sqrt{N} \cdot H' \right) \) with \( H' \) runtime of \( H \) and space \( \Theta \left( 3\sqrt{N} \right) \)
Applying Grover - Finding Minimum

- Given function \( f : \{0, \ldots, N - 1\} \rightarrow \mathbb{R} \)
- Algorithm to find minimal \( f(x) \):
  1. Choose randomly \( x' \in N \)
  2. Threshold \( t := f(x') \)
  3. while(true)
     a) \( r := \text{Grover's Search for unknown } k \text{ with oracle } f(x) < t \)
     b) if \( f(r) < t \) then \( t := f(r) \) else return \( t \)
- Runtime Complexity: \( O \left( \frac{45}{4} \sqrt{N} + \frac{7}{10} \log^2(N) \right) \)
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 $T_1, \ldots, T_p$ 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: $\forall i \in \{1, \ldots, n\} : t[\{R_i\}] = cost(\{R_i\})$
  3. while($s \leq n$)
     - $\forall S \subseteq \{R_1, \ldots, R_n\}$ with $|S| = s$:
       - $t[S] := \min\{t[S_1] + t[S_2] + cost(S_1 \bowtie S_2) | S = S_1 \cup S_2 \land S_1 \neq \emptyset \land S_2 \neq \emptyset\}$
     - $s := s + 1$
  4. return $t[\{R_1, \ldots, R_n\}]$
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 \(S_1 = \emptyset \lor S_2 = \emptyset\) as well as the symmetric cases
  
  \(- \implies \left\lfloor \log_2(2^{s-1} - 1) \right\rfloor\) 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 \(\left\lfloor \log_2(2^{s-1} - 1) \right\rfloor + \max\{nrOfBits(i) + nrOfBits(s - i)\} \mid i \in 1, \ldots, \left\lfloor \frac{s}{2} \right\rfloor\)

<table>
<thead>
<tr>
<th>(s)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(nrOfBits(s)) (\equiv \sum_{i=2}^{s} \left\lfloor \log_2(2^{s-1} - 1) \right\rfloor)</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>14</td>
<td>20</td>
<td>27</td>
<td>35</td>
<td>44</td>
</tr>
</tbody>
</table>
Quantum Dynamic Programming -
Example: Join Order Optimization,
Subset Choice
Quantum Dynamic Programming - Example: Join Order Optimization, Subset Choice

Unused Case! reduces domain!

Symmetric Cases!
Quantum Dynamic Programming - Example: Join Order Optimization

- Algorithm $\text{SubsetChoice}(s, b : \text{uint} \left[\lceil \log_2(2^{s-1} - 1) \rceil \right], [R_1, ..., R_s])$
  - 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)
  - $S_1 = \emptyset; S_2 = \emptyset$
  - $\forall i \in \{0, ..., s - 1\}$:
    - 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 $\text{recCost}([R_1, ..., R_n], x, 0)[1]$ // 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 $(b, cost(R[0]) + cost(R[1]) + cost(R[0] \bowtie R[1]))$
  - $bits := \left\lceil \log_2(2^{s-1} - 1) \right\rceil$
  - $(S_1, S_2) := SubsetChoice(s, x[b, bits - 1], R)$
  - $b := b + bits$
  - $l := recCost(S_1, x, b)$
  - $b := b + l[0]$
  - $r := recCost(S_2, x, b)$
  - 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):
    $\land \lor (\neg x_i)$
\( k \)-SAT-Problem

- **Example for 3SAT:**

\[
f(v_1, v_2, v_3) = (\neg v_1 \lor \neg v_2 \lor \neg v_3) \land (v_1 \lor \neg v_2 \lor v_3) \land (v_1 \lor v_2 \lor \neg v_3) \land (v_1 \lor \neg v_2 \lor \neg v_3)
\]

<table>
<thead>
<tr>
<th>( v_1 )</th>
<th>( v_2 )</th>
<th>( v_3 )</th>
<th>( f(v_1, v_2, v_3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
$k$-SAT-Problem

- Apply Grover's search with following oracle function:
  - $oracle(x : \text{uint}[k])$
  - $\forall i \in 1, \ldots, k : v_i := x[i - 1]$
  - return $f(v_1, \ldots, v_k)$
- If Grover's search returns a solution, then $f(v_1, \ldots, v_k)$ 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