Improved Classical and Quantum Algorithms for Subset-Sum

Xavier Bonnetain\textsuperscript{1}, Rémi Bricout\textsuperscript{2,3}, André Schrottenloher\textsuperscript{3}, Yixin Shen\textsuperscript{4}

\begin{itemize}
  \item \textsuperscript{1} Institute for Quantum Computing, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada
  \item \textsuperscript{2} Sorbonne Université, Collège Doctoral, F-75005 Paris, France
  \item \textsuperscript{3} Inria, France
  \item \textsuperscript{4} Université de Paris, IRIF, CNRS, F-75013 Paris, France
\end{itemize}

March 6, 2020
Outline

1. Introduction
2. Representations
3. Subset-Sum with Quantum Search
4. Subset-Sum with Quantum Walks
Introduction
The Subset-Sum Problem

Problem

Given: $a = (a_1, \ldots, a_n)$ a vector of $\ell$-bit integers, and an $\ell$-bit target $S$, find $e = (e_1, \ldots, e_n) \in \{0, 1\}^n$ such that $e \cdot a = \sum_i e_i a_i = S \mod 2^\ell$.

- The decision version is NP-complete
- The low-density case ($\ell \gg n$) is related to lattice SVP
- The high-density case ($\ell \ll n$) is solvable efficiently
- The density-1 case ($\ell \approx n$) is hard
Subset-sums in (post-quantum) cryptography

- Repeatedly used as a hard problem for post-quantum cryptography \(^a\)
- Similar techniques that we will see in this presentation apply to other problems (generic decoding algorithms) \(^b\)
- Solving subset-sums is also useful in quantum hidden shift algorithms \(^c\)

\(^a\) Lyubashevsky, Palacio, and Segev, “Public-Key Cryptographic Primitives Provably as Secure as Subset Sum”, TCC 10
\(^b\) Kachigar and Tillich, “Quantum Information Set Decoding Algorithms”, PQCrypto 17
\(^c\) Bonnetain, Improved Low-qubit Hidden Shift Algorithms, 2019
The random Subset-Sum Problem

Problem

Given: a = (a₁, ..., aₙ) a vector of \( n \)-bit integers, and an \( n \)-bit target \( S \), find \( e = (e₁, ..., eₙ) \in \{0, 1\}^n \) such that \( e \cdot a = \sum_i e_i a_i = S \mod 2^n \); where \( a, S \) are selected uniformly at random.

- Classical and quantum algorithms run in time \( \tilde{O}(2^{\beta n}) \): we are interested in the value of \( \beta \)
- In this talk, we optimize the time exponent (not the memory)
- We consider w.l.o.g. that \( e \) has Hamming weight \( \frac{n}{2} \)
The time is $\widetilde{O}(2^{\beta n})$.

<table>
<thead>
<tr>
<th>Technique</th>
<th>$\beta$</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIM</td>
<td>0.5</td>
<td>HS74 (Slide 8)</td>
</tr>
<tr>
<td>4-list merge</td>
<td>0.5</td>
<td>SS81 (Slide 9)</td>
</tr>
<tr>
<td>${0, 1}$</td>
<td>0.3370</td>
<td>HGJ10 (Slide 18)</td>
</tr>
<tr>
<td>${-1, 0, 1}$</td>
<td>0.2909</td>
<td>BCJ11 (Slide 23)</td>
</tr>
<tr>
<td>${-1, 0, 1} + NN$</td>
<td>0.287</td>
<td>Ilya Ozerov’s PhD thesis</td>
</tr>
<tr>
<td>${-1, 0, 1, 2}$</td>
<td>0.283</td>
<td>Ours</td>
</tr>
</tbody>
</table>
Classical algorithm: meet-in-the-middle

Cut the solution \( e = (0 \ldots 0 | * \ldots * ) + ( * \ldots * | 0 \ldots 0 ) \)

\[ \text{n/2 bits} \quad \text{n/2 bits} \]

\[ \text{2}^\frac{n}{2} \text{ choices: } L_l \]

\[ \text{n/2 bits} \quad \text{n/2 bits} \]

\[ \text{2}^\frac{n}{2} \text{ choices: } L_r \]

Then find \( e_l \in L_l, e_r \in L_r \) s.t. \( e_l \cdot a = -e_r \cdot a + S \mod 2^n \).

**Complexities**

Time: \( \mathcal{O}(2^{n/2}) \) (best worst-case time); Memory: \( \mathcal{O}(2^{n/2}) \)

---

Horowitz and Sahni, “Computing Partitions with Applications to the Knapsack Problem”, J. ACM
Schroeppel and Shamir’s 4-list merging

By cutting in 4 instead of 2, we can decrease the memory to $2^{n/4}$.

$$e = (\star|0|0|0) + (0|\star|0|0) + (0|0|\star|0) + (0|0|0|\star)$$

$e_0 \in L_0 \quad e_1 \in L_1 \quad e_2 \in L_2 \quad e_3 \in L_3$

We now look for $(e_0, e_1, e_2, e_3) \in L_0 \times L_1 \times L_2 \times L_3$ s.t.

$$a \cdot (e_0 + e_1 + e_2 + e_3) = S \mod 2^n$$

$\Rightarrow$ there is still one solution among $2^{n/4} \times 2^{n/4} \times 2^{n/4} \times 2^{n/4}$.

---

Schroeppel and Shamir, “A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems”, SIAM 81
Schroeppel and Shamir’s 4-list merging (ctd.)

- Choose an \( \frac{n}{4} \)-bit number \( c \mod M \)
- Repeat for every value of \( c \):

\[
\begin{align*}
\mathbf{e}_0 & \in L_0 \\
\mathbf{e}_1 & \in L_1 \\
\mathbf{e}_2 & \in L_2 \\
\mathbf{e}_3 & \in L_3
\end{align*}
\]

\[
\begin{align*}
\mathbf{e}_0 + \mathbf{e}_1 & \text{ s. t. } \\
\mathbf{a} \cdot (\mathbf{e}_0 + \mathbf{e}_1) & = (c + S) \mod M \\
\text{Size } 2^{n/4}
\end{align*}
\]

\[
\begin{align*}
\mathbf{e}_2 + \mathbf{e}_3 & \text{ s. t. } \\
\mathbf{a} \cdot (\mathbf{e}_2 + \mathbf{e}_3) & = -c \mod M \\
\text{Size } 2^{n/4}
\end{align*}
\]

Solution with prob. \( 2^{-n/4} \)

Complexities

- **Time:** \( O(2^{n/4} \times 2^{n/4}) \)
- **Memory:** \( O(2^{n/4}) \)
<table>
<thead>
<tr>
<th>Introduction</th>
<th>Representations</th>
<th>Subset-Sum with Quantum Search</th>
<th>Subset-Sum with Quantum Walks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Representations
Breaking the $2^{n/2}$ bound

- When we have one solution among $2^n$ tuples, we don't know of any better time than $2^{n/2}$
- The idea of Howgrave-Graham and Joux (HGJ): cut $e$ with respect to its Hamming weight

Suppose that $e$ is of weight $n/2$ (worst case). Write for example:

\[
\begin{align*}
e &= e_0 + e_1 + e_2 + e_3 \\
&= \text{Weight } n/2 + \text{Weight } n/8 + \text{Weight } n/8 + \text{Weight } n/8
\end{align*}
\]

notice that: \( \binom{n}{n/8}^4 \approx 2^{2.174n} \gg \binom{n}{n/2} \approx 2^n \) : many solution tuples!

---

Notations

We introduce distributions and weight constraints.

**Distributions**

\[ e \in D^n[\alpha] \text{ if } e \text{ contains } \alpha n \text{ "1" and } (1 - \alpha) n \text{ "0".} \]

Then if \( e_1 \in D^n[\alpha_1], e_2 \in D^n[\alpha_2] \) we have \( e_1 + e_2 \in D^n[\alpha_1 + \alpha_2] \) with some probability (to be continued).

**Weight constraints**

\( e \) has “a \( cn \)-bit weight constraint” if we are able to constrain the (knapsack) weight of \( e \) as \( e \cdot a = s \mod M \) for (previously) chosen \( cn \)-bit integers \( M \) and \( s \).

If \( e_1 \) has a \( cn \)-bit-weight cons. and \( e_2 \) has a \( cn \)-bit-weight cons., then \( e_1 + e_2 \) as well by linearity (but the precise moduli don’t matter!)
Example: Schroeppel-Shamir

$L_0, 0$
$D^{n/4}[\frac{1}{2}] \times \{0\}^{3n/4}$

$L_1, 0$
$\{0\}^{n/4} \times D^{n/4}[\frac{1}{2}] \times \{0\}^{n/2}$

$L_2, 0$
$\{0\}^{n/2} \times D^{n/4}[\frac{1}{2}] \times \{0\}^{n/4}$

$L_3, 0$
$\{0\}^{3n/4} \times D^{n/4}[\frac{1}{2}]$

$L_1^1, c = \frac{1}{4}$
$D^{n/2}[\frac{1}{2}] \times \{0\}^{n/2}$

$L_1^0, c = \frac{1}{4}$
$\{0\}^{n/2} \times D^{n/2}[\frac{1}{2}]$
The solution $e$ is the only vector with an $n$-bit weight constraint and $e \in D^n[1/2]$.

- We start from vectors $e$ with a 0-bit weight constraint and distributions $D^n[\alpha]$.
- We sum them, trying to increase the weight constraint ("merging").
- Eventually we get to a $n$-bit weight constraint and a distribution $D^n[1/2]$. 
## Merging and filtering

**List** $L_2$  
$\subseteq D^n[\alpha_2]$  
cons. $cn$

**List** $L_1$  
$\subseteq D^n[\alpha_1]$  
cons. $cn$

**List** $L$  
size $|L| = |L_1||L_2|/(2^{\text{nd}})$  
cons. $(c+d)n$

**List** $L^f$  
$\subseteq D^n[\alpha_1 + \alpha_2]$  
size $|L^f| = |L| \times p$  
cons. $(c+d)n$

---

**Step 1: merging**

Find pairs with more constrained weights.

We produce $L$ in time  
\[
\max(\min(|L_1|, |L_2|), |L|).
\]

**Step 2: filtering**

Keep only the $e_1 + e_2$ that conform to the expected distribution.

$p$ is the “filtering probability” for:  
\[
D^n[\alpha_1] \times D^n[\alpha_2] \rightarrow D^n[\alpha_1 + \alpha_2]
\]
Merging and filtering (ctd.)

Heuristic

The vectors in $L^f$ are uniformly distributed in $D^n[\alpha_1 + \alpha_2]$.

Approximations

- Representation sets have size: $D^n[\alpha] = \binom{n}{\alpha n} \approx 2^{h(\alpha)n}$
- $h(x) = -x \log x - (1 - x) \log(1 - x)$
- In general we have a filtering probability $D^n[\alpha_1] \times D^n[\alpha_2] \rightarrow D^n[\alpha_1 + \alpha_2]$ of: $\approx 2^{(h(\alpha_1/(1-\alpha_2))-h(\alpha_1))n}$
The HGJ algorithm

$L_0^3 \{0^{n/2}\} \times D^{n/2}[\frac{1}{8}]$

$L_1^3 D^{n/2}[\frac{1}{8}] \times \{0^{n/2}\}$

$L_0^2, c_2$

$L_1^2 D^n[\frac{1}{8}]$

$L_1^1, c_1$

$L_0^1 D^n[\frac{1}{4}]$

$L^0, 1 D^n[\frac{1}{2}]$

$L_4^3 \{0^{n/2}\} \times D^{n/2}[\frac{1}{8}]$

$L_3^3 D^{n/2}[\frac{1}{8}] \times \{0^{n/2}\}$

$L_2^2, c_2 D^n[\frac{1}{8}]$

$L_2^1, c_1 D^n[\frac{1}{4}]$

$L_6^3 \{0^{n/2}\} \times D^{n/2}[\frac{1}{8}]$

$L_5^3 D^{n/2}[\frac{1}{8}] \times \{0^{n/2}\}$

$L_4^2, c_2 D^n[\frac{1}{8}]$

$L_3^1, c_1 D^n[\frac{1}{4}]$

$L_7^3 D^{n/2}[\frac{1}{8}] \times \{0^{n/2}\}$
HGJ step 3: left-right split with a modulus

At this level we can afford to **merge without filtering**: find vectors $\mathbf{e} \in D^n[\frac{1}{8}]$ with a $c_2$-weight constraint (on $\mathbf{e} \cdot \mathbf{a}$).
HGJ step 2 and 1: merge and filter

\[ L_0^2, c_2 \]
\[ D^n[\frac{1}{8}] \]
\[ L_1^2, c_2 \]
\[ D^n[\frac{1}{8}] \]
\[ L_2^2, c_2 \]
\[ D^n[\frac{1}{8}] \]
\[ L_3^2, c_2 \]
\[ D^n[\frac{1}{8}] \]
\[ c_1 \]
\[ D^n[\frac{1}{8}] \times D^n[\frac{1}{8}] \]
\[ c_1 \]
\[ D^n[\frac{1}{8}] \times D^n[\frac{1}{8}] \]
\[ L_0^1, c_1 \]
\[ D^n[\frac{1}{4}] \]
\[ L_0^1, c_1 \]
\[ D^n[\frac{1}{4}] \]
\[ L_0^0, 1 \]
\[ D^n[\frac{1}{2}] \]
An optimization problem

We write all parameters in $\log_2$, relative to $n$:

$$
\begin{align*}
|D^{n/2}[\frac{1}{8}]| & \quad h(1/8)/2 \simeq 0.2718 \\
|L^j_i| & \quad \ell^j_i \\
|L_1||L_2|/(2^{\text{nd}}) & \quad \ell_1 + \ell_2 - d \\
|L| \times p & \quad \ell + pf
\end{align*}
$$

We compute the filtering probabilities:

- $D^n[0, \frac{1}{8}] \times D^n[0, \frac{1}{8}] \rightarrow D^n[0, \frac{1}{4}]$: $2^{-0.02585n}$
- $D^n[0, \frac{1}{4}] \times D^n[0, \frac{1}{4}] \rightarrow D^n[0, \frac{1}{2}]$: $2^{-0.12256n}$
Optimized parameters for HGJ

- $0.272 \begin{array}{c} 0 \end{array}$
- $0.272 \begin{array}{c} 0 \end{array}$
- $0.272 \begin{array}{c} 0 \end{array}$
- $0.272 \begin{array}{c} 0 \end{array}$
- $0.272 \begin{array}{c} 0 \end{array}$
- $0.272 \begin{array}{c} 0 \end{array}$
- $0.272 \begin{array}{c} 0 \end{array}$

- $0.294, 0.250 \begin{array}{c} D^n[\frac{1}{8}] \end{array}$
- $0.294, 0.250 \begin{array}{c} D^n[\frac{1}{8}] \end{array}$
- $0.294, 0.250 \begin{array}{c} D^n[\frac{1}{8}] \end{array}$
- $0.294, 0.250 \begin{array}{c} D^n[\frac{1}{8}] \end{array}$

- $0.337, 0.5 \begin{array}{c} D^n[\frac{1}{8}] \times D^n[\frac{1}{8}] \end{array}$
- $0.337, 0.5 \begin{array}{c} D^n[\frac{1}{8}] \times D^n[\frac{1}{8}] \end{array}$

- $0.311, 0.5 \begin{array}{c} D^n[\frac{1}{4}] \end{array}$
- $0.311, 0.5 \begin{array}{c} D^n[\frac{1}{4}] \end{array}$

- $0, 1 \begin{array}{c} D^n[\frac{1}{2}] \end{array}$

- $D^n[\frac{1}{2}]$
Better representations: BCJ

Sample distributions with \{-1, 0, 1\}.

- Of course we still need to obtain $D^n[\frac{1}{2}]$ in the end.
- The “-1” need to be canceled out by “1”.
- The “-1” shouldn’t sum up to “-2”!
- More parameters, new filtering probabilities.
- Improvement on the time exponent: 0.291 < 0.337.

New results

Idea 1: do not saturate the lists
Starting lists are not equal to $D^{n/2}[*]$ but sampled u.a.r. from it.

BCJ without saturation: $0.289 < 0.291$

Idea 2: still better representations
Why stop at “-1”? Add some “2”.
- Of course we still need to obtain $D^n[\frac{1}{2}]$ in the end
- Some “1” can sum up to “2” (but not too much)
- A “-1” and a “2” give a “1”

BCJ without saturation and with “2”: $0.283 < 0.289$

Bonnetain et al., *Improved Classical and Quantum Algorithms for Subset-Sum*, ePrint 2020/168
Going Quantum
Classical search

Let $X = G \cup B$.

- $X$: Search space, size $N$.
- $G$: Good ones, size $T$.
- $B$: Bad ones, size $N - T$.

Let $Sample$ and $Test$ be functions to sample $x$ from $X$ and test if $x \in G$, in time $t_{Sample}$ and $t_{Test}$.

There exists a function $Sample_G$ that samples from $G$ in time:

$$\frac{N}{T} (t_{Sample} + t_{Test})$$

⇒ we transform a sampling procedure for the “search space” into a sampling procedure for the “solution space”.

Xavier B., Rémi B., André S., Yixin S.  
Classical and Quantum Algorithms for Subset-Sum
Quantum search (amplitude amplification)

Let $QSample$ and $QTest$ be quantum algorithms to quantumly sample $X$ and quantumly test if $x \in G$, in time $t_{Sample}$ and $t_{Test}$.

There exists an algorithm $QSample_G$ that samples $G$ in time:

$$\sqrt{\frac{N}{T}} (t_{QSample} + t_{QTest})$$

Quantum test: means testing any $x \in X$ in superposition

Quantum sample: means producing a uniform superposition of $X$

Brassard et al., “Quantum amplitude amplification and estimation”, 2002
Classical search vs. quantum search

In the classical realm, we test elements $x$ at random until we have found (a random) $x \in G$. 
Classical search vs. quantum search

In the classical realm, we test elements $x$ at random until we have found (a random) $x \in G$. 

![Diagram showing a set of circles and squares, with one circle highlighted as the solution.]
Classical search vs. quantum search

In the classical realm, we test elements $x$ at random until we have found (a random) $x \in G.$
In the classical realm, we test elements $x$ at random until we have found (a random) $x \in G$. 
Classical search vs. quantum search

In the classical realm, we test elements $x$ at random until we have found (a random) $x \in G$. 
Classical search vs. quantum search

In the quantum realm, we move globally from $X$ to $G$. 
Classical search vs. quantum search

In the quantum realm, we move globally from $X$ to $G$. 

![Diagram showing classical and quantum search concepts]
Classical search vs. quantum search

In the quantum realm, we move globally from $X$ to $G$. 
Classical search vs. quantum search

In the quantum realm, we move globally from $X$ to $G$. 
## Interlude: quantum memory models

<table>
<thead>
<tr>
<th>What happens if $X$ is in memory?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classical sample:</strong> only reads a single $x \in X$ (easy)</td>
</tr>
<tr>
<td><strong>Quantum sample:</strong> must read all of $X$ in superposition (maybe not easy): this is quantum random access</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Quantum random access</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classical write</strong></td>
<td>Classical memory quantum random access</td>
</tr>
<tr>
<td></td>
<td>QRACM</td>
</tr>
<tr>
<td><strong>Quantum write</strong></td>
<td>Quantum memory quantum random access</td>
</tr>
<tr>
<td></td>
<td>QRAQM</td>
</tr>
</tbody>
</table>

⇒ This section

Previous quantum subset-sum algos
Quantum algorithms for subset-sum

The time is $\tilde{O} \left( 2^{\beta n} \right)$.

<table>
<thead>
<tr>
<th>Technique</th>
<th>$\beta$</th>
<th>Ref.</th>
<th>Classical version</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIM</td>
<td>0.3334</td>
<td>BHT98</td>
<td>HS74</td>
<td>QRACM</td>
</tr>
<tr>
<td>4-list merge</td>
<td>0.3</td>
<td>BJLM13</td>
<td>SS81</td>
<td>QRAQM</td>
</tr>
<tr>
<td>${0, 1}$</td>
<td>0.241</td>
<td>BJLM13</td>
<td>HGJ10</td>
<td>QRAQM + conj.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td></td>
<td>QRACM</td>
</tr>
<tr>
<td>${-1, 0, 1}$</td>
<td>0.226</td>
<td>HM18</td>
<td>BCJ11</td>
<td>QRAQM + conj.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td></td>
<td>QRAQM</td>
</tr>
<tr>
<td>${-1, 0, 1, 2}$</td>
<td>0.2156</td>
<td>Ours</td>
<td></td>
<td>QRAQM + conj.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td></td>
<td>QRAQM</td>
</tr>
</tbody>
</table>
Subset-Sum with Quantum Search
“Sampling-and-filtering”

Let’s separate:

- **The sampled list**
- **The intermediate list**

We turn samples from \( L_1 \) into:

- **Samples from \( L \):**
  \[
  t_{\text{Sample}}(L) = \max \left( \frac{2^{\text{nd}} |L_2|}{|L_2|}, 1 \right) t_{\text{Sample}}(L_1)
  \]
  (find an element with same modulus)

- **Samples from \( L^f \):**
  \[
  t_{\text{Sample}}(L^f) = \frac{1}{p} t_{\text{Sample}}(L)
  \]
  (wait until the filter is passed)
Quantum “sampling-and-filtering”

Assume that we have quantum samples from $L_1$.

Then we have:

- Quantum samples from $L$:
  $$t_{\text{QSample}}(L) = \max \left( \sqrt{\frac{2^{\text{nd}} |L_2|}}{1}, 1 \right) t_{\text{QSample}}(L_1)$$

- Quantum samples from $L^f$:
  $$t_{\text{Sample}}(L^f) = \sqrt{\frac{1}{p}} t_{\text{Sample}}(L)$$

List $L_1$
- size $|L_1|$  
- cons. $c$

List $L_2$
- size $|L_2|$  
- cons. $c$

List $L$
- size $|L| = |L_1||L_2|/(2^{\text{nd}})$  
- cons. $c + d$

List $L^f$
- size $|L^f| = |L| \times p$  
- cons. $c + d$
HGJ in the “sampling” framework
HJG in the “sampling” framework

- $L^0_0, 1$
- $D^n[\frac{1}{2}]$
- $L^1_0, c_1$
- $D^n[\frac{1}{4}]$
- $L^2_1, c_2$
- $D^n[\frac{1}{8}]$
- $L^3_2$
- $D^n[\frac{1}{8}]$
- $L^3_3$
- $D^n[\frac{1}{8}]$
- $L^3_4$
- $D^n[\frac{1}{8}]$
- $L^3_5$
- $D^n[\frac{1}{8}]$
- $L^3_6$
- $D^n[\frac{1}{8}]$
- $L^3_7$
- $D^n[\frac{1}{8}]$
HGJ in the “sampling” framework

\[ \begin{align*}
\mathcal{L}^3_0 & \quad \ldots \\
\mathcal{L}^3_1 & \quad \ldots \\
\mathcal{L}^3_2 & \quad \ldots \\
\mathcal{L}^3_3 & \quad \ldots \\
\mathcal{L}^3_4 & \quad \ldots \\
\mathcal{L}^3_5 & \quad \ldots \\
\mathcal{L}^3_6 & \quad \ldots \\
\mathcal{L}^3_7 & \quad \ldots \\
\mathcal{L}^2_0, \ c_2 & \quad \mathcal{D}^n[\frac{1}{8}] \\
\mathcal{L}^2_1, \ c_2 & \quad \mathcal{D}^n[\frac{1}{8}] \\
\mathcal{L}^1_0, \ c_1 & \quad \mathcal{D}^n[\frac{1}{4}] \\
\mathcal{L}^1_1, \ c_1 & \quad \mathcal{D}^n[\frac{1}{4}] \\
\mathcal{L}^0, 1 & \quad \mathcal{D}^n[\frac{1}{2}] \\
\end{align*} \]
HGJ in the “sampling” framework

$L^3_0, 1$
$L^3_1$
$L^3_2$
$L^3_3$
$L^3_4$
$L^3_5$
$L^3_6$
$L^3_7$

$L^2_0, c_2$
$L^2_1$
$L^2_2$
$L^2_3$
$L^2_4$
$L^2_5$
$L^2_6$
$L^2_7$

$L^1_0, c_1$
$L^1_1$
$L^1_2$
$L^1_3$

$L^0, 1$
Using quantum search

Quantum search will square-root the sampling time of $L^0$. But it’s useless if the intermediate lists cost the same as before.

$\Rightarrow$ we make the tree unbalanced.
Details and result

- Unbalanced left-right split of $L_0^3$ and $L_1^3$, unbalanced weights for the lists
- $L_1^3, L_2^2, L_1^1$ are intermediate lists stored in QRACM (classical data with quantum random access)
- only $\text{poly}(n)$ quantum storage needed

\[
0.226 \text{ (HM18)} < 0.2356 < 0.241 \text{ (BJLM13)}
\]

We use only $\{0, 1\}$ representations

We filter more efficiently
Subset-Sum with Quantum Walks
A classical walk for HGJ

Reduce the HGJ merging tree to a smaller tree, with smaller starting lists. Now $L^0$ does not always contain a solution.
Walking on the graph

We use a (regular, undirected) Johnson graph $J(D, L)$.

- A vertex contains a product of 8 small lists $L^3_i, 0 \leq i \leq 7$, of size $|L^3_i| = L$, chosen among distributions $|D^i| = D$, and the whole tree built from these lists.
- There are $\binom{D}{L}^8$ vertices.
- Some vertices are marked: they contain the knapsack solution.
- We go from one to another by changing an element in a list $L^3_i$ and updating the tree.
Classical random walk

We move to random neighbors until we find a marked vertex.
Classical random walk

We move to random neighbors until we find a marked vertex.
Classical random walk

We move to random neighbors until we find a marked vertex.
Classical random walk

We move to random neighbors until we find a marked vertex.
Cost of a classical random walk

We need procedures:

- To **setup** a starting arbitrary vertex (S)
- To **move** from one vertex to one of its neighbors (U)
- To **check** if a vertex is marked (trivial) (C)

We will find a marked vertex in time:

\[
S + \frac{1}{\epsilon} \left( \underbrace{\epsilon \text{ proportion of marked vertices}}_{\epsilon \text{ proportion of marked vertices}} \right) \left( \underbrace{\frac{1}{\delta}}_{\delta \text{ spectral gap of the graph}} \right) U + C
\]

where \( \frac{1}{\delta} \) is the number of updates before we reach a new uniformly random vertex. In a Johnson graph \( J(D, L) \), \( \frac{1}{\delta} \approx L \). (We need to replace all elements.)
Quantum walk

As in quantum search, the walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.
Quantum walk

As in quantum search, the walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.
Quantum walk

As in quantum search, the walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.
Quantum walk

As in quantum search, the walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.
The setup now requires to create a superposition over all vertices.

As in quantum search, we perform $\sqrt{\frac{1}{\epsilon}}$ steps instead of $\frac{1}{\epsilon}$.

But the mixing is also accelerated!

$$S + \sqrt{\frac{1}{\epsilon}} \left( \sqrt{\frac{1}{\delta}} U + C \right)$$

The Update handles all vertices and all edges in superposition.

Magniez et al., “Search via quantum walk”, SIAM 11
Tracking the updates

- The update $U$ must replace one element in a lower-level list and update the merging tree data structure.

- On average, there is a single replacement to make at each level (no problem classically).
Superposition updates

- The update needs to take a fixed time.
- Since we are handling all vertices and all edges in superposition, there are cases when updating the tree would cost an exponential time.

Can we abort the bad cases?

Not in the MNRS framework: the data structure (the tree) must depend only on the vertex (the initial lists).

The quantum walk conjecture of Helm and May (TQC18)

With an update of expected time $O(1)$, we can still do the runtime analysis as if it had an exact time $O(1)$.

---

Helm and May, “Subset Sum Quantumly in $1.17^n$”, TQC 18
New results

- We have modified the **data structure** to guarantee the update time.
- This reduces (marginally) the number of marked vertices.

**Fact**

*Previous quantum walks for subset-sum do not require the update conjecture.*

- However, this data structure is not enough for our best algorithms . . .
## Summary of quantum walk results

<table>
<thead>
<tr>
<th>Technique</th>
<th>Time</th>
<th>Ref.</th>
<th>Classical version</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0, 1}</td>
<td>0.241</td>
<td>BJLM13</td>
<td>HGJ10</td>
<td>QRAQM + conj.</td>
</tr>
<tr>
<td>{-1, 0, 1}</td>
<td>0.226</td>
<td>HM18</td>
<td>BCJ11</td>
<td>QRAQM + conj.</td>
</tr>
<tr>
<td>{-1, 0, 1, 2}</td>
<td>0.2156</td>
<td>Ours</td>
<td></td>
<td>QRAQM + conj.</td>
</tr>
<tr>
<td>{-1, 0, 1, 2}</td>
<td>0.2182</td>
<td>Ours</td>
<td></td>
<td>QRAQM</td>
</tr>
</tbody>
</table>
Conclusion and open questions

On classical algorithms
More symbols and nearest-neighbor techniques should improve the exponent ⇒ how far could we go?

On quantum algorithms with quantum search
Better representations should improve the exponent … if we manage to make the optimization converge.
Conclusion and open questions (ctd.)

On quantum walks

- The update conjecture can be removed from previous works
  ...but not completely from ours
- It seems that we still need to adapt the MNRS Quantum Walk framework

ePrint 2020/168
Thank you!