Quantum Adiabatic Optimization Without Heuristics

Michael Jarret\textsuperscript{a}, Brad Lackey\textsuperscript{a,c}, Aike Liu\textsuperscript{a}, Kianna Wan\textsuperscript{a} [kianna@stanford.edu],
\textsuperscript{a}Perimeter Institute for Theoretical Physics, \textsuperscript{b}Microsoft Quantum, \textsuperscript{c}University of Maryland, \textsuperscript{d}University of Illinois Urbana-Champaign, \textsuperscript{e}Stanford University

Summary

(1) we present a general algorithm for QAO for any Hamiltonian $H(s)$, assuming access to an oracle that estimates the spectral gap of $H(s)$
(2) we explicitly construct an efficient gap oracle for a nontrivial class of optimization problems, demonstrating that given knowledge only of $H(0)$, QAO can be reliably performed in at least some circumstances

Motivation

• can the spectral gap be estimated on the fly?
• is it possible to adaptively determine near-optimal annealing schedules, by using estimates of the gap?

Problem

given $W : V \to \mathbb{R}_{\geq 0}$, find $m \approx \arg\min_{v \in V} W(v)$

QAO can in principle be used: initialize a system in a known state and adiabatically evolve to the ground state of $W = \text{diag}(W(v_1), W(v_2), \ldots, W(v_{|V|}))$

example: Grover’s search problem

$\implies W = W^{(G)} = I - |m\rangle\langle m| = \text{diag}(0, 1, 1, \ldots)$

evolve for a time $T$ under the Hamiltonian

$H(t) = (1 - s(t))L + s(t)W,$

where $L \gg I - |\psi\rangle\langle \psi|$, and $s(0) = 0, s(T) = 1$

Roland and Cerf\textsuperscript{1}:  

• if $s(t) = t/T$ is used, require $T = O(|V|)$

• but by analyzing the spectral gap $\gamma(s)$, can find an optimal schedule $s(t)$, for which $T = O(1)$

query complexity:

define the sublevel sets $I_k = \{s \in [0, 1] : \Gamma(s) \leq \lambda_{\text{max}}/2^k\}$

if $\mu(I_k) \leq C/2^k$ for some constant $C$ and each $I_k$ can be written as the union of at most $R$ disjoint intervals, then QAO requires at most $O(R \log(\lambda_{\text{max}}/\Gamma_{\text{min}}))$ queries to $\Gamma(s)$

in general, however, not much is known about $\gamma(s)$ a priori, so we cannot determine the optimal schedule in advance

instead, we propose a discretized procedure which, by dynamically learning the spectral gap, efficiently performs QAO

Bashful Adiabatic Algorithm

given

• the ground state and spectral gap $\gamma(0)$ of $H(0)$

• an oracle $\Gamma$ that returns an approximation of the spectral gap $\gamma(s)$ of $H(s)$.

BAA returns the ground state of $H(1)$ within precision $\epsilon$ in time $O(\min(\gamma^{-1} , \epsilon^{-1}))$ using $O(\log(\gamma^{-1}))$ queries to $\Gamma$

basic outline of algorithm:

starting at $s = 0$ (with $\gamma(0)$ known), at each step $s_i < 1$:

1. query $\Gamma$ to estimate $\gamma(s_i)$

2. use Weyl's inequality to find $s_{i+1}$ such that

$|\gamma(s_{i+1}) - \gamma(s_i)| \leq \gamma(s_i)/2$

3. determine the evolution time $T_i$ over the interval $(s_i, s_{i+1})$

in accordance with the adiabatic theorem\textsuperscript{2}

visual example: behaviour of BAA on Grover’s problem

Oracle Construction

as proof of concept, we apply BAA to a specific architecture

$H(s) = \gamma(s)L + sW$,

where $L$ is the Laplacian of the complete graph on $V$ vertices, and $W$ encodes an unknown cost function $W$ with $W(m) = 0$

we provide an efficient algorithm for the gap “oracle” $\Gamma$, that can be called to estimate $\gamma(s)$ at any point $s$

note that even in the less general case where $W \propto W^{(G)}$, the gap minimum is hard to locate ahead of time:

our oracle and its complexity analysis involve the Cheeger ratio $h(m)^{-1}$ for ground state $|\psi\rangle = \sum \phi_v |v\rangle$

$h(m)^{-1} \approx \inf_{\text{min} \{\phi_v, 1-\phi_v\} \leq \min h(m)}$

we prove that for all $s \in [0, 1]$

$2\gamma(s) \leq \gamma \geq h(m)^{-1}$

where $\lambda_{\text{max}}$ is the spectral ratio of $W$ (not necessarily known beforehand), and use this as a starting point for defining $\Gamma$

accurate approximations of $h(m)^{-1}$ can be obtained using computational basis measurements and classical root-finding

using our algorithm for $\Gamma$, BAA returns $m$ with constant probability in time $T = \tilde{O}(\sqrt{\gamma} + [(\gamma - 1)\gamma]^{1/2})$

in particular, when the input is a Grover-type problem ($\gamma = 1$), this achieves scaling $T = \tilde{O}(\sqrt{\gamma})$ without needing to know beforehand that the problem is indeed Grover search!

\footnotesize

**Motivation & Problem:** When writing code, programmers spend majority of time debugging. Can we use machine learning to automatically repair programs from errors?

**Our findings:**
1) Error message (feedback) is a crucial signal for learning repair
2) **Program-feedback graph:** a joint graph of code and error message helps model the reasoning of repair (e.g. tracking the variable that caused the error).
3) **Self-supervised learning:** freely-available, unlabeled programs (e.g. github, codeforces) can be turned into useful training examples of program repair

**Performance of our system:** State-of-the-art on two benchmarks
- DeepFix\(^1\): fix C coding assignments
- SPoC\(^2\): fix C++ program synthesis

---

Automatically Neutralizing Subjective Bias in Text
Reid Pryzant, Richard Martinez, Nathan Dass, Sadao Kurohashi, Dan Jurafsky, Diyi Yang

**Biased!**

**John McCain Exposed As An Agent Of The Rothschilds**

**Neutralized!**

Mccain Accused of Improper Donations from the Rothschilds
Reliability and Security Challenges in Modern Computing Systems

- Modern computing systems are concurrent and heterogeneous.
- High degree of concurrency and heterogeneity lead to large challenge problems for ensuring correctness of event orderings and interferences.
- Event interferences form the cornerstone of reliability and security.
- Reliability. Memory consistency models (MCMs) are one notable design feature that rely heavily on event orderings to guarantee correctness for programs that rely on an implementation.
- Security. As it turns out, a significant portion of hardware security vulnerabilities lead to violations of confidentiality and integrity in modern computing systems.
- My dissertation work addresses reliability and security via interference specification design and verification against interfaces.
- Key features of my approach formal methods, formal specifications, early-stage verification, microarchitectural "happens-before" (mhb) graphs.

TriCheck Memory Consistency Model Verification at the Trisection of Software, Hardware, and ISA

An analysis methodology and associated tool for verifying that a high-level-language (HLL) instruction set architecture (ISA) and microarchitectural implementation all align well on MCM requirements.

Why Full-Stack MCM Verification?

- What can go wrong?
  - Instruction set architecture (ISA)
  - Microarchitectural implementation
- Why is it required?
  - To ensure correct implementation of MCM
- Why is it necessary?
  - To verify the implementation against the specification

Evaluation of the Draft RISC-V ISA Specification

- RISC-V Base vs. Atomic Lack of Cumulative Fences
- Relaxed memory model
  - Interthread comm & sync, via shared memory
  - FENCE required to guarantee any ordered FENCE instructions or any combination of reads and writes (e.g., FENCE R/W or FENCE R/W R/W)
- Non-atomic (NCA) stores
  - Read-modify-write (RMW) instructions
  - Fetch-and-atomic instructions (AMO)
- Load-Reserve-Store-Conditional (LRS)
- Support for release consistency and C1 MCM
- Eg: Acquire (e.g., AMO urls) Release (e.g., AMO urls)

Impact and Future Directions

- My work produced the first tool, TriCheck, to verify a full software/hardware stack from HLL to microarchitecture.
  - Demonstrated that software and hardware MCM verification in solution can miss bugs
  - High-speed, interactive, early-stage analysis/verification
  - Clearly how to mitigate bugs.
- Concrete examples of shortcomings in the RISC-V MCM which led to a redesign and much-improved formal method.
- Demonstration of the value of SHTP and RMW techniques for hardware-aware analysis.
- Observation that the issues present in MCM analysis are similar to those in security analysis
- Nearly seamlessly transition MCM verification into security verification space.
- Principled alternatives to the ad hoc "bizarre" analysis that has long dominated the security field.

Microarchitecturally Happens-Before (mhb) Analysis

Modeling Hardware-Specific Program Executions as mhb Graphs

- Original mhb graphs: microarchitectural events in a program execution
- mhb edge: "happens-before" relationship between nodes
- Augmenting for security: mhb graph with execution pattern; mhb sub-graph

Evaluating "Observability" of Microarchitectural Executions

- Initial insight: Can we identify mhb patterns that can be used by software to observe microarchitectural events?
- Microarchitectural specification
  - Hardware-aware (HAWA) MCM implementation
- Keys to approach: Cycle Counting + Genetic Pattern Checking

CheckMate: Automated Exploit Program Generation for Hardware Security Verification

An approach and automated tool for determining if a microarchitecture is susceptible to specified classes of security exploits and for synthesizing proof-of-concept exploit code when it is.

Formalizing Exploit Patterns

- Mappings Cache Occupancy with the Value in Cache Lifetime (VCL) Abstraction
  - Writes have VCL, Create & Expire nodes
  - VCL abstraction is partially associated with the value in a cache line
  - Read misses have VCL, Create & Expire nodes
  - Going from unusable state to usable state
  - Back link do not have VCL, Create & Expire nodes
  - Going from unusable state to usable state

Spectrum: Prime & Probe

- Pattern: Two consecutive same-address accesses with an "interleaving" flush, but to VCL, Create/Expire nodes for the second access.

Synthesizing Real-World Attacks

Exploit-specific cache-line attack pattern

Sample CheckMate Runtimes

- Attack In: Flush-Reload attack
  - Primehole
    - Exploit specific cache-line attack pattern
  - SpectrePrime
    - Exploit specific cache-line attack pattern
- CheckMate
  - 2 new attacks: RedEyePrime, SpectrePrime

Concurrence and Security Verification in Heterogeneous Parallel Systems
Caroline Trippe
Stanford University