ELECTRICAL
& COMPUTER
ENGINEERING

Benchmarking (Some) Optimization Hardware and Algorithms Using 3-Regular 3-XORSAT

Tameem Albash @ University of New Mexico

19/20 July 2021

Based on arXiv:2103.08464 with
M. Kowalsky, I. Hen, and D. A. Lidar

CENTER FOR QUANTUM
INFORMATION & CONTROL

Motivation

The “end of Moore’s law” has motivated a search for new approaches for tackling hard computational problems

We will focus on NP-optimization problems:
Given an efficiently computable cost function $C(x)$ defined on N-bits $x \in \{0,1\}^N$, find the minimizing configuration x_*

$$x_* = \arg\min_x \; C(x)$$

ex. Quadratic Unconstrained Binary Optimization (QUBO)

$$C(x) = \sum_{i,j} Q_{ij} x_i x_j$$

ex. Polynomial Unconstrained Binary Optimization (PUBO)
(Sometimes called a higher-order unconstrained binary optimization)

$$C(x) = \sum_{m=1}^{p} a_m \prod_{i \in \mathscr{M}_m} x_i$$

Motivation

The “end of Moore’s law” has motivated a search for new approaches for tackling hard computational problems

We will focus on NP-optimization problems:
Given an efficiently computable energy function $E(s)$ defined on N Ising spins $s \in \{-1,1\}^N$, find the ground state s_*

$$s_* = \arg\min_s \; E(s)$$

ex. Ising (2-body) Hamiltonians

$$E(s) = \sum_{i,j} J_{ij} s_i s_j + \sum_i h_i s_i$$

ex. Generalized Ising (p-body) Hamiltonians

$$E(s) = \sum_{i_1, i_2, \ldots, i_p} J_{i_1 i_2 \ldots i_p} s_{i_1} s_{i_2} \ldots s_{i_p} + \sum_{i_1, i_2, \ldots, i_{p-1}} J_{i_1 i_2 \ldots i_{p-1}} s_{i_1} s_{i_2} \ldots s_{i_{p-1}} + \ldots + \sum_i h_i s_i$$

Motivation

For many of these problems, it is not expected for there to exist an efficient algorithm; we expect a runtime that scales exponentially with the problem size:

$$\tau \sim 10^{\alpha N+\beta}$$

The objective is to reduce the constant overhead (the factor β), and/or reduce the coefficient of the exponential scaling (the factor α)

This includes alternative paradigms of computing, like quantum computing, but also dedicated classical hardware for tackling specific problems

We should not expect a single device/implementation to be advantageous for all problems. Ideally, we would like a suite of benchmark problems that simulate different different aspects of real world problems

Today we consider only one problem class

3-regular 3-XORSAT (3R3X)

System of N linear equations modulo 2 for N Boolean variables x:

$$(x_{i_1} + x_{i_2} + x_{i_3}) \text{ mod } 2 = b_i \leftrightarrow x_{i_1} \oplus x_{i_2} \oplus x_{i_3} = b_i$$

$$x_i, b_i \in \{0,1\}$$

such that each Boolean variable appears in **exactly** 3 equations (3-regular)
Without loss of generality, we will take $b_i = 0$.
We assume the N equations are independent

Our aim is to find the solution to this system of equations

We can cast the problem in terms of minimizing a cost function in terms of Ising spin variables $s = 1 - 2x \in \{-1,1\}^N$

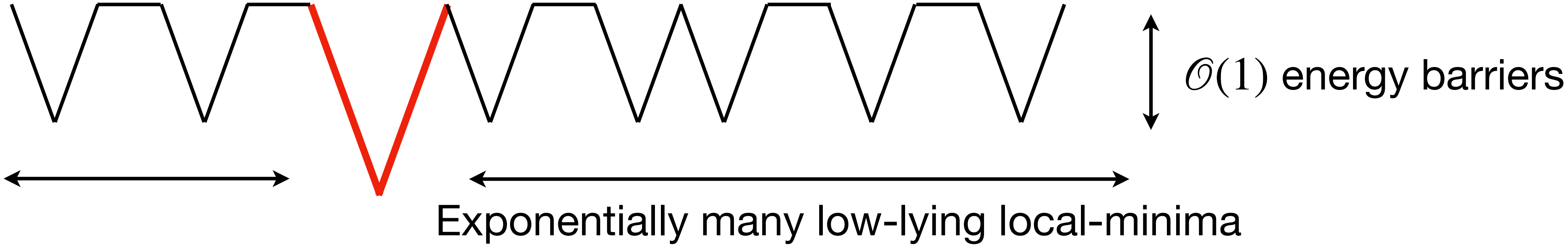
$$E(s) = -\sum_i s_{i_1} s_{i_2} s_{i_3}$$

All-zero (all-up spin) state is guaranteed to be an optimal configuration with $E(s_*) = -N$.

Complexity of 3R3X

The system of linear equations can be solved efficiently (in a time that scales as $\mathrm{poly}(N)$), however solving the optimization problem is known to take exponential time for heuristic algorithms [1,2]

This classical hardness arises from the presence of ‘entropic barriers’ and not energy barriers [3].


$\mathcal{O}(1)$ energy barriers
Exponentially many low-lying local-minima

Starting from a random state, it is exponentially unlikely that you will reach the global optimum by steepest descent. “Golf-course” low-energy landscape means searching the low energy landscape also takes exponential time.

[1] S. Franz, et al.,” Europhys. Lett. 55, 465 (2001).
[2] F. Ricci-Tersenghi, Science 330, 1639–1640 (2010).
[3] Matteo Bellitti, et al., arXiv:2102.00182

The Good and the Bad of 3R3X

Why we like these instances

- Can be constructed with at least one known global optimum solution.
- Efficient classical algorithm can be used to determine degeneracy of the ground state.
- Degree of each spin is fixed.
- Exponential hardness for heuristic algorithms is manifest even at relatively small problem sizes.

Why we don’t like these instances

- Interaction graph is random, so hardware with a fixed connectivity will not natively run the problem.
- Entropic barriers mean that fast repeated runs are more likely a better strategy than algorithms that try to perform advanced moves to overcome energy barriers.
- Energy landscape is such that approximate solutions are easy to find. Therefore, this may not be a meaningful benchmark for practical world problems.

What you learn from benchmarking using a single problem class will be limited.
A suite of problem classes are needed to identify the pros and cons of each hardware/algorithm implementation

Measuring Performance

We quantify the computational cost of solving an instance of 3R3X using the time-to-solution (TTS) metric [4]

Given a heuristic algorithm with a single-run runtime of t_f and a probability $p_i(t_f)$ of finding the solution of the i-th instance, the total time required to find the solution at least once with 0.99 probability is given by:

$$\mathrm{TTS}_i(t_f) = t_f \boxed{\frac{\log(1-0.99)}{\log(1-p_i(t_f))} = R(t_f)\text{, number of repetitions of the algorithm}}$$

For an ensemble of instances, we can consider different quantiles q of the TTS metric for a fixed runtime:

$$\langle \mathrm{TTS}(t_f) \rangle_{q=0.5} = \mathrm{median}\left[\mathrm{TTS}_i(t_f)\right]$$

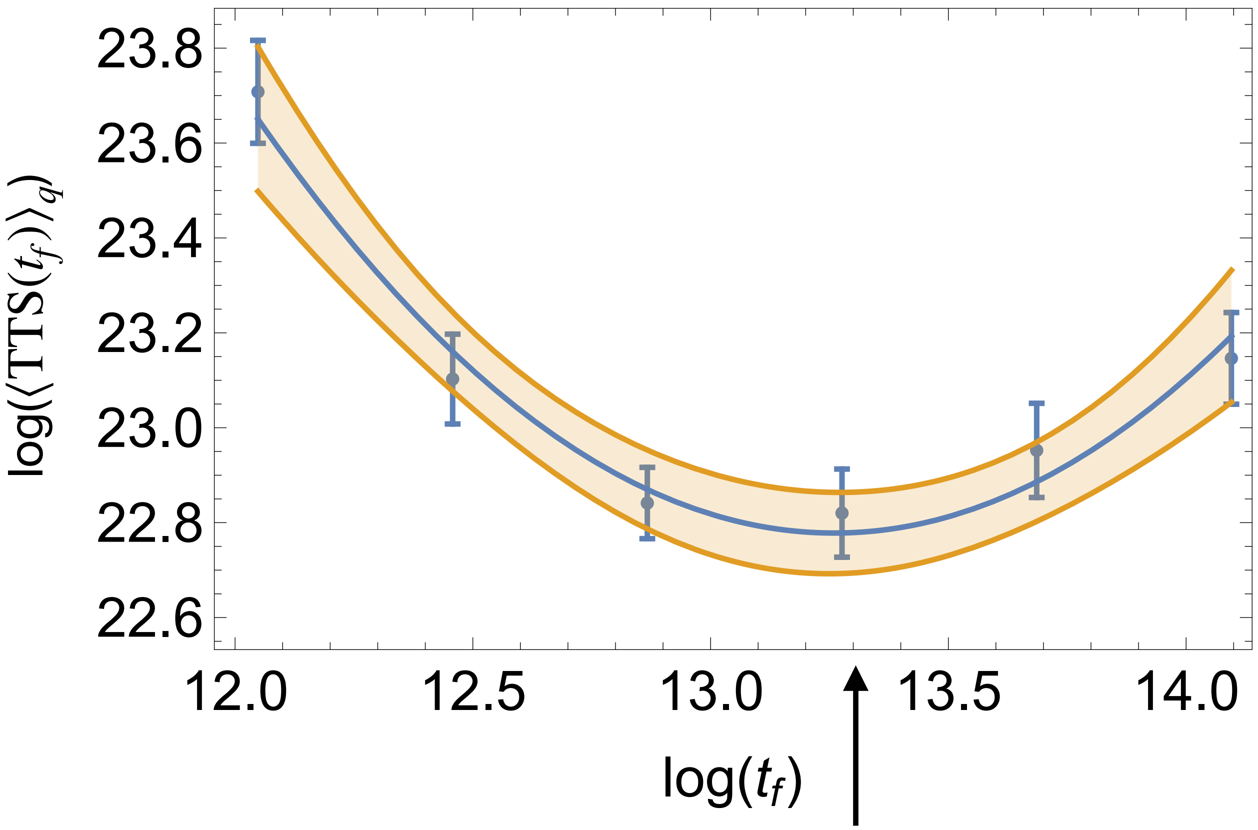
[4] T. Rønnow, et al. Science 345, 420–424 (2014).

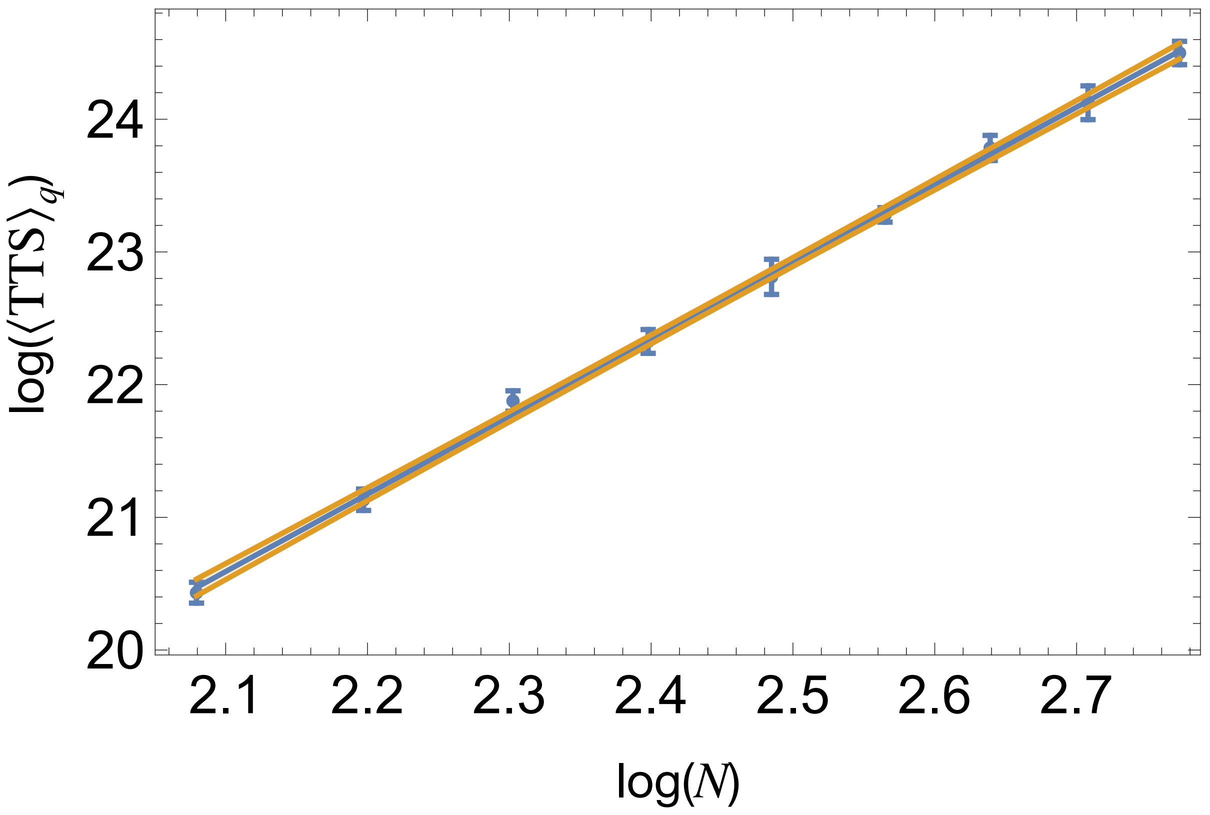
Optimal TTS

Choose the runtime t_f (and all other simulation parameters) that minimizes the particular quantile of TTS

$$\langle \mathrm{TTS} \rangle_{q=0.5} = \min_{t_f} \langle \mathrm{TTS}(t_f) \rangle_{q=0.5}$$

The optimum TTS is a balance between running many short repetitions/trials of the algorithm and running one long repetition of the algorithm.



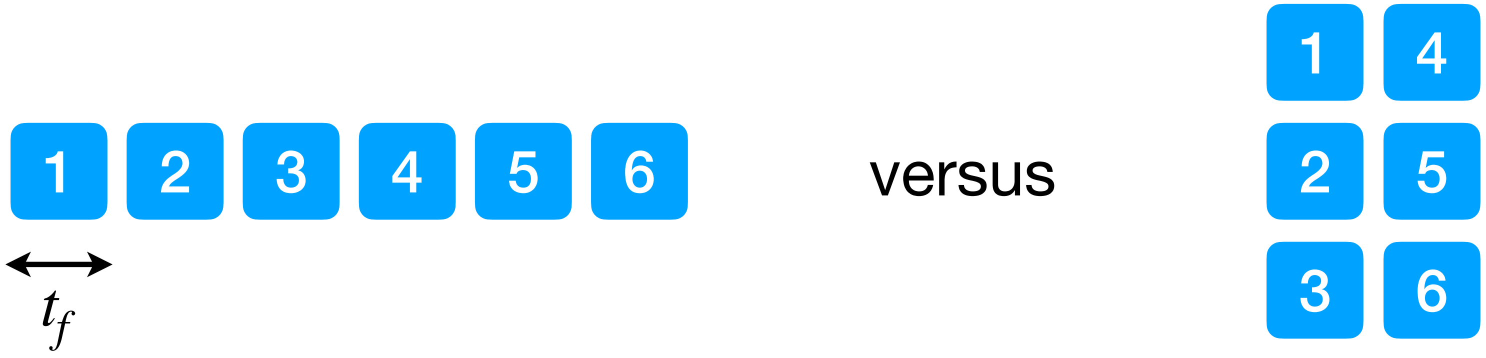


t_f value that minimizes TTS

Parallelization

The TTS metric assumes the independent runs of the algorithm are performed serially.

Independent repetitions can in principle be performed in parallel, i.e. an algorithm implemented efficiently on a GPU can use the multi-thread capability to run independent trials of the algorithm



$$\mathrm{TTS} = 6t_f \qquad\qquad \mathrm{TTS} = 2t_f$$

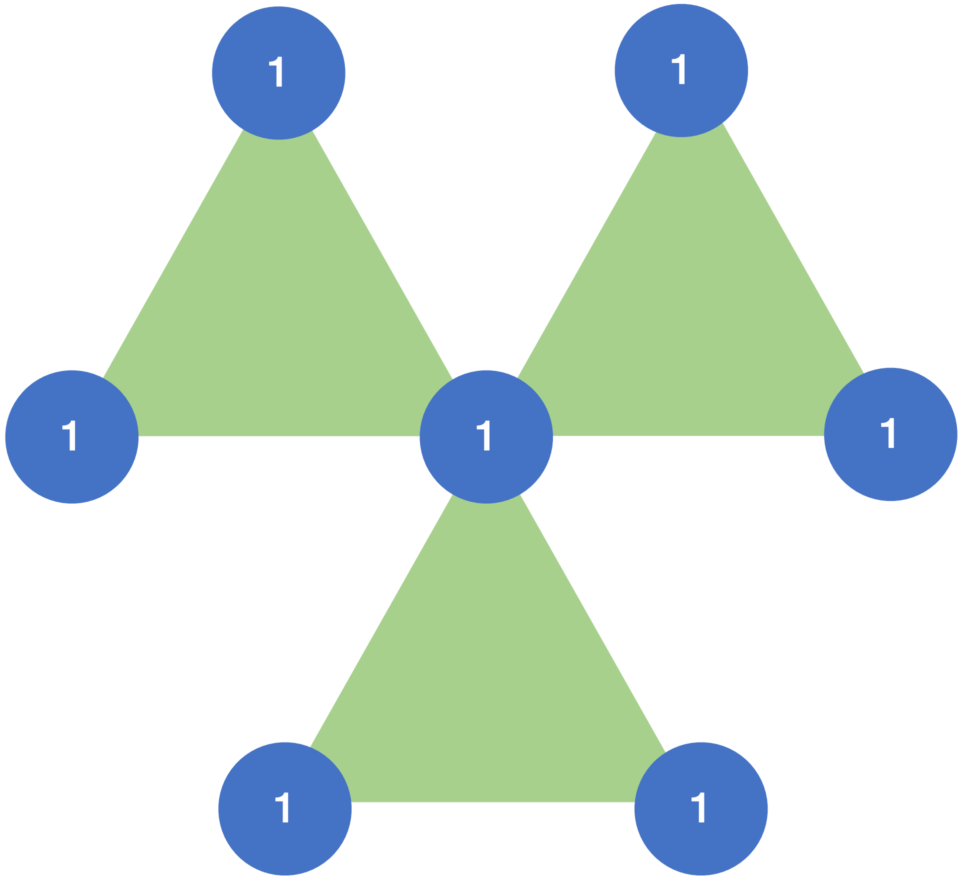
We can include this parallelization factor in our TTS calculation by dividing by a factor $f(N)$ that accounts for this parallelization. This is a finite-size correction to the TTS since all hardware has limitations (e.g., finite memory), so we expect the degree of parallelism to decrease or stay constant with increasing system size.

Quasi-greedy Algorithm

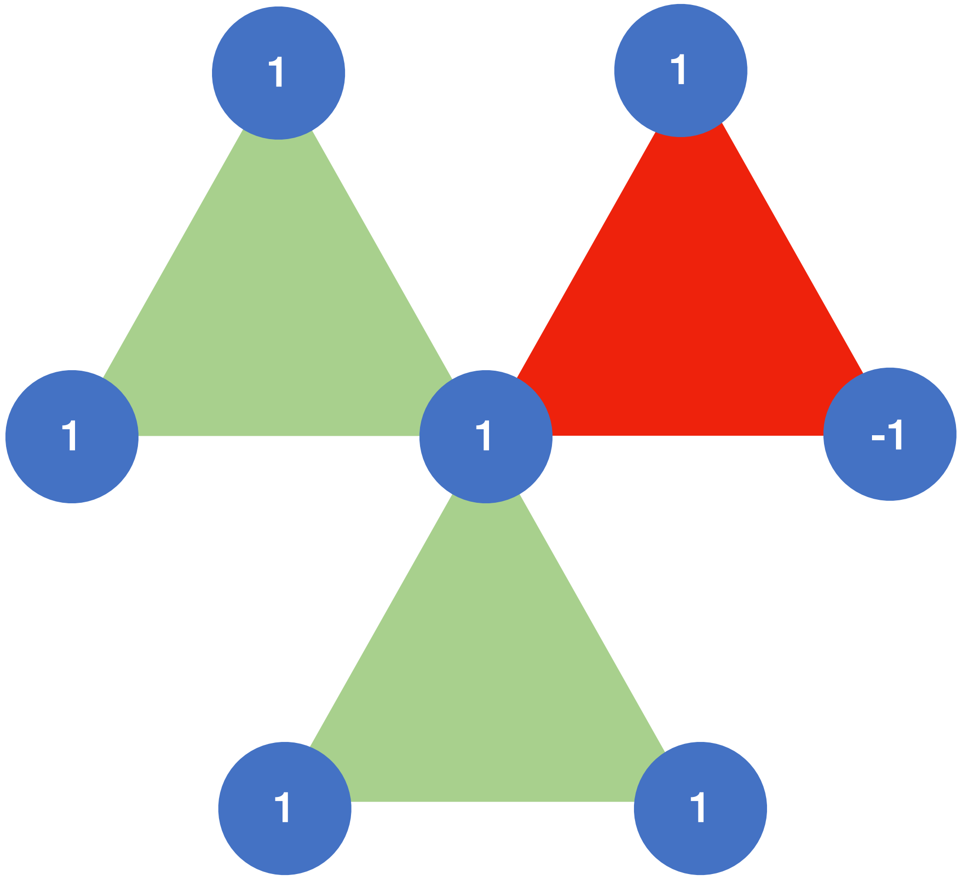
Quasi-greedy algorithm [3] is characterized by the choice of $\vec{w} = \left(w_0, w_1, w_2, w_3\right)$

- A spin is of type $k \in \{0,1,2,3\}$ if it violates k interaction terms.

$$E(s) = -\sum_i s_{i_1}s_{i_2}s_{i_3}$$



$k = 0$



$k = 1$

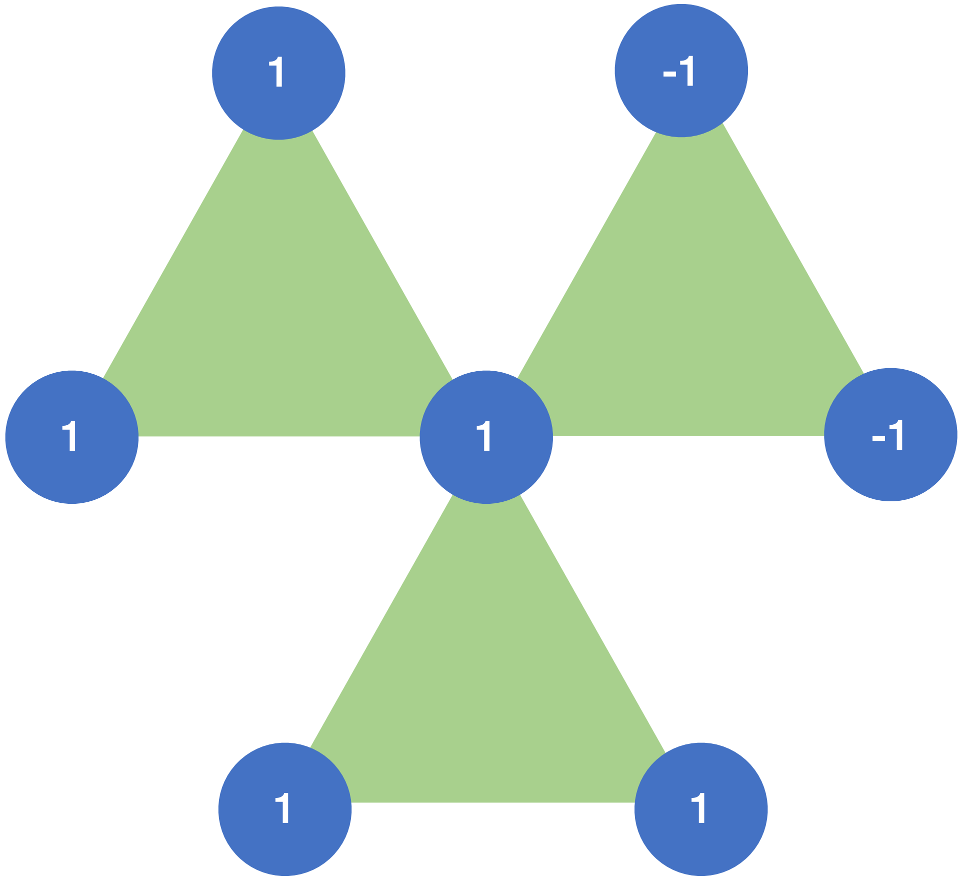
[3] M. Bellitti, et al., arXiv:2102.00182

Quasi-greedy Algorithm

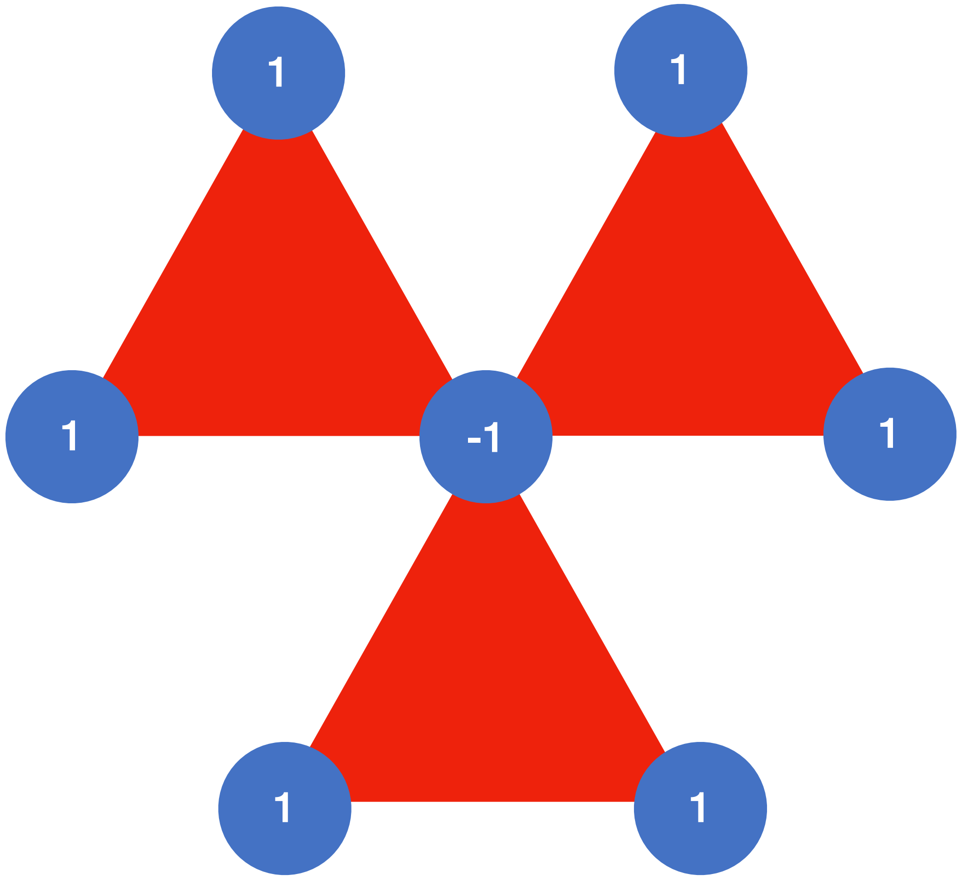
Quasi-greedy algorithm [3] is characterized by the choice of $\overrightarrow{w} = \left(w_0, w_1, w_2, w_3\right)$

- A spin is of type $k \in \{0,1,2,3\}$ if it belongs to k unsatisfied interactions.

$$E(s) = -\sum_i s_{i_1}s_{i_2}s_{i_3}$$



$k = 0$



$k = 3$

Flipping a spin of type k gives:

$k = 0 \rightarrow k = 3$ $\qquad$ $k = 2 \rightarrow k = 1$

$k = 1 \rightarrow k = 2$ $\qquad$ $k = 3 \rightarrow k = 0$

[3] M. Bellitti, et al., arXiv:2102.00182

Quasi-greedy Algorithm

- At time step t, the fraction of variables that are of type k is given by $f_k(t)$.
- At each time step, we choose a variable of type k with probability $p_k(t) \propto w_k f_k(t)$ and flip it.
- $w_0 = 0$ guarantees that if the ground state is reached, then the algorithm stops.
- Greedy algorithm (only flip variables that result in a lower energy) corresponds to $\overrightarrow{w} = (0,0,1,1)$

For 3R3X, an optimal choice is close to $\overrightarrow{w} = \left(0,0.054,1,1\right)$ [5]. The small value of w_1 means that the energy only increases when there are no more energy-decreasing spin-flip-updates.

Highly optimized implementation of the algorithm on GPUs (SATonGPU) [5] achieves a scaling with system size of

$$\langle \mathrm{TTS} \rangle_{q=0.5} \sim 10^{0.034N}$$

[5] M. Bernaschi, et al., EPL, 133 60005 (2021)

Quantum Adiabatic Optimization

The quantum analogue [6] to simulated annealing [7]

$$H(t) = -\left(1 - \frac{t}{t_f}\right)\sum_i \sigma_i^x - \frac{t}{t_f}\sum_i \sigma_{i_1}^z\sigma_{i_2}^z\sigma_{i_3}^z$$

- At $t = 0$, system is in the ground state of $H(0)$.
- If system is evolved in accordance with the adiabatic condition

$$t_f \gg \min_{t\in[0,t_f]} \Delta(t)^{-2}$$

then the adiabatic theorem guarantees that the ground state of 3R3X will be reached with high probability. Algorithm performance will depend on the scaling of the minimum gap along the interpolation.

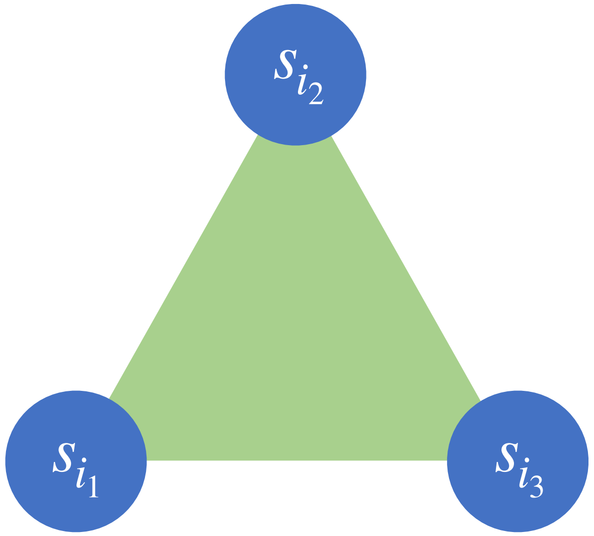
- Could be quadratically improved to $t_f \gg \min_{t\in[0,t_f]} \Delta(t)^{-1}$ using a tailored schedule [8].

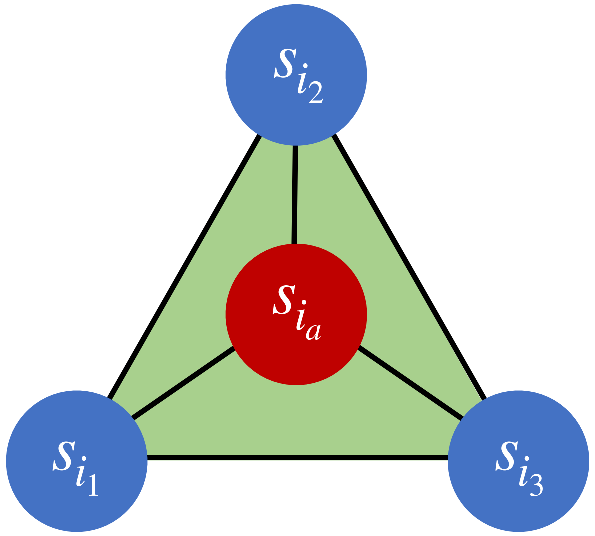
Estimate of the gap scaling [9], $\min_{t\in[0,t_f]} \Delta(t) \sim 10^{0.0348N}$, suggests no quantum speedup for this quantum algorithm on this problem class.

[6] T. Kadowaki and H. Nishimori, Phys. Rev. E 58, 5355–5363 (1998).
[7] S. Kirkpatrick, et al., Science 220, 671–680 (1983).
[8] J. Roland and N. J. Cerf, Phys. Rev. A 65, 042308 (2002).
[9] E. Farhi, et al., Phys. Rev. A 86, 052334 (2012).

3-body to $\leq$ 2-body

Dedicated hardware often does not natively implement 3-body interactions; we can use a gadget with one additional spin that uses 1 and 2-body interactions such that the minimizing spin configurations of a 3-body term are encoded into the ground states of the gadget [10]





$$-s_{i_1}s_{i_2}s_{i_3}$$

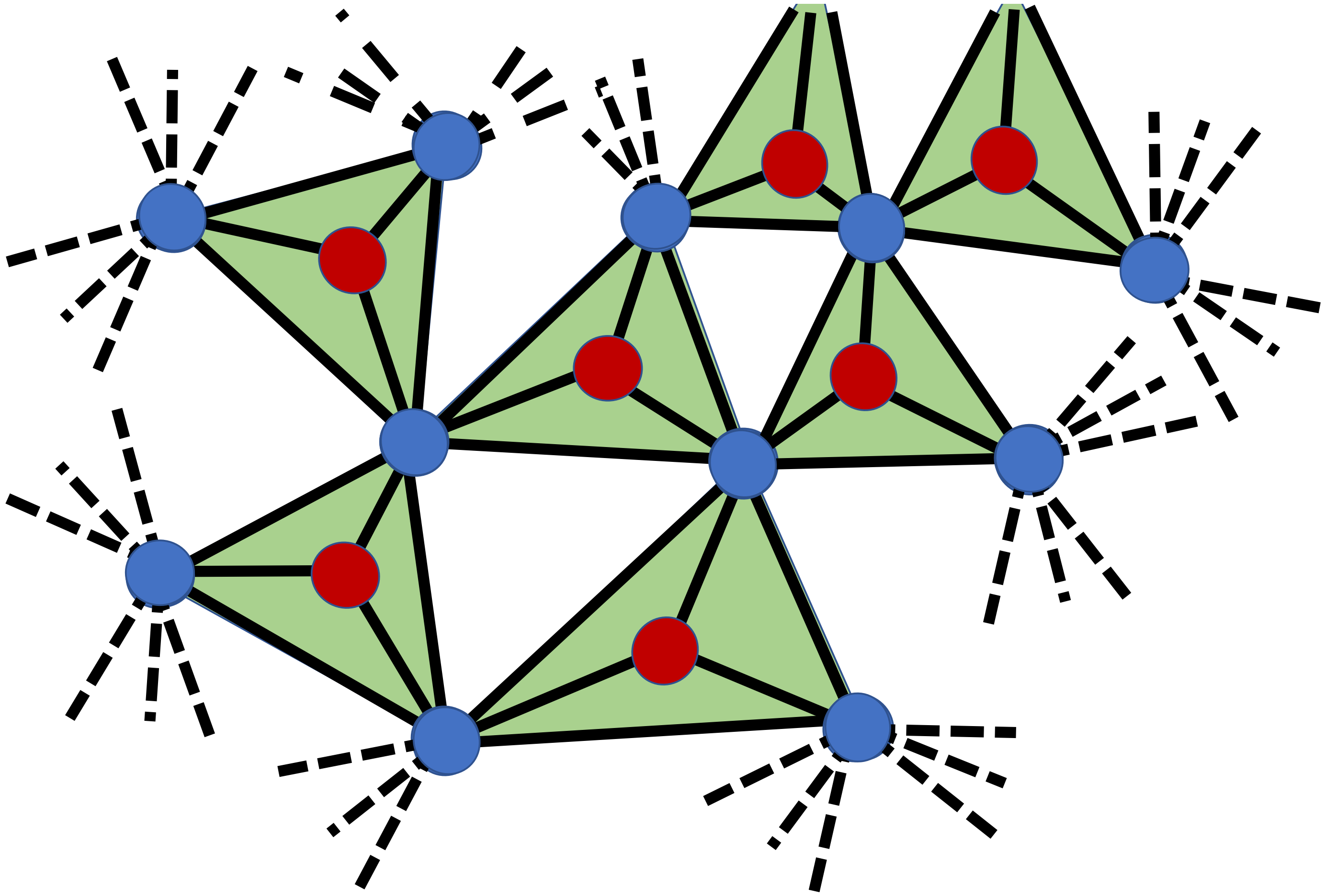
$$-\left(s_{i_1}+s_{i_2}+s_{i_3}\right)-2s_{i_a}$$
$$+\left(s_{i_1}s_{i_2}+s_{i_2}s_{i_3}+s_{i_3}s_{i_1}\right)+2s_{i_a}\left(s_{i_1}+s_{i_2}+s_{i_3}\right)$$

Ground states $(s_{i_1}, s_{i_2}, s_{i_3}) : (1,1,1),\ (1,-1,-1),\ (-1,1,-1),\ (-1,-1,1)$

[10] I. Hen, Phys. Rev. Applied 12, 011003 (2019)

Our 2-local Problem Instances

- We generate 100 random 3-regular 3-XORSAT instances with N variables and a unique solution.
- Each 3-body term is reduced in locality using the gadget; the resulting 2-local instances are defined with $n = 2N$ variables.

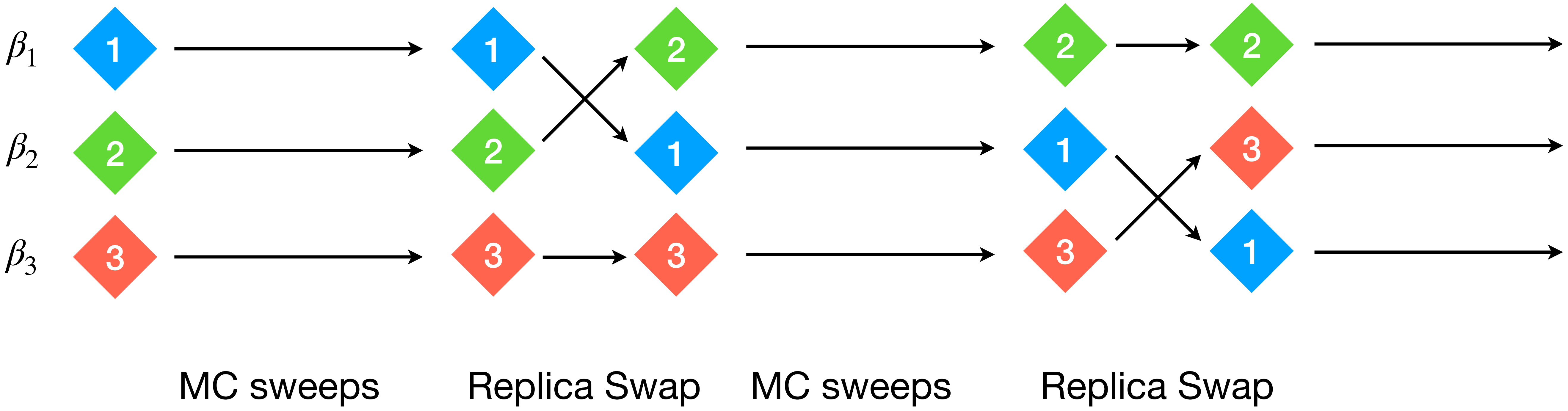


For each problem size, we identify the success probability of each instance for each solver, calculate the TTS, and identify the optimal TTS.

Parallel Tempering (PT)

Algorithm has two parts [11,12,13]

- N_β independent Markov-chain Monte Carlo simulations (replicas) evolved for N_{sw} sweeps. Each replica is associated with an inverse-temperature from $\{\beta_i\}_{i=1}^{N_\beta}$
- Spin configurations of neighboring temperature replicas are exchanged (satisfying detailed balance)



- The lowest energy configuration is tracked throughout this process.

[11] R. H. Swendsen and J.-S. Wang, Phys. Rev. Lett. 57, 2607–2609 (1986).
[12] C. J. Geyer, in Computing Science and Statistics Proceedings of the 23rd Symposium on the Interface (1991)
[13] K. Hukushima and K. Nemoto, Journal of the Physical Society of Japan 65, 1604–1608 (1996).

Temperature Optimization

The number and distribution of temperatures in parallel tempering should be optimized for best performance. The choice needs to ensure enough of an overlap between the energy histograms of neighboring inverse-temperature replicas. Two approaches:

- Best performance: Pick distribution to maximize rate of roundtrips performed by each replica [14]. Computationally costly since it requires replicas to perform several round trips to work, which makes it not very practical when the round trip time is very long.

- Most commonly used: pick distribution such that the probability to swap neighboring replicas is constant (pick swap probability to be ≈ 0.23 [15])

Performance difference between the two approaches varies between different class of problem instances [16]: sometimes no difference, sometimes only overhead difference, sometimes scaling difference.

A parallel tempering simulation with a constant swap probability of 0.23 should achieve the same scaling as the quasi-greedy algorithm albeit with a higher overhead [3]

[3] M. Bellitti, et al., arXiv:2102.00182.

[14] H. G. Katzgraber, et al., Stat. Mech. P03018 (2006)

[15] A. Kone and D. A. Kofke, J. Chem. Phys. 122, 206101 (2005).

[16] I. Rozada, et al., Phys. Rev. E 100, 043311 (2019)

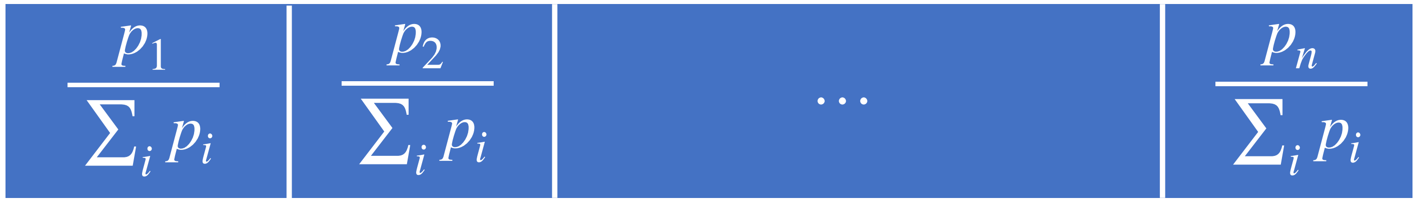
Fujitsu’s Digital Annealer Unit (DAU)

- ASIC hardware implementing parallel tempering.
- Tested hardware allows for a total of 8192 spins with programmable local fields and all-to-all Ising couplings.
- Limited form of parallelization: 8, fully-connected 1024 spins can be run in parallel.
- The temperature distribution is dynamically adjusted during the first 10^5 Monte Carlo updates
- Implements a form of “Rejection-free” Monte Carlo updates [17]

All spins calculate their Metropolis update probability in parallel

$$p_i = \min(1, e^{-\beta \Delta_i})$$
$$i = 1, \dots, n$$

“Roulette wheel selection”
One spin is randomly selected to be flipped according to its relative Metropolis update probability



$$\frac{p_1}{\sum_i p_i} \quad \frac{p_2}{\sum_i p_i} \quad \cdots \quad \frac{p_n}{\sum_i p_i}$$

If properly optimized, we expect the DAU to achieve a similar scaling to optimized PT, although the efficient hardware implementation should mean a significantly reduced overhead compared to a generic CPU implementation.

[17] E. A. J. F. Peters and G. de With, Phys. Rev. E 85, 026703– (2012).

Toshiba's Simulated Bifurcation Machine (SBM)

A ***simulation*** of the classical dynamics of Kerr-nonlinear parametric oscillators [18]

$$\frac{d^2}{dt^2} y_i(t) = -\left(y_i(t)^2 - p(t) + 1\right) y_i(t) + C \left(h_i + \sum_{\langle i,j \rangle} J_{ij} y_j(t) \right)$$

- $y_i(t)$ is a continuous variable and is projected to an Ising spin value at the end of the evolution depending on whether it is positive or negative.
- $\{h_i\}$ and $\{J_{ij}\}$ are the Ising local fields and couplings of the problem instance
- $p(t)$ is 'annealed' from 0 to 1 in increments of Δt (discretization of the differential equation).
- In our benchmarking, we use the implementation on Amazon AWS, which can auto-adjust the parameter C, leaving the time parameters Δt and t_f to be optimized.

We expect the simulation to give the same algorithm performance in terms of scaling with system size as a physical device implementing the same classical dynamics, but a physical device may have a reduced overhead.

[18] H. Goto, Journal of the Physical Society of Japan 88, 061015 (2019).

D-Wave Advantage (DWA)

Hardware based on superconducting circuits implementing quantum annealing

Low energy description:

$$H(s) = -A(s)\sum_i \sigma_i^x + B(s)\left(\sum_i h_i\sigma_i^z + \sum_{\langle i,j\rangle} J_{ij}\sigma_i^z\sigma_j^z\right)$$

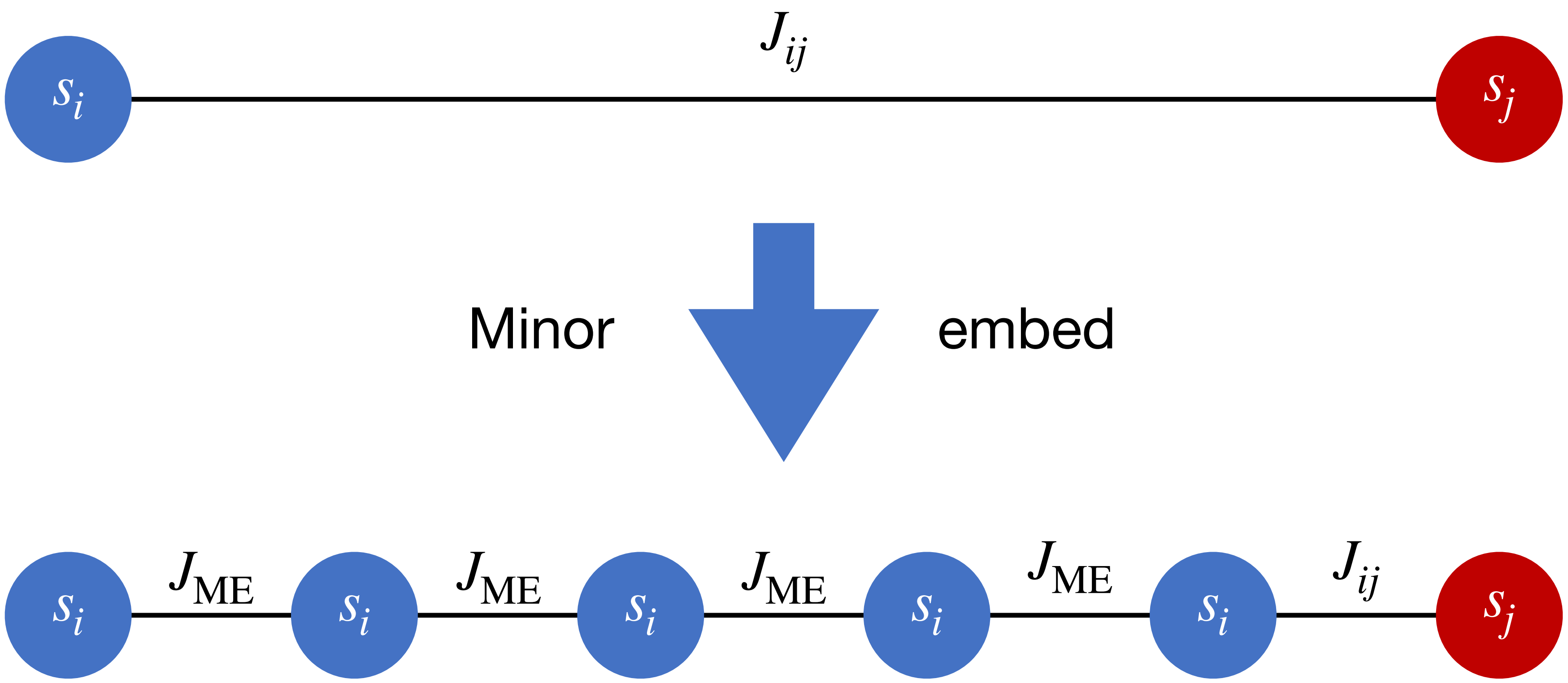
Starting from the thermal state of $H(0)$, system is evolved from $s = 0$ with $A(0) \gg B(0)$ to $s = 1$ with $B(1) \gg A(1)$.

How is this different from the quantum adiabatic optimization algorithm?

- The evolution need not be adiabatic
- The evolution is not restricted to be a closed system evolution. In fact, we know that the output statistics of the D-Wave quantum annealers are dominated by open quantum system effects.

D-Wave's Disadvantage

Connectivity graph restricted by hardware connectivity; to implement all-to-all connectivity requires a minor-embedding



Minor embed

Effective system size is larger than the actual problem instance, making it even more susceptible to noise (both thermal and implementation errors).

Introduces yet another optimization parameter: the embedding chain strength J_{ME}. Limited interaction strength range means optimizing this parameter can be problematic.

Recap

- Quasi-greedy algorithm (SATonGPU): Will operate on the 3-body problem (uses $N = n/2$ variables). Highly efficient implementation using GPUs.

- Quantum adiabatic optimization: Results known for the 3-body problem. No physical device to implement the algorithm.

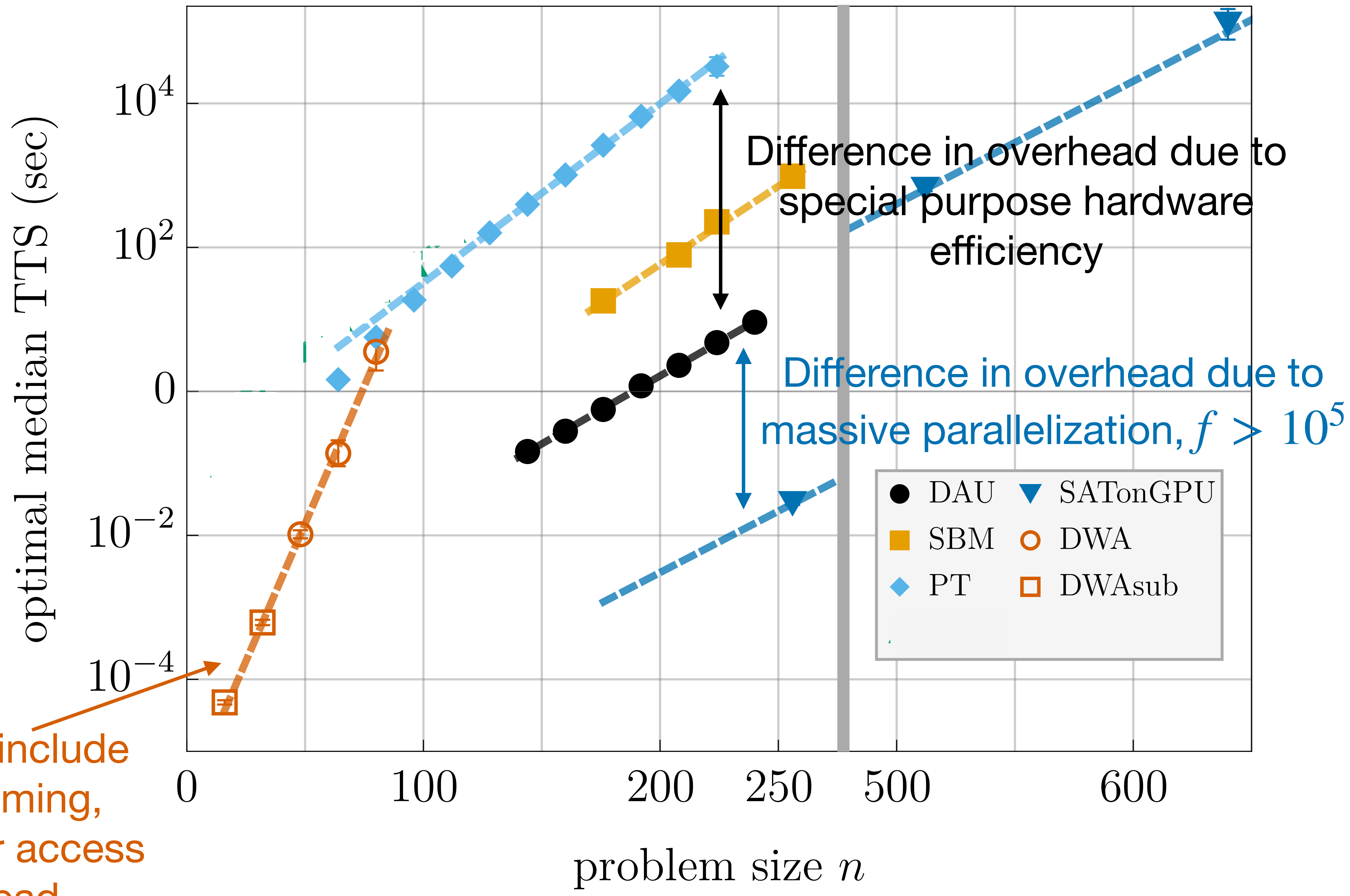
- Digital Annealer Unit (DAU): Will operate on the 2-body problem. Includes temperature optimization.

- Parallel tempering (PT): Will operate on the 2-body problem. To differentiate from the DAU, will be run on a single CPU core and not include temperature optimization.

- Simulated Bifurcation Machine (SBM): Will operate on the 2-body problem. Amazon AWS implementation uses 8 GPUs.

- D-Wave Advantage (DWA): Will operate on the 2-body problem. Requires a minor embedding onto the physical connectivity graph.

Results [19]



Difference in overhead due to special purpose hardware efficiency

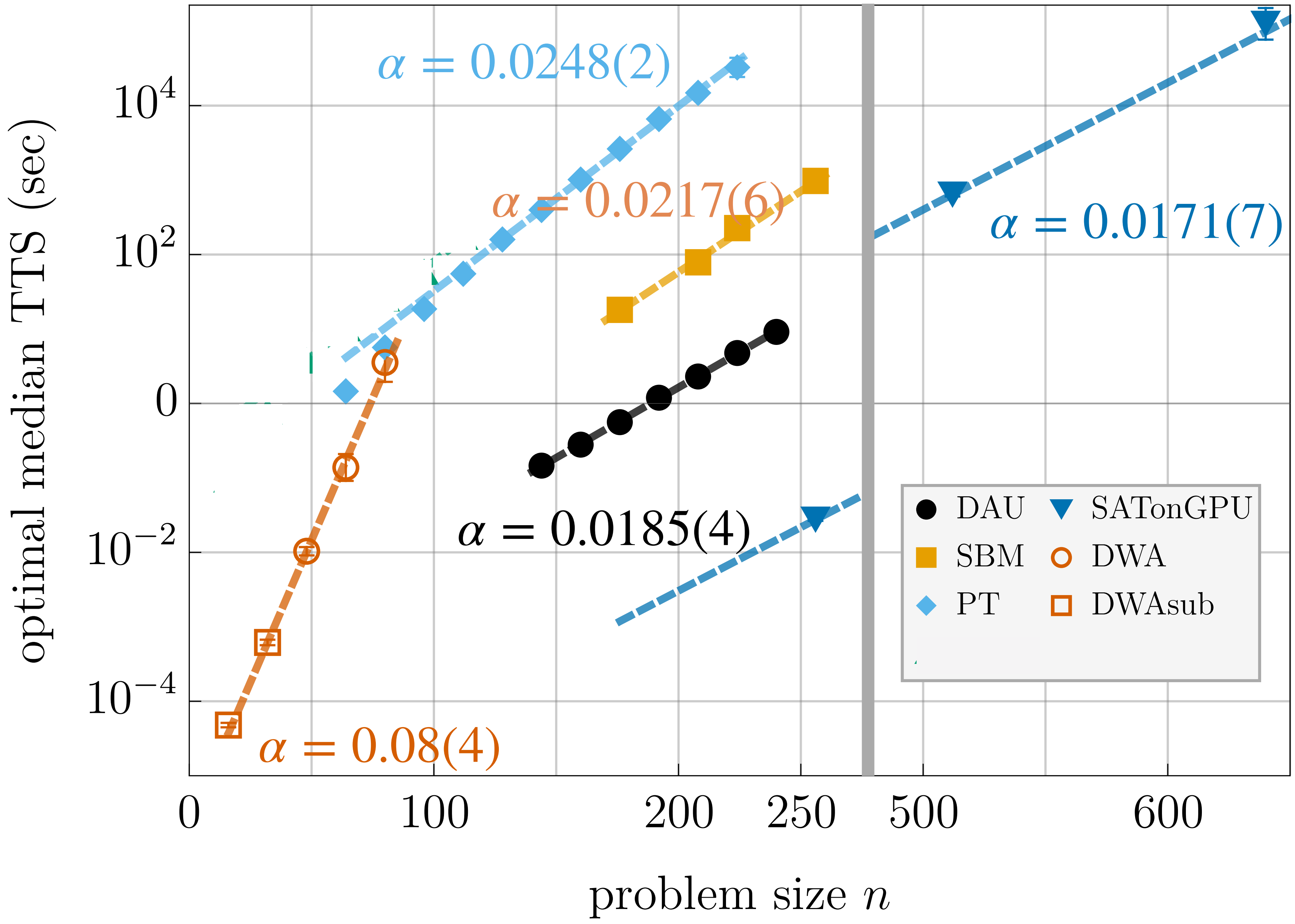
Difference in overhead due to massive parallelization, $f > 10^5$

Does not include programming, readout, or access overhead

[19] M. Kowalsky, TA, I. Hen, and D. A. Lidar, arXiv:2103.08464

Results [19]

Measure exponential scaling as $10^{\alpha n}$



[19] M. Kowalsky, TA, I. Hen, and D. A. Lidar, arXiv:2103.08464

Conclusions

- The limited connectivity, binary couplings, and entropic barriers of 3-regular 3-XORSAT mean that an extremely efficient implementation of the quasi-greedy on GPUs is possible. This approach will not generalize to other problem classes.

- DAU achieves the best scaling and best overhead for our suite of algorithms that worked on the 2-body problem. It's ability to tackle generic QUBO's make it an interesting solver, especially if it can be scaled to larger sizes and possibly include higher order interaction terms.. It would be interesting to study how it performs on other problem classes.

- The difference between the DAU and vanilla PT shows how important it is to optimize the temperature distribution.

- The SBM results suggests the a physical classical bifurcation machine will not be competitive in terms of scaling for this problem class.

- For the DWA, the process of minor embedding and the inherent noise on the device make it a poor choice for solving this class of problems. In principle, the quantum adiabatic optimization algorithm on the 3-body problem should do better, but we do not expect a quantum speedup for this class of problems.

Final Remarks

Identifying the optimum TTS of a quantile scaling with problem size of an algorithm requires identifying the optimal parameters of the algorithm that minimize the TTS for that quantile. In principle, this is a one-time cost, but this initial cost can be very large. This is the kind of approach that is needed when we are interested in identifying algorithmic improvements in terms of asymptotic scalings. This is what we did in this study.

But this might not be very practical; for example, if you want to provide your optimization algorithm as a Cloud service, you might expect to receive a wide range of problem classes, and you won't have the time to find the optimal simulation parameters for each problem class.

Instead, you might be more interested in finding reasonably good simulation parameters quickly and getting reasonable results in a short time for as wide a range of problem classes as possible. In this kind of situation, asymptotic scaling is probably not your most pressing concern.

These considerations suggest a different kind of benchmarking study; for example, the user specifies only the problem instances and the total allowed time; all simulation parameters must be optimized by the algorithm/hardware as well as solve (as many?) instances from the problems classes as best as possible within the specified total time. A score is then given based on the quality of solutions returned given the total time used. (Or use the procedure of SAT competitions?)