Sherrington-Kirkpatrick model

Xing Liu

Sherrington-Kirkpatrick model

This project focuses on solving the ground state energy problem of the classical **Sherrington-Kirkpatrick (SK) model**. The SK model is a fundamental example of spin glass systems in statistical physics, defined by a set of interacting spin variables $S = (S_1, S_2, \ldots, S_N)$, where each spin S_i takes values ± 1.

The Hamiltonian (energy function) of the model is given by:

$$H(S) = -\sum_{1\leq i<j\leq N} J_{ij} S_i S_j$$

Here, the coupling matrix $J = (J_{ij}) \in \mathbb{R}^{N\times N}$ represents the interaction strengths between spins. The goal is to find the spin configuration S that minimizes the system energy $H\left(S\right)$.

Mathematical Abstraction

Assuming J is a symmetric matrix with zero diagonal entries, the Hamiltonian can be expressed as

$$H\left(S\right) = -\frac{1}{2} S^T J S.$$

The problem then reduces to finding

$$S^* = \arg \min_{S\in\{\pm 1\}^N} H\left(S\right).$$

This problem holds significant importance in both combinatorial optimization and computational physics. It is a typical **Quadratic Unconstrained Binary Optimization (QUBO)** problem, known to be NP-hard, with computational complexity growing exponentially as the number of spins increases.

Algorithm Selection

Traditional methods struggle to directly solve large-scale instances of the problem. Therefore, this project implements multiple optimization algorithms, including:

- **Exhaustive Search** (CUDA-accelerated)

- **Simulated Annealing**
- **Semidefinite Programming (SDP) Relaxation**
- **Hybrid Strategy**
- And others

The goal is to compare the performance and effectiveness of these algorithms, providing a solid foundation for subsequent theoretical analysis and practical applications.

Algorithm Details

1. Exhaustive Search (CUDA-Accelerated, Reference Only)

Due to the relatively small dimensionality of the input data (only 30 spins), it is feasible to perform an exhaustive search over all spin configurations using CUDA. This algorithm has been tested on a Linux system equipped with an NVIDIA 4060 Ti GPU. In theory, any NVIDIA GPU with **more than 4 GiB of memory** should be able to run it without modification.

> **Note:** This method is *not* intended as a general-purpose algorithm. It is included solely to provide a **ground truth reference** for evaluating the accuracy of other algorithms in this project.

Core Idea

- First, note that the Hamiltonian satisfies the symmetry

  $$H\left(S\right)=H\left(-S\right),$$

  which allows us to **fix the last spin to -1**, reducing the search space by half. Similarly, for consistency, all later methods **only report one representative** of such symmetric configurations, and this property will not be reiterated.

- To avoid excessive data transfer and expensive configuration generation, each spin configuration is encoded as an unsigned long integer. Its binary representation (from least to most significant bit) corresponds to spins from the first to the last particle, with '0' representing spin -1 and '1' representing spin $+1$.

- We compute the energy of all configurations in parallel, store the energies in a global array, and use a CPU reduction to find the minimum. The total work is

  $$O\left(N^2\cdot 2^{N-1}\right),$$

  while the parallel span is

  $$O\left(N^2\right).$$

Although CUDA threads may not fully reduce the time complexity to $O(N^2)$ due to hardware constraints, thread multiplexing ensures the computation remains within practical bounds. The total expected time is on the order of **10 million operations**, which is well within the processing capability of modern GPUs.

- The memory usage consists of storing the coupling matrix (N^2 floats) and the energy array (2^{N-1} floats), totaling roughly **2 GiB** of VRAM. With additional overhead, a **4 GiB GPU** is sufficient.

In practice, this method takes less than a second to find the true ground state configuration.

Results

For the first instance in the provided dataset `sk30Jij.npy` (hereafter referred to as the *given data*), the spin configuration that yields the lowest energy is:

$$-1,+1,-1,-1,-1,-1,+1,+1,-1,-1,-1,+1,-1,+1,-1,$$
$$-1,-1,+1,+1,-1,-1,+1,+1,-1,+1,-1,-1,-1,-1,-1$$

with a corresponding minimum energy of -21.6217.

Since this result is theoretically exact with 100% accuracy, no further datasets were tested using this method. From the data generation process, we can observe that the time required to find an optimal solution is approximately between 300 and 600 milliseconds.

Generate Test Data

To evaluate the accuracy of subsequent methods, we generate a large number of spin glass instances with known optimal solutions. Since the coupling matrix J is symmetric with zero diagonal, only its upper triangular part (excluding the diagonal) needs to be specified, containing $\frac{N(N-1)}{2}$ elements.

These elements are independently sampled from the Gaussian distribution:

$$J_{ij} \sim \mathcal{N}(0, 0.2^2), \quad \text{for } i < j.$$

We generate 10,000 pairs of J and their corresponding ground truth optimal spin configurations S^*. For storage:

- The upper triangular part of each J is flattened into a 1D array of length $\frac{N(N-1)}{2}$;
- Each S^* is stored as an integer, as described in the second point of the *Core Idea* section;
- The decoding procedures are implemented in the functions `flat2J` and `idx2result`, which reconstruct the full J matrix and the corresponding spin configuration from the encoded format.

2. Simulated Annealing

Among all the implemented algorithms, Simulated Annealing is arguably the simplest, most classical, and most physically intuitive approach for this problem. It uses a gradually decreasing temperature variable `T` to control the acceptance probability of energy-increasing transitions. This mechanism allows the algorithm to more easily escape local minima in the early stages and to converge toward a local optimum in the later stages. This behavior mirrors the physical intuition that, in the early phase, a system is thermally unstable and more likely to escape from local energy minima due to thermal fluctuations.

Core Idea

- The algorithm maintains a temperature T that **decays over time**. At each iteration, a new state S^* is generated from the current state S by **flipping a randomly selected spin** (i.e., changing its sign).
- The energy difference is computed as

  $$\Delta E = H\left(S^*\right) - H\left(S\right),$$

  and the new state is **accepted with probability**

  $$P = \exp\left(-\frac{\Delta E}{T}\right).$$

  When $\Delta E < 0$, $P > 1$, so the new state is always accepted.
- The temperature T typically follows an **exponential decay** schedule, for example:

  $$T \leftarrow \alpha T, \quad \alpha < 1.$$

  Alternatively, more complex decay schemes can be used.

Results

Experimental results show that when the random seed is set to `1`, the algorithm can reproduce the true optimal solution on the given data. However, using the commonly adopted seed value of `42` does not yield the global optimum.

To evaluate the practical accuracy of the method, we tested it on the data generated before. The method achieved an accuracy of only **18.1%**. Nevertheless, due to the inherent randomness of the algorithm, we can significantly improve accuracy by **repeating the optimization multiple times** and selecting the lowest energy result.

And its advantage lies in the **short computation time**. The average time to complete one optimization is **1.56 ms**. Therefore, this method remains a viable and acceptable approach in practice.

3. Semidefinite Programming (SDP) Relaxation

This method relaxes the original combinatorial optimization problem into an SDP problem, which is a convex optimization problem and hence polynomial-time solvable under fixed precision. The relaxed problem is then repeatedly projected back into the feasible set of the original discrete problem using randomized rounding, and the lowest-energy solution among all projections is selected as the final output.

Core Idea

- First, introduce a matrix variable $X = SS^T$, then the objective function becomes

$$H\left(S\right) = S^T JS = \mathrm{Tr}\left(JSS^T\right) = \mathrm{Tr}\left(JX\right)$$

  The original optimization problem is thus rewritten as:

$$\begin{aligned} \min_{X \in \mathbb{R}^{n\times n}} \quad & \mathrm{Tr}(JX) \\ \text{s.t.} \quad & \mathrm{rank}(X) = 1, \quad X_{ii} = 1 \ \forall i, \quad X \succeq 0 \end{aligned}$$

- Then, we **relax the rank constraint** $\mathrm{rank}\left(X\right) = 1$, and keep only the diagonal constraints and the positive semidefiniteness constraint. This yields a convex **SDP relaxation**.
- Solve the relaxed SDP using `cvxpy` or any SDP solver to obtain X, then perform a Cholesky decomposition or eigendecomposition to factorize $X = VV^T$, where $V \in \mathbb{R}^{n\times r}$.
- Although V does not yield a feasible spin configuration directly, it provides a lower bound on the ground state energy.
- To recover a binary spin vector $\hat{S} \in \{-1, +1\}^n$, sample a Gaussian random vector $g \sim \mathcal{N}(0, I_r)$ and set

$$\hat{S}_i = \mathrm{sign}(V_i^T g)$$

- Repeat the above random projection step `num_rounds` times, and return the spin configuration with the lowest energy among all rounds.

Results

On the *given data*, using multiple random seeds consistently yields the true optimal solution. To evaluate the general applicability of this method, we also conduct tests on a large dataset. The results are as follows:

`num_rounds`	100	500	1000	10000
Accuracy	62.8%	74.4%	76.3%	79.9%

The average time to complete one optimization is **19.3 ms**.

In our experiments, setting `num_rounds = 100` yields an accuracy of approximately **62.8%**. Theoretically, increasing `num_rounds` to 1000 should reduce the error rate to less than 5×10^{-5} if each round is independent and identically distributed. However, the observed accuracy is only **76.3%**, suggesting that in some instances (about $\frac{1}{5}$ to $\frac{1}{4}$) the SDP relaxation introduces the integrality gap.

In our case (i.e., finding ground states of the Ising model), this integrality gap is not analytically bounded, but empirical evidence clearly suggests that for some instances, the SDP solution lies too far from any valid spin configuration, making it difficult for random projections to recover the true ground state, and thereby limiting the achievable accuracy.

4. Hybrid strategy

Core Idea

To address the integrality gap issue, I propose a **hybrid strategy**. Since the solution obtained via SDP can sometimes get stuck in local optima of the discrete problem, we incorporate a **simulated annealing** approach to improve upon it.

Specifically, we fix the number of projection rounds (`num_rounds`) to 500. After obtaining a theoretically optimal solution from SDP, we use it as the starting point for simulated annealing. However, to prevent excessive perturbations, we set a **lower initial temperature**.

This design ensures that if the SDP solution is trapped in a local optimum, it still has a chance to **escape**. On the other hand, if the solution is already globally optimal, even if it temporarily escapes, the annealing process will eventually return to the **globally optimal**.

Result

To determine the appropriate initial temperature, I conducted experiments with four different values: `T_init` = 10.0, 1.0, 0.5, and 0.1. `num_round` = 500.

`T_init`	10.0	1.0	0.5	0.1
Accuracy	78.6%	83.2%	83.6%	82.4%

The average time to complete one optimization is **20.7 ms**.

It can be seen that compared to plain SDP relaxation, this method increases the computation time by less than one-tenth, while achieving an improvement of more than one-tenth in accuracy when $T_{\text{init}} \leq 1.0$. For accuracy values already close to 1, a gain of one-tenth constitutes a significant improvement.

5. Other methods

I previously attempted a neural network-based strategy to directly model the spin optimization problem in fixed dimensions. The idea was to input the upper triangular part of the coupling matrix J and output a multi-label classification representing the optimal spin configuration. This approach ultimately failed, as the spin labels are highly coupled and cannot be effectively treated as independent classification targets based on their indices.

Due to the overall redundancy caused by multiple methods explored throughout the project, I performed a code refactoring. The dataset encoding used for this neural network approach was incompatible with the newly structured format, and thus the corresponding code was not preserved.

Despite the failure, this attempt provided valuable insights. If one seeks to apply artificial intelligence to this problem, a more promising direction may involve using models to learn the flipping strategies in simulated annealing, or even to approximate the temperature schedule itself—potentially replacing the standard exponential decay with a learned curve. The goal would be to escape local minima more efficiently and reach the global optimum faster.

Project Structure

```
SherringtonKirkpatrick
├── .gitignore
├── README.md
├── data
│   ├── generated_data.pkl
│   ├── sk30Jij.npy
│   └── sk_data.bin
└── src
    ├── data_manager.py
    ├── exhaustive_search
    │   ├── CMakeLists.txt
    │   ├── data
    │   │   └── sk30Jij.txt
    │   ├── include
    │   │   └── solver.cuh
    │   └── src
    │       ├── generate_data.cpp
    │       ├── result.cpp
    │       └── solver.cu
    ├── methods
    │   ├── __init__.py
    │   ├── exhaustive_search.py
    │   ├── sdp_relaxation.py
    │   └── simulated_annealing.py
    ├── npy2txt.py
    ├── result.py
    └── test.py
```

Usage Instructions

Both `result.py` and `test.py` in the `src` directory can be executed directly. The former evaluates performance on the **given data**, while the latter tests **accuracy and runtime** on the **generated dataset**.

If you want to switch to a different dataset **without** using exhaustive search, you should modify `result.py` accordingly to correctly load your data.

If you intend to use exhaustive search with a new dataset to find the true ground state energy, please follow these steps:

1. **Ensure your system meets the requirements:**
   - NVIDIA GPU
   - At least $\left(2^{N-29} + 2\right)$ GiB of GPU memory
2. **Configure the CUDA environment:**
   - Make sure the CUDA toolkit is properly installed. If `nvcc` is not in your system `PATH`, you must specify its location by setting the `nvcc_path` variable in `methods.exhaustive_search`.
3. **Run the exhaustive search:**
   - Either execute `exhaustive_search.py` directly or set `run_exhaustive_search` to be True in `result.py` and then run `result.py`.
4. **Optional -Use the C++ version:**
   - A C++ implementation is provided for CUDA-based ground state computation and data generation. You can use `npy2txt.py` to convert coupling matrices to `.txt` format, then modify `CMakeLists.txt` and relevant paths before compiling and executing.