GPU-accelerated Algorithms on solving Stochastic Shortest Path Problems Midpoint

Duo Zhou
Department of Industrial & Enterprise Engineering
University of Illinois Urbana-Champaign
Urbana, IL 61801
duozhou2@illinois.edu

Yilan Jiang
Department of Industrial & Enterprise Engineering
University of Illinois Urbana-Champaign
Urbana, IL 61801
yilanj2@illinois.edu

Tharunkumar Amirthalingam
Department of Industrial & Enterprise Engineering
University of Illinois Urbana-Champaign
Urbana, IL 61801
ta9@illinois.edu

Abstract

The stochastic shortest path problem is a special version of general shortest path problem combining with a Markov decision process. In this paper, we propose a parallel version of the stochastic shortest path algorithm. The stochastic shortest path problem is a significant area of academic research in the field of optimization. In this paper, we plan that not only would propose a parallel version of the stochastic shortest path algorithm depending on CUDA, but we would also provide a detailed analysis of its performance and characteristics. Our research aims to contribute to the advancement of the field by accelerating the time efficiency of algorithms on stochastic shortest path and exploring its potential uses in Big Data contexts.

1 Introduction

Stochastic shortest path is a problem in which we need to find the shortest path from a given start node to a goal node in a graph, where the edge weights are not deterministic but instead are random variables. This problem is commonly used in many applications, such as robotics, transportation, and network routing.

Regular shortest path algorithms, such as Dijkstra's algorithm or the A* algorithm, assume that the edge weights in a graph are deterministic and known in advance. These algorithms work well when the edge weights are fixed and do not change over time. However, in many real-world applications, the edge weights are not fixed but instead are subject to randomness and uncertainty. For example, in transportation networks, the travel time between two locations can vary depending on traffic conditions, weather, accidents, etc. Similarly, in robotic navigation, the cost of moving

Preprint. Under review.

from one location to another can depend on sensor readings, terrain features, obstacles, etc. In such scenarios, regular shortest path algorithms may not be appropriate as they do not take into account the randomness and uncertainty of the edge weights. Stochastic shortest path algorithms, on the other hand, explicitly model the probabilistic nature of the edge weights and aim to find the path with the lowest expected cost.

The stochastic shortest path problem is a challenging problem in stochastic optimization, and its solution requires a combination of mathematical and computational techniques. The need for using GPUs, particularly CUDA, to speed up the stochastic shortest path algorithm depends on various factors such as the size of the graph, the complexity of the edge weight distributions, and the available computing resources. In general, stochastic shortest path algorithms involve computations with matrices and vectors, which can be computationally intensive for large graphs. GPUs, with their massively parallel architecture and high memory bandwidth, can accelerate these computations significantly, leading to faster computation times. Moreover, if the edge weight distributions are complex and involve high-dimensional probability distributions, such as multivariate normal or mixture distributions, then the computations involved in the algorithm may be even more demanding. In such cases, GPUs can provide significant speedups compared to CPU-based implementations.

2 Literature Review

There have been plenty of algorithms innovated to resolve the general shortest path problem in the past decades. Gallo and Pallottino discussed eight algorithms for the shortest path problem and compared their performances with various data structures [1]. To further increase the computational efficiency, Crauser et alṗroposed a parallelized version of the Dijkstra's algorithm [2]. Even though the shortest path problem can be applied to many areas like road networks and social media, it is less generic to be applied to some other real life problems such as robotics, autonomous driving, and network routing.

Bertsekas and Tsitsiklis generalized the shortest path problem and first introduced the idea of stochastic shortest path (SSP) problem in 1991 [3]. They further extended the corresponding theory of Markovian decision problems by removing the assumptions that cost can either be all positive or all negative. Building on top of that, Polychronopoulos and Tsitsiklis developed a dynamic programming algorithm to resolve a SSP problem to devise a policy that leads from an origin to a destination with minimal expected cost (4). Further works have been done to expand the ecosystem of SSP problem by proposing new concepts and frameworks [5, 6]. In addition, SSP is defined as an instance of a Markov Decision Process.

On Markov Decision Processes with Parallel Algorithm, Archibald, T. W. et al. looked at serial value iteration algorithms for Markov decision processes and develops efficient parallel alternatives [7]. Jóhannsson, Á.Þ. et al. introduced two CUDA-based implementations of the Value Iteration algorithm: Block Divided Iteration and Result Divided Iteration and showed a substantial performance improvement for the parallel algorithms compared to a sequential implementation on a CPU [12]. Ruiz and Hernandez formulated a MDP solver based on the Value Iteration algorithm that uses matrix multiplications. This allows us to leverage GPUs to produce interactive obstacle-free paths in the form of an Optimal Policy [10].

Ortega-Arranz, H. et al. presented a GPU SSSP algorithm implementation significantly sped up the computation of the SSSP. They also enhanced the GPU algorithm implementation using proper choice of threadblock size and the modification of the GPU L1 cache memory state of NVIDIA devices [11]. Sapio et al. introduced two new MDP solvers for embedded systems: Sparse Value Iteration (SVI), which operates on small, single-threaded CPU platforms using sparse matrix methods, and Sparse Parallel Value Iteration (SPVI), which takes advantage of embedded graphics processing units (GPUs) to increase performance on more advanced embedded systems [8]. Steimle et al. developed exact and fast approximation methods with error bounds for Multi-model Markov decision process (MMDP), which generates a single policy maximizing performance over all models. This approach allows the decision maker to trade-off conflicting data sources while creating a policy of the same complexity for models that consider only one data source [9].

Table 1: Milestons

Time	Tasks
Mar. 9	Proposal Investigation
Mar. 23	Review of Literature & Algorithm Design
Mar. 28	Finish up Miderm Report
Apr. 13	Mathematical proof of Algorithm
Apr. 23	Experimental Design & Validation
Apr. 25	Final Report Completion

3 Proposed work

3.1 Tasks

In Task 1, we will spend sometime to explore the mathematical theories as well as algorithms relevant to stochastic shortest path and markov decision processes in greater details. By diving into these topics, we will be able to better understand the underlying principles that govern these processes and apply this knowledge to real-world situations.

For Task 2, in order to solve the stochastic shortest path problem, we can reproduce various existing algorithms. These algorithms can be implemented using a CPU as a baseline scenario. By utilizing these techniques, we can efficiently identify the optimal path in a given graph, even when the graph has stochasticity. Additionally, we may consider implementing these algorithms on GPUs especially using CUDA in order to further optimize the solution.

3.2 Data

We will evaluate our method on a list of infinite-horizon Multi-model Markov Decision Processes (MMDPs). These MMDPs are test instances generated by using the codes provided by University of Michigan-Deep Blue Data. We will use their codes to generate three sets of MMDPs, each corresponding to different medical decision-making problems. These problems include the initiation of HIV therapy, a machine maintenance problem, and a set of randomly constructed instances. A more detailed description of the data under each problem can be found online.

`https://deepblue.lib.umich.edu/data/concern/data_sets/t722h899p`

3.3 Milestons

Referring to the schedule Table **??**, our milestone is shown above.

3.4 Testing and Expected results

We will choose CUDA enabled NVIDIA GPU and C++ to realize the paralleled computations and use python to take care of all other programming works including data preprocessing and pipeline construction. We will test our method compared to the baseline model using CPU counterparts in terms of speed. It is important to choose the appropriate method or algorithm for the specific problem at hand and to carefully analyze the results to ensure that the solution is accurate and effective.

4 Problem Description

The Stochastic Shortest Path (SSP) problem is an extension of the classical shortest path problem, where the cost of traversing edges is uncertain and modeled as random variables. In the SSP, we aim to find a path with the minimum expected total cost from a source node to a destination node in a directed graph. To formulate this problem, we define a few notations.

Let $G = (V, E)$ be the graph containing a set of vertices V and a set of edges E. Let $s \in V$ and $t \in V$ be the source and sink nodes. Let the set of paths from s to t be P. The costs of traversing edge $(i, j) \in E$ is given by the random variable c_{ij}. Let the probability distribution function of c_{ij} be $f_{ij}(c_{ij})$. We assume the costs are non-negative and independent.

We seek to find the least expected cost path from s to t:

$$p^* = \arg\min_{p \in P} \mathbb{E}(C_p),$$

where

$$\mathbb{E}(C_p) = \sum_{(i,j) \in p} \mathbb{E}(c_{ij}) = \sum_{(i,j) \in p} \int_0^\infty c_{ij} f_{ij}(c_{ij})\, dc_{ij}.$$

We can discretize the cost parameter to approximate the integration term of the expected cost. Let there be N possible values of $c_{ij} \forall (i,j) \in E$ given by $\{c_{ij}^1, c_{ij}^2, ..., c_{ij}^N\}$ with corresponding probabilities $\{p_{ij}^1, p_{ij}^2, ..., p_{ij}^N\}$. The discretization can be done by methods like scenario generation or Monte Carlo sampling. The strength of the solution depends on how fine the discretization is but

4.1 Network Flow Representation

Let x_{ij} be the binary decision variable which is 1 if edge $(ij) \in E$ is selected, and 0 otherwise. Now, we can write the SSP in terms of expected costs as:

$$\min \sum_{(i,j) \in E} \left(\sum_{k=1}^{N} p_{ij}^k c_{ij}^k \right) x_{ij} \tag{1}$$

$s.t.$

$$\sum_{j:(i,j) \in E} x_{ij} - \sum_{j:(i,j) \in E} x_{ji} = b_i \quad \forall i \in V \tag{2}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E, k = 1, ..., N \tag{3}$$

where

$$b_i = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise.} \end{cases}$$

We can solve the above problem with standard LP solvers if the expected costs are known, or can be computed, beforehand. Otherwise, it can be tricky to solve the problem [4].

5 Methodology

Due to the combinatorial nature of the problem and randomness of the cost variables, solving the SSP problem can be computationally difficult. However, a number of approaches, including dynamic programming, modified Dijastra's algorithm, and Monte Carlo simulation-based methods, have been suggested to solve the problem. Conventionally, these methods use discretized versions of the random cost distributions to determine the shortest path with minimum expected cost.

5.1 Dynamic Programming

Dynamic programming [4] is a method commonly used to solve SSP problem. The have been many variations of dynamic programming algorithms targeting different types of SSP problems such as value iteration, policy iteration, and Q-learning. In our case, we define the dynamic programming algorithm as follows. Before we delve into the pseudocodes, let us define the value function $V(i)$ using Bellman equation [].

$$V(i) = minc(i,j) + \mathbb{E}[V(j)] \tag{4}$$

where the minimum is taken over all outgoing edges (i,j) from node i to j, and $\mathbb{E}[V(j)]$ is the expectted cost of the shortest path from node j to the terminal node t.

Algorithm 1 Stochastic Shortest Path Using Dynamic Programming

Input: Graph $G = (V, E)$, source node s, sink node t, discretized_costs c, probability distribution function p
Output: Shortest expected path from s to t, min_expected_cost

procedure SSP_DP(G, s, t, c, p)
  Initialization:
  for $v \in V$ **do**
    $V(v) \leftarrow None$
  end for
  min_expected_c ← DP_Helper(G, s, t, c, p)
  path ← Reconstruction_Path(G, s, t, c, p, V)
  return min_expected_c, path
end procedure

procedure DP_HELPER(G, u, t, c, p, V)
  if $u == t$ **then**
    return 0
  end if
  if $V[u]! = None$ **then**
    return V[u]
  end if

  Initialize: min_expected_c ← ∞
  for neighbor v of u $\in G$ **do**
    expected_cost[u][v] ← $\sum_{(u,v)\in E} c[u][v] \times p[u][v]$
    subproblem_cost ← DP_Helper(G, v, t, c, p, V)
    total_expected_cost ← expected_cost[u][v] + subproblem_cost
    if total_expected_cost < min_expected_c **then**
      min_expected_c ← total_expected_cost
    end if

    V[u] ← min_expected_cost
    return min_expected_cost
  end for
end procedure

procedure RECONSTRUCT_PATH(G, u, t, c, p, V)
  Initialize: an empty path p
  current_node ← u

  while current_node ! = t **do**
    for neighbor v of current_node that minimize the total expected cost **do**
      expected_cost[current_node][v] ← $\sum_{(\text{current\_node},v)\in E} c[\text{current\_node}][v] \times p[\text{current\_node}][v]$
      total_expected_cost ← expected_cost[current_node][v] + V[v]
      append v to p
      current_node ← v
    end for
  end while
  return p
end procedure=0

5.2 Modified Dijkstra's Algorithm

Algorithm 2 SPP_Dijkstra

Input: Graph $G = (V, E)$, source node s, sink node t, discretized_costs c, probability distribution function $p : E \to [0, 1]$
Output: Shortest expected path from s to t

procedure SSP_DIJKSTRA(G, s, t, c, p)
  Initialize: an empty priority queue Q, an empty path dictionary P, expected cost array E_costs
  for $v \in$ E_costs **do**
    E_costs $\leftarrow \infty$
  end for
  E_costs[s] $\leftarrow 0$
  Insert (s,0) into Q

  while Q is not empty **do**
    u $\leftarrow$ Q.pop()
    if $u == t$ **then**
      break
    end if

    for neighbor v of u in G **do**
      expected_cost[u][v] $\leftarrow \sum_{(u,v)\in E} c[u][v] \times p[u][v]$
      new_E_cost = E_costs[u] + expected_cost[u][v]
      if new_E_cost $<$ E_costs[v] **then**
        E_costs[v] = new_E_cost
        P[v] = u
        Insert (v, new_E_cost) into Q
      end if
    end for
  end while
  return Reconstruct_Path(P, s, t)
end procedure

procedure RECONSTRUCT_PATH(P, s, t)
  Initialize: an empty list path p
  current_node $\leftarrow$ t
  while current_node != s **do**
    Prepend current_node to p
    current_node $\leftarrow$ P[current_node]

    Prepend s to p
  end while
  return p
end procedure
=0

6 Outline of Numerical Application on CUDA

A. Motivation for using parallel algorithms on CUDA to solve Stochastic Shortest Path problems

Implementing a parallel algorithm for the SSP problem can be challenging due to the need for efficient parallel data structures and handling race conditions. However, when implemented correctly, parallel algorithms can significantly speed up the computation, especially for large graphs.

B. Overview of the algorithmic approach and kernel design

It's possible to develop parallel algorithms for solving the Stochastic Shortest Path (SSP) problem implementing a parallel programming model in CUDA. The most commonly used parallel algorithms

for shortest path problems are based on Dijkstra's or Bellman-Ford's algorithms. For the SSP problem, we can adapt these algorithms to work with the discretized stochastic costs.

6.1 Preprocessing

A. Discretization of random costs

1. Conversion of random edge costs into deterministic costs 2. Assignment of corresponding probabilities to each deterministic cost

B. Preprocessing on the CPU

C. Data partitioning for efficient kernel execution

1. Dividing the graph into subproblems 2. Assigning subproblems to CUDA thread blocks

6.2 Data Representation

A. Graph representation techniques

1. Adjacency matrix 2. Adjacency list

B. Compressed Sparse Row (CSR) format

1. Efficient memory usage for large graphs 2. Compatibility with parallel algorithms

C. GPU data transfer

1. cudaMemcpy for data transfer between CPU and GPU 2. Asynchronous data transfer for overlapping computation and communication

6.3 Kernel Design for Parallel Algorithms

A. Designing a kernel for parallel Dijkstra's Algorithm

1. Thread assignment for processing nodes and edges 2. Shared memory usage for priority queue and node data

B. Designing a kernel for parallel Bellman-Ford's Algorithm

1. Thread assignment for processing nodes and edges 2. Shared memory usage for node data and intermediate results

C. Load balancing and thread cooperation

1. Dynamic thread assignment to minimize idle threads 2. Thread cooperation for atomic operations and memory access

6.4 Iteration and Synchronization

A. Intra-kernel synchronization

1. Use of syncthreads or thread barriers within a kernel 2. Ensuring completion of all threads before proceeding

B. Inter-kernel synchronization

1. Use of CUDA events for kernel synchronization 2. Overlapping kernel execution and data transfer for efficiency

6.5 Termination Criteria

A. No updates to the shortest path estimates

B. Maximum number of iterations reached

C. GPU-based termination checking

1. Reduction-based global termination check 2. Asynchronous termination check using CUDA streams

6.6 Path Reconstruction

A. Extraction of the shortest path from source to destination node

1. Parallel backtracking on the GPU 2. Efficient data structures for storing path information

B. CPU-based path reconstruction

1. Transfer of path data from GPU to CPU 2. Formatting and output of the final path

7 Discussion and Evaluation

A. Summary of the parallel algorithm implementation on CUDA with efficient kernel design

B. Potential applications and future work in solving Stochastic Shortest Path problems

C. Evaluation of performance improvements and scalability

8 Conclusion

TBD

References

[1] Gallo, G., & Pallottino, S. (1988). Shortest path algorithms. *Annals of operations research* **13**(1): 1-79.

[2] Crauser, A., Mehlhorn, K., Meyer, U., & Sanders, P. (1998). A parallelizttion of Dijkstra's shortest path algorithm. *In Mathematical Foundatio s of Computer Science 1998: 23rd International Symposium MFCS'98 Brno, Czech Republic, August 24-28, 1998 Proceedings* **23**(pp. 722-731). Springer Berlin Heidelberg.

[3] Bertsekas, D. P., & Tsitsiklis, J. N. (1991). An analysis of stochastic shortest path problems. *Mathematics of Operations Research* **16**(3): 580-595.

[4] Polychronopoulos, G. H., & Tsitsiklis, J. N. (1996). Stochastic shortest path problems with recourse. *Networks: An International Journal* **27**(2): 133-143.

[5] Ji, X. (2005). Models and algorithm for stochastic shortest path problem. *Applied Mathematics and Computation* **170**(1): 503-514.

[6] Guillot, M., & Stauffer, G. (2020). The stochastic shortest path problem: a polyhedral combinatorics perspective. *European Journal of Operational Research* **285**(1): 148-158.

[7] Archibald, T. W., McKinnon, K. I. M., & Thomas, L. C. (1993). Serial and parallel value iteration algorithms for discounted Markov decision processes. *European journal of operational research* **67**(2): 188-203.

[8] Sapio, A., Bhattacharyya, S. S., & Wolf, M. (2020, July). Efficient model solving for Markov decision processes. *In 2020 IEEE Symposium on Computers and Communications (ISCC)*(pp. 1-5). IEEE.

[9] Steimle, L. N., Kaufman, D. L., & Denton, B. T. (2021). Multi-model Markov decision processes. *IISE Transactions* **53**(10): 1124-1139.

[10] Ruiz, S., & Hernández, B. (2015, October). A parallel solver for Markov decision process in crowd simulations. *In 2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*(pp. 107-116). IEEE.

[11] Ortega-Arranz, H., Torres, Y., Gonzalez-Escribano, A., & Llanos, D. R. (2015). Comprehensive evaluation of a new GPU-based approach to the shortest path problem. *International Journal of Parallel Programming* **43**(5), 918-938.

[12] Jóhannsson, Á.Þ. (2009). GPU-based Markov decision process solver.