GPU-based Split algorithm for Large-Scale CVRPSD

Jingyi Zhao[1], Linxin Yang[1,2], Haohua Zhang[4], Tian Ding[1,3,4]
1. Shenzhen Research Institute of Big Data, Shenzhen, China
2. The Chinese University of Hongkong, Shenzhen, China
3. Shenzhen International Center for Industrial and Applied Mathematics, Shenzhen, China
4. AutoKernel, Shenzhen, China

Abstract

Dynamic programming (DP) is a cornerstone of combinatorial optimization, yet its inherently sequential structure has long limited its scalability in scenario-based stochastic programming (SP). This paper introduces a GPU-accelerated framework that reformulates a broad class of forward DP recursions as batched min–plus matrix–vector products over layered DAGs, collapsing actions into masked state-to-state transitions that map seamlessly to GPU kernels. Using this reformulation, our approach takes advantage of massive parallelism across both scenarios and transitions, enabling the simultaneous evaluation of *over one million uncertainty realizations* in a single GPU pass – a scale far beyond the reach of existing methods. We instantiate the framework in two canonical applications: the capacitated vehicle routing problem with stochastic demand and a dynamic stochastic inventory routing problem. In both cases, DP subroutines traditionally considered sequential are redesigned to harness two- or three-dimensional GPU parallelism. Experiments demonstrate near-linear scaling in the number of scenarios and yield one to three orders of magnitude speedups over multithreaded CPU baselines, resulting in tighter SAA estimates and significantly stronger first-stage decisions under fixed time budgets. Beyond these applications, our work establishes a general-purpose recipe for transforming classical DP routines into high-throughput GPU primitives, substantially expanding the computational frontier of stochastic discrete optimization to the million-scenario scale.

1 Introduction

Background. The Capacitated Vehicle Routing Problem (CVRP, [18, 8, 14, 1, 19]) is a classical NP-hard problem with broad industrial relevance in logistics, last-mile delivery, and supply chains. The task is to design cost-minimizing routes for a fleet of identical, capacity-limited vehicles that start and end at a central depot, ensuring that all customers are served exactly once while respecting vehicle capacity constraints and satisfying customer demand. Its stochastic extension, the CVRPSD ([9, 5, 3, 4, 13]), models customer demands as random variables within a two-stage program: routes are chosen before demand revelation, followed by recourse actions such as restocking or rerouting. Despite extensive research ([15, 7, 12]), solving CVRPSD remains challenging in data-driven settings with correlated or complex demand distributions. State-of-the-art set-partitioning approaches often rely on restrictive independence assumptions to stay tractable. Scenario-based formulations offer greater modeling flexibility, but evaluating large scenario sets is computationally prohibitive on CPUs.

Stochastic programming naturally lends itself to parallelism: given a first-stage decision, the evaluation of second-stage costs across different scenarios is inherently independent. Thus, we propose a GPU-accelerated framework that reformulates the split algorithm into a vectorized, GPU-parallelizable form, enabling simultaneous evaluation across scenarios and transitions. This design supports millions of scenarios within practical runtimes, substantially reducing estimation bias in

sample-average approximation. While researchers in both operations research and machine learning have increasingly explored GPU-based parallelism to accelerate optimization algorithms (e.g., GPU-ADMM [16],CuPDLP [11], GPU-QP [2], GPU-IPM [10]), to the best of our knowledge, this work is the first to introduce an ultra-large-scale scenario-based solver for stochastic combinatorial optimization, capable of efficiently handling *over 1,000,000 of uncertainty realizations*. This represents a significant advancement in the computational frontier of scenario-based stochastic programming, which has traditionally been constrained by the computational cost of evaluating massive scenario sets.

Contribution. Our framework evaluates a first-stage routing decision by computing the second-stage recourse costs for all demand scenarios simultaneously on the GPU. This parallel evaluation enables rapid and reliable comparison of candidate solutions across ultra-large-scale scenario sets, providing an almost unbiased estimate of the expected cost. Consistent with the theoretical insight that larger scenario sets reduce estimation bias, our experimental results confirm that first-stage solutions derived from richer scenario sets achieve superior performance on out-of-sample tests.

To the best of our knowledge, this is the first approach capable of evaluating over 1,000,000 of demand scenarios within practical runtimes for first-stage decision making. More broadly, it represents one of the earliest systematic applications of GPU-based parallelization to stochastic combinatorial optimization, moving beyond the conventional reliance on CPU-based multi-threading. Our results further demonstrate that GPU parallelization enables much finer-grained computation than scenario-level simulation, achieving speedups of up to $65\times$ over CPU-based multi-threading and thereby unlocking unprecedented scalability.

Our parallel evaluation strategy integrates seamlessly into the Hybrid Genetic Search (HGS) framework originally proposed by [21], a widely used metaheuristic that has achieved state-of-the-art performance across numerous VRP variants [22, 23, 20, 6]. Importantly, the GPU-based parallel sub-problem solution module is generic: it can be embedded not only into metaheuristics but also into exact and approximate algorithms that require large-scale scenario evaluation. Beyond the CVRPSD, our parallel simulation approach is readily extensible to dynamic programming and rollout-based methods under massive scenario sets, offering a scalable tool for a broad class of stochastic optimization problems.

Viewed from the perspective of stochastic programming, our framework bridges a key methodological gap by substantially alleviating the long-standing scalability barrier in data-driven combinatorial optimization, thereby opening the door to previously intractable real-world applications.

2   Problem Setting

The stochastic programming community has extensively studied scenario-based formulations, where uncertainty is modeled by a finite set of realizations. However, scenario-based evaluation quickly becomes computationally prohibitive on CPUs, where even tens of thousands of scenarios can overwhelm multi-threaded implementations. In our work, we adopt such a *scenario-based modeling* framework, in which customer demands are represented by sampled realizations. This approach naturally accommodates correlated demand structures and supports data-driven modeling when historical records are available. This scenario-based approach falls naturally into the two-stage stochastic programming paradigm, whose general form is: $\min_{x \in X} f_1(x) + \mathbb{E}_\xi \left[f_2(x, \xi) \right]$, where x denotes the first-stage routing decisions and the corresponding cost $f_1(x)$, ξ is a random vector representing a realization of customer demands, and $f_2(x, \xi)$ denotes the second-stage recourse cost under each scenario ξ.

In our two-stage stochastic optimization problem, the first stage determines the visiting sequence of customers, commonly referred to as a *giant tour* in the context of genetic algorithms [21]. This representation encodes a solution as a permutation of customers, from which feasible vehicle routes can be recovered via a split operator under fixed vehicle capacity and scenario-dependent customer demands. We assume *full demand revelation prior to the second stage*, enabling the plan to adapt to the realized scenario. Once demands are known, the giant tour is *split into feasible routes* such that the demand on each route does not exceed vehicle capacity. Thus, given a fixed giant tour (i.e., the visiting sequence), the second-stage evaluation is computationally simple: it only requires splitting

the tour according to realized demands. **The objective is to determine a first-stage giant tour that minimizes the expected total travel cost across all scenarios.**
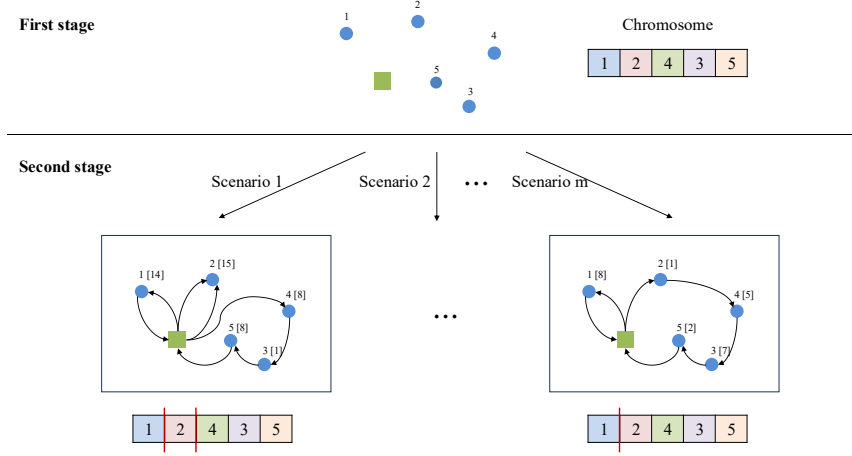

Figure 1: Example of splitting a giant tour into feasible routes under a given demand scenario.

Example 1. *Suppose the first-stage giant tour is $(1, 2, 4, 3, 5)$ with vehicle capacity $Q = 17$, as illustrated in Figure 1.*

- *(i) In Scenario 1, the realized demands are $[14, 15, 8, 1, 8]$. The second-stage routing may then split the tour into three feasible routes: (1), (2), and $(4, 3, 5)$.*

- *(ii) In Scenario m, the realized demands are $[8, 1, 5, 7, 2]$ with three vehicles available. The second-stage routing may then split the tour into two feasible routes: (1) and $(2, 4, 3, 5)$.*

It is well understood that training robust first-stage policies benefits from incorporating a large number of demand scenarios to capture the full breadth of uncertainty ([17], see details in the Appendix). Theoretical results show that a sufficiently large scenario set is critical for achieving accurate and stable estimations. However, in practice, evaluating even tens of thousands of scenarios can be prohibitively expensive, especially for combinatorial problems such as the CVRPSD, since each scenario requires solving a non-trivial routing evaluation (typically through a dynamic programming algorithm with time complexity $O(nP)$, where n is the number of customers and P is the number of possible transitions) rather than a simple function call.

3 GPU-Accelerated Dynamic Programming-based Split Algorithm

Dynamic Programming-based Second-stage Split Algorithm. Let a first-stage decision *giant tour* be denoted by $\sigma = [\sigma_0, \sigma_1, \ldots, \sigma_n, \sigma_{n+1}]$, where $\sigma_0 = 0$ is the departure depot, $\sigma_{n+1} = n + 1$ is the return depot, and $\sigma_1, \ldots, \sigma_n$ are the customers in the first-stage visiting sequence.

In the second-stage, for a given demand scenario $\omega \in \Omega$ with realized demand vector $\xi^\omega = (q^\omega_{\sigma_1}, \ldots, q^\omega_{\sigma_n})$, the task is to *split* the giant tour into a set of feasible vehicle routes that respect both the vehicle capacity Q and, if applicable, maximum route length or duration constraints, while minimizing the total travel cost.

We solve this splitting problem using a dynamic programming (DP) formulation. Let $f^\omega(i)$ denote the minimum total cost of serving customers $[\sigma_1, \ldots, \sigma_i]$ starting from the depot and returning to it under scenario ω. The base case is $f^\omega(0) = 0$. For $i \geq 1$, $f^\omega(i)$ is computed recursively from preceding states:

$$f^\omega(i) = \min_{0 \leq p \leq i-1} \left\{ f^\omega(p) + c_{0,\sigma_{p+1}} + \sum_{k=p+1}^{i-1} c_{\sigma_k, \sigma_{k+1}} + c_{\sigma_i, n+1} \;\middle|\; \sum_{k=p+1}^{i} q^\omega_{\sigma_k} \leq Q \right\}, \quad (1)$$

3

where $c_{a,b}$ denotes the travel cost from node a to node b. Here, p represents the index of the last customer in the previous route, so that σ_{p+1} is the first customer in the current route. The capacity constraint in (1) ensures that the total demand from σ_{p+1} to σ_i in scenario ω does not exceed Q. The recursion effectively enumerates all feasible last segments of the solution, selecting the one that yields the minimal total cost.
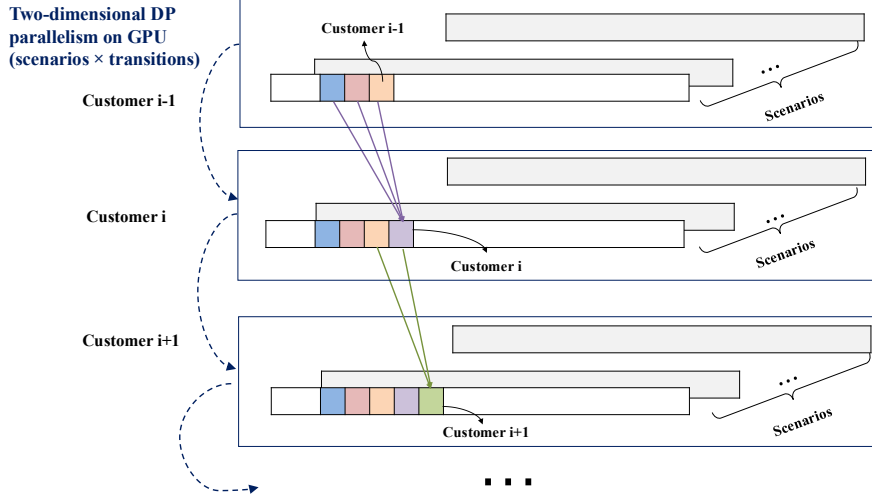

Figure 2: Example of Vectorized Split Algorithm.

GPU-Parallelized Vectorized Split Algorithm.   While the vanilla split algorithm in (1) is straightforward, it requires checking the capacity constraint for every candidate split point p at runtime. On a GPU, such per-transition feasibility checks can cause thread divergence and redundant memory access, severely limiting throughput when evaluating a large number of scenarios in parallel.

To address this, we introduce a *vectorized split algorithm* that precomputes, for each scenario $\omega \in \Omega$ and each position i, the earliest feasible split point satisfying the capacity constraint:

$$\text{mask}^\omega(i) \;=\; \min\left\{ p \;\middle|\; 0 \le p < i, \;\; \sum_{k=p+1}^{i} q_{\sigma_k}^\omega \le Q \right\}. \tag{2}$$

Here, $\text{mask}^\omega(i)$ denotes the smallest index p such that the segment $[\sigma_{p+1}, \dots, \sigma_i]$ is feasible under scenario ω. This precomputation can be carried out in parallel across all scenarios and all customer positions using prefix-sum operations on the demand vectors ξ^ω.

With $\text{mask}^\omega(i)$ available, the DP recursion becomes:

$$f^\omega(i) = \min_{\text{mask}^\omega(i) \le p \le i-1}\left\{ f^\omega(p) + c_{0,\sigma_{p+1}} + \sum_{k=p+1}^{i-1} c_{\sigma_k,\sigma_{k+1}} + c_{\sigma_i,n+1} \right\}. \tag{3}$$

The capacity feasibility check is now implicit in the lower bound $\text{mask}^\omega(i)$, eliminating conditional branching inside the DP loop.

The vectorized formulation naturally enables parallelization along two dimensions, as shown in Figure 2:

1. *Scenario-level parallelism:* each demand scenario ω is processed independently, making it straightforward to evaluate thousands of scenarios simultaneously on different GPU threads.
2. *Transition-level parallelism:* for each state i, the minimization over candidate split points $p \in [\text{mask}^\omega(i), i-1]$ can itself be parallelized, since all candidate costs are independent and can be reduced to a minimum using parallel reduction.

These two levels of parallelism substantially improve GPU efficiency: (1) all threads follow the same execution path, reducing warp divergence, and (2) memory accesses are more regular since $\text{mask}^\omega(i)$

4

values are stored contiguously, allowing coalesced reads. In practice, this reduces per-scenario runtime significantly, enabling the evaluation of tens or hundreds of thousands of scenarios within practical time limits.

4 Experiments

We conduct preliminary experiments to evaluate the efficiency of our GPU-accelerated split algorithm against CPU baselines. All implementations were written in C++/CUDA, and experiments were run on a machine with an AMD Ryzen 7 9700X CPU (8 cores) and an NVIDIA RTX 2080Ti GPU with 11 GB memory. The CPU baselines include (i) a single-threaded implementation, and (ii) a multi-threaded implementation with 8 threads. The GPU implementation exploits the two-dimensional parallelism described in Section 3.

4.1 Scalability with the Number of Scenarios

We first compare the runtime required to split a single giant tour under varying numbers of considered scenarios, ranging from 10^4 to 10^6. The GPU-based split algorithm is compared against single thread and 8-threads CPU baseline. Figure 3 presents the average runtime (in milliseconds) computed over 10 independent runs.
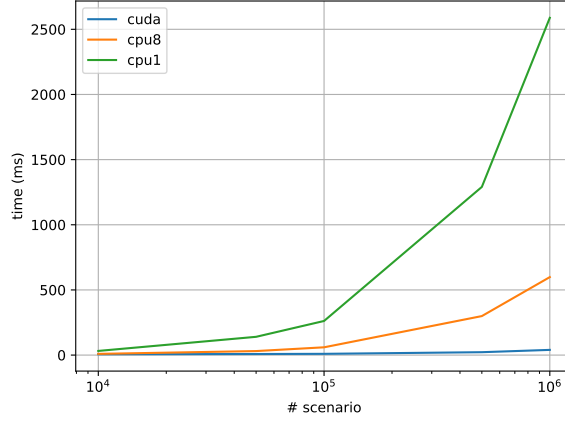

Figure 3: Runtime comparison for processing a splits evaluation under 10^4–10^6 demand scenarios. GPU parallelization scales almost linearly with the number of scenarios, while CPU runtimes grow super-linearly and quickly become prohibitive.

Results. On the CPU single-thread baseline, the runtime grows rapidly and exceeds several minutes when 10^6 scenarios are evaluated. The multi-threaded CPU implementation achieves moderate improvements, but scaling saturates due to synchronization and memory bandwidth bottlenecks. In contrast, the GPU implementation achieves near-linear scaling with the number of scenarios, maintaining runtimes within a few seconds even at 10^6 scenarios.

Overall, the GPU implementation provides *two to three orders of magnitude speedups* over CPU single-thread and more than one order of magnitude over CPU multi-threading. This validates our claim that GPU parallelism fundamentally shifts the computational frontier for scenario-based stochastic evaluation.

4.2 Impact of Training Scenario Set Size on Decision Quality

We next examine how the number of evaluated scenarios influences the quality of the first-stage decision. Specifically, we solve the problem under different scenario counts, ranging from 1 to 10^4 (i.e., 1, 100, 1,000). For each scenario setting, the obtained first-stage solution is evaluated on a fixed large out-of-sample test set of 10^6 scenarios. Figure 4 reports the out-of-sample cost achieved by the best observed solution on two CVRPSD instances: x-n128 and x-n105.
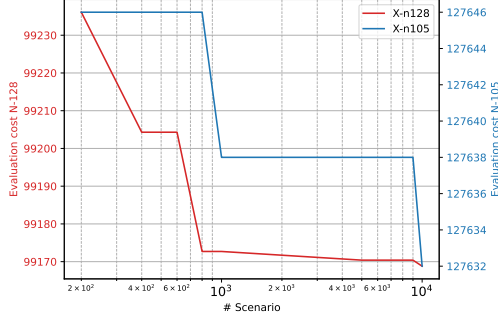
Figure 4: Out-of-sample performance of first-stage solutions obtained with varying observed scenario settings. Larger evaluation set yield more robust and lower-cost solutions.
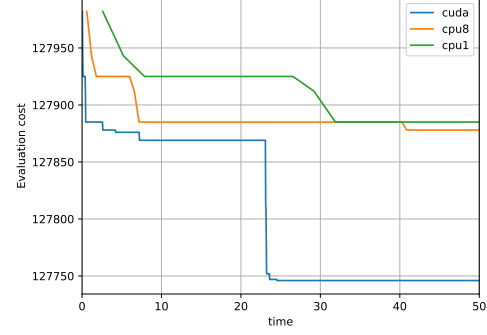

Figure 5: Quality of the best solution obtained at each time under a fixed time budget. GPU consistently achieves better decisions due to faster evaluation and thus larger effective search effort.

Results. When observing only few scenarios (e.g., a single scenario), the resulting first-stage solution is severely biased and performs poorly. Increasing the scale of available scenarios consistently improves robustness, with significant gains observed throughout Figure 4. Such performance confirms the theoretical insight that larger sample sizes reduce estimation bias in sample-average approximation. These results demonstrate that our GPU-based framework, by enabling the evaluation of tens or hundreds of thousands of scenarios within practical runtimes, leads to significantly more reliable first-stage solutions compared to CPU-based methods that are restricted to only a few thousand scenarios.

4.3 Decision Quality under Fixed Time Budgets

Finally, we compare the decision quality obtained under identical wall-clock time limits across the three implementations: CPU single-thread, CPU multi-thread, and GPU. Each method is given a fixed runtime budget , during which the split algorithm splits giant tours to obtain first-stage solutions. For fairness, all approaches are evaluated on the same problem instance with 10^4 available scenarios. Figure 5 reports the best penalized cost obtained within the allowed runtime.

Results. With small time budgets, all methods return feasible but suboptimal solutions, yet GPU already provides a noticeable advantage. As the time limit increases, the quality gap widens: GPU produces solutions that are consistently closer to the true optimum, while CPU single-thread stagnates and CPU multi-thread improves only modestly. This matches intuition: faster scenario evaluation allows the GPU to explore many more candidate first-stage tours within the same runtime, thereby improving the probability of discovering high-quality solutions.

These results confirm that beyond scalability, GPU acceleration directly translates into superior decision quality under realistic time constraints, making it particularly valuable in operational settings where decisions must be made quickly.

5 Conclusion

We proposed a GPU-accelerated framework for scenario-based stochastic combinatorial optimization, exemplified by the CVRPSD. By reformulating the split algorithm into a vectorized GPU-compatible form, our approach achieves orders-of-magnitude speedups over CPU baselines while enabling evaluation on ultra-large scenario sets. Experiments show that GPU parallelism not only scales efficiently but also yields more robust first-stage solutions and superior decision quality under fixed time budgets. These results demonstrate that GPU acceleration fundamentally advances the scalability and effectiveness of scenario-based stochastic programming, with broad applicability to logistics and other data-driven optimization domains.

6

6 Acknowledge

References

[1] Luca Accorsi and Daniele Vigo. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, 55(4):832–856, 2021.

[2] Arun L Bishop, John Z Zhang, Swaminathan Gurumurthy, Kevin Tracy, and Zachary Manchester. Relu-qp: A gpu-accelerated quadratic programming solver for model-predictive control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13285–13292. IEEE, 2024.

[3] Alexandre M Florio, Richard F Hartl, and Stefan Minner. New exact algorithm for the vehicle routing problem with stochastic demands. *Transportation Science*, 54(4):1073–1090, 2020.

[4] Ricardo Fukasawa and Joshua Gunter. The complexity of branch-and-price algorithms for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters*, 51(1):11–16, 2023.

[5] Charles Gauvin, Guy Desaulniers, and Michel Gendreau. A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands. *Computers & Operations Research*, 50:141–153, 2014.

[6] A. Goel, T. Vidal, and A.L. Kok. To team up or not: single versus team driving in european road freight transport. *Flexible Services and Manufacturing Journal*, pages 1–35, 2021.

[7] Chrysanthos E Gounaris, Wolfram Wiesemann, and Christodoulos A Floudas. The robust capacitated vehicle routing problem under demand uncertainty. *Operations research*, 61(3):677–693, 2013.

[8] Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

[9] Gilbert Laporte, François V Louveaux, and Luc Van Hamme. An integer l-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research*, 50(3):415–423, 2002.

[10] Felix Liu, Albin Fredriksson, and Stefano Markidis. A gpu-accelerated interior point method for radiation therapy optimization. *arXiv preprint arXiv:2405.03584*, 2024.

[11] Haihao Lu and Jinwen Yang. cupdlp. jl: A gpu implementation of restarted primal-dual hybrid gradient for linear programming in julia. *arXiv preprint arXiv:2311.12180*, 2023.

[12] Jorge E Mendoza, Bruno Castanier, Christelle Guéret, Andrés L Medaglia, and Nubia Velasco. Constructive heuristics for the multicompartment vehicle routing problem with stochastic demands. *Transportation science*, 45(3):346–363, 2011.

[13] Matheus J Ota and Ricardo Fukasawa. Hardness of pricing routes for two-stage stochastic vehicle routing problems with scenarios. *Operations Research*, 73(4):2177–2187, 2025.

[14] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[15] Walter Rei, Michel Gendreau, and Patrick Soriano. *Local branching cuts for the 0-1 integer L-shaped algorithm*. CIRRELT, 2007.

[16] Michel Schubiger. Gpu acceleration of admm for large-scale convex optimization. Master's thesis, ETH Zurich, 2019.

[17] Alexander Shapiro. Monte carlo sampling methods. *Handbooks in operations research and management science*, 10:353–425, 2003.

[18] Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1-3):487–512, 2002.

[19] T. Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[20] T. Vidal, M. Battarra, A. Subramanian, and G. Erdogan. Hybrid metaheuristics for the clustered vehicle routing problem. *Computers & Operations Research*, 58:87–99, 2015.

[21] T. Vidal, T.G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[22] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, 40(1):475–489, 2013.

[23] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673, 2014.

A   Monte Carlo Method Proposition

Empirical Risk Minimization (ERM) method is analogous to the Monte Carlo method for estimating a population mean via sample averages and fits naturally within the ERM framework for stochastic programming. Formally, we distinguish between:

- *True problem*:

$$(P) \quad z^* = \min_{x \in \mathbb{X}} \mathbb{E}[f(x, \tilde{\xi})],$$

- *Sample-average problem* with m scenarios:

$$(P_m) \quad z_m^* = \min_{x \in \mathbb{X}} \frac{1}{m} \sum_{i=1}^{m} f(x, \tilde{\xi}^i),$$

As the sample size m increases, the optimal solution x_m^* of the sample problem converges to the true optimal solution x^*, and the optimal value z_m^* approaches z^*. The following result summarizes the fundamental properties of the ERM method under mild regularity conditions (cf. [17]).

Proposition 1. *Let $\{\tilde{\xi}^1, \ldots, \tilde{\xi}^m\}$ be i.i.d. samples of $\tilde{\xi}$. Denote by*

$$(P) \quad z^* = \min_{x \in \mathbb{X}} \mathbb{E}[f(x, \tilde{\xi})], \qquad (P_m) \quad z_m^* = \min_{x \in \mathbb{X}} \frac{1}{m} \sum_{i=1}^{m} f(x, \tilde{\xi}^i),$$

the true and sample average problems, respectively, with optimal solutions x^* and x_m^*. Then:

$$\textit{(Bias)} \qquad \mathbb{E}[f(x_m^*, \tilde{\xi})] \leq z^*, \tag{4}$$

$$\textit{(Consistency)} \qquad \mathbb{E}[f(x_m^*, \tilde{\xi})] \xrightarrow{a.s.} z^* \quad \textit{as } m \to \infty, \tag{5}$$

$$\textit{(Probabilistic Convergence)} \qquad \lim_{m \to \infty} \Pr\left\{ \mathbb{E}[f(x_m^*, \tilde{\xi})] \leq z^* + \tilde{\epsilon}_m \right\} \geq 1 - \alpha, \quad \tilde{\epsilon}_m \downarrow 0, \tag{6}$$

$$\textit{(Rate of Convergence)} \qquad \sqrt{m}\left(z_m^* - z^*\right) \xrightarrow{d} \mathcal{N}\left(0, \sigma^2(x^*)\right). \tag{7}$$