**Step 9 — Stochastic VRPTW with SAA (Sample Average Approximation)**

**Abstract (1 paragraph)**

We extended the deterministic and quantile-buffer baselines by solving the VRPTW under travel-time uncertainty with Sample Average Approximation (SAA). For each instance we drew scenario samples that capture correlated traffic variability and searched for routes that minimize expected cost (distance plus a weighted tardiness term). Across the 56 Solomon-style CSVs, SAA delivered near-perfect on-time service (≈98–99%) at higher cost than the deterministic baseline and slightly higher than the 1.2× quantile method; the quantile method achieved ≈100% on-time at a lower cost than SAA, while the deterministic plans remained much cheaper but unreliable (≈40–45% on-time). The SAA results confirm that explicitly optimizing against scenarios produces robust routes and a clear, controllable cost–reliability trade-off.

**What we did**

**1) Turn uncertainty into scenarios (same generator as Step 7/8)**

* For each instance III we read your *original, untouched* CSV (C/R/RC families).
* We built a **baseline time matrix** from coordinates (Euclidean).
* We generated KKK travel-time **scenarios** with **correlated noise**:
  + **Global factor** Gs∼lognormalG\_s\sim\text{lognormal}Gs​∼lognormal (captures network-wide traffic swings).
  + **Link residuals** ϵij(s)∼lognormal\epsilon\_{ij}^{(s)}\sim\text{lognormal}ϵij(s)​∼lognormal (local variability).
  + Scenario time: Tij(s)=baseij⋅Gs⋅ϵij(s)T^{(s)}\_{ij} = \text{base}\_{ij}\cdot G\_s\cdot \epsilon\_{ij}^{(s)}Tij(s)​=baseij​⋅Gs​⋅ϵij(s)​.
* Parameters used during solving: cv\_global=0.20, cv\_link=0.10.

**2) Solve VRPTW with SAA**

* We implemented **scripts/vrptw\_saa.py** (SAA wrapper around your OR-Tools model and metaheuristics).
* Objective inside the search (per move/route):

score=Es[distance]  +  β⋅Es[tardiness]\text{score} = \mathbb{E}\_s[\text{distance}] \;+\; \beta\cdot \mathbb{E}\_s[\text{tardiness}]score=Es​[distance]+β⋅Es​[tardiness]

where β\betaβ (0–1 scale) **controls conservatism** (higher β⇒\beta\Rightarrowβ⇒ safer, costlier).

* Same hard constraints as before (capacities, time windows, service).
* Metaheuristics reused: **GLS** and **TABU**.
* Vehicle cost term was kept (--vehicle\_cost 10000) to penalize extra vehicles.

**3) Configurations we ran (your exact commands)**

You ran three SAA sweeps (on all 56 instances):

1. **Moderate SAA**
2. python scripts/vrptw\_saa.py --all --K 32 --seed 123 \
3. --cv\_global 0.20 --cv\_link 0.10 --beta 0.5 \
4. --time\_limit 30 --vehicle\_cost 10000 --meta GLS

→ wrote to data/solutions\_saa/k32\_b0p5/

1. **Less conservative / faster**
2. python scripts/vrptw\_saa.py --all --K 16 --seed 123 \
3. --cv\_global 0.20 --cv\_link 0.10 --beta 0.3 \
4. --time\_limit 30 --vehicle\_cost 10000 --meta GLS

→ wrote to data/solutions\_saa/k16\_b0p3/

1. **More conservative / slower (TABU)**
2. python scripts/vrptw\_saa.py --all --K 64 --seed 123 \
3. --cv\_global 0.20 --cv\_link 0.10 --beta 0.7 \
4. --time\_limit 60 --vehicle\_cost 10000 --meta TABU

→ wrote to data/solutions\_saa/k64\_b0p7/

Each folder contains:

* summary.csv (vehicles, distance, feasibility per instance)
* One JSON plan per instance (routes)

**4) Fair evaluation on common scenarios (like Step 8)**

You evaluated **DET**, **Q120** (quantile 1.2×) and SAA on the *same* 200 test scenarios:

python scripts/evaluate\_plans.py \

--dirs "data/solutions\_ortools" \

"data/solutions\_quantile/m1.2\_a0" \

"data/solutions\_saa/k32\_b0p5" \

"data/solutions\_saa/k16\_b0p3" \

--labels DET Q120 SAA32-b0p5 SAA16-b0p3 \

--K 200 --seed 42 --cv\_global 0.20 --cv\_link 0.10

This wrote:

* data/reports/step8\_eval.csv (instance-level on-time %, tardiness per method)
* data/reports/step8\_eval\_by\_method.csv (method-level averages)

**5) Step-9 summary plots**

With summarize\_step9.py you produced two figures (saved under data/figures/):

* **On-time by family & method** (boxplots)
* **Cost vs On-time (method level)** (scatter of means)

**What we got (headline results)**

The precise numbers are in your CSVs; below is the clear pattern seen in the plots and summaries.

* **Deterministic (DET)**
  + **On-time**: low (median ≈ 30–45% depending on family).
  + **Cost**: lowest (mean total distance ≈ *~1.03× baseline family means* in your scatter).
  + **Takeaway**: cheap but unreliable under variability.
* **Quantile 1.2× (Q120)**
  + **On-time**: ≈ **~100%** across all families (very tight spread).
  + **Cost**: moderate (≈ **+9%** vs DET in your scatter).
  + **Takeaway**: simple buffer already achieves near-perfect punctuality at moderate cost.
* **SAA (K=32, β=0.5)**
  + **On-time**: **≈98–99%**, very close to Q120.
  + **Cost**: highest of the three (**≈ +20–22%** vs DET; **~+10%** vs Q120).
  + **Takeaway**: explicitly optimizing over scenarios yields robust plans; with β=0.5 the solver accepted more distance to crush tardiness risk.
* **SAA (K=16, β=0.3)**
  + **On-time**: high (≫ DET, slightly below K=32/β=0.5).
  + **Cost**: between DET and SAA32-β0.5 (closer to Q120).
  + **Takeaway**: dialing β and K tunes the cost–reliability trade-off.
* **By family (C, R, RC)**
  + Under **DET**, C/R/RC show low medians with many outliers;
  + Under **Q120** and **SAA**, all three families cluster near the top (≈ 98–100% on-time), confirming robustness holds regardless of spatial pattern or horizon.

**Why we did Step 9**

* **Deterministic routing ignores variability**; it yields optimistic ETAs and frequent window violations when travel times fluctuate.
* **SAA** is a standard and credible way to **optimize against uncertainty**: we approximate the expected objective by averaging over realistic scenarios (with correlation), so the search prefers routes that remain feasible across many futures—not just the average one.
* **β (tardiness weight)** and **K (sample size)** give you **two intuitive knobs** to control conservatism and runtime.
* This step validates whether a principled stochastic method beats (or complements) the simpler **quantile-buffer** approach.

**Files produced in Step 9**

**Solutions**

* data/solutions\_saa/k16\_b0p3/ → JSONs + summary.csv
* data/solutions\_saa/k32\_b0p5/ → JSONs + summary.csv
* data/solutions\_saa/k64\_b0p7/ → JSONs + summary.csv

**Evaluation (reused Step 8 evaluator)**

* data/reports/step8\_eval.csv (instance × method metrics; overwritten with the latest run)
* data/reports/step8\_eval\_by\_method.csv (method means; overwritten)

**Plots (Step 9)**

* data/figures/step9\_ontime\_by\_family.png (your boxplot)
* data/figures/step9\_cost\_vs\_ontime.png (your scatter)

*(If your script named the figures differently, keep your actual filenames in the report.)*

**How to describe it in your thesis/report**

**Method.** “We solved VRPTW under uncertainty using Sample Average Approximation. Each candidate route is scored by expected distance plus a weighted expected tardiness computed over KKK correlated travel-time scenarios (global network factor + link residuals; lognormal noise with CV=0.20/0.10). We tested K∈{16,32,64}K\in\{16,32,64\}K∈{16,32,64} and β∈{0.3,0.5,0.7}\beta\in\{0.3,0.5,0.7\}β∈{0.3,0.5,0.7}, with OR-Tools (GLS/TABU) and a 30–60 s time limit.”

**Evaluation.** “All methods—Deterministic (DET), Quantile 1.2× (Q120), and SAA—were evaluated on the same 200 out-of-sample scenarios (common random numbers) and compared on on-time %, average tardiness, #vehicles, and total distance.”

**Results.** “DET is cheapest but unreliable (≈40–45% on-time). Q120 achieves ≈100% on-time with ≈+9% distance vs DET. SAA (K=32, β=0.5) achieves ≈98–99% on-time but with ≈+20% distance vs DET; tuning β and K reduces cost at some on-time loss. Family-wise, robust methods lift all C/R/RC groups near 100% on-time.”

**Conclusion.** “Both robust approaches work; if we want the **best cost at ≥99% on-time**, the **quantile 1.2×** method is an excellent, simple choice. **SAA** provides a principled framework with tunable conservatism; it is valuable when we want explicit control via β and when quantile inflations become hard to set uniformly.”

**(Optional) Next micro-steps**

* Add SAA **comparison at β=0.4** with K=32 (often a sweet spot).
* Re-run **evaluation** including k64\_b0p7 to complete the trade-off curve.
* Export a **single “champion” per instance** (best method given your SLA target, e.g., ≥99% on-time with minimal distance).

Second version :  
**Step 9 — Stochastic optimization (SAA), fair evaluation, and “champion” plan selection**

**Why this step**

Until now we had:

* a deterministic OR-Tools baseline (DET),
* a **quantile-buffer** method (Q120) from Step 8 that inflates travel times to guard against traffic variability.

Those are robust, but still *heuristics*. In Step 9 we introduce a **stochastic optimizer** that *sees scenarios while it builds the routes* (Sample Average Approximation, SAA). We also evaluate **all methods under the same randomness** and then **pick the best plan per instance** that achieves a target service level (e.g., ≥ 99% on-time).

**What we implemented**

1. **SAA solver** (scripts/vrptw\_saa.py)
   * Reads each original Solomon CSV (unchanged).
   * Builds a set of travel-time scenarios with correlation (global traffic factor + per-arc residual noise).
   * Scores moves by a **risk-aware objective**  
     objective = expected distance + vehicle\_cost × vehicles + β × expected tardiness.
   * Parameters you used:
     + Scenarios inside the solver: K ∈ {16, 32, 64}
     + Risk weight β ∈ {0.3, 0.5, 0.7}
     + Metaheuristics: GLS and TABU (same primitives as in Step 6)
     + Time limits: 30–60s per instance.
2. **Common, fair evaluation** (scripts/evaluate\_plans.py)
   * Uses **the same 100/200 test scenarios** for every plan (DET, Q120, SAA…) → apples-to-apples.
   * Metrics per instance & method: mean on-time %, p05/p50/p95 on-time, mean tardiness.
3. **Result summarization** (scripts/summarize\_step9.py)
   * Plots **on-time by family & method** and **cost vs on-time (method level)**.
4. **Champion picker** (scripts/pick\_champions.py)
   * For each instance, selects the **lowest-cost plan that reaches the target on-time** (you used 99%).
   * Copies the winning JSON to data/champions/ and writes summary tables and a cost-vs-on-time figure.

**How we ran it (the exact commands you used)**

* Medium SAA (balanced risk)

python scripts/vrptw\_saa.py --all --K 32 --seed 123 --cv\_global 0.20 --cv\_link 0.10 \

--beta 0.5 --time\_limit 30 --vehicle\_cost 10000 --meta GLS

* Less conservative SAA (cheaper, still robust)

python scripts/vrptw\_saa.py --all --K 16 --seed 123 --cv\_global 0.20 --cv\_link 0.10 \

--beta 0.3 --time\_limit 30 --vehicle\_cost 10000 --meta GLS

* More conservative SAA (safer, costlier)

python scripts/vrptw\_saa.py --all --K 64 --seed 123 --cv\_global 0.20 --cv\_link 0.10 \

--beta 0.7 --time\_limit 60 --vehicle\_cost 10000 --meta TABU

* Fair evaluation (DET, Q120, SAA runs together)

python scripts/evaluate\_plans.py \

--dirs "data/solutions\_ortools" \

"data/solutions\_quantile/m1.2\_a0" \

"data/solutions\_saa/k32\_b0p5" \

"data/solutions\_saa/k16\_b0p3" \

"data/solutions\_saa/k64\_b0p7" \

--labels DET Q120 SAA32-b0p5 SAA16-b0p3 SAA64-b0p7 \

--K 200 --seed 42 --cv\_global 0.20 --cv\_link 0.10

* Pick champions at 99% SLA

python scripts/pick\_champions.py \

--dirs "data/solutions\_ortools" \

"data/solutions\_quantile/m1.2\_a0" \

"data/solutions\_saa/k16\_b0p3" \

"data/solutions\_saa/k32\_b0p5" \

"data/solutions\_saa/k64\_b0p7" \

--labels DET Q120 SAA16-b0p3 SAA32-b0p5 SAA64-b0p7 \

--target 99

**What files were produced**

* Per SAA run  
  data/solutions\_saa/k16\_b0p3/summary.csv (and .json per instance)  
  data/solutions\_saa/k32\_b0p5/summary.csv  
  data/solutions\_saa/k64\_b0p7/summary.csv
* Common evaluation  
  data/reports/step8\_eval.csv (instance-level metrics for all methods)  
  data/reports/step8\_eval\_by\_method.csv (method-level means)
* Step 9 visual summaries  
  data/figures/step9\_family\_method\_ontime.png *(On-time by family & method)*  
  data/figures/step9\_cost\_vs\_ontime\_methods.png *(Cost vs on-time, method level)*
* Champions (target 99% on-time)  
  data/reports/champions.csv *(winner per instance: instance, method, on-time, distance, vehicles, …)*  
  data/reports/champions\_stats\_by\_method.csv *(means for the champions subset)*  
  data/figures/champions\_cost\_vs\_ontime.png *(your scatter plot)*  
  **Final plans** copied to data/champions/ (one JSON per instance).

**What we learned (key results)**

* **Deterministic plans collapse under variability.**  
  In your boxplot, DET shows ~20–50% on-time across families, with many late deliveries.
* **Quantile-buffer (Q120) is a strong baseline.**  
  Near-universal ~99% on-time with a **moderate distance increase** versus DET. This validates Step 8.
* **SAA works—and you can tune it.**
  + SAA16-b0p3 (lighter risk) achieved **very high on-time** while keeping **cost close to baseline** in many instances.
  + SAA32-b0p5 and SAA64-b0p7 are more conservative; they push on-time close to 99–100% but typically at **higher distance**.
* **Champion selection @ 99% SLA** (your champions\_cost\_vs\_ontime.png):
  + For the instances where multiple methods hit ≥ 99% on-time, **SAA16-b0p3** often delivered the **lowest mean distance**, making it a frequent champion.
  + **Q120** is the **simplest and very competitive** fallback—slightly more cost than SAA16-b0p3 in your champions set, but extremely reliable.
  + Heavier SAA settings (K=32/64, β high) are **safest** but tend to be **costlier**; they’re useful if you must protect against extreme variability.

Overall message: *Compared to deterministic routing, both Q120 and SAA eliminate lateness. Among them, SAA16-b0p3 strikes the best cost–service trade-off in your runs; Q120 is a simple, near-optimal fallback.*

**How to read the figures**

* **On-time by family & method**: every label “Family | Method” shows the distribution across the 56 instances.  
  You can see DET far lower, while Q120 and SAA boxes are tight near the top (≈ 99–100%).
* **Cost vs on-time (method level)**: each dot is a method; right/up is better service, left is cheaper.  
  Q120 sits near the top with moderate cost; SAA32/64 are further right (safer) but also further right on distance; SAA16-b0p3 stays high on on-time while closer to the left (cheaper).
* **Champions scatter**: only the **winning** plans per instance (meeting ≥ 99%).  
  Cluster of SAA16-b0p3 to the **left-top** (cheap + high on-time) confirms it’s the best default.

**Executive abstract (Step 9 — one paragraph)**

We introduced a stochastic optimizer (SAA) that builds VRPTW routes using scenario-based travel times, and we compared it fairly against our deterministic baseline and the Step 8 quantile-buffer method. Using common random numbers (200 evaluation scenarios), the deterministic plans achieved only ~20–50% on-time, while **Q120 and SAA reach ≈ 99–100%**. Among SAA settings, **SAA16-b0p3** provided the **best cost–service trade-off**, frequently winning the champion selection at a **99% on-time target**; **Q120** remained a strong, simple alternative. Final champion plans (JSON per instance) are saved in data/champions/ with their summaries and plots.

**What’s next (to close the project convincingly)**

* **Fix a final method choice** for deployment (recommend SAA16-b0p3, with Q120 as fallback).
* **Produce the final comparison table** for the report appendix: per instance → (cost, vehicles, on-time p50/p95, runtime).
* **Write the 2–3 page discussion**: why stochastic beats deterministic, sensitivity to K and β, and practical guidance (when to use Q-buffer vs SAA).
* **Freeze artifacts**: keep the champions folder + all CSVs/figures under version control.