Measuring "Fastness" of AVA vs. Baselines

Scope

Compare AVA against a baseline (e.g., Lambert+filter or a shape-based prefilter) on: (1) per-candidate latency, (2) sweep wall-clock to top-K, and (3) solver convergence from seeds.

Metrics

- **Per-candidate latency** t_{cand} (ms): median time to score one candidate (e.g., one date).
- **Throughput** cands/s $= 1/t_{\text{cand}}$.
- **Sweep time** T_{sweep}: end-to-end wall-clock to evaluate N candidates and return top-K.
- **Speedup** $S = T_{\text{baseline}}/T_{\text{AVA}}$ (report for both t_{cand} and T_{sweep}).
- **Convergence (seed quality)**: iterations and wall-clock for the high-fidelity solver to converge from each seed.
- **Quality-adjusted speed** (optional): time to reach a solution within 1% of the best Δv.
- **Recall@K**: fraction that the final best solution appears in the prefilter's top-K list.

Benchmark Protocol (fair & repeatable)

1. Fix cases (e.g., LEO→TLI with a small plane change; GTO→GEO apogee tweak) and the sweep grid (dates, step).
2. Same machine and settings for all runs; pin BLAS/threads to 1; warm up (e.g., 50 dummy calls).
3. Measure per-candidate latency over ≥ 1000 candidates; report median and p90.
4. Run full sweep to gather top-K candidates; record T_{sweep}.
5. Feed each method's top-K to the same high-fidelity solver (GMAT/EMTG/IPOPT); record iterations and wall-clock to convergence.

6. (Optional) Monte-Carlo: thrust $\pm 2\%$, pointing $\pm 0.5^\circ$, nav errors; confirm results are robust.

Pseudocode: AVA Candidate Scoring (Impulsive)

```
start = now()
for x in candidates:                    # e.g., dates or C3 grid
    g  = goal_vector(state0, target(x))     # v_req - v_now in RTN
    G,N = metric_and_constraints(state0)    # choose per-leg
    P  = projector(G,N)                     # G-orthogonal (Cholesky/QR)
    Pg = P * g
    if norm_G(Pg) == 0: continue
    dv = rho * Pg / norm_G(Pg)              # best impulsive arc
    score = cost(dv)                        # e.g., |dv|, arrival error
    record(score, x)
t_cand = (now() - start) / len(candidates)
```

Pseudocode: Lambert Baseline (Comparable Skeleton)

```
start = now()
for x in candidates:
    r1,v1 = state_at_epoch(state0, x)       # parking state
    r2,v2 = target_state(x)                 # target conic/state
    dv1,dv2, tof = lambert(r1,r2, mu)       # two-impulse
    score = cost(dv1, dv2)                  # same cost function policy
    record(score, x)
t_cand_baseline = (now() - start) / len(candidates)
```

Seed Convergence Test (High-Fidelity)

For each case, export the top-K candidates from AVA *and* the baseline to the same nonlinear program (collocation/pseudospectral or multiple shooting). Record: iterations to tolerance; solver wall-clock; final Δv and constraint margins.

Result Tables (fill in after runs)

Per-candidate latency

Method	Median t_{cand} (ms)	p90 (ms)	cands/s	Speedup S
AVA				
Baseline (Lambert)				

Sweep wall-clock to top-K

Method	N candidates	T_{sweep} (s)	Speedup S
AVA			
Baseline (Lambert)			

Solver convergence from seeds (top-K)

Seed	Mean iterations	Median iterations	Mean time (s)	Final Δv (km/s)
AVA seed				
Lambert seed				

Quality-adjusted speed & Recall

Method	Time to within 1% of best (s)	Recall@K
AVA		
Baseline (Lambert)		

Reporting Tips

Always state CPU, cores/threads, compiler/interpreter version, and whether JIT or caching was used. Report medians and p90 (not only means). Include the cost function used for ranking (e.g., Δv, arrival error, or weighted sum).