Language Models as Optimizers: Towards a Framework for Combinatorial Optimization

Mohand Mezmaz, PhD, HDR

SnT, Univ. of Luxembourg

NEURESA Workshop

Outline

Most Optimisation Methods Build a Tree

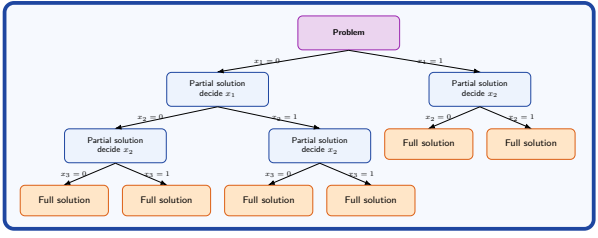
Metaheuristics

- Greedy / Randomized greedy (GRASP)
- Ant Colony Optimisation (ACO)
- Particle Swarm Optimisation (PSO)
- Beam Search
- ...

Exact methods

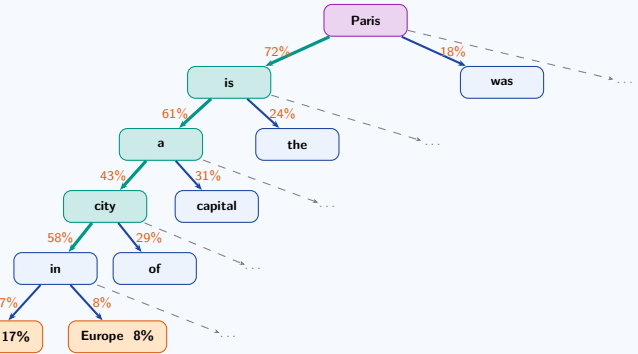
- Branch & Bound
- Branch & Cut
- Constraint Programming
- Dynamic Programming
- ...

Key observation: The vast majority of resolution methods build a **decision tree** (= a Markov Decision Process).



GPT-2 style decoder-only Transformer (Tree over tokens)

Prompt: *"Paris"*



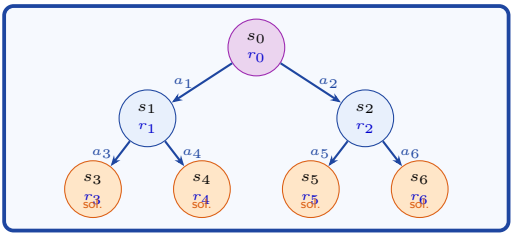
Answer: *"Paris is a city in ____"*

Reinforcement & Imitation Learning (Also Tree-Based)

The MDP View (with Policy Updates)

An agent traverses the tree to update its policy π:

1. **Observe** state s_t
2. **Select** action $a_t \sim \pi(s_t)$
3. **Transition** to s_{t+1} and **receive** r_t
4. **Update policy π using rewards**
   - ↪ **RL (Trial & Error):** shifts probability toward actions that yielded higher *cumulative* tree rewards (e.g., via Policy Gradients).
   - ↪ **IL (Imitation):** shifts probability to minimize the distance between $\pi(s_t)$ and an expert's target choice at that node.



The Decision-Making Analogy

	Operations Research	Machine Learning		
Decision Tree	**Optimisation**	**NLP**	**RL**	**IL**
Programmer (Builder)	Expert (hand-crafted)	Transformer	Agent	
Program (Tree)	Solver	LLM	Policy π	
Decision (Step)	Var. (Job, City, ...)	Token	Action	
Leaf (Outcome)	Solution	Text	Trajectory	
Quality	Cost	Likelihood	Cumulative reward	Distance to expert
Randomness (Exploration)	Explore/exploit	Top-k / Temp.	Stochastic policy	

Outline

Our Objectives

Replace Optimisation with Learning

- Use a **Language Model** instead of traditional heuristics
  - Minimises the need for **hand-crafted, domain-specific rules**

Hybridise Optimisation & Learning

- Combine LM-based generation with **optimisation techniques or solvers**
  - Use high-quality solver's solutions to **train or fine-tune** the LM

Along the Way: COLM

Developing these hybrid methods, we open-sourced **COLM**: a lightweight framework coupling **Transformers** with **metaheuristics** (**GRASP**, Local Search) for combinatorial optimisation.

- Already validated on **GRASP**
- Also tested on NSGA and Memetic Algorithm
- PSO integration **in progress**

`https://gitlab.com/uniluxembourg/snt/pcog/colm`

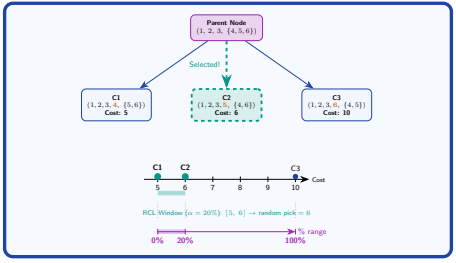
GRASP: Construction, Local Search & Limitations

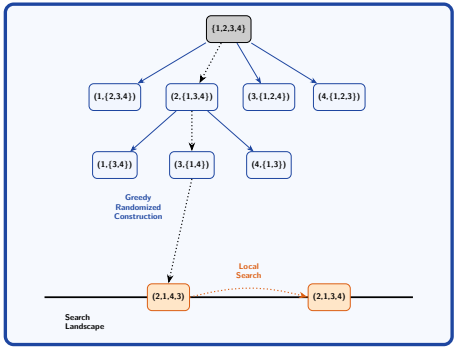
GRASP(α)

while not (stopping criterion) **do**

- $\pi_{cur} \leftarrow$ greedy_randomized_construction($\emptyset$)
- $\pi_{cur} \leftarrow \text{LS}(\pi_{cur})$
- $\pi_{best} \leftarrow \text{BEST}(\pi_{best}, \pi_{cur})$

return π_{best}





Limitations

- **Myopic** — local scalar metric only
- **Memoryless** — ignores past solutions
- **Hand-crafted** — requires domain expertise

LM-GRASP(α)

Phase 1 — Bootstrap

- Archive $\leftarrow$ random_permutations(N_{boot})
- Archive $\leftarrow$ evaluate(Archive, f)
- *no domain knowledge used*

Phase 2 — Learn–Infer–Improve

while not (stopping criterion) **do**

(a) $\pi_\theta \leftarrow$ **train**(Archive) (behavioral cloning)

(b) $\mathcal{C} \leftarrow$ **infer**(π_θ, M) (autoregressive sampling)

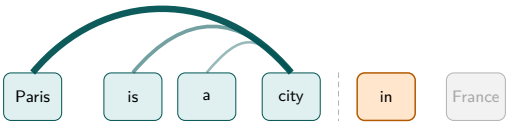
(c) $\mathcal{C} \leftarrow$ **filter**($\mathcal{C}$) (invalid / duplicate)

(d) $\mathcal{C} \leftarrow$ **refine**($\mathcal{C}$) (local search)

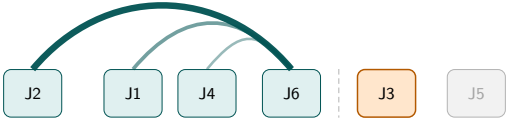
(e) Archive $\leftarrow$ **update**(Archive, $\mathcal{C}$) (evict worst)

return best(Archive)

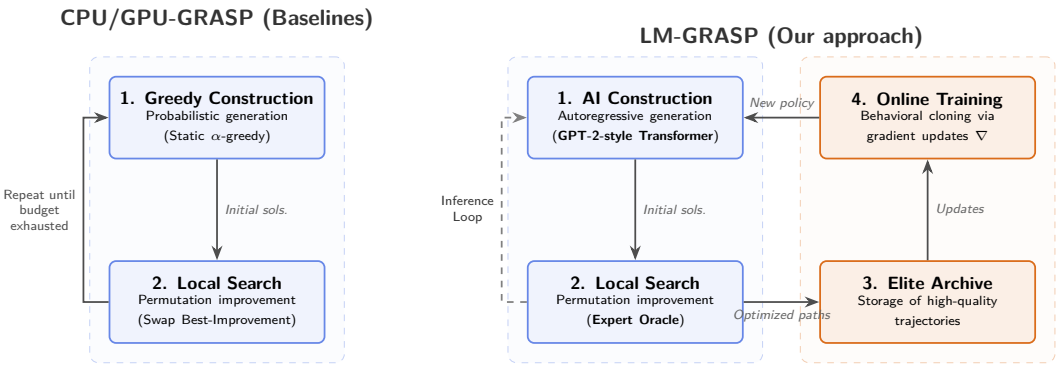
General LLM Attention:



Flowshop LM Attention:



LM-GRASP


CPU/GPU-GRASP (Baselines)
1. Greedy Construction
Probabilistic generation
(Static α-greedy)
Initial sols.
2. Local Search
Permutation improvement
(Swap Best-Improvement)
Repeat until budget exhausted
LM-GRASP (Our approach)
1. AI Construction
Autoregressive generation
(GPT-2-style Transformer)
Initial sols.
2. Local Search
Permutation improvement
(Expert Oracle)
Inference Loop
New policy
4. Online Training
Behavioral cloning via gradient updates ∇
Updates
3. Elite Archive
Storage of high-quality trajectories
Optimized paths

Outline

PFSP: Permutation Flow-Shop Scheduling

Problem Definition

N jobs to be scheduled on M machines

- Each job has a processing time for each machine

Constraints

- A machine cannot be simultaneously assigned to more than one job
- The order of the jobs is the same on all machines

The objective is to minimize the makespan

- The end date of the last job on the last machine

Cost of a permutation FS $(1, 3, 4, 5, 2)$

Flow-Shop Standard Instances

Benchmark [Taillard 1993]**:**

- `ta51`–`ta60` (10 instances in total)
  - 50 jobs $\times$ 20 machines
- 10 runs each instance
- 5 hours each run

Results: Best Makespan Found (1/3)

Instance	Minimum Makespan ($\min$)			Best	Is it	Win Count (↑)		
	CPU-GRASP	GPU-GRASP	LM-GRASP	known	Opt.?	CPU-GRASP	GPU-GRASP	LM-GRASP
ta51	3950	3927	**3898**	3846	—	0	0	**10**
ta52	3806	3777	**3751**	3699	✓	0	0	**10**
ta53	3758	3731	**3693**	3640	✓	0	0	**10**
ta54	3823	3795	**3769**	3719	—	0	0	**10**
ta55	3719	3706	**3662**	3610	—	0	0	**10**
ta56	3775	3757	**3725**	3679	✓	0	0	**10**
ta57	3810	3784	**3760**	3704	✓	0	0	**10**
ta58	3812	3789	**3753**	3691	✓	0	0	**10**
ta59	3856	3822	**3797**	3741	—	0	0	**10**
ta60	3832	3821	**3811**	3755	—	0	0	**10**
Average / Total	3814	3790	**3761**	3708		0	0	**100**

Key observations:

- LM-GRASP achieves the best makespan on **all 10 instances**
- Algorithmic gain: $+28.4$ **units** (GPU vs. LM)
- Hardware gain: $+27.2$ units (CPU vs. GPU)
- Two deltas of **comparable magnitude**

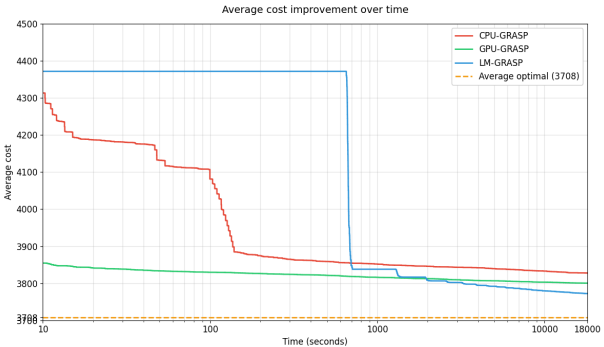
Results: Mean Makespan per Run (2/3)

		Average Makespan Per Method			Pairwise Gap Evolution	
Instance	**#Runs**	**CPU-GRASP**	**GPU-GRASP**	**LM-GRASP**	**CPU/GPU**	**GPU/LM**
ta51	10	3958.6 ± 05.8	3933.3 ± 03.7	**3907.4** ± 04.1	25.1 ± 06.2	26.0 ± 04.9
ta52	10	3820.1 ± 08.4	3787.4 ± 08.0	**3758.8** ± 05.0	37.3 ± 12.0	27.8 ± 10.4
ta53	10	3774.9 ± 09.1	3740.7 ± 05.9	**3710.3** ± 09.7	31.1 ± 12.5	29.7 ± 12.9
ta54	10	3831.2 ± 06.6	3802.4 ± 03.8	**3776.6** ± 04.5	28.0 ± 10.1	26.0 ± 06.5
ta55	10	3740.9 ± 13.4	3714.6 ± 05.5	**3675.4** ± 07.3	24.4 ± 16.0	39.3 ± 10.8
ta56	10	3789.5 ± 10.1	3768.7 ± 07.2	**3739.7** ± 06.6	21.3 ± 14.7	28.3 ± 07.8
ta57	10	3822.4 ± 07.2	3797.6 ± 07.1	**3767.4** ± 04.0	25.0 ± 12.7	30.7 ± 08.1
ta58	10	3825.6 ± 09.9	3793.3 ± 04.3	**3761.7** ± 04.8	32.4 ± 10.0	31.0 ± 08.4
ta59	10	3864.1 ± 05.5	3836.1 ± 06.2	**3801.8** ± 03.2	27.4 ± 05.9	34.2 ± 06.7
ta60	10	3849.0 ± 09.1	3830.2 ± 06.5	**3818.8** ± 05.0	19.6 ± 09.9	11.2 ± 07.2
All instances	100	3827.6 ± 08.5	3800.4 ± 05.8	**3771.8** ± 05.4	27.2 ± 11.0	28.4 ± 08.4

Key observations:

- LM-GRASP wins on **all 100 individual runs**

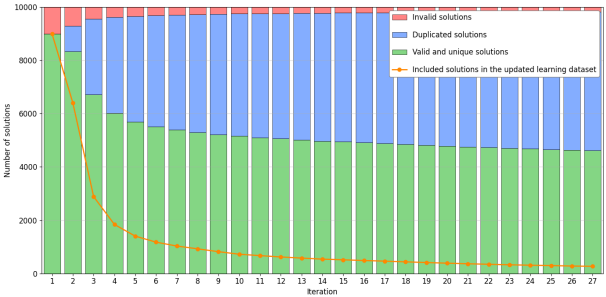
Results: Convergence Trajectories Over Time (3/3)



Key observations:

- GPU-GRASP plateaus within **first 30 seconds**
- LM-GRASP invests ≈**660s** in bootstrapping
- LM-GRASP improves **steadily** throughout budget
- **No plateau** at budget exhaustion

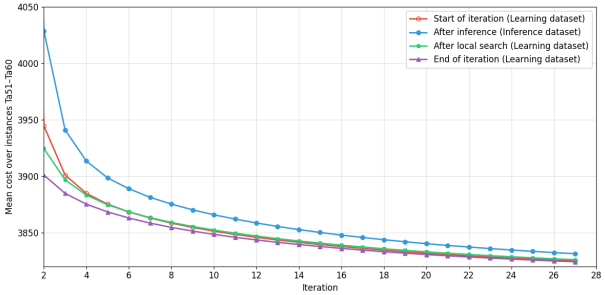
Results: Candidate Batch Breakdown per Iteration (1/2)



Key observations:

- Archive-improving candidates:
  - $\approx$**8,986** $\rightarrow$ $\approx$**299**
- Duplicates increase as search **converges**
- Natural signature of exhausting high-quality regions

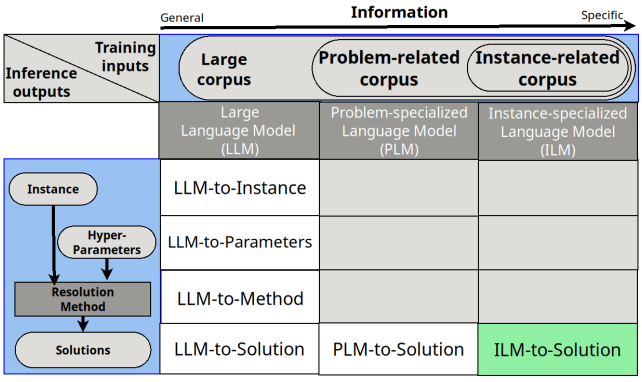
Results: Intra-Iteration Policy Generalization (2/2)



Key observations:

- Policy **generalizes beyond** its training distribution
- Inference quality exceeds training corpus from iteration 5
- End-of-iteration values:
  - $\approx 3900 \rightarrow \approx 3824$
- **Stable convergence** confirmed

Outline



Existing approaches require:

- Millions of samples (Chin et al. 2024: 38.1M pairs)
- Fixed problem and size (Kool 2019, POMO 2020)
- Or rely on zero-shot LLM generalization

LM-GRASP

- **No external data.**
- Training corpus for the active instance.
- No problem-specific features.

We introduced LM-GRASP

- Replaces hand-crafted GRASP constructor
  - with an **online-trained language model**
- Trained via imitation learning
  - on self-generated elite solutions
- No domain knowledge, no external data, no pretraining

Key result on Taillard `ta51–ta60`

- Algorithmic gain ($+28.4$ units)
  - comparable to hardware acceleration ($+27.2$ units)
- Wins on **all 100 individual runs**
- No plateau at budget exhaustion

Conclusions

Instance-specific language models:

- a **practical/competitive alternative** ...
- ... to hand-engineered constructors
- ... in metaheuristic search.

Future work

- Cross-domain validation (TSP, bin-packing, ...)
- Continuous policy updates
- Systematic hyperparameter optimization