Metaheuristics based on local search and populations for the Delivery Scheduling problem

Guillem Rodríguez Corominas[1], Albert López Serrano[1], Roberto Maria Rosati[2], Mehmet Anil Akbay[1], Christian Blum[1]

(1) Artificial Intelligence Research institute (IIIA), CSIC, Bellaterra, Spain

(2) University of Udine, Udine, Italy

Outline

Outline for section 1

Informal Problem Description

- $\mathcal{A}$ asteroids, $\mathcal{M}$ materials, $\mathcal{S}$ processing stations
- Deliver asteroids in $\mathcal{A}$ to the stations in $\mathcal{S}$, within a scheduling horizon T_{max} days.
- $\mathcal{O}$: finite set of delivery opportunities, $o \in \mathcal{O} : (a, s, t, m_1, ..., m_n)$

Constraints

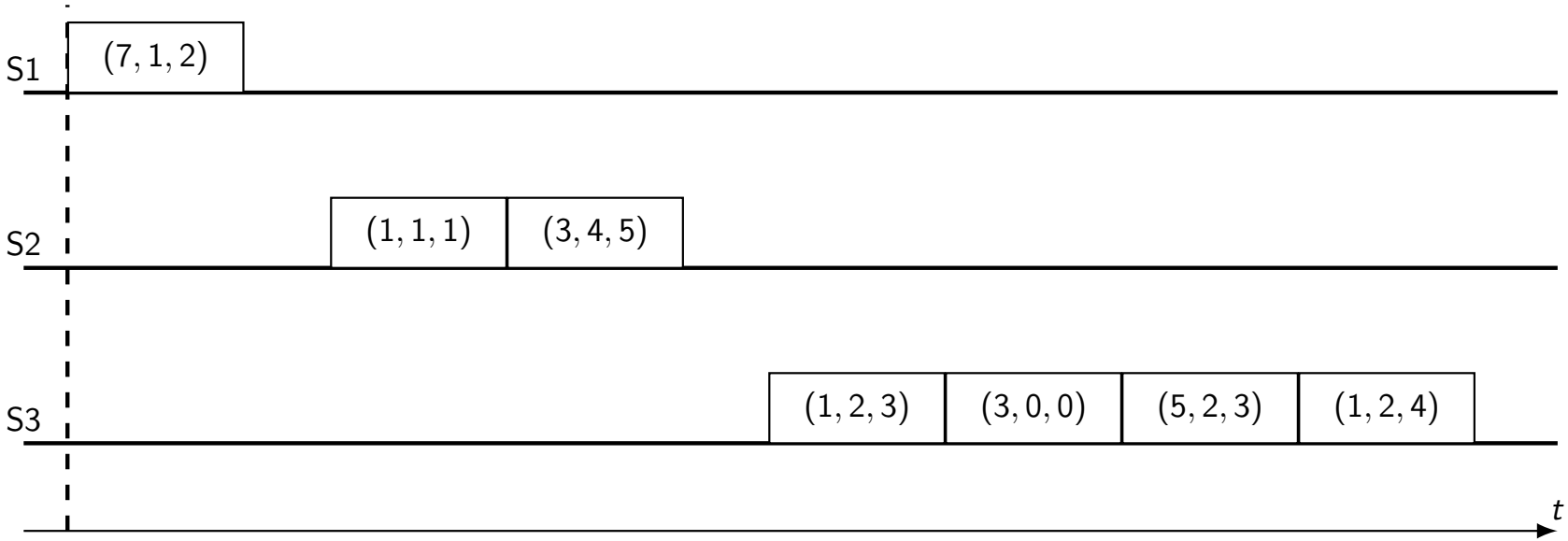
- Each asteroid is delivered at most once
- Each processing station can only become active for one time interval.
- Only one processing station can be active at a time.
- Stations can only receive asteroids during their active windows.
- Stations active windows must be separated by at least one day.
- Asteroids delivered to a given station must have arrival times included in the station's activity window.

Objective

Maximize the minimum mass collected among all processing stations and among all materials

$$\max \left(\min_{i \in \mathcal{S}, j \in \mathcal{M}} q_{ij} \right)$$

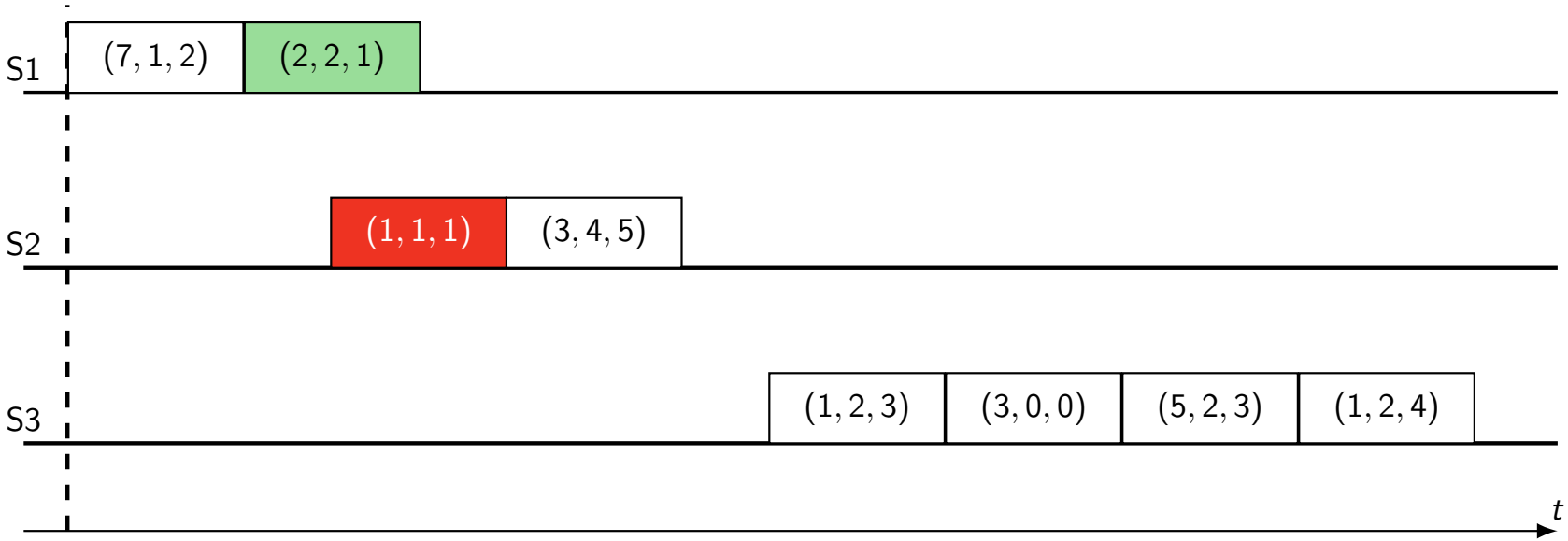
Example 1



	m1	m2	m3
S1	7	1	2
S2	4	5	6
S3	10	6	10

$$o.f. = 1$$

Example 2



	m1	m2	m3
S1	9	3	3
S2	3	4	5
S3	10	6	10

$$o.f. = 3$$

Outline for section 2

Solution representations

Binary vector

1	0	0	1	0	1	1	0	0	1	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12

Vector of inserted opportunities

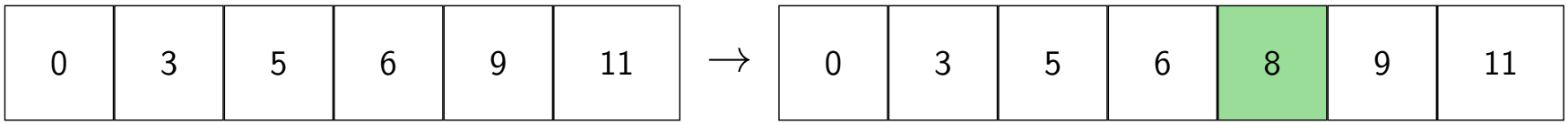
0	3	5	6	9	11

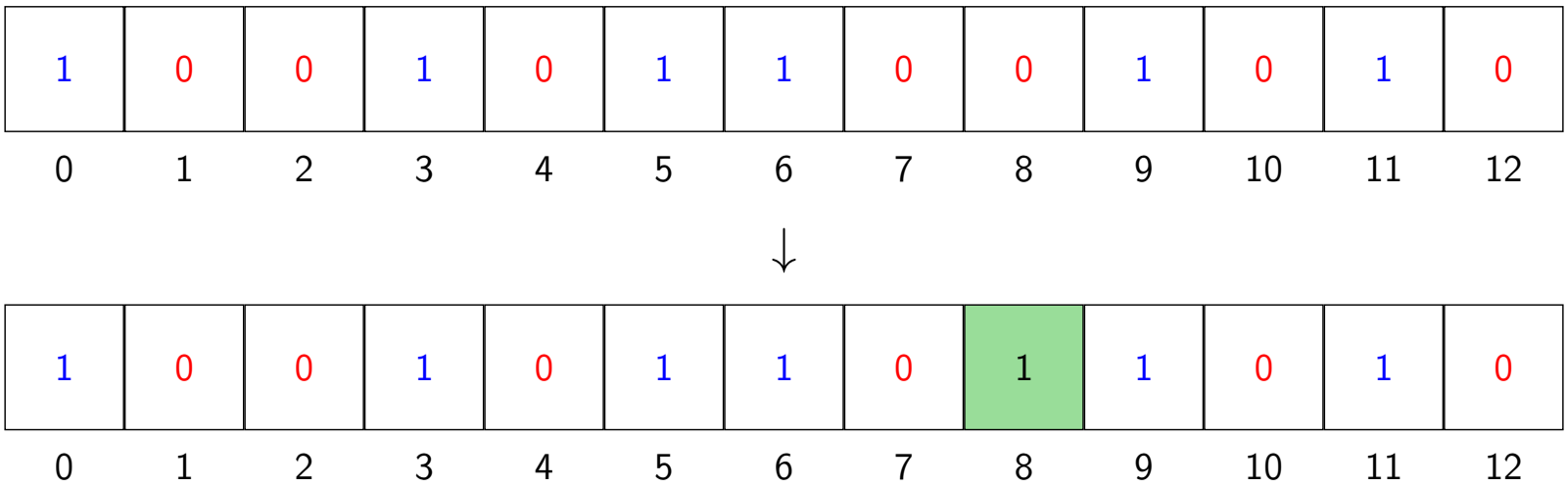
Outline

Local Search Neighborhoods

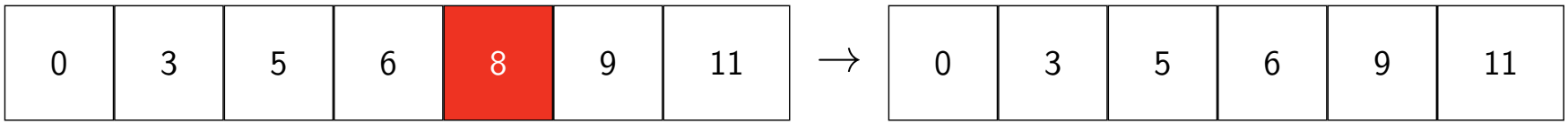
- Problem-agnostic neighborhoods:
  - **Insert**: insert a new opportunity in the solution
  - **Remove**: remove an opportunity from the solution
  - **Swap**: swap an opportunity in the solution with an opportunity not in the solution
- Problem-specific neighborhoods:
  - ...

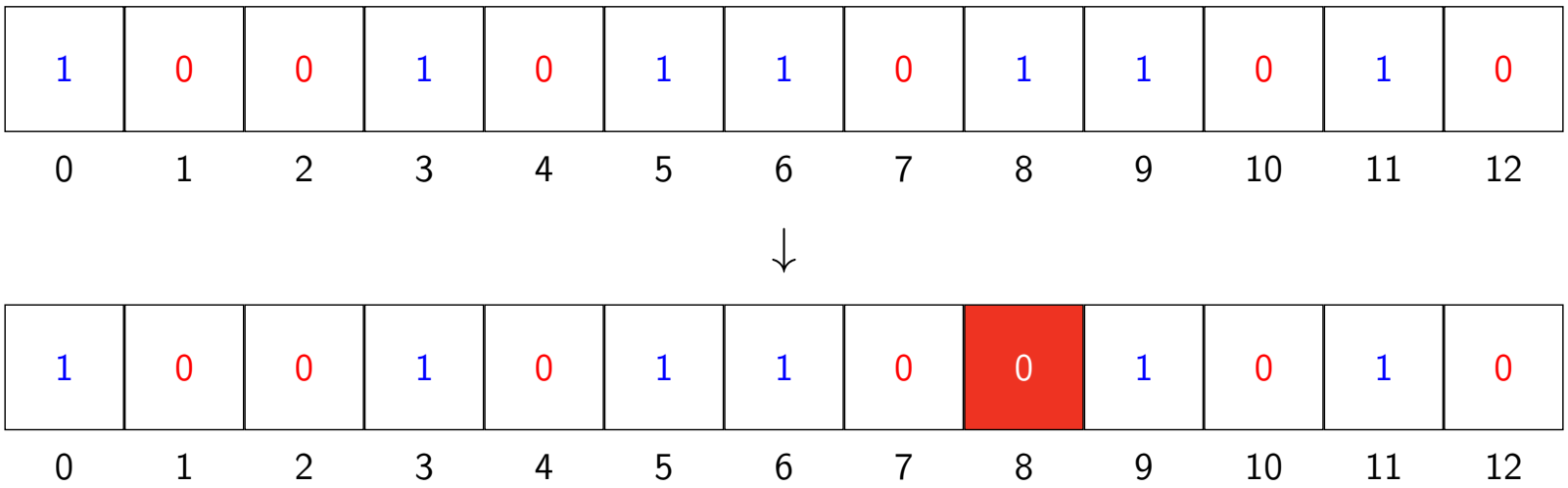
Neighborhoods — Insert



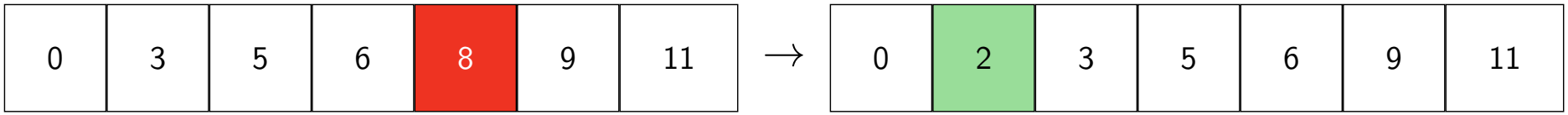


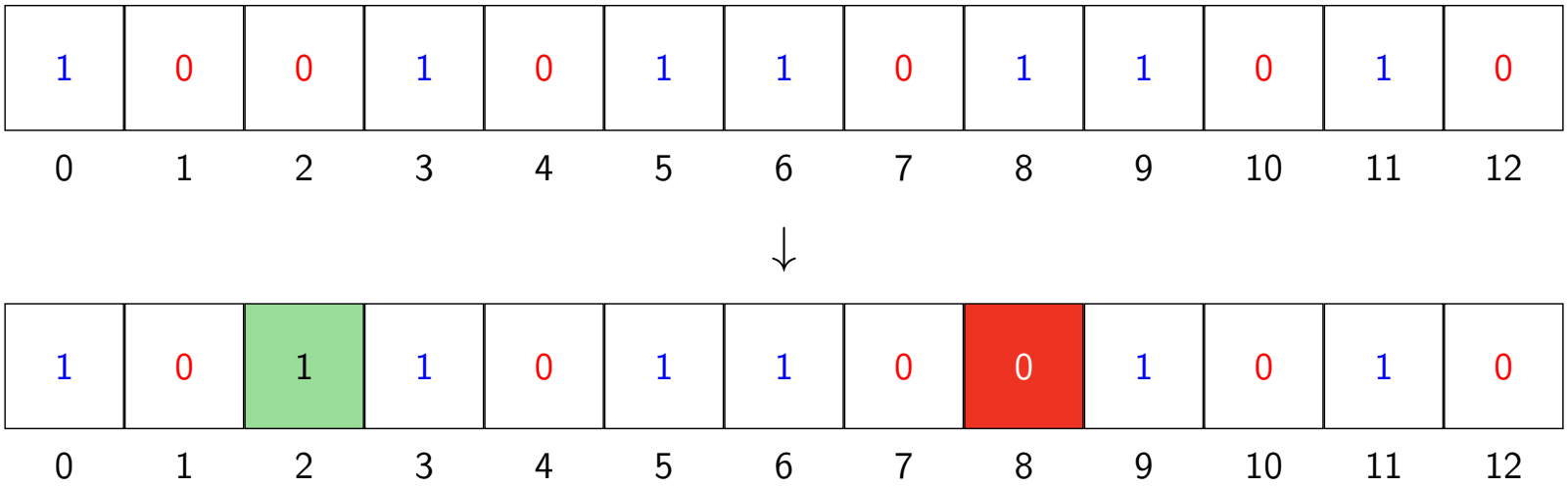
Neighborhoods — Remove





Neighborhoods — Swap

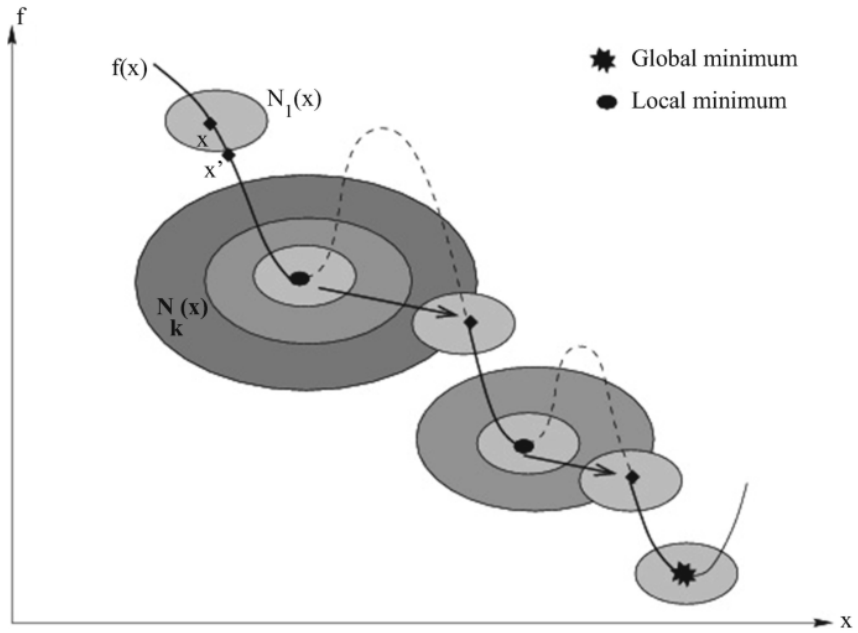




Local-Search based Metaheuristics

- Variable Neighborhood Search [Mladenović & Hansen, 1997]
- Simulated Annealing [Kirkpatrick, Gelatt Jr & Vecchi, 1983]

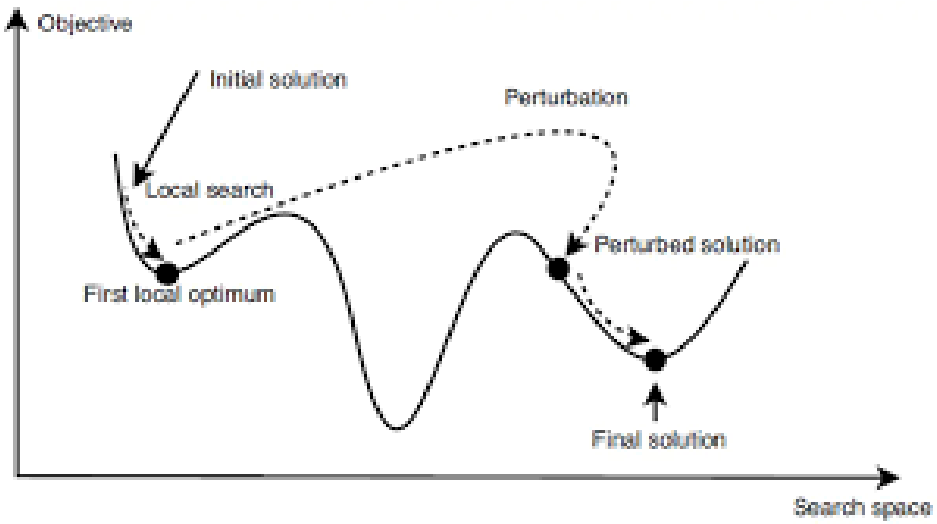
Variable Neighborhood Search Algorithm



Background

- The main idea of VNS is to successively explore a set of predefined neighborhoods to provide better solution.

Variable Neighborhood Search Algorithm



Background

- VNS systematically changes the neighborhood in two phases: descent to find a local optimum (local search phase) and a shaking (perturbation) phase to get out of the corresponding valley.

Variable Neighborhood Search Algorithm

VNS is built upon the following perceptions:

- A local minimum with respect to one neighborhood structure is not necessarily a local minimum for another neighborhood structure.
- A global minimum is a local minimum with respect to all possible neighborhood structures.
- For many problems, local minima with respect to one or several neighborhoods are relatively close to each other.

Variable Neighborhood Search Algorithm

Initialization:

Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$; find an initial solution x; choose a stopping condition.

while *stopping condition is not met* **do**

- Set $k \leftarrow 1$;
- **while** $k \neq k_{max}$ **do**
  - **Shaking:** Shaking: Generate a solution $x^{'}$ from the k-th neighborhood of solution x ($x^{'} : N_k(x)$).
  - **Local Search:** Apply local search to solution $x^{'}$ and denote the resulting solution by $x^{''}$.
  - **if** $x^{''}$ is better than the incumbent, set $x = x^{''}$ and continue with the initial neighborhood ($k \leftarrow 1$).
  - **Otherwise**, set $k \leftarrow k + 1$
- **end**

end

Simulated Annealing

- Random neighborhood exploration.
- Starts from high temperatures (very likely to accept worsening moves) and ends at low temperatures (very likely to accept mostly improving or side moves)
- Main parameters: T_s: start temperature, T_f: final temperature, α: cooling scheme, N_s: move evaluations per temperature
- Stop criterion: total number of evaluations
- Cut-off: decrease temperature after N_a accepted moves, $N_a < N_s$

Simulated Annealing with Cut-Off

procedure *SimulatedAnnealing*(SearchSpace $\mathcal{S}$, Neighborhood $\mathcal{N}$,
CostFunction F, Parameters T_0, T_f, α, N_s, N_a)

1: $T \leftarrow T_0$
2: $s \leftarrow RandomState(\mathcal{S})$
3: $s_{best} \leftarrow s$
4: **while** $T \geq T_f$
5: $n_s \leftarrow 0$, $n_a \leftarrow 0$
7: **while** $n_s < N_s \wedge n_a < N_a$
8: $m \leftarrow RandomMove(s, \mathcal{N})$
9: $\Delta F \leftarrow F(s \oplus m) - F(s)$
10: **if** $(\Delta F \leq 0)$
11: $s \leftarrow s \oplus m$, $n_a \leftarrow n_a + 1$
13: **if** $(F(s) < F(s_{best}))$
14: $s_{best} \leftarrow s$
15: **else**
16: **if** $(RandomReal(0,1) < e^{-\Delta F/T})$
17: $s \leftarrow s \oplus m$, $n_a \leftarrow n_a + 1$
19: $n_s \leftarrow n_s + 1$
20: $T \leftarrow T \cdot \alpha$
21: **return** s_{best}

Local-search-based Metaheuristics

Advantages:

- Simple structure
- Easy to apply for many hard combinatorial optimization problems
- Require few, and sometimes no parameters.
- Provide very good solutions for many problems.

Disadvantages:

- Require **problem-specific** neighborhood structures, shaking and local search operators.

Outline

Negative Learning Ant Colony Optimization

- Traditional ACO is based on **positive learning**
  - Higher probability for "promising" solution components.
- Traditional ACO + **negative learning**
  - lower probability to "bad" explored solution components.
  - NL-ACO learns also from negative samples, not only from positive ones.

Ant Colony Optimization

procedure *Negative Learning-ACO*(Problem Instance $\mathcal{C}$, Cost Function F, Parameters n_{ants}, α^+, d_{rate})

1: $\rho^+ \leftarrow$ *InizializePheromoneMatrix*()
2: **while** $t < T_{max}$
3: $i \leftarrow 0$
4: **while** $i < n_{ants}$
5: $sol_i \leftarrow$ *GenerateSolution*$(\mathcal{C}, \rho^+, d_{rate})$
6: $i \leftarrow i + 1$
7: $\rho^+ \leftarrow$ *UpdatePositivePheromone*$(S_{best}, S_{best_{iter}}, \alpha^+)$
8: **return** S_{best}

Negative Learning Ant Colony Optimization

procedure *Negative Learning-ACO*(Problem Instance $\mathcal{C}$, Cost Function F, Parameters n_{ants}, α^+, α^-, d_{rate})

1: $\rho^+, \rho^-, \leftarrow$ *InizializePheromoneMatrix*()
2: **while** $t < T_{max}$
3: $i \leftarrow 0$, $\mathcal{C}' \leftarrow \emptyset$
4: **while** $i < n_{ants}$
5: $sol_i \leftarrow$ *GenerateSolution*$(\mathcal{C}, \rho^+, \rho^-, d_{rate})$
6: $\mathcal{C}' \leftarrow \mathcal{C}' \cup sol_i$
7: $i \leftarrow i + 1$
8: $\rho^+ \leftarrow$ *UpdatePositivePheromone*$(S_{best}, S_{best_{iter}}, \alpha^+)$
9: $S_{mip} \leftarrow$ *ApplyMIPSolver*$(\mathcal{C}', S_{best})$
10: $\rho^- \leftarrow$ *UpdateNegativePheromone*$(S_{mip}, \mathcal{C}', \alpha^-)$
11: **return** S_{best}

NL—ACO with Local Search

procedure *Negative Learning-ACO*(Problem Instance $\mathcal{C}$, Cost Function F, Parameters n_{ants}, α^+, α^-, d_{rate})

1: $\rho^+, \rho^-, \leftarrow$ *InizializePheromoneMatrix*()
2: **while** $t < T_{max}$
3: $i \leftarrow 0$, $\mathcal{C}' \leftarrow \emptyset$
4: **while** $i < n_{ants}$
5: $sol_i \leftarrow$ *GenerateSolution*$(\mathcal{C}, \rho^+, \rho^-, d_{rate})$
6: $sol_i \leftarrow$ *SteepestDescent*$(sol_i, \{Insert, Remove, Swap\})$
7: $\mathcal{C}' \leftarrow \mathcal{C}' \cup sol_i$
8: $i \leftarrow i + 1$
9: $\rho^+ \leftarrow$ *UpdatePositivePheromone*$(S_{best}, S_{best_{iter}}, \alpha^+)$
10: $S_{mip} \leftarrow$ *ApplyMIPSolver*$(\mathcal{C}', S_{best})$
11: $\rho^- \leftarrow$ *UpdateNegativePheromone*$(S_{mip}, \mathcal{C}', \alpha^-)$
12: **return** S_{best}

Negative Learning Ant Colony Optimization

- At each iteration of the ACO:
  1. Create a **sub-instance** $\mathcal{C}'$ of the tackled problem instance $\mathcal{C}$, by merging the n_{ants} constructed solutions.
  2. Use a **MIP solver** to find the best possible solution to this sub-instance (within a pre-defined time limit)
  3. **Update** negative pheromones ρ^-
- **Positive** pheromone model ρ^+ and **negative** pheromone model ρ^-
  - ρ^+: $c \in S_{best} \cup S_{best_{iter}}$ are reinforced positively at every iteration
  - ρ^-: $c \in \mathcal{C}' \setminus S_{mip}$ are reinforced negatively at every iteration

Outline for section 3

Instance(s)

- 1 instance provided for the competition.
- 320 asteroids, 12 stations, 3 materials, 80 days.
- 13920 delivery opportunities.

Station	m1	m2	m3
S1	408	320	261
S2	410	322	262
S3	407	323	265
S4	407	324	267
S5	408	326	271
S6	407	326	271
S7	411	329	272
S8	409	328	270
S9	405	322	264
S10	405	321	264
S11	409	324	268
S12	411	324	266

Parameters Tuning (NL-ACO)

- `irace` package (López-Ibáñez et al., 2016)
  - iterated racing procedure, extension of Iterated F-race (I/F-Race))
- Tuning done on the only available instance (overtuning)

parameter	name	domain	value
ants number	n_{ants}	$\{3, 5, 10, 20\}$	20
positive learning rate	α^+	$[0.1, 0.5]$	0.5
negative learning rate	α^-	$[0.1, 0.5]$	0.2
deterministic rate	d_{rate}	$[0.0, 0.9]$	0.5

Experimental Setting

- C++, Ubuntu 20.04.4 LTS
- Intel Xeon Processor (Cascadelake), 16 cores, CPU frequency 2.4 GHz, and 117 GB of RAM
- 1 CPU per run.
- ACO time limit: $57600s$

Best Results

Results obtained by our ILP model, solved by CPLEX 20.1:

After 1h: **Null** After 10h: **Null**

Best result obtained by our VNS/SA implementation:

9.235

Best result obtained by our NL-ACO:

9.520

Our Best Solution

- Score: **9.52011**
- Delivered asteroids: 310/320
- Stations order: S3, S7, S8, S12, S5, S9, S1, S4, S10, S6, S11, S2

Station	T_{start}	T_{end}	m1	m2	m3
S1	55.735	59.582	11.0544	10.1541	9.54644
S2	75.537	77.608	10.6248	10.0219	**9.52011**
S3	7.773	25.244	9.53458	9.61497	9.69257
S4	60.852	64.045	10.0951	9.87847	9.55666
S5	47.166	51.371	10.3537	10.029	9.53246
S6	68.299	70.629	10.5164	10.0139	9.55802
S7	26.317	33.221	9.67706	9.66398	9.52532
S8	34.232	39.953	9.63146	9.60481	9.55556
S9	52.505	54.724	10.7889	10.0172	9.5293
S10	65.241	67.239	10.689	10.1874	9.52464
S11	71.702	74.501	11.7597	10.5221	9.57924
S12	41.396	46.019	9.68404	9.53862	9.57326

Outline for section 4

Conclusion and Future Work

- NL-ACO for the Delivery Scheduling Problem:
  - Randomized Greedy + Local Search for solution generation.
  - Positive and negative pheromones.
- 2nd position achieved in Delivery Scheduling problem
- Future Work:
  - Larger or more diverse instances
  - Problem-specific local search neighborhoods for VNS and SA
  - Large Neighborhood Search or Construct, Merge, Solve & Adapt approach

Questions?