Heuristic Search and Metaheuristics

Algorithmics study and revision notes

Jaak Vilo

2026-08-22

Contents

1 Why use heuristics?

Many optimisation problems have enormous state spaces, expensive objectives, or NP-hard worst cases. Exact optimisation may be unnecessary or impossible within the available time. A heuristic aims to return a useful solution under a resource budget, but it does not remove the need for precise modelling or evidence.

The right question is not simply "Did the method find a good answer?" Ask what "good" means, which guarantee applies, what budget was used, which baseline was beaten, and whether the result repeats on unseen instances.

2 Learning goals

After studying this note, you should be able to:

- distinguish exact, approximation, heuristic, and metaheuristic methods;
- define a candidate representation, constraints, objective, neighbourhood, and budget;
- derive greedy set cover and its approximation guarantee;
- explain how A* can use a heuristic and remain exact;
- implement nearest-neighbour and 2-opt search for TSP;
- trace simulated annealing, tabu search, and stochastic SAT moves;
- implement the main loop of an evolutionary algorithm and differential evolution; and
- design a fair, reproducible empirical comparison.

3 Prerequisites

- Graphs, shortest paths, priority queues, and hash-based visited-state tables.
- Elementary probability and random sampling.
- Asymptotic analysis and the difference between a proof and an experiment.
- Basic vector arithmetic for continuous optimisation.

4 Four kinds of method

The sections move from the strongest claims to the most empirical ones: a proved approximation, exact A*, single-solution local search, population methods, and finally experimental evaluation. For every method, keep the representation, move generator, stopping budget, and guarantee visible; the biological metaphor is never the specification.

For a first pass, focus on the four method classes, problem specification, set cover, A*, local search, annealing, and evaluation. Then use that common vocabulary to compare tabu search, evolutionary search, DE, ACO, and PSO without treating their metaphors as explanations.

These labels should not be used interchangeably:

- An **exact algorithm** returns an optimum and has a correctness proof. A* can be exact.
- An **approximation algorithm** runs in polynomial time and proves that its answer is within a factor or additive error of optimum. Greedy set cover is an approximation algorithm.
- A **heuristic algorithm** uses problem-informed choices without necessarily proving solution quality.
- A **metaheuristic** is a reusable search framework, such as simulated annealing or evolutionary search, whose concrete behaviour depends on representation, moves, parameters, and budget.

A method may also be **complete**, meaning it eventually finds a solution if one exists, without being an efficient optimiser. Stochastic local SAT methods are usually incomplete; systematic SAT solvers can be complete.

5 Define the search problem first

Specify at least:

1. **candidate representation:** what one state stores;
2. **feasibility:** which candidates satisfy the constraints;
3. **objective or fitness:** what is minimised or maximised;
4. **variation or neighbourhood:** which new candidates can be generated;
5. **initialisation:** where search begins;
6. **selection rule:** which candidate becomes current or survives; and
7. **termination and budget:** time, iterations, or objective evaluations.

Representation determines reachability and cost. A TSP tour stored as a permutation supports swaps, insertions, and reversals while remaining a tour. Independent bit flips do not preserve a permutation.

Constraints can be handled by a feasibility-preserving representation, repair, rejection, a penalty term, or a solver embedded inside the search. Each choice changes the landscape. State whether a larger objective is better; the algorithms below use minimisation unless noted.

6 Account for work

For iterative methods, a useful first estimate is

$$\text{total work} \approx \text{iterations} \times \text{candidates examined per iteration} \times \text{cost per evaluation}.$$

Incremental evaluation can matter more than the name of the metaheuristic. In TSP, recomputing a whole tour costs $\Theta(n)$, while the cost change of a 2-opt move depends on only four affected edges and can be computed in $O(1)$.

7 Greedy construction and set cover

A greedy method makes the best-looking available irreversible choice. It is exact only when a proof such as an exchange or cut argument establishes that local choices compose into a global optimum. Otherwise it may still have an approximation guarantee.

In **weighted set cover**, a universe U must be covered by sets in a family $\mathcal{S}$. Set S has cost $c(S) > 0$. The goal is a minimum-cost subfamily whose union is U.

```
greedy_set_cover(U, family):
    uncovered = U
    answer = empty list
    while uncovered is not empty:
        if no T in family covers a new element: report infeasible
        choose T in family minimising c(T) / |T intersect uncovered|
            among sets that cover a new element
        append T to answer
        remove T's elements from uncovered
    return answer
```

For unweighted set cover, every cost is one, so choose the set covering most currently uncovered elements.

7.1 Why the logarithmic guarantee holds

Charge the cost of a selected set equally to the elements it covers for the first time. The algorithm's total charge equals its total cost.

Fix an optimal solution, and assign every universe element to one set of that solution which covers it. Consider one optimal set S^* of cost c with $q \geq 1$ elements assigned to it; sets with no assigned elements contribute nothing. When r assigned elements remain uncovered, S^* covers at least those r elements, so its current ratio is at most c/r.

Greedy's chosen ratio, and hence the charge to each element newly covered in that step, is no larger. Ordering the q assigned elements by when they are first covered bounds their total charge by

$$c\left(\frac{1}{q} + \frac{1}{q-1} + \cdots + 1\right) = cH_q \leq cH_d,$$

where $d = \max_{S \in \mathcal{S}} |S|$. Summing over the sets in the fixed optimum counts every element's charge once. Greedy therefore costs at most $H_d \leq 1 + \ln d$ times optimum. This is a proved worst-case guarantee, not an empirical claim that greedy is usually optimal.

8 A* as an exact use of heuristics

A* searches a finite state graph with nonnegative transition costs for a minimum-cost path. It orders the frontier by

$$f(n) = g(n) + h(n),$$

where $g(n)$ is the best known cost from the start and $h(n)$ estimates remaining cost.

```
put start in a priority queue with priority h(start)
while the queue is not empty:
    n = remove smallest f
    if n is a goal: return its reconstructed path
    for each successor y:
        if g[n] + cost(n,y) improves g[y]:
            update g[y], parent[y], and its queue priority
```

An admissible heuristic satisfies $0 \leq h(n) \leq h^*(n)$, where $h^*(n)$ is the true remaining cost; therefore $h = 0$ at every goal. A consistent heuristic also satisfies

$$h(n) \leq cost(n, y) + h(y)$$

for every transition. Test for a goal when it is removed from the priority queue, not merely when it is generated. With consistency, a standard closed-set graph search need not reopen expanded states. With admissibility but not consistency, reopening improved states preserves optimality. The graph note gives the correctness argument in full.

Heuristic quality affects work, not the optimum: $h = 0$ gives Dijkstra; a more informative admissible heuristic can expand fewer states. If $h_1(n) \geq h_2(n)$ everywhere and both are admissible, h_1 **dominates** h_2.

9 TSP: construction and local improvement

The travelling salesperson problem (TSP) asks for a minimum-weight Hamiltonian cycle: visit every city exactly once and return to the start. A symmetric n-city instance has $(n-1)!/2$ distinct tours after ignoring start rotation and reversal, so exhaustive enumeration grows too quickly.

Nearest neighbour starts at a city and repeatedly visits the closest unvisited city, finally returning to the start. A straightforward implementation is $O(n^2)$. It is a useful baseline and can be poor because an attractive early edge can force an expensive final connection. Try every start before crediting the method itself with one lucky start.

A common local move is **2-opt**. Choose two nonadjacent tour edges (a, b) and (c, d), remove them, reverse the intervening segment, and add (a, c) and (b, d). For a symmetric TSP, its cost change is

$$\Delta = w(a, c) + w(b, d) - w(a, b) - w(c, d).$$

A negative Δ improves a minimisation tour. Repeating improving 2-opt moves reaches a 2-opt local optimum, not necessarily the global optimum.

10 Local search

Local search keeps one or a few complete candidates and moves through a neighbourhood:

```
local_search(x):
    repeat:
        choose an improving neighbour y of x
        if no improving neighbour exists: return x
        x = y
```

Best improvement scans the whole neighbourhood and takes the largest gain. **First improvement** takes the first gain found and may perform many cheaper iterations. The result depends on neighbourhood, scan order, tie policy, and initial state.

Plateaus contain equal-valued neighbours; ridges require coordinated moves; local optima have no improving neighbour. Random restarts sample different attraction basins. Allowing sideways or worsening moves can escape a basin but introduces new control parameters.

11 Simulated annealing

For minimisation, propose a random neighbour y and set

$$\Delta = cost(y) - cost(x).$$

Accept every move with $\Delta \leq 0$. Accept a worsening move with probability

$$\exp(-\Delta/T),$$

where temperature $T > 0$ decreases over time.

```
x = initial candidate
best = x
for t = 1, 2, ... until budget ends:
    T = schedule(t)
    y = random neighbour of x
    delta = cost(y) - cost(x)
    if delta <= 0 or random(0,1) < exp(-delta/T):
        x = y
        if cost(x) < cost(best): best = x
return best
```

High temperature explores; low temperature becomes nearly greedy. A scale that is hot for one objective may be frozen for another, so inspect typical move deltas when choosing the initial temperature. Extremely slow theoretical cooling can guarantee asymptotic convergence under restrictive conditions but is rarely practical. Finite-budget performance depends on proposal moves, schedule, reheating/restarts, and stopping.

12 Tabu search

Tabu search commonly chooses the best available neighbour even when it is worse, while short-term memory prevents immediate cycles. Rather than storing every complete solution, it records move attributes such as "edge (a, b) was removed" for a fixed **tabu tenure**.

An **aspiration rule** overrides tabu status when a move produces a new global best. Tenure that is too short permits cycling; tenure that is too long excludes useful regions. Long-term frequency memory can encourage rarely used components or diversify after stagnation.

13 Stochastic SAT search

For a Boolean formula in conjunctive normal form, a complete assignment is a state. A one-flip neighbourhood changes one variable, and an objective may count unsatisfied clauses.

A WalkSAT-style step selects an unsatisfied clause, then either flips a random variable in it or flips a variable with a favourable break/make score. Noise helps cross local minima. Restarts reduce dependence on one assignment.

This is incomplete stochastic local search: failure within a budget does not prove unsatisfiability. Complete DPLL/CDCL solvers add systematic branching, propagation, conflict analysis, learned clauses, and backtracking.

14 Evolutionary algorithms

An evolutionary algorithm maintains a population:

1. initialise feasible or repairable candidates;
2. evaluate fitness;
3. select parents with controlled selection pressure;
4. create offspring by recombination and mutation;
5. repair or penalise infeasible offspring;
6. select survivors, often preserving an elite; and
7. stop at the evaluation budget.

Representation-specific operators are essential. One-point crossover on bit strings preserves length, but naïvely crossing two TSP permutations can duplicate cities. Order-based crossover and swap/inversion mutation preserve permutation structure.

Selection pressure that is too weak makes progress slow; too strong destroys diversity and can converge prematurely. Population size, mutation scale, survivor policy, and constraint handling are part of the algorithm and must be reported.

15 Differential evolution

Differential evolution (DE) is designed for vectors $x_i \in \mathbb{R}^d$. In the common **DE/rand/1/bin** variant, for each target vector x_i:

1. choose distinct population indices r_1, r_2, r_3, all different from i; thus this variant needs at least four population members;
2. create a donor

$$v = x_{r_1} + F(x_{r_2} - x_{r_3});$$

3. choose one forced coordinate j_{rand} and create the trial vector coordinatewise:

$$u_j = \begin{cases} v_j, & \text{if } R_j \le CR \text{ or } j = j_{\text{rand}}, \\ x_{i,j}, & \text{otherwise,} \end{cases} \qquad R_j \sim \text{Uniform}(0,1);$$

4. repair, reflect, clip, or resample coordinates outside their bounds; and
5. record u for the next generation if it is no worse than x_i; otherwise retain x_i.

$F > 0$ controls differential step scale, $CR \in [0, 1]$ controls coordinate mixing, and population size controls diversity and evaluations per generation. Donors are normally drawn from the unchanged current generation and replacements installed together, avoiding iteration-order effects. One generation evaluates roughly one trial per population member. Compare methods by total objective evaluations when objective cost dominates.

15.1 Robust regression example

For polynomial fitting, a candidate vector contains coefficients. RMSE strongly penalises large residuals and is sensitive to outliers. A median absolute or squared residual is robust to a minority of extreme points but may ignore broad moderate error. Report the precise loss and data scale.

If linear, quadratic, and cubic models are all allowed, fitting error alone tends to prefer extra flexibility. Penalise complexity, validate on held-out data, or search degree separately. Optimising coefficients and claiming the true degree has been recovered are different tasks.

16 ACO and PSO

These are useful survey methods; their equations matter more than their biological metaphors.

In **ant colony optimisation (ACO)**, a construction step chooses component j with probability proportional to

$$\tau_j^\alpha \eta_j^\beta,$$

where τ is learned pheromone and η is problem-specific desirability. More precisely, if $F(x)$ is the set of feasible next components from partial solution x, initialise pheromones positively and sample

$$\Pr(j \mid x) = \frac{\tau_j^\alpha \eta_j^\beta}{\sum_{\ell \in F(x)} \tau_\ell^\alpha \eta_\ell^\beta} \quad \text{for } j \in F(x),$$

with probability zero outside $F(x)$, where $\alpha, \beta \geq 0$ and the denominator must be positive. If $F(x)$ is empty before a solution is complete, the construction needs a stated repair, backtracking, or rejection policy. After constructing and evaluating complete solutions, evaporate and deposit nonnegative, quality-dependent amounts:

$$\tau_j \leftarrow (1 - \rho)\tau_j + \Delta\tau_j, \qquad 0 < \rho \leq 1.$$

Evaporation discounts old evidence; reinforcement favours components of selected good solutions. Without positive lower bounds or another exploration rule, pheromones can become so unequal that the search effectively locks in early choices.

In **particle swarm optimisation (PSO)**, particle i has position x_i, velocity v_i, personal best p_i, and a neighbourhood or global best g:

$$v_i \leftarrow \omega v_i + c_1 r_1 \odot (p_i - x_i) + c_2 r_2 \odot (g - x_i), \qquad x_i \leftarrow x_i + v_i.$$

Initially set $p_i = x_i$ and choose each particle's g from the initial personal bests according to the communication topology. Here r_1 and r_2 are normally independent vectors whose coordinates are sampled uniformly from $[0, 1]$. After moving and applying the boundary rule, evaluate the new position, replace p_i if x_i is better, and update the relevant neighbourhood best g.

One synchronous iteration uses about one objective evaluation per particle. Inertia ω, attraction coefficients, velocity control, boundary rules, and the communication topology all affect convergence and diversity; omitting any of these choices leaves the algorithm underspecified.

17 Worked trace: 2-opt and annealing

Suppose tour $A-B-C-D-E-A$ has edge costs $5, 7, 4, 6, 8$, total 30. Replace edges (A, B) and (D, E) by (A, D) of cost 3 and (B, E) of cost 4. Reversing the middle segment gives

$$A - D - C - B - E - A$$

with cost $3 + 4 + 7 + 4 + 8 = 26$. The 2-opt delta is $3 + 4 - 5 - 6 = -4$, so every improving local-search rule accepts it.

Later, suppose a proposed move is worse by $\Delta = 3$. At temperature $T = 2$, annealing accepts it with probability

$$e^{-3/2} \approx 0.223.$$

At $T = 0.5$, the probability is $e^{-6} \approx 0.0025$. The same move changes from plausible exploration to almost certain rejection as the search cools.

18 Experimental evaluation

Stochastic optimisation needs repeated independent runs. Report:

- instance set, preprocessing, and train/tuning/test split;
- representation, constraint handling, and all stopping conditions;
- objective-evaluation count and wall-clock time;
- parameter-selection procedure;
- random seeds and number of independent runs;
- feasibility and failure rate;
- median, spread, and preferably the whole quality distribution;
- an anytime curve or time-to-target when appropriate; and
- simple baselines under the same budget.

Use paired instances and, where meaningful, paired random scenarios when comparing methods. Do not tune on the same instances used for final claims. The best run answers "Was this ever achieved?"; it does not estimate typical performance.

19 Complexity and trade-offs

Method	Guarantee	Typical work unit	Main risk
Greedy set cover	H_d approximation	scan coverage gains	irreversible early choices
A*	optimal under stated heuristic/search conditions	priority-queue expansion	exponential time and memory
Nearest-neighbour TSP	heuristic baseline	choose next city	expensive closing edge
2-opt local search	local optimum	edge-pair delta	trapped basin
Simulated annealing	no practical finite-budget guarantee	one sampled neighbour	schedule mismatch
Tabu search	heuristic	candidate neighbourhood plus memory	tenure/attribute design
Stochastic SAT	incomplete heuristic	one variable flip plus clause updates	cannot certify unsatisfiability
Evolutionary search	heuristic	population generation	lost diversity
DE	heuristic	one trial per target vector	scale and boundary handling

Method	Guarantee	Typical work unit	Main risk
ACO	heuristic	constructed population plus pheromone update	stagnation and construction policy
PSO	heuristic	one move and evaluation per particle	premature convergence and boundary handling

20 Connections

- Set cover shows that a greedy heuristic can also have a proof.
- A* reuses graph search and priority queues while introducing heuristic estimates.
- Hashing stores visited or tabu states and Zobrist hashes mutable game positions.
- TSP, set cover, and SAT turn combinatorial objects into search spaces.
- Differential evolution leads directly into robust-regression experiments paired with dynamic programming in the next homework.
- Experimental design connects algorithm analysis with measurement rather than replacing it.

21 Common mistakes

- Omitting the representation, feasibility rule, or objective direction.
- Calling a local optimum globally optimal.
- Treating an approximation guarantee, A* heuristic, and metaheuristic as the same concept.
- Recomputing a complete objective when a constant-time move delta exists.
- Using a crossover that creates invalid permutations.
- Reporting only one fortunate stochastic run.
- Comparing methods with unequal objective-evaluation budgets.
- Tuning on the final test instances.
- Describing DE only by its donor equation and omitting crossover, selection, and bounds.

22 Self-check

1. Classify greedy set cover, A*, nearest-neighbour TSP, and simulated annealing by guarantee.
2. Reproduce the charging argument for the H_d set-cover bound.
3. What must be true before A* may permanently close a state?
4. Why does 2-opt preserve a Hamiltonian tour?
5. Calculate annealing acceptance for $\Delta = 5$ at two temperatures.
6. How do tabu tenure and aspiration serve different purposes?
7. Why does stochastic SAT failure not prove unsatisfiability?
8. Write every step of DE/rand/1/bin after donor creation.
9. Which evidence distinguishes one lucky run from a reliable method?

23 Revision summary

- Define representation, constraints, objective, moves, and budget before choosing a method.
- Exact, approximation, heuristic, and metaheuristic methods make different promises.
- Greedy set cover has a logarithmic proof; A* can remain exact.
- TSP local search illustrates neighbourhoods, deltas, and local optima.
- Annealing, tabu memory, and restarts escape basins in different ways.
- Evolutionary methods trade selection pressure against diversity.
- DE requires donor, crossover, boundary, and survivor rules.
- Reproducible evaluation is part of the algorithmic claim.

24 Implementations and hands-on exploration

- SciPy's Python `differential_evolution` implements several DE strategies, bounds, constraints, parallel evaluation, and optional polishing. It is stochastic and often uses many more objective evaluations than a gradient method, so fix the random generator and count evaluations.
- DEAP is a Python framework for explicit evolutionary representations, operators, statistics, checkpoints, and parallel evaluation. Its boxed algorithms are starting points, not universal definitions; report the registered selection, variation, and replacement operators.
- Google OR-Tools routing search exposes construction heuristics, guided local search, simulated annealing, and tabu search through C++ with Python, Java, and C# APIs. A time-limited feasible routing solution is not automatically a proof of optimality, so inspect the returned search status.
- jMetal is a Java framework for multi-objective metaheuristics, benchmark problems, and reproducible experiments. Pareto-front quality needs multi-objective indicators and repeated runs; one scalar best value is not an adequate comparison.
- Rust's `pathfinding` crate implements A*, BFS, Dijkstra, flow, and related searches over caller-supplied successor functions. Correct A* optimality still depends on nonnegative costs and an appropriate heuristic; the library cannot infer those semantic conditions.

24.1 Small experiments

1. On the same blocked grid, compare A* with $h = 0$, Manhattan distance, and an intentionally overscaled Manhattan heuristic. Record returned path cost, expanded states, and peak frontier size; explain which runs retain an optimality guarantee.
2. Run simulated annealing or DE for at least 30 fixed seeds under one objective-evaluation budget. Plot median and interquartile-range anytime curves, retain the best-so-far value per run, and compare with random search and one deterministic local-search baseline.

25 Sources and further study

- Heuristic Search I, II, and III, Jaak Vilo.
- (Kleinberg and Tardos 2006), Chapters 11-12.
- Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 4th edition, 2021.

References

Kleinberg, Jon, and Éva Tardos. 2006. *Algorithm Design*. Pearson.