Constructive meta-heuristics
Heuristic algorithms

Giovanni Righini

University of Milan
Department of Computer Science (Crema)
Improving constructive algorithms

For many problems constructive algorithms show strong limitations. One can iterate their execution \( \ell \) times, with different \( \mathcal{F} \) and \( \phi \) in order to obtain potentially different solutions \( x^{[1]}, \ldots, x^{[\ell]} \).

- **Less efficiency**: computing times add up.
- **More effectiveness**: the final solution is the best found.

There is a trade-off to carefully calibrate!

Without loss of generality, we assume

- to always consider the whole search space \( \bar{\mathcal{F}} = \bigcup_{l=1}^{\ell} \mathcal{F}^{[l]} \)
- to obtain the specific search space \( \mathcal{F}^{[l]} \subset \mathcal{F} \) at iteration \( l \) by imposing that \( \phi(i, x) = +\infty \) for each \( (i, x) : x \cup \{i\} \in \bar{\mathcal{F}} \setminus \mathcal{F} \).
Main constructive meta-heuristics

We will consider

1. **multi-start** or **restart**: it runs a set of algorithms $A^{[l]}$ or the same algorithm $A$ with different initialization.

2. **roll-out**: for each $i \in \text{Ext}(x)$ compute the solution $x_A(x \cup \{i\})$ of value $z_A(x \cup \{i\})$ obtained from the heuristic $A$ initialized with $x \cup \{i\}$ and select the element with the best estimate

$$i^{(t)} = \arg \min_{i \in \text{Ext}(x^{(t-1)})} z_A(x^{(t-1)} \cup \{i\})$$

*(Remark: $z_A$ is not $z$, but $z(x_A)$!)*

3. Adaptive Research Technique (**ART**), or **Tabu Greedy**: defines $\mathcal{F}^{[l]}$ forbidding some classes of subsets of $\mathcal{F}$; prohibitions are updated on the basis of the results obtained;

4. **semi-greedy** and **GRASP**: make the choice partially random $(\phi(i, x, \omega))$

5. **Ant System**: makes the choice partially random and partially depending on the results obtained $(\phi(i, x, \omega, x^{[1]}_A, \ldots, x^{[l]}_A))$

The **ART** uses memory, the **GRASP** uses randomness, the **AS** uses both.
Multi-start

Multi-start is a very intuitive method:

- define different selection criteria \( \phi[l] (i, x) \);
- run the corresponding algorithms \( A[l] \);
- keep the best solution found.

It is often used when \( \phi (i, x) \) depends on numerical parameters \( \mu \)

Example: the TSP on a complete graph

- insertion criterion: instead of \( c_{s_i,k} + c_{k,s_{i+1}} - c_{s_i,s_{i+1}} \) one can use
  \[
  \min_i \gamma_{i,k} = \alpha_1 (c_{s_i,k} + c_{k,s_{i+1}}) - (1 - \alpha_1) c_{s_i,s_{i+1}} \quad \text{with } \alpha_1 \in [0; 1]
  \]

- selection criterion: instead of \( \max_k d(x, k) \) or \( \min_{ik} \gamma_{ik} \), find
  \[
  \max_k \phi(k, x) = \alpha_2 d(x, k) - (1 - \alpha_2) \gamma_{i^*,k} \quad \text{with } \alpha_2 \in [0; 1]
  \]
Given a base constructive heuristic $A$

- start from the empty set: $x^{(0)} = \emptyset$
- at each iteration $t = 1, 2, \ldots$
  - extend the solution in every feasible way: $x^{(t-1)} \cup \{i\}, \forall i \in \text{Ext}(x)$
  - from each feasible extension run $A$: the value of each solution obtained is an estimate of the result: $z^A(x^{(t-1)} \cup \{i\})$
  - select the extension that produces the best estimate

$$i^{(t)} = \arg \min_{i \in \text{Ext}(x^{(t-1)})} z^A(x^{(t-1)} \cup \{i\})$$

- stop when $\text{Ext}(x) = \emptyset$.

It is also called single-step look-ahead constructive heuristic.

The result of the roll-out heuristic based on $A$ dominates the result of $A$ when some conditions are satisfied.
Example: roll-out for the *SCP*

\[
\begin{array}{c}
c & 25 & 6 & 8 & 24 & 12 \\
\hline
A \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

1. Start with \(x^{(0)} = \emptyset\).

2. For each column \(i\) run the constructive heuristic from the initial solution \(x^0 \cup \{i\} = \{i\}\):
   - for \(i = 1\), we obtain \(x_A(\{1\}) = \{1\}\) with cost \(z^A(\{1\}) = 24\)
   - for \(i = 2\), we obtain \(x_A(\{2\}) = \{2, 3, 5, 4\}\) with cost \(z^A(\{2\}) = 50\)
   - for \(i = 3\), we obtain \(x_A(\{3\}) = \{3, 2, 5, 4\}\) with cost \(z^A(\{3\}) = 50\)
   - for \(i = 4\), we obtain \(x_A(\{4\}) = \{4, 2, 5\}\) with cost \(z^A(\{4\}) = 42\)
   - for \(i = 5\), we obtain \(x_A(\{5\}) = \{5, 2, 3, 4\}\) with cost \(z^A(\{5\}) = 50\)

3. The best estimate is the first one. Hence: \(i^{(1)} := 1\).

4. The algorithm stops producing the optimal solution.
Properties of roll-out heuristics

It is possible to mix roll-out and multi-start meta-heuristics: given several constructive heuristics $A[1], \ldots, A[\ell]$

- start from the empty set;
- for each iteration $t$
  - for each feasible extension $i \in \text{Ext}(x^{(t-1)})$
    run each algorithm $A[l]$ from $x^{(t-1)} \cup \{i\}$
  - obtain an estimate $z^{A[l]}(x^{(t-1)} \cup \{i\})$
  - take the decision corresponding to the best estimate

$$i^{(t)} = \arg \min_{l=1,\ldots,\ell} \min_{i \in \text{Ext}(x^{(t-1)})} z^{A[l]}(x^{(t-1)} \cup \{i\})$$

- stop when $\text{Ext}(x)$ is empty.

The result of the hybrid roll-out algorithm dominates the results of each constructive heuristic $A[l]$ if some conditions are satisfied.

The computational complexity is much higher with respect to the algorithms $A[l]$ but it remains polynomial (in the worst case it is multiplied by $|E|^2$).
Adaptive Research Technique

Proposed by Patterson et al. (1998) for the Capacitated Minimum Spanning Tree.

Given a constructive heuristic $A$, one often observes that seemingly good elements included in early steps of $A$ lead to bad final solutions.

- Roll-out heuristics tentatively include elements in the solution;
- $ART$ tentatively forbids elements in order to avoid that they lead the solution $x$ on a wrong path in $\mathcal{F}$.

By forbidding some elements of the already visited solutions one forbids to obtain solutions similar to them (diversification).
Adaptive Research Technique

\textit{ART} requires a constructive heuristic $A$.

Create an initially empty list of forbidden elements ($l_i = -\infty, \forall i \in E$)

At each iteration $l \in \{1, \ldots, \ell\}$

1. run $A$ with forbidden elements and obtain a solution $x^{[l]}$;
2. decide with probability $\pi$ whether to forbid or not each element $i \in x^{[l]}$;
3. for each forbidden element, keep the iteration $l_i$ when it was first forbidden;
4. remove the prohibitions lasting since more than $d$ iterations (expiration time) \((in \ step \ 1 \ elements \ i \ for \ which \ l - l_i \leq d \ are \ forbidden)\);
5. keep the best solution found and the corresponding $l_i$;
6. possibly modify the list of prohibitions.

At the end of the loop, it returns the best solution found.
Parameters

ART has at least three parameters:

- the duration of the prohibition, $d$,
- the probability of the prohibition, $\pi$,
- the number of iterations, $\ell$.

In general, the experimental tuning of parameters has drawbacks:

1. it requires very long experimental tests, because the number of possible settings grows combinatorially with
   - the number of parameters
   - the number of tested values for each parameter
   (the less robust the algorithm is, the more values one needs to test.)

2. one risks to overfit, i.e. to tune the parameters so that the results are very good but only on the specific benchmark used for the calibration.

An excessive number of parameters is an undesirable property of heuristics; it often reveals an insufficient study of the problem and the algorithm.
Self-tuning of the parameters

An alternative to the experimental calibration consists in a self-adaptation procedure.

One can use different values for $\pi$ and $d$, each one for $\ell$ iterations

1. in an inner loop $\pi$ starts from small values and increases:
   $$\pi^{(r)} := \alpha_{\pi} \pi^{(r-1)}$$
   \text{(one forbids more and more elements to diversify more and more)}

2. in an outer loop $d$ starts from large values and decreases:
   $$d^{(r)} := d^{(r-1)}/\alpha_d$$
   \text{(the duration of each prohibition decreases to allow for other prohibitions)}

This procedure has the disadvantage of

- requiring more computing time
- introducing additional parameters
  - the number of iterations in the two loops
  - the initial value $\pi^{(0)}$ and the increase rate $\alpha_{\pi}$
  - the initial value $d^{(0)}$ and the decrease rate $\alpha_d$

but the result is likely to be less sensitive to $\alpha_{\pi}$ and $\alpha_d$ than to $\pi$ and $d$. 
Intensification

An excessive diversification may hamper the discovery of the optimum.

**Intensification** is the mechanism opposite to **diversification**, i.e. the concentration of the search on the most promising subsets.

**Intensification mechanisms** can be guided by data:
- increasing the probability $\pi$ for the elements with large cost
- increasing the duration $d$ for the elements with large cost in order to favor the choice of the elements with small cost (assuming they are more likely to be part of an optimal solution).

Or they can be guided by the solutions found:
- each block of $\ell$ iterations is initialized with the list of prohibitions associated with the best known solution, instead of an empty list;
- the elements in the intersection of the best known solutions cannot be prohibited;

in order to favor the choice of the elements that appear in the best known solutions.

**Diversification** and **intensification** play complementary roles in the search.
Example: ART for the Set Covering Problem

\[
\begin{array}{ccccc}
 c & 25 & 6 & 8 & 24 & 12 \\
\hline
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
\end{array}
\]

1. the constructive heuristic finds the solution \( x^{[1]} = \{2, 3, 5, 4\} \) with cost \( z(x^{[1]}) = 50 \); forbid (at random) column 2;

2. the constructive heuristic finds the solution \( x^{[2]} = \{3, 1\} \) with cost \( z(x^{[2]}) = 33 \); forbid (at random) column 3;

3. the constructive heuristic finds the solution \( x^{[3]} = \{1\} \) with cost \( z(x^{[3]}) = 25 \), which is optimal.

4. . . .
Semi-greedy heuristics

At each iteration constructive heuristics select the seemingly best element according to a function $\phi(i, x)$

$$i(t) = \operatorname{arg\ min}_{i \in \text{Ext}(x)} \phi(i, x)$$

One or more of these choices is wrong, when the heuristic fails to produce an optimal solution.

A semi-greedy algorithm (Hart and Shogan, 1987) is based on the assumption that at least one right element (keeping the path to the optimum open) is always among the best elements according to $\phi(i, x)$, although it is not always the best one.

If one cannot improve $\phi(i, x)$, the selection is done at random

$$i(t) = \operatorname{arg\ min}_{i \in \text{Ext}(x)} \phi(i, x, \omega) \text{ with } \omega \in \Omega$$

It is necessary to define a probability distribution on the elements $i \in \text{Ext}(x)$ favoring elements with better values of $\phi(i, x)$.
Semi-greedy heuristics

The advantages:

- the probability of finding an optimal solution is always non-zero, provided that a path from \( \emptyset \) to \( X^* \) exists in \( \mathcal{F} \)
- one can repeatedly run the heuristic, obtaining different solutions: the probability of reaching the optimum increases with the number of trials.

These heuristics are probabilistically approximately complete.
Random walk

*Random walk* is a constructive heuristic where *the probabilities associated to the possible choices at each step are the same (uniform distribution).*

- If a path to the optimum exists, RW will eventually find it
- but the computing time can be very long.

It is the opposite extreme of a *deterministic constructive heuristic*, where the probability is concentrated on one element only.

In general, *non-deterministic constructive heuristics* tend to favor the most promising elements:

- they *accelerate the average convergence time*
- they *reduce the convergence guaranteed time in the worst case*

    There is a trade-off between the average and the worst case.
Semi-greedy and GRASP

GRASP stands for Greedy Randomized Adaptive Search Procedure (Feo e Resende, 1989). It is a sophisticated version of the semi-greedy heuristic.

- **Greedy**: it uses a constructive heuristic;
- **Randomized**: the constructive heuristic is non-deterministic;
- **Adaptive**: the heuristic uses an adaptive selection criterion $\phi(i, x)$, which also depends on $x$;
- **Search**: it alternates the constructive heuristic with exchange heuristics.
The probability distribution function

Several different functions $\pi(i, x)$ respect the monotonicity condition

- **uniform probability**: each feasible element $i$ has the same probability $\pi(i, x)$; the algorithm executes a random walk in $\mathcal{F}$;
- **Heuristic-Biased Stochastic Sampling (HBSS)**:
  - the feasible elements are sorted by non-increasing values of $\phi(i, x)$
  - decreasing probabilities according to the rank following a simple function (linear, exponential, etc.)
- **Restricted Candidate List (RCL)**:
  - sort the feasible elements by non-increasing values of $\phi(i, x)$
  - insert the first $r$ elements in a list (\textit{r must be tuned})
  - assign uniform probabilities to the elements in the list and zero to the others.

The most common strategy is the \textit{RCL}. It may lose the global convergence property because of the elements with zero probability.
Definition of the RCL

How many elements are kept in the RCL?

There are two main strategies

- **cardinality-based**: the RCL contains the best $r$ elements of $\text{Ext}(x)$, where $r \in \{1, \ldots, |\text{Ext}(x)|\}$ is a user-defined parameter;
  - $r = 1$: deterministic constructive heuristic;
  - $r = |E|$: random walk.

- **value-based**: the RCL contains all elements of $\text{Ext}(xt)$ with value between $\phi_{\text{min}}$ and $\phi_{\text{min}} + \alpha(\phi_{\text{max}} - \phi_{\text{min}})$ where

  $$
  \phi_{\text{min}}(x) = \min_{i \in \text{Ext}(x)} \phi(i, x) \quad \phi_{\text{max}}(x) = \max_{i \in \text{Ext}(x)} \phi(i, x)
  $$

  and $\alpha \in [0; 1]$ is a user-defined parameter;
  - $\alpha = 0$: deterministic constructive heuristic;
  - $\alpha = 1$: random walk.
Algorithm GRASP(\(l\))

\[x^* := \emptyset; \quad z^* := +\infty;\]  \hspace{1cm} \{ Best \ incumbent \ solution \} 

For \(l = 1\) to \(\ell\) do 

\{ Non-deterministic constructive heuristic \} 

\(x := \emptyset;\)

While \(\text{Ext}(x) \neq \emptyset\) do 

\(\phi_i := \phi(i, x)\) for each \(i \in \text{Ext}(x)\)

\(L := \text{Sort}(\phi);\)

\(L := \text{BuildRCL}(L, \alpha);\)

\(i := \text{RandomExtract}(L);\)

\(x := x \cup \{i\};\)

If \(x \in X\) and \(z(x) < z^*\) then \(x^* := x; \quad z^* := z(x);\)

EndWhile;

\(x := \text{Search}(x);\)

EndFor;

Return \((x^*, z^*);\)
Example: GRASP for the Set Covering Problem

\[
\begin{array}{ccccc}
25 & 6 & 8 & 24 & 12 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
\end{array}
\]

1. start with the empty subset: \( x^{(0)} = \emptyset \)
2. build a RCL with \( \alpha = 2 \) candidates: columns 2 (\( \phi_2 = 2 \)) and 3 (\( \phi_3 = 4 \)); select (at random) column 3;
3. build a RCL with \( \alpha = 2 \) candidates: columns 2 (\( \phi_2 = 3 \)) and 1 (\( \phi_3 = 6.25 \)); select (at random) column 1;
4. the solution obtained is \( x = \{2, 1\} \) with cost \( z(x) = 33 \).

With \( \alpha = 2 \) one cannot obtain the optimal solution; with \( \alpha = 3 \) it is possible.
Reactive semi-greedy algorithm

There are parameters to calibrate:
- the number of iterations, \( \ell \)
- the size of the RCL, \( \alpha \)

An idea is to learn “good” values of the parameters from the results, exploiting memory:

1. set \( m \) configurations of the parameters \((\alpha_1, \ell), \ldots, (\alpha_m, \ell)\);
2. try each \( \alpha_r \) for \( \ell_r \) iterations (initially \( \ell_r = \ell \));
3. compute the average \( \bar{z}(\alpha) \) of the results obtained by each configuration \( \alpha_r \);
4. calibrate the number of iterations \( \ell = \ell_r \) for each \( \alpha_r \)

\[
\ell_r = \frac{1}{1 \sum_{s=1}^m \frac{1}{\bar{z}(\alpha_s)}} \bar{z}(\alpha_r)
\]

This favors the configurations that produce the best results.

5. while the total number \( R \) of iterations has not been reached, go to step 2.
Cost perturbation methods

Instead of
- forcing some decisions,
- forbidding some decisions,
- modifying the probability of the possible decisions,
one can modify the appeal of the feasible decisions.

Given a constructive heuristic $A$, at each iteration of iteration $l$
- modify the selection criterion $\phi(i, x)$ by a factor $\tau_{i,x}^{[l]} \in [0; 1]$

$$\psi^{[l]}(i, x) = \tau_{i,x}^{[l]} \phi(i, x)$$

- by updating $\tau_{i,x}^{[l]}$ at each iteration $l$ different solutions $x^{[l]}$ are generated.

The elements with a better value of $\phi(i, x)$ tend to remain favored, but $\tau_{i,x}^{[l]}$ modulates this effect, pushing the search to
- **diversification**: to obtain solutions different from the previous ones,
  one must increase $\tau_{i,x}^{[l]}$ for the most frequently selected elements;
- **intensification**: to obtain solutions similar to the previous ones,
  one must decrease $\tau_{i,x}^{[l]}$ for the most frequently selected elements.
Ant Colony Optimization

The ACO algorithm (Dorigo, Maniezzo and Colorni, 1991) is inspired by the social behavior of ants.

**Stigmergy** is defined as indirect communication between agents that are stimulated and guided by the results of the actions done by themselves and by the others.

Every agent (ant) is an instance of the constructive heuristic:
- it leaves a trace on the data, depending on the solution produced with them;
- it takes decisions on the basis of the traces left by all agents.

The decisions of each agent also have a randomized component.
Trace ("pheromone")

As in the semi-greedy heuristic
- a constructive heuristic is given;
- a randomized selection is done at each step.

Differently from the semi-greedy heuristic
- every iteration \( l \) runs the constructive heuristic \( f \) times (size of the population);
- all decisions in \( \text{Ext}(x) \) are allowed (no RCL is used);
- the probability \( \pi(i, x) \) depends on
  1. the selection criterion \( \phi(i, x) \)
  2. an auxiliary piece of information \( \tau(i, x) \) called trace, which is produced in the previous iterations and sometimes even in the same iteration by the other agents.

The trace is initially uniform \( (\tau(i, x) = \tau_0) \), and then it is modified
- it is increased to favor promising choices
- it is decreased to penalize too repetitive choices

For simplicity \( \tau(i, x) \) is assumed to depend only on the element \( i \) and not on \( x \).
Randomization

instead of selecting the best element according to $\phi(i, x)$, each element $i$ in $\text{Ext}(x)$ is assigned a probability

$$\pi(i, x) = \frac{\tau(i, x)^{\alpha_\tau} \eta(i, x)^{\alpha_\eta}}{\sum_{j \in \text{Ext}(x)} \tau(j, x)^{\alpha_\tau} \eta(i, x)^{\alpha_\eta}}$$

where the denominator normalizes the probability and

- one defines the visibility

$$\eta(i, x) = \begin{cases} 
\phi(i, x) & \text{for maximization problems} \\
\frac{1}{\phi(i, x)} & \text{for minimization problems}
\end{cases}$$

- $\alpha_\tau$ and $\alpha_\eta$ calibrate the effect of the trace $\tau$ and the visibility $\eta$
Trace update

At each iteration $l$

1. $f$ instances of the constructive heuristic are executed;
2. a subset $\tilde{X}^{[l]}$ of the generated solutions is selected, to favor its elements in the next iterations;
3. the trace is updated according to the following formula:

$$
\tau(i, x) := (1 - \rho)\tau(i, x) + \rho \sum_{y \in \tilde{X}^{[l]} : i \in y} F(y)
$$

where $\rho \in [0; 1]$ is an oblivion parameter:

- for $\rho = 1$, the old trace is completely lost,
- for $\rho = 0$, the old trace is kept as it is,

and $F(y)$ is a fitness function that expresses the quality of the solution $y$

(frequently $F(y) = Q/z(y)$ with a constant $Q$ such that $F > \tau$).

The update has two main effects:

1. it increases the trace on the elements belonging to the solutions in $\tilde{X}^{[l]}$;
2. it decreases the trace on the other elements.
The ACO heuristic

Algorithm AntSystem(I)

\[ x^* := \emptyset; \quad z^* := +\infty; \quad \{ \text{Best incumbent solution} \} \]

For \( l = 1 \) to \( \ell \) do

\[ \text{For } g = 1 \text{ to } f \text{ do} \]

\[ x := A(I); \quad \{ \text{Constructive heuristic } A \text{ with random steps and memory} \} \]

\[ x := \text{Search}(x); \quad \{ \text{Improvement with an exchange heuristic} \} \]

If \( z(x) < z^* \) then \( x^* := x; \quad z^* := z(x); \)

\[ \tau := \text{LocalUpdate}(\tau, x); \quad \{ \text{Local update of the trace} \} \]

EndFor;

\[ \tau := \text{GlobalUpdate}(\tau, x); \quad \{ \text{Global update of the trace} \} \]

EndFor;

Return \((x^*, z^*)\);
Selection of the influential solutions

\( \tilde{X}^{[l]} \) collects the solutions around which to search:

- in the classical Ant System, they are all the solutions at iteration \( l - 1 \);
- in elitist methods they are the best known solutions:
  - the best solution of iteration \( l - 1 \)
  - the best solution up to iteration \( l - 1 \)

With elitist methods

- one obtains better solutions earlier;
- one risks premature convergence; to avoid it, one needs additional techniques.
Data vs. memory

Some parameters calibrate the relative weight of data and memory:

- $\alpha_\eta \gg \alpha_\tau$ favors data: it tends to the constructive heuristic
- $\alpha_\eta \ll \alpha_\tau$ favors memory: it tends to reproduce the solutions in $\tilde{X}[t]

*Remark:* this holds for $\tau > 1$ and $F > 1$.

Small values of both push towards random search.

The Ant Colony System has $\alpha_\eta = \alpha_\tau = 1$, but at each iteration $t$ the heuristic

- chooses $i$ randomly with probability $q$
- chooses $i$ optimizing $\phi(i, x)$ with probability $(1 - q)$

In this way it obtains the two complementary actions:

- with $q \approx 1$ favors data
  It makes sense when the known solutions are bad/meaningless;
- with $q \approx 0$ favors memory
  It makes sense when the known solutions are good/meaningful.
Data vs. memory

The oblivion plays a similar role:

- **high oblivion** ($\rho \approx 1$) deletes the accumulated trace, because
  - we are not confident in the solutions obtained so far;
  - we want to explore different regions of the search space (diversification);
- **low oblivion** ($\rho \approx 0$) keeps the accumulated trace, because
  - we are confident in the solutions obtained so far;
  - we want to better explore the same region of the search space (intensification).
Other variations

- **MAX – MIN Ant System**: imposes a limited range \([\tau_{\text{min}}; \tau_{\text{max}}]\) to the trace values; it is experimentally tuned.

- **HyperCube Ant Colony Optimization (HC-ACO)**: normalizes the trace between 0 and 1.

- **Ant Colony System**:
  - Each iteration \(l\) modifies the trace (global update)
  - Each run \(g\) of the heuristic updates the trace reducing it, to make the next iterations unlikely to repeat the same choices (local update)

\[
\tau(i, x) := (1 - \rho)\tau(i, x) \quad \text{for each } i \in x^{[l,g]} \\
\text{(The aim is to diversify.)}
\]
Convergence to optimality

Some variants of the Ant System heuristic converge to optimality with probability 1 (Gutjahr, 2002)

The proof is based on the analysis of the construction graph:

- The trace $\tau(i, x)$ accumulates on the arcs $(x, x \cup \{i\})$;
- no information about the data is used, i.e. $\eta(i, x) \equiv 1$ (this strange assumption simplifies the analysis, but it is not necessary);
- $\tau^{[l]}$ is the trace function at the beginning of iteration $l$;
- $\gamma^{[l]}$ is the best path in the graph at the end of iteration $l$;
- $(\tau^{[l]}, \gamma^{[l-1]})$ is the state of a non-homogeneous Markov process:
  - the probability of each state depends only on the previous iteration;
  - the process is non-homogeneous because the dependence varies with $l$.

The proof concludes that for $\ell \to +\infty$, with probability 1

1. at least a run follows the optimal path in $\mathcal{F}$
2. the trace $\tau$ tends to a maximum along one of the optimal paths and to zero on the other arcs.
First variation with global convergence

The trace is updated with a variable oblivion coefficient

\[ \tau[l](i, x) := \begin{cases} (1 - \rho[l-1])\tau[l-1](i, x) + \rho[l-1] \frac{Q}{Z^{*}[l-1]} & \text{if } (x, x \cup \{i\}) \in \gamma[l-1] \\ (1 - \rho[l-1])\tau[l-1](i, x) & \text{otherwise} \end{cases} \]

where \( \gamma[l-1] \) is the best path found up to iteration \( l - 1 \) and \( z^{*}[l-1] \) is the value of the corresponding solution.

If the oblivion rate increases slowly enough

\[ \rho[l] \leq 1 - \frac{\log l}{\log(l + 1)} \quad \text{and} \quad \sum_{l=0}^{+\infty} \rho[l] = +\infty \]

then with probability 1 the state converges to \((\tau^{*}, \gamma^{*})\), where

- \( \gamma^{*} \) is the optimal path in the construction graph;
- \( \tau^{*}(i, x) = \frac{1}{Z^{*}} \) for \((x, x \cup \{i\}) \in \gamma^{*}, 0 \) otherwise.
Second variation with global convergence

Alternatively, if the oblivion rate $\rho$ remains constant, but a slowly decreasing lower threshold is imposed to the trace value,

$$\tau(i, x) \geq \frac{c_l}{\log(l + 1)} \quad \text{and} \quad \lim_{l \to +\infty} c_l > 0$$

then with probability 1 the state converges to $(\tau^*, \gamma^*)$.

In practice, all algorithms proposed so far

- associate the trace to subsets of arcs $(x, x \cup \{i\})$ (e.g. to each single element $i$)
- use constant parameters $\rho$ and $\tau_{\min}$. 