STOCHASTIC LOCAL SEARCH
FOUNDATIONS AND APPLICATIONS

# Generalised Local Search Machines

Holger H. Hoos & Thomas Stützle

# Outline

1. The Basic GLSM Model
2. State, Transition and Machine Types
3. Modelling SLS Methods Using GLSMs
4. Extensions of the Basic GLSM Model

# The Basic GLSM Model

Many high-performance SLS methods are based on combinations of *simple (pure) search strategies* (*e.g.*, ILS, MA).

These hybrid SLS methods operate on two levels:

- **lower level:** execution of underlying simple search strategies
- **higher level:** activation of and transition between lower-level search strategies.

**Key idea underlying Generalised Local Search Machines:**
Explicitly represent higher-level search control mechanism in the form of a *finite state machine*.

# The Basic GLSM Model

Many high-performance SLS methods are based on combinations of *simple (pure) search strategies* (*e.g.*, ILS, MA).

These hybrid SLS methods operate on two levels:

- **lower level:** execution of underlying simple search strategies
- **higher level:** activation of and transition between lower-level search strategies.

**Key idea underlying Generalised Local Search Machines:**
Explicitly represent higher-level search control mechanism in the form of a *finite state machine*.

# The Basic GLSM Model

Many high-performance SLS methods are based on combinations of *simple (pure) search strategies* (*e.g.*, ILS, MA).

These hybrid SLS methods operate on two levels:

- **lower level:** execution of underlying simple search strategies
- **higher level:** activation of and transition between lower-level search strategies.

**Key idea underlying Generalised Local Search Machines:**
Explicitly represent higher-level search control mechanism in the form of a *finite state machine*.

# The Basic GLSM Model

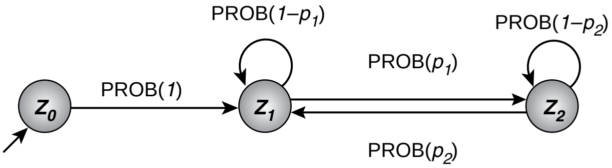
Many high-performance SLS methods are based on combinations of *simple (pure) search strategies* (*e.g.*, ILS, MA).

These hybrid SLS methods operate on two levels:

- **lower level:** execution of underlying simple search strategies
- **higher level:** activation of and transition between lower-level search strategies.

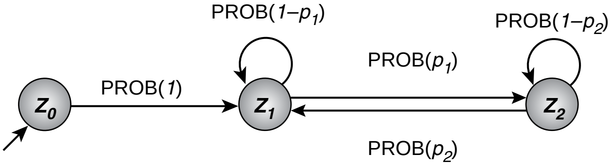
**Key idea underlying Generalised Local Search Machines:**
Explicitly represent higher-level search control mechanism in the form of a *finite state machine*.

## Example: Simple 3-state GLSM




- States $z_0, z_1, z_2$ represent simple search strategies, such as Random Picking (for initialisation), Iterative Best Improvement and Uninformed Random Walk.
- PROB($p$) refers to a probabilistic state transition with probability $p$ after each search step.

## Example: Simple 3-state GLSM




- States $z_0, z_1, z_2$ represent simple search strategies, such as Random Picking (for initialisation), Iterative Best Improvement and Uninformed Random Walk.
- PROB($p$) refers to a probabilistic state transition with probability $p$ after each search step.

## Generalised Local Search Machines (GLSMs)

- States $\cong$ simple search strategies.
- State transitions $\cong$ search control.

- GLSM $\mathcal{M}$ starts in initial state.
- In each iteration:
  - $\mathcal{M}$ executes one search step associated with its current state $z$;
  - $\mathcal{M}$ selects a new state (which may be the same as $z$) in a nondeterministic manner.
- $\mathcal{M}$ terminates when a given termination criterion is satisfied.

## Generalised Local Search Machines (GLSMs)

- States $\cong$ simple search strategies.
- State transitions $\cong$ search control.

- GLSM $\mathcal{M}$ starts in initial state.
- In each iteration:
  - $\mathcal{M}$ executes one search step associated with its current state $z$;
  - $\mathcal{M}$ selects a new state (which may be the same as $z$) in a nondeterministic manner.

- $\mathcal{M}$ terminates when a given termination criterion is satisfied.

## Generalised Local Search Machines (GLSMs)

- States $\cong$ simple search strategies.
- State transitions $\cong$ search control.
- GLSM $\mathcal{M}$ starts in initial state.
- In each iteration:
  - $\mathcal{M}$ executes one search step associated with its current state $z$;
  - $\mathcal{M}$ selects a new state (which may be the same as $z$) in a nondeterministic manner.
- $\mathcal{M}$ terminates when a given termination criterion is satisfied.

## Formal definition of a GLSM

A *Generalised Local Search Machine* is defined as a tuple $\mathcal{M} := (Z, z_0, M, m_0, \Delta, \sigma_Z, \sigma_\Delta, \tau_Z, \tau_\Delta)$ where:

- $Z$ is a set of states;
- $z_0 \in Z$ is the *initial state*;
- $M$ is a set of *memory states* (as in SLS definition);
- $m_0$ is the *initial memory state* (as in SLS definition);
- $\Delta \subseteq Z \times Z$ is the *transition relation*;
- $\sigma_Z$ and $\sigma_\Delta$ are sets of *state types* and *transition types*;
- $\tau_Z : Z \mapsto \sigma_Z$ and $\tau_\Delta : \Delta \mapsto \sigma_\Delta$ associate every state $z$ and transition $(z, z')$ with a state type $\sigma_Z(z)$ and transition type $\tau_\Delta((z, z'))$, respectively.

## Formal definition of a GLSM

A *Generalised Local Search Machine* is defined as a tuple $\mathcal{M} := (Z, z_0, M, m_0, \Delta, \sigma_Z, \sigma_\Delta, \tau_Z, \tau_\Delta)$ where:

- $Z$ is a set of states;
- $z_0 \in Z$ is the *initial state*;
- $M$ is a set of *memory states* (as in SLS definition);
- $m_0$ is the *initial memory state* (as in SLS definition);
- $\Delta \subseteq Z \times Z$ is the *transition relation*;
- $\sigma_Z$ and $\sigma_\Delta$ are sets of *state types* and *transition types*;
- $\tau_Z : Z \mapsto \sigma_Z$ and $\tau_\Delta : \Delta \mapsto \sigma_\Delta$ associate every state $z$ and transition $(z, z')$ with a state type $\sigma_Z(z)$ and transition type $\tau_\Delta((z, z'))$, respectively.

# Formal definition of a GLSM

A *Generalised Local Search Machine* is defined as a tuple $\mathcal{M} := (Z, z_0, M, m_0, \Delta, \sigma_Z, \sigma_\Delta, \tau_Z, \tau_\Delta)$ where:

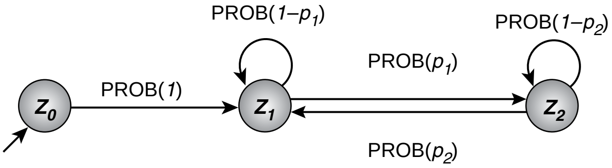
- $Z$ is a set of states;
- $z_0 \in Z$ is the *initial state*;
- $M$ is a set of *memory states* (as in SLS definition);
- $m_0$ is the *initial memory state* (as in SLS definition);
- $\Delta \subseteq Z \times Z$ is the *transition relation*;
- $\sigma_Z$ and $\sigma_\Delta$ are sets of *state types* and *transition types*;
- $\tau_Z : Z \mapsto \sigma_Z$ and $\tau_\Delta : \Delta \mapsto \sigma_\Delta$ associate every state $z$ and transition $(z, z')$ with a state type $\sigma_Z(z)$ and transition type $\tau_\Delta((z, z'))$, respectively.

## Formal definition of a GLSM

A *Generalised Local Search Machine* is defined as a tuple $\mathcal{M} := (Z, z_0, M, m_0, \Delta, \sigma_Z, \sigma_\Delta, \tau_Z, \tau_\Delta)$ where:

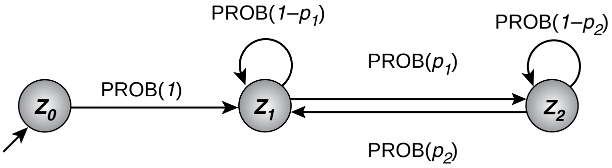
- $Z$ is a set of states;
- $z_0 \in Z$ is the *initial state*;
- $M$ is a set of *memory states* (as in SLS definition);
- $m_0$ is the *initial memory state* (as in SLS definition);
- $\Delta \subseteq Z \times Z$ is the *transition relation*;
- $\sigma_Z$ and $\sigma_\Delta$ are sets of *state types* and *transition types*;
- $\tau_Z : Z \mapsto \sigma_Z$ and $\tau_\Delta : \Delta \mapsto \sigma_\Delta$ associate every state $z$ and transition $(z, z')$ with a state type $\sigma_Z(z)$ and transition type $\tau_\Delta((z, z'))$, respectively.

## Example: Simple 3-state GLSM (formal definition)



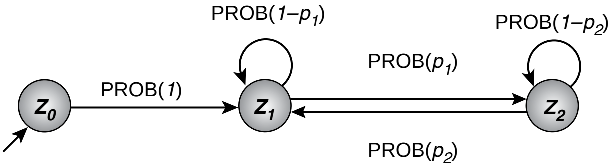

- $Z := \{z_0, z_1, z_2\}$; $z_0 =$ initial machine state
- no memory ($M := \{m_0\}$; $m_0 =$ initial and only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\text{PROB}(p) \mid p \in \{1, p_1, p_2, 1 - p_1, 1 - p_2\}\}$
- $\tau_Z(z_i) := z_i, \quad i \in \{0, 1, 2\}$
- $\tau_\Delta((z_0, z_1)) := \text{PROB}(1)$, $\tau_\Delta((z_1, z_2)) := \text{PROB}(p_1)$, ...

## Example: Simple 3-state GLSM (formal definition)



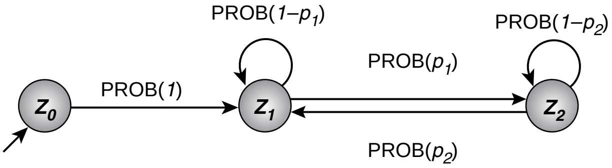

- $Z := \{z_0, z_1, z_2\}$; $z_0$ = initial machine state
- no memory ($M := \{m_0\}$; $m_0$ = initial and only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\text{PROB}(p) \mid p \in \{1, p_1, p_2, 1 - p_1, 1 - p_2\}\}$
- $\tau_Z(z_i) := z_i, \quad i \in \{0, 1, 2\}$
- $\tau_\Delta((z_0, z_1)) := \text{PROB}(1)$, $\tau_\Delta((z_1, z_2)) := \text{PROB}(p_1)$, ...

# Example: Simple 3-state GLSM (formal definition)



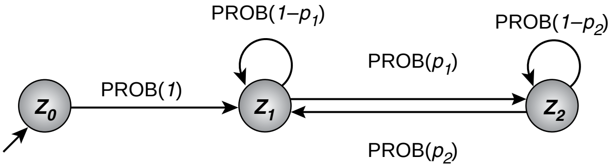

- $Z := \{z_0, z_1, z_2\}$; $z_0$ = initial machine state
- no memory ($M := \{m_0\}$; $m_0$ = initial and only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\text{PROB}(p) \mid p \in \{1, p_1, p_2, 1 - p_1, 1 - p_2\}\}$
- $\tau_Z(z_i) := z_i, \quad i \in \{0, 1, 2\}$
- $\tau_\Delta((z_0, z_1)) := \text{PROB}(1)$, $\tau_\Delta((z_1, z_2)) := \text{PROB}(p_1)$, ...

# Example: Simple 3-state GLSM (formal definition)




- $Z := \{z_0, z_1, z_2\}$; $z_0 =$ initial machine state
- no memory ($M := \{m_0\}$; $m_0 =$ initial and only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\text{PROB}(p) \mid p \in \{1, p_1, p_2, 1 - p_1, 1 - p_2\}\}$
- $\tau_Z(z_i) := z_i, \quad i \in \{0, 1, 2\}$
- $\tau_\Delta((z_0, z_1)) := \text{PROB}(1)$, $\tau_\Delta((z_1, z_2)) := \text{PROB}(p_1)$, ...

## Example: Simple 3-state GLSM (formal definition)




- $Z := \{z_0, z_1, z_2\}$; $z_0$ = initial machine state
- no memory ($M := \{m_0\}$; $m_0$ = initial and only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\mathrm{PROB}(p) \mid p \in \{1, p_1, p_2, 1-p_1, 1-p_2\}\}$
- $\tau_Z(z_i) := z_i, \quad i \in \{0, 1, 2\}$
- $\tau_\Delta((z_0, z_1)) := \mathrm{PROB}(1)$, $\tau_\Delta((z_1, z_2)) := \mathrm{PROB}(p_1)$, ...

## Example: Simple 3-state GLSM (semantics)

- Start in initial state $z_0$, memory state $m_0$ (never changes).
- Perform one search step according to search strategy associated with state type $z_0$ (*e.g.*, random picking).
- With probability 1, switch to state $z_1$.
- Perform one search step according to state $z_1$; switch to state $z_2$ with probability $p_1$, otherwise, remain in state $z_1$.
- In state $z_2$, perform one search step according to $z_2$; switch back to state $z_1$ with probability $p_2$, otherwise, remain in state $z_2$.

$\rightsquigarrow$ After one $z_0$ step (initialisation), repeatedly and nondeterministically switch between phases of $z_1$ and $z_2$ steps until termination criterion is satisfied.

## Example: Simple 3-state GLSM (semantics)

- Start in initial state $z_0$, memory state $m_0$ (never changes).
- Perform one search step according to search strategy associated with state type $z_0$ (*e.g.*, random picking).
- With probability 1, switch to state $z_1$.
- Perform one search step according to state $z_1$;
  switch to state $z_2$ with probability $p_1$,
  otherwise, remain in state $z_1$.
- In state $z_2$, perform one search step according to $z_2$;
  switch back to state $z_1$ with probability $p_2$,
  otherwise, remain in state $z_2$.

$\rightsquigarrow$ After one $z_0$ step (initialisation), repeatedly and nondeterministically switch between phases of $z_1$ and $z_2$ steps until termination criterion is satisfied.

## Example: Simple 3-state GLSM (semantics)

- Start in initial state $z_0$, memory state $m_0$ (never changes).
- Perform one search step according to search strategy associated with state type $z_0$ (*e.g.*, random picking).
- With probability 1, switch to state $z_1$.
- Perform one search step according to state $z_1$; switch to state $z_2$ with probability $p_1$, otherwise, remain in state $z_1$.
- In state $z_2$, perform one search step according to $z_2$; switch back to state $z_1$ with probability $p_2$, otherwise, remain in state $z_2$.

$\rightsquigarrow$ After one $z_0$ step (initialisation), repeatedly and nondeterministically switch between phases of $z_1$ and $z_2$ steps until termination criterion is satisfied.

## Example: Simple 3-state GLSM (semantics)

- Start in initial state $z_0$, memory state $m_0$ (never changes).
- Perform one search step according to search strategy associated with state type $z_0$ (*e.g.*, random picking).
- With probability 1, switch to state $z_1$.
- Perform one search step according to state $z_1$;
  switch to state $z_2$ with probability $p_1$,
  otherwise, remain in state $z_1$.
- In state $z_2$, perform one search step according to $z_2$;
  switch back to state $z_1$ with probability $p_2$,
  otherwise, remain in state $z_2$.

$\rightsquigarrow$ After one $z_0$ step (initialisation), repeatedly and nondeterministically switch between phases of $z_1$ and $z_2$ steps until termination criterion is satisfied.

## Example: Simple 3-state GLSM (semantics)

- Start in initial state $z_0$, memory state $m_0$ (never changes).
- Perform one search step according to search strategy associated with state type $z_0$ (*e.g.*, random picking).
- With probability 1, switch to state $z_1$.
- Perform one search step according to state $z_1$; switch to state $z_2$ with probability $p_1$, otherwise, remain in state $z_1$.
- In state $z_2$, perform one search step according to $z_2$; switch back to state $z_1$ with probability $p_2$, otherwise, remain in state $z_2$.

$\rightsquigarrow$ After one $z_0$ step (initialisation), repeatedly and nondeterministically switch between phases of $z_1$ and $z_2$ steps until termination criterion is satisfied.

## Example: Simple 3-state GLSM (semantics)

- Start in initial state $z_0$, memory state $m_0$ (never changes).
- Perform one search step according to search strategy associated with state type $z_0$ (*e.g.*, random picking).
- With probability 1, switch to state $z_1$.
- Perform one search step according to state $z_1$;
  switch to state $z_2$ with probability $p_1$,
  otherwise, remain in state $z_1$.
- In state $z_2$, perform one search step according to $z_2$;
  switch back to state $z_1$ with probability $p_2$,
  otherwise, remain in state $z_2$.

$\rightsquigarrow$ After one $z_0$ step (initialisation), repeatedly and nondeterministically switch between phases of $z_1$ and $z_2$ steps until termination criterion is satisfied.

## Note:

- *States types* formally represent (subsidiary) search strategies, whose definition is not part of the GLSM definition.
- *Transition types* formally represent mechanisms used for switching between GLSM states.
- Multiple states / transitions can have the same type.
- $\sigma_Z, \sigma_\Delta$ should include only state and transition types that are actually used in given GLSM ('no junk').
- Not all states in $Z$ may actually be reachable when running a given GLSM.
- *Termination condition* is not explicitly captured in GLSM model, but considered part of the execution environment.

## Note:

- *States types* formally represent (subsidiary) search strategies, whose definition is not part of the GLSM definition.
- *Transition types* formally represent mechanisms used for switching between GLSM states.
- Multiple states / transitions can have the same type.
- $\sigma_Z, \sigma_\Delta$ should include only state and transition types that are actually used in given GLSM ('no junk').
- Not all states in $Z$ may actually be reachable when running a given GLSM.
- *Termination condition* is not explicitly captured in GLSM model, but considered part of the execution environment.

Note:

- *States types* formally represent (subsidiary) search strategies, whose definition is not part of the GLSM definition.
- *Transition types* formally represent mechanisms used for switching between GLSM states.
- Multiple states / transitions can have the same type.
- $\sigma_Z, \sigma_\Delta$ should include only state and transition types that are actually used in given GLSM ('no junk').
- Not all states in $Z$ may actually be reachable when running a given GLSM.
- *Termination condition* is not explicitly captured in GLSM model, but considered part of the execution environment.

## Note:

- *States types* formally represent (subsidiary) search strategies, whose definition is not part of the GLSM definition.
- *Transition types* formally represent mechanisms used for switching between GLSM states.
- Multiple states / transitions can have the same type.
- $\sigma_Z, \sigma_\Delta$ should include only state and transition types that are actually used in given GLSM ('no junk').
- Not all states in $Z$ may actually be reachable when running a given GLSM.
- *Termination condition* is not explicitly captured in GLSM model, but considered part of the execution environment.

## Note:

- *States types* formally represent (subsidiary) search strategies, whose definition is not part of the GLSM definition.
- *Transition types* formally represent mechanisms used for switching between GLSM states.
- Multiple states / transitions can have the same type.
- $\sigma_Z, \sigma_\Delta$ should include only state and transition types that are actually used in given GLSM ('no junk').
- Not all states in $Z$ may actually be reachable when running a given GLSM.
- *Termination condition* is not explicitly captured in GLSM model, but considered part of the execution environment.

## Note:

- *States types* formally represent (subsidiary) search strategies, whose definition is not part of the GLSM definition.
- *Transition types* formally represent mechanisms used for switching between GLSM states.
- Multiple states / transitions can have the same type.
- $\sigma_Z, \sigma_\Delta$ should include only state and transition types that are actually used in given GLSM ('no junk').
- Not all states in $Z$ may actually be reachable when running a given GLSM.
- *Termination condition* is not explicitly captured in GLSM model, but considered part of the execution environment.

## GLSM Semantics

Behaviour of a GLSM is specified by *machine definition* $+$ *run-time environment* comprising specifications of

- state types,
- transition types;
- problem instance to be solved,
- search space,
- solution set,
- neighbourhood relations for subsidiary SLS algorithms;
- termination predicate for overall search process.

## GLSM Semantics

Behaviour of a GLSM is specified by *machine definition* + *run-time environment* comprising specifications of

- state types,
- transition types;
- problem instance to be solved,
- search space,
- solution set,
- neighbourhood relations for subsidiary SLS algorithms;
- termination predicate for overall search process.

## GLSM Semantics

Behaviour of a GLSM is specified by *machine definition* $+$ *run-time environment* comprising specifications of

- state types,
- transition types;
- problem instance to be solved,
- search space,
- solution set,
- neighbourhood relations for subsidiary SLS algorithms;
- termination predicate for overall search process.

**Run GLSM $\mathcal{M}$:**

set *current machine state* to $z_0$; set *current memory state* to $m_0$;

While *termination criterion* is not satisfied:

- perform *search step* according to type of current machine state;
  this results in a new *search position*
- select *new machine state* according to *types of transitions*
  from *current machine state*, possibly depending on
  *search position* and *current memory state*; this may
  change the *current memory state*

**Run GLSM $\mathcal{M}$:**

set *current machine state* to $z_0$; set *current memory state* to $m_0$;

While *termination criterion* is not satisfied:

perform *search step* according to type of current machine state; this results in a new *search position*

select *new machine state* according to *types of transitions* from *current machine state*, possibly depending on *search position* and *current memory state*; this may change the *current memory state*

**Run GLSM $\mathcal{M}$:**

set *current machine state* to $z_0$; set *current memory state* to $m_0$;

While *termination criterion* is not satisfied:

- perform *search step* according to type of current machine state; this results in a new *search position*

- select *new machine state* according to *types of transitions* from *current machine state*, possibly depending on *search position* and *current memory state*; this may change the *current memory state*

**Run GLSM $\mathcal{M}$:**

set *current machine state* to $z_0$; set *current memory state* to $m_0$;

While *termination criterion* is not satisfied:

- perform *search step* according to type of current machine state; this results in a new *search position*
- select *new machine state* according to *types of transitions* from *current machine state*, possibly depending on *search position* and *current memory state*; this may change the *current memory state*

Note:

- The *current search position* is only changed by the subsidiary search strategies associated with states, *not* as side-effect of machine state transitions.
- The *machine state* and *memory state* are only changed by state-transitions, *not* as side-effect of search steps. (Memory state is viewed as part of higher-level search control.)
- The operation of $\mathcal{M}$ is uniquely characterised by the evolution of *machine state*, *memory state* and *search position* over time.

Note:

- The *current search position* is only changed by the subsidiary search strategies associated with states, *not* as side-effect of machine state transitions.
- The *machine state* and *memory state* are only changed by state-transitions, *not* as side-effect of search steps. (Memory state is viewed as part of higher-level search control.)
- The operation of $\mathcal{M}$ is uniquely characterised by the evolution of *machine state*, *memory state* and *search position* over time.

Note:

- The *current search position* is only changed by the subsidiary search strategies associated with states, *not* as side-effect of machine state transitions.
- The *machine state* and *memory state* are only changed by state-transitions, *not* as side-effect of search steps. (Memory state is viewed as part of higher-level search control.)
- The operation of $\mathcal{M}$ is uniquely characterised by the evolution of *machine state*, *memory state* and *search position* over time.

## GLSMs are factored representations of SLS strategies:

- Given GLSM represents the way in which *initialisation* and *step function* of a hybrid SLS method are composed from respective functions of *subsidiary component SLS methods*.

- When modelling hybrid SLS methods using GLSMs, *subsidiary SLS methods* should be as simple and pure as possible, leaving *search control* to be represented explicitly at the GLSM level.

- *Initialisation* is modelled using *GLSM states* (advantage: simplicity and uniformity of model).

- *Termination of subsidiary search strategies* are often reflected in *conditional transitions* leaving respective GLSM states.

## GLSMs are factored representations of SLS strategies:

- Given GLSM represents the way in which *initialisation* and *step function* of a hybrid SLS method are composed from respective functions of *subsidiary component SLS methods*.
- When modelling hybrid SLS methods using GLSMs, *subsidiary SLS methods* should be as simple and pure as possible, leaving *search control* to be represented explicitly at the GLSM level.
- *Initialisation* is modelled using *GLSM states* (advantage: simplicity and uniformity of model).
- *Termination of subsidiary search strategies* are often reflected in *conditional transitions* leaving respective GLSM states.

## GLSMs are factored representations of SLS strategies:

- Given GLSM represents the way in which *initialisation* and *step function* of a hybrid SLS method are composed from respective functions of *subsidiary component SLS methods*.
- When modelling hybrid SLS methods using GLSMs, *subsidiary SLS methods* should be as simple and pure as possible, leaving *search control* to be represented explicitly at the GLSM level.
- *Initialisation* is modelled using *GLSM states* (advantage: simplicity and uniformity of model).
- *Termination of subsidiary search strategies* are often reflected in *conditional transitions* leaving respective GLSM states.

## GLSMs are factored representations of SLS strategies:

- Given GLSM represents the way in which *initialisation* and *step function* of a hybrid SLS method are composed from respective functions of *subsidiary component SLS methods*.

- When modelling hybrid SLS methods using GLSMs, *subsidiary SLS methods* should be as simple and pure as possible, leaving *search control* to be represented explicitly at the GLSM level.

- *Initialisation* is modelled using *GLSM states* (advantage: simplicity and uniformity of model).

- *Termination of subsidiary search strategies* are often reflected in *conditional transitions* leaving respective GLSM states.

# State, Transition and Machine Types

In order to completely specify the search method represented by a given GLSM, we need to define:

- the GLSM model (states, transitions, ...);
- the search method associated with each *state type*, *i.e.*, step functions for the respective subsidiary SLS methods;
- the semantics of each *transition type*, *i.e.*, under which conditions respective transitions are executed, and how they effect the memory state.

# State, Transition and Machine Types

In order to completely specify the search method represented by a given GLSM, we need to define:

- the GLSM model (states, transitions, ...);
- the search method associated with each *state type*, *i.e.*, step functions for the respective subsidiary SLS methods;
- the semantics of each *transition type*, *i.e.*, under which conditions respective transitions are executed, and how they effect the memory state.

# State, Transition and Machine Types

In order to completely specify the search method represented by a given GLSM, we need to define:

- the GLSM model (states, transitions, . . . );
- the search method associated with each *state type*, *i.e.*, step functions for the respective subsidiary SLS methods;
- the semantics of each *transition type*, *i.e.*, under which conditions respective transitions are executed, and how they effect the memory state.

## State types

- State type semantics are often most conveniently specified procedurally (see algorithm outlines for 'simple SLS methods' from Chapter 2).

- *initialising state type* = state type $\tau$ for which search position after one $\tau$ step is independent of search position before step.

  *initialising state* = state of initialising type.

- *parametric state type* = state type $\tau$ whose semantics depends on memory state.

  *parametric state* = state of parametric type.

## State types

- State type semantics are often most conveniently specified procedurally (see algorithm outlines for ‘simple SLS methods’ from Chapter 2).

- *initialising state type* $=$ state type $\tau$ for which search position after one $\tau$ step is independent of search position before step.

  *initialising state* $=$ state of initialising type.

- *parametric state type* $=$ state type $\tau$ whose semantics depends on memory state.

  *parametric state* $=$ state of parametric type.

## State types

- State type semantics are often most conveniently specified procedurally (see algorithm outlines for 'simple SLS methods' from Chapter 2).

- *initialising state type* $=$ state type $\tau$ for which search position after one $\tau$ step is independent of search position before step.

  *initialising state* $=$ state of initialising type.

- *parametric state type* $=$ state type $\tau$ whose semantics depends on memory state.

  *parametric state* $=$ state of parametric type.

## Transitions types (1)

- *Unconditional deterministic transitions* – type *DET*:
  - executed always and independently of memory state or search position;
  - every GLSM state can have at most one outgoing DET transition;
  - frequently used for leaving initialising states.
- *Conditional probabilistic transitions* – type *PROB(p)*:
  - executed with probability $p$, independently of memory state or search position;
  - probabilities of PROB transitions leaving any given state must sum to one.

## Transitions types (1)

- *Unconditional deterministic transitions* – type *DET*:
  - executed always and independently of memory state or search position;
  - every GLSM state can have at most one outgoing DET transition;
  - frequently used for leaving initialising states.
- *Conditional probabilistic transitions* – type *PROB(p)*:
  - executed with probability $p$, independently of memory state or search position;
  - probabilities of PROB transitions leaving any given state must sum to one.

## Transitions types (1)

- *Unconditional deterministic transitions* – type *DET*:
  - executed always and independently of memory state or search position;
  - every GLSM state can have at most one outgoing DET transition;
  - frequently used for leaving initialising states.
- *Conditional probabilistic transitions* – type *PROB(p)*:
  - executed with probability $p$, independently of memory state or search position;
  - probabilities of PROB transitions leaving any given state must sum to one.

## Transitions types (1)

- *Unconditional deterministic transitions* – type *DET*:
  - executed always and independently of memory state or search position;
  - every GLSM state can have at most one outgoing DET transition;
  - frequently used for leaving initialising states.
- *Conditional probabilistic transitions* – type *PROB(p)*:
  - executed with probability $p$, independently of memory state or search position;
  - probabilities of PROB transitions leaving any given state must sum to one.

## Transitions types (1)

- *Unconditional deterministic transitions* – type *DET*:
  - executed always and independently of memory state or search position;
  - every GLSM state can have at most one outgoing DET transition;
  - frequently used for leaving initialising states.
- *Conditional probabilistic transitions* – type *PROB(p)*:
  - executed with probability $p$, independently of memory state or search position;
  - probabilities of PROB transitions leaving any given state must sum to one.

Note:

- DET transitions are a special case of PROB transitions.
- For a GLSM $\mathcal{M}$ any state that can be reached from initial state $z_0$ by following a chain of PROB($p$) transitions with $p > 0$ will eventually be reached with arbitrarily high probability in any sufficiently long run of $\mathcal{M}$.
- In any state $z$ with a PROB($p$) self-transition $(z, z)$ with $p > 0$, the number of GLSM steps before leaving $z$ is distributed geometrically with mean and variance $1/p$.

## Note:

- DET transitions are a special case of PROB transitions.
- For a GLSM $\mathcal{M}$ any state that can be reached from initial state $z_0$ by following a chain of PROB($p$) transitions with $p > 0$ will eventually be reached with arbitrarily high probability in any sufficiently long run of $\mathcal{M}$.
- In any state $z$ with a PROB($p$) self-transition $(z, z)$ with $p > 0$, the number of GLSM steps before leaving $z$ is distributed geometrically with mean and variance $1/p$.

Note:

- DET transitions are a special case of PROB transitions.
- For a GLSM $\mathcal{M}$ any state that can be reached from initial state $z_0$ by following a chain of PROB($p$) transitions with $p > 0$ will eventually be reached with arbitrarily high probability in any sufficiently long run of $\mathcal{M}$.
- In any state $z$ with a PROB($p$) self-transition $(z, z)$ with $p > 0$, the number of GLSM steps before leaving $z$ is distributed geometrically with mean and variance $1/p$.

## Transitions types (2)

- *Conditional probabilistic transitions* – type *CPROB($C$, $p$)*:
  - executed with probability proportional to $p$ iff *condition predicate $C$* is satisfied;
  - all CPROB transitions from the current GLSM state whose condition predicates are not satisfied are *blocked*, *i.e.*, cannot be executed.

### Note:

- Special cases of CPROB($C, p$) transitions:
  - PROB($p$) transitions;
  - *conditional deterministic transitions*, type *CDET($C$)*.
- Condition predicates should be efficiently computable (ideally: $\leq$ linear time w.r.t. size of given problem instance).

## Transitions types (2)

- *Conditional probabilistic transitions* – type *CPROB*$(C, p)$:
  - executed with probability proportional to $p$ iff *condition predicate* $C$ is satisfied;
  - all CPROB transitions from the current GLSM state whose condition predicates are not satisfied are *blocked*, *i.e.*, cannot be executed.

Note:

- Special cases of CPROB$(C, p)$ transitions:
  - PROB$(p)$ transitions;
  - *conditional deterministic transitions*, type *CDET*$(C)$.
- Condition predicates should be efficiently computable (ideally: $\leq$ linear time w.r.t. size of given problem instance).

## Transitions types (2)

- *Conditional probabilistic transitions* – type *CPROB*($C, p$):
  - executed with probability proportional to $p$ iff *condition predicate* $C$ is satisfied;
  - all CPROB transitions from the current GLSM state whose condition predicates are not satisfied are *blocked*, *i.e.*, cannot be executed.

## Note:

- Special cases of CPROB($C, p$) transitions:
  - PROB($p$) transitions;
  - *conditional deterministic transitions*, type *CDET*($C$).
- Condition predicates should be efficiently computable (ideally: $\leq$ linear time w.r.t. size of given problem instance).

## Transitions types (2)

- *Conditional probabilistic transitions* – type *CPROB($C$, $p$)*:
  - executed with probability proportional to $p$ iff *condition predicate $C$* is satisfied;
  - all CPROB transitions from the current GLSM state whose condition predicates are not satisfied are *blocked*, *i.e.*, cannot be executed.

## Note:

- Special cases of CPROB($C$, $p$) transitions:
  - PROB($p$) transitions;
  - *conditional deterministic transitions*, type *CDET($C$)*.

- Condition predicates should be efficiently computable (ideally: $\leq$ linear time w.r.t. size of given problem instance).

## Commonly used simple condition predicates:

| | |
|---|---|
| $\top$ | always true |
| count($k$) | total number of GLSM steps $\geq k$ |
| countm($k$) | total number of GLSM steps modulo $k = 0$ |
| scount($k$) | number of GLSM steps in current state $\geq k$ |
| scountm($k$) | number of GLSM steps in current state modulo $k = 0$ |
| lmin | current candidate solution is a local minimum w.r.t. the given neighbourhood relation |
| evalf($y$) | current evaluation function value $\leq y$ |
| noimpr($k$) | incumbent candidate solution has not been improved within the last $k$ steps |

All based on local information; can also be used in negated form.

## Commonly used simple condition predicates:

| | |
|---|---|
| $\top$ | always true |
| count($k$) | total number of GLSM steps $\geq k$ |
| countm($k$) | total number of GLSM steps modulo $k = 0$ |
| scount($k$) | number of GLSM steps in current state $\geq k$ |
| scountm($k$) | number of GLSM steps in current state modulo $k = 0$ |
| lmin | current candidate solution is a local minimum w.r.t. the given neighbourhood relation |
| evalf($y$) | current evaluation function value $\leq y$ |
| noimpr($k$) | incumbent candidate solution has not been improved within the last $k$ steps |

All based on local information; can also be used in negated form.

## Commonly used simple condition predicates:

| | |
|---|---|
| $\top$ | always true |
| count($k$) | total number of GLSM steps $\geq k$ |
| countm($k$) | total number of GLSM steps modulo $k = 0$ |
| scount($k$) | number of GLSM steps in current state $\geq k$ |
| scountm($k$) | number of GLSM steps in current state modulo $k = 0$ |
| lmin | current candidate solution is a local minimum w.r.t. the given neighbourhood relation |
| evalf($y$) | current evaluation function value $\leq y$ |
| noimpr($k$) | incumbent candidate solution has not been improved within the last $k$ steps |

All based on local information; can also be used in negated form.

## Commonly used simple condition predicates:

| | |
|---|---|
| $\top$ | always true |
| count($k$) | total number of GLSM steps $\geq k$ |
| countm($k$) | total number of GLSM steps modulo $k = 0$ |
| scount($k$) | number of GLSM steps in current state $\geq k$ |
| scountm($k$) | number of GLSM steps in current state modulo $k = 0$ |
| lmin | current candidate solution is a local minimum w.r.t. the given neighbourhood relation |
| evalf($y$) | current evaluation function value $\leq y$ |
| noimpr($k$) | incumbent candidate solution has not been improved within the last $k$ steps |

All based on local information; can also be used in negated form.

## Commonly used simple condition predicates:

| | |
|---|---|
| $\top$ | always true |
| count($k$) | total number of GLSM steps $\geq k$ |
| countm($k$) | total number of GLSM steps modulo $k = 0$ |
| scount($k$) | number of GLSM steps in current state $\geq k$ |
| scountm($k$) | number of GLSM steps in current state modulo $k = 0$ |
| lmin | current candidate solution is a local minimum w.r.t. the given neighbourhood relation |
| evalf($y$) | current evaluation function value $\leq y$ |
| noimpr($k$) | incumbent candidate solution has not been improved within the last $k$ steps |

All based on local information; can also be used in negated form.

## Commonly used simple condition predicates:

| | |
|---|---|
| $\top$ | always true |
| count($k$) | total number of GLSM steps $\geq k$ |
| countm($k$) | total number of GLSM steps modulo $k = 0$ |
| scount($k$) | number of GLSM steps in current state $\geq k$ |
| scountm($k$) | number of GLSM steps in current state modulo $k = 0$ |
| lmin | current candidate solution is a local minimum w.r.t. the given neighbourhood relation |
| evalf($y$) | current evaluation function value $\leq y$ |
| noimpr($k$) | incumbent candidate solution has not been improved within the last $k$ steps |

All based on local information; can also be used in negated form.

## Transition actions:

- Associated with individual transitions; provide mechanism for modifying current memory states.
- Performed whenever GLSM executes respective transition.
- Modify memory state only, *cannot* modify GLSM state or search position.
- Have read-only access to search position and can hence be used, *e.g.*, to memorise current candidate solution.
- Can be added to any of the previously defined transition types.

## Transition actions:

- Associated with individual transitions; provide mechanism for modifying current memory states.
- Performed whenever GLSM executes respective transition.
- Modify memory state only, *cannot* modify GLSM state or search position.
- Have read-only access to search position and can hence be used, *e.g.*, to memorise current candidate solution.
- Can be added to any of the previously defined transition types.

## Transition actions:

- Associated with individual transitions; provide mechanism for modifying current memory states.
- Performed whenever GLSM executes respective transition.
- Modify memory state only, *cannot* modify GLSM state or search position.
- Have read-only access to search position and can hence be used, *e.g.*, to memorise current candidate solution.
- Can be added to any of the previously defined transition types.

## Transition actions:

- Associated with individual transitions; provide mechanism for modifying current memory states.
- Performed whenever GLSM executes respective transition.
- Modify memory state only, *cannot* modify GLSM state or search position.
- Have read-only access to search position and can hence be used, *e.g.*, to memorise current candidate solution.
- Can be added to any of the previously defined transition types.

## Transition actions:

- Associated with individual transitions; provide mechanism for modifying current memory states.
- Performed whenever GLSM executes respective transition.
- Modify memory state only, *cannot* modify GLSM state or search position.
- Have read-only access to search position and can hence be used, *e.g.*, to memorise current candidate solution.
- Can be added to any of the previously defined transition types.

## Machine types:

Capture *structure of search control mechanism*, obtained by abstracting from state and transition types of GLSMs.

- *1-state machines*:
  - simplest machine type, single initialising state only;
  - realises iterated sampling processes, such as Uninformed Random Picking.
- *1-state+init machines*:
  - one initialising + one working state;
  - good model for many simple SLS methods.

## Machine types:

Capture *structure of search control mechanism*, obtained by abstracting from state and transition types of GLSMs.

- *1-state machines*:
  - simplest machine type, single initialising state only;
  - realises iterated sampling processes, such as Uninformed Random Picking.
- *1-state+init machines*:
  - one initialising + one working state;
  - good model for many simple SLS methods.

## Machine types:

Capture *structure of search control mechanism*, obtained by abstracting from state and transition types of GLSMs.

- *1-state machines*:
  - simplest machine type, single initialising state only;
  - realises iterated sampling processes, such as Uninformed Random Picking.
- *1-state+init machines*:
  - one initialising + one working state;
  - good model for many simple SLS methods.

## Machine types:

Capture *structure of search control mechanism*, obtained by abstracting from state and transition types of GLSMs.

- *1-state machines*:
  - simplest machine type, single initialising state only;
  - realises iterated sampling processes, such as Uninformed Random Picking.
- *1-state+init machines*:
  - one initialising + one working state;
  - good model for many simple SLS methods.

## Machine types:

Capture *structure of search control mechanism*, obtained by abstracting from state and transition types of GLSMs.

- *1-state machines*:
  - simplest machine type, single initialising state only;
  - realises iterated sampling processes, such as Uninformed Random Picking.
- *1-state+init machines*:
  - one initialising + one working state;
  - good model for many simple SLS methods.

- *sequential 1-state machines*:


  - visit initialising state $z_0$ only on once.

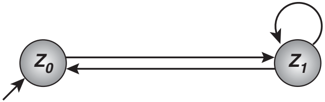
- *alternating 1-state+init machines*:

  - may visit initialising state $z_0$ multiple times;
  - good model for simple SLS methods with restart mechanism.

- *sequential 1-state machines*:

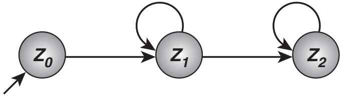

  - visit initialising state $z_0$ only on once.

- *alternating 1-state+init machines*:



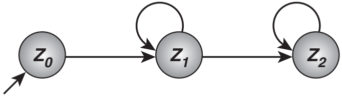

  - may visit initialising state $z_0$ multiple times;
  - good model for simple SLS methods with restart mechanism.

- *2-state+init sequential machines*:
  - one initialising state (visited only once), two working states;



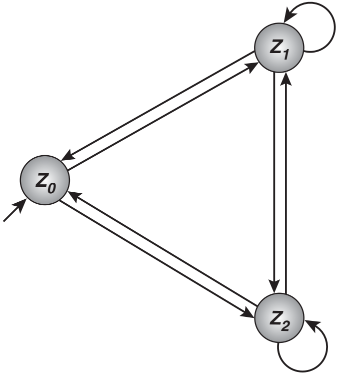

  - any search trajectory can be partitioned into three phases: one initialisation step, a sequence of $z_1$ steps and a sequence of $z_2$ steps.

- *2-state+init sequential machines*:
  - one initialising state (visited only once), two working states;




  - any search trajectory can be partitioned into three phases: one initialisation step, a sequence of $z_1$ steps and a sequence of $z_2$ steps.

- *2-state+init alternating machines*:
  - one initialising state, two working states;
  - arbitrary transitions between any states are possible.



## Generalisations:

- *k-state+init sequential machines*:
  - one initialising state (visited only once), $k$ working states;
  - every search trajectory consists of $1+k$ phases.

- *k-state+init alternating machines*:
  - one initialising state, $k$ working states;
  - arbitrary transitions between states;
  - may have multiple initialising states (*e.g.*, to realise alternative restart mechanisms).

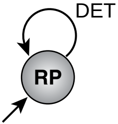
## Generalisations:

- *$k$-state+init sequential machines*:
  - one initialising state (visited only once), $k$ working states;
  - every search trajectory consists of $1+k$ phases.

- *$k$-state+init alternating machines*:
  - one initialising state, $k$ working states;
  - arbitrary transitions between states;
  - may have multiple initialising states (*e.g.*, to realise alternative restart mechanisms).

# Modelling SLS Methods Using GLSMs

## Uninformed Picking and Uninformed Random Walk





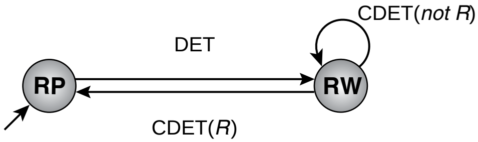
**procedure** *step-RP*$(\pi, s)$
  **input:** *problem instance* $\pi \in \Pi$,
    *candidate solution* $s \in S(\pi)$
  **output:** *candidate solution* $s \in S(\pi)$

  $s' := selectRandom(S)$;
  **return** $s'$
**end** *step-RP*

**procedure** *step-RW*$(\pi, s)$
  **input:** *problem instance* $\pi \in \Pi$,
    *candidate solution* $s \in S(\pi)$
  **output:** *candidate solution* $s \in S(\pi)$

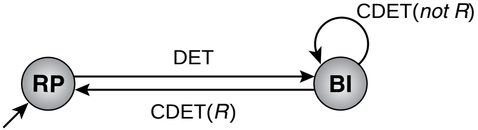
  $s' := selectRandom(N(s))$;
  **return** $s'$
**end** *step-RW*

## Uninformed Random Walk with Random Restart




$R$ = restart predicate, *e.g.*, countm($k$)

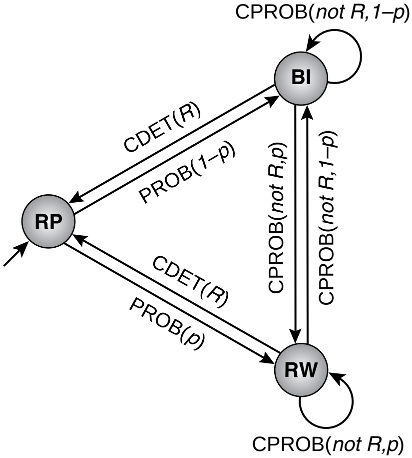
## Iterative Best Improvement with Random Restart



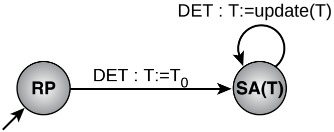

**procedure** *step-BI*($\pi, s$)
  **input:** *problem instance* $\pi \in \Pi$, *candidate solution* $s \in S(\pi)$
  **output:** *candidate solution* $s \in S(\pi)$

  $g^* := \min\{g(s') \mid s' \in N(s)\}$;
  $s' := selectRandom(\{s' \in N(s) \mid g(s') = g^*\})$;
  **return** $s'$
**end** *step-BI*

# Randomised Iterative Best Improvement with Random Restart

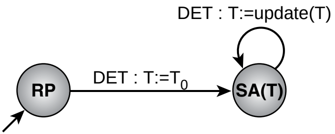


## Simulated Annealing



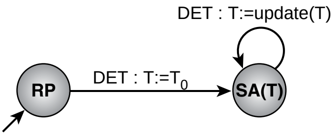

- Note the use of transition actions and memory for temperature $T$.
- The parametric state SA($T$) implements probabilistic improvement steps for given temperature $T$.
- The initial temperature $T_0$ and function *update* implement the annealing schedule.

## Simulated Annealing



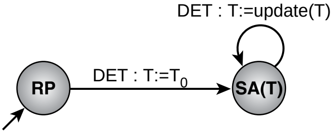

- Note the use of transition actions and memory for temperature $T$.
- The parametric state SA($T$) implements probabilistic improvement steps for given temperature $T$.
- The initial temperature $T_0$ and function *update* implement the annealing schedule.

## Simulated Annealing



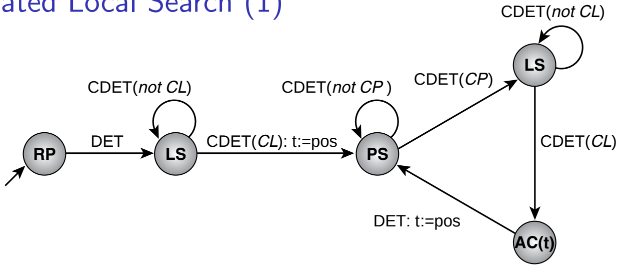

- Note the use of transition actions and memory for temperature $T$.
- The parametric state SA($T$) implements probabilistic improvement steps for given temperature $T$.
- The initial temperature $T_0$ and function *update* implement the annealing schedule.

## Simulated Annealing



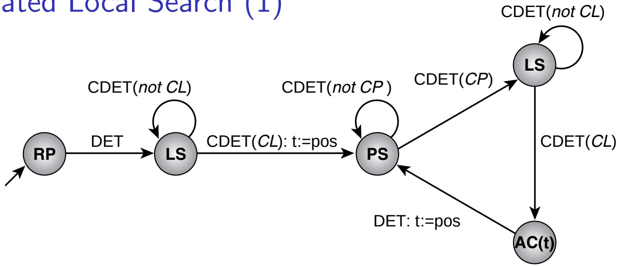

- Note the use of transition actions and memory for temperature $T$.
- The parametric state SA($T$) implements probabilistic improvement steps for given temperature $T$.
- The initial temperature $T_0$ and function *update* implement the annealing schedule.

# Iterated Local Search (1)



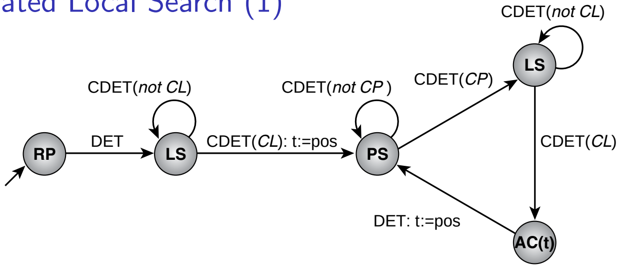

- The acceptance criterion is modelled as a state type, since it affects the search position.
- Note the use of transition actions for memorising the current candidate solution (pos) at the end of each local search phase.
- Condition predicates *CP* and *CL* determine the end of perturbation and local search phases, respectively; in many ILS algorithms, $CL :=$ lmin.

# Iterated Local Search (1)



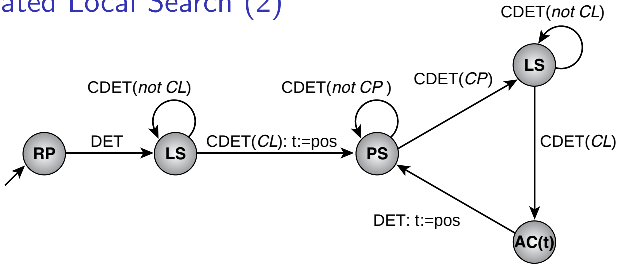

- The acceptance criterion is modelled as a state type, since it affects the search position.
- Note the use of transition actions for memorising the current candidate solution (pos) at the end of each local search phase.
- Condition predicates *CP* and *CL* determine the end of perturbation and local search phases, respectively; in many ILS algorithms, $CL :=$ lmin.

## Iterated Local Search (1)




- The acceptance criterion is modelled as a state type, since it affects the search position.
- Note the use of transition actions for memorising the current candidate solution (pos) at the end of each local search phase.
- Condition predicates *CP* and *CL* determine the end of perturbation and local search phases, respectively; in many ILS algorithms, $CL :=$ lmin.

# Iterated Local Search (2)




**procedure** *step-AC*$(\pi, s, t)$
  **input:** *problem instance* $\pi \in \Pi$,
    *candidate solution* $s \in S(\pi)$
  **output:** *candidate solution* $s \in S(\pi)$
  **if** $C(\pi, s, t)$ **then**
    **return** $s$
  **else**
    **return** $t$
  **end**
**end** *step-AC*

## Ant Colony Optimisation (1)

- General approach for modelling population-based SLS methods, such as ACO, as GLSMs:

  Define search positions as *sets of candidate solutions*; search steps manipulate some or all elements of these sets.

  *Example:* In this view, Iterative Improvement (II) applied to a population *sp* in each step performs one II step on each candidate solution from *sp* that is not already a local minimum.

  (Alternative approaches exist.)

- Pheromone levels are represented by memory states and are initialised and updated by means of transition actions.

## Ant Colony Optimisation (1)

- General approach for modelling population-based SLS methods, such as ACO, as GLSMs:

  Define search positions as *sets of candidate solutions*; search steps manipulate some or all elements of these sets.

  *Example:* In this view, Iterative Improvement (II) applied to a population *sp* in each step performs one II step on each candidate solution from *sp* that is not already a local minimum.

  (Alternative approaches exist.)

- Pheromone levels are represented by memory states and are initialised and updated by means of transition actions.

## Ant Colony Optimisation (1)

- General approach for modelling population-based SLS methods, such as ACO, as GLSMs:

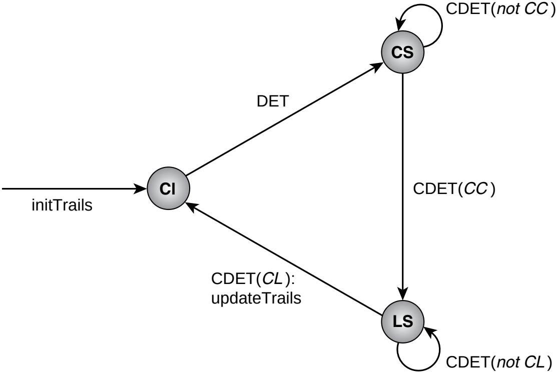
  Define search positions as *sets of candidate solutions*; search steps manipulate some or all elements of these sets.

  *Example:* In this view, Iterative Improvement (II) applied to a population *sp* in each step performs one II step on each candidate solution from *sp* that is not already a local minimum.

  (Alternative approaches exist.)

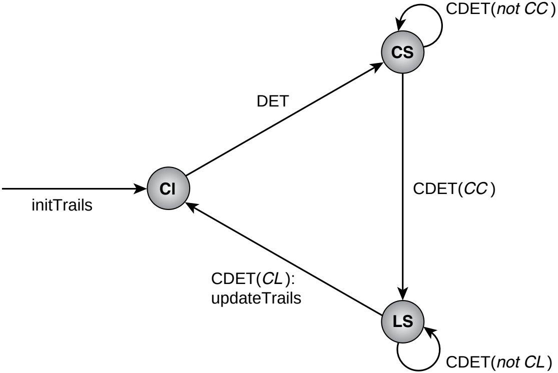
- Pheromone levels are represented by memory states and are initialised and updated by means of transition actions.

# Ant Colony Optimisation (2)



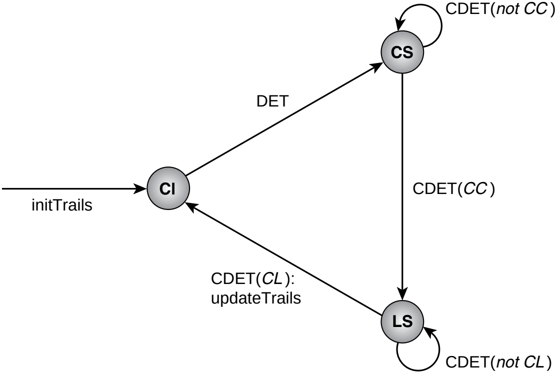

- The condition predicate *CC* determines the end of the construction phase.
- The condition predicate *CL* determines the end of the local search phase; in many ACO algorithms, *CL* := lmin.

# Ant Colony Optimisation (2)




- The condition predicate $CC$ determines the end of the construction phase.
- The condition predicate $CL$ determines the end of the local search phase; in many ACO algorithms, $CL :=$ lmin.

# Ant Colony Optimisation (2)




- The condition predicate *CC* determines the end of the construction phase.
- The condition predicate *CL* determines the end of the local search phase; in many ACO algorithms, $CL :=$ lmin.

# Extensions of the Basic GLSM Model

The basic GLSM model can be generalised and extended in various rather straightforward ways, such as:

- Co-operative GLSM models
- Learning GLSM models
- Evolutionary GLSM models
- Continuous GLSM models

**Note:** So far, these extensions remain mostly unexplored — lots of opportunities for interesting research!

# Extensions of the Basic GLSM Model

The basic GLSM model can be generalised and extended in various rather straightforward ways, such as:

- Co-operative GLSM models
- Learning GLSM models
- Evolutionary GLSM models
- Continuous GLSM models

**Note:** So far, these extensions remain mostly unexplored — lots of opportunities for interesting research!

## Co-operative GLSM models

- **Key idea:** Apply multiple GLSMs simultaneously to the same problem instance
- Naturally captures population-based SLS approaches.
- *Homogeneous co-operative GLSM models:*
  Population of identical GLSMs; equivalent to performing multiple independent runs of the respective SLS method.
- *Heterogenous co-operative GLSM models:*
  Population of different GLSMs; model *algorithm portfolios*.

## Co-operative GLSM models

- **Key idea:** Apply multiple GLSMs simultaneously to the same problem instance
- Naturally captures population-based SLS approaches.
- *Homogeneous co-operative GLSM models:*
  Population of identical GLSMs; equivalent to performing multiple independent runs of the respective SLS method.
- *Heterogenous co-operative GLSM models:*
  Population of different GLSMs; model *algorithm portfolios*.

## Co-operative GLSM models

- **Key idea:** Apply multiple GLSMs simultaneously to the same problem instance
- Naturally captures population-based SLS approaches.
- *Homogeneous co-operative GLSM models:*
  Population of identical GLSMs; equivalent to performing multiple independent runs of the respective SLS method.
- *Heterogenous co-operative GLSM models:*
  Population of different GLSMs; model *algorithm portfolios*.

## Co-operative GLSM models

- **Key idea:** Apply multiple GLSMs simultaneously to the same problem instance
- Naturally captures population-based SLS approaches.
- *Homogeneous co-operative GLSM models:*
  Population of identical GLSMs; equivalent to performing multiple independent runs of the respective SLS method.
- *Heterogenous co-operative GLSM models:*
  Population of different GLSMs; model *algorithm portfolios*.

## Co-operative GLSM models with communication

- GLSMs in population exchange information about their search trajectories, *e.g.*, via message passing or blackboard mechanism.
- Communication can be modelled via shared memory state or special transition actions (*e.g.*, *send*, *receive*).
- These models are naturally suited for representing population-based algorithms that use communication between individual search agents, such as ACO.

## Co-operative GLSM models with communication

- GLSMs in population exchange information about their search trajectories, *e.g.*, via message passing or blackboard mechanism.
- Communication can be modelled via shared memory state or special transition actions (*e.g.*, *send*, *receive*).
- These models are naturally suited for representing population-based algorithms that use communication between individual search agents, such as ACO.

## Co-operative GLSM models with communication

- GLSMs in population exchange information about their search trajectories, *e.g.*, via message passing or blackboard mechanism.
- Communication can be modelled via shared memory state or special transition actions (*e.g.*, *send*, *receive*).
- These models are naturally suited for representing population-based algorithms that use communication between individual search agents, such as ACO.

## Learning via dynamic transition probabilities

- **Key idea:** In a GLSM with probabilistic transitions, let transition probabilities evolve over time to adaptively optimise search control strategy.
- Can build on concepts from learning automata theory.
- *Single-instance learning:*
  Optimise control strategy on one problem instance during search process.
- *Multi-instance learning:*
  Adapt control strategies to features common to a class of problem instances.
- Transition probabilities can be adapted via external mechanism or via specialised transition actions.

## Learning via dynamic transition probabilities

- **Key idea:** In a GLSM with probabilistic transitions, let transition probabilities evolve over time to adaptively optimise search control strategy.
- Can build on concepts from learning automata theory.
- *Single-instance learning:*
  Optimise control strategy on one problem instance during search process.
- *Multi-instance learning:*
  Adapt control strategies to features common to a class of problem instances.
- Transition probabilities can be adapted via external mechanism or via specialised transition actions.

## Learning via dynamic transition probabilities

- **Key idea:** In a GLSM with probabilistic transitions, let transition probabilities evolve over time to adaptively optimise search control strategy.
- Can build on concepts from learning automata theory.
- *Single-instance learning:*
  Optimise control strategy on one problem instance during search process.
- *Multi-instance learning:*
  Adapt control strategies to features common to a class of problem instances.
- Transition probabilities can be adapted via external mechanism or via specialised transition actions.

## Learning via dynamic transition probabilities

- **Key idea:** In a GLSM with probabilistic transitions, let transition probabilities evolve over time to adaptively optimise search control strategy.
- Can build on concepts from learning automata theory.
- *Single-instance learning:*
  Optimise control strategy on one problem instance during search process.
- *Multi-instance learning:*
  Adapt control strategies to features common to a class of problem instances.
- Transition probabilities can be adapted via external mechanism or via specialised transition actions.

## Learning via dynamic transition probabilities

- **Key idea:** In a GLSM with probabilistic transitions, let transition probabilities evolve over time to adaptively optimise search control strategy.
- Can build on concepts from learning automata theory.
- *Single-instance learning:*
  Optimise control strategy on one problem instance during search process.
- *Multi-instance learning:*
  Adapt control strategies to features common to a class of problem instances.
- Transition probabilities can be adapted via external mechanism or via specialised transition actions.

## Evolutionary GLSM models

- **Key idea:** Achieve learning/adaptation in co-operative GLSM models by varying number or type of individual GLSMs over time.
- Distinction between *single*- and *multi-instance learning* as before; similar mechanisms for controlling adaptation process.
- Can easily model, for example, self-optimising portfolios of SLS algorithms.
- Further extensions:
  - support mutation / recombination operations on GLSMs;
  - additionally support learning in individual GLSMs
    $\rightsquigarrow$ evolving ensembles of dynamic GLSMs;
  - include communication between GLSMs in population.

## Evolutionary GLSM models

- **Key idea:** Achieve learning/adaptation in co-operative GLSM models by varying number or type of individual GLSMs over time.
- Distinction between *single-* and *multi-instance learning* as before; similar mechanisms for controlling adaptation process.
- Can easily model, for example, self-optimising portfolios of SLS algorithms.
- Further extensions:
  - support mutation / recombination operations on GLSMs;
  - additionally support learning in individual GLSMs
    $\rightsquigarrow$ evolving ensembles of dynamic GLSMs;
  - include communication between GLSMs in population.

## Evolutionary GLSM models

- **Key idea:** Achieve learning/adaptation in co-operative GLSM models by varying number or type of individual GLSMs over time.
- Distinction between *single-* and *multi-instance learning* as before; similar mechanisms for controlling adaptation process.
- Can easily model, for example, self-optimising portfolios of SLS algorithms.
- Further extensions:
  - support mutation / recombination operations on GLSMs;
  - additionally support learning in individual GLSMs
    $\rightsquigarrow$ evolving ensembles of dynamic GLSMs;
  - include communication between GLSMs in population.

## Evolutionary GLSM models

- **Key idea:** Achieve learning/adaptation in co-operative GLSM models by varying number or type of individual GLSMs over time.
- Distinction between *single-* and *multi-instance learning* as before; similar mechanisms for controlling adaptation process.
- Can easily model, for example, self-optimising portfolios of SLS algorithms.
- Further extensions:
  - support mutation / recombination operations on GLSMs;
  - additionally support learning in individual GLSMs $\rightsquigarrow$ evolving ensembles of dynamic GLSMs;
  - include communication between GLSMs in population.

## Evolutionary GLSM models

- **Key idea:** Achieve learning/adaptation in co-operative GLSM models by varying number or type of individual GLSMs over time.
- Distinction between *single-* and *multi-instance learning* as before; similar mechanisms for controlling adaptation process.
- Can easily model, for example, self-optimising portfolios of SLS algorithms.
- Further extensions:
  - support mutation / recombination operations on GLSMs;
  - additionally support learning in individual GLSMs $\rightsquigarrow$ evolving ensembles of dynamic GLSMs;
  - include communication between GLSMs in population.

## Evolutionary GLSM models

- **Key idea:** Achieve learning/adaptation in co-operative GLSM models by varying number or type of individual GLSMs over time.
- Distinction between *single-* and *multi-instance learning* as before; similar mechanisms for controlling adaptation process.
- Can easily model, for example, self-optimising portfolios of SLS algorithms.
- Further extensions:
  - support mutation / recombination operations on GLSMs;
  - additionally support learning in individual GLSMs
    $\rightsquigarrow$ evolving ensembles of dynamic GLSMs;
  - include communication between GLSMs in population.

## Continuous GLSM models

- *Note:* Many previously discussed hybrid SLS methods can be extended to continuous optimisation problems and give rise to high-performance algorithms for solving these.
- The main feature of the GLSM model, namely its clear distinction between *lower-level, simple search strategies* and *higher-level search control*, equally applies to continuous SLS algorithms.
- **Key idea:** Model complex continuous SLS methods by using continuous optimisation procedures as subsidiary local search strategies.

**Note:** The GLSM model is well-suited for modelling algorithms for *hybrid combinatorial problems* that involve discrete as well as continuous solution components.

## Continuous GLSM models

- *Note:* Many previously discussed hybrid SLS methods can be extended to continuous optimisation problems and give rise to high-performance algorithms for solving these.
- The main feature of the GLSM model, namely its clear distinction between *lower-level, simple search strategies* and *higher-level search control*, equally applies to continuous SLS algorithms.
- **Key idea:** Model complex continuous SLS methods by using continuous optimisation procedures as subsidiary local search strategies.

**Note:** The GLSM model is well-suited for modelling algorithms for *hybrid combinatorial problems* that involve discrete as well as continuous solution components.

## Continuous GLSM models

- *Note:* Many previously discussed hybrid SLS methods can be extended to continuous optimisation problems and give rise to high-performance algorithms for solving these.
- The main feature of the GLSM model, namely its clear distinction between *lower-level, simple search strategies* and *higher-level search control*, equally applies to continuous SLS algorithms.
- **Key idea:** Model complex continuous SLS methods by using continuous optimisation procedures as subsidiary local search strategies.

**Note:** The GLSM model is well-suited for modelling algorithms for *hybrid combinatorial problems* that involve discrete as well as continuous solution components.

## Continuous GLSM models

- *Note:* Many previously discussed hybrid SLS methods can be extended to continuous optimisation problems and give rise to high-performance algorithms for solving these.
- The main feature of the GLSM model, namely its clear distinction between *lower-level, simple search strategies* and *higher-level search control*, equally applies to continuous SLS algorithms.
- **Key idea:** Model complex continuous SLS methods by using continuous optimisation procedures as subsidiary local search strategies.

**Note:** The GLSM model is well-suited for modelling algorithms for *hybrid combinatorial problems* that involve discrete as well as continuous solution components.