Generalised Local Search Machines and Fitness Landscape Characterisation

FIT4012 Advanced topics in computational science

This material is based on the book 'Stochastic Local Search: Foundations and Applications' by Holger H. Hoos and Thomas Stützle (Morgan Kaufmann, 2004) - see www.sls-book.net for further information.

August 26, 2014

The Basic GLSM Model

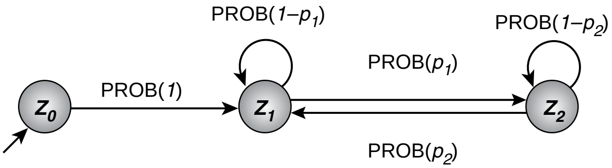
Many high-performance SLS methods are based on combinations of *simple (pure) search strategies* (*e.g.*, ILS, MA).

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.

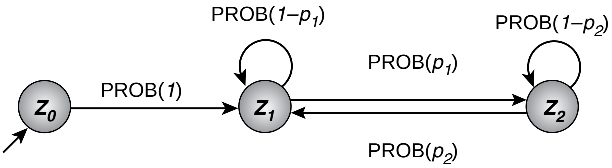
Key idea: Explicitly represent higher-level search control mechanism in the form of a *finite state machine*.

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*;
- σ_Z and σ_Δ 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*;
- σ_Z and σ_Δ 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:

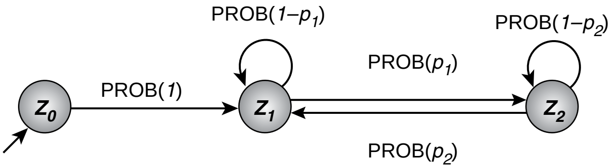
- Z is a set of states;
- $z_0 \in Z$ is the *initial state*;
- M is a set of *memory states* (as in SLS definition)
- m_0 is the *initial memory state* (as in SLS definition);
- $\Delta \subseteq Z \times Z$ is the *transition relation*;
- σ_Z and σ_Δ are sets of *state types* and *transition types*;
- $\tau_Z : Z \mapsto \sigma_Z$ 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:

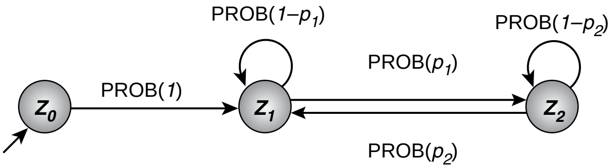
- Z is a set of states;
- $z_0 \in Z$ is the *initial state*;
- M is a set of *memory states* (as in SLS definition)
- m_0 is the *initial memory state* (as in SLS definition);
- $\Delta \subseteq Z \times Z$ is the *transition relation*;
- σ_Z and σ_Δ are sets of *state types* and *transition types*;
- $\tau_Z : Z \mapsto \sigma_Z$ 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)



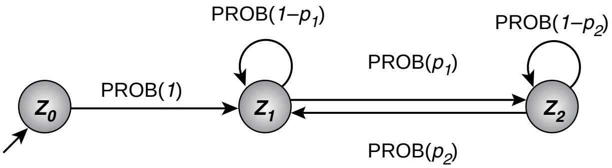
- $Z := \{z_0, z_1, z_2\}$; $z_0 =$ initial machine state
- no memory ($M := \{m_0\}$; $m_0 =$ initial & only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\text{PROB}(p) \mid p \in \{1, p_1, p_2, 1 -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

Example: Simple 3-state GLSM (formal definition)



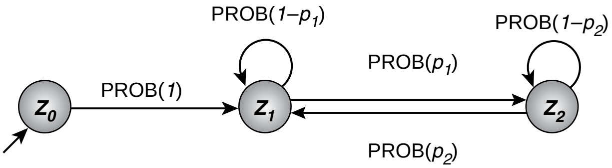
- $Z := \{z_0, z_1, z_2\}$; z_0 = initial machine state
- no memory ($M := \{m_0\}$; m_0 = initial & only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\text{PROB}(p) \mid p \in \{1, p_1, p_2, 1 - p_1,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

Example: Simple 3-state GLSM (formal definition)



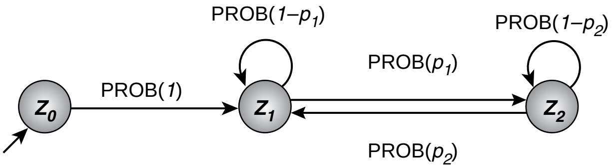
- $Z := \{z_0, z_1, z_2\}$; $z_0 =$ initial machine state
- no memory ($M := \{m_0\}$; $m_0 =$ initial & only memory state)
- $\Delta := \{(z_0, z_1), (z_1, z_2), (z_1, z_1), (z_2, z_1), (z_2, z_2)\}$
- $\sigma_Z := \{z_0, z_1, z_2\}$
- $\sigma_\Delta := \{\mathrm{PROB}(p) \mid p \in \{1, p_1, p_2, 1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

Example: Simple 3-state GLSM (formal definition)



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

- *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.
- σ_Z, σ_Δ 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.

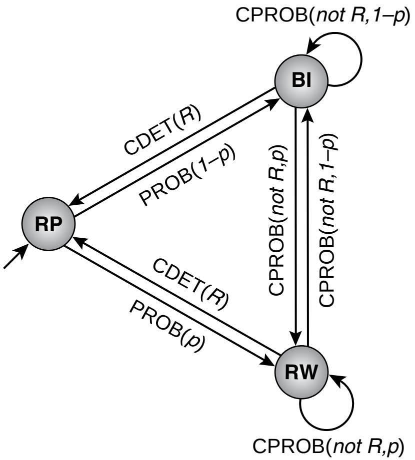
- *States types* formally represent (subsidiary) search strategies, whose definition is not part of the GLSM definition.
- *Transition types* formally represent mechanisms used for switching between GLSM states.
- σ_Z, σ_Δ should include only state and transition types that are actually used in given 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.

- *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.
- σ_Z, σ_Δ 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.

- *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.
- σ_Z, σ_Δ 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.

- *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.
- σ_Z, σ_Δ 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.

Example: Randomised Iterative Best Improvement with Random Restart



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*

- 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.

- 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.

- 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

- 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

- 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

- 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

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

  initialising state = state of initialising type.
- *parametric state type* = state type τ whose semantics depends on memory state.
  parametric state = state of parametric type.

State types

- State type semantics are often most conveniently specified procedurally.
- *initialising state type* = state type τ for which search position after one τ step is independent of search position before step.

  initialising state = state of initialising type.
- *parametric state type* = state type τ whose semantics depends on memory state.

  parametric state = state of parametric type.

State types

- State type semantics are often most conveniently specified procedurally.
- *initialising state type* $=$ state type τ for which search position after one τ step is independent of search position before step.

  initialising state $=$ state of initialising type.
- *parametric state type* $=$ state type τ whose semantics depends on memory state.
  parametric state $=$ state of parametric type.

Transitions types

- *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

- *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.

- 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 $1/p$.

- 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 $1/p$.

- 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

- *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.
- 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

- *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.
- 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

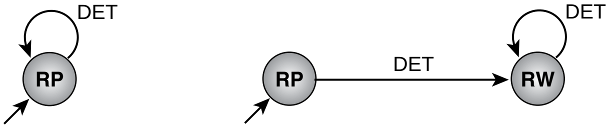
All based on local information; can also be used in negated form.

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 to memorise current candidate solution.
- Can be added to any of the previously defined transition types.

Modelling SLS Methods Using GLSMs

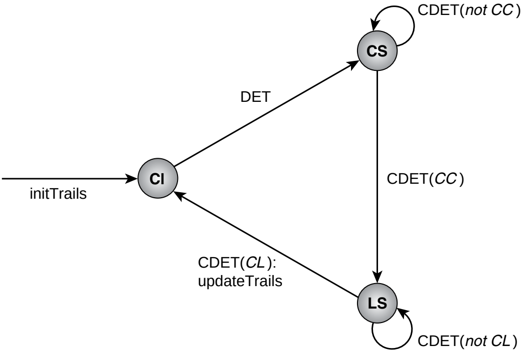
Uninformed Picking and Uninformed Random Walk



procedure *step-RP*(π, 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*

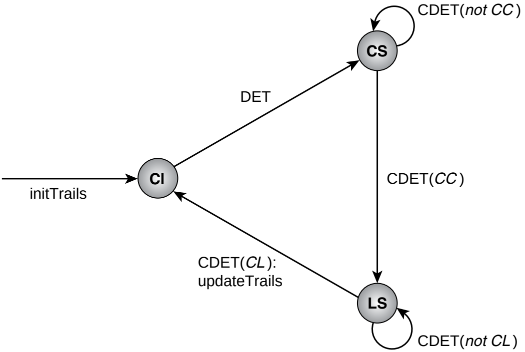
procedure *step-RW*(π, s)
input: *problem instance* $\pi \in \Pi$,
candidate solution $s \in S(\pi)$
output: *candidate solution* $s \in S(\pi)$
$s' := selectRandom(N(s))$;
return s'
end *step-RW*

Ant Colony Optimisation



- 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 algorithms, $CL :=$ lmin.

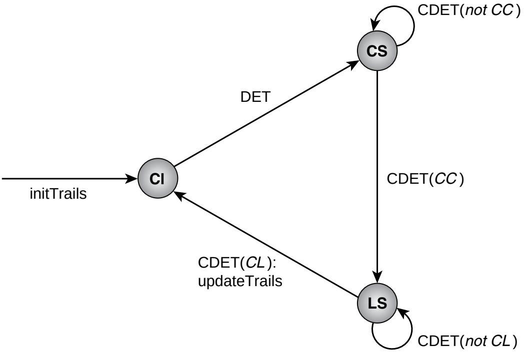
Ant Colony Optimisation



- 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 algorithms, $CL :=$ lmin.

Ant Colony Optimisation



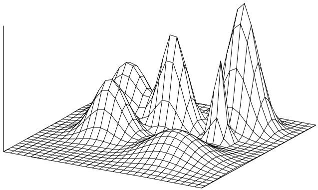
- The condition predicate CC determines the end of the construction phase.
- The condition predicate CL determines the end of the local search phase; in many algorithms, $CL :=$ lmin.

Fitness Landscape Characterisation

- study the search space
- description of the search space 'geometry'
- to understand what makes problems difficult
- to design effective search algorithms

Fitness landscapes in biology

Origin in biological science: Wright 1932 [45]

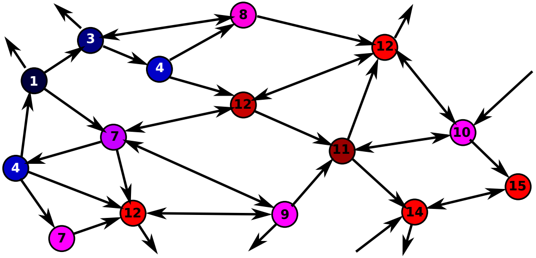


Fitness landscape is a graph (S, N , f)

- S is the search space
- $N : S \to 2^S$ is a neighbourhood relation
- $f : S \to R$ is a objective function

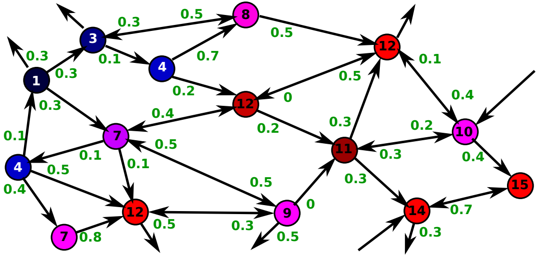
Fitness landscapes are graphs

- nodes are solutions which have a value (fitness),
- edges are defined by the neighbourhood relation.



Fitness landscapes are graphs

Specific local search puts probability transitions on edges according to f and history of the search

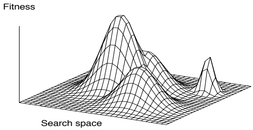


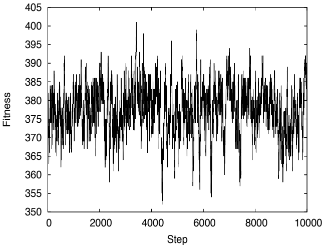
How does fitness landscape characterisation work?

- sample the neighbourhood to have information on local features of the search space
- from local information: deduce some global features like general shape of search space, ‘difficulty’, etc.
- study of the geometry of the landscape allows to study the difficulty, and design a good optimisation algorithm

What makes a problem difficult to optimise?

Number and size of attractive basins (Garnier et al [10])

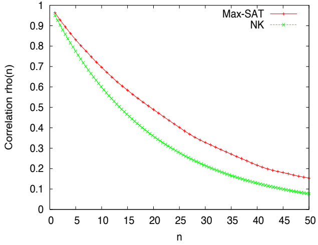




Ruggedness

Autocorrelation of time series of fitnesses $(f(s_1), f(s_2), \ldots)$ along a random walk $(s_1, s_2, \ldots)$ [37] :

$$\rho(n) = \frac{E[(f(s_i) - \overline{f})(f(s_{i+n}) - \overline{f})]}{var(f(s_i))} \quad (1)$$



Results

Problem	parameter	$\rho(1)$
symmetric TSP	n number of towns	$1 - \frac{4}{n}$
anti-symmetric TSP	n number of towns	$1 - \frac{4}{n-1}$
Graph Coloring Problem	n number of nodes α number of colors	$1 - \frac{2\alpha}{(\alpha-1)n}$
NK landscapes	N number of proteins K number of epistasis links	$1 - \frac{K+1}{N}$

Fitness Distance Correlation (FDC) (Jones 95 [15])

Correlation between distance to global optimum and fitness

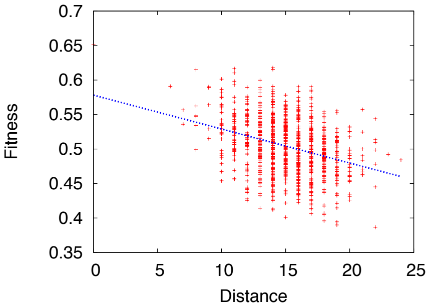
- given a set $F = f_1, f_2, ..., f_n$ of solution fitnesses
- with corresponding $D = d_1, d_2, ..., d_3$ Hamming distances to the nearest global optimum

$$\text{fdc} = \frac{(f_i - \overline{f})(d_i - \overline{d})}{n} \sum_{i=1}^{n} \tag{2}$$

$$r = \frac{\text{fdc}}{\sigma_F \sigma_D} \tag{3}$$

where σ_F and σ_D are the standard deviations of F and D.

FDC



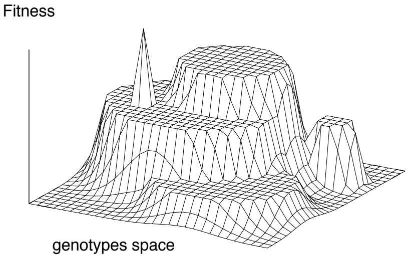
Classification based on experimental studies:

- $\rho < 0.15 \rightarrow$ easy optimization
- $\rho > 0.15 \rightarrow$ hard optimization
- $0.15 < \rho < 0.15 \rightarrow$ undecided zone

Neutral Fitness Landscapes

Neutral sets: set of solutions with the same fitness
Neutral networks: includes neighbourhood information

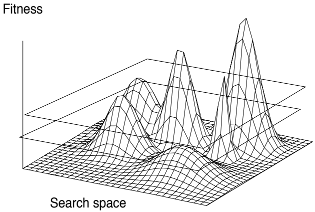
- Redundant problem (symmetries, ...) (Goldberg 87 [12])
- Problem 'not well' defined or dynamic environment (Torres 04 [14])

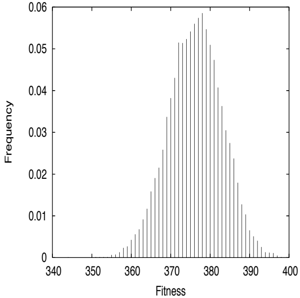


Neutrality and difficulty

- there is no definitive answer about neutrality / problem hardness
- certainly, it is dependent on the 'nature' of neutrality
- no information is better than bad information : Hard trap functions are more difficult than needle-in-a-haystack functions

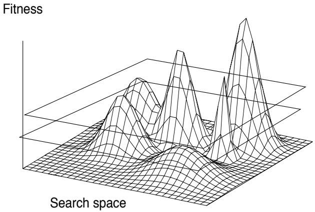
Measuring neutrality: Density Of States

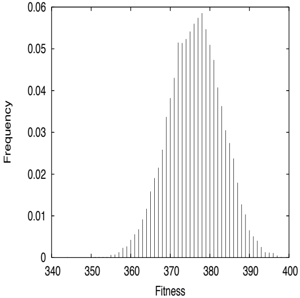




Tail of the distribution is an indicator of difficulty: the faster the decay, the harder the problem

Measuring neutrality: Density Of States

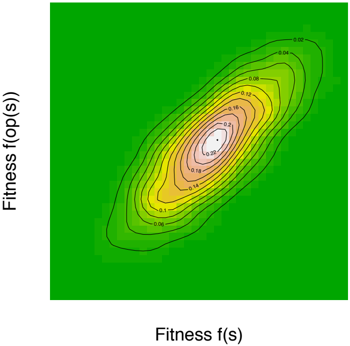


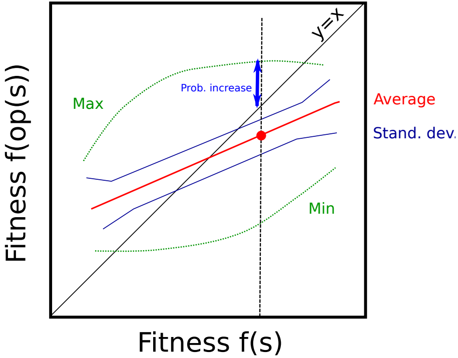


Tail of the distribution is an indicator of difficulty: the faster the decay, the harder the problem

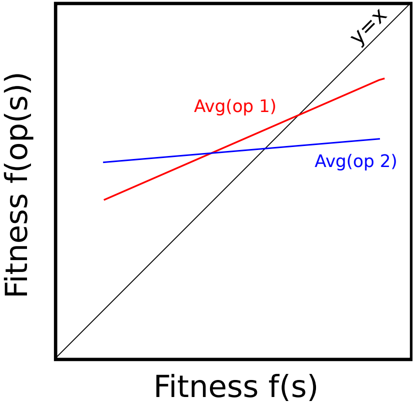
Measuring evolvability: Fitness Cloud [Verel et al. 2003]

Ability to evolve: fitness in the neighbourhood compared to the fitness of the solution



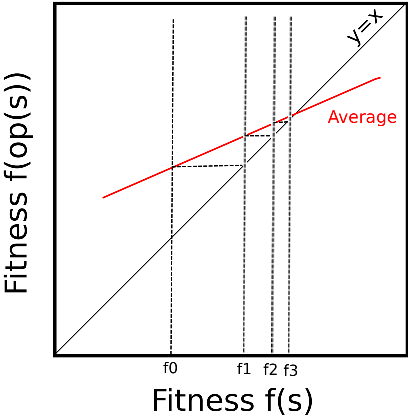


Evolvability as an indication of problem difficulty

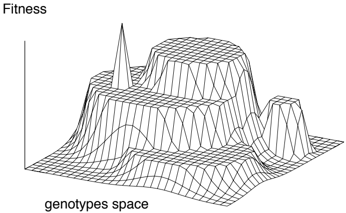


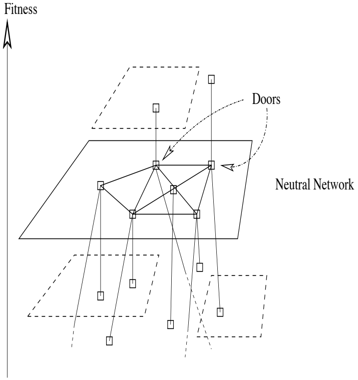
Predicting fitness using fitness clouds

- Approximation of the fitness value after few steps of local operator
- Indication on the quality of the operator



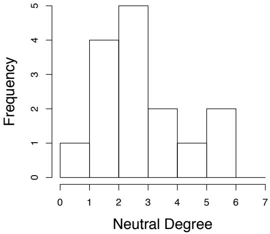
Neutral networks (Schuster 1994 [27])



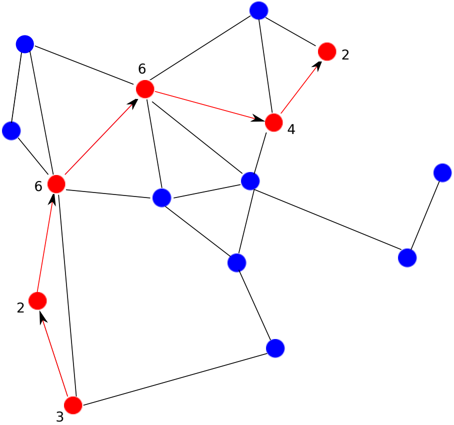


Metrics of neutral networks

- Size of NN: number of nodes of NN,
- Neutral degree distribution



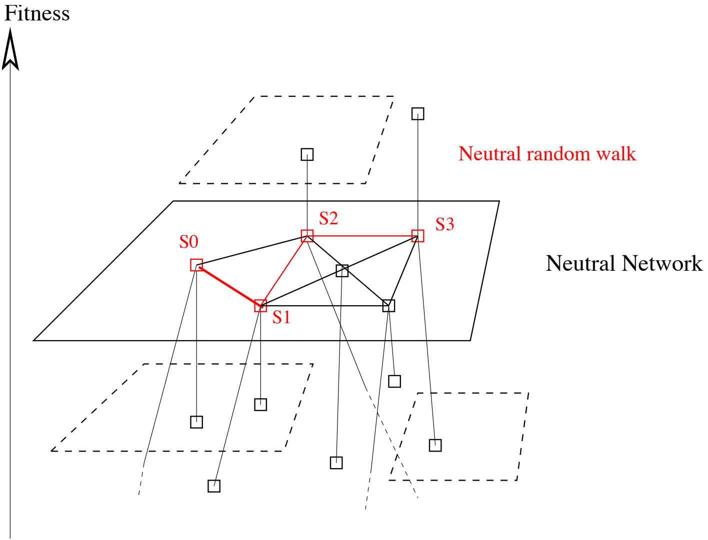
Autocorrelation of neutral degree (Bastolla 03 [3])



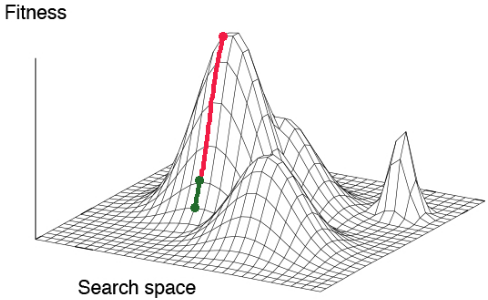
- random walk on NN
- autocorrelation of degrees

Rate of innovation (Huynen 96 [13])

The number of new accessible structures (fitness) per mutation



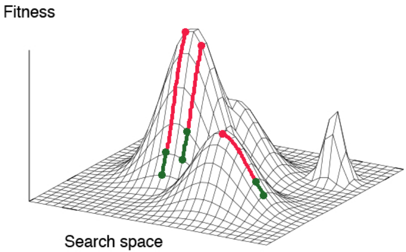
Predictors



Predictor definition:

- $p'_i = \frac{f(s'_i) - f(s_i)}{f(s'_i)}$
- $p''_i = \frac{f(s''_i) - f(s_i)}{f(s'_i) - f(s_i)}$

Predictive Local Search



Learning phase

- Predictor pool

$$\{(p'_1, p''_1), (p'_2, p''_2), \cdots, (p'_n, p''_n)\}$$

- Similarity measure

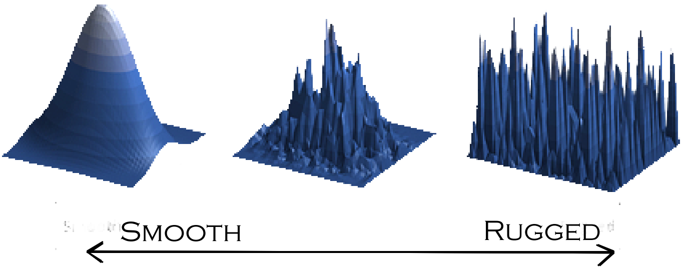
$$C = \frac{|p'_i - p'_j|}{|p'_i| + |p'_j|} + \frac{|p''_i - p''_j|}{|p''_i| + |p''_j|}$$

Predictive Local Search (continued)

Testing phase

1. Create a random solution
2. Calculate its fitness
3. Perform a local search (hill climbing) for a certain number of function evaluations
4. Calculate the fitness of the improved solution
5. Select the predictor that best matches the improvement in the first step
6. Perform a local search to the local optimum
7. Calculate the fitness of the local optimum
8. Calculate the error in prediction

How hard is an optimisation problem?



So what?

- The nature of the search-space is the key factor determining the performance of the optimisation algorithm,
- Define/characterize the search-space,
- Analyse what makes problems difficult,
- Guide the optimisation process
  - Select the right search strategy