Chapter 3

Generalised Local Search Machines

In this chapter, we introduce a novel formal framework for stochastic local search algorithms, the Generalised Local Search Machines (GLSM) model. The underlying idea is that adequate SLS algorithms are obtained by combining simple (pure) search strategies using a control mechanism; in the GLSM framework, the control mechanism is essentially realised by a non-deterministic finite state machine (FSM). This model provides a uniform framework capable of representing most modern SLS algorithms in an adequate way; it facilitates representations which clearly separate between search and search-control. As a consequence, the GLSM model can be very useful in developing and testing new, hybrid SLS algorithms. Furthermore, it offers analytical advantages, as well-known concepts from FSM theory can be applied to analyse SLS algorithms and search control mechanisms.

Here, we will mainly concentrate on general aspects of the GLSM model, while concrete GLSM realisations of SLS algorithms will be discussed in the following chapters. After a short introduction to local search algorithms, we introduce the basic GLSM model and define its semantics. Then, we establish the relation between the general scheme of local search and the GLSM model. Next, we discuss several structural GLSM types, transitions types, and state types. Finally, we will address extensions of the basic GLSM model, such as cooperative, evolutionary, and learning GLSMs. The chapter ends, as usual, with a brief overview of related work and a conclusion summarising its main contents.

3.1 Local Search Algorithms

Generally, local search algorithms are working in the following way: After initialising the search process at some point of the given problem instance's search space, the search iteratively moves from one position to a neighbouring position where the decision on each step is based on information about the local neighbourhood only. Thus, the following components

are required to define a local search procedure for a problem class Π applied to a given problem instance $\pi \in \Pi$:

- the *search space* S_π of P which is a set of *positions* $a \in S_\pi$ (also called locations or states)
- a *set of solutions* $S' \subseteq S_\pi$
- a *neighbourhood relation* $N \subseteq S_\pi \times S_\pi$ on S_π
- an *initial distribution* $init : S_\pi \mapsto \mathbb{R}$ for selecting the initial search positions
- a *step function* $step : S_\pi \mapsto (S_\pi \mapsto \mathbb{R})$ mapping each position onto a probability distribution over its neighbouring states, for specifying the local search steps.

Often, local search algorithms make use of an *objective function* $f : S_\pi \mapsto \mathbb{R}$, mapping each search space position onto a real number in such a way, that the global optima correspond to the solutions. Without loss of generality, in this work we will always assume that the solutions correspond to the global minima of the objective function. This objective function can also be used to define the step function. For optimisation problems, the values of the objective function usually correspond to the quantity which is optimised. This way, in the context of local search, decision problems and optimisation problems can be treated quite analogously; only for the former, the result of the local search algorithm is generally useless if it is not a global minimum, while for optimisation problems, suboptimal solutions (usually local minima) can be useful on their own.

Examples for local search algorithms are stochastic local hill-climbing [MJPL90, MJPL92, SLM92], steepest descent, Simulated Annealing [KJV83], Tabu Search [Glo89, Glo90], iterated local search [MOF91, Joh90], Evolutionary Algorithms [Rec73, Sch81], and Ant Colony Optimisation algorithms [DMC96]. These algorithms are applied to a variety of hard combinatorial optimisation and decision problems, like Satisfiability in Propositional Logic (SAT), Constrained Satisfaction Problems (CSPs), the Travelling Salesperson Problem (TSP), scheduling and planning problems, etc.

Note that for $\mathcal{NP}$-complete problems like SAT or CSP, the search space is typically exponentially larger than the problem size. However, for a problem like SAT, local search algorithms are not restricted to operate on a search space defined by a set of candidate solutions (*i.e.*, assignments in the SAT case). It is also possible to use search spaces consisting of a set of partial solutions (for SAT, partial assignments) which contain the actual solution candidates as a subset. Also for SAT, local search methods need not operate on assignments at all; instead, they could, for example, use the set of all resolution proofs of a given formula as a search space and thus be used to solve the complementary UNSAT or the equivalent VAL problem.

Modern local search algorithms are often a combination of several pure strategies, like steepest descent and random walk, tabu search and random restart, or the search and diversification phases in iterated local search. This suggests that these algorithms operate

on two levels: at a lower level, the pure search strategies are executed, while activation of and transitions between different strategies is controlled at a higher search control level. The GLSM model which we introduce in the following section is based on this distinction between search strategies and search control.

3.2 The Basic GLSM Model

Intuitively, a Generalised Local Search Machine (GLSM) for a given problem class Π is a finite state machine (FSM) each state of which corresponds to a simple local search strategy for instances of Π. The machine starts with an initial state s_0 and executes one step of the local search method associated with the current state. Then, according to a transition relation Δ, a new state is selected in a nondeterministic manner. This is iterated until either (1) a solution for the given problem instance is found or (2) a given maximal number ms of local search steps have been performed without satisfying condition (1). Note, that in this model the machine has no final states. Although it would be possible, and maybe more elegant from a theoretical point of view, to base the model on the notion of absorbing final states, this requires the introduction of additional states and transition types, which is avoided here for the sake of simplicity.

A *GLSM* is formally defined as a tuple $M = (S, s_0, \Delta, \sigma_S, \sigma_\Delta, \tau_S, \tau_\Delta)$ where S is a set of *states* and $s_0 \in S$ the initial state. $\Delta \subseteq S \times S$ is the *transition relation for* M; σ_S and σ_Δ are sets of state-types and transitions types resp., while $\tau_S : S \mapsto \sigma_S$ and $\tau_\Delta : \Delta \mapsto \sigma_\Delta$ associate the corresponding types to states and transitions. We call $\tau_S(s)$ the *type of state* s and $\tau_\Delta((s_1, s_2))$ the *type of transition* (s_1, s_2), respectively. The number $ms \in \mathbb{N}$ which specifies an upper bound on the number of steps, *i.e.*, state transitions, is not part of this definition because it is considered rather a parameter controlling the execution time behaviour than a structural aspect of the model.

It is useful to assume that σ_S, σ_Δ do not contain any types which are not associated with at least one state or transition of the given machine (*i.e.*, τ_S, τ_Δ are surjective). In this case, we define the *type of machine* M by $\tau_M := (\sigma_S, \sigma_\Delta)$. However, we allow for several states of M having the same type (*i.e.*, τ_s need not be injective). Note, that we do not require that each of the states in S can be actually reached when begining in state s_0; as we will shortly see, it is generally not trivial to decide this reachability. Thus, however, by adding unreachable states, the type of a given machine can be extended in an arbitrary way such that for any two GLSMs M_1, M_2, one can always find functionally equivalent models M_1', M_2' of the same type (*i.e.*, $\tau_{M_1'} = \tau_{M_2'}$).

The semantics of a GLSM M are defined by specifying an interpretation for its state- and transition types and then introducing a function $\pi^*_M : \mathbb{N} \mapsto \mathcal{D}(\Sigma)$, where $\mathbb{N}$ denotes the set of natural numbers and $\mathcal{D}(\Sigma)$ a distribution over the positions of search space Σ which is underlying the local search strategies for the given problem instance. Note that in the actual search process, there is only one position at each given instant. However, to capture the non-determinism of the search process, we have to use distributions over positions instead

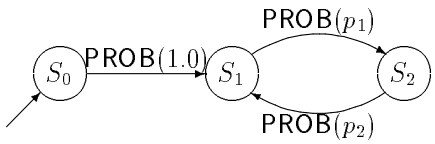

Figure 3.1: 3-state GLSM

when formalising the semantics of a GLSM. Intuitively, $\pi^*_M(n)$ determines the distribution over search space positions of the overall local search process realised by M at time n; it is therefore called the *search trajectory* of M.

Before giving a formal definition of the general semantics here, we demonstrate the specification and application of the GLSM concept by giving several examples. However, it should be noted that the specific semantics of a given GLSM always depends on the given problem class Π and the set of local search strategies which are used as interpretations of the state-types in σ_S. For the sake of simplicity, we will assume here that different state-types are always interpreted by different local search strategies (*i.e.*, the interpretation function for state-types is injective).

We will usually specify GLSMs by giving a graph representation for the finite state machine part (as it is commonly used in FSM theory) and additionaly labelling the states and transitions with their resp. types. As long as the meaning is clear from the context, we use the same symbol for denoting a state and its type. The initial state is marked by an ingoing arrow without a source. This is demonstrated for a small example in Figure 3.1; the corresponding machine is defined as

$$M = (\{S_0, S_1, S_2\}, S_0, \Delta, \sigma_S, \sigma_\Delta, \tau_S, \tau_\Delta)$$

with

$$\begin{aligned}
\Delta &= \{(S_0,S_1),(S_1,S_2),(S_2,S_1),(S_1,S_1),(S_2,S_2)\} \\
\sigma_S &= \{S_0, S_1, S_2\} \\
\sigma_\Delta &= \{\mathrm{PROB}(p) | p \in \mathbb{R}\} \\
\tau_S(S_i) &= S_i, \quad i \in \{1,2,3\} \\
\tau_\Delta((S_0,S_1)) &= \mathrm{PROB}(1.0) \\
\tau_\Delta((S_1,S_2)) &= \mathrm{PROB}(p_1) \\
\tau_\Delta((S_2,S_1)) &= \mathrm{PROB}(p_2) \\
\tau_\Delta((S_1,S_1)) &= \mathrm{PROB}(1-p_1) \\
\tau_\Delta((S_2,S_2)) &= \mathrm{PROB}(1-p_2)
\end{aligned}$$

The generic transition types PROB(p) correspond to unconditional, probabilistic transitions with an associated transition probability p. For simplicity's sake, we omit the transitions between a state and itself in the diagrammatic representation, using the default assumption, that whenever no other transition is selected, the next state is identical to the current state.

Thus the semantics of this small example can be intuitively described in the following man-

ner: For a given problem instance π, the local search process is initialised by setting the machine to its initial state S_0 and executing one local search step corresponding to state type S_0. With a probability of 1.0, the machine then switches to state S_1, executing one step of the local search strategy corresponding to state type S_1. Now, with a probability of p_1, the machine switches to state S_2, performing one local search step of type S_2; otherwise it remains in S_1 and does an S_1-step. When in S_2, an analogous behaviour is observed; resulting in a local search process which repeatedly and nondeterministically switches between S_1 and S_2 steps. However, only once in each run of the machine, an S_0 step is performed, and that is at the very begin of the local search process. As described above, the local search process terminates when either a solution for the given problem instance is found or a given number ms of local search steps has been performed without finding a solution. Note, that if a solution is found, the machine terminates in the state in which the solution was discovered.

3.3 GLSM Semantics

To formally define the semantics of a GLSM as described above, we assume that the semantics of each state type τ are already defined in form of a trajectory $\pi_\tau : \Sigma \mapsto \mathcal{D}(\Sigma)$, where Σ denotes the set of positions in the search space induced by the given problem instance, and $\mathcal{D}(\Sigma)$ the set of distributions over Σ. Intuitively, π_τ determines for each position in the search space the resulting position after one τ-step has been performed.

We further need the functions $\pi_s : \Sigma \times S \mapsto \mathcal{D}(S)$ which model the direct transitions between states of the GLSM. These are defined on the basis of the transitions from a given state s and their respective types.

The $\pi_s(a, s)$ are given by the specific transition types of $\tau((s, s'))$;
for $\tau((s_i, s_k)) = \text{PROB}(p_{i,k})$, $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = p_{i,k}$.

Remark: To facilitate a concise formulation of the definitions, we often use the functional form of discrete probability distributions; thus for $D = \{\ldots, (e, p), \ldots\}$, $D(e)$ is equal to p and denotes the probability of event e.

The direct state transition functions π_s can be generalised into functions $\pi'_s : D(\Sigma) \times D(S) \mapsto D(S)$, mapping state distributions onto distributions of seachspace positions instead of a single, fixed positions.

$\pi'_s(D, D_s) = D'_s$
with $D'_s(s_k) = \sum_{a \in \Sigma, s \in S} D''_s(s_k) \cdot D(a) \cdot D_s(s)$
where $D''_s = \pi_s(a, s)$.

Based on the functions π_τ and π_s, we next define the direct search transition function

$\pi : \Sigma \times S \mapsto D(\Sigma)$ which determines for a given search space position and a state distribution the search position distribution after one step of the GLSM.

$\pi(a_k, s) = D''$
with $D''(a_j) = P(\textit{go from state } s \textit{ to } s') \cdot P(\textit{in state } s' \textit{ go from } a_k \textit{ to } a_j)$.
$P(\textit{go from state } s \textit{ to } s') = D'_s(s)$
where $D'_s = \pi_s(a_k, s)$
$P(\textit{in state } s' \textit{ go from } a_k \textit{ to } a_j) = D'_a(a_j)$
where $D'_a = \pi_{\tau(s')}(a_k)$

Again, this is generalised into the corresponding function $\pi' : D(\Sigma) \times D(S) \mapsto D(\Sigma)$.

$\pi'(D, D_s) = D'$
with $D'(a_k) = \sum_{a \in \Sigma, s \in S} D''(a_k) \cdot D(a) \cdot D_s(s)$
where $D'' = \pi(a, s)$.

Finally, we inductively define the state and search position trajectories $\pi^*_s : \mathbb{N} \mapsto D(S)$ and $\pi^* : \mathbb{N} \mapsto D(\Sigma)$. The interlocked inductive definitions reflect the operation of a GLSM, where in each step, the next search space position and the next state are determined based on the current state and position. The initial search space position $a_0 \in S$ can be arbitrarily chosen, since usually the state distribution determined in the first step does not depend on a_0.

π^* and π^*_s are defined inductively by:
$\pi^*(0) = D_0$ with $D_0(a_0) = 1, \forall a_k \in S - \{a_0\} : D_0(a_k) = 0$
$\pi^*(t+1) = \pi'(\pi^*(t), \pi^*_s(t))$

$\pi^*_s(0) = s_0$
$\pi^*_s(t+1) = \pi'_s(\pi^*(t), \pi^*_s(t))$

Remark: To keep the definitions simple and concise, we did not consider the termination condition here (*cf.* Section 3.2); however, this can be easily incorporated into the semantics of the individual search states.

Actual GLSM trajectories

Based on the semantics defined above, it is quite simple to define the notion of an *actual search trajectory* $\delta^* : \mathbb{N} \mapsto \Sigma$ which determines a sequence of search space positions visited by a given GLSM when actually being executed. Note, that due to the inherent non-determinism of GLSMs, generally each actual search trajectory will only be observed with a certain probability. To formally define δ^*, we use a function $\mathit{draw}(D)$ which

for each probability distribution D, randomly selects an element e' from its domain such that $P(draw(D) = e') = D(e')$. Based on this, we define functions $\delta : \Sigma \times S \mapsto \Sigma$ and $\delta_s : \Sigma \times S \mapsto S$ which for each given position and state, randomly determine the position and state after one step of the GLSM:

$\delta(a, s) = draw(\pi'(a, s))$
$\delta_s(a, s) = draw(\pi'_s(a, s))$

Assuming that the given GLSM is started in state s_0 and at search position a_0, we now define the actual position and state trajectory by another double induction:

$\delta^* : \mathbb{N} \mapsto \Sigma$ and $\delta^*_s : \mathbb{N} \mapsto S$ are defined inductively by:
$\delta^*(0) = a_0$
$\delta^*(t+1) = \delta(\delta^*(t), \delta^*_s(t))$

$\delta^*_s(0) = s_0$
$\delta^*_s(t+1) = \delta_s(\delta^*(t), \delta^*_s(t))$

Note the similarity between these definitions and the ones for π^* and π^*_s; the only difference is in the use of the *draw* function to randomly select elements from the underlying probability distributions.

3.4 GLSMs and Local Search

The GLSM model has been introduced to provide a generalisation of the standard local search scheme presented in Section 3.1. Each GLSM, however, still realises a local search scheme and can therefore be described using the components of such a scheme. The notions of search space and solution set are not part of the model. This is mainly because both are not only problem- but actually instance-specific, they thus form the environment in which a given GLSM operates. Consequently, to characterise the behaviour of a GLSM when applied to a given instance, both the machine definition and this environment are required. The initial distribution is also not an explicit part of the GLSM model. The reason for this is the fact that the initial state, which is part of the model, can be easily used to generate arbitrary initial distributions of search space positions. The general local search scheme's step function is what is actually realised by the states of the GLSM and the finite control mechanism as given by the state transition relation.

The remaining component of the general local search scheme, the neighbourhood relation, generally does not have a direct representation in the GLSM model. This is because for a GLSM, each state type can be based on a different neighbourhood relation. However, for each GLSM as a whole, a neighbourhood relation can be defined by constructing a

Figure 3.2: Sequential *(left)* and alternating *(right)* 1-state+init GLSM.

generalised relation which contains the neighbourhood relation for each state type as a special case.

Taking a slightly different point of view, each GLSM state represents a local search algorithm of its own. While all these share one search space and solution set, they generally differ in their neighbourhood relations and step functions. In this case, however, initial distributions for the individual local search algorithms are not needed since they are defined by the context in which a GLSM state is activated.

3.5 Machine Types

One of the major advantages of using the GLSM model for characterising hybrid local search schemes is the clear distinction between search control and the actual search strategies thus facilitated. Abstracting from state and transition types, and thus concentrating on the structure of the search control mechanism alone, GLSMs can be categorised into the following structural classes:

1-state machines This is the minimal form of a GLSM. Since initialisation of the local search process has to be realised using a GLSM state, 1-state machines realise a very basic form of local search which basically only picks search space positions without doing actual local search. Consequently, the practical relevance of this machine type is extremely limited. It can, however, be used for analytical purposes, *e.g.* as a reference model when evaluating other types of GLSMs.

1-state+init machines These machines have one state for search initialisation and one working state, realising the search strategy. There are two sub-types of these machines: *1-state+init sequential machines* visit the initialisation state only once, while *alternating machines* might visit it again in the course of the search process, causing re-initialisations. As we will see in Chapter 4, most of today's popular SLS algorithms for SAT can be modelled by machines of this type.

2-state+init sequential machines This machine type has three states, one of which is an init state that is only visited once while the other two are working states. However, once the machine has switched from the first of these to the second, it will never switch back to the former again. Thus, each trajectory of such a machine can be partitioned into three phases: one initialisation step, a number of steps in the first working state and a number

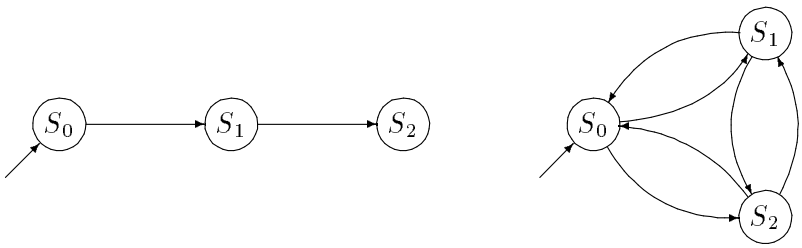

Figure 3.3: Sequential *(left)* and alternating *(right)* 2-state+init GLSM.

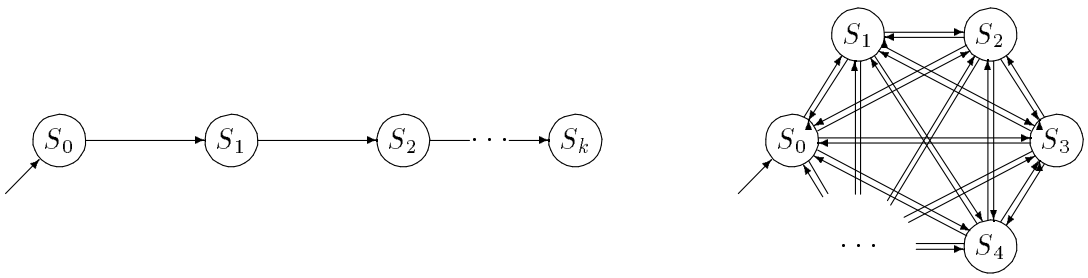

Figure 3.4: Sequential *(left)* and alternating *(right)* k-state+init GLSM.

of steps in the second working state.

2-state+init alternating machines Like the 2-state+init sequential machine, this machine type has one initialisation state and two working states. Here, however, arbitrary transitions between all states are possible. An interesting sub-type of these machines is given by the special case in which the initial state is only visited once, while the machine might arbitrarily switch between the two working states. Another sub-type that might be distinguished is a uni-directional cyclic machine model which allows the three states to be visited only in one fixed order.

Of course, the categorisation can be easily continued in this manner by successively increasing the number of working states. However, as we will later see, to describe state-of-the-art stochastic local search algorithms, usually three-state-machines are sufficient. We therefore conclude this categorisation with a brief look at two potentially interesting cases of the k-state+init machine types:

k-state+init sequential machines As a straightforward generalisation of the sequential 2-state+init machines, in this machine type we have $k+1$ states which are visited in a linear order. Consequently, after a machine state has been left, it will never be visited again.

k-state+init alternating machines These machines allow arbitrary transitions between the $k+1$ states and may therefore re-initialise the search process and switch between strategies as often as desired. Some special cases worth noting are the uni- and bi-directional cyclic machine models which allow to switch between states in a cyclic manner. In the

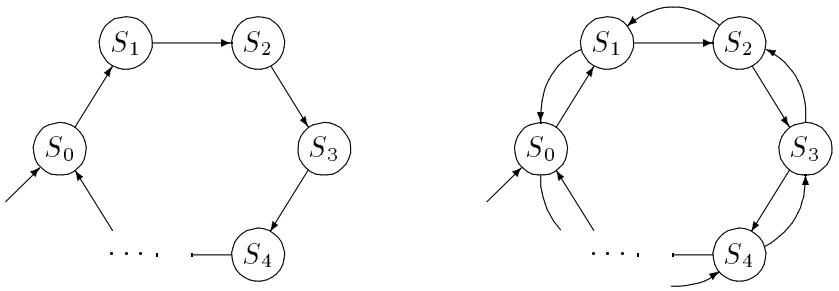

Figure 3.5: Uni-directional *(left)* and bi-directional *(right)* cyclic k-state+init GLSM.

former case, the cyclic structure can be traversed only in one direction, in the latter case the machine can switch from any state to both its neighbouring states.

This categorisation of machines according to their structure is useful for characterising the structural aspects of the search control as realised by the GLSM model. Of course, this is a very high-level view of GLSMs which can be refined in many ways, but nevertheless in the context of this work it will prove to be useful for capturing some fundamental differences between various stochastic local search schemes.

3.6 Transition Types

In refining the structural view of GLSMs given above, we next focus on transition types. As mentioned before, properties of the transition types are used as a basis for defining GLSM semantics. Here, we will introduce transition types in terms of a hierarchy of increasingly complex (or expressive) types and define the semantics in terms of the state transition functions π_s for each transition type.

Unconditional deterministic transitions, DET

This is the most basic transition type; DET transitions from state s_i to state s_k cause, when the GLSM is in state s_i, always an immediate transition into state s_k. Formally, if $\tau((s_i, s_k)) = \text{DET}$, this behaviour is captured by $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = 1$ for arbitrary $a \in \Sigma$. Note that because D''_s is a probability distribution, the above condition implies $D''_s(s) = 0$ for all states $s \neq s_k$.

The use of this transition type is fairly limited, because it causes a state with such a transition as its source to be left immediately after being entered. This obviously implies that for each state there can be only one transition leaving it. Consequently, using exclusively DET transitions, one can only realise a very limited class of GLSM structures. However, at least for the transition leaving the initial state, DET transitions are frequently used in practically occurring GLSMs.

Unconditional probabilistic transitions, PROB(p)

A PROB(p) transition from the actual state is executed with probability p. Thus, DET transitions are actually equivalent to a special case of this transition type, namely to PROB(1). For the moment, we can therefore assume without loss of generality that all transition types in a given GLSM are of type PROB. To define the semantics of this transition type, we consider an arbitrary GLSM state s_i and assume that the set of transitions leaving s_i is given as $\{t_1, \ldots, t_n\}$. If further $\tau(t_j) = \text{PROB}(p_j)$, we can define the semantics of PROB transitions by $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = p'$ for arbitrary $a \in \Sigma$ where p' is given by $\tau((s_i, s_k)) = \text{PROB}(p')$. Note that to guarantee that D''_s is a probability distribution, and $\sum_{j=1}^{n} p_j$ must be equal to one.

Note that by using PROB(0) transitions we can restrict ourselves to fully connected GLSMs, where for each pair of states (s_i, s_k), a transition of type PROB(p_{ik}) is defined. This allows a more uniform representation of this class of GLSMs which in turn will facilitate both theoretical investigations and practical implementations of this GLSM type. Furthermore, the behaviour of these GLSMs can be easily modeled using Markov processes [Çin75], which facilitates their analysis, as well-known techniques for studying Markov processes can be directly applied.

Conditional probabilistic transitions, CPROP(C, p)

While until now we have focused on transitions the execution of which only depends on the actual state, the next generalisation from PROB(p) introduces context dependent transitions. A CPROP(C, p) transition from state s_i to state s_k is executed with a probability proportional to p only when a condition predicate C is satisfied. If C is not satisfied, all transitions CPROP(C, p) from the current state are blocked, *i.e.*, they cannot be executed. For practical reasons, the condition predicates C will depend on local information only; this includes information on the current search space position, its immediate neighbourhood, the number of local search steps performed up to this point, and, possibly, some simple statistics on these. We will see later some predicates which can be practically used. Generally, the crucial condition restricting the choice of condition predicates C is that these have to be efficiently computable (when compared to the cost for executing local search steps).

Obviously, PROB(p) transitions are equivalent to CPROB$(\top, p)$ conditional probabilistic transitions, where $\top$ is the predicate which is always true. Without loss of generality, we can therefore assume that for a given GLSM all transitions are of type CPROB(C, p). To define the semantics of this transition type we consider the actual GLSM state s_i. As s_i is the actual state, we have all the local information to decide the condition predicates of all transitions leaving s_i. Since in this situation only a *non-blocked* transition can be executed, *i.e.*, a transition for which C is satisfied and therefore equivalend to $\top$ in the given situation, we can now define the semantics like in the case of PROB(p) transitions. To this end, we assume that the set of *non-blocked* transitions leaving s_i is given as $\{t_1, \ldots, t_n\}$, and for $\tau(t_j) = \text{CPROB}(C_j, p_j)$, C_j currently satisfied, we define $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = p'/c$

$\top$	always true
tcount(k)	total number of local search steps $\geq k$
mcount(k)	total number of local search modulo $k = 0$
lmin	current local search position is local minimum w.r.t. its direct neighbours
obf(x)	current objective function value $\leq x$
$\neg$ impr(k)	objective function value could not be improved within last k steps

Table 3.1: Some examples for simple condition predicates.

for arbitrary $a \in \Sigma$ where p' is given by $\tau((s_i, s_k)) = \text{CPROB}(C', p')$ and $c = \sum_{j=1}^{n} p_j$ is the normalisation factor which ensures that D_s'' is a probability distribution.

An important special case of conditional transitions are *conditional deterministic* transitions. These occur, if for a given GLSM state s_i, from all the transitions leaving it at most one transition is not blocked. One way to obtain deterministic GLSMs using conditional transitions is to make sure that by their logical structure, the condition predicates for the transitions leaving each state are mutually exclusive (or non-overlapping). Generally, depending on the nature of the condition predicates used, the decision whether a conditional transition is deterministic or not can be very difficult. For the same reasons it can be difficult to decide for a given GLSM with conditional probabilistic transitions, whether a particular state is reachable from the initial state.

For practically using GLSMs with conditional transitions it is important to make sure that the condition predicates can be evaluated in a sufficiently efficient way. There are two kinds of condition predicates, the first of which is based on the search space position and/or its local neighbourhood. The other, however, is based on search-control aspects alone, like the time that has been spent in the current GLSM state, or the overall run-time. Of course, these two kinds of conditions can also be mixed. Some concrete examples for condition predicates can be seen in Table 3.1. Note that all these predicates are based on only local information and can thus be efficiently evaluated during the search.

Usually, for each condition predicate, a positive as well as a negative (negated) form will be defined. Using propositional connectives like "$\wedge$" or "$\vee$", these simple predicates can be combined into complex predicates. However, it is not difficult to see that every GLSM using complex condition predicates can be reduced to an equivalent GLSM using only simple predicates by introducing additional states and/or transitions. Thus, using complex condition predicates can generally be avoided without restricting the expressive power of the model.

Transition actions

After discussing a hierarchy of increasingly general transition types, we now introduce another conceptual element into the context of transitions: transition actions. Transition actions are associated with the individual transitions and are executed whenever the GLSM

executes the corresponding transition. At this point, the motivation for adding this notion to the GLSM model might not be obvious. However, as we will see, there are some situations in which transition actions provide an adequate method of modelling SLS algorithms. One such case is the manipulation of global search parameters, like adapting the length of a tabu list or a noise parameter.

Generally, transition actions can be added to each of the transition types defined above, while the semantics of the transition (in terms of π_s) is not affected. If T is a transition type, by $T : A$ we denote the same transition type with associated action A. The nature of the actions A has to be such that they neither directly affect the state of the GLSM, nor its search space position. Instead, the actions generally can be used for

- modifying global parameters of one or more state types,
- realisation of input / output functionality in actual GLSM realisations,
- communication between GLSMs in cooperative GLSM models.

By introducing an action NOP without any effects we obtain uniform GLSMs in which all transitions have associated actions. Note, however, that we do not need multiple actions (*i.e.*, sequences or sets of actions which are associated with the same transition), because by introducing copies of a transition's destination state the (intuitive) semantics of multiple actions can be emulated.

From a minimalist point of view, of course, even simple transition actions are not strictly required because they, too, can be emulated by embedding the corresponding actions into the search strategies associated with the GLSM states. This, however, means to mix conceptually different notions, namely the local search strategies and the actions which are rather part of the search control mechanism that is represented by the modified finite state machines underlying the GLSM model. Because here our main motivation is to devise an *adequate* representation of SLS algorithms, if the notion of transition actions occurs naturally, we prefer to rather model them explicitly in the way outlined above.

3.7 State Types

At this point, the only component for specifying concrete GLSMs which is still missing are state types. As outlined above, for formally specifying the semantics of a GLSM, the semantics of the individual state types are required to be specified in the form of a trajectory $\pi_\tau : \Sigma \mapsto D(\Sigma)$. For practical purposes, however, state types will usually be rather defined in a procedural way, usually by using some form of pseudo-code. In some cases, more adequate descriptions of complex state types can be obtained by using other formalisms, such as decision trees. Concrete examples for various state types will be give in Chapter 4, where we show how existing local search algorithms for SAT can be represented as GLSMs.

Here, we want to concentrate on some fundamental distinctions between certain state types. One of these concerns the role of the state within the general local search scheme presented in Section 3.1. Since we are modelling the search initialisation and local search steps using the same mechanism, namely GLSM states, there is a distinction between initialisation states and search step states. An initialisation state is usually different from a search step state in that it is left after one corresponding step has been performed. Also, while search step states correspond to moves in a restricted local neigbourhood (like flipping one variable in SAT), one initialisation step can lead to arbitrary search space positions (like a randomly chosen assignment of all variables of a SAT instance). Formally, we define an *initialising state type* as a state type τ, for which the local search position after one τ-step is independent of the local search position before the step; the states of an initialising type τ are called *initialising states*. Generally, each GLSM will have at least one initialising state, which is also its initial state. A GLSM can, however, have more than one initialising state and use these states for dynamically restarting the local search process.

If for a given problem there is a natural common neighbourhood relation for local search, we can also distinguish *single-step states* from *multi-step states*. For the SAT problem, most local search algorithms use a neighbourhood relation where two variable assignments are direct neighbours if they differ in exactly one variable's value. In this context, a single-step state would flip one variable's value in each step, whereas a multi-step state could flip several variables per local search step. Consequently, initialising states are an extreme case of multi-step states, since they can affect all variable's values at the same time.

3.8 Extensions of the Basic GLSM Model

In this section we discuss various extensions of the basic GLSM model. One of the strengths of the GLSM model lies in the fact that these extensions arise quite naturally and can be easily realised within the basic framework. However, in the context of this work none of the extensions proposed here has been studied in detail, with the exception of the homogeneous cooperative model discussed later in this section. The main reason for this lies in the fact that in the context of SAT and CSP, the existing state-of-the-art SLS algorithms are conceptually fairly simple but nevertheless very powerful. Consequently, as we will see later, improvements of these algorithms can be achieved using simple GLSM techniques, but this requires a rather detailed understanding of their behaviour. The extensions as outlined in the following are of a more complex nature, but at the same time can be applied to all domains for which local search techniques are available.

Learning via dynamic transition probabilities

One of the features of the basic GLSM model with probabilistic transitions is the fact that the transition probabilities are static, *i.e.*, they are fixed when designing the GLSM. An obvious generalisation, along the lines of learning automata theory [NT89], is to let the

transition probabilities evolve over time as the GLSM is running. The search control in this model corresponds to a variable structure learning automaton. The environment such a dynamic GLSM is operating in, is given by the objective function induced by an individual problem instance or a class of objective functions induced by a class of instances. In the first case (*single-instance learning*), the idea is to optimise the control strategy on one instance during the local search process. The second case (*multi-instance learning*), is based on the assumption that for a given problem domain (or sub-domain), all instances share certain features to which the search control strategy can be adapted.

The modification of the transition probabilities can either be realised by an external mechanism (external adaption control), or within the GLSM framework by means of specialised transition actions (internal adaption control). In both cases, suitable criteria for transition probability updates have to be developed. Two classes of such criterias are those based on trajectory information, and those based on GLSM statistics. The latter category includes state occupancies and transition frequencies, while the former comprises primarily basic descriptive statistics of the objective function value along the search trajectory, possibly in conjunction with discounting of past observations. The approach as outlined here captures only a specific form of parameter learning for a given parameterised class of GLSMs. Conceptually this can be further extended to allow for dynamic changes of transition types (which is equivalent to parameter learning for a more general transition model, such as conditional probabilistic transitions).

Concepts and methods from learning automata theory can be used for analysing and characterising dynamic GLSMs; basic properties, such as expedience or optimality can be easily defined. We conjecture, however, that theoretically proving such properties will be extremely difficult, as the theoretical analysis of standard SLS behaviour is already very complex and limited in its results. Nevertheless, we believe that based on empirical methodology, it should be possible to devise and analyse dynamic GLSMs.

Cooperative GLSM models

Another line of extending the basic GLSM model is to apply several GLSMs simultaneously to the same problem instance. In the simplest case, such an ensemble consists of a number of identical GLSMs and there is no communication between the individual machines. We call this the *homogenous cooperative GLSM model without communication*; its semantics are conceptually equivalent to executing multiple independent tries of an individual GLSM. In Chapter 5 we will use this scheme in the context of efficiently parallelising SLS algorithms for SAT. It is particularly attractive for parallelisation, because it is very easy to implement, involves virtually no communications overhead, and can be almost arbitrarily scaled.

The restrictions of this model can be relaxed in two directions. One is to allow ensembles of different GLSMs. This *heterogeneous cooperative GLSM model without communication* is particularly useful for modelling robust combinations of various SLS algorithms, each of which shows superior performance on certain types of instances, when the features of

the given problem instances are not known *a priori*. This approach has been recently studied in the context of complete algorithms for hard combinatorial problems [GS97a]; in this context the heterogenous ensembles were called *algorithm portfolios*. Generally, this cooperative model has almost the same advantages as its homogeneous variant; it is easy to implement and almost free of communication overhead.

Another generalisation is to allow communication between the individual GLSMs of a cooperative model. This communication can be easily realised by means of transition actions (*e.g.*, `send` and `receive`); possible communication schemes include using a blackboard, synchronous broadcasting, and one-to-one message passing in a fixed network topology. These different variants of *cooperative GLSMs with communication* are more difficult to design and to realise, since issues like preventing and detecting deadlocks and starvation situations generally have to be considered. Furthermore, the communication between individual GLSMs usually involves a certain amount of overhead. This overhead has to be amortised by the performance gains which can be achieved by using this model in a given application situation. These gains may be realised in terms of speedup when applied to a specific problem class, but also in terms of increased robustness w.r.t. different problem types.

Generally, one way of using communication to improve the performance of cooperative GLSMs is to propagate search space positions with low objective function values (or other attractive properties) within the ensemble such that individual GLSMs which detect that they are not doing particularly well can pick up these "hints" and restart their local search from there. This can be easily realised as a homogeneous cooperative GLSM with communication. In such a model, the search effort will be more focussed on exploring promising parts of the search space than in a cooperative model without communication. Another general scheme uses two types of GLSMs, analysts and solvers. Analysts do not attempt to find solutions but rather try to analyse features of the search space. The solvers try to use this information to improve their search strategy. This architecture is an instance of the heterogeneous cooperative GLSM model with communication. It can be extended in a straightforward way to allow for different types of analysts and solvers, or several independent sub-ensembles of analysts and solvers.

Evolutionary GLSM models

From the cooperative GLSM models discussed in the previous section it is only a short step to evolutionary GLSMs. These are cooperative models where the number or type of the individual GLSMs may vary over time; these population dynamics can be interpreted as another form of learning. As for the learning GLSMs described earlier, we can distinguish between *single-instance* and *multi-instance learning* and base the dynamic adaption process on similar criteria. In the conceptually most simple case, the evolutionary process only affects the composition of the cooperative ensemble: machines which are doing well will spawn off numerous offspring replacing individuals showing inferior performance. This mechanism can be applied to both, homogeneous and heterogeneous models for single-instance learning. In the former case, the selection is based on the trajctory information of

the individual machines and achieves a similar effect as described above for the homogeneous cooperative GLSM with communication: The search is concentrated on exploring promising parts of the search space. When applied to heterogeneous models, this scheme allows to realise self-optimising algorithm portfolios, which can be useful for single-instance as well as multi-instance learning.

This scheme can be further extended by introducing mutation, and possibly cross-over operators. It is also possible to combine evolutionary and indivual learning by evolving ensembles of learning GLSMs. And finally, these models can also allow communication within the ensemble. Thus, combining different extensions we arive at very complex and potentially powerful GLSM models; while these are very expressive, in general they will also be extremely difficult to analyse. Nevertheless, their implementation is quite simple and straightforward and an empirical approach for analysing and optimising their behaviour seems viable enough. We expect that such complex models, which allow for a very flexible and fine-grained search control, will be most effective when applied to problem classes which contain a lot of structural features. There is little doubt that, to some extent, this is the case for most real-world problem domains.

Continuous GLSM models

The basic GLSM model and all extensions thereof discussed until here model local search algorithms for solving discrete decision or optimisation problems. But of course, the model can easily be applied to continuous problems; the only changes required are to use continuous local search strategies for the GLSM state types instead of discrete ones. Although our experience and expertise is mainly limited to discrete combinatorial problems, we assume that the GLSM model's main feature, the clear distinction between simple search strategies and search-control, is also a useful architectural and conceptual principle for continuous optimisation algorithms.

3.9 Related Work

The main idea underlying the GLSM model, namely to realise adequate algorithms as a combination of several simple strategies, seems to be common lore. However, our application of this general metaphor to local search algorithms for combinatorial decision and optimisation problems using suitably extended finite state machines for search control, is to our best knowledge novel and original. The GLSM model is partly inspired by Amir Pnueli's work on hybrid systems [MMP92] and Thomas Henzinger's work on hybrid automata; the latter uses finite state machines to model systems with continuous and discrete components and dynamics [ACHH93, Hen96] and is therefore conceptually related to the GLSM model.

The GLSM definition and semantics are heavily based on well-known concepts from automata theory (for a general references, *cf.* [Har78, RS97]). However, when using condi-

tional transitions or transition actions, the GLSM model extends the conventional model of a finite state machine. In its most general form, the GLSM model bears close resemblance to a restricted form of Petri nets [Kri89], where only one token is used. Of course, the same type of search control mechanism could be represented using formal systems other than a FSM-based formalism. First of all, other types of automata, such as pushdown automata or Turing machines could be considered. However, we feel that the FSM model, one of the simplest types of automata, is powerful enough for representing most interesting search-control strategies we found in the local search literature. Furthermore, it is our impression that it leaves enough room for extension, while being analytically more tractable than more complex automata models. Finally, FSMs offer the potential advantage of being implementable in hardware in a rather straightforward way, which might be interesting in the context of applying local search algorithms to time-critical problems (*cf.* [HM97]). Summarising these arguments, the use of FSMs for formalising the search-control mechanism seems to be sufficient and adequate. Of course, formalisms equivalent to the FSM model, such as rule-based descriptions, could be chosen instead. While this could be easily done and might be advantageous in certain contexts (such as reasoning about properties of a given GLSM), we find that the automaton model provides a slightly more intuitive and easier-to-handle framework for designing and implementing local search algorithms, the nature of which is mainly procedural.

The GLSM model allows to adequately represent existing algorithmic frameworks for local search, such as GenSAT [GW93a] or iterated local search [MOF91, Joh90]. These frameworks are generally more specific and more detailed than the GLSM model; however, they can be easily realised as generic GLSMs without losing any detail of description. This is done by using structured generic state types to capture the more specific aspects of the framework to be modeled. The same technique will be used in Chapter 4 to model modern SLS algorithms for SAT, such as WalkSAT or R-Novelty. In these cases, the structure of the simple search strategies can be represented by decision trees which are associated with the state types. While the GLSM model can be used to represent *any* local search algorithm, many of these do not really make use of the search control mechanism it provides. Note, however, that some of the most successful local search algorithms for various problem classes (such as R-Novelty for SAT [MSK97], *hrts* for MaxSAT [BP96], and iterated local search schemes for TSP [MOF91, Joh90, JM97]) rely on search control mechanisms of the type provided by the GLSM model.

The various extensions of the basic model discussed in this chapter are closely related to established work on learning automata [NT89], parallel algorithm architectures [Jáj92], and evolutionary algorithms [Bäc94]. While most of the proposed extensions have not been implemented and empirically evaluated so far, they appear to be promising, especially when taking into account our results on homogeneous cooperative GLSM models reported in Chapter 5 and recent work on multiple independent tries parallelisation [Sho93, GSK98], algorithm portfolios [GS97a], and learning local search approaches for solving hard combinatorial problems [BM98, Min96].

3.10 Conclusions

Based on the intuition that adequate local search algorithms are usually obtained by combining several simple search strategies, in this chapter we introduced and discussed the GLSM model. This framework formalises the search control using a finite state machine (FSM) model, which associates the pure search strategies with the FSM states. FSMs belong to the most basic and yet fruitful concepts in computer science; using them to model local search control offers a number of advantages. First, FSM-based models are conceptually simple; consequently, they can be implemented easily and efficiently. At the same time, the formalism is expressive enough to allow for the adequate representation of a broad range of modern local search algorithms (this will be exemplified in Chapter 4, where GLSM-realisations of several state-of-the-art SLS algorithms for SAT are given). Secondly, there is a huge body of work on FSMs; many results and techniques can be directly applied to GLSMs which is especially interesting in the context of analysing and optimising GLSMs. And finally, in our experience, the GLSM model facilitates the development and design of new, hybrid local search algorithms. In this context, both conceptual and implementational aspects play a role: due to the conceptual simplicity of the GLSM model and its clear representational distinction between search strategies and search control, hybrid combinations of existing local search algorithms can be easily formalised and explored. Using a generic GLSM simulator, which is not difficult to implement, new hybrid GLSM algorithms can be realised and evaluated in a very efficient way.

As we have shown, based on a clean and simple definition of a GLSM, the semantics of the model can be formalised in a rather straightforward way. We then discussed the tight relation between the GLSM model and a standard generic local search scheme. By categorising GLSM types according to their structure and transition types, we demonstrated how the general model facilitates the systematic study of search control mechanisms. Finally, we pointed out several directions into which the basic GLSM model can be extended. Most of these are very natural generalisations, such as continous or cooperative GLSMs; however, these proposed extensions demonstrate the scope of the general idea and suggest numerous routes for further research.

In the context of this work, GLSMs will be used for formalising, realising, and analysing SLS algorithms for SAT. In Chapter 5 we will show how by applying minor modifications to these GLSMs, some of the best known SAT algorithms can be further improved. In Chapter 6, we will use very simple GLSMs as probes for analysing the search space structure of SAT instances. While these applications demonstrate the usefulness of the GLSM model, they hardly exploit its full power. Most modern SLS algorithms can be represented by very simple GLSMs; the fact that the most efficient of them are structurally slightly more complex than others suggests that further improvements can be achieved by developing more complex combinations of simple search strategies. While some of our results presented in Chapter 5 support this hypothesis, its general validity remains to be shown by developing novel hybrid GLSM algorithms for different domains.