

Chapter 8

Local Search Methods

Holger H. Hoos and Edward Tsang

Local search is one of the fundamental paradigms for solving computationally hard combinatorial problems, including the constraint satisfaction problem (CSP). It provides the basis for some of the most successful and versatile methods for solving the large and difficult problem instances encountered in many real-life applications. Despite impressive advances in systematic, complete search algorithms, local search methods in many cases represent the only feasible way for solving these large and complex instances. Local search algorithms are also naturally suited for dealing with the optimisation criteria arising in many practical applications.

The basic idea underlying local search is to start with a randomly or heuristically generated candidate solution of a given problem instance, which may be infeasible, sub-optimal or incomplete, and to iteratively improve this candidate solution by means of typically minor modifications. Different local search methods vary in the way in which improvements are achieved, and in particular, in the way in which situations are handled in which no direct improvement is possible.

Most local search methods use randomisation to ensure that the search process does not stagnate with unsatisfactory candidate solutions and are therefore referred to as *stochastic local search (SLS) methods*. Prominent examples of SLS methods are randomised iterative improvement (also known as stochastic hill-climbing), evolutionary algorithms, simulated annealing, tabu search, dynamic local search and, more recently, ant colony optimisation. These classes of local search algorithms are also widely known as *metaheuristics*.

Many SLS methods are conceptually rather simple and relatively easy to implement compared to many other techniques. At the same time, they often show excellent performance and in many cases define the state-of-the-art in the respective problems.¹ Furthermore, SLS algorithms are often very flexible in that they can be easily adapted to changes in the specification of a problem. This makes them a very popular choice for solving conceptually complex application problems that are sometimes not fully formalised at the

¹Still, the efficient implementation of some high-performance SLS algorithms requires considerable effort and sophisticated data structures (see, e.g., [79]).

beginning of a project. Consequently, SLS algorithms are amongst the most prominent and widely used combinatorial problem solving techniques in academia and industry.

It may be noted that when taking a very broad view of constraint programming, many combinatorial problems, including scheduling, sequencing, configuration and routing problems, can be seen as constraint programming problems. Local search algorithms for these problems are widely studied by researchers from various disciplines, and the corresponding, vast body of literature would be difficult (if not impossible) to survey within a single book chapter.

Therefore, this chapter primarily provides an overview of SLS algorithms for the constraint satisfaction problem (CSP), one of the most prominent problems in constraint programming. We focus on widely known and high-performing algorithms for the general CSP and for SAT, the propositional satisfiability problem, a special case of CSP which plays an important role not only in constraint programming and reasoning research, but also in many other areas of computing science and beyond. We also briefly cover SLS algorithms for constraint optimisation problems such as MAX-CSP and MAX-SAT, the optimisation variants of CSP and SAT, respectively, and point the reader to some of the best-known frameworks and toolkits for implementing local search algorithms for constraint programming problems.

8.1 Introduction

Constraint programming is a powerful conceptual framework that can express many types of combinatorial problems. In this chapter, we mainly focus on the *finite discrete constraint satisfaction problem (CSP)*, a problem of central importance within the area of constraint programming with many applications in artificial intelligence, operations research and other areas of computing science and related disciplines.

The Constraint Satisfaction Problem (CSP)

An instance of the CSP is defined by a set of variables, a set of possible values (or *domain*) for each variable and a set of *constraints* each of which involve one or more of the variables. The *Constraint Satisfaction Problem (CSP)* is to decide for a given CSP instance whether it is possible to assign to each variable a value from its respective domain such that all constraints are simultaneously satisfied. Formally, this can be expressed as follows [52]:

Definition 8.1. A CSP instance is a triple $P = (V, \mathcal{D}, \mathcal{C})$, where $V = \{x_1, \dots, x_n\}$ is a finite set of n variables, \mathcal{D} is a function that maps each variable x_i to the set D_i of possible values it can take (D_i is called the domain of x_i), and $\mathcal{C} = \{C_1, \dots, C_m\}$ is a finite set of constraints. Each constraint C_j is a relation over an ordered set $\text{Var}(C_j)$ of variables from V , i.e., for $\text{Var}(C_j) = (y_1, \dots, y_k)$, $C_j \subseteq \mathcal{D}(y_1) \times \dots \times \mathcal{D}(y_k)$. The elements of the set C_j are referred to as satisfying tuples of C_j , and k is called the arity of the constraint C_j . A CSP instance P is called n -ary, if the arity of all constraints in P have arity at most n ; in particular, binary CSP instances have only constraints of arity at most two.

P is a finite discrete CSP instance if all variables in P have discrete and finite domains. A variable assignment of P is a mapping $a : V \mapsto \bigcup_{i=1}^n D_i$ that assigns to each variable $x \in V$ a value from its domain $\mathcal{D}(x)$. (The assignment of a value to an individual variable is called an atomic assignment.) Let $\text{Assign}(P)$ denote the set of all possible variable

assignments for P ; then a variable assignment $a \in \text{Assign}(P)$ is a solution of P if, and only if, it simultaneously satisfies all constraints in \mathcal{C} , i.e., if for all $C_j \in \mathcal{C}$ with, say, $\text{Var}(C_j) = (y_1, \dots, y_k)$ the assignment a maps y_1, \dots, y_k to values v_1, \dots, v_k such that $(v_1, \dots, v_k) \in C_j$.

CSP instances for which at least one solution exists are called *consistent* (or *soluble*), while instances that do not have any solutions are called *inconsistent* (or *insoluble*).

The finite discrete CSP is the problem of deciding whether a given finite discrete CSP instance P is consistent.

The Propositional Satisfiability Problem (SAT)

The well-known satisfiability problem in propositional logic (SAT) can be seen as a prominent special case of the general CSP. Consider a propositional formula F in conjunctive normal form, i.e., of the form

$$F := \bigwedge_{i=1}^m c_i \quad \text{with} \quad c_i := \bigvee_{j=1}^{k(i)} l_{ij}$$

where each of the l_{ij} is a propositional variable or its negation; the l_{ij} are called *literals*, while the disjunctions c_i are referred to as the *clauses* of F . The objective of the satisfiability problem is then to decide whether F is *satisfiable*, i.e., whether there exists an assignment a of truth values *true* and *false* to the variables x_k such that every clause contains at least one literal rendered true by a . Obviously, this corresponds to a CSP instance where all variables have domains $\{\text{true}, \text{false}\}$ and for every clause c_i there is a constraint C_i between the variables appearing in c_i that is satisfied if, and only if, c_i is satisfied under the (partial) assignment of its variables. Hence, a clause with k literals corresponds to a k -ary constraint relation.

As the prototypical \mathcal{NP} -complete problem, SAT is of central importance to the theory of computing; it also plays an important role in circuit design and verification (see, e.g., Biere et al. [6] or Gu and Puri [37]). Other practical applications of SAT include various scheduling tasks [126, 15] as well as problems from machine vision, robotics, database systems and computer graphics [38]. SAT also plays an important role in the development of algorithms; its conceptual simplicity facilitates the design, implementation and evaluation of new algorithms. Particularly with respect to local search algorithms, many ideas and techniques have been first developed for SAT, before they were generalised to more general types of CSP instances.

SAT-Encodings of CSP

CSP instances can be encoded into SAT in a number of ways (see, e.g., Prestwich [84] or Hoos and Stützle [52] for an overview), and using such encodings, arbitrary CSP instances can be solved by state-of-the-art SAT solvers, including powerful local and systematic search algorithms as well as preprocessing techniques. The main appeal of this approach stems from the previously mentioned advantages of SAT for algorithm development and implementation in combination with the substantial amount of research on SAT solving techniques. SAT encodings may, however, lead to potentially significant increases in the size of problem instances and the respective search spaces; more problematically, they

can obfuscate structural aspects of CSP instances that are important for efficiently solving these.

There is some evidence that the ‘encode and solve as SAT’ approach can work surprisingly well (see, *e.g.*, Kautz and Selman [58], Ernst et al. [24] and Hoos [48]); furthermore, it has been shown that some encodings allow SAT-solvers to directly exploit important aspects of CSP structure [5]. However, it is still unclear whether and to which extent finding good SAT-encodings is any easier than developing good native CSP algorithms — particularly in the case of local search methods, which can often benefit directly from insights and improvements achieved on SAT. It should be noted that the general issue of *modelling* (*i.e.*, finding good formulations of a problem) plays an important role in constraint programming (see Chapter 11).

Local Search

Given a combinatorial problem such as the CSP, the key idea behind local search is very simple: starting at an *initial search position* (for the CSP, typically a randomly chosen, complete variable assignment), in each *step* the search process moves to a position selected from the *local neighbourhood* (typically based on a heuristic evaluation function). This process is iterated until a *termination criterion* is satisfied. To avoid stagnation of the search process, almost all local search algorithms use some form of randomisation, typically in the generation of initial positions and in many cases also in the search steps. This leads to the concept of *stochastic local search (SLS) algorithms*, which is formally defined as follows [52]:

Definition 8.2. *Given a (combinatorial) problem Π , a stochastic local search algorithm for solving an arbitrary problem instance $\pi \in \Pi$ is defined by the following components:*

- *the search space $S(\pi)$ of instance π , which is a finite set of candidate solutions $s \in S$ (also called search positions, locations, configurations, or states) — in the case of the CSP, this is typically the set of all complete variable assignments;*
- *a set of (feasible) solutions $S'(\pi) \subseteq S(\pi)$ — for the CSP, this set typically consists of all solutions of the given CSP instance;*
- *a neighbourhood relation on $S(\pi)$, $N(\pi) \subseteq S(\pi) \times S(\pi)$ — this determines the positions that can be reached in one search step at any given time during the search process;*
- *a finite set of memory states $M(\pi)$, which, in the case of SLS algorithms that do not use memory, may consist of a single state only, and in other cases holds information about the state of the search mechanism beyond the search position (*e.g.*, tabu tenure values in the case of tabu search);*
- *an initialisation function $init(\pi) : \emptyset \mapsto \mathbb{D}(S(\pi) \times M(\pi))$, which specifies a probability distribution over initial search positions and memory states — this function characterises the initialisation of the search process;*
- *a step function $step(\pi) : S(\pi) \times M(\pi) \mapsto \mathbb{D}(S(\pi) \times M(\pi))$ mapping each search position and memory state onto a probability distribution over its neighbouring*

search positions and memory states — this function specifies what happens in every search step;

- a termination predicate $\text{terminate}(\pi) : S(\pi) \times M(\pi) \mapsto \mathbb{D}(\{\text{true}, \text{false}\})$ mapping each search position and memory state to a probability distribution over truth values, which indicates the probability with which the search is to be terminated upon reaching a specific search position and memory state.

In the above, $\mathbb{D}(S)$ denotes the set of probability distributions over a given set S , where formally, a probability distribution $D \in \mathbb{D}(S)$ is a function $D : S \mapsto \mathbb{R}_0^+$ that maps elements of S to their respective probabilities.

Note that this definition includes deterministic local search algorithms as special cases, in which the probability distributions used in the initialisation and step function as well as in the termination criterion are degenerate, with all probability mass concentrated on one value of the underlying domain. As previously noted, completely deterministic local search algorithms are seldom used in research or applications.

In the case of almost all local search algorithms for CSP, the search space consists of all complete variable assignments of the given CSP instance, the solution set is comprised of all satisfying assignments, and the so-called *1-exchange neighbourhood* is used, under which two assignments are direct neighbours if, and only if, they differ at most in the value assigned to one variable. In the special case of SAT, a variant of this neighbourhood relation known as the *1-flip neighbourhood* is typically used, under which two assignments are direct neighbours if, and only if, one can be obtained from the other by flipping the truth value assigned to exactly one of the variable from true to false or vice versa. In many cases the initial search position is determined by generating a variable assignment uniformly at random, and the termination criterion is satisfied if a solution is found or a given bound on the number of search steps has been exceeded.

The various local search algorithms for CSP (and SAT) differ from each other mainly with respect to their step function, which for all but the most simple (and ineffective) algorithms incorporates heuristic guidance in the form of an *evaluation function*. This function typically maps the candidate solutions of the given problem instance π onto a real number such that its global minima correspond to the solutions of π . The evaluation function is used for assessing or ranking the direct neighbours of the current search positions. A commonly used evaluation function for the CSP maps each assignment to the number of constraints violated under it. Note that solutions are characterised by evaluation function value zero, and can hence be easily recognised.

The simplest local search method that effectively uses a given evaluation function g is called *Iterative Improvement* (or *II*; also known as *hill-climbing* or *iterative descent*). In each search step, II selects an *improving position* from the current neighbourhood, i.e., a position $s' \in N(s)$ with $g(s') < g(s)$, where s is the current search position. There are various commonly used heuristics for selecting such an improving neighbour. In *Iterative Best-Improvement*, a neighbour s' with minimal value $g(s')$ within $N(s)$ is chosen; if multiple such neighbours exist, one is chosen uniformly at random. In *Iterative First-Improvement*, on the other hand, the neighbourhood is evaluated in a given order, and the first improving neighbour encountered during this process is selected as the next search position. Variants of Iterative Improvement form the basis for almost all local search methods for CSP, SAT and other combinatorial problems.

```

procedure MCH ( $P$ ,  $maxSteps$ )
  input: CSP instance  $P$ , positive integer  $maxSteps$ 
  output: solution of  $P$  or “no solution found”
   $a$  := randomly chosen assignment of the variables in  $P$ ;
  for  $step := 1$  to  $maxSteps$  do
    if  $a$  satisfies all constraints of  $P$  then return  $a$  end
     $x$  := randomly selected variable from conflict set  $K(a)$ ;
     $v$  := randomly selected value from the domain of  $x$  such that
      setting  $x$  to  $v$  minimises the number of unsatisfied constraints;
     $a := a$  with  $x$  set to  $v$ ;
  end
  return “no solution found”
end MCH

```

Figure 8.1: The basic MCH algorithm; all random selections are according to a uniform probability distribution over the underlying sets.

The Min-Conflicts Heuristic

The simplest and probably most widely known iterative improvement algorithm for the CSP is the *Min Conflicts Heuristic (MCH)* [76, 77]. MCH iteratively modifies the assignment of a single variable in order to minimise the number of violated constraints, which is achieved as follows (see also Figure 8.1): Given a CSP instance P , the search process is initialised by assigning to each variable in P a value that is chosen uniformly at random from its domain. Then, in each local search step, first a CSP variable x is selected uniformly at random from the so-called *conflict set* $K(a)$, the set of all variables that appear in a constraint that is unsatisfied under the current assignment a . A new value v is then chosen from the domain of x , such that by assigning v to x , the number of unsatisfied constraints (conflicts) is minimised. If there are several values of v with that property, one of them is chosen uniformly at random. The search is terminated when a solution is found or a user-specified bound on the number of search steps has been exceeded.

A variant of the basic MCH algorithm that uses a greedy initialisation procedure has been very successfully applied to the n -Queens Problem (a prominent special case of the general CSP, for which specialized polynomial-time algorithms exist) with very large n (say, n equals a million). Furthermore, the efficacy of MCH has been demonstrated in applications to graph colouring and real-world scheduling problems [77].

Like most iterative improvement methods, MCH can get stuck in local minima of the underlying evaluation function; it is therefore *essentially incomplete* [52], *i.e.*, even if run arbitrarily long, the probability for finding a solution to soluble CSP instance may approach a value strictly smaller than one. A simple generic approach for overcoming this problem is to extend MCH with a static restart mechanism that re-initialises the search process every $maxSteps$ search steps, where $maxSteps$ is a user-specified parameter of the algorithm. Unfortunately, the performance of the resulting algorithm depends critically and quite sensitively on good choices of $maxSteps$, which vary substantially between different CSP instances. Substantially more effective variants of the MCH will be discussed later in this chapter.

```

procedure GSAT (F, maxTries, maxSteps)
  input: CNF formula F, positive integers maxTries and maxSteps
  output: model of F or “no solution found”
  for try := 1 to maxTries do
    a := randomly chosen assignment of the variables in formula F;
    for step := 1 to maxSteps do
      if a satisfies F then return a end
      x := randomly selected variable flipping which minimises
        the number of unsatisfied clauses;
      a := a with x flipped;
    end
  end
  return “no solution found”
end GSAT

```

Figure 8.2: The basic GSAT algorithm; all random selections are according to a uniform probability distribution over the underlying sets.

The GSAT Algorithm

Basic GSAT [91] is a simple iterative best-improvement algorithm for SAT that uses the number of clauses unsatisfied under a given assignment as its evaluation function. The algorithm works as follows (see also Figure 8.2): Starting from a complete variable assignment chosen uniformly at random, in each local search step, a single propositional variable is flipped from *true* to *false* or vice versa. The variable to be flipped is chosen such that a maximal decrease in the number of unsatisfied clauses is achieved; if there are several variables with that property, one of them is selected uniformly at random. The iterative best-improvement search used in GSAT gets easily stuck in local minima of the evaluation function. Therefore, GSAT uses a simple static restart mechanism that re-initialises the search at a randomly chosen assignment every *maxSteps* flips. The search is terminated when a model of the given formula *F* has been found, or after *maxTries* sequences (also called ‘tries’) of *maxSteps* variable flips each have been performed without finding a model of *F*.

Straightforward implementations of GSAT are rather inefficient, since in each step the scores of all variables, *i.e.*, the changes in the number of unsatisfied clauses caused by the respective flips, have to be calculated from scratch. The key to efficiently implementing GSAT is to compute the complete set of scores only once at the beginning of each try, and then after each flip to update only the scores of those variables that were possibly affected by the flipped variable (details on this mechanism can be found in Chapter 6 of Hoos and Stützle [52]).

For any fixed setting of the *maxTries* parameter, GSAT is essentially incomplete [44, 47], and severe stagnation behaviour is observed on most SAT instances. Still, when it was first introduced, GSAT outperformed the best systematic search algorithms for SAT available at that time. As one of the first SLS algorithms for SAT, basic GSAT had a very significant impact on the development of a broad range of much more powerful SAT solvers, including most of the current state-of-the-art SLS algorithms for SAT. Further-

more, GSAT and its variants (described later in this chapter) also had a significant impact on the development of high-performance SLS algorithms for the CSP.

8.2 Randomised Iterative Improvement Algorithms

The main limitation of iterative improvement algorithms stems from the fact that they get stuck in local minima of the given evaluation function. A simple approach to deal with this problem is to occasionally allow non-improving search steps, *i.e.*, the selection of neighbours $s' \in N(s)$ with $g(s') \geq g(s)$ from the current neighbourhood. There are numerous different mechanisms that implement this approach; many of these make use of randomised decisions in order to balance the diversification effects of worsening search steps with the search intensification provided by Iterative Improvement.

Randomised Iterative Improvement (RII) is an extension of Iterative Improvement where in each step with a fixed probability wp , a position s' is selected uniformly at random from the current neighbourhood $N(s)$ — this is called a *random walk step*; otherwise (*i.e.*, with probability $1 - wp$), a standard II step is performed. Note that using this mechanism, arbitrarily long sequences of (possibly worsening) random walk steps can be performed. Therefore, as long as the given neighbourhood graph is connected (*i.e.*, any two candidate solutions can be reached from each other by means of a sequence of search steps), RII, when run arbitrarily long, will find a solution to any soluble problem instance with probability approaching one, *i.e.*, $\lim_{t \rightarrow \infty} P_s(RT \leq t) = 1$, where $P_s(RT \leq t)$ is the probability that a solution is found in time at most t .² Algorithms with this property are called *probabilistically approximately complete (PAC)*.

The Min Conflicts Heuristic with Random Walk (WMCH)

By extending the Min Conflicts Heuristic with a simple random walk mechanism, a randomised iterative improvement algorithm called *WMCH* is obtained [116]. In each WMCH step, first a variable x_i is chosen uniformly at random from the conflict set (as in MCH). Then, with probability $wp \geq 0$, a random walk step is performed, *i.e.*, x_i is assigned a value from its domain D_i that has been chosen uniformly at random. In the remaining cases, that is, with probability $1 - wp$, a conflict-minimising value is chosen and assigned, as in a conventional MCH step.

The walk probability wp (also called *noise setting*) has a critical impact on the behaviour of the algorithm. For $wp = 0$, the algorithm is equivalent to the standard Min-Conflicts Heuristic and hence essentially incomplete, but for $wp > 0$, WMCH is provably probabilistically approximately complete (PAC). Intuitively, for low walk probabilities, the search process is likely to have difficulties escaping from local minima regions of the search space, while for very high walk probabilities, its behaviour starts to resemble that of an uninformed random walk, and it will increasingly lack effective heuristic guidance towards solutions. However, for suitably chosen wp settings, WMCH has been empirically observed to perform substantially better than MCH with random restart [101].

²Most local search algorithms for CSP use connected neighbourhoods; however, for more complex constraint programming problems, connected neighbourhoods that can be searched efficiently are sometimes difficult to construct.

Random walk steps in WMCH always involve a variable that appears in a currently unsatisfied constraint; they are therefore also called *conflict-directed random walk steps*. However, different from the GWSAT algorithm [92] (described in the following), which preceded and inspired WMCH, the conflict-directed random walk steps in WMCH do not necessarily render satisfied any previously unsatisfied constraint. WMCH can be varied slightly such that in each random walk step, after choosing a variable x_i involved in a currently violated constraint C , x_i is assigned a value v such that C becomes satisfied; if no such v exists, a value is chosen at random. This variant was found to perform marginally better than the random walk mechanism used in WMCH [101].

GSAT with Random Walk (GWSAT)

The basic GSAT algorithm can be significantly improved by extending it with a random walk mechanism similar to that used in WMCH. Here, in each conflict-directed random walk step, the variable to be flipped is selected uniformly at random from the set of all variables appearing in currently unsatisfied clauses. Note that as a result of any such step, at least one previously unsatisfied clause will become satisfied. This mechanism is closely related to (and in fact inspired by) the conflict-directed random walk algorithm by Papadimitriou, which has been proven to solve 2-SAT in quadratic expected time [80].

GSAT with Random Walk (GWSAT) [92] is variant of basic GSAT that in each local search step probabilistically decides between performing a basic GSAT step and a conflict-directed random walk step (as previously explained). The latter type of step is chosen with a fixed walk probability w_p , and basic GSAT steps are performed otherwise. While for $w_p = 0$, GWSAT is equivalent to basic GSAT, it has been proven to be probabilistically approximately complete (PAC) for any $w_p > 0$ [47].

For suitably chosen walk probability settings (which vary between problem instances, but in many cases can be as high as 0.5), GWSAT achieves substantially better performance than basic GSAT [92]. Furthermore, it has been shown that when using sufficiently high noise settings, GWSAT does not suffer from stagnation behaviour; also, for hard SAT instances, it typically shows exponential run-time distributions (RTDs) [44, 51]. Therefore, static restarts are ineffective, and optimal speedup can be obtained by multiple independent runs parallelisation [52]. For low noise settings, stagnation behaviour is frequently observed; recently, there has been evidence that the corresponding RTDs can be characterised by mixtures of exponential distributions [46].

WalkSAT

The WalkSAT algorithm is conceptually closely related to both GWSAT and MCH; it was first published by Selman, Kautz and Cohen [92], and is now commonly known as *WalkSAT/SKC*. This algorithm was later extended into an algorithmic framework called the *WalkSAT architecture* [73], which includes the original WalkSAT/SKC algorithm as well as several other high-performance SLS algorithms for SAT as special cases (several of these will be covered in later sections of this chapter). Furthermore, variants of WalkSAT have been developed for more general classes of CSP instances as well as for constraint optimisation problems (see Section 8.6).

WalkSAT/SKC (and all other WalkSAT algorithms) are based on a 2-stage variable selection process similar to that used in MCH. In each local search step, first a clause c

is selected uniformly at random from the set of all currently unsatisfied clauses. Then, one of the variables appearing in c is flipped to obtain a new assignment. The choice of this variable is based on a heuristic scoring function $score_b(x)$ that counts the number of currently satisfied clauses that will be broken, *i.e.*, become unsatisfied, by flipping a given variable x . Using this scoring function, the following variable selection scheme is applied: If there is a variable x with $score_b(x) = 0$ in the clause c selected in stage 1, that is, if c can be satisfied without breaking another clause, x is flipped (this is called a *zero damage step*). If more than one such variable exists in c , one of them is selected uniformly at random and flipped. If no such variable exists, with a certain probability $1-p$, the variable with minimal $score_b$ value is selected (*greedy step*; ties are broken uniformly at random); in the remaining cases, that is, with probability p (the so-called *noise setting*), one of the variables from c is selected uniformly at random (*random walk step*).

Note that — like in GWSAT, but unlike in MCH — every step in WalkSAT/SKC is guaranteed to satisfy at least one previously unsatisfied clause (but may at the same time cause many others to become unsatisfied). Otherwise, WalkSAT/SKC uses the same random search initialisation, static random restart mechanism and termination criterion as GSAT.

Although it has been proven that WalkSAT/SKC with fixed *maxTries* parameter has the PAC property when applied to 2-SAT [16], it is not known whether the algorithm is PAC in the general case. In practice, WalkSAT/SKC does not appear to suffer from any stagnation behaviour when using sufficiently high (instance-specific) noise settings, in which case its run-time behaviour is characterised by exponential RTDs [44, 51, 49]. As in the case of GWSAT, stagnation behaviour is frequently observed for low noise settings, and there is some evidence that the corresponding RTDs can be characterised by mixtures of exponential distributions [46].

Typically, when using (somewhat instance-specific) optimised noise settings, WalkSAT/SKC performs substantially better than GWSAT.³ Furthermore, because of its two-stage variable selection scheme, WalkSAT/SKC (like all other WalkSAT algorithms and MCH variants), can be implemented efficiently without using the incremental score update technique essential for the efficient implementation of GSAT and GWSAT.

8.3 Tabu Search and Related Algorithms

The key idea behind *Tabu Search (TS)* [35, 36] is to use memory to prevent the search process from stagnating in local minima or, more generally, attractive non-solution areas of the given search space. In *Simple Tabu Search*, an iterative improvement strategy is enhanced with a short-term memory that allows it to escape from local minima. This memory is used to prevent the search from returning the most recently visited search positions for a fixed number of search steps. Simple TS can be implemented by explicitly memorising previously visited candidate solutions and ruling out any step that would lead back to those. More commonly, reversing recent search steps is prevented by forbidding the re-introduction of solution components (such as assignments of individual CSP variables) which have just been removed from the current candidate solution. A parameter

³Several techniques have been proposed for automatically tuning the noise setting for WalkSAT algorithms (see Patterson and Kautz [83], Hoos [45]).

called *tabu tenure* determines the number of search steps for which these restrictions apply. Note that forbidding possible steps using a tabu mechanism has the same effect as dynamically restricting the neighbourhood $N(s)$ of the current candidate solution s to a subset $N' \subseteq N(s)$ of *admissible neighbours*.

As an undesirable side-effect, this tabu mechanism can sometimes rule out search steps that lead to interesting, unvisited areas of the search space. Therefore, many tabu search algorithms make use of a so-called *aspiration criterion*, which specifies conditions under which the tabu status of candidate solutions or solution components is overridden. One of the most commonly used aspiration criteria overrides the tabu status of steps that lead to an improvement in the *incumbent candidate solution*, *i.e.*, the best candidate solution seen throughout the search process.

Min Conflicts Heuristic with Tabu Search (TMCH)

Extending MCH with a simple tabu search mechanism leads to the *TMCH* algorithm [101, 98]. TMCH works exactly as MCH, except that after each search step, *i.e.*, after the value of a variable x_i is changed from v to v' , the variable/value pair (x_i, v) is declared tabu for the next tt steps, where tt is the *tabu tenure* parameter. While (x_i, v) is tabu, value v is excluded from the selection of values for x_i , unless assigning v to x_i leads to an improvement over the incumbent assignment (aspiration criterion).

According to empirical evidence, TMCH typically performs better than WMCH. Interestingly, a tabu tenure setting of $tt = 2$ was found to consistently result in good performance for CSP instances of different types and sizes [101].

The Tabu Search Algorithm by Galinier and Hao (TS-GH)

Although conceptually quite similar to TMCH, the tabu search algorithm by Galinier and Hao [29], *TS-GH*, typically shows much better performance. TS-GH is based on the same neighbourhood and evaluation function as MCH, but uses a different heuristic for selecting the variable/value pair involved in each search step: Amongst all pairs (x, v') for which variable x appears in a currently violated constraint and v' is any value from the domain of x , TS-GH chooses the one that leads to a maximal decrease in the number of violated constraints. If multiple such pairs exist, one of them is selected uniformly at random. As in MCH, the actual search step is then performed by assigning v' to x . This best-improvement strategy is augmented with the same tabu mechanism used in TMCH: After changing the assignment of x from v to v' , the variable value pair (x, v) is declared tabu for tt search steps. Furthermore, the same aspiration criterion is used to enable the algorithm to perform search steps that lead to improvements over the incumbent assignment regardless of the tabu status of respective variable/value pair.

Unlike for the MCH variants discussed so far, efficient implementations of TS-GH crucially depend on an incremental update mechanism for evaluation function values similar to the one used in GSAT. The basic idea is to maintain the effects of any potential search step on the evaluation function (*i.e.*, the number of conflicts resulting from any search step) in a two-dimensional table of size $n \times k$, where n is the number of variables, and k is the size of the largest domain in the given CSP instance.

Furthermore, the tabu mechanism can be implemented efficiently as follows. For each variable/value pair (x, v) the search step number $t_{x,v}$ when x was last set to v is memorised.

When initialising the search, all the $t_{x,v}$ are set to $-tt$; subsequently, every time a variable x is set to a value v , $t_{x,v}$ is set to the current search step number t , where search steps are counted starting from 0 at the initialisation of the search process. A variable/value pair (x, v) is tabu if, and only if, $t - t_{x,v} \leq tt$. By using this technique in combination with the previously described incremental update mechanism, search steps of TS-GH can be performed as efficiently as those of MCH.

TS-GH was originally introduced as an algorithm for MAX-CSP, the optimisation variant of CSP in which the objective is to find a variable assignment that satisfies a maximal number of constraints (see Section 8.6). Empirical studies suggest that when applied to the conventional CSP, TS-GH generally achieves better performance than any other MCH variant, including TMCH, rendering it one of the best SLS algorithms for the CSP currently known [101]. Unlike in the case of TMCH, the optimal setting of the tabu tenure parameter in TS-GH tends to increase with instance size; this makes it considerably harder to solve new CSP instances with peak efficiency [101].

GSAT with Tabu Search

Integrating a simple tabu search strategy into the best-improvement procedure underlying basic GSAT leads to an algorithm called *GSAT with Tabu Search (GSAT/Tabu)* [72, 98]. In GSAT/Tabu, tabu status is associated with the propositional variables in the given formula. After a variable x has been flipped, it cannot be flipped back within the next tt steps, where the tabu tenure, tt , is a parameter of the algorithm. In each search step, the variable to be flipped is selected as in basic GSAT, except that the choice is restricted to variables that are currently not tabu. Upon search initialisation, the tabu status of all variables is cleared. Otherwise, GSAT/Tabu works exactly as GSAT; in particular, it uses the same restart mechanism and termination criterion. As in the case of TMCH, to implement GSAT/Tabu efficiently, it is crucial to use incremental score updating and tabu mechanisms.

Unlike in the case of GWSAT, it is not clear whether GSAT/Tabu with fixed $maxTries$ parameter has the PAC property. Intuitively, for low tt , the algorithm may not be able to escape from extensive local minima regions, while for high tt settings, all the routes to a solution may be cut off, because too many variables are tabu. However, when using instance-specific, optimised tabu tenure settings, GSAT/Tabu typically performs significantly better than GWSAT with similarly optimised parameters. This is particularly the case for large and structured SAT instances [49]; there are, however, a few exceptional cases where GSAT/Tabu performs substantially worse than GWSAT, including well-known SAT-encoded instances of logistics planning problems. Analogously to basic GSAT, GSAT/Tabu can be extended with a random walk mechanism; limited experimentation suggests that typically this hybrid algorithm does not perform better than GSAT/Tabu [98].

WalkSAT with Tabu Search

Like GSAT and MCH, WalkSAT/SKC can be extended with a simple tabu search mechanism. *WalkSAT/Tabu* [73] uses the same two stage selection mechanism and the same scoring function $score_b$ as WalkSAT/SKC and additionally enforces a tabu tenure of tt steps for each flipped variable. (To implement this tabu mechanism efficiently, the same approach is used as previously described for TS-GH.) If the selected clause c does not allow a zero damage step, of all the variables occurring in c that are not tabu, WalkSAT/Tabu

picks the one with the highest $score_b$ value; when there are several variables with the same maximal score, one of them is selected uniformly at random. In cases where all variables appearing in the selected clause c are tabu, the variable assignment remains unchanged (a so-called *null-flip*), but the current tabu tenure values for all variables decrease exactly as after any other flip.

WalkSAT/Tabu with fixed $maxTries$ parameter has been proven to be essentially incomplete [44, 47]. Although this is mainly caused by null-flips, it is not clear whether replacing null-flips by random walk steps, for instance, would be sufficient for obtaining the PAC property. In practice, however, WalkSAT/Tabu typically performs significantly better than WalkSAT/SKC, especially on structured SAT instances, such as large SAT-encoded blocks world planning problems [49].

Novelty and Variants

The *Novelty* algorithm [73] is derived from the WalkSAT framework. Like tabu search, Novelty uses a limited information on the search history to avoid search stagnation. More specifically, its variable selection mechanism is based on the intuition that repeatedly flipping back and forth the same variable should be avoided. This mechanism is based on the *age of a variable* (see also Gent and Walsh [33]), *i.e.*, the number of flips that have occurred since it was last flipped. Different from WalkSAT/SKC and WalkSAT/Tabu, Novelty and its more recent variants use the same variable scoring function as GSAT, *i.e.*, the difference in the number of unsatisfied clauses caused by the respective flip.

In each step of Novelty, after an unsatisfied clause c has been chosen uniformly at random (exactly as in WalkSAT/SKC), the variable to be flipped is selected as follows. If the variable x with the highest score does not have minimal age among the variables in c , it is always selected. Otherwise, x is only selected with a probability of $1-p$, where p is a parameter called the *noise setting*. In the remaining cases, the variable with the next lower score is selected. When sorting the variables according to their scores, ties are broken according to decreasing age. (If there are several variables with identical score and age, the reference implementation by Kautz and Selman always chooses the one that appears first in c .) Novelty (and the advanced variants described below) use the same initialisation procedure, restart mechanism and termination condition as WalkSAT/SKC.

Note that even for $p > 0$, Novelty is significantly greedier than WalkSAT/SKC, since always one of the two most improving variables from a clause is selected, where WalkSAT/SKC may select any variable if no improvement without breaking other clauses can be achieved. Precisely for this reason, Novelty is provably essentially incomplete for fixed $maxTries$ setting and has been shown to occasionally suffer from extreme stagnation on several commonly used benchmark instances [44, 49]. It may also be noted that, different from WalkSAT/SKC, the Novelty strategy for variable selection within a clause is completely deterministic for both $p = 0$ and $p = 1$. Still, in most cases, Novelty shows significantly improved performance over WalkSAT/SKC and WalkSAT/Tabu [73, 49].

R-*Novelty* [73], a variant of Novelty that uses a more complex variable selection mechanism, performs often, but not always, better than Novelty (for details, see McAllester et al. [73] or Chapter 6 of Hoos and Stützle [52]). Despite its use of a loop-breaking strategy designed to prevent search stagnation, this algorithm suffers from the effects of its provable essential incompleteness [44, 47], but it sometimes performs somewhat better than Novelty [73, 49].

Both Novelty and R-Novelty can be easily extended with a simple conflict-directed random walk mechanism similar to that used in GWSAT; this way, the essential incompleteness as well as the empirically observed stagnation behaviour are effectively overcome. The *Novelty*⁺ algorithm [44, 47] selects the variable to be flipped according to the standard Novelty mechanism with probability $1 - wp$, and makes a uniform random choice from the selected clause in the remaining cases. *R-Novelty*⁺ is obtained from R-Novelty in the same way, but does not make use of R-Novelty's loop-breaking mechanism.

Novelty⁺ is provably PAC for $wp > 0$ and shows exponential RTDs for sufficiently high (instance-specific) settings of the primary noise parameter, p . In practice, small walk probabilities, wp , are generally sufficient to prevent the extreme stagnation behaviour occasionally observed for Novelty and to achieve substantially superior performance compared to Novelty. In fact, the algorithm's behaviour appears to be much more robust w.r.t. the wp parameter than w.r.t. the primary noise setting, p , and uniformly good performance has been observed for $wp = 0.01$ [47]. It may be noted that in cases where Novelty does not suffer from stagnation behaviour, *Novelty*⁺'s performance for $wp = 0.01$ is typically almost identical to Novelty's. Similar observations hold for R-*Novelty*⁺; however, there is some indication that R-*Novelty*⁺ does not reach the performance of the conceptually simpler *Novelty*⁺ algorithm on several classes of structured SAT instances, including SAT-encoded hard graph colouring and planning problems [49].

Adaptive *Novelty*⁺ [45] is an extension of *Novelty*⁺ that dynamically adapts the noise parameter during the search process and hence does not require this parameter to be tuned manually. An efficient implementation of this algorithm won first prize in the random category of the SAT 2004 SAT Solvers Competition. *Novelty*⁺⁺, a more recent variant of Novelty, has been found to perform better than *Novelty*⁺ in many cases; its performance can be further improved by hybridising the underlying variable selection mechanism with a greedy iterative improvement strategy similar to that underlying GSAT [69].

8.4 Penalty-based Local Search Algorithms

An alternative to extending an iterative improvement strategy such that it can escape from local minima of a given evaluation function is to modify the evaluation function when the search process is about to stagnate in a local minimum [71]. This approach is also known as *Dynamic Local Search (DLS)* [52].

Penalty-based algorithms modify the evaluation function by means of *penalty weights*, which are associated with solution components or other features of candidate solutions; in the case of the CSP, penalty weights are usually associated with the constraint relations of the given CSP instance and for SAT, analogously, with the clauses of the given CNF formula (in the latter case, the penalty weights are often referred to as *clause weights*). These penalty weights are modified during the search process. Various penalty-based algorithms differ in their underlying local search strategy and the mechanism used for penalty modification. The latter, in particular, can have a significant impact on the performance of the algorithm.

Penalty-based algorithms have sometimes been motivated by the intuition that by modifying the evaluation function, local minima can be eliminated or, in the case of CSP, the search process can learn to distinguish 'important' from less critical constraints, thus making it easier to find solutions (*i.e.*, global optima). There is, however, increasing evi-

dence that the primary reason for the excellent performance of current penalty-based algorithms lies rather in the effective search diversification caused by the penalty modifications [111, 103]. The idea of diversifying search effort to different parts of the search space as needed in a specific situation has a long history in operations research – see, for example, Koopman [61] and Stone [99].

GENET and the Breakout Method

GENET [119, 107, 20, 17] was one of the earliest penalty-based algorithms in constraint satisfaction. It is based on a neural network design with nodes representing atomic variable assignments and links connecting conflicting atomic assignments. More precisely, a binary CSP instance is represented by a network in which for each variable there is a cluster of *label nodes* that correspond to the values the variable can take. Any pair of label nodes that correspond to variable assignments violating any constraint is connected by a link. A penalty weight is associated with every link in the network; at the beginning of the search process, these weights are all set to one and a random label node in each cluster is switched on.

At any stage of the search, exactly one node per cluster is switched on, that is, every variable has a unique value assigned to it, and the state of the network corresponds to a complete variable assignment. Each label node receives a signal from each of its neighboring nodes that are switched on. The strength of the signal is equal to the weight associated with the connection. For each cluster, the node that receives the least amount of inhibitory signals is switched on. Note that when all penalty weights are one, GENET resembles the Min-Conflicts Heuristic [76, 77].

Motivated by hardware implementations of neural networks, the variable whose cluster is updated in a given search step is chosen asynchronously. Implemented on a sequential machine, clusters are updated sequentially in a random order in each iteration. Whenever the network settles in a stable state, that is, when there is no change of the active node within any cluster that would reduce the total weight of edges between active nodes, the weight of all edges between active nodes are increased by one. As a result, the network may become unstable again.

The ‘energy’ of a network state (which is returned by the evaluation function) is the total amount of input received by all the nodes that are switched on in that state [18]. The stable states of the network correspond to the local minima of this evaluation function, and GENET reaches these by performing iterative improvement steps. If the energy is 0, then a solution to the CSP has been found.

GENET was extended to non-binary constraint satisfaction problems by using hyper-edges as links in the network [119, 20, 67, 68]. Stuckey and Tam [100] used such an extension of GENET to mutate chromosomes in an evolutionary algorithm. The resulting memetic algorithm was demonstrated to be effective in solving hard CSP instances. Variants of GENET have also been used to solve challenging instances of a car sequencing problem [19].

The Breakout Method [78] is another early penalty-based algorithm for the CSP. Unlike GENET, it associates a single penalty weight with each constraint of the given CSP instance and uses an evaluation function that maps each variable assignment a to the total weighted of the constraints violated under a . Otherwise, the two algorithms are basically identical. In particular, like GENET, the Breakout Method initialises all penalty weights

to one and uses iterative improvement until a local minima of its evaluation function is reached, at which point the weights of all unsatisfied constraints are incremented by one before the search is continued.

Guided Local Search (GLS)

Unlike GENET or the Breakout Method, which were designed rather specifically for constraint satisfaction problems, *Guided Local Search (GLS)* [111] is a more general penalty-based method that has been used for combinatorial decision and optimisation problems (such as SAT and TSP, respectively) [113].

As a penalty-based method, GLS associates penalties with the constraints of the given CSP instance. GLS uses an augmented evaluation function of the form

$$g'(a) = g(a) + \lambda \sum_{i=1}^m p_i I_i(a), \quad (8.1)$$

where a is a complete variable assignment, $g(a)$ is the evaluation function value of a (here: the number of constraints unsatisfied under a), p_i is the penalty of constraint i , and $I_i(a)$ is an indicator function with value 1 if constraint i is violated under a and 0 otherwise.

All penalties are initialised to 0 at the beginning of the search, and penalty changes are applied whenever the search process reaches a local minimum of f . The penalties to be increased in a given local minimum are selected such that they maximise the *utility function*

$$util_i(a) = I_i(a) \cdot c(i) / (1 + p_i) \quad (8.2)$$

where a , $I_i(a)$, $g(a)$ and p_i are defined as in Eq. 8.1, and $c(i)$ is the cost of having constraint i unsatisfied. This cost is set to one for all constraints in a standard CSP, but by using different cost values, GLS can be easily extended to optimisation variants of the CSP with weighted constraints. This selection mechanism ensures that only penalties of currently violated constraints are increased. Secondly, the more a constraint has been penalised, the less incentive there is for penalising it again; this facilitates diversification of the search. Each penalty selected is increased by one at a time. Finally, when non-uniform constraint costs are used, this strategy keeps the search focused on satisfying higher-cost constraints. This carefully designed penalty update mechanism has been proven to be useful in various applications, including BT's scheduling problem [106] and a version of the *Radio-Link Frequency Assignment Problem* [112], an abstracted military communications problem originating from the CALMA project [8].

One of the attractive properties of GLS is that it has only one major parameter, namely λ , to tune. One good heuristic is to set the value of λ to a fraction (between 0 and 1) of the cost of the first local minimum encountered by GLS. This allows λ to be selected according to the characteristics of the given problem instance. At the time of publication, GLS was shown to be competitive with other high-performance algorithms on a widely studied set of 11 benchmark problems. More recently, GLS has been extended to incorporate random moves and aspiration [74]. The resulting algorithm, *Extended GLS*, was shown to be at least as effective as GLS, but significantly less sensitive to the value of the λ parameter for the problems it was tested on (which included SAT, Weighted MAX-SAT and the Quadratic Assignment Problem).

GSAT with Clause Weights

This early penalty-based algorithm for SAT was motivated by the observation that when performing multiple runs of basic GSAT on some types of structured SAT instances, certain clauses tend to be unsatisfied at the end of each run. The idea behind *GSAT with Clause Weights* [90] is to bias the search process towards satisfying such ‘problem clauses’ by associating weights with them. More precisely, weights are associated with each clause. These are initially set to one; but before each restart, the weights of all currently unsatisfied clauses are increased by $\delta = 1$. The underlying local search procedure is a variant of basic GSAT that uses a modified evaluation function $g'(F, a)$ which measures the total weight of all clauses in the given formula F that are unsatisfied under assignment a . Search initialisation, restart and termination are as in basic GSAT. (A variant called ‘Averaging In’ uses a modified search initialisation that introduces a bias towards the best candidate solutions reached in previous local search phases [90].)

GSAT with Clause Weights was found to perform substantially better than basic GSAT on various classes of structured SAT instances, including SAT-encoded graph colouring problems; furthermore, there is some indication that by using the same clause weighting mechanism with GWSAT, further performance improvements can be achieved [90]. To date, both of these algorithms are outperformed by WalkSAT algorithms such as Novelty⁺ and by state-of-the-art penalty-based algorithms, such as SAPS (which is covered later in this section) and PAWS [102]. Several variants of GSAT with Clause Weights have been studied by Cha and Iwama [12]. Some of these use slight variations of the weight update scheme and a simple form of tabu search. However, from their limited empirical results it is doubtful that any of these variations achieves significant performance improvements over the original GSAT with Clause Weights algorithm.

Several variants of GSAT with Clause Weights that perform weight updates after each local search step have been proposed and studied by Frank [26, 27]. These are based on the idea that GSAT should benefit from discovering which clauses are most difficult to satisfy relative to recent assignments. The most basic of these variants, called *WGSAT*, uses the same weight initialisation and update procedure as GSAT with Clause Weights, but performs only a single GSAT step before updating the clause weights. A modification of this algorithm, called *UGSAT*, restricts the neighbourhood considered in each search step to the set of variables appearing in currently unsatisfied clauses [26]. (This is the same neighbourhood as used in the random walk steps of GWSAT.) While this leads to considerable speedups for naïve implementations of the underlying local search procedure, the difference for efficient implementations is likely to be insufficient to render UGSAT competitive with other GSAT variants, such as GWSAT.

Frank also studied a variant of WGSAT in which the clause weights are subject to a uniform decay over time [27]. The underlying idea is that the relative importance of clauses w.r.t. their satisfaction status can change during the search, and hence a mechanism is needed that focuses the weighted search on the most recently unsatisfied clauses. Although using this decay mechanism slightly improves the performance of WGSAT when measured in terms of variable flips, this gain is insufficient to amortise the added time complexity of the frequent weight update steps. However, similar mechanisms for focusing the search on recently unsatisfied clauses play a crucial role in state-of-the-art penalty-based algorithms for SAT that are covered later in this section.

The Discrete Lagrangian Method (DLM)

The use of penalties in dynamic local search is conceptually closely related to the use of Lagrange multipliers for solving continuous constrained optimisation problems [93, 115]. For a constrained optimisation problem in which a function $f(\vec{x})$ is to be minimised subject to equality constraints $g_i(\vec{x}) = 0$, we can define the Lagrangian function

$$L(\vec{x}, \vec{\lambda}) = f(\vec{x}) + \sum_i \lambda_i g_i(\vec{x}) \quad (8.3)$$

where the λ_i are continuous variables called *Lagrange multipliers*. Note that these play the same role as the penalty weights in the augmented evaluation function typically used in the previously discussed penalty-based algorithms. It can be shown that a local minimum satisfying all equality constraints can be obtained by finding a saddle point of L , *i.e.*, a point $(\vec{x}^*, \vec{\lambda}^*)$ such that

$$L(\vec{x}^*, \vec{\lambda}) \leq L(\vec{x}^*, \vec{\lambda}^*) \leq L(\vec{x}, \vec{\lambda}^*) \quad (8.4)$$

for all $(\vec{x}^*, \vec{\lambda})$ and $(\vec{x}, \vec{\lambda}^*)$ sufficiently close to $(\vec{x}^*, \vec{\lambda}^*)$. Based on this result, the problem of finding a local minimum of a constrained optimisation problem can be reduced to the problem of finding a saddle point of an unconstrained optimisation problem. This latter task can be achieved by performing iterative improvement (*e.g.*, in the form of gradient descent) on L using the variables \vec{x} in combination with iterative ascent on L using the Lagrange multipliers $\vec{\lambda}$. In a local minimum \vec{x} of f that does not satisfy all constraints, increasing the Lagrange multipliers has the effect of more heavily penalising violated constraints. Eventually, for some value of $\vec{\lambda}$, $L(\vec{x}, \vec{\lambda})$ is no longer a local minimum, such that further minimisation by modifying \vec{x} becomes possible, resulting in fewer violated constraints.

This well-known approach for solving continuous constrained optimisation problems provided the motivation for Shang and Wah's DLM algorithm for SAT [93]. The basic idea behind this dynamic local search algorithm is to perform iterative best improvement on the same augmented evaluation function used in GSAT with clause weights (this corresponds to the minimisation of $L(\vec{x}, \vec{\lambda})$ over \vec{x}). Whenever a local minimum is reached, the penalties for all unsatisfied clauses are increased (this corresponds to the ascent on $L(\vec{x}, \vec{\lambda})$ by modifying $\vec{\lambda}$), until some previously worsening variable flip becomes improving, and hence the search process is no longer stuck in a local minimum. The basic version of DLM for SAT also uses a tabu mechanism equivalent to that found in GSAT/Tabu, as well as periodic decreases of all clause penalties to avoid numerical overflow. Furthermore, before the search process is started, the given formula is simplified by performing a complete pass of unit propagation.

Several extensions of the basic DLM algorithm have been shown to achieve improved performance [122, 121]; these use various memory-based mechanisms for avoiding and overcoming search stagnation more effectively (for an overview of these methods, see Hoos and Stützle [52].) All of these algorithms have a relatively large number of parameters that need to be tuned carefully in order to achieve peak performance. DLM has also been applied to weighted MAX-SAT problems [115], while extensions to non-binary problems represent an interesting research direction.

It should be noted that despite the close conceptual relationship between the approaches, important mathematical properties of Lagrangian methods for continuous optimisation do

not carry over to DLM. This is primarily due to the heuristic mechanisms used by DLM for determining search steps, as opposed to the rigorous use of derivatives of the objective function in continuous Lagrangian methods.

ESG and SAPS

The *Exponentiated Subgradient (ESG) algorithm* [89] was originally motivated by sub-gradient optimisation, a well-known method for minimising Lagrangian functions that is widely used for generating lower bounds for branch-and-bound algorithms. As a penalty-based algorithm for SAT, ESG associates penalty weights with the clauses of the given CNF formula that are modified during the search process. The search is started from a randomly selected variable assignment after initialising all clause weights to one. The local search procedure underlying ESG for SAT is based on a best improvement search method that can be seen as a simple variant of GSAT; in each local search step, the variable to be flipped is selected uniformly at random from the set of all variables that appear in currently unsatisfied clauses and whose flipping leads to a maximal decrease in the total weight of unsatisfied clauses. When reaching a local minimum (*i.e.*, an assignment in which flipping any variable that appears in an unsatisfied clause would not lead to a decrease in the total weight of unsatisfied clauses), with probability η , the search is continued by flipping a variable that is uniformly chosen at random from the set of all variables appearing in unsatisfied clauses; otherwise, the local search phase is terminated.

After each local search phase, the clause weights are updated in two stages: First, the weights of all clauses are multiplied by a factor that depends on the respective satisfaction status (scaling stage): weights of satisfied clauses are multiplied by α_{sat} , weights of unsatisfied clauses by α_{unsat} . Then, all clause weights are updated using the formula $clw(c) := clw(c) \cdot \rho + (1 - \rho) \cdot \bar{w}$ (smoothing stage), where \bar{w} is the average of all clause weights after scaling, and the parameter ρ has a fixed value between zero and one. The algorithm terminates when a satisfying assignment for F has been found or when a given bound on the number of search steps has been reached.

Compared to the underlying local search steps, a weight update is computationally expensive, since it involves modifications of all clause weights. Additionally, experimental evidence indicates that local search phases in ESG are typically quite short, and therefore the expensive smoothing operations have to be performed rather frequently [52, 53]. Even with the use of special implementation techniques that help ameliorate this problem, Southey and Schuurmans' highly optimised reference implementation of ESG for SAT does not always reach the performance of high-performance WalkSAT algorithms such as Novelty⁺. Compared to DLM-2000-SAT, ESG-SAT typically requires fewer steps for finding a model of a given formula, but in terms of CPU-time, both algorithms show very similar performance [89, 53]. It may be noted that the general ESG framework has been originally proposed for the more general Boolean linear programming (BLP) problem, and it has also been applied quite successfully to combinatorial auctions winner determination problems [89].

The *Scaling and Probabilistic Smoothing (SAPS) algorithm* by Hutter et al. [53] is based on the insight that the expensive weight update scheme in ESG can be replaced by a much more efficient procedure without negative impact on the underlying search procedure. SAPS can be seen as a variant of ESG that uses a modified weight update scheme, in which the scaling stage is restricted to the weights of currently unsatisfied clauses, and

smoothing is only performed with a certain probability p_{smooth} . The first of these modifications is also used in Southey and Schuurmans' efficient ESG implementation; but it is the probabilistic, and hence less frequent, smoothing that results in a substantial performance improvement over ESG and also renders superfluous the special implementation tricks that are crucial for achieving good performance in ESG. SAPS was shown to perform substantially better than ESG, DLM-2000-SAT and high-performance WalkSAT variants [53]; however, there are some types of SAT instances (in particular, hard and large SAT encoded graph colouring instances), for which SAPS does not reach the performance of Novelty⁺.

A reactive variant of SAPS, *RSAPS* [53], automatically adjusts the smoothing probability p_{smooth} during the search, using a mechanism that is very similar to the one underlying Adaptive WalkSAT [45]. *RSAPS* sometimes achieves significantly better performance than SAPS; however, it still has other parameters, in particular, the scaling factor α_{unsat} , that need to be manually optimised.

8.5 Other Approaches

Besides the algorithms covered in the previous sections, many other local search methods have been applied in the context of solving CSPs. Within the confines of this chapter it is impossible to present a complete survey of the large and ever-increasing number of local search algorithms for the CSP and closely related problems, such as the Graph Colouring Problem and SAT. Therefore, the algorithms mentioned in the following were selected to illustrate some of the major approaches.

There is a large body of work on evolutionary algorithms for constraint satisfaction problems. Some of the earliest work include Tsang and Warwick [108], Paredis [82], Hao and Dorne [39], Warwick and Tsang [120] and Riff Rojas [87]; Craenen et al. [14] provides an overview and comparison of more recent evolutionary algorithms. GENET and GLS have been used as subsidiary search procedures in memetic algorithms for constraint satisfaction [100] and optimisation [43]. Galinier and Hao [30] have developed a specialised memetic algorithm for the Graph Colouring Problem (GCP) that uses short runs of an effective tabu search algorithm as its subsidiary search procedure; this algorithm is one of the most effective GCP algorithms currently known.

Hao and Dorne [39] used a specialised genetic algorithm to search the space of partial assignments. Lau [64] developed the Guided Genetic Algorithm (GGA), which applies the principle of Guided Local Search in a genetic algorithm. The idea is to use penalties to construct a fitness template, which guides crossover and mutation in a genetic algorithm such that better assignments will be chosen in the selection process with higher probability. GGA has been applied successfully to the Processor Configuration Problem [65], the General Assignment Problem in scheduling [64], and to a version of the Radio-Link Frequency Assignment Problem [66].

Constraints are used to help evolutionary algorithms search efficiently. This is done by modifying the objective functions in evolutionary computation. For example, Yu et al. [125] used penalties to guide the search away from 'poor' areas of the search space, whereas Li [70], Tsang and Li [105], and Jin [54] used incentives to guide the search towards promising areas.

Ant colony optimisation (ACO), a population-based stochastic local search method inspired by the path-finding behaviour of ants [22], has been applied with some success to the CSP [96], and in particular, to permutation constraint satisfaction problems, such as car sequencing [95], and to binary CSPs [110]. Other widely used stochastic local search methods have been applied to specific types of CSP instances. For example, there are various *simulated annealing algorithms* for the graph colouring problem (GCP) [55] and SAT [97]. Likewise, several *iterated local search algorithms* have been developed for the GCP [13, 81] and MAX-SAT [123, 94]. A generalisation of GSAT to CSP that also includes various additional SLS mechanisms, including random walk and clause penalties, was developed by Kask and Dechter [56] and later extended with a tree search mechanism based on cycle-cutsets [57]. Walser [117] has introduced a WalkSAT algorithm for Pseudo-Boolean CSP (a well-known special case of CSP), which includes a tabu mechanism as well as biased random search initialisation.

Local search does not have to be incomplete. In *Systematic Local Search* [40] and related approaches (e.g., Richards et al. [86]), completeness is achieved through the recording and resolution of *no-goods* whenever the underlying local search algorithm encounters a local minimum. When a no-good is encountered, resolution is attempted: for example, if both “ $P=\text{true}$ and $Q=\text{true}$ ” and “ $P=\text{true}$ and $Q=\text{false}$ ” have been encountered, then they are replaced by “ $P=\text{true}$ ” (a technique often used in *truth maintenance systems*; see, e.g., Doyle [23]). These no-goods help Systematic Local search to escape from local optima and to achieve completeness, a desirable property which most other local search methods do not enjoy: When both “ $P=\text{true}$ ” and “ $P=\text{false}$ ” are found to be no-goods for any P , the given CSP instance has been shown to be unsatisfiable. To achieve completeness, Systematic Local Search may record an exponential number of no-goods in the worst case. However, with careful memory management, the algorithm has been demonstrated to be effective for job shop scheduling problems [21].

Constrained Local Search [85] is an example for an approach that searches over partial assignments that do not violate any constraints. Based on *dynamic backtracking* [34], Constrained Local Search conducts a depth-first search. Whenever a partial assignment cannot be further extended, a randomly chosen atomic assignment is removed from it, such that the search can be continued in a different direction. Despite its use of depth-first search, Constrained Local Search is incomplete.

Most local search algorithms for CSP use neighbourhood relations that restrict search steps to modifying the value of only one variable at a time. However, the use of larger neighbourhoods can sometimes be advantageous; for example, the swap neighbourhood, in which search steps swap the values of two variables, has been used successfully on sequencing problems in conjunction with GENET [19]. Large neighbourhoods are more commonly used in SLS algorithms for constraint optimisation problems (see next section).

8.6 Local Search for Constraint Optimisation Problems

Many real-life problems are over-constrained. For example, in a production planning application, there may be insufficient resources to complete all given jobs within their respective deadlines. In this situation, it may be desirable to find a feasible assignment of resources such that the total amount of revenue generated is maximised; this type of optimisation problem is referred to as a *maximal utility problem* [104]. In other cases, some constraints

may be violated, but doing so incurs a penalty cost. The objective is then to find a solution with minimal penalty; this is known as the *minimal violation problem* [104].

These types of problems can be modelled by extending constraint satisfaction problems to include optimisation objectives. In the simplest case, the problem is represented as a CSP instance, but the objective becomes to find a variable assignment that satisfies a maximal number of constraints (*MAX-CSP*). Note that this is equivalent to finding a variable assignment that minimises the total number of violated constraints. In many cases, not all constraints are equally important. In *Weighted MAX-CSP*, this is captured by weights associated with the individual constraints, and the objective is to maximise the total weight of the satisfied constraints. More general formalisations of constraint optimisation problems include Partial CSP [28], Semi-Ring Based CSP [7] and Valued CSP [88].

A widely studied special case of MAX-CSP and Weighted MAX-CSP is the optimisation variant of SAT, *MAX-SAT*: Given a propositional formula F in conjunctive normal form, the objective in MAX-SAT is to find an assignment of truth values to the variables in F such that a maximum number of clauses in F is satisfied. In *Weighted MAX-SAT*, each clause has an associated weight, and the goal is to find an assignment that maximises the total weight of the satisfied clauses. MAX-SAT and Weighted MAX-SAT are of particular interest in algorithm development because of their conceptual simplicity in combination with the fact that any Weighted MAX-CSP instance can be transformed into a Weighted MAX-SAT instance (at the price of losing structures of the constraint graph and searching a somewhat larger space).

Local search methods are naturally suited for solving constraint optimisation problems [42]. In particular, most local search algorithms for the CSP can be directly applied to MAX-CSP, since their evaluation function directly corresponds to the optimisation objective of minimising the number of violated constraints. Moreover, these algorithms can be extended to Weighted MAX-CSP by modifying the standard evaluation function (number of constraints violated under a given assignment) such that it maps each variable assignment to the total weight of the constraints violated under it (see, *e.g.*, Lau [63]). In special cases, different evaluation functions may be useful; for example, Walser's WalkSAT algorithm for Overconstrained Pseudo-Boolean CSP with hard and soft constraints uses an evaluation function that takes into account the degree of violation of the given linear pseudo-Boolean constraint relations [118].

It is worth noting that when generalising dynamic local search methods to Weighted MAX-CSP, there is no single 'correct' way to integrate the constraint weights and the penalty values into the augmented evaluation function. Perhaps the most obvious approach is to simply add the weights and penalties over all violated constraints (see, *e.g.*, Wah and Shang [115]). An alternate solution was found to work better in GLS, where constraint weights are used for determining the penalty values to be increased after each local search phase, but do not appear in the augmented evaluation function (see Section 8.4). A similar approach is taken in Wu and Wah's DLM algorithm for Weighted MAX-SAT [122], where the clause weights are used for penalty initialisation and update, but not in the evaluation function.

Larger neighbourhoods, which allow more than one variable to be changed in a single local search step, have been more extensively studied in the context of local search for constraint optimisation than in the case of CSP. For example, Yagiura and Ibaraki [123] have developed various types of SLS algorithms for MAX-SAT based on 2- and 3-flip neighbourhoods. Large neighbourhoods have also been used successfully in vari-

ous application-relevant combinatorial optimisation problems (see, *e.g.*, Yao [124], Tsang and Voudouris [106], Ahuja et al. [2], Abdullah et al. [1]). In all of these cases, special techniques have to be used in order to search these large neighbourhoods efficiently.

Local search algorithms play a major role in solving real-life constraint optimisation problems, because in many cases, they are able to find high-quality solutions more efficiently than other approaches. For example, GLS has been incorporated into ILOG's Dispatcher system (ILOG is the market leader in commercial constraint programming software) [3, 60]. Dispatcher was specifically designed for vehicle routing, a prominent problem in Operations Research which is of central importance in the transportation business (see Chapter 23). Generally, local search algorithms can often be very usefully applied in combination with other methods. For example, branch-and-bound algorithms can benefit significantly from high-quality bounds obtained by high-performance local search methods.

8.7 Frameworks and Toolkits for Local Search

Both the development of local search algorithm for solving constraint satisfaction and optimisation problems and their practical application are often greatly facilitated by software frameworks and programming toolkits. This is particularly the case when dealing with conceptually complex constraint programming problems. Such systems can substantially ease the burden associated with achieving efficient implementations of SLS algorithms. They also facilitate software reuse and support the separation of problem formulation (modelling) and solving. In the following, we give a brief overview of some of the better known frameworks and toolkits that support SLS algorithms; while some of these are general combinatorial optimisation or constraint programming systems, others are specifically focused on local search methods.

ILOG Solver is a commercial system which provides users with a C++ library that implements state-of-the-art algorithms for constraint satisfaction and optimisation. The OPL interface to ILOG Solver supports a rich declarative syntax that can be used to define the structure of problems and heuristics [41]. *ILOG Dispatcher* is a specialised package for vehicle routing that supports a variety of local search algorithms.

The commercial *iOpt* system implements a wide range of SLS methods. Through a graphic interface, *iOpt* allows users to experiment with different local search strategies and to construct hybrid algorithms. It also provides an abstract class library in Java that can be used to implement local search methods [114]. Similarly, the freely available object-oriented frameworks *EasyLocal++* [31] and *HotFrame* [25] support the design and implementation of local search algorithms in C++. In these general optimisation systems, problem-independent parts of the algorithms are captured in the form of abstract classes, which are specialised by the user to implement problem-specific algorithms.

The *COMET* programming language supports both modeling and search abstractions in constraint programming. It allows users to specify and control local search algorithms using constraints, modelling and search abstractions, and it has been applied to a wide range of combinatorial problems [42]. The conceptually related *SALSA* language facilitates the concise, declarative definition of local, systematic and hybrid search algorithms [62]. Finally, the freely available *ZDC* system aims to help non-experts in constraint programming by providing them with a simple declarative language (EaCL) and a graphic

user interface. It implements a number of local search algorithms, including Guided Local Search [10, 109].

Regardless of whether local search algorithms are realised within such a framework or environment or implemented ‘from scratch’, it is very important for the reproducibility of empirical results to ensure that their published descriptions are accurate and complete (covering also all performance-critical implementation details). Furthermore, whenever possible, reference implementations should be made available to the research community.⁴

8.8 Conclusions and Outlook

Among the various approaches for solving constraint programming problems such as the CSP, local search methods are of considerable interest to researchers and practitioners. Although most local search algorithms are incomplete, in many cases, their performance scales better with instance size than that of complete, systematic search algorithms. Consequently, high-performance local search methods are often the only practical tool for solving large and difficult real-world problems, which often involve thousands of variable with large domains. This is especially true for decision problems where the main objective is to find feasible solutions quickly and for optimisation problems where high-quality or (near-)optimal solutions need to be obtained as efficiently as possible.

Local search methods have been shown to be very successful in solving many important classes of problems, including SAT, MAX-SAT, travelling salesman and quadratic assignment problems. Their effectiveness and efficiency has also been demonstrated for many real-world problems, including scheduling, vehicle routing and radio-frequency assignment tasks. In many of these applications, local search algorithms achieve comparable or superior performance compared to all other methods.

Although efficient local search algorithms typically incorporate problem-specific knowledge (often in the form of the neighbourhood relation and evaluation function), there are general, high-level strategies that have been shown to be effective across a broad range of combinatorial problems. Most of these general local search strategies involve randomisation to avoid search stagnation in or around local minima of the given evaluation function and are therefore captured in the general framework of Stochastic Local Search (SLS). SLS methods such as randomised iterative improvement, tabu search and dynamic local search have provided the basis for some of the most prominent and best performing algorithms for CSP and SAT. Other methods, including simulated annealing, evolutionary algorithms, ant colony optimisation and iterated local search have also been applied to these and many other constraint programming problems, and were shown to be effective for solving certain types of instances. These search strategies employ different mechanisms for balancing the exploration of the given search space (*diversification*) against the efficient exploitation of heuristic information (*intensification*). Intensification and diversification mechanism often interact in complex ways, and minor variations can have significant impact on the performance of the resulting algorithms.

For this reason, in combination with the fact that theoretical results in this area are difficult to obtain and typically very limited in their practical relevance, SLS algorithms are mostly studied empirically, by means of computational experiments. (It may be noted that a similar situation is encountered for most, if not all, other high-performance CSP

⁴The same applies, of course, to any other constraint programming algorithm that is evaluated empirically.

algorithms.) In the case of SLS algorithms for CSP, many empirical studies have been focused on distributions of relatively unstructured, random binary CSP instances. The same holds for MAX-CSP, and the situations for prominent special cases, such as SAT and MAX-SAT, is similar. While such instances can be useful for evaluating the efficacy of search strategies, they lack the type of structure found in many real-world problems. Consequently, there is an increasing emphasis on using structured problem instances for the empirical analysis of SLS algorithms for the CSP and related problems. This endeavour, as well as the comparability of empirical results between studies, is facilitated by public collections of benchmark problems, such as CSPLIB [32] and SATLIB [50].

Furthermore, while currently the design of new SLS algorithms largely resembles a craft in that it requires experience and intuition to a significant extent, there is substantial interest in developing more principled approaches that will facilitate the engineering of high-performance SLS algorithms. In this context, advanced empirical methods (see, *e.g.*, Chapter 4 of Hoos and Stützle [52]) in combination with frameworks that specifically support the formulation and implementation of local search algorithms (see Section 8.7) are likely to play a major role. Furthermore, our understanding of the factors causing the relative hardness of certain problem instances for a given SLS algorithm is fairly limited. The investigation of these factors, for example, by means of search space analysis, is an active research area with many open problems.

Another attractive research direction is to develop SLS algorithms that adapt their behaviour based on information collected during the search process or over runs on various problem instances. Interesting work in this area includes studies by Battiti and Tecchiolli [4], Glover [35] and Minton [75], as well as Boyan and Moore [9], Patterson and Kautz [83], Hoos [45], Mills [74], Hutter et al. [53], Burke and Newall [11] and [59].

It may be noted that in many ways, the development and understanding of SLS algorithms is significantly further advanced for SAT than for the general CSP. (The situation for MAX-SAT and MAX-CSP is analogous.) This is mostly caused by the fact that as a conceptually simpler problem, SAT for CNF formulae better facilitates the development, analysis and efficient implementation of SLS algorithms. This raises the question to which extent more efficient SLS algorithms for the general CSP can be obtained by augmenting suitably generalised high-performance SLS algorithms for SAT with specific methods for handling certain types of complex constraints known from other constraint programming approaches. Furthermore, it is likely that advanced SLS methods that have been demonstrated to be very successful in solving other combinatorial problems, such as iterated local search, variable depth search or scatter search, may still hold considerable and largely unexplored potential for solving constraint satisfaction and optimisation problems.

Overall, local search methods are among the most powerful and versatile tools for solving constraint programming problems. They give rise to a broad range of interesting research challenges, and continuing efforts to improve these methods and our understanding of them will further enhance their usefulness in a broad range of challenging real-world applications.

Bibliography

- [1] S. Abdullah, S. Ahmadi, E. Burke, and M. Dror. Applying Ahuja-Orlin's large neighbourhood for constructing examination timetabling solution. In *5th Interna-*

- tional Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 413–419, 2004.
- [2] R. Ahuja, O. Ergun, J. Orlin, and P. Punnen. A survey of very large scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.
 - [3] B. D. Backer, V. Furnon, P. Kilby, P. Prosser, and P. Shaw. Solving vehicle routing problems using constraint programming and meta heuristics. *Journal of Heuristics*, 6(4):501–525, 2000.
 - [4] R. Battiti and G. Tecchioli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
 - [5] C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in SAT. In *Theory and Applications of Satisfiability Testing, 6th International Conference (SAT 2003), Selected Revised Papers*, LNCS 2919, pages 299–314. Springer Verlag, 2004.
 - [6] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS '99)*, LNCS 1579, pages 193–207. Springer Verlag, Berlin, Germany, 1999.
 - [7] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
 - [8] A. Bouju, J. Boyce, C. Dimitropoulos, G. vom Scheidt, and J. Taylor. Intelligent search for the radio link frequency assignment problem. In *International Conference on Digital Signal Processing*, Cyprus, 1995.
 - [9] J. Boyan and A. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.
 - [10] R. Bradwell, P. M. J. Ford, E. Tsang, and R. Williams. An overview of the CACP project: modelling and solving constraint satisfaction/optimisation problems with minimal expert intervention. In *CP 2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers*, 2000.
 - [11] E. Burke and J. Newall. A new adaptive heuristic framework for examination timetabling problems. *Annals of Operations Research*, 129:107–134, 2004.
 - [12] B. Cha and K. Iwama. Performance test of local search algorithms using new types of random CNF formulas. In *14th International Joint Conference on Artificial Intelligence*, pages 304–310. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1995.
 - [13] M. Chiarandini and T. Stützle. An application of iterated local search to the graph coloring problem. In *Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, Ithaca, New York, USA, 2002.
 - [14] B. Craenen, A. Eiben, and J. van Hemert. Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 7(5):424–445, 2003.
 - [15] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *12th National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1092–1097, Seattle, Washington, USA, 1994. AAAI Press/MIT Press. ISBN 0-262-51078-2.
 - [16] J. Culberson, I. Gent, and H. Hoos. On the probabilistic approximate completeness of WalkSAT for 2-SAT. Technical Report APES-15a-2000, APES Research Group, 2000.
 - [17] A. Davenport. A comparison of complete and incomplete algorithms in the easy and hard regions. In *Workshop on Studying and Solving Really Hard Problems, 1st*

- International Conference on Principles and Practice of Constraint Programming*, pages 43–51, September 1995.
- [18] A. Davenport. *Extensions and evaluation of GENET in constraint satisfaction*. PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, 1997.
 - [19] A. Davenport and E. Tsang. Solving constraint satisfaction sequencing problems by iterative repair. In *1st International Conference on the Practical Application of Constraint Technologies and Logic Programming (PACLP)*, pages 345–357, London, April 1999.
 - [20] A. Davenport, E. Tsang, C. Wang, and K. Zhu. GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *12th National Conference for Artificial Intelligence*, pages 325–330, 1994.
 - [21] B. Dilkina, L. Duan, and W. Havens. Extending systematic local search for job shop scheduling problems. In *11th International Conference on Principles and Practice of Constraint Programming (CP 2005)*, LNCS 3709, pages 762–766. Springer-Verlag, 2005.
 - [22] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA, 2004.
 - [23] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.
 - [24] M. D. Ernst, T. D. Millstein, and D. S. Weld. Automatic SAT-compilation of planning problems. In *15th International Joint Conference on Artificial Intelligence*, pages 1169–1177. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.
 - [25] A. Fink and S. Voß. Hotframe: A heuristic optimization framework. In *Optimization Software Class Libraries*, pages 81–154. Kluwer, 2002.
 - [26] J. Frank. Weighting for Godot: Learning heuristics for GSAT. In *13th National Conference on Artificial Intelligence*, pages 776–783. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1996.
 - [27] J. Frank. Learning short-term clause weights for GSAT. In *15th International Joint Conference on Artificial Intelligence*, pages 384–389. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.
 - [28] E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence, Special Volume on Constraint Based Reasoning*, 58(1-3):21–70, 1992.
 - [29] P. Galinier and J.-K. Hao. Tabu search for maximal constraint satisfaction problems. In *Principles and Practice of Constraint Programming – CP 1997*, LNCS 1330, pages 196–208. Springer Verlag, Berlin, Germany, 1997.
 - [30] P. Galinier and J. K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
 - [31] L. D. Gaspero and A. Schaerf. Easylocal++: an object-oriented framework for the flexible design of local-search algorithms. *Software Practice and Experience*, 33(8):733–765, 2003.
 - [32] I. Gent and T. Walsh. CSPLib: a benchmark library for constraints. Technical report, APES-09-1999, 1999.
 - [33] I. P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *10th National Conference on Artificial Intelligence*, pages 28–33. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1993.
 - [34] M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1: 25–46, 1993.
 - [35] F. Glover. Tabu search and adaptive memory programming — advances, applica-

- tions and challenges. In *Interfaces in Computer Science and Operations Research*. Kluwer Academic Publishers, 1996.
- [36] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, USA, 1997.
- [37] J. Gu and R. Puri. Asynchronous Circuit Synthesis with Boolean Satisfiability. *IEEE Transactions of Computer-Aided Design of Integrated Circuits and Systems*, 14(8): 961–973, 1995.
- [38] J. Gu, P. Purdom, J. Franco, and B. Wah. Algorithms for the satisfiability (SAT) problem: A survey. In *Satisfiability problem: Theory and Applications*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 19–151. American Mathematical Society, Providence, RI, USA, 1997.
- [39] J.-K. Hao and R. Dorne. A new population-based method for satisfiability problems. In *11th European Conference on Artificial Intelligence*, pages 135–139, Amsterdam, 1994. John Wiley & Sons.
- [40] W. Havens and B. Dilkina. A hybrid schema for systematic local search. In *Advances in Artificial Intelligence: 17th Conference of the Canadian Society for Computational Studies of Intelligence*, LNCS 3060, pages 248–260. Springer Verlag, 2004.
- [41] P. V. Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, USA, 1999.
- [42] P. V. Hentenryck and L. Michel. *Constraint-Based Local Search*. MIT Press, Cambridge, MA, USA, 2005.
- [43] D. Holstein and P. Moscato. Memetic algorithms using guided local search, a case study. In *New ideas in optimization*, pages 235–243. McGraw Hill, 1999.
- [44] H. Hoos. *Stochastic Local Search — Methods, Models, Applications*. PhD thesis, TU Darmstadt, FB Informatik, Darmstadt, Germany, 1998.
- [45] H. Hoos. An adaptive noise mechanism for WalkSAT. In *18th National Conference on Artificial Intelligence*, pages 655–660. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2002.
- [46] H. Hoos. A mixture-model for the behaviour of SLS algorithms for SAT. In *18th National Conference on Artificial Intelligence*, pages 661–667. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2002.
- [47] H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *16th National Conference on Artificial Intelligence*, pages 661–666. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1999.
- [48] H. Hoos. SAT-encodings, search space structure, and local search performance. In *16th International Joint Conference on Artificial Intelligence*, pages 296–302. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1999.
- [49] H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.
- [50] H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In *SAT 2000*, volume 63, pages 283–292. IOS Press, Amsterdam, The Netherlands, 2000.
- [51] H. Hoos and T. Stützle. Characterising the behaviour of stochastic local search. *Artificial Intelligence*, 112(1–2):213–232, 1999.
- [52] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier / Morgan Kaufmann, 2004.
- [53] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing:

- Efficient dynamic local search for SAT. In *Principles and Practice of Constraint Programming – CP 2002*, LNCS 2470, pages 233–248. Springer Verlag, Berlin, Germany, 2002.
- [54] N. Jin. Equilibrium selection by co-evolution for bargaining problems under incomplete information about time preferences. In *Congress on Evolutionary Computation (CEC 2005)*, pages 2661–2668, Edinburgh, September 2005.
- [55] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation: Part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.
- [56] K. Kask and R. Dechter. GSAT and local consistency. In *14th International Joint Conference on Artificial Intelligence*, pages 616–623. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1995.
- [57] K. Kask and R. Dechter. A graph-based method for improving GSAT. In *13th National Conference on Artificial Intelligence*, pages 350–355. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1996.
- [58] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *13th National Conference on Artificial Intelligence*, volume 2, pages 1194–1201. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1996.
- [59] M. Kern. *Parameter Adaptation in heuristic search - a population-based approach*. PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, 2005.
- [60] P. Kilby, P. Prosser, and P. Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints*, 5(4):389–414, 2000.
- [61] B. Koopman. The theory of search, part iii, the optimum distribution of searching effort. *Operations Research*, 5:613–626, 1957.
- [62] F. Laburthe and Y. Caseau. SALSALSA: A language for search algorithms. In *4th International Conference on Principles and Practice of Constraint Programming*, LNCS 1520, pages 310–324. Springer Verlag, 1998.
- [63] H. C. Lau. A new approach for weighted constraint satisfaction. *Constraints*, 7(2): 151–165, 2002.
- [64] T. Lau. *Guided Genetic Algorithm*. PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, 1999.
- [65] T. Lau and E. Tsang. Solving the processor configuration problem with a mutation-based genetic algorithm. *International Journal on Artificial Intelligence Tools (IJAIT)*, 6(4):567–585, December 1997.
- [66] T. Lau and E. Tsang. Guided genetic algorithm and its application to radio link frequency assignment problems. *Constraints*, 6(4):373–398, 2001.
- [67] J. Lee, H. Leung, and H. Won. Extending GENET for non-binary CSPs. In *17th International Conference on Tools with Artificial Intelligence*, pages 338–342, 1995.
- [68] J. Lee, H. Leung, and H. Won. Towards a more efficient stochastic constraint solver. In *2nd International Conference on Principles and Practice of Constraint Programming*, pages 338–352, Cambridge, Massachusetts, USA, August 1996.
- [69] C. M. Li and W. Huang. Diversification and determinism in local search for satisfiability. In *8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, LNCS 3569, pages 158–172. Springer Verlag, 2005.
- [70] J. Li. *FGP: A genetic programming based tool for financial forecasting*. PhD thesis,

- Department of Computer Science, University of Essex, Colchester, UK, 2001.
- [71] D. Luenberger, editor. *Linear and nonlinear programming*. Addison-Wesley Publishing Co., Inc., 1984.
 - [72] B. Mazure, L. Sais, and E. Gregoire. TWSAT: A new local search algorithm for SAT – performance and analysis. In *14th National Conference on Artificial Intelligence*, pages 281–285. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1997.
 - [73] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *14th National Conference on Artificial Intelligence*, pages 321–326. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1997.
 - [74] P. Mills. *Extensions to guided local search*. PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, 2002.
 - [75] S. Minton. Automatically configuring constraint satisfaction programs, a case study. *Constraints*, 1(1&2):7–43, 1996.
 - [76] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *8th National Conference on Artificial Intelligence*, pages 17–24. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1990.
 - [77] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1–3):161–205, 1992.
 - [78] P. Morris. The breakout method for escaping from local minima. In *National Conference on Artificial Intelligence*, pages 40–45, 1993.
 - [79] D. Neto. *Efficient Cluster Compensation for Lin-Kernighan Heuristics*. PhD thesis, University of Toronto, Department of Computer Science, Toronto, Canada, 1999.
 - [80] C. H. Papadimitriou. On selecting a satisfying truth assignment. In *32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 163–169. IEEE Computer Society Press, Los Alamitos, CA, USA, 1991.
 - [81] L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In *Applications of Evolutionary Computing*, LNCS 2279, pages 122–131. Springer Verlag, Berlin, Germany, 2002.
 - [82] J. Paredis. Genetic state-space search for constrained optimization problems. In *13th International Joint Conference on Artificial Intelligence*, pages 967–972, 1993.
 - [83] D. J. Patterson and H. Kautz. Auto-walksat: A self-tuning implementation of walksat. In *LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*. Elsevier, Amsterdam, The Netherlands, 2001.
 - [84] S. Prestwich. Local search on SAT-encoded CSPs. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, pages 388–399, 2003.
 - [85] S. Prestwich. Stochastic local search in constrained spaces. In *Practical Applications of Constraint Technology and Logic Programming (PACLP '00)*, pages 27–39, 2000.
 - [86] T. Richards, Y. Jiang, and B. Richards. Ng-backmarking – an algorithm for constraint satisfaction. *British Telecom Technology Journal*, 13(1):102–109, 1995.
 - [87] M. Riff Rojas. From quasi-solutions to solution: an evolutionary algorithm to solve CSPs. In *2nd International Conference on Principles and Practice of Constraint Programming*, pages 367–381, August 1996.
 - [88] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems:

- Hard and easy problems. In *14th International Joint Conference on Artificial Intelligence*, pages 631–639. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1995.
- [89] D. Schuurmans, F. Southey, and R. C. Holte. The exponentiated subgradient algorithm for heuristic Boolean programming. In *17th International Joint Conference on Artificial Intelligence*, pages 334–341. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001.
- [90] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *13th International Joint Conference on Artificial Intelligence*, pages 290–295. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.
- [91] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *10th National Conference on Artificial Intelligence*, pages 440–446. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1992.
- [92] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *12th National Conference on Artificial Intelligence*, pages 337–343. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1994.
- [93] Y. Shang and B. W. Wah. A discrete Lagrangian-based global-search method for solving satisfiability problems. *Journal of Global Optimization*, 12(1):61–99, 1998.
- [94] K. Smyth, H. H. Hoos, and T. Stützle. Iterated robust tabu search for MAX-SAT. In *Advances in Artificial Intelligence, 16th Conference of the Canadian Society for Computational Studies of Intelligence*, LNCS 2671, pages 129–144. Springer Verlag, Berlin, Germany, 2003.
- [95] C. Solnon. Solving permutation constraint satisfaction problems with artificial ants. In *14th European Conference on Artificial Intelligence*, pages 118–122, Berlin, Germany, August 2000.
- [96] C. Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4):347–357, 2001.
- [97] W. M. Spears. Simulated annealing for hard satisfiability problems. Technical report, Naval Research Laboratory, Washington D.C., USA, 1993.
- [98] O. Steinmann, A. Strohmaier, and T. Stützle. Tabu search vs. random walk. In *KI-97: Advances in Artificial Intelligence*, LNAI 1303, pages 337–348. Springer Verlag, Berlin, Germany, 1997.
- [99] L. D. Stone. The process of search planning: current approaches and continuing problems. *Operations Research*, 31:207–233, 1983.
- [100] P. Stuckey and V. Tam. Improving evolutionary algorithms for efficient constraint satisfaction. *International Journal on Artificial Intelligence Tools (IJAIT), World Scientific*, 8(4):363–383, 1999.
- [101] T. Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. PhD thesis, TU Darmstadt, FB Informatik, Darmstadt, Germany, 1998.
- [102] J. Thornton, D. Pham, S. Bain, and V. Ferreira. Additive versus multiplicative clause weighting for SAT. In *19th National Conference on Artificial Intelligence*, pages 191–196. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2004.
- [103] D. Tompkins and H. Hoos. Warped landscapes and random acts of SAT solving. In *8th International Symposium on Artificial Intelligence and Mathematics*, 2004.
- [104] E. Tsang. *Foundations of constraint satisfaction*. Academic Press, London and San

- Diego, 1993.
- [105] E. Tsang and J. Li. EDDIE for financial forecasting. In *Genetic Algorithms and Programming in Computational Finance*, pages 161–174. Kluwer Academic Publishers, 2002.
 - [106] E. Tsang and C. Voudouris. Fast local search and guided local search and their application to British Telecom’s workforce scheduling problem. *Operations Research Letters*, 20(3):119–127, 1997.
 - [107] E. Tsang and C. Wang. A generic neural network approach for constraint satisfaction problems. In *Neural network applications*, pages 12–22. Springer-Verlag, 1992.
 - [108] E. Tsang and T. Warwick. Applying genetic algorithms to constraint satisfaction problems. In *9th European Conference on AI*, pages 649–654, 1990.
 - [109] E. Tsang, J. Ford, P. Mills, R. Bradwell, R. Williams, and P. Scott. Towards a practical engineering tool for rostering. *Annals of Operational Research, Special Issue on Personnel Scheduling and Planning*, 2006 (to appear).
 - [110] J. van Hemert and C. Solnon. A study into ant colony optimization, evolutionary computation and constraint programming on binary constraint satisfaction problems. In *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004)*, LNCS 3004, pages 114–123. Springer Verlag, Berlin, Germany, 2004.
 - [111] C. Voudouris and E. Tsang. Partial constraint satisfaction problems and guided local search. In *Practical Application of Constraint Technology (PACT’96)*, pages 337–356, London, April 1996.
 - [112] C. Voudouris and E. Tsang. Solving the radio link frequency assignment problem using guided local search. In *13th NATO symposium on Frequency Assignment, Sharing and Conservation Systems (Aerospace), Research and Technology Organization (RTO)*. North Atlantic Treaty Organization (NATO), 1999.
 - [113] C. Voudouris and E. Tsang. Guided local search, chapter 7. In *Handbook of meta-heuristics*. Kluwer, 2003.
 - [114] C. Voudouris, R. Dorne, D. Lesaint, and A. Liret. iopt: A software toolkit for heuristic search methods. In *7th International Conference on Principles and Practice of Constraint Programming*, pages 716–729. Springer Verlag, 2001.
 - [115] B. Wah and Y. Shang. Discrete Lagrangian-based search for solving MAX-SAT problems. In *15th International Joint Conference on Artificial Intelligence*, pages 378–383, 1997.
 - [116] R. J. Wallace and E. C. Freuder. Heuristic methods for over-constrained constraint satisfaction problems. In *Over-Constrained Systems*, LNCS 1106, pages 207–216. Springer Verlag, Berlin, Germany, 1995.
 - [117] J. P. Walser. Solving linear pseudo-Boolean constraint problems with local search. In *14th National Conference on Artificial Intelligence*, pages 269–274. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1997.
 - [118] J. P. Walser. *Integer Optimization by Local Search: A Domain-Independent Approach*. LNCS 1637. Springer Verlag, Berlin, Germany, 1999.
 - [119] C. Wang and E. Tsang. Solving constraint satisfaction problems using neural-networks. In *IEE 2nd International Conference on Artificial Neural Networks*, pages 295–299, 1991.
 - [120] T. Warwick and E. Tsang. Using a genetic algorithm to tackle the processors configuration problem. In *ACM Symposium on Applied Computing (SAC)*, pages 217–221, 1994.

- [121] Z. Wu and B. W. Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *17th National Conference on Artificial Intelligence*, pages 310–315. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2000.
- [122] Z. Wu and B. W. Wah. Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *16th National Conference on Artificial Intelligence*, pages 673–678. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1999.
- [123] M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation. *Journal of Heuristics*, 7(5):423–442, 2001.
- [124] X. Yao. Dynamic neighbourhood size in simulated annealing. In *International Joint Conference on Neural Networks (IJCNN'92)*, volume 1, pages 411–416. IEEE Press, Piscataway, NJ, USA, 1992.
- [125] X. Yu, W. Zheng, B. Wu, and X. Yao. A novel penalty function approach to constrained optimization problems with genetic algorithms. *Journal of Advanced Computational Intelligence*, 2(6):208–213, 1998.
- [126] H. Zhang. Generating college conference basketball schedules by a SAT solver. In *5th International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*, pages 281–291, 2002.