7

MAX-SAT and MAX-CSP

MAX-SAT and MAX-CSP are the optimisation variants of SAT and CSP. These problems are theoretically and interesting, because they are among the conceptually simplest combinatorial optimisation problems, yet instances of optimisation problems from many application domains can be represented as MAX-SAT or MAX-CSP in an easy and natural way. SLS algorithms are amongst the most powerful and successful methods for solving large and hard MAX-SAT and MAX-CSP instances.

In this chapter, we first introduce MAX-SAT. Next, we present some of the most powerful SLS algorithms for solving various types of MAX-SAT instances and give an overview of results on their behaviour and relative performance. In the second part of this chapter, we introduce MAX-CSP and discuss SLS methods for solving the general problem as well as the closely related overconstrained pseudo-Boolean and Integer Optimisation problems.

7.1 The MAX-SAT Problem

MAX-SAT can be seen as a generalisation of SAT for propositional formulae in conjunctive normal form in which, instead of satisfying all clauses of a given CNF formula F with $n$variables and m clauses (and hence F as a whole), the objective is to satisfy as many clauses of F as possible. A solution to an instance of this problem is a variable assignment, *i.e.*, a mapping

of variables in F to truth values, that satisfies a maximal number of clauses in F.

Definition 7.1 ((Unweighted) MAX-SAT)

Given a CNF formula $F = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} l_{ij}$ let $f(F, a)$ be the number of clauses in F that are unsatisfied under variable assignment a. The *(Unweighted) Maximum Satisfiability Problem (MAX-SAT)* is to find $a^* \in argmin\{f(F, a) \mid a \in Assign(F)\} = argmax\{m - f(F, a) \mid a \in Assign(F)\}$, *i.e.*, a variable assignment a^* that maximises the number of the satisfied clauses in F. □

Remark: Maximising the number of satisfied clauses in F is equivalent to minimising the number of unsatisfied clauses. Although MAX-SAT is intuitively defined as a maximisation problem, it is often formally more convenient to consider the equivalent minimisation problem; this is the reason for using the objective function $f(F, a)$, whose value is to be minimised, in our definition of MAX-SAT. In the following we will consider MAX-SAT as a minimisation problem.

This definition captures the search variant of MAX-SAT; the evaluation variant and associated decision problems can be defined in a similar way. Given a CNF formula F, in the evaluation variant, the objective is to determine the minimum number of clauses unsatisfied under any assignment; The associated decision problem for a given solution quality bound b is to determine whether there is an assignment that leaves at most b clauses in F unsatisfied. Note that SAT is equivalent to the decision variant of unweighted MAX-SAT with solution quality bound $b = 0$, *i.e.*, to deciding whether for a given CNF formula F an assignment a exists such that the number of clauses in F unsatisfied under a is equal to zero.

Example 7.1: A Simple MAX-SAT Instance

Let us consider the following propositional formula in CNF:

$$\begin{aligned} F = \quad & (\neg x_1) \\ & \wedge (\neg x_2 \vee x_1) \\ & \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (x_1 \vee x_2) \\ & \wedge (\neg x_4 \vee x_3) \\ & \wedge (\neg x_5 \vee x_3) \end{aligned}$$

The minimum number of clauses in F that are unsatisified under any assignment, $f(F, a^*)$, is one; two of the (many) assignments that achieve this optimal solution quality are $x_1 = x_2 = x_3 = x_4 = x_5 = \bot$ and $x_1 = \bot, x_2 = x_3 = x_4 = x_5 = \top$.

It may be noted that while the SAT problem is defined for arbitrary propositional formulae, the definition of MAX-SAT is restricted to formulae in CNF. Furthermore, different from SAT, MAX-SAT is not invariant under certain logical equivalence transformations, *i.e.*, there exist MAX-SAT instances whose underlying CNF formulae are logically equivalent but whose solutions are different. In particular, the solutions of a MAX-SAT instance can change when introducing multiple copies of clauses in the given CNF formula; in unweighted MAX-SAT the number of copies of a clause can be used to express its importance relative to other clauses. As a consequence, standard simplification techniques for SAT are not applicable to MAX-SAT, including unit propagation and pure literal reduction.

Weighted MAX-SAT

In many applications of MAX-SAT, the constraints represented by the CNF clauses are not all equally important. These differences can be represented explicitly and compactly using weights associated with each clause of a CNF formula.

Definition 7.2 (Weighted CNF Formulae)

A *weighted CNF formula* is a pair (F, w) where F is a CNF formula $F = \bigwedge_{i=1}^{m} c_i$ with $c_i = \bigvee_{j=1}^{k_i} l_{ij}$, and $w : \{c_i \mid i \in$

$[1 \ldots m]\} \mapsto \mathbb{R}^+$ is a function that assigns a positive real value to each clause of F; $w(c_i)$ is called the *weight* of clause c_i. □

Intuitively, the clause weights in a weighted CNF formula reflect the relative importance of satisfying them; in particular, appropriately chosen clause weights can indicate the fact that satisfying a certain clause can be more important than satisfying several other clauses. Weighted MAX-SAT is a straightforward generalisation of unweighted MAX-SAT in which the objective is to minimise the total weight of the unsatisfied clauses rather than just their number.

Definition 7.3 (Weighted MAX-SAT)

Given a clause weighted CNF formula $F' = (F, w)$, let $f(F', a)$ be the total weight of the clauses of F unsatisfied under assignment a, *i.e.*, $f(F', a) = \sum_{i=1}^{m}\{w(c_i) \mid c_i$ is a clause of F that is unsatisfied under $a\}$. The *Weighted Maximum Satisfiability Problem (Weighted MAX-SAT)* is to find a variable assignment a^* that maximises the total weight of the satisfied clauses in F, *i.e.*, $a^* \in argmin\{f(F', a) \mid a \in Assign(F)\} = argmax\{\bar{f} - f(F', a) \mid a \in Assign(F)\}$, where $\bar{f} = \sum\{w(c_i) \mid c_i$ is a clause of $F\}$. □

Although the definition allows for real-valued clause weights, it is easy to show that integer clause weights are sufficient for expressing arbitrary relative importance relations between clauses. Primarily for historically motivated efficiency reasons, many implementations of MAX-SAT algorithms support only integer clause weights. (Many older types of microprocessors performed integer operations substantially faster than floating point operations; this is not the case for modern CPUs.) However, because in most programming languages the range of integer data types is very limited compared to floating point data types, such implementations can sometimes not handle certain types of MAX-SAT instances.

Many combinatorial optimisation problems contain logical conditions that have to be satisfied for any feasible solution; these conditions are often called *hard constraints*, while constraints whose violation does not preclude

feasibility are referred to as *soft constraints*. When representing such problems as weighted MAX-SAT instances, the hard constraints can be captured by choosing the weights of the corresponding CNF clauses high enough that no combination of soft constraint clauses can outweigh a single hard constraint clause. The decision problem with solution quality bound b associated with such a weighted MAX-SAT instance, where b is lower than the weight of a single hard constraint clause, but at least as high as the combined weight of any set of soft constraint clauses, then accurately represents the given problem; in particular, any solution to such a weighted MAX-SAT instance corresponds to a feasible solution of the underlying combinatorial optimisation problem.

Example 7.2: A Simple Weighted MAX-SAT Instance

Consider the formula F from Example 7.1 with the following clause weights:

$$\begin{array}{lcll} w(c_1) & = & w(\neg x_1) & = & 2 \\ w(c_2) & = & w((\neg x_2 \vee x_1) & = & 1 \\ w(c_3) & = & w(\neg x_1 \vee \neg x_2 \vee \neg x_3) & = & 7 \\ w(c_4) & = & w(x_1 \vee x_2) & = & 3 \\ w(c_5) & = & w(\neg x_4 \vee x_3) & = & 7 \\ w(c_6) & = & w(\neg x_5 \vee x_3) & = & 7 \end{array}$$

The total weight of the clauses unsatisfied under assignment $x_1 = \bot, x_2 = x_3 = x_4 = x_5 = \top$ is 1, which is the optimal solution quality for this weighted MAX-SAT instance (F, w).

Furthermore, when considering this weighted MAX-SAT instance with solution quality bound 6, clauses $c_3, c_5,$ and c_6 can be seen as hard constraints while all other clauses represent soft constraints. The assignment $x_1 = x_2 = x_3 = x_4 = x_5 = \bot$, which was optimal for the unweighted MAX-SAT instance F, has objective function value 7 and is hence not a feasible solution in this context.

Complexity and Approximability Results

MAX-SAT (unweighted as well as weighted) is an $\mathcal{NP}$-hard optimisation problem, since SAT can be reduced to MAX-SAT in a straight-forward way. Interestingly, while 2-SAT, the restriction of SAT to CNF formulae with clauses of length 2, can be solved in polynomial time, MAX-2-SAT, the corresponding restriction of MAX-SAT, is known to be $\mathcal{NP}$-hard, as is MAX-3-SAT, *i.e.*, MAX-SAT for CNF formulae with clause length 3.

However, there are polynomial-time algorithms for MAX-SAT that are guaranteed to find solutions within a certain range of the optimum for arbitrary MAX-SAT instances. The first such approximation algorithm is a relatively simple greedy construction method that was proposed and shown to solve any weighted MAX-SAT instance within a factor (approximation ratio) of at most 2 from the respective maximum total weight of the clauses satisfied under any variable assignment [Johnson, 1974]. (More recently, it has been shown that Johnson's algorithm guarantees an approximation ratio of 1.5 [**?**].)

Since 1994, a series of polynomial-time algorithms with substantially improved approximation ratios has been introduced [**?**; Goemans and Williamson, 1994; 1995; Feige and Goemans, 1995; **?**; **?**; Asano and Williamson, 2000]; the most recent of these guarantees an approximation ratio of 1.275 [Asano and Williamson, 2000]. (Assuming the correctness of a conjecture by Uri Zwick [**?**], that is supported by numerical evidence, this latter result can be improved to 1.201 [Asano and Williamson, 2000].) For the special cases MAX-3-SAT and MAX-2-SAT, the best approximation algorithms guarantee solutions within 8/7=1.1429 [**?**] and 1.075 [Feige and Goemans, 1995; **?**], respectively. It is interesting to note that a simple iterative improvement algorithm with a non-oblivious evaluation function (see Section 7.2) has been proven to achieve a worst-case approximation ratio of $2^k/(2^k - 1)$ for MAX-k-SAT [Khanna *et al.*, 1994b].

There are limitations on the theoretical performance guarantees that can be obtained from polynomial-time algorithms for MAX-SAT: If $\mathcal{P} \neq \mathcal{NP}$, there exists no polynomial-time approximation algorithm for MAX-3-SAT, and hence for MAX-SAT, with a (worst-case) approximation ratio lower than 8/7=1.1429; for MAX-2-SAT, an analogous result holds rules out approximation ratios lower than 1.0472 [Hastad, 1997; 2001]. Arbitrarily improved approximation ratios α can be obtained at the cost of run-times that

are exponential in the size of MAX-SAT instance and depend on the desired value of α [Dantsin *et al.*, 1998]. It is worth noting that approximation algorithms for MAX-SAT such as the ones mentioned above can be empirically shown to achieve much better solution qualities for many types of MAX-SAT instances; however, their performance is usually substantially inferior to that of state-of-the-art SLS algorithms for MAX-SAT (see, *e.g.*, [Hansen and Jaumard, 1990]).

Randomly Generated MAX-SAT Instances

As in the case of SAT, various classes of randomly generated problem instances play a prominent role in the empirical analysis of the performance and the behaviour of MAX-SAT algorithms. Uniform Random-3-SAT instances have been used in many studies; typically, the respective test-sets are sampled from overconstrained distributions, *i.e.*, the clauses per variable ratio is larger than the critical value of approximately 4.3 and the instances are unsatisfiable with very high probability (see also Chapter **??**, page **??**).

A number of empirical studies have used test-sets obtained from the random clause length model, in which each of the possible $2n$ literals over n variables is included with a fixed probability in any clause (see Chapter **??**, page **??**). A well-known set of such instances is part of the DIMACS collection of SAT benchmark instances; these `jnh` instances have 100 clauses and between 800 and 900 clauses each, including satisfiable and unsatisfiable instances. Sets of weighted MAX-SAT instances have been derived from test-sets of random clause length formula, including the `jnh` instances, by determining for each clause an integer weight between 1 and $1,000$ uniformly at random [Resende *et al.*, 1997; Yagiura and Ibaraki, 1998; 2001]. Particularly the weighted `jnh` (`wjnh`) instances have been widely used for evaluating the performance of MAX-SAT algorithms.

A range of test-sets of weighted MAX-SAT has been introduced by Hoos *et al.*; these consist of Uniform Random-3-SAT instances with truncated discretised Gaussian clause weight distributions [Hoos *et al.*, in preparation]. The weight distributions are characterised by three parameters μ, σ', and δ, where μ is the mean, σ' the standard deviation (before truncation), and δ the granularity of the underlying Gaussian probability distribution, which is symmetrically truncated to the interval $[1, 2\mu - 1]$ (see Figure **??**); the granularity specifies the minimum difference between non-identical clause

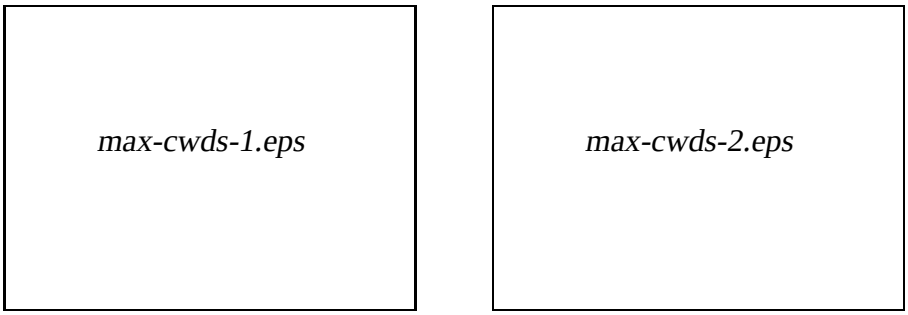

Figure 7.1: Truncated discretised Gaussian distributions used for generating clause weights for Weighted Uniform Random-3-MAXSAT test-sets; distributions for $\mu = 500$ and various values of σ' *(left)* and δ *(right)*.

weights and hence together with the range $[1, 2\mu - 1]$ the number of values that clause weights can take. It may be noted that for $\sigma' = 0$ or $\delta > 2\mu$, all clause weights are identical, which renders the respective instances equivalent to unweighted MAX-SAT instances. Furthermore, for very large values of σ'/μ, the clauese weight distributions approach a uniform distribution over the interval $[1, 2\mu - 1]$. These test-sets have been designed and used for investigating the impact on the variance and granularity on the performance of MAX-SAT algorithms.

MAX-SAT Encodings of Other Combinatorial Problems

Many $\mathcal{NP}$-hard combinatorial optimisation problems can be quite easily and naturally encoded into MAX-SAT. A good example for this is the following Minimum-Cost Graph Colouring Problem (Min-Cost GCP): Given an (undirected) edge-weighted graph $G = (V, E, w)$ and an integer k, determine a minimum cost k-colouring of G, where a k-colouring of G is a mapping a that assigns an integer from the interval $[1..k]$ to each vertex in V and the cost of a colouring a is the sum of all edge weights $w(e)$ for which e is an edge whose two incident vertices are assigned the same colour under a.

Any instance G of this problem can be transformed into a weighted MAX-SAT instance $F(G)$ as follows: For each edge $e = \{v, v'\}$ and colour

c, we create a clause $c_e = \neg x_{v,c} \vee \neg x_{v',c}$ with weight $w(e)$ (the weight of edge e in G). Furthermore, for each vertex v in G, we create a clause $c_v = \bigvee_{c=1}^{k} x_{v,c}$ with weight $\widehat{w} = \max\{\widehat{w_v} \mid v \in V\} + 1$, where $\widehat{w_v} = \sum\{w(e) \mid e \in E \text{ and } e \text{ is incident to } v\}$; intuitively, $\widehat{w}$ is defined in such a way that it just exceeds the maximum total weight of all edges incident to any particular vertex in G. It is easy to see that the optimal solution of the weighted MAX-SAT instance $F(G)$ thus obtained corresponds exactly to the optimal solution of the given Min-Cost GCP instance G. Furthermore, under the 1-flip neighbourhood, the locally optimal candidate solutions of $F(G)$ correspond exactly to the k-colourings of G. MAX-SAT-encoded Min-Cost GCP instances with integer weights have been used in some studies on SLS algorithms for MAX-SAT [Yagiura and Ibaraki, 1998; 2001; Hoos *et al.*, in preparation].

Another problem that can be easily encoded into weighted MAX-SAT is the Minimum-Cost Set Covering Problem (Min-Cost SCP), in which given a set S, a collection $C = \{S_1, \ldots, S_m\}$ of subsets $S_j \subseteq S$, and a weight function $w : C \mapsto \mathbb{R}^+$. The objective is to find a minimal cost set cover of S, where a vertex cover of S is a subset C' of C such that the sets in C' cover all elements of S, *i.e.*, $\bigcup\{C'\} = S$, and the cost of C' is the total weight of its elements, *i.e.*, $\sum\{w(S') \mid S' \in C'\}$. This problem is $\mathcal{NP}$-hard and has applications, *e.g.*, in Boolean circuit optimisation. MAX-SAT encodings of Min-Cost SCP instances from the ORLIB benchmark library [**?**] have been used for evaluating the performance of MAX-SAT algorithms [Yagiura and Ibaraki, 1998; 2001; Hoos *et al.*, in preparation].

Other hard combinatorial optimisation problems that have been encoded into MAX-SAT and used in the context of various studies on MAX-SAT algorithms include time-tabling problems (including a real-world university class scheduling problem [Cha *et al.*, 1997]) [Yagiura and Ibaraki, 1998; 2001; Hoos *et al.*, in preparation], the problem of finding most probable explanation in Bayesian networks (MPE) [Park, 2002], and the problem of minimising the crossings that arise when embedding level-graphs into a plane [**?**; Hoos *et al.*, in preparation]. In almost all cases, these problems contain hard and soft constraints, which are captured by appropriately chosen weights of the respective CNF clauses. Furthermore, all of these problems have real-world applications in diverse areas, such as system diagnosis and database design.

[hh/ts: Space permitting, we'd like to add one more section on the search space structure of various types of MAX-SAT instances.]

7.2 SLS Algorithms for MAX-SAT

Many SLS methods have been applied to MAX-SAT leading to a large number of algorithms for unweighted and weighted MAX-SAT. In this section, we present some of the most prominent and best-performing algorithms, including straightforward applications of SLS algorithms for SAT to unweighted MAX-SAT, variants of WalkSAT, Dynamic Local Search, and Tabu Search, and Iterated Local Search algorithms. Additionally, we discuss some SLS algorithms that are based on larger neighbourhoods and non-oblivious evaluation function; these approaches are rather specific to MAX-SAT. Other MAX-SAT algorithms based on other SLS methods, such as Simulated Annealing, GRASP, or Ant Colony Optimisation, will be briefly mentioned in Section 7.4.

Solving MAX-SAT Using SLS Algorithms for SAT

Any SLS algorithm for SAT can be applied to unweighted MAX-SAT in a straightforward way. The only modification required in this context is the addition of a simple mechanism that keeps track of the incumbent candidate solution and returns it at the end of the search process, provided its solution quality meets a given bound, if such a bound has been specified as an input to the algorithm. Hence, in principle any of the SLS algorithms for SAT described in Chapter 6 can be used for solving unweighted MAX-SAT.

It is not clear that SLS algorithms that are known to perform well on SAT can be expected to show equally strong performance on unweighted MAX-SAT. There is some empirical evidence that for long run-times, GWSAT obtains consistently higher solution qualities than a number of earlier SLS algorithms for MAX-SAT, including algorithms based on Simulated Annealing and Tabu Search, when applied to Uniform Random-3-SAT instances of varying constrainedness [Selman *et al.*, 1994b; Hansen and Jaumard, 1990]; however, the different termination criteria used in these comparative studies render these results somewhat inconclusive (see also **??**). Similar results

have been obtained for GSAT/TABU [Battiti and Protasi, 1997b]; these will be discussed below in more detail.

More recent results show that Novelty$^+$, one of the best-performing SLS algorithms for SAT known to-date, performs poorly compared to state-of-the-art SLS algorithms for MAX-SAT (which will be discussed below) on Uniform Random-3-SAT instances; this is particularly the case for highly constrained instances [Hoos *et al.*, in preparation]. Intuitively, WalkSAT algorithms such as Novelty$^+$ have difficulties in selecting effective search steps in situations where a relatively large number of clauses is unsatisfied: In each search step they select the variable to be flipped from an unsatisfied clause that is uniformly chosen at random; but with many unsatisfied clauses, only few of which contain variables whose flip leads to improved candidate solutions, selecting an unsatisfied clause from which such a variable can be selected becomes rather unlikely, particularly for highly constrained instances in which all candidate solutions, including optimal quality solutions, have a high number of unsatisfied clauses. GSAT algorithms, on the contrary, do not suffer from this problem, since they allow the variable whose flip achieves the maximal improvement in solution quality to be chosen with a probability that is independent from the number of unsatisfied clauses and instance constrainedness.

There are very few results on the performance of dynamic local search algorithms for SAT for unweighted MAX-SAT; recent empirical results suggest that SAPS, a state-of-the-art SAT algorithm (see Chapter 6, page **??**), outperforms GLS [Mills and Tsang, 2000] in terms of the CPU time required for finding quasi-optimal (best known) solutions for overconstrained Uniform Random-3-SAT instances, but does not reach the performance of ILS-HSS (a state-of-the-art Iterated Local Search algorithm for MAX-SAT described later in this section) on these instances [Tompkins and Hoos, in preparation]. Interestingly, an RTD analysis suggests that GLS tends to suffer from search stagnation, whereas this is not the case for SAPS, which shows regular exponential RTDs.

WalkSAT Algorithms for Weighted MAX-SAT

GSAT and WalkSAT algorithms can be generalised to weighted MAX-SAT by using the objective function for weighted MAX-SAT, *i.e.*, the total weight of the clauses unsatisfied under a given assignment, as the evaluation func-

tion based on which the variable to be flipped in each search step is selected.

A WalkSAT variant for weighted MAX-SAT with explicit hard and soft constraints was proposed by Jiang, Kautz, and Selman in 1995 [Jiang *et al.*, 1995]. Applied to standard weighted MAX-SAT, this algorithm closely resembles WalkSAT/SKC, but differs in that it allows random walk steps even in situations where "zero damage" flips are available (see Chapter 6, Section **??**). When hard constraints are explicitly identified (via a lower bound on the weights of CNF clauses that are to be treated as hard constraints), this WalkSAT algorithm restricts the clause selection in the first stage of the variable selection mechanism to unsatisfied hard constraint clauses unless all hard constraints are satisfied by the current candidate assignment. This WalkSAT algorithm for weighted MAX-SAT achieved impressive results on various sets of MAX-SAT-encoded Steiner tree problems [Jiang *et al.*, 1995]; it should be noted, however, that these results crucially rely on a particularly effective encoding of the original Steiner tree problems into MAX-SAT.

In principle, the 2-stage variable selection mechanism underlying all WalkSAT algorithms can be extended to MAX-SAT in two different ways: by using the objective function for weighted MAX-SAT in the second stage as in the WalkSAT variant by Jiang *et al.* (*we* mechanism), and by considering clause weights in the selection of an unsatisfied clause in the first stage (*wcs* mechanism) [Hoos *et al.*, in preparation]. The motivation behind the latter mechanism is based on the following observations: In situations where many clauses are unsatisfied, the probability for selecting the best clause, *i.e.*, the unsatisfied clause that contains one of the variables whose flip leads to a maximal improvement in the objective function value, can be very small when basing this selection on a uniform distribution as in standard WalkSAT. By selecting an unsatisfied clause c with a probability proportional to the weight of c, the WalkSAT search process becomes more focused on satisfying clauses with high weights. (This probabilistic clause selection method is analogous to the well-known roulette-wheel selection used in many Evolutionary Algorithms; see also Chapter 2, Section **??**.)

The *we* and *wcs* mechanisms can be used individually or combined, yielding three weighted MAX-SAT variants of any WalkSAT algorithm for SAT. A recent empirical study indicates that these variants of WalkSAT/SKC are typically outperformed by the respective Novelty$^+$ variants (note that an analogous situation holds for the SAT versions of these WalkSAT algo-

rithms). Furthermore, Novelty$^+$/*wcs+we* typically performs better than the two other variants and standard Novelty$^+$, except for satisfiable weighted MAX-SAT instances (*i.e.* instances (F, w) where F is a satisfiable CNF formula), for which standard Novelty$^+$ tends to outperform the *wcs* and *we* variants [Hoos *et al.*, in preparation]. (On the `wjnh` instances, Novelty$^+$/*wcs* also tends to perform better than Novelty$^+$/*wcs+we*.) Novelty$^+$/*wcs+we* tends to find optimal solutions to the `wjnh` instances faster (both in terms of CPU time and search steps) than other state-of-the-art algorithms for weighted MAX-SAT, including the GLS and ILS-HSS algorithms described below. On other types of weigthed MAX-SAT instances, including Weighted Uniform Random-3-SAT instances of various constrainedness, none of the Novelty$^+$ variants appears to reach state-of-the-art performance.

However, for various types of MAX-SAT-encoded instances of other problems, including minimum-cost graph colouring or minimal crossing level graphs, Novelty$^+$/*wcs+we* appears to find quasi-optimal (*i.e.*, best known) solutions in significantly less CPU time than other high-performance algorithms for MAX-SAT, such as ILS-HSS or GLS, and appears to be the best-performing MAX-SAT algorithm known todate [Hoos *et al.*, in preparation].

Dynamic Local Search Algorithms for Weighted MAX-SAT

Generalising DLS algorithms for SAT to weighted MAX-SAT raises an interesting issue: how should the dynamically changing clause penalties used within DLS interact with the fixed clause weights that are part of any weighted MAX-SAT instance? The first DLM algorithm for weighted MAX-SAT, proposed by Yi Shang and Benjamin Wah, used an evaluation function of the form $g'((F, w), a) = \sum\{clp(i)+w(i) \mid \text{clause } i \text{ is unsatisfied by } a\}$, where $clp(i)$, the penalty associated with clause i, is dynamically adjusted during the search process as in the basic DLM algorithm for SAT, and $w(i)$ is the clause weight as specified in the given weighted MAX-SAT instances (F, w) [Shang and Wah, 1997]. Different from Basic DLM for SAT, the local search procedure underlying this first DLM algorithm for weighted MAX-SAT is a iterative first improvement algorithm (based on the standard 1-flip neighbourhood relation). There is some evidence that this algorithm performs better than the WalkSAT variant by Jiang *et al.*, but does not reach the performance of the Novelty$^+$/*wsc* variants on the `wjnh` instances w.r.t. to the solution quality reached after a fixed number of search steps [Mills

and Tsang, 1999a].

Another approach for integrating clause penalties and clause weights has been followed in a straight-forward generalisation of DLM-99-SAT to weighted MAX-SAT [**?**]; this variant of DLM differs from the SAT version only in the initialisation of the clause penalties and in the parameter settings δ^+, δ^-, and δ^s. The weighted MAX-SAT variant initialises the clause penalties to $w(i) + 1$ (where i is the weight of the respective clause), and chooses the parameters δ^+, δ^-, and δ^s, which control the modification of the clause penalties during the search, individually for each clause i proportional to its weight $w(i)$. This approach for handling clause weights in the context of a dynamic local search algorithm differs notably from the one followed in the first DLM algorithm for weighted MAX-SAT. When applied to the `wjnh` instances, DLM-99-SAT for weighted MAX-SAT appears to perform better than the earlier DLM algorithm by Shang and Wah [Wu and Wah, 1999], but it typically fails to reach the performance of the Novelty$^+$/*wsc* variants.

Like DLM, GLSSAT, another high-performance dynamic local search algorithm for SAT, has been extended to weighted MAX-SAT [Mills and Tsang, 1999a; 2000]. The resulting GLSSAT variant considers the clause weights of the given weighted MAX-SAT instance only in the utility value of a clause, defined as $util(a, i) = w(i)/(1 + clp(i))$ if clause i is unsatisfied under assignment a and zero otherwise. Otherwise, the algorithm is identical to GLSSAT (see also Chapter **??**, page **??**). It is worth noting that this approach for handling clause weights is conceptually similar to the one underlying the WalkSAT/*wsc*: in both cases, the clause weights are not reflected directly in the evaluation function underlying the search process, but influence the search trajectory in a different way. In GLS for MAX-SAT, only the penalty values of clauses with maximal utility are increased after each local search phase; hence, clauses with high weights will typically receive high penalties, which biases the subsidiary local search algorithm towards preferentially satisfying them.

On the `wjnh` instances, this GLS variant performs substantially better than the previously discussed DLM and WalkSAT algorithms in terms of solution quality reached after a fixed number of iterations [Mills and Tsang, 1999a; 2000]. However, when comparing the CPU time required for finding optimal solutions, both Novelty$^+$/*wsc* and Novelty$^+$/*wsc+we* typically show better performance [Hoos *et al.*, in preparation]. For Weighted Uniform Random-3-SAT instances, GLS for MAX-SAT generally outper-

forms Novelty$^+$/*wsc+we* in terms of search steps required for finding quasi-optimal solutions; but in many cases this performance advantage is insufficient to amortise the substantially higher time-complexity of search steps in GLS. For certain types of weighted MAX-SAT instances, such as Uniform Random-3-SAT instances with low variance clause weight distributions, GLS appears to be the best-performing MAX-SAT algorithms known todate [Hoos *et al.*, in preparation].

However, GLS for MAX-SAT does not reach the state-of-the-art performance of Novelty$^+$/*wcs+we* on various types of MAX-SAT-encoded instances of other problems, such as minimum-cost graph colouring or minimal crossing level graphs. Furthermore, limited RTD analyses indicate that, different from other state-of-the-art MAX-SAT algorithms, such as Novelty$^+$/*wcs+we* and ILS-HSS (described below), GLS for MAX-SAT tends to suffer from stagnation behaviour, which often compromises the robustness of its performance; this appears to be even the case when all penalty values are regularly decayed, as in GLSSAT2 [Hoos *et al.*, in preparation].
[hh: We could include illustrative performance results for GLS *vs* **Novelty$^+$/*wcs+we*, Novelty$^+$/*wcs* on `wjnh` test-set as an example.]**

Tabu Search Algorithms for MAX-SAT

Hansen and Jaumard's Steepest Ascent Mildest Descent (SAMD) algorithm for unweighted MAX-SAT can be seen as one of the earliest applications of Tabu Search to MAX-SAT or SAT [**?**; Hansen and Jaumard, 1990]. (The name of the algorithm is derived from a formulation of MAX-SAT as a maximisation problem.) SAMD can be seen as a variant of GSAT/TABU that imposes a tabu tenure of *tl* steps only on variables flipped in non-improving steps; variables flipped in improving steps are not declared tabu. Furthermore, SAMD terminates if after a fixed number of search steps no improvement in the objective function value has been achieved. SAMD has been shown to outperform a standard SA algorithm for MAX-SAT as well as various approximation algorithms with theoretical performance guarantees (see also Section **??**) on a number of Uniform Random-k-SAT instances with $k \in \{2, 3, 4\}$ and varying constrainedness [Hansen and Jaumard, 1990]. Although GWSAT has been reported to achieve better solution qualities than SAMD [Selman *et al.*, 1994b], the differences in the underlying termination criteria and run-times make a meaningful comparison very difficult [Hansen

and Jaumard, 1990; Battiti and Protasi, 1997c].

A tabu-search algorithm that is equivalent to GSAT/TABU without random restart has been applied to unweighted MAX-SAT; experimental results on Uniform Random-3-SAT instances suggest that this variant, performs slightly better than SAMD and might exceed the performance of GWSAT [Battiti and Protasi, 1997b]. There is also some indication that a variant of this tabu search algorithm that uses an aspiration criterion (which allows a search step to be performed regardless of the tabu status of the corresponding variable if it achieves an improvement in the incumbent candidate solution) and a slightly modified tie-breaking rule for choosing one of several search steps that lead to an identical improvement in objective function value, achieves further slight performance improvements.

A further variant of tabu search for MAX-SAT, TS-YI, is based on a first improvement search strategy [Yagiura and Ibaraki, 1998; 2001]. Like all SLS algorithms for MAX-SAT discussed so far, it is based on the 1-flip neighbourhood relation and uses the objective function for evaluating the search steps. The search is started from a randomly chosen assignment, and none of the variables are tabu. Then, in each step, the neighbourhood of the current variable assignment is scanned in random order and the first variable flip that leads to an improving neighbouring variable assignment is executed. If no improving search step is possible, a minimally worsening step (w.r.t. to the standard evaluation function) is performed. Any variable that is flipped is declared tabu for a fixed number *tl* of subsequent search steps. The search process is terminated after a fixed CPU time or a fixed number of search steps.

TS-YI has been applied to various types of unweighted and weighted MAX-SAT instances. There is some empirical evidence that for unweighted MAX-SAT instances generated according to the random clause length model, this tabu search algorithm appears to perform better than WalkSAT/SKC (with optimal noise parameter setting) and substantially better than basic GSAT. For various test-sets of weighted MAX-SAT instances, particularly MAX-SAT-encoded minimum cost graph colouring, set cover, and time tabling problems, its performance appears to be worse than that of WalkSAT/SKC but substantially better than that of basic GSAT [Yagiura and Ibaraki, 2001]. While it is not clear how its performance compares to that of the previously discussed tabu search algorithms for MAX-SAT, there is no evidence that this algorithm generally reaches or exceeds the performance

of GLS or of the *wcs* variants of Novelty$^+$.

Finally, Robust Tabu Search (RoTS; see also Chapter 2, page **??**) has recently been applied to MAX-SAT [Hoos *et al.*, in preparation]. The RoTS algorithm for MAX-SAT is closely related to GSAT/TABU for weighted MAX-SAT. In each search step, one of the non-tabu variables that achieves a maximal improvement in the total weight of the unsatisfied clauses is flipped and declared tabu for the next *tl* steps. Different from GSAT/TABU, RoTS uses an aspiration criterion which allows a variable to be flipped regardless of its tabu status if this achieves an improvement in the incumbent candidate solution. Additionally, RoTS forces any variable whose value has not been changed over the last $10n$ search steps to be flipped (where n is the number of variables appearing in the given MAX-SAT instance). This diversification mechanism helps to avoid stagnation of the search process. Finally, instead of using a fixed tabu tenure, every n search steps, RoTS randomly chooses the tabu tenure *tl* from an interval $[tl_{min}, tl_{max}]$ according to a uniform distribution. The tabu status of variables is determined by comparing the number of search steps that have been performed since the most recent flip of a given variable with the current tabu tenure; hence, changes in *tl* immediately affect the tabu status and tenure of all variables. An outline of RoTS for MAX-SAT is given in Figure 7.2. Note that if several variables give the same best improvement for the evaluation function, one of these variable is randomly chosen.

Limited empirical results indicate that on the `wjnh` instances, RoTS requires generally more search steps but in many cases less CPU time than the weighted MAX-SAT version of GLS for finding optimal solutions; but it does not reach the performance of the *wsc* variants of Novelty$^+$ on these instances. On Weighted Uniform Random-3-SAT instances, RoTS typically shows significantly better performance than Novelty$^+$/*wsc*+*we*, both in terms of search steps and CPU time required for finding quasi-optimal solutions. In terms of CPU time, it typically also exceeds the performance of GLS for MAX-SAT for both weighted and unweighted Uniform Random-3-SAT instances; this performance advantage appears to be particularly pronounced for highly constrained instances [Hoos *et al.*, in preparation].

```
procedure RoTS(F', tl_min, tl_max, maxNoImpr)
    input weighted CNF formula F',
          positive integers tl_min, tl_max, maxNoImpr
    output variable assignment â
    n := number of variables in F';
    a := randomly chosen assignment of variables in F';
    â := a;
    k := 0;
    repeat
        if (k mod n = 0) then
            tl := random([tl_min ... tl_max])
        end
        v := randomly selected variable whose flip results
                in the maximal decrease in g(F', a)
        if g(F', a with v flipped) < g(F', â) then
            a := a with v flipped
        else if ∃ variable v that has not been flipped for ≥ 10n steps then
            a := a with v flipped
        else
            v := randomly selected non-tabu variable whose flip results
                in the maximal decrease in g(F', a)
            a := a with v flipped
        end
        if g(F', a) < g(F', â) then â := a;
        k := k + 1;
    until no improvement in â for ≥ maxNoImpr steps
    return â
end RoTS
```

Figure 7.2: Algorithmic outline of the Robust Tabu Search for MAX-SAT. $g(F', a)$ denotes the total weight of the clauses in F' unsatisfied under a; a variable is tabu if and only if it has been flipped during the last *tl* search steps. (For details, see text.)

Iterated Local Search for MAX-SAT

Yagiura and Ibaraki proposed and studied a simple ILS algorithm for MAX-SAT, ILS-YI, which initialises the search at a randomly chosen assignment, uses a subsidiary iterative first improvement search procedure, and a perturbation phase that consists of a fixed number of (undirected) random walk steps; the acceptance criterion always selects the better of the two given candidate solutions [Yagiura and Ibaraki, 1998; 2001]. While ILS-YI generally appears to perform better than GSAT in terms of solution quality reached after a fixed amount of CPU time, for various sets of benchmark instances, including MAX-SAT-encoded minimum cost graph colouring problems, its performance is weaker than that of WalkSAT/SKC or TS-YI. There are some cases, in particular a large MAX-SAT encoded real-world time-tabling instance, for which ILS-YI appears to perform better than TS-YI and WalkSAT/SKC [Yagiura and Ibaraki, 2001].

Another ILS algorithm for MAX-SAT was recently proposed by Hoos, Smyth, and Stützle [Hoos *et al.*, in preparation]. This algorithm, ILS-HSS, uses the same random initialisation as most other SLS algorithms for SAT and MAX-SAT. Its subsidiary local search and perturbation phases are both based on the RoTS algorithm described above. Each local search phase executes RoTS steps until no improvement in the incumbent solution has been achieved for a given number of steps. The perturbation phase consists of a fixed number of RoTS search steps with tabu tenure values that are substantially higher than the ones used in the local search phase. In both phases, RoTS returns the highest quality assignment encountered since the beginning of the respective search phase. At the beginning of each local search and perturbation phase, all variables are declared non-tabu, irrespectively of their previous tabu status. If applying perturbation and subsequent local search to a candidate solution s results in a candidate solution s' that is better than the best candidate solution accepted since the search was initialised, the search is continued from s'. If s and s' have the same solution quality, one of them is chosen uniformly at random. In all other cases, the worse of the two candidate solutions s and s' is chosen with probability 0.9, and the better one otherwise.

Empirical results show that when comparing the CPU time required for finding optimal or quasi-optimal solutions, ILS-HSS typically performs significantly better than GLS and Novelty$^+$/*wsc+we* on weighted and un-

weighted Uniform Random-3-SAT instances; the performance advantage of ILS-HSS is particularly large for highly constrained instances with low-variance clause weight distributions. Overall, ILS-HSS appears to be the best-performing MAX-SAT algorithm for these types of instances. On the `wjnh` instances, ILS-HSS does not reach the performance of the *wsc* variants of Novelty$^+$, but finds optimal solutions for a significant fraction of the unsatisfiable instances faster (in terms of CPU time) than GLS. On the satisfiable `wjnh` instances, however, it does not reach the performance of GLS or Novelty$^+$. Similarly, for several classes of MAX-SAT-encoded instances of other combinatorial optimisation problems, such as minimal cost graph colouring problems and set covering problems, ILS-HSS performs significantly worse than GLS for weighted MAX-SAT.

Limited experimentation suggests that using a perturbation phase consisting of a sequence of random walk steps instead of the Robust Tabu Search procedure described above results in a decrease in performance.

MAX-SAT Algorithms Based on Larger Neighbourhoods

While all prominent and high-performance SLS algorithms for SAT are based on the 1-flip neighbourhood, there are very successful SLS algorithms for MAX-SAT that are based on larger neighbourhoods. Yagiura and Ibaraki studied various such algorithms, ranging from simple iterative first improvement to iterated local search methods [Yagiura and Ibaraki, 1998; 1999; 2001]. The key to the success of these algorithms is a combination of a clever reduction of the 2- and 3-flip neighbourhoods with an efficient caching scheme for evaluating moves in these larger neighbourhoods. This reduction is done in such a way that no possible improving neighbour is lost, *i.e.*, local optimality remains invariant under the neighbourhood reduction. Furthermore, under realistic assumptions, each local search step requires time $O(n + m)$ for the 2-flip neighbourhood and time $O(m + t^2 n)$ for the 3-flip neighbourhood in the average case given an input formula with n variables, m clauses, and no more than t occurrences of each variable; this result was empirically confirmed for a range of Weighted Uniform Random-3-SAT test-sets [Yagiura and Ibaraki, 1998; 1999]. **[hh: We could add an add in-depth section to explain the details of the reduction and efficient implementation.]**

Empirical results for variants of TS-YI and ILS-YI that use the reduced

2- and 3-flip neighbourhoods indicate that on various test-sets of weighted MAX-SAT instances, these larger neighbourhoods lead to significant performance improvementsin terms of the solution quality reached after a fixed amount of CPU time.Particularly for MAX-SAT-encoded minimum-cost graph colouring and set covering instances,as well as for a big, MAX-SAT-encoded real-world time-tabling instance,the 2-flip variant of ILS-YI performs better than the other versions of ILS-YI and any of the TS-YI variants. It is presently not clear whether other, state-of-the-art MAX-SAT algorithms can reach or exceed the performance of ILS-YI (or TS-YI) on these types of instances. It is also unclear whether the use of larger neighbourhoods might lead to performance improvements in state-of-the-art SLS algorithms for MAX-SAT, such as Novelty/*wcs+we*, GLS, or ILS-HHS.

Non-oblivious SLS Algorithms for MAX-SAT

All SLS algorithms for SAT and MAX-SAT discussed so far use evaluation functions that are *oblivious* in the sense that they are not affected by the degree of satisfaction of any given clause c, *i.e.*, by the number of literals that are satisfied in c under a given assignment. *Non-oblivious evaluation functions*, in contrast, reflect the degree of satisfaction of the clauses satisfied by a given variable assignment.

Theoretical analyses have shown that iterative improvement local search achieves better worst-case approximation ratios for unweighted MAX-SAT when using non-oblivious evaluation functions than when the standard, oblivious evaluation function is used which counts the number of clauses unsatisfied under a given assignment [Alimonti, 1994; 1996; Khanna *et al.*, 1994b; **?**]. In particular, using the non-oblivious evaluation functions $g_2(F, a) = 3/2 \cdot w(S_1) + 2 \cdot w(S_2)$ and $g_3(F, a) = w(S_1) + 9/7 \cdot w(S_2) + 10/7 \cdot w(S_3)$, where $w(S_i)$ is the total weight of the set of all clauses satisfied by exactly i literals under assignment a, in conjunction with iterative improvement algorithms leads to worst-case approximation ratios of 4/3 and 8/7 for MAX-2-SAT and MAX-3-SAT, respectively. (Similar non-oblivious evaluation functions and respective approximation results exist for MAX-k-SAT, $k > 3$.)

Battiti and Protasi proposed and studied a number of SLS algorithms-for MAX-SAT that make use of these non-oblivious evaluationfunctions [Battiti and Protasi, 1997c; 1997b]. The simplest of these is aniterative

best improvement algorithm; it can be seen as a variant of basic GSAT that terminates as soon as a local minimum state isreached. For this algorithm (applied to MAX-3-SAT), using thenon-oblivious evaluation function g_3 instead of the standardGSAT evaluation function leads to improved solution qualities;however, both of these algorithm perform significantly worse thanGWSAT and SAMD, except when applied to weakly constrained UniformRandom-3-SAT instances. Furthermore, GSAT, GWSAT and GSAT/TABUperform significantly worse when using a non-oblivious evaluationfunction [Battiti and Protasi, 1997b].and oblivious evaluation functions have different local minima. Based on this observation, Battiti and Protasi designed a hybrid SLS algorithm that first performs non-oblivious iterative best improvement until a local minimum w.r.t. to the non-oblivious evaluation function is reached, followed by an oblivious iterative best improvement phase that is continued beyond its first local minimum. This hybrid SLS algorithm reaches better solution qualities than SAMD for various Uniform Random-3-SAT test-sets, but its performance is inferior to GWSAT for long run-times [Battiti and Protasi, 1997c].

Better performance is achieved by H-RTS, a complex hybrid SLS algorithm that combines non-oblivious and oblivious iterative best improvement with an oblivious reactive tabu search procedure [Battiti and Protasi, 1997b]. H-RTS starts the search from a randomly chosen variable assignment; next, non-oblivious iterative best improvement steps are performed until a local minimum (w.r.t. the non-oblivious evaluation function) is reached. Then, phases of oblivious iterative best improvement (OIBI) search and reactive tabu search (RTS) are alternated until the total number of variable flips performed since initialising the search reaches $10n$, where n is the number of variables in the given MAX-SAT instance, at which point the search is re-initialised (see Figure 7.3). Each OIBI search phase ends when a local minimum w.r.t. the standard oblivious evaluation function is reached. The subsequent RTS phase performs $2(tl + 1)$ steps of oblivious iterative best improvement tabu search with fixed tabu tenure *tl*.

When the search is initialised (or restarted), *tl* is set to a fixed value *tl-init*. After each RTS phase, *tl* is adjusted based on the Hamming distance covered within that search phase (*i.e.*, the number of variables that are assigned different truth values immediately before and after the $2tl + 1$ RTS steps): if that distance is small, the tabu tenure is increased in order to diversify the search; if the distance is big, the tabu tenure is decreased to keep the

Figure 7.3: GLSM representation of the H-RTS algorithm; ... (explanation of GLSM states and transition types) ... (For details, see text.)

search process focused on promising regions of the search space. Additionally, an upper and lower bound on the tabu tenure are imposed (for details, see [Battiti and Protasi, 1997b]).

H-RTS has been applied to various sets of unweighted Uniform Random-3-SAT and Uniform Random-4-SAT instances. In terms of solution quality achieved after a fixed number of search steps (variable flips), H-RTS performs significantly better than basic GSAT, GWSAT, and GSAT/TABU, especially for large, highly constrained problem instances [Battiti and Protasi, 1997b]. Furthermore, H-RTS shows substantially more robust performance w.r.t. to the initial tabu tenure setting *tl-init* than GSAT/TABU w.r.t. to its tabu tenure parameter, *tl*. When it was first proposes, H-RTS was one of the best-performing algorithms for unweighted MAX-SAT; to date, it is unclear whether H-RTS reaches the performance of GLS, Novelty$^+$/*wcs*, or ILS-HSS. Interestingly, there is some evidence that the performance of H-RTS does not significantly depend on the initial non-oblivious local search phase, but is rather due to the (oblivious) RTS procedure.

7.3 SLS Algorithms for MAX-CSP

MAX-CSP generalises CSP analogous to the way in which MAX-SAT generalises SAT: given a CSP instances, the objective is to satisfy as many constraints as possible. The importance of MAX-CSP resides in the fact that it

is one of the simplest extensions of CSP to constrained optimisation problems; as such it is typically used as a first step for extending algorithmic CSP techniques to optimisation problems. As in Chapter 6, we will focus on finite discrete MAX-CSP, where the domains of all CSP variables are finite and discrete.

The MAX-CSP Problem

The simplest case of MAX-CSP gives all constraints the same importance and the goal is to maximise the number of satisfied constraints.

Definition 7.4 ((Unweighted) MAX-CSP)

Given a CSP instance $P = (V, \mathcal{D}, \mathcal{C})$, let $f(P, a)$ be the number of constraints satisfied under variable assignment a. The *(Unweighted) Maximum Constraint Satisfaction Problem (MAX-CSP)* is to find $a^* \in argmax\{m - f(P, a) \mid a \in Assign(P)\} = argmin\{f(P, a) \mid a \in Assign(P)\}$, *i.e.*, a variable assignment a^* that maximises the number of the satisfied constraints in P. □

As in MAX-SAT, *maximising* the number of *satisfied* constraints is equivalent to *minimising* the number of *unsatisfied* constraints; in the following we consider MAX-CSP as a minimisation problem. Note that CSP is the decision variant of MAX-CSP in which the objective is to determine whether there is a CSP variable assignment that simultaneously satisfies all constraints. The evaluation and search variant are defined as in the case of MAX-SAT.

The MAX-CSP problem arises in the context of over-onstrained CSP instances, in which it is typically impossible to satisfy all given constraints simultaneously; to deal with this situation, some of the constraints are marked as "soft" and the objective becomes to find a CSP variable assignment that maximises the number of satisfied soft constraints. MAX-CSP is a particular case of overconstrained problems, where each constraint is handled as a soft constraint and all constraints are given the same importance. Similar to the MAX-SAT case, it is straightforward to extend the MAX-CSP formalism to include constraint weights, which indicate the importance of

satisfying specific constraints. A weighted MAX-CSP instance can be defined as follows.

Definition 7.5 (Weighted CSP instance)

A *weighted CSP* instance is a pair (P, w), where P is CSP instance and $w : \{C_i \mid i \in [1 \ldots m]\} \mapsto \mathbb{R}^+$ is a function that assigns a positive real value to each constraint C_i of P; $w(C_i)$ is called the *weight* of constraint C_i. □

The objective in weighted MAX-CSP is to find a CSP variable assignment that minimises the total weight of the unsatisfied constraints.

Definition 7.6 (Weighted MAX-CSP)

Given a weighted MAX-CSP instance $P' = (P, w)$, let $f(P', a)$ be the total weight of the constraints of P satisfied under CSP variable assignment a, *i.e.*, $f(P', a) = \sum_{i=1}^{m}\{w(C_i) \mid C_i$ is a constraint of P and a satisfies $C_i\}$. The *Weighted Maximum Constraint Satisfaction Problem (Weighted MAX-CSP)* is to find a variable assignment a^* that maximises the total weight of the satisfied constraints in P, *i.e.*, $a^* \in argmin\{f(P', a) \mid a \in Assign(P)\} = argmax\{\bar{f} - f(P', a) \mid a \in Assign(P)\}$, where $\bar{f} = \sum\{w(C_i) \mid C_i$ is a constraint of $P\}$. □

The constraint weights reflect the different priorities in satisfying the respective constraints. They can be used to encode problems that involve hard constraints that must be satisfied in any feasible solution as well as soft constraints that represent an optimisation goal.

MAX-CSP is an $\mathcal{NP}$-hard problem, since generalises CSP, which itself is $\mathcal{NP}$-complete. As might be expected, even finding high-quality suboptimal solutions for MAX-CSP is difficult in the worst case: for k-ary MAX-CSP with domains of size d, achieving approximation ratios of $d^{k-2\sqrt{k+1}+1} - \epsilon$ is $\mathcal{NP}$-hard for any constant $\epsilon > 0$ is $\mathcal{NP}$-hard [Engebretsen, 2000]. The efficient approximation algorithm with provably worst-case performance guarantees is based on linear programming and randomized rounding and achievs an approximation ratio of d^{k-1} [Serna *et al.*, 1998].

Randomly Generated and Structured MAX-CSP Instances

Algorithms for MAX-CSP have been mostly evaluated on instances that are randomly generated according to the Uniform Random Binary CSP model described in Chapter 6, Section 6.5. This generative model has four parameters: the number of CSP variables, n; the domain size for each CSP variable, k; the constraint graph density, α; the constraint tightness, β. In the context of MAX-CSP, these parameters are typically chosen in such a way that the resulting instances are unsatisfiable [Wallace, 1996b; Galinier and Hao, 1997]. Random weighted MAX-CSP instances are obtained by assigning randomly chosen weights to the constraints, these are typically sampled from a uniform distribution over a given range of integers [Lau, 2002].

Other combinatorial optimisation problems from a wide range of application areas can be encoded into MAX-CSP in a straightforward way. One example for such a problem is university examination timetabling: Given a set of examinations and a set of time-slots as well as a set of students and for each student the set of examinations that student needs to take, the objective is to assign a set of of examinations to a set of time slots such that certain hard constraints are satisfied and additional criteria are optimised. (To keep things simple, this version of the problem does not capture room assignments.) A typical hard constraint is to forbid a any temporal overlaps between the examinations taken by the same student; a typical example of a soft constraint is to maintain a minimum temporal distance between any pair of examinations for the same student (see [Burke *et al.*, 1996] for an extensive list of possible constraints found in real life exam timetabling problems).

A set of benchmark instances that has been commonly used to evaluate algorithms for examination timetabling with exactly these two types of constraints was defined by Carter *et al.* [Carter *et al.*, 1996]. In particular, the soft constraints penalise timetables in which the temporal distance t between two exams taken by the same student is less than six time slots; the penalty is $6 - t$ if $t < 6$ and zero otherwise. In the weighted MAX-CSP formulation, this penalisation is represented by five constraints for every student; each of these is violated if teh temporal distance between two examinations is equal to t time slots, where $0 < t < 6$, and has a weight of $6 - t$. The hard constraints, which forbid overlapping time slots for exams

taken by the same student, are assigned weight larger than the sum of the weights of all soft constraints.

Another example of a problem that can be easily represented as MAX-CSP arises in the context of the Radio Link Frequency Assignment Problem (RLFAP). In RLFAP, the objective is to assign a limited number of available frequencies to each cell in a radio network such that the electromagnetic interferences is minimised. There exist a number of variants of this problem (see [Aardal *et al.*, 2001] for an extensive overview), the simplest being the following. Given are:

- a set of radio links
- a fixed set of frequencies available for each link
- a set of frequency separation constraints that state for each pair of links (i, j) the minimum difference d_{ij} between the frequencies assigned to the pair of links
- a cost c_{ij} for violating a frequency separation constraint for a pair of links (i, j).

[ts: We could add an explicit RLFAP example here.]

In this variant, the objective is simply to assign frequencies to links such that the total cost of violated frequency separation constraints is minimised. Instances of this problem can be easily represented as a weighted MAX-CSP instances, where the frequency separation constraint for each pair of links (i, j) is captured by a binary constraint with weight c_{ij}. Some extensions of this simple RLFAP, such as problems with pre-assigned frequencies that have high modification costs, can be encoded easily as weighted MAX-CSP by using additional unary constraints.

SLS Algorithms for Unweighted MAX-CSP

Because of the way SLS algorithms for CSP evaluate and minimise constraint violations in order to find solutions to a given CSP instance, these algorithms can generally be applied directly to unweighted MAX-CSP instances.

Variants of the Min-Conflicts Heuristic (MCH; see Chapter 6, Section 6.6) were amongst the first SLS algorithms applied to unweighted MAX-CSP.

Empirical results on a set of randomly generated MAX-CSP instances show that WMCH performs better than Basic MCH and Basic MCH with random restart [Wallace and Freuder, 1995]. Interestingly, a parametric study of WMCH's performance indicates that the performance-optimising setting of *wp* (the probability for executing random walk steps rather than basic MCH steps) depends on the number of constraints violated in the optimal solutions to the MAX-CSP: larger optimal solution quality values require smaller *wp* settings.

In a further experimental study, the performance of the same three MCH variants was compared to three additional CSP algorithms:

- the Breakout Method [Morris, 1993] (see Chapter 6, page **??**);
- EFLOP, a hybrid SLS algorithm that combines iterative improvement with value propagation techniques [**?**];
- weak commitment search, a method that starts from a complete CSP variable assignment and then tries to iteratively build sets variables that are not involved in any constraint violations [**?**].

On a set of randomly generated MAX-CSP instances, While on sets of small instances (with 30 variables and domain size 5) WMCH did not perform significantly better than these three methods, it did achieve better performance on larger instances [Wallace, 1996b].

Probably the best results for randomly generated MAX-CSP so far were reported for the Tabu Search algorithm by Galinier and Hao (TSGH) [Galinier and Hao, 1997]. (In fact, TSGH was applied to MAX-CSP before it was evaluated on soluble CSP instances.) Different from MCH variants, which in each step choose a variable involved in a conflict and then considers changing the value of this variable, TSGH determines each search step by considering the set of all variable-value pairs (v, y) for which v occurs in a currently violated constrain (see Chapter 6, page **??**).

On randomly generated MAX-CSP instances with up to 500 CSP variables and domain sizes 30, TSGH has been shown to outperform WMCH: TSGH reached the same solution quality as WMCH in about three to four times less search steps and found better quality solutions when allowed the same run-time (in terms of search steps). It should be noted that due to the speed-up techniques used in TSGH, its search steps are only slightly more

expensive than those of WMCH; the difference in CPU time was measured at about 15% [Galinier and Hao, 1997].

One may conjecture that the better performance of TSGH when compared to WMCH is a result of the the larger neighbourhood searched by TSGH in each single step. However, limited empirical results indicate that a variant of TSGH that uses random walk instead of tabu search for escaping from local optima performs significantly worse than WMCH [Galinier and Hao, 1997]. On the other hand, it is known that the restriction of the neighbourhood to variables involved in conflicts is important to reach high performance for TSGH. There is some empirical evidence that suggests that if all variable–value pairs (including those variables that are not involved in constraint conflicts) are searched, the performance of TSGH drops significantly [Hao and Pannier, 1998].

SLS Approaches to Weighted MAX-CSP

The previously described algorithms for unweighted MAX-CSP can be easily extended to weighted MAX-CSP. Somewhat surprisingly, so far this approach has remained largely unexplored. An exception is the work of Lau, who developed an approximation algorithm for weighted MAX-CSP based on semidefinite programming and randomised rounding; For domain sizes two and three, this algorithm has a fixed approximation ratio [Lau, 2002].

A variant of this algorithm that applies iterative improvement to the solution obtained from the approximation algorithm (APII) has been empirically compared to (i) an SLS algorithm that consists of a greedy construction heuristic followed by an iterative improvement procedure (GII), and (ii) an extension of MCH to weighted MAX-SAT. Applied to "forced" instances, which are randomly generated in a way that guarantees their solubility, APII achieved substantially better solution qualities than GII and MCH; on randomly generated instances that were not soluble by construction, the performance advantages observed for APII were less pronounced [Lau, 2002]. In these experiments, MCH and APII were alotted approximately the same run-time, while GII terminated within roughly 5% of this time. Furthermore, for the forced instances the approximation algorithm without the subsequent local search phase performed better than GII; this suggests that the excellent performance of APII on forced instances may be an artifact of the instance generation.

Pseudo-Boolean Optimisation

Pseudo-Boolean CSP can be seen as a restriction of CSP in which all variables have domains $\{0, 1\}$, but more expressive constraints are supported than the CNF clauses used in SAT or MAX-SAT. The Pseudo-Boolean CSP formalism can be extended to consider optimisation objectives in addition to the conventional, hard constraints. In the resulting over-constrained Pseudo-Boolean problems (OCPBP), optimisation goals are encoded as competing soft constraints. The general form of a over-constrained Pseudo-Boolean problems can be written as

$$\begin{array}{rcl} A\mathbf{x} & \geq & \mathbf{b} \\ C\mathbf{x} & \geq & \mathbf{d} \quad \textit{(soft)} \\ x_i & \in & \{0,1\}, \end{array} \tag{7.1}$$

where A and C are real valued coefficient matrices, $\mathbf{b}$ and $\mathbf{d}$ are real valued vectors and $\mathbf{x}$ is the variable vector [Walser, 1998; Walser *et al.*, 1998]. From an optimisation perspective, the formulation 7.1 is interpreted as

$$\begin{array}{rl} & \min \|C\mathbf{x} - \mathbf{d}\| \\ \textit{subject to:} & A\mathbf{x} \geq \mathbf{b} \\ & x_i \in \{0,1\}, \end{array} \tag{7.2}$$

where the metric $\|\mathbf{y}\| \equiv \sum_{\mathbf{i=1}}^{\mathbf{n}} \max\{\mathbf{0}, \mathbf{y_i}\}$ measures the degree of violation of the soft constraints.

SLS algorithms for this optimisation problem need to make use of a suitable evaluation function. In WSAT($\mathcal{PB}$), a well-known SLS algorithm for Pseudo-Boolean CSP (see Chapter 6, page **??**), the evaluation function value for a given variable assignment $\mathbf{x}$ is defined as:

$$f(\mathbf{x}) = \|\mathbf{Ax} - \mathbf{b}\|_{\mathbf{w}} + \|\mathbf{Cx} - \mathbf{d}\|, \tag{7.3}$$

where $\mathbf{w}$ is a vector of positive weights that is assigned to the hard constraints. (Similarly, also the soft constraints could be assigned additional weights to distinguish their importance.)

A major difference between the weighted MAX-CSP formalism and OCPBP using evaluation function 7.3 lies in the fact that in the latter case, the degree to which constraints are violated is taken into account. This is

possible because in OCPBP the constraints are defined as algebraic equations, while in the more general weighted MAX-CSP no assumption on the structure of the constraints is made. Furthermore, it can be shown that every OCPBP can be converted into integer linear programming problems [Walser, 1998].

To handle hard constraints efficiently, the WSAT($\mathcal{PB}$) variable selection strategy is extended by first randomly selecting an unsatisfied hard constraint with probability wp_h, while a violated soft constraint is chosen with probability $1-wp_h$, and then selecting the from this constrained the variable to be flipped, according to the strategy described in Section 6.6.

OCPBP can be extended by allowing ranges of integers instead of $\{0, 1\}$ as variable domains. The resulting overconstrained integer programs (OIPs) can be solved using WSAT(OIP), a generalisation of the WSAT($\mathcal{PB}$) algorithm that can handle integer variables. Different from WSAT($\mathcal{PB}$), WSAT(OIP) allows modifications of the current value y of a given integer variable to values y' with $|y' - y| \leq 2$. An executable of WSAT(OIP) is available at http://www.ps.uni-sb.de/ walser/wsatpb/wsatpb.html; this supersedes the earlier implementation of WSAT($\mathcal{PB}$), which can be seen as a restricted variant of WSAT(OIP).

A large number of practically relevant problems can be formulated easily and naturally within the Pseudo-Boolean CSP framework. WSAT(OIP) was tested on a variety of problems that can be encoded using Boolean variables. These problems include radar surveillance problems (which include soft constraints) and the Progressive Party Problem [Smith *et al.*, 1996]. For both problems, WSAT($\mathcal{PB}$) showed significantly improved performance over a state-of-the-art commercial integer programming package (CPLEX) and other methods for solving these problems. WSAT(OIP) also achieved excellent performance on capacitated production planning and AI planning problems, which were represented using non-Boolean integer variables [Walser *et al.*, 1998; Kautz and Walser, 1999].

[ts/hh: The description of computational results for pseudo-Boolean CSP in Chapter 6 is partially subsumed here and will be shortened accordingly.]

7.4 Further Readings and Related Work

MAX-SAT is one of the most widely studied simple combinatorial optimisation problems, and a wide range of SLS algorithms for MAX-SAT have been proposed and evaluated in the literature. Hansen and Jaumard studied a Simulated Annealing algorithm that uses the Metropolis distribution as an acceptance criterion and a standard geometric annealing schedule [Hansen and Jaumard, 1990]. This algorithm was found to perform worse than SAMD on various sets of unweighted Uniform-Random-k-SAT instances; however, in some cases, it reaches better quality solutions than SAMD with substantially higher run-times (both algorithms are terminated when no improvement in the incumbent solution has been ovserved for a specified number of search steps).

GRASP was one of the first SLS algorithms for weighted MAX-SAT [Resende *et al.*, 1997]. It was originally evaluated on the `wjnh` instances described in Section 7.1, but was later found to be substantially outperformed by the first DLM algorithm for weighted MAX-SAT [Shang and Wah, 1997] and other state-of-the-art SLS weighted MAX-SAT algorithms. Recently, Variable Neighborhood Search (VNS) [Hansen and Mladenović, 1999] has been applied to weighted MAX-SAT [Hansen *et al.*, 2000; Hansen and Mladenović, 1999]. A variant called skewed VNS, which accepts worse solutions depending on the amount of deterioration and the distance from the incumbent solution, was shown to perform much better than a basic version of VNS and a basic Tabu Search algorithm. However, it is not clear how skewed VNS performs compared to state-of-the-art algorithms for weighted MAX-SAT, such as GLS or ILS-HSS.

Roli, Blum, and Dorigo have studied various Ant Colony Optimisation algorithms for CSP and MAX-CSP. They mainly investigated different ways of using pheromones and presented limited computational results for their algorithms on a small set of MAX-SAT instances. These results indicate that their ACO algorithms (without using local search) perform substantially worse state-of-the-art algorithms for MAX-SAT [Roli *et al.*, 2001]. Evolutionary Algorithms can be easily applied to MAX-SAT because the candidate solutions can naturally be represented as binary strings and all the standard crossover and mutation operators can be applied in a straightforward way. Although some insights into the behavior of genetic algorithms for MAX-SAT have been obtained, pure genetic algorithms (with-

out local search) perform relatively poorly [Rana, 1999; Rana and Whitley, 1998; Bertoni *et al.*, 2000].

Most complete algorithms for MAX-SAT are based either on Branch & Bound type extensions of backtracking algorithms derived from the Davis-Logeman-Loveland procedure (DLL) [Davis *et al.*, 1962] or on Branch-and-Cut approaches. A comparison of an algorithm based on DLL and a Branch-and-Cut algorithm by Joy, Mitchell and Borchers [Joy *et al.*, 1997] showed that the DLL-based approach performed significantly better than the Branch-and-Cut algorithm on MAX-3-SAT, while the Branch-and-Cut algorithm was found to be superior on MAX-2-SAT problems MAX-SAT-encoded Steiner tree problems. Hence, none of the currently exisiting exact algorithms is dominating over the whole set of benchmark problems. It may be noted that the `wjnh` instances as well as some weighted MAX-SAT instances with up to 500 variables that were used to evaluate VNS [Hansen *et al.*, 2000] were solved to optimality with CPLEX, a well known general-purpose integer programming software. However, all of these methods appear to be substantially less efficient in finding high-quality solutions for large and hard MAX-SAT instances than state-of-the-art SLS algorithms [Resende *et al.*, 1997; Hoos *et al.*, in preparation].

MAX-CSP has received considerable attention from the constraint programming community as a straightforward extension of CSP to optimisation problems. MAX-CSP is a special case of partial constraint satisfaction, which involves finding values for a subset of variables satisfying only a subset of the constraints [Freuder, 1989; Freuder and Wallace, 1992]. More recently, two general framewworks for constraint satisfaction and optimisation were introduced, semi-ring based CSPs [Bistarelli *et al.*, 1997] and valued CSPs [Schiex *et al.*, 1995]. So far, most research concentrated on establishing formal comparisons of these frameworks or adapting propagation techniques or complete algorithms to solve problems formulated with these frameworks; we are not aware of SLS algorithms for the latter two frameworks.

For MAX-CSP, significant research efforts were directed to the development of efficient complete algorithms. Since the first Branch-and-Bound algorithm for unweighted MAX-CSP [Freuder and Wallace, 1992], especially the lower bounds were enhanced significantly by more refined techniques, leading to much better performing Branch-and-Bound algorithms [Wallace, 1994; 1996a; Larrosa *et al.*, 1999; Kask and Dechter, 2001; Larrosa and

Dechter, 2002; Larrosa and Meseguer, 2002].

Only few results are available for SLS algorithms for MAX-CSP other than the ones described in this chapter. Kalev Kask compared the performance of an implementation of the Breakout Method to a state-of-the-art Branch-and-Bound algorithm and found that Breakout outperforms the Branch-and-Bound algorithm for Random MAX-CSP with dense constraint graphs, while for sparse constraint graphs, the complete algorithm was slightly faster [Kask, 2000]. Hao and Pannier compared TSGH to a Simulated Annealing algorithm for MAX-CSP [Hao and Pannier, 1998]; their computational results suggest that Simulated Annealing is clearly inferior to TSGH. Battiti and Protasi [Battiti and Protasi, 1999] extended H-RTS to the Maximum k-Conjunctive Constraint Satisfaction problem (MAX-k-CCSP), where each constraint consists of the conjunction of up to k literals and the goal is to satisfy as many conjunctive clauses as possible.

As previously stated, the overconstrained integer programs (OIP) model introduced by Walser is a special case of integer linear programming (ILP). There exist several SLS algorithms for 0–1 ILP (*i.e.*, Pseudoboolean optimisation) and ILP with general integer variables. Computational results by Walser [Walser, 1998] suggest that the "general-purpose" Simulated Annealing stratey (GPSIMAN) by Conolly [Connolly, 1992] is outperformed by WSAT(OIP) on a variety of problems [Walser, 1998]. Extensions of GPSIMAN were later applied by Abramson *et al.* to set partitioning problems [Abramson *et al.*, 1996]. Abramson and Randall applied Simulated Annealing to encodings of optimisation problems into general ILP problems [Abramson and Randall, 1999] and later introduced a modelling environment based on dynamic list structures [Randall and Abramson, 2001]. Recently, adaptations of evolutionary algorithms and GRASP for integer linear programming were proposed [Pedroso, 1999; Neto and Pedroso, 2001].

Several SLS algorithms were developped to tackle the more general mixed integer linear programming problem (MILP), that allow $\{0, 1\}$ variable domains as well as continuous intervals domains. Obviously, these algorithms can also be applied to pure ILP problems, which can be seen as a special case of MILP in whitch no continuous variables occur. For an overview of SLS algorithms for MILP we refer to [kketangen, 2002].

7.5 Summary

MAX-SAT is the optimisation variant of SAT in which the goal is to find a variable assignment that maximises the number or total weight of satisfied clauses. As one of the conceptuallly simplest hard combinatorial optimisation problems, MAX-SAT is of considerable theoretical interest. Furthermore, a diverse range of hard combinatorial optimisation problems, many of which have direct real-world applications, can be encoded into MAX-SAT efficiently and naturally. By using appropriately chosen clause weights and solution quality bounds, combinatorial optimisation problems with hard and soft constraints can be represented by weighted MAX-SAT instances.

Considerable effort has been spent in designing efficient (*i.e.*, polynomial-time) approximation algorithms for MAX-SAT that have certain worst-case performance guarantees. For widely used benchmark problems for MAX-SAT, including test-sets of randomly generated MAX-SAT instances as well as encodings of other combinatorial optimisation problems, such as set covering and time-tabling, into MAX-SAT, these approximation algorithms do not reach the performance of even relatively simple SLS algorithms. Furthermore, different from the situation for SAT, systematic search algorithms for MAX-SAT are substantially less efficient that SLS algorithms in finding high-quality solutions to typical MAX-SAT instances.

The most successful SLS algorithms for MAX-SAT fall into four categories: Tabu Search algorithms, in particular Robust Tabu Search (RoTS) and Reactive Tabu Search (H-RTS); Dynamic Local Search algorithms, particularly Guided Local Search (GLS); Iterated Local Search algorithms, particularly the ILS algorithm by Hoos, Smyth, and Stützle (ILS-HSS); and generalisations of high-performance SAT algorithms, in particular Novelty$^+$ with weighted clause selection (*wcs*). Some of these algorithms reach state-of-the-art performance on mildly overconstrained instances whose optimal solutions leave relatively few clauses unsatisfied (GLS as well as Novelty$^+$ and its variants for weighted MAX-SAT seem to fall into this category), while others, such as ILS-HSS, appear to be state-of-the-art for highly over-constrained instances.

All of these algorithms make use of information on the search history, mainly in form of a tabu list or dynamically adjusted clause penalties. There is some evidence that by using large neighbourhoods, such as reduced versions of the 2-flip and 3-flip neighbourhoods, high-performance ILS and

Tabu Search algorithms for MAX-SAT can be further improved; these improvements, however, critically rely on efficient mechanisms for searching these larger neighbourhoods. On the other hand, although the use of non-oblivious evaluation functions, *i.e.*, evaluation functions that are not indifferent w.r.t. to the number of literals that are simultaneously satisfied in a given clause, leads to theoretical and practical improvements in the performance of simple iterative improvement methods for unweighted MAX-SAT, there is little evidence that non-oblivious evaluation functions are instrumental in reaching state-of-the-art SLS performance on MAX-SAT instances of any type.

Although a wide range of other SLS methods have been applied to MAX-SAT, including Simulated Annealing, GRASP, ACO, and Evolutionary Algorithms, there is currently no evidence that any of these can achieve state-of-the-art performance.

MAX-CSP can be seen as a generalisation of CSP where the objective is to find a CSP variable assignment that maximises the number or total weight of satisfied constraints. Current empirical results suggest that the best performing SLS algorithms for CSP are also best for MAX-CSP; in particular, the Tabu Search algorithm by Galinier and Hao (TSGH) appears to be the most efficient algorithm for unweighted MAX-SAT known todate. However, most existing experimental studies on SLS algorithms for MAX-CSP are limited to particular classes of randomly generated MAX-CSP instances. Furthermore, the potential of many advanced SLS approaches, such as Dynamic Local Search or Iterated Local Search, in the context of MAX-CSP is largely unexplored. Overall, considerably more research is necesary to yield a more complete picture on the relative performance and behaviour of SLS algorithms for MAX-CSP.

On the other hand, generalisations of WalkSAT to overconstrained pseudo-Boolean problems and integer programs, which can be seen as special cases of MAX-CSP, have been used successfully to solve various application problems and in many cases achieved substantially better performance than specialised algorithms and state-of-the-art commercial optimisation tools. Still, compared to state-of-the-art complete integer or constraint programming algorithms, SLS methods for these problems are much less explored, leaving very likely considerable room for further improvement.

7.6 Exercises

Exercise 7.1 (Easy) How can an implementation of a standard SLS algorithm for SAT, such as GSAT, be used for solving weighted MAX-SAT instances with integer clause weights? Discuss potential drawbacks of this approach to solving weighted MAX-SAT instances.

Exercise 7.2 (Easy) Consider the following minimum cost graph colouring problem: ...

[ts/hh: specification of min-cost GCP instance will be added here, including graphical illustration.]

Represent this problem

(a) as a weighted MAX-SAT instance;

(b) as a weighed (discrete finite) MAX-CSP instance.

Exercise 7.3 (Medium) Give an extended definition of weighted MAX-SAT that allows weights to arbitrary subformulae of a propositional formula. Discuss potential advantages and disadvantages of such a generalised version of weighted MAX-SAT, particularly w.r.t. to solving such problems with SLS algorithms.

Exercise 7.4 (Easy) Give a detailed description of the examination timetabling problem in terms of a weighted MAX-CSP instance. Exemplify your encoding by applying it to a small examination timetabling instance with four exams, six students and four time slots and encode it as a weighted MAX-CSP.

Exercise 7.5 (Medium) Extend the definitions of weighted MAX-CSP to allow penalties to be given to assignments of particular values to CSP variables. Does this increase the representative power of weighted MAX-CSP?

Exercise 7.6 (Medium) Formulate the Frequency Assignment Problem, which was introduced in Section 7.3,

(a) as a Pseudo-Boolean optimisation problem;

(b) as a weighted (discrete finite) MAX-CSP instance.