22 Random Local Iterative Search Heuristics

22.1 RWalkSAT

In this chapter we consider a fairly wide range of algorithms for K-SAT and other constraint satisfaction problems (CSPs), all of which are in the family of local rearrangement rather than construction heuristics. Many of them are somewhat carelessly called "walksat", although there are several different heuristics lumped under the general phrase. We shall separate them by their histories.

The first proposal for a local improvement heuristic for random CSPs was made by Papadimitriou in 1991 [160]. In this method, one starts with a random configuration of the variables of a SAT formula and focuses on the unsatisfied clauses. In each step, one chooses an unsatisfied clause, selected at random, and chooses one variable found within that clause, also selected at random. Reversing the value of that variable has the primary effect of satisfying the clause. It may also satisfy other clauses that were unsatisfied or cause other clauses to become unsatisfied. These secondary effects should cancel out initially, but as the density of unsatisfied clauses decreases, one would expect that the clauses that are made unsatisfied would increase in number and limit the effectiveness of this heuristic. However, Papadimitriou was able to prove that this heuristic will solve 2-SAT formulas with $\alpha \leq 1$ (in the SAT phase) in at worst N^2 steps. We will refer to this simple rule as RWalksat, since it is essentially a random walk with one clever trick.

Actually, the heuristic works much better than that. Alekhnovich and Ben-Sasson [7] proved that it will reach a solution of 3-SAT random formulas with $\alpha < 1.63$ at a cost in steps that is linear in N. They optained an upper bound for the length of the search by use of the pure literal rule. In fact, it works in linear time over an even wider range than this analysis could prove. Two recent studies [200, 16] discovered that RWalkSAT finds satisfying configurations for 3-SAT in time linear in N as long as $\alpha \leq 2.7$. Although the search cost increases only linearly with increasing N, the proportionality constant diverges as α approaches its "dynamical transition", α_{d}. When $\alpha > \alpha_{\mathrm{d}}$, the cost of solving the problem using RWalkSAT increases exponentially with N. There is a simple explanation for this. In the easy region, $\alpha \leq \alpha_{\mathrm{d}}$, the random walk reaches a ground state directly, but at larger values of α, the initial decrease in energy reaches a nonzero average value, but with large fluctuations about that average value. Above α_d, the time to reach a satisfying configuration using RWalkSAT is dominated by the waiting time for a large enough fluctuation about this average to occur that a ground state is reached.

Desroulers and Monasson [48] have shown that the characteristics of the endpoint of the linear region and onset of exponential cost are common to several algorithms. They propose, based on an analysis of decimation rules, that the probability of finding a solution with these simple heuristics has a universal form, decreasing as $\exp(-N^{1/6})$. In addition, the width of the crossover region scales as $N^{-1/3}$. They argue that this form should apply to all heuristics that work for sufficiently easy problems, those outside the Hard-SAT region, so this could include both construction heuristics and methods based on local moves, such as RWalkSAT.

22.2 WalkSAT

Selman, Kautz, and various coworkers have extended Papadimitriou's idea into a widely used local rearrangement search method for solving many sorts of CSP problems, both model problems and those arising in real-world contexts [196, 197]. These methods are all called WSAT, but at least six variants of the actual moves employed have been proposed. They are reviewed in [135]. The most widely used procedure is still their original version, which adds two tricks to the random variable selection of [160]. After choosing an unsatisfied clause to satisfy, they define a "greedy" move as choosing to reverse the vari-

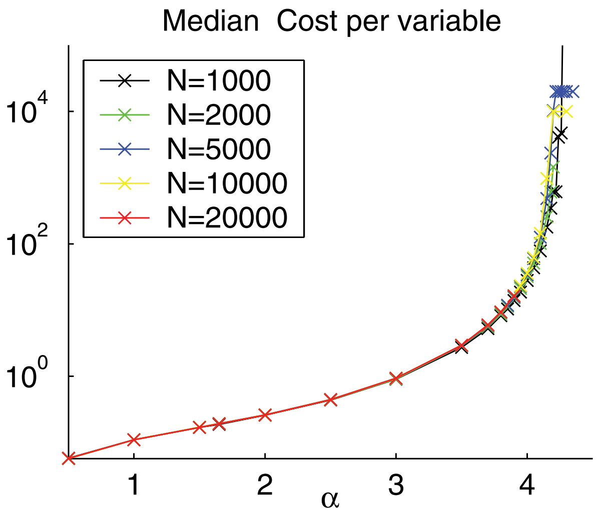

Fig. 22.1. Median cost of WSAT random walk steps per variable taken to solve 3-SAT formulas with α ranging from 0.5 to 4.3 (from [12])

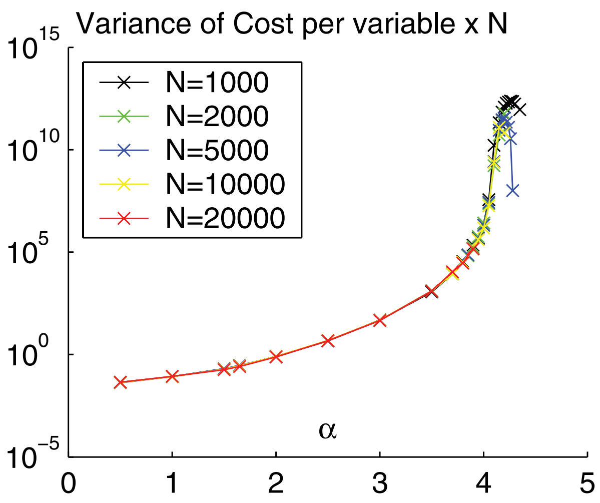

Fig. 22.2. Variance of the WSAT median cost, multiplied by N (from [12])

able that breaks the fewest previously satisfied clauses by its reversal. (One could also choose to reverse the variable for which the number of clauses that become satisfied minus the number that become unsatisfied is maximized, but this seems not to do as well.) Next, for robustness they mix moves using the random selection with moves using the greedy selection of a variable. Their recommendation is an equal proportion of the two moves, selected at random, but the fraction of each type of move used is an obvious tuning parameter. Finally, Kautz and coworkers test each variable in the unsatisfied clause to see if it is a "pure literal". If any variable proves to be a pure literal, they reverse it without considering any of the other moves.

Using the "greedy" move but without testing for pure literals, Barthel et al. [16] found little improvement in power over pure RWalkSAT. However, by testing first for pure literal moves, Aurell et al. [12] found that the cost of solving 3-SAT formulas with the full WSAT remained linear in N up to roughly $\alpha = 4.15$, as shown in Fig. 22.1. The cost per variable of the WSAT solution increases by about six orders of magnitude over this range of α, and the distribution of times observed in [12] is broad. Aurell et al. also evaluated the first four moments of this distribution to ensure that it became concentrated with increasing N on the linear dependence reported. Their results for the variance of the solution cost, scaled up by N, are shown in Fig. 22.2. The third and fourth moments of the distribution narrow in proportion to N and to N^2, respectively. Thus the cost of WSAT behaves in exactly the way that a process governed by the usual laws of large numbers is expected to behave, in spite of the very large increase in the magnitude of the effects observed.

22.3 Simulated Annealing

After this discussion of the application of algorithms specifically designed for the satisfiability problem, we would like to mention that this problem can also be solved by a standard approach like simulated annealing (SA). For the application of SA, a cost function $\mathcal{H}(\sigma)$ for a configuration $\sigma = (x_1, \dots, x_N)$ must be defined that is then minimized with SA. We chose $\mathcal{H}$ to simply be the number of unsatisfied clauses. Thus, when the Hamiltonian $\mathcal{H}$ reaches a value of 0 some time in the optimization run, we know that all clauses are fulfilled and thus a feasible configuration has been reached.

After initializing all binary variables x_i randomly with either true or false, we simply perform a standard SA run, starting at an initial temperature given by the overall number of clauses and then reducing the temperature exponentially by a factor of 0.9 to a final temperature of 10^{-2}. In each temperature step, we performed 1000 sweeps, i. e., $1000N$ moves. Each move simply selects one variable x_i at random and intends to perform the move $x_i \to \neg x_i$. This move usually changes the number of satisfied clauses. The move is then accepted or rejected according to the Metropolis acceptance criterion. A next higher move (but there was no need to implement it) would be to try to change two randomly selected variables at the same time.

We downloaded two large libraries of benchmark instances provided by SATLIB [94, 91], namely, the satisfiable uniform random 3-SAT instances that lie in the phase-transition region and all random 3-SAT instances with backbone-minimal subinstances contributed by Singer. With our simple SA approach, we were able to solve all of these instances with system sizes (N, M) between $(20, 91)$ and $(250, 1065)$. For the smaller instances, usually only one optimization run was needed to find a configuration in which all clauses were fulfilled. For the larger instances, it was sometimes necessary to run the SA program with up to ten different random seeds in order to get a feasible configuration.

Figure 22.3 shows the results of applying SA to one of the benchmark instances, using the parameters given above. Just as for the traveling salesman problem, we find that the mean energy decreases sigmoidally with decreasing temperature and finally freezes in the optimum value of 0. The curve of the specific heat is of course rather rough, due to the shortness of the optimization run. The specific heat exhibits its peak in the temperature range between 0.1 and 1. There might be a multipeak structure because of clustering and ordering effects. But, due to the small amount of calculation time, it could also be that the occurrence of a few peaks is only a nonequilibrium effect. The acceptance rate also decreases sigmoidally. However, it does not vanish for the smallest energies, although the energy no longer changes there. Thus, trivial moves can be performed that do not change the energy at all. Therefore, this problem instance has a degenerate ground state.

As these benchmark instances could so easily be solved with a simple SA approach, there was no need to think of more complex optimization schemes,

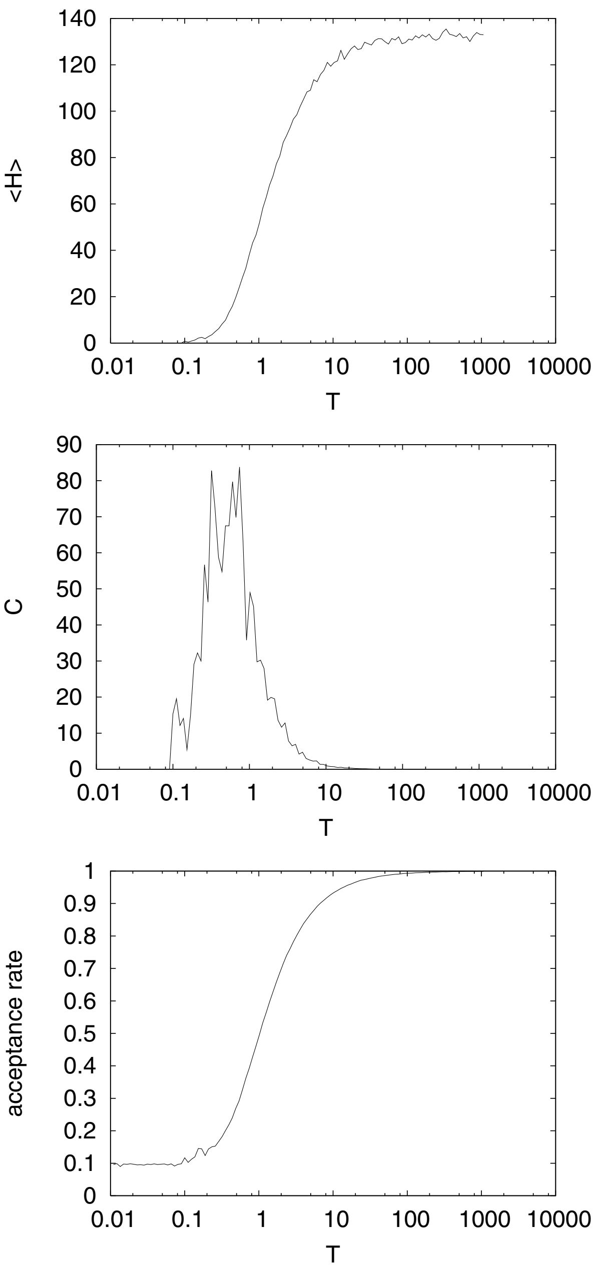

Fig. 22.3. Results of applying SA to the uf250-01.cnf 3-SAT benchmark instance [91]. The graphics show the change of the mean energy $\langle \mathcal{H} \rangle$ (*top*), of the specific heat C (*middle*), and of the rate at which the move was accepted (*bottom*) with decreasing temperature T

like, for example, ruin & recreate (R & R) moves or the searching for backbones (SfB) algorithm. But of course for more difficult instances, one might use these approaches: when working with R & R, one would randomly select some variables, remove them from the system in such a way that neither true nor false values were assigned to them, and reinsert them in a random order while trying to maximize the number of satisfied clauses by setting them to either true or false.

In the SfB algorithm, one would compare the solutions for equal parts, i. e., find out whether a variable x_i is set to either true or false in all solutions. If so, then it is again assumed that this setting also applies to the optimum solution. Therefore, the variable is removed from the system by replacing it with the value of true or false in all clauses of which it is a part. If it is replaced by true, then the clauses are automatically fulfilled, such that they can be removed from the system. If it is false, then the clause containing three variables that are connected by the OR operator are reduced to two variables with an OR operator in them. Thus, in both cases, the complexity of the system is reduced.