ARTICLES

OPTIMIZATION BY SIMULATED ANNEALING: AN EXPERIMENTAL EVALUATION; PART II, GRAPH COLORING AND NUMBER PARTITIONING

DAVID S. JOHNSON
AT&T Bell Laboratories, Murray Hill, New Jersey

CECILIA R. ARAGON
University of California, Berkeley, California

LYLE A. MCGEOCH
Amherst College, Amherst, Massachusetts

CATHERINE SCHEVON
Johns Hopkins University, Baltimore, Maryland

(Received February 1989; revision received June 1990; accepted September 1990)

This is the second in a series of three papers that empirically examine the competitiveness of simulated annealing in certain well-studied domains of combinatorial optimization. Simulated annealing is a randomized technique proposed by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi for improving local optimization algorithms. Here we report on experiments at adapting simulated annealing to graph coloring and number partitioning, two problems for which local optimization had not previously been thought suitable. For graph coloring, we report on three simulated annealing schemes, all of which can dominate traditional techniques for certain types of graphs, at least when large amounts of computing time are available. For number partitioning, simulated annealing is not competitive with the differencing algorithm of N. Karmarkar and R. M. Karp, except on relatively small instances. Moreover, if running time is taken into account, natural annealing schemes cannot even outperform multiple random runs of the local optimization algorithms on which they are based, in sharp contrast to the observed performance of annealing on other problems.

Simulated annealing is a new approach to the approximate solution of difficult combinational optimization problems. It was originally proposed by Kirkpatrick, Gelatt and Vecchi (1983) and Cerny (1985), who reported promising results based on sketchy experiments. Since then there has been an immense outpouring of papers on the topic, as documented in the extensive bibliographies of Collins, Eglese and Golden (1988) and Van Laarhoven and Aarts (1987). The question of how well annealing stacks up against its more traditional competition has remained unclear, however, for a variety of important applications. The series of papers, of which this is the second, attempts to rectify this situation.

In Part I (Johnson et al. 1989), we describe the simulated annealing approach and its motivation, and report on extensive experiments with it in the context of the graph partitioning problem (given a graph $G = (V, E)$, find a partition of the vertices into two equal-sized sets V_1 and V_2, which minimizes the number of edges with endpoints in both sets). We were concerned with two main issues: 1) how the various choices made in adapting simulated annealing to a particular problem affect its performance, and 2) how well an optimized annealing implementation for graph partitioning competes against the best of the more traditional algorithms for the problem. (For graph partitoning, the answer to the second question was mixed: simulated annealing

Subject classifications: Mathematics, combinatorics: number partitioning heuristics. Networks/graphs, heuristics: graph coloring heuristics. Simulation, applications: optimization by simulated annealing.

0030-364X/91/3903-0000$01.25

tends to dominate traditional techniques on random graphs as the size and/or density of the graphs increases, but was roundly beaten on graphs with built-in geometric structure.)

In this paper, we consider the same issues in the context of two additional well-studied, NP-hard combinatorial optimization problems: graph coloring and number partitioning. These problems were chosen because they have been studied extensively, but neither had traditionally been approached using local optimization, the algorithmic template upon which simulated annealing is based.

The graph coloring problem has widespread applications in areas such as scheduling and timetabling, e.g., see Leighton (1979), Opsut and Roberts (1981), and de Werra (1985). We are given a graph $G = (V, E)$, and asked to find the minimum k such that the vertices of G can be partitioned into *k color classes* $V_1, \ldots, V_k$, none of which contains both endpoints of any edge in E. Here, the apparent neglect of local optimization in the past may not have been totally justified. By changing the definition of the cost of a solution, and possibly by extending the notion of what a solution is, one can come up with a variety of plausible proposals for local optimization that might yield good colorings as a side effect of attempting to minimize the new cost. We investigate annealing schemes based on three of these proposals: 1) a penalty-function approach that originated with the current authors, 2) a variant that uses Kempe chain interchanges and was devised by Morgenstern and Shapiro (1986), and 3) a more recent and somewhat orthogonal approach due to Chams, Hertz and de Werra (1987). None of these versions can be counted on to create good colorings quickly, but if one has large amounts of time available, they appear to be competitive with (and often to dominate) alternative CPU-intensive approaches. (Not one of the three annealing approaches dominates the other two across the board.)

The second problem we study, number partitioning, was chosen less for its applications than for the severe challenges it presents to the simulated annealing approach. In this problem, one is given a sequence of real numbers $a_1, a_2, \ldots, a_n$ in the interval $[0, 1]$, and asked to partition them into two sets A_1 and A_2 such that

$$\left| \sum_{a_i \in A_1} a_i - \sum_{a_i \in A_2} a_i \right|$$

is minimized. The challenge of this problem is that the natural "neighborhood structures" for it, those in which neighboring solutions differ as to the location of only one or two elements, have exceedingly "mountainous" terrain, in which neighboring solutions differ widely in quality. Thus, traditional local optimization algorithms are not competitive with other techniques for this problem, in particular the "differencing" algorithm of Karmarkar and Karp (1982). Consequently, it seems unlikely that simulated annealing, which in essence is a method for improving local optimization, can offer enough of an improvement to bridge the gap. Our experiments verify this intuition. Moreover, they show that for this problem even multiple random-start local optimization outperforms simulated annealing, a phenomenon we have not observed in any of the other annealing implementations we have studied (even the mediocre ones).

Although some familiarity with simulated annealing will be helpful in reading this paper, our intention is that it be self-contained. In particular, although we shall frequently allude to Part I for background material, the reader should be able to understand the results we present here without reference to that paper. The remainder of this paper is organized as follows. In Section 1, we briefly outline the generic annealing algorithm that is the basis for our implementations, as developed in Part I of this paper. Sections 2 and 3 are devoted to graph coloring and number partitioning, respectively. Section 4 concludes with a brief summary and a preview of the third and final paper in this series, which will cover our experiments in applying simulated annealing to the infamous traveling salesman problem.

All running times quoted in this paper are for an individual processor of a Sequent Balance™ 21000 multicomputer, running under the Dynix™ operating system (Balance and Dynix are trademarks of Sequent Computer Systems, Inc.). Comparable times would be obtained on a VAX™ 750 without a floating point accelerator running under Unix™ (VAX is a trademark of the Digital Equipment Corporation; Unix is a trademark of AT&T Bell Laboratories). These are slow machines by modern standards; speedups by factors of 10 or greater are possible with currently available workstations. This should be kept in mind when evaluating some of the larger running times reported, and we shall have more to say about it in the Conclusion.

1. THE GENERIC ANNEALING ALGORITHM

Both local optimization and simulated annealing require that the problem to which they are applied be describable as follows: For each instance I of the problem, there is a set F of *solutions*, each solution S having a cost $c(S)$. The goal is to find a solution of minimum cost. (Note that both problems mentioned in the Introduction have this form.)

In order to adapt either of the two approaches to such

a problem, one must additionally define an auxiliary *neighborhood graph* on the space of solutions for a given instance. This is a directed graph whose vertices are the solutions, with the *neighbors* of a solution S being those solutions S' for which (S, S') is an arc in the neighborhood graph.

A local optimization algorithm uses this structure to find a solution as follows: Starting with an initial solution S generated by other means, it repeatedly attempts to find a better solution by moving to a neighbor with lower cost, until it reaches a solution none of whose neighbors have a lower cost. Such a solution is called *locally optimal*. Simulated annealing is motivated by the desire to avoid getting trapped in poor local optima, and hence, occasionally allows "uphill moves" to solutions of higher cost, doing this under the guidance of a control parameter called the *temperature*.

All our annealing implementations start with the parameterized generic annealing algorithm summarized in Figure 1. This generic procedure relies on several problem-specific subroutines. They are READ_INSTANCE( ), INITIAL_SOLUTION( ), NEXT_CHANGE( ), CHANGE_SOLN( ) and FINAL_SOLN( ). In addition, the procedure is parameterized by the variables *INITPROB*, *SIZEFACTOR*, *CUTOFF*, *TEMPFACTOR*, *FREEZE_LIM* and *MINPERCENT*. (See Part I for observations about the best values for these parameters and the interactions between them.) Note that the generic algorithm never deals with the solutions themselves, only their costs. The current solution, its proposed neighbor, the best solution found so far, and the cost of the latter are kept as static variables in the problem-specific part of the code. As in Part I, we allow for the possibility that the "solution

1. Call READ_INSTANCE() to read input, compute an upper bound c^* on the optimal solution value, and return the average neighborhood size N.
2. Call INITIAL_SOLUTION() to generate an initial solution S and return $c = cost(S)$.
3. Choose an initial temperature $T > 0$ so that in what follows the *changes/trials* ratio starts out approximately equal to *INITPROB*.
4. Set *freezecount* = 0.
5. While *freezecount* < *FREEZE_LIM* (i.e., while not yet "frozen") do the following:

    5.1 Set *changes* = *trials* = 0.
    While *trials* < *SIZEFACTOR*·*N* and *changes* < *CUTOFF*·*N*, do the following:

        5.1.1 Set *trials* = *trials* + 1.
        5.1.2 Call NEXT_CHANGE() to generate a random neighbor S' of S and return $c' = cost(S')$.
        5.1.3 Let $\Delta = c' - c$.
        5.1.4 If $\Delta \leq 0$ (downhill move),
            Set *changes* = *changes* + 1 and $c = c'$.
            Call CHANGE_SOLN() to set $S = S'$ and, if S' is feasible and $cost(S') < c^*$, to set $S^* = S'$ and $c^* = cost(S')$.
        5.1.5 If $\Delta > 0$ (uphill move),
            Choose a random number r in [0,1].
            If $r \leq e^{-\Delta/T}$ (i.e., with probability $e^{-\Delta/T}$),
                Set *changes* = *changes* + 1 and $c = c'$.
                Call CHANGE_SOLN().

    5.2 Set $T = TEMPFACTOR \cdot T$ (reduce temperature).
    If c^* was changed during 5.1, set *freezecount* = 0.
    If *changes/trials* < *MINPERCENT*, set *freezecount* = *freezecount* + 1.

6. Call FINAL_SOLN() to output S^*.

Figure 1. The generic simulated annealing algorithm.

space'' may have been expanded to include more than just the feasible solutions to the original problem, and thus, our algorithm is careful to output the best feasible solution found, rather than simply the best solution.

The only substantive difference between the algorithm summarized in Figure 1 and the generic algorithm of Part I is the inclusion here of *cutoffs* to limit the time spent at high temperatures (where, say, 50% or more of the moves are accepted). As we observe in Part I for graph partitioning, and confirm in preliminary experiments for the problems studied here, time spent at high temperatures does not appear to contribute much to the quality of the final solution. One way to limit this time is simply to start at lower temperatures. Solution quality can degrade, however, if we make the starting temperature too low. Cutoffs allow us to save time while still leaving a margin of safety in the starting temperature. With the addition of cutoffs, our generic algorithm closely mirrors the annealing structure implicit in Kirkpatrick's original code.

In each implementation that we discuss, we shall describe the relevant subroutines and specify the values chosen for the parameters. We shall also discuss how Step 3 is implemented, be it by the ''trial run'' approach used for graph partitioning in Part I, or more ad hoc methods.

2. GRAPH COLORING

The graph coloring problem does not seem at first to be a prime candidate for heuristics based on local optimization, and indeed none of the standard heuristics for it have been of this type. Recall that in the graph coloring problem we are given a graph $G = (V, E)$, and asked to find a partition of V into a minimum number of *color classes* $C_1, C_2, \ldots, C_k$, where no two vertices u and v can be in the same color class if there is an edge between them, i.e., if E contains the edge $\{u, v\}$. The minimum possible number of color classes for G is called the *chromatic number* of G and denoted by $\chi(G)$. Graph coloring has widespread applications, many having to do with scheduling (in situations where the vertices of G model the events being scheduled, with conflicts between events represented by edges) (de Werra 1985, Leighton 1979, Opsut and Roberts 1981).

Because graph coloring is NP-hard, it is unlikely that efficient optimization algorithms for it exist (i.e., algorithms guaranteed to find optimal colorings quickly) (Garey and Johnson 1979). The practical question is thus that of developing heuristic algorithms that find near-optimal colorings quickly. Even here, there are complexity-theoretic obstacles. Garey and Johnson (1976) show that for any $r < 2$, it is NP-hard to construct colorings using no more than $r\chi(G)$ colors. Fortunately, NP-hardness is a worst case measure, and does not rule out the possibility of heuristics that work well in practice. There have thus been many attempts to devise such heuristics, e.g., see Welsh and Powell (1967), Matula, Marble and Isaacson (1972), Johnson (1974), Grimmet and McDiarmid (1975), Brélaz (1979), Leighton (1979), and Johri and Matula (1982). Until recently, the literature has concentrated almost exclusively on heuristics that use a technique we might call *successive augmentation*, as opposed to local optimization. In this approach, a partial coloring is extended, vertex by vertex, until all vertices have been colored, at which point the coloring is output without any attempt to improve it by perturbation. In the next section, we describe several such algorithms because they illustrate annealing's competition, and they provide the basic insights that can lead us to simulated annealing implementations.

2.1. Successive Augmentation Heuristics

Perhaps the simplest example of a successive augmentation heuristic is the ''sequential'' coloring algorithm (denoted in what follows by SEQ). Assume that the vertices are labeled $v_1, \ldots, v_n$. We color the vertices in order. Vertex v_1 is assigned to color class C_1, and thereafter, vertex v_i is assigned to the lowest indexed color class that contains no vertices adjacent to v_i (i.e., no vertices u such that $\{u, v_i\} \in E$). This algorithm performs rather poorly in the worst case; 3-colorable graphs may end up with $\Omega(n)$ colors (Johnson). For random graphs with edge probability $p = 0.5$, however, it is expected (asymptotically as $n = |V|$ gets large) to use no more than $2 \cdot \chi(G)$, i.e., twice the optimal number of colors (Grimmet and McDiarmid). No polynomial time heuristic has been proved to have better average case behavior. (The best *worst case* bound proved for a polynomial time heuristic is only a slight improvement over the worst case bound for SEQ: Berger and Rompel (1990) improve on constructions of Johnson (1974) and Wigderson (1983) to construct an algorithm that will never use more than $O(n(\log\log n/\log n)^3)$ times the optimal number of colors.)

Experimentally, however, SEQ is outperformed on average by a variety of other successive augmentation algorithms, among the best of which are the DSATUR algorithm of Brélaz and the Recursive Largest First (RLF) algorithm of Leighton. The former dynamically chooses the vertex to color next, picking one that is adjacent to the largest number of distinctly colored vertices. The latter colors the vertices one color class at a

time, in the following "greedy" fashion. Let C be the next color class to be constructed, V' be the set of as-yet-uncolored vertices, and U be an initially empty set of uncolored vertices that cannot legally be placed in C.

1. Choose a vertex $v_0 \in V'$ that has the *maximum* number of edges to other vertices in V'. Place v_0 in C and move all $u \in V'$ that are adjacent to v_0 from V' to U.
2. While V' remains nonempty, do the following:
   Choose a vertex $v \in V'$ that has a maximum number of edges to vertices in U. Add v to C and move all $u \in V'$ that are adjacent to v from V' to U.

Let G_R be the residual graph induced by the vertices left uncolored after C is formed (i.e., the vertices in U when V' has finally been emptied). The goal of this procedure is to make C large while assuring that G_R has as many edges eliminated from it as possible, with the additional constraint that since v_0 has to be in *some* color class it might as well be in this one.

To get a feel for the relative effectiveness and efficiency of these three algorithms, let us first see how they do on what has become a standard test case for graph coloring heuristics, the 1,000-vertex random graph. In the notation of Part I, this is $G_{1,000,0.5}$, the 1,000-vertex graph obtained by letting a pair $\{u, v\}$ of vertices be an edge with probability $p = 0.5$, independently for each pair. Although unlikely to have much relevance to practical applications of graph coloring, such a test bed has the pedagogical advantage that results for it seem to be stable (behavior on one graph of this type is a good predictor for behavior on any other) and that different heuristics yield decidedly different results for it. Papers that have used this graph as a prime example include Johri and Matula (1982), Bollobás and Thomason (1985), Morgenstern and Shapiro (1986), Chams, Hertz and de Werra (1987). (We shall subsequently consider a selection of other types of graphs, but the well-studied $G_{1,000,0.5}$ graph provides a convenient setting in which to introduce our ideas.)

Figure 2 presents a histogrammatic comparison of the three algorithms on a typical $G_{1,000,0.5}$ random graph. Although none of the three algorithms is, as defined, a randomized algorithm, each depends, for tie-breaking if nothing else, on the initial permutation of the vertices. By varying that permutation, one can get different results, and the figure plots histograms obtained for the results of each for 100 different initial permutations. Note that each algorithm produces colorings within a tight range, that the ranges do not overlap, and that RLF is significantly better than the other two, albeit at a substantial cost in running time. Its average of 107.9 colors is also better than the results reported in Johri and Matula for other successive augmentation algorithms, such as DSATUR With Interchange (111.4) and Smallest Last With Interchange (115.0).

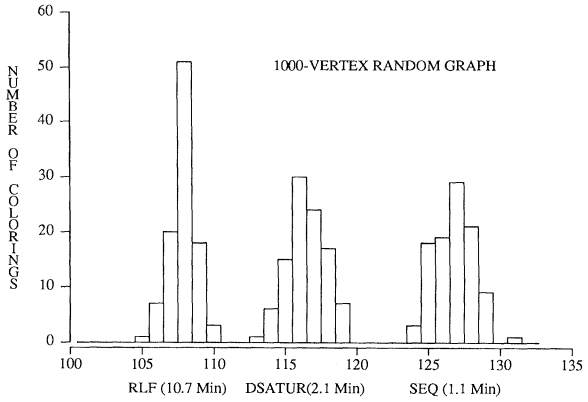

Figure 2. Histogram of the colorings found by running each of RLF, DSATUR and SEQ for 100 different starting permutations of the vertices of a typical $G_{1,000,0.5}$ random graph. (The average times per run on a Sequent processor are given in parentheses.)

None of these algorithms, however, uses close to the optimal number of colors for $G_{1,000,0.5}$, which is estimated to be about 85 by Johri and Matula. (This is only a heuristic estimate. All that can currently be said with rigor is that $\chi(G_{1,000,0.5}) \geq 80$ with very high probability, as shown by Bollobás and Thomason.) It appears likely that, if we want to approach 85 colors, we will need much larger running times than used by the typical successive augmentation algorithms. Moreover, given the narrow variance in results of such algorithms, as typified by the histograms in Figure 2, the approach of performing multiple runs of any one seems unlikely to yield significantly better colorings even if a large amount of time is available. Thus, the way is open for computation-intensive approaches such as simulated annealing.

2.2. Three Simulated Annealing Implementations

Despite the lack of traditional neighborhood search algorithms for graph coloring, the problem has proved a surprisingly fertile area for simulated annealing implementations. We will describe and compare three serious candidates.

2.2.1. The Penalty Function Approach

We begin with the historically first of the three, the one with which we began our studies in 1983. This approach was motivated in part by the success of RLF.

Problem-Specific Details. Consider the following neighborhood structure, one that, as in the graph partitioning implementation of Part I, involves infeasible solutions and penalty functions. A *solution* will be *any* partition of V into nonempty disjoint sets $C_1, C_2, \ldots, C_k$, $1 \leqslant k \leqslant |V|$, whether the C_i are legal color classes or not. Two solutions will be neighbors if one can be transformed to the other by moving a vertex from one color class to another. To generate a random neighbor, we will randomly pick a (nonempty) color class C_{OLD}, a vertex $v \in C_{OLD}$, and then an integer i, $1 \leqslant i \leqslant k+1$, where k is the current number of color classes. The neighbor is obtained by moving v to color class C_i. If $i = k+1$ this means that v is moved to a new, previously empty class. If v is already in class C_i we try again. Note that this procedure biases our choice of v toward vertices in smaller color classes, but this is presumably desirable because it is our goal to empty such classes.

The key to making annealing work using this neighborhood structure is the cost function we choose, and here is where we adapt the general philosophy of RLF, which constructs its colorings with the aid of a subroutine for generating large independent sets. Our cost function has two components, the first favors large color classes, the second favors independent sets. Let $\Pi = (C_1, \ldots, C_k)$ be a solution, and E_i, $1 \leqslant i \leqslant k$ be the set of edges from E both of whose endpoints are in C_i, i.e., the set of *bad* edges in C_i. We then have

$$cost(\Pi) = -\sum_{i=1}^{k} |C_i|^2 + \sum_{i=1}^{k} 2|C_i| \cdot |E_i|.$$

An important observation about this cost function is that all its local minima correspond to legal colorings. To see this, suppose that E_i is nonempty, and let v be an endpoint of one of the bad edges contained in E_i. Note that moving v from C_i to the previously empty class C_{k+1} reduces the cost function because we reduce the second component of the cost by at least $2|C_i|$ while increasing the first by at most

$$|C_i|^2 - \left((|C_i| - 1)^2 + 1^2\right) = 2|C_i| - 2.$$

Observe that this cost function does not explicitly count the number of k of color classes in Π; we hope to minimize this as a *side-effect* of minimizing the cost function. The use of such indirect techniques has accounted for more than one practical success claimed for annealing, from the graph partitioning problem mentioned before to problems of circuit layout and compaction (e.g., see Kirkpatrick, Gelatt and Vecchi 1983, Vecchi and Kirkpatrick 1983, Collins, Eglese and Golden 1988). Note also that for a given number of color classes the cost function is biased toward colorings that are unbalanced, rather than ones where all classes are approximately the same size. Consequently, an optimal solution with respect to this cost function need not use the minimum possible number of colors, although in practice, this does not seem to be a major drawback. Moreover, the bias may be profitable in certain applications. For instance, colorings for which $\Sigma|C_i|^2$ is maximized are precisely what is needed in a scheme for generating error-correcting codes due to Brouwer et al. (1990), and our annealing software has been useful in this application, as reported in that paper. (Aarts and Korst (1989) describe an alternative cost function with respect to which solutions of minimum cost *do* have the minimum possible number of colors, but their function has other drawbacks.)

To complete the specification of the problem-specific details in our penalty function implementation, we must say how we generate initial solutions. One possibility would be to start with all the vertices in a single class C_1; the other extreme would be to start each vertex in its own unique one-element class. On the basis of limited experiments, an intermediate approach seems reasonable, in which one assigns the vertices randomly to *CHROM_EST* classes, where *CHROM_EST* is a rough estimate of the chromatic number. The neighborhood size returned is then $CHROM\_EST \cdot |V|$, a good estimate of the number of neighbors a solution will have toward the end of the annealing schedule. We did not follow this approach precisely, however. For our $G_{1,000,0.5}$ graph, we set $CHROM\_EST = 90$, a reasonable estimate, but then for simplicity we left it at this value in our experiments with other graphs, even though in some cases 90 was a substantial over or underestimate. Given that we were varying *SIZEFACTOR* anyway, errors in neighborhood size were not deemed to be significant, and fixing *CHROM_EST* left us with one less parameter to worry about.

Nevertheless, the effective neighborhood size (the number of neighbors that a near-optimal solution can have) can be substantially bigger than that for graph partitioning in Part I (where it was simply the number of vertices). Here, the higher the chromatic number, the bigger the effective neighborhood size gets. Assuming, as our experiments with graph partitioning suggest, that the number of trials we should perform at each temperature must be a sizeable multiple of this neighborhood size, we can see that we are in for much larger running times than we encountered with graph partitioning: for $G_{1,000,0.5}$ the running times might blow up by a factor of 90 or more!

As with graph partitioning, however, the time for proposing and accepting moves is manageable. If we

store the graph in adjacency matrix form and the color classes as doubly-linked lists, the time to propose a move is proportional to the sizes of the two color classes involved, and the time to accept a proposed move is constant. (Our data structures are optimized for dense graphs with relatively small color classes, as these are common in applications and are the main ones we study in this paper. For sparser graphs with larger color classes, it may be more appropriate to represent the graph in adjacency list form and maintain an array that specifies the color of each vertex. The average time for proposing a move would then be proportional to the average vertex degree.)

Penalty Function Local Optimization. Before we turn to the implementation details in the generic part of the algorithm, let us briefly examine how well this neighborhood scheme performs in the context of pure local optimization. In our implementation of local optimization based on this scheme, we limit ourselves to a maximum of 200 color classes, thus giving us at most $200 \mid V \mid$ possible moves from any given partition. We start with a random assignment of the vertices to 200 color classes, and a random permutation of the $200 \mid V \mid$ possible moves. We then examine each move in turn, performing the move only if it results in a net decrease in cost. Once all $200 \mid V \mid$ moves have been considered, we re-permute them and try again, repeating this procedure until for some permutation of the moves, none is accepted. Then we know we have reached a locally optimal solution and stop.

For our standard $G_{1,000,0.5}$ random graph, we performed 100 runs of this local optimization algorithm. The results, although better than what was obtainable in practice by pure sequential coloring, were unimpressive: the average time per run was 37.3 minutes (slower than RLF), but the median solution used 117 colors (worse than both RLF and the much faster DSATUR algorithm). No solutions were found using fewer than 115 colors. Fortunately, this neighborhood structure is better for simulated annealing than for local optimization.

Generic Details of Penalty Function Annealing. Although all our annealing implementations follow the generic outline of Figure 1, certain parameters and routines therein must be specified before the description of any given implementation is complete. For penalty function annealing, we obtained our starting temperature by trial and error, discovering that a single initial temperature usually sufficed for a given class of graphs. In the case of $G_{n,0.5}$ graphs, an initial temperature of 10 tended to yield an initial acceptance ratio between 0.3 and 0.4, which seemed adequate based on the experiments of Part I. (Limited experiments with higher starting temperatures yielded longer running times but no better solutions.) To further reduce running time, we used cutoff with $CUTOFF = 0.10$.

For our termination condition, we set $MINPERCENT = 2\%$, allowing for the likelihood that a certain small number of zero-cost moves will always be possible and hence accepted, and set $FREEZE\_LIM = 5$. Finally, rather than perform explicit exponentiation each time we need the value $e^{-\Delta/T}$, we use the table-lookup method described in Part I for quickly obtaining a close approximation. (This method is also used in our other two annealing implementations.) Running times were then varied by changing the values of *TEMPFACTOR* and *SIZEFACTOR*.

The Dynamics of Penalty Function Annealing. Figure 3 presents "time exposures" of an annealing run under this scheme on our $G_{1,000,0.5}$ graph with

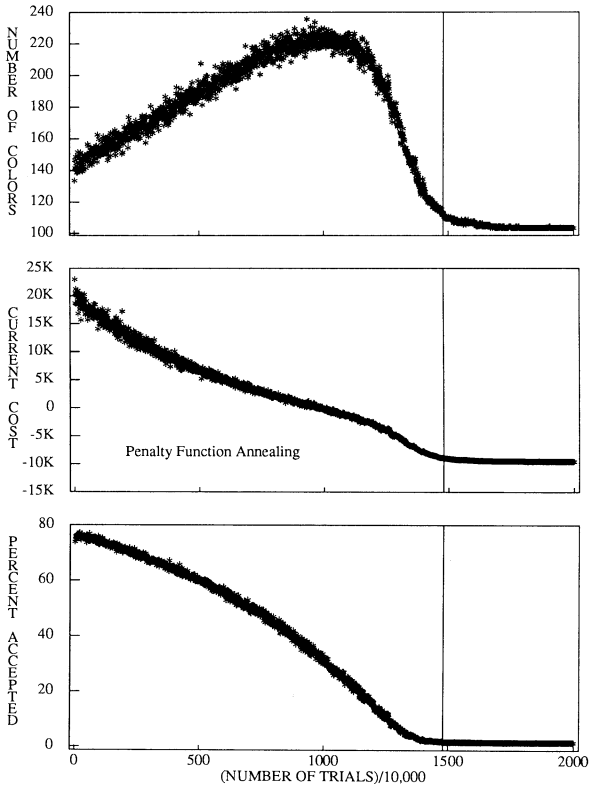

Figure 3. Three views of the evolution of an annealing run for a $G_{1,000,0.5}$ graph under the penalty function annealing scheme. (The time at which the first legal coloring was encountered is marked by a vertical line in all three displays. Temperature was reduced by a factor of 0.95 every 20 data points.)

CHROM_EST = 100 (slightly higher than our standard value, although this has no significant effect on the picture), *TEMPTFACTOR* = 0.95 and *SIZEFACTOR* = 2. So that more of the total annealing process can be seen, we used an initial temperature of 96, rather than the value of 10 used in the later experiments. The temperature 10 was reached when the Number of Trials)/10,000 reached 918. Also, for the sake of a full picture, cutoffs were turned off and the run was stopped manually once it was clear that convergence had set in.

The top display shows the evolution of the number of sets in the current partition. Note that by trial 10,000 (the first data point), the number of sets has already jumped from the initial 100 to something close to 140, and from then on it increases more or less smoothly to a peak of 233 before declining to a final value of 102. Interestingly, this behavior does not correlate with the movement of the ''current cost'' presented in the middle display, which is consistently declining. The reason for this lack of correlation lies in our method for choosing a ''random neighbor.'' Recall that we pick a random, nonempty color class and then a random member of that class to move. This introduces a bias toward the members of small classes. Since a move always reduces the size of the chosen class by one, this means that at high temperatures, small classes will tend to empty faster than they are filled. Indeed, had we started our run at a temperature at which 99% of the moves were accepted (here we start at roughly 75%), the number of colors would have been fluctuating between 30 and 40 or so. As the temperature drops, the part of the cost function that penalizes ''bad edges'' begins to take effect, driving up the number of colors until there are few enough bad edges for the component of the cost function that rewards big color classes to begin driving the number of colors back down.

The appearance of the first *legal* coloring is marked by a vertical line through the display. Although the current cost (middle display) declines more-or-less monotonically, there is a definite bump in the curve occurring slightly to the left of that line, a bump that we did not see in the time exposures of Part I for graph partitioning. Bumps of this sort regularly occur in runs of penalty function annealing. Unlike the other changes in slope for the curve, they do not reflect a similar change in acceptance rate. (The latter is depicted in the bottom display.) Annealers with physics backgrounds might suggest that such bumps indicate ''phase transition'' (Kirkpatrick, Gelatt and Vecchi). Unfortunately, there is no good explanation of why they arise or whether they can be exploited.

The total running time for the time exposure was about 11 hours. This would have been reduced significantly had we used cutoffs and the lower standard initial temperature of 10, but it is already less than the 17.9 hours it took to perform 100 runs of RLF. (Recall that RLF never used fewer than 105 colors, 3 more than we needed here). By further increasing the running time (via increased values for *TEMPFACTOR* and *SIZEFACTOR*), still better colorings are obtainable by this approach. As we shall see in Section 2.4, it is possible with this approach to get colorings using as few as 91 colors, if one is willing to spend 182 hours.

2.2.2. The Kempe Chain Approach

Preliminary reports of the penalty function implementation and the results for it inspired Morgenstern and Shapiro to proposed the following alternative, which retains the cost function but makes a major change in the neighborhood structure.

Problem-Specific Details. *Solutions* are now restricted to be partitions $C_1, \ldots, C_k$ that are legal colorings, i.e., are such that no edge has both endpoints in the same class. (Note that this means that all the sets E_i of bad edges are empty, and so the cost function simplifies to just $-\sum_{i=1}^{k} |C_i|^2$.) In order to ensure that moves preserve the legality of the coloring, Morgenstern and Shapiro go to a much more complex sort of move, one involving *Kempe chains*.

Suppose that C and D are disjoint independent sets in a graph G. A Kempe chain for C and D is any connected component in the subgraph of G induced by $C \cup D$. Let $X \Delta Y$ denote the symmetric difference $(X - Y) \cup (Y - X)$ between two sets X and Y. The key observation is that if H is a Kempe chain for disjoint independent sets C and D, then $C \Delta H$ and $D \Delta H$ are themselves disjoint independent sets whose union is $C \cup D$. This suggests the following move generation procedure: Randomly choose a nonempty color class C and a vertex $v \in C$, as in the penalty function approach. Then randomly choose a nonempty color class D other than C, and let H be the Kempe chain for C and D that contains v. Repeat the above procedure until one obtains C, v, D and H such that $H \neq C \cup D$ (i.e., such that H is not ''full''), in which case the next partition is obtained by replacing C by $C \Delta H$ and D by $D \Delta H$ in the current one. (Using a full Kempe chain in this operation simply changes the names of the two colors, a meaningless change, which is why we ignore such moves.)

This procedure appears to be substantially more expensive than the move generation procedure in the penalty function approach, and it is. It also makes substantially bigger changes in the solutions, however, and so may be worth the extra effort. Moreover, it is not *exorbitantly*

expensive, at least for dense graphs. For such graphs the color classes tend to be small, and so the following technique can find the Kempe chain H relatively quickly. Whenever a new vertex u is added to H (including the first vertex v), we scan the members of the other color class that are not yet in H and add to H each one that is adjacent to u. Furthermore, we use two auxiliary tables to help us whenever possible avoid the wasted time of constructing full Kempe chains that must be abandoned. One stores for each pair C, D the time at which they were last discovered to have a full Kempe chain; the other stores for each C the time at which it was last modified. When C and D are first chosen, we check to see whether we have seen a full Kempe chain for them since the last time they were modified, and if so, abandon their further consideration immediately.

To complete the specification of the problem-specific details of the Kempe chain approach, we must say how instance "size" is determined and how initial solutions are generated. We perform random sequential colorings for both purposes. When an instance is read, we perform a random sequential coloring, and return the size $K \mid V \mid$, where K is the number of colors used in the coloring. A second random sequential coloring is performed each time a new initial solution is requested.

Kempe Chain Local Optimization. As with the penalty function approach, the Kempe chain approach can serve as the basis for a local optimization algorithm. Here, because we are restricted to legal colorings, we limit the total allowable number of colors to 150, yielding $150 \mid V \mid$ possible moves. Initial solutions are generated by random sequential colorings as just specified. Otherwise, the details are the same as for penalty function local optimization, as outlined previously. The results were also similar (and similarly mediocre). The average running time over 100 runs was 33.3 minutes (slightly better than for the penalty function approach but still much slower than RLF), and the median number of colors was again 117.

The Dynamics of Kempe Chain Annealing. Before describing the generic parameters governing the start and finish of a run, let us compare the operation of Kempe chain annealing to the penalty function approach. Figure 4 presents "time exposures" for Kempe chain annealing, analogous to those in Figure 3 for the penalty function approach. All parameters except SIZEFACTOR were given the same values as in the penalty function run; because of the greater expense of the moves here, we reduced SIZEFACTOR from 200 to 20 (and the run still took 18 hours, as opposed to the 11 for the penalty

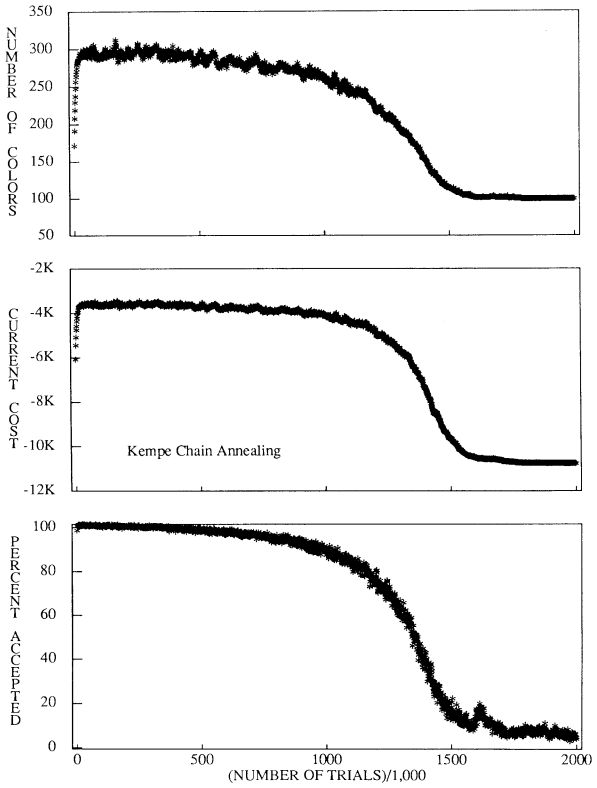

Figure 4. Three views of the evolution of an annealing run for a $G_{1,000,0.5}$ graph under the Kempe chain annealing scheme. (The temperature was reduced by a factor of 0.95 every 20 data points.)

function approach). For comparison purposes, we also set the neighborhood size to the same 100,000 value used in the previous run, and used the same starting temperature $T = 96$. A first observation is that the curves are significantly more irregular than those for penalty function annealing. For the most part, this may be attributable to the reduced value of SIZEFACTOR, since this means that each data point represents 1,000 rather than 10,000 trials, and so successive data points may be correlated more closely. There may, however, be a different reason for the extreme excursion in the acceptance rate curve. Such excursions occur in the tails of other Kempe chain runs, but at random places (unlike the regularly occurring smooth bump in the middle of the "current cost" curve for penalty function annealing in Figure 3). Here the excursion seems attributable to the topography of the Kempe chain solution space, as we shall hypothesize in more detail below.

A second difference between the runs in Figures 3 and

4 results from the fact that temperature correlates differently with acceptance percentage under the two regimes. Whereas the initial temperature $T = 96$ yields a 75% acceptance rate under the earlier approach, here it yields a 99% rate. Thus, much of the time in Figure 4 is wasted at too high a temperature. If we instead choose a starting temperature $T = 4$, which yields approximately the same acceptance ratio of 70% as did $T = 96$ for the penalty function approach, we converge to roughly the same number of colors as we do here, but in only six hours. Note also that the curves for "number of colors" and "current cost" are similar, whereas in the penalty function approach the corresponding curves differ substantially. There the number of colors initially increase while the cost declines, here they both jump quickly to high values and then undergo correlated declines. (Our Kempe chain implementation, like the one for the penalty function approach, is biased toward choosing vertices in small color classes for recoloring. Here, however, a move is roughly as likely to increase the size of the chosen class as to decrease it. Thus, classes do not tend to empty at high temperatures, and the equilibrium number is high rather than low.)

A third observation about the Kempe chain run of Figure 4 is that the solution cost converges while the percentage of accepted moves is still rather high. The penalty function run of Figure 3 is typical of most annealing implementations we have seen in that the best solution value is not seen until the acceptance rate is quite low. In the run of Figure 3, the first legal 102-coloring does not appear until the acceptance rate drops to about 0.5%. For the Kempe chain run, however, the acceptance rate still hovers around 7% when *its* best number of colors is first encountered. This is largely attributable to the topography of the solution space, in particular the structure of the neighborhoods of "good" colorings. Such a coloring is likely to have far more Kempe changes that improve its "cost" (or at least leave it the same) than it will have vertices that can individually change color without negative effect (the analogous moves under the penalty function approach).

Moreover, in explanation of the abovementioned excursion in the acceptance rate, some good colorings are likely to have far more "good" neighbors than others. This inhomogeneity of the solution space means that the acceptance rate at convergence can vary wildly from run to run, going as high as 15% one time and as low as 5% the next. This makes it difficult to fine-tune the convergence parameters of our implementation. To be conservative in the experiments to be reported, we set $MINPERCENT = 15\%$ and allow *freezecount* to go up to 10 before terminating. (We again set our initial temperatures by manual trial and error, observing as with penalty function annealing that the same initial temperature seems to work well across entire classes of graphs. For random graphs with $p = 0.5$, we use an initial temperature of $T = 5$, which generally yields an initial acceptance rate between 50% and 80%. As with penalty function annealing, we use $CUTOFF = 0.10$ in our main experiments.)

A final observation about the run in Figure 4 (and presumably the most important) is that the number of colors to which the run converges is 94, as opposed to 101 for the penalty function approach. The running time is somewhat longer, 17.9 hours versus 11, but in 17.9 hours the fewest number of colors we have been able to obtain with the penalty function approach, even using cutoffs, is only 98. In the 182 hours it takes penalty function annealing to find a 91-coloring, Kempe chain annealing can find one using only 89 colors, and, as we shall see in Section 2.4, it can do even better with just a bit more time. Thus, it appears that the extra complexity of the neighborhood structure for Kempe chain annealing can more than pay for itself, and we shall confirm this in the more extensive experiments that follow.

2.2.3. The Fixed-*K* Approach

Our final annealing implementation is derived from a paper by Chams, Hertz and de Werra, and solves a slightly different problem. Instead of attempting to minimize the number of colors used in a legal coloring, this approach attempts to minimize the number of monochromatic edges in a not-necessarily-legal coloring with a fixed number of color classes.

Problem-Specific Details. Given a graph $G = (V, E)$ and a number of colors K, the solutions are all partitions of V into K sets (empty sets are allowed), and the cost of a solution is simply the total number of edges that do not have endpoints in different classes (the "bad edges"). A partition Π_2 is a neighbor of a partition Π_1 if the two partitions differ only as to the location of a single vertex v, and v is an endpoint of a bad edge in Π_1.

Note that here the neighbor relation is not symmetric; in particular, a legal coloring has no neighbors because it has no bad edges. This is of course no problem, for if ever the annealing process finds a legal coloring, there is no point in proceeding any further. Limited experimentation indicates that this neighborhood structure is much more effective than the less-restrictive one in which v need not be an endpoint of a bad edge. (The less-restrictive neighborhood was essential in our penalty function adaptation, since the goal was to reduce the number of color classes, which might entail emptying

out a class even though it contained no bad edges.) To choose a random neighbor, we first choose a random "bad vertex" v (v is *bad* if it is the endpoint of a bad edge), and then choose a random new color class for v from among the $K-1$ that do not contain v.

The remaining problem-specific details are as follows: The size parameter is set to $K \mid V \mid$, reflecting a worst case situation in which all vertices are bad. The initial solution is a random partition into K sets.

Fixed-K Local Optimization. A local optimization algorithm based on the fixed-K neighborhood structure and cost function can be implemented in much the same way as we implemented local optimization versions of the two previous approaches. There are just $K \mid V \mid$ possible moves, which we cycle through as before. Each run starts with a random partition into K color classes.

The evaluation of an algorithm of this type is, however, a different matter. For this local optimization approach to be useful, it *must* reach a solution of cost 0 (i.e., a legal coloring). Thus, the relevant question to ask is what is the minimum K for which such a success occurs regularly. Unfortunately, the answer is quite discouraging. We performed 100 runs each for various values of K (this was not too burdensome, as the running times were much smaller than those for the two previous approaches, less than 90 seconds per run). The first value of K for which *any* of the 100 trials produced a successful coloring was $K = 141$, well above the number of colors in the *worst* coloring we ever found using sequential coloring. The success rate did not reach 50% until $K = 150$. Thus, simulated annealing has far more to redeem for this approach than for the previous two.

Generic Details and the Dynamics of Fixed-K Annealing. In performing fixed-K annealing, we again set the initial temperature manually ($T = 2.0$ yields a 50–60% initial acceptance rate for the random graphs we tested), and cutoffs are used with $CUTOFF = 0.10$. For termination we use $MINPERCENT = 30\%$ (large numbers of 0-cost moves are likely to exist), and quit when either a solution with no bad edges is achieved or the *freezecount* reaches 10. Running time is adjusted, as before, by varying *SIZEFACTOR* and *TEMPFACTOR*.

Time exposures for this approach, analogous to those in Figures 3 and 4, will be omitted, as they do not display any of the anomalies we observed for penalty function and Kempe chain annealing. That is, they look remarkably like the standard curves seen in Kirkpatrick, Gelatt and Vecchi and in Part I of this paper, except that, in those cases where a 0-cost solution (i.e., legal coloring) is found, the "converged" tail of the curve is truncated, as explained before. The one fact of note is that, with *SIZEFACTOR* = 4, *TEMPFACTOR* = 0.95 and cutoffs turned off, runs with K fixed at 96, 97 and 98 all succeeded (in roughly 11.8 hours), whereas a run with $K = 95$ took 13.9 hours and failed to find a legal coloring. Thus, this approach too seems to dominate penalty function annealing, which took 11.1 hours to find a 101-coloring. This domination is not complete, however, as we shall see in Section 2.4 when we compare the two approaches with cutoffs enabled and with their standard fixed starting temperatures in place.

Moreover, the domination assumes that one knows in advance which values of K go with a given (*SIZEFACTOR*, *TEMPFACTOR*) pair. The extra experimentation to match up these parameters provides fixed-K annealing with an additional overhead not present for the previous two approaches. We shall have more to say about this overhead after we present our more detailed experimental results.

2.3. Exhaustive Search Alternatives

As indicated, the domain of applicability for our simulated annealing implementations consists of those situations where the computing time available is far larger than that required by traditional successive augmentation heuristics like RLF, DSATUR and SEQ. Annealing is not the only way to apply large amounts of time to the problem, however, and in this section we describe two major competitors in the arena of multihour computation.

The first is exhaustive search. On seeing reports of 100+ hour running times, the reader might be excused for asking why, with all that time available, one does not simply use exhaustive search and find an optimal coloring? As we shall see, however, even when using branch-and-bound techniques to prune the search space dynamically, this approach becomes infeasible well before 100 vertices. In particular, we implement the branch-and-bound algorithm outlined in Figure 5, which includes most of the obvious shortcuts (e.g., see Brélaz) and seems competitive with the best previous implementations. Figure 6 reports the results of running this algorithm on random $G_{n,0.5}$ graphs. Three samples each were generated for $n = 40, 45, \ldots, 85, 90$. As can be seen, the growth rate in running time is clearly exponential, and only two of the three 85-vertex samples (and none of the 90-vertex samples) finished within 1,000 hours.

Our second alternative is more competitive: a parameterized generalization of RLF that can make productive use of long running times when the time is available. This algorithm, which we shall denote by XRLF, is

Program CHROM_NUM(G) (Given a graph $G = (V,E)$, outputs $\chi(G)$.)

1. Output COLOR(V,ϕ,$\lfloor |V| \rfloor$0).

Function COLOR(U,C,B,K)

U is the set of as yet uncolored vertices.
C is a set of pairs (u,i), where $u \in V - U$ and i, $1 \le i \le \lfloor |V| \rfloor$ is a color.
B is the number of colors in the best legal coloring seen so far.
$K < B$ is the number of colors used in C.

(This function returns the minimum of B and the fewest number of colors in an extension of C to a full legal coloring.)

1. If $|U| = 1$, let u be the single member of U, and do the following:
   1.1. If there is any color j, $1 \le j \le K$, such that no vertex adjacent to u has color j, return K.
   1.2. If $K + 1 < B$, return $K + 1$.
   1.3. Return B.
2. Otherwise, choose a $u \in U$ that is adjacent to already colored vertices with the maximum number of different colors, breaking ties in favor of vertices that are adjacent to the most as yet uncolored vertices.
3. If u is adjacent to $B - 1$ colors, return B.
4. For each color j, $1 \le j \le K$, to which u is not adjacent, do the following:
   4.1. Set B = COLOR($U - \{u\}$,$C \cup \{(u,j)\}$,B,K).
5. If $K < B - 1$, set B = COLOR($U - \{u\}$,$C \cup \{(u,K+1)\}$,B,$K+1$).
6. Return B.

Figure 5. Branch-and-bound algorithm for finding $X(G)$. (In the implementation, U and C are maintained in a global data structure to which pointers are passed. Data structures are also maintained for vertex degrees and color adjacencies.)

based on ideas first suggested by Johri and Matula, augmented here by a final "exact coloring" phase that is invoked when the set of vertices remaining to be colored is sufficiently small. The details of XRLF are sketched in Figure 7. To understand what is going on, note that in RLF one can view the process of constructing the next color class as a heuristic attempt to find a near-optimal solution to the following NP-hard subproblem: Find an independent set C contained in the current set V' of uncolored vertices such that $|\{\{u,v\} \in E: u \in C \text{ and } v \in V' - C\}|$ is maximized, and hence the number of edges in the residual graph is minimized. (Experiments indicate that this is a slightly better goal than simply finding an independent set C of maximum size, a goal that it tends to approximate anyway.) If one lets RLF* denote the algorithm in which the residual edge minimization subproblem is solved optimally at each step, one can view XRLF (with the parameter *EXACTLIM* set to 0) as providing a full range of approximations to RLF*. Depending on the values of the parameters *SETLIM*, *TRIALNUM* and *CANDNUM*, these range

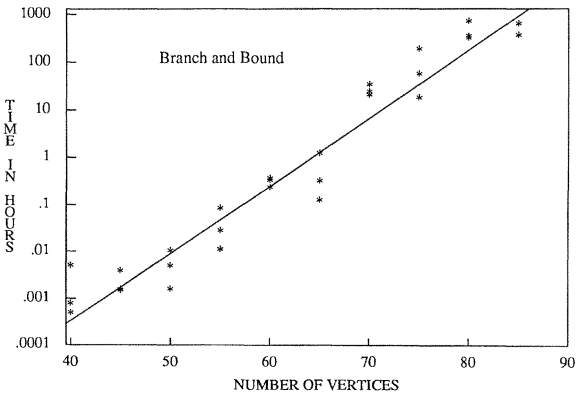

Figure 6. Running times for brand-and-bound on $G_{n,0.5}$ random graphs.

from ones that are even weaker than RLF all the way up to RLF* itself.

Algorithm XRLF constructs a new color class C by repeating the following experiment for *TRIALNUM* iterations and then taking the best result: Initially all

uncolored vertices are candidates and set C is empty. If the number of remaining candidates is less than *SETLIM*, use exhaustive search to find the best extension to C. If there are more than *SETLIM* candidates and C is empty, choose a random candidate, add it to C, and declare all its neighbors to be noncandidates. If C is not empty, randomly sample *CANDNUM* candidates, let v be one that is adjacent to the most uncolored noncandidates, add v to C, and declare all neighbors of v to be noncandidates. (When $TRIALNUM = 1$, the first vertex chosen is actually one of maximum degree, as in RLF, although when more trials are performed, random choices seem to do better. The algorithm of Figure 7 also contains an optimization to handle the case when *TRIALNUM* is so large that exhaustive search would be faster than repeated trials.) Within this basic algorithmic structure, RLF is obtained, at least approximately, by setting $(EXACTLIM, SETLIM, TRIALNUM, CANDNUM) = (0, 0, 1, N)$, where N is sufficiently large that all vertices are likely to be considered as candidates in Step 4.3.2. RLF* is obtained by setting $EXACTLIM = 0$ and $SETLIM = N$. (For random $G_{n,0.5}$ graphs, this is feasible for N as large as 250, if one uses a tightly coded implementation of Step 4.3.1 that avoids considering any subset more than once.)

We point out, however, that even though the limiting algorithm RLF* solves an NP-hard problem as a subroutine, it is not guaranteed to find optimal colorings. Constructions in Johnson can be modified to show that, as with the simpler heuristics mentioned earlier, RLF* can in the worst case use numbers of colors that are arbitrary multiples of the optimal number. Nevertheless, as we shall see, even approximations to RLF* can do well in practice, and our use of exhaustive search to finish up the coloring can make up for some of RLF*'s drawbacks. In particular, for the $G_{1,000,0.5}$ random graph we have been considering, XRLF with $(EXACTLIM, SETLIM, TRIALNUM, CANDNUM)$ (70, 63, 640, 50) finds 86-colorings in roughly 68 hours, substantially outperforming all our annealing implementations.

In the next section, we examine more carefully the tradeoffs between running time and the quality of solution for the graph coloring heuristics we have discussed, and how they depend on the type and size of graph in question. We report on experiments both with random $G_{n,0.5}$ graphs and with graphs of distinctly different character. As we shall see, the dominance of XRLF for the $G_{1,000,0.5}$ graph is not necessarily typical.

2.4. Experiments in Graph Coloring

In this section, we report more extensively on our experimental comparison of the three annealing implementations for graph coloring and their competitors. Our experiments cover a variety of types and sizes of graphs, and we discover that the approach of choice can depend strongly on the type of instance in question, and how much computing time is available. The first set of experiments covers the $G_{1,000,0.5}$ graph that has been our standard example so far. As hinted, these experiments paint a rather bleak picture of annealing (although, as we shall see subsequently, not necessarily a typical one).

2.4.1. Random $p = 0.5$, 1,000-Vertex Graphs

For the $G_{1,000,0.5}$ graph that has been our standard example, Figure 8 illustrates the tradeoff between running time and the number of colors for the four main approaches we have been considering (penalty function annealing, Kempe chain annealing, fixed-K annealing, and algorithm XRLF). Note that for all approaches, reducing the number of colors used requires substantial increases in running time (effected by altering the appropriate algorithmic parameters). From this picture, we can see that XRLF clearly dominates all three approaches based on annealing, and Kempe chain annealing clearly dominates penalty function annealing. The comparison between penalty function and fixed-K annealing is less clearcut, with an apparent crossover occurring at 92 colors. (Conclusions based on running time differences of less than a factor of two are somewhat suspect, however, given that our annealing implementations were not thoroughly optimized.)

A more detailed presentation of the data is presented in Table I, which gives for each of the approaches the computing time needed to find legal colorings with specified numbers of colors. (For comparison, we also include the median and best number of colors for 100 runs of RLF, together with the time required for 1 and 100 runs, respectively, and the percent of times the best value occurred in the 100 runs. Since only 100 runs were performed, the value quoted for "best" may not be very robust unless the percentage of occurrence is high enough. It is clear, however, that the 17.9 hours needed for 100 runs of RLF can be more productively put to use by any of the other four algorithms.) All annealing parameters except *TEMPFACTOR* and *SIZEFACTOR* (TF and SF in Table I) were fixed as described in Section 2.2. The values of the latter two parameters are given in parentheses, with *TEMPFACTOR* represented by a shorthand that emphasizes the fact that halving the cooling rate should approximately double the running time, as should doubling *SIZEFACTOR* (an effect studied in more detail in Part I). To be specific, if the code is i, *TEMPFACTOR* is approximately $0.95^{(1/i)}$, representing an i-fold decrease in the cooling rate over the base of $TEMPFACTOR = 0.95$. (The precise

Program XRLF(G) (Given graph $G=(V,E)$, outputs an upper bound on $\chi(G)$.)

1. Set $R=V$, $K=0$
   (in what follows, G_R is the subgraph of G induced by R).
2. While $|R| > EXACTLIM$, do the following.
   2.1. Set $R=R$ - IND_SET(G_R) and $K=K+1$.
3. Output K + CHROM_NUM(G_R).

Function IND_SET(H) (Given graph $H=(U,F)$, returns an independent set $C^* \subseteq U$.)

1. Set $best=-1$, $C^*=C_0=\phi$, and let D_{min} and D_{max} be the minimum and maximum vertex degrees in H.
2. If $TRIALNUM=1$ and $|U| > SETLIM$, let v_{max} be a vertex of degree D_{max} in H, and let $C_0=\{v_{max}\}$.
3. If $\min\{TRIALNUM, SETLIM+D_{min}\} \geq |U|$, set $SETLIM=|U|$ and $TRIALNUM=1$.
4. For $TRIALNUM$ iterations, grow a trial independent set C as follows:
   4.1. If $C_0 \neq \phi$, set $C=C_0$, $X=\{u \in U : \{v_{max},u\} \in F\}$.
   Else if $|U| > SETLIM$, choose a random vertex $v \in U$ and set $C=\{v\}$, $X=\{u \in U : \{v,u\} \in F\}$.
   Else set $C=X=\phi$.
   4.2 Let $W=U-(C \cup X)$ (the set of vertices still eligible for C).
   4.3. While W is not empty, do the following:
      4.3.1. If $|W| \leq SETLIM$, do the following:
         Use exhaustive search to find a set $W' \subseteq W$ that maximizes $|\{\{u,v\} \in F : u \in W' \text{ and } v \in U-(C \cup W')\}|$.
         Set $C=C \cup W'$.
         If $|\{\{u,v\} \in F : u \in C \text{ and } v \in U-C\}| > best$,
            set $C^*=C$ and
            set $best=|\{\{u,v\} \in F : u \in C \text{ and } v \in U-C\}|$.
         Exit loop beginning with statement 4.3.
      4.3.2. Set $bestdegree=-1$, $cand=\phi$.
      For CANDNUM iterations, do the following:
         Choose a random vertex $u \in W$.
         Let $s(u)=|\{\{u,v\} \in F : v \in X\}|$.
         If $s(u) > bestdegree$, set $bestdegree=s(u)$ and $cand=u$.
      4.3.3. Set $C=C \cup \{cand\}$, $X=X \cup \{v \in W : \{cand,v\} \in F\}$, and $W=W-X-\{cand\}$.
5. Output C^*.

Figure 7. The Algorithm XRLF with parameters EXACTLIM, SETLIM, TRIALNUM and CANDNUM. (Although XRLF, as described, outputs only the number of colors used, it is easily modified to produce the coloring found.)

values taken for $i=0.25, 0.5, 1, 2, 4, 8$ are 0.8145, 0.9025, 0.95, 0.9747, 0.9873 and 0.99358, respectively.) For XRLF, parameters *SETLIM* and *CANDNUM* were fixed at 63 and 50, respectively, with an entry XRLF[i, j] indicating that $TRIALNUM = i$ and $EXACTLIM = j$. Typically, we chose *EXACTLIM* to be either 0 or the maximum value for which CHROM_NUM( ) could be expected to terminate in reasonable amounts of time (in this case, $EXACTLIM = 70$), and for both options we adjust running times by

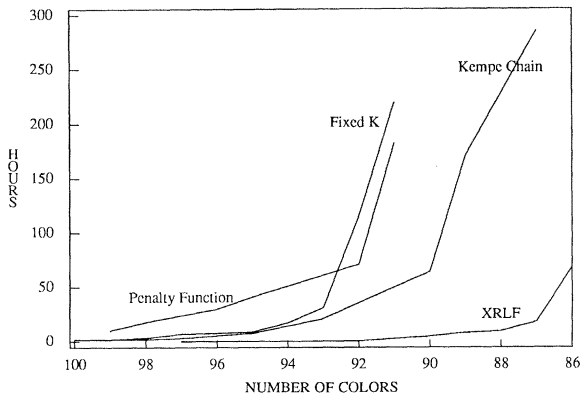

Figure 8. Tradeoffs between time and colors for a $G_{1,000,0.5}$ random graph.

letting *TRIALNUM* increase by factors of 2. (Note that increasing *EXACTLIM* from 0 to 70 does not always have a significant effect on running time because the residual graph on which CHROM_NUM( ) is called need not have the full 70 vertices.)

Since all these approaches involve randomization, they need not generate the same number of colors on every run, even when the parameter values are fixed. Since we were interested only in general trends, the results summarized in Table I for penalty function annealing, Kempe chain annealing, and XRLF are for the most part based on one or two runs for each parameter setting. We report the average running time, and give for the number of colors the smallest integer k such that k or fewer colors were used on more than half the runs. Here the one exception is marked by an asterisk: for penalty function annealing, the [2, 64] parameter setting yielded one 91-coloring and one 92-coloring. Since fixed-K annealing, unlike the other algorithms, does not produce legal colors when it fails, it is more important to know how likely it is to succeed for a given choice of parameters. Thus, we typically performed more runs for it. Table I indicates the fraction of successful runs for each listed parameter setting; the entry (a, b) specifies that b trials were performed, of which a resulted in legal colorings.

If the last entry in a column has a parenthesized running time, this indicates that the given coloring was *never* successfully constructed for any parameter choice, with the reported run being the longest attempted. In general, the entries in the table are for the parameter settings that generated the given colorings in the least amount of time. (We typically tested nearby values for TEMPFACTOR and SIZEFACTOR, although we did not study the parameter space exhaustively.) Note that with fixed-K annealing and a given fixed K, the results often passed through three phases as the parameters were changed to allow increased running time: until a certain

Table I
Running Times Required to Obtain Given Colorings for the $G_{1000,0.5}$ Random Graph of Figure 8[a]

	Penalty Function Annealing		Kempe Chain Annealing		Fixed-K Annealing			Successive Augmentation	
Colors	Hours	[*TF*, *SF*]	Hours	[*TF*, *SF*]	Hours	[*TF*, *SF*]	(Trials)	Hours	Algorithm
108	—	—	—	—	—	—	—	0.5	RLF[median]
105	—	—	—	—	—	—	—	17.9	RLF[best: 1%]
102	5.0	[1, 2]	—	—	—	—	—	—	—
100	—	—	1.4	[0.25, 0.1]	1.8	[1, 1]	(10/10)	—	—
99	10.2	[1, 4]	—	—	2.0	[1, 1]	(8/10)	—	—
98	18.0	[1, 8]	2.0	[0.5, 0.1]	3.7	[1, 2]	(7/7)	—	—
97	—	—	3.1	[1, 0.1]	4.3	[1, 2]	(10/16)	0.2	XRLF[1, 0]
96	30.0	[1, 16]	—	—	7.7	[1, 4]	(8/10)	—	—
95	41.3	[2, 16]	7.6	[1, 0.25]	9.0	[1, 4]	(4/10)	—	—
94	—	—	—	—	17.3	[1, 8]	(5/10)	—	—
93	—	—	21.2	[1, 0.5]	31.3	[1, 16]	(4/7)	0.5	XRLF[4, 0]
92	70.9	[2, 32]	35.7	[1, 1]	62.1	[2, 16]	(3/7)	0.6	XRLF[4, 70]
91	182.3*	[2, 64]	—	—	122.8	[2, 32]	(2/6)	—	—
90	(343.1)	[4, 64]	64.1	[2, 2]	(236.6)	[4, 32]	(0/1)	4.7	XRLF[40, 0]
89	—	—	170.8	[4, 4]	—	—	—	8.0	XRLF[20, 70]
88	—	—	—	—	—	—	—	9.6	XRLF[40, 70]
87	—	—	285.3	[8, 4]	—	—	—	18.3	XRLF[160, 70]
86	—	—	—	—	—	—	—	68.3	XRLF[640, 70]

[a]For algorithms that always yield legal colorings, the listed number of colors was attained more than half the time for the given parameter settings unless the time for the entry is marked by a *, in which case more details can be found in the text. For penalty function annealing, the (Trials) column gives the fraction of runs that resulted in legal colorings. A parenthesized running time indicates that the desired coloring was never found using the given parameter settings. See text for elaborations of these points and explanations of other shorthands used.

threshold is reached, no legal colorings are found. Near the threshold, an occasional legal coloring was found. Once past it, legal colorings were found on almost all runs. (This at least held true for the easier colorings.) The table entries correspond to the fastest parameter settings for the best success rate attained, where rates of 50% or greater are deemed equally good.

Another observation on our fixed-K results is that when raw machine speed is taken into account, our running times appear to be significantly faster than those reported by Chams, Hertz and de Werra, at least for the more difficult values of K. Although Chams et al. report 1.8 hours for a 98-coloring compared to our 3.7, the processor used by Chams et al. is a CDC Cyber 170-855, which should be four or more times faster than the Sequent Balance 21000 processor we used. (One possible explanation is that the cooling parameters by Chams et al. seem to have been optimized for high rather than low values of K.)

Before passing on to other graphs, we remark that for $G_{1,000,0.5}$ random graphs, XRLF can be improved if one modifies it to take advantage of the expected properties of such graphs. Bollobás and Thomason developed an alternative approximation to RLF* that uses estimates on tail probabilities to prevent their analogue of Step 4.3.1 from being entered except under the most promising conditions. With this algorithm, they were able to obtain colorings that averaged 86.9 colors over ten sample $G_{1,000,0.5}$ graphs, using less than one hour per run on an IBM 3081. This corresponds to 8–10 hours on our processor and hence is half the time it took us to get an 87-coloring for our graph. It is not clear what Bollobás and Thomason might have achieved if they had allowed as much running time as we did. For the record, our best results on $G_{1,000,0.5}$ graphs were obtained with $EXACTLIM = 70$ and $TRIALNUM = 1{,}260$. For these settings, the average run length was 136 hours, but we averaged 85.5 colors over a sample of four $G_{1,000,0.5}$ graphs. The graph in Table I was *not* one of those for which an 85-coloring was found. Perhaps coincidentally, it also had a slightly higher-than-expected density (0.5000152), whereas the graphs for which 85-colorings were found both had densities slightly less than the expected 0.50.

2.4.2. Random $p = 0.5$ Graphs With 500 and Fewer Vertices

The same sort of experiments that we performed on the $G_{1,000,0.5}$ random graph of the previous section were also performed on $G_{n,0.5}$ random graphs with $n = 125$, 250 and 500. As was the case for $n = 1{,}000$, we concentrated on just one sample of each graph. Our purpose was to spot trends rather than estimate the precise expected results for any particular choice of n and p. For the validity of the trends we observe, we rely on limited "confirmation" tests on other sample instances, and on past observations that experimental results for graphs of this type do not vary substantially from instance to instance.

Results are summarized in Table II, whose entries obey the same conventions as those for Table I. (For XRLF, if the value for *trialnum* is listed as "ex," this indicates that XRLF was run in the "exhaustive mode," i.e., with *SETLIM* set to the number of vertices and $TRIALNUM = CANDNUM = 1$.) For comparison purposes, we once again include the results for RLF (the best of the traditional heuristics on these graphs), giving both the median and best coloring found over 100 runs, and the times for 1 and 100 runs, respectively. To put the results in perspective, we also give for each graph both its computed density and the current best lower bound on the expected chromatic number for graphs of its type (e.g., $D = 0.5020$, $LB = 46$ for the case of $n = 500$). The lower bound, like that of Bollobás and Thomason for $G_{1,000,0.5}$, is determined by computing the smallest K for which the expected number of K-colorings exceeds 0.5. Typically, if L is this lower bound, the expected number of L-colorings is in fact something like 10^5, whereas the expected number of $(L-1)$-colorings is 10^{-10}. (The expected number of K-colorings for $G_{n,p}$ can be computed using standard counting arguments; we used a cleverly-optimized program for doing this provided by Thomason 1987.)

Note that for these smaller graphs, simulated annealing is a much stronger competitor. For the 125-vertex graph, both Kempe chain and fixed-K annealing succeed in finding a 17-coloring, whereas the best that XRLF can do, even with its parameters turned as high as they could feasibly go, is 18 colors. Moreover, 18-colorings could be found more quickly with the two annealing approaches than with XRLF. For the 250-vertex, XRLF *was* capable of finding the best coloring we saw, but only on 2 out of 5 runs, and the running times required by Kempe chain and fixed-K annealing for the best colorings are at least in the same ballpark. (It is interesting to note that we could actually perform the limiting algorithm RLF*, i.e., XRLF[ex, 0], for both the 250- and 125-vertex graphs, and in each case it required two colors more than the minimum found by other methods.)

For $n = 500$, the situation begins to look more like that in Table I for $n = 1{,}000$, in that for 50 or more colors, XRLF is substantially faster than any of the other approaches. Even here, however, despite our best efforts at increasing its running time, we were never able to get it to obtain a 49-coloring, which Kempe chain annealing found in 161.3 hours (on one out of two tries with the

Table II
Running Times Used to Obtain Given Colorings for $G_{n,0.5}$ Random Graphs, $n \leqslant 500$[a]

	Penalty Function Annealing		Kempe Chain Annealing		Fixed-K Annealing			Successive Augmentation	
Colors	Hours	[TF, SF]	Hours	[TF, SF]	Hours	[TF, SF]	(Trials)	Hours	Algorithm
				125 Vertex, $p = 0.5$ Random Graph ($D = 0.5021$, $LB = 16$)					
21	—	—	—	—	—	—	—	0.0	RLF[median]
20	—	—	—	—	—	—	—	0.2	RLF[best:37%]
19	0.2	[1, 1]	0.0	[0.5, 0.5]	0.0	[1, 1]	(8/10)	0.0	XRLF[ex, 0]
18	1.7	[1, 16]	0.2	[1, 2]	0.1	[1, 4]	(7/10)	0.5	XRLF[80, 65]
17	(24.1)	[2, 128]	21.6	[16, 64]	1.8	[1, 64]	(2/8)	(6.4)	XRLF[ex, 75]
				250-Vertex, $p = 0.5$ Random Graph ($D = 0.5034$, $LB = 27$)					
35	—	—	—	—	—	—	—	0.0	RLF[median]
33	—	—	—	—	—	—	—	1.2	RLF[best:2%]
31	1.5	[1, 4]	0.1	[0.5, 0.25]	0.2	[1, 2]	(7/10)	0.1	XRLF[ex, 0]
30	2.5	[1, 8]	0.8	[1, 1]	0.9	[1, 8]	(6/10)	1.3	XRLF[160, 0]
29	14.4	[2, 32]	6.2	[4, 2]	6.4	[2, 32]	(5/10)	2.2*	XRLF[160, 65]
				500-Vertex, $p = 0.5$ Random Graph ($D = 0.5020$, $LB = 46$)					
60	—	—	—	—	—	—	—	0.1	RLF[median]
59	—	—	—	—	—	—	—	7.5	RLF[best:7%]
55	3.7	[1, 4]	—	—	—	—	—	0.1	XRLF[1, 0]
54	—	—	1.5	[0.5, 0.5]	1.1	[1, 2]	(5/10)	0.1	XRLF[2, 0]
53	8.4	[1, 8]	2.2	[1, 0.5]	2.1	[1, 4]	(5/10)	0.2	XRLF[4, 0]
52	—2	—	10.6	[1, 2]	8.4	[1, 16]	(5/10)	0.3	XRLF[8, 0]
51	42.2	[2, 32]	16.9	[2, 2]	28.0	[4, 16]	(3/14)	4.5	XRLF[160, 0]
50	136.8	[2, 128]	45.2	[4, 8]	(212.4)	[4, 128]	(0/1)	9.8	XRLF[320, 65]
49	—	—	161.3*	[4, 16]	—	—	—	(73.8)	XRLF[2560, 70]

[a]The notational shorthands of Table I continue to apply here. An ex under XRLF means that the set-finding in the algorithms as performed in exhaustive search mode (see text), D stands for the actual edge density, and LB for the lower bound on expected chromatic number described in the text.

given parameters). Indeed, without the final exact coloring phase, we never got XRLF to use fewer than 52 colors, even when run in the mode where each color class was constructed by exhaustive search, subject to the constraint that it contain the current maximum degree vertex.

Finally, observe that for all three graphs, fixed-K annealing is a much stronger rival to Kempe chain annealing than it was for $n = 1{,}000$, although on the 500-vertex graph it weakens considerably once one drops below 52 colors, and is surpassed even by the penalty function approach at 50 colors, mirroring its decline on the larger graph.

2.4.3. Graphs With Unexpectedly Good Colorings

In this section, we consider the ability of the various graph coloring heuristics to find unexpectedly good colorings. We generated graphs that superficially looked like $G_{n,0.5}$ random graphs, but in fact had colorings that used only about half the number of colors found in the experiments of the previous section.

In particular, having chosen a chromatic number K and a number of vertices n, we generated our graphs as follows:

1. Randomly assign vertices with equal probability to K color classes.
2. For each pair $\{u, v\}$ of vertices not in the same color class, place an edge between u and v with probability $K/(2(K-1))$, i.e., the probability required to make the average degree roughly $n/2$.
3. Pick one vertex as a representative of each class and add any necessary edges to ensure that these K vertices form a clique (assuming that K is not too large, no class will be empty, so such representatives will exist).

Using this procedure, we generated "cooked" graphs to match the graphs of the previous two sections, and with chromatic numbers as indicated in Table III. The table also presents, for each of these graphs, the running times (per 100 runs) and the number of colors obtained by each of the standard heuristics of Section 2.1, and compares these results to those obtained for the more-truly-random counterparts of these graphs that were studied in Sections 2.4.1 and 2.4.2. Note that although a clever special purpose algorithm might be able to find

Table III
Performance of Traditional Heuristics on Standard and Cooked Graphs

Graph		100*SEQ			100*DSATUR			100*RLF		
$\|V\|$	$\chi(G)$	Median	Best	Time	Median	Best	Time	Median	Best	Time
125	~ 17	25	23	1.8 m	22	20	4.2 m	21	20	11.4 m
	9	23	19	1.8 m	17	10	4.1 m	12	10	11.2 m
250	~ 29	42	40	6.9 m	38	36	14.7 m	35	33	62.2 m
	15	41	38	6.9 m	36	32	14.6 m	26	23	62.1 m
500	~ 49	73	70	27.0 m	66	63	55.0 m	60	59	7.5 h
	25	72	69	27.1 m	65	61	54.9 m	56	47	7.1 h
1000	~ 85	127	124	1.8 h	117	114	3.6 h	108	106	52.2 h
	45	126	123	1.8 h	116	113	3.5 h	106	102	51.2 h

the hidden colorings by identifying the vertices of the constructed clique (which should have slightly higher-than-normal degrees), none of the standard heuristics succeeds (even with 100 tries) on any one of the four graphs. Indeed, for the larger graphs, the heuristics use a number of colors on the cooked graphs that is only slightly better than that which they use for the corresponding standard graphs, despite the large difference in the true chromatic numbers.

The optimal number of colors *can*, however, be found by each of the four approaches we have been considering, given enough time. Table IV indicates approximately how much time that is for each approach. For each approach, the time given in the table was sufficient to find an optimal coloring for the corresponding graph, and half that time did not suffice. (For fixed-K annealing, the given settings yield legal colorings at least 90% of the time for all four graphs.) Note that all three annealing approaches are faster than XRLF on all but the 125-vertex graph, and fixed-K annealing is the fastest by far of the three. This latter observation must be taken with several grains of salt, however, given that the times quoted are for runs where K is already fixed at its optimal value, i.e., with advance knowledge of the very secret we are trying to discover. If one had thought that these graphs were the $G_{n,0.5}$ graphs they mimic, we would never have thought to run the fixed-K approach with such small value of K, although we might well have chosen the parameter settings needed for the other three approaches to find the hidden coloring.

Also, that penalty function annealing seems to be competitive with Kempe chain annealing for these graphs, whereas it lagged far behind for the uncooked examples. This is most likely because the ultimate color class size for the cooked graphs is roughly twice what it was for the originals. (Recall that our implementation of Kempe chain annealing can take time proportional to the square of the largest color class to generate a move, whereas our penalty function implementation takes time only linear in that size.)

2.4.4. Random $p = 0.1$ and $p = 0.9$ Graphs

In this section, we consider random $G_{n,p}$ graphs with values of p different from the $p = 0.5$ or previous sections, to determine the effect of increased and decreased edge density on our comparisons. Table V summarizes the results of experiments with $G_{n,0.1}$ graphs for our standard values of n. For these graphs, certain changes had to be made in the standard parameter choices for some of the algorithms. First, the sparseness of the graphs meant that color classes would be much larger, and so we could no longer afford to run XRLF with as

Table IV
Times Required by the Three Annealing Approaches and XRLF to Find Optimal Colorings of the Four Cooked Graphs[a]

Graph		Penalty Function Annealing		Kempe Chain Annealing		Fixed-K Annealing		XRLF	
$\|V\|$	$\chi(G)$	Time	[*TF*, *SF*]	Time	[*TF*, *SF*]	Time	[*TF*, *SF*]	Time	[TN, XL]
125	9	34.1 s	[0.5, 0.1]	47.8 s	[0.5, 0.1]	16.3 s	[0.5, 0.5]	34.2 s	[4, 0]*
250	15	4.6 m	[0.5, 0.5]	4.3 m	[0.5, 0.1]	1.4 m	[0.5, 0.5]	12.5 m	[5, 0]
500	25	35.1 m	[1, 1]	18.0 m	[1, 0.1]	8.9 m	[1, 0.5]	107.8 m	[20, 0]
1,000	45	15.2 h	[2, 8]	14.3 h	[1, 1]	3.7 h	[2, 1]	64.1 h	[640, 0]

[a]Here the parameter settings for XRLF on the 125-vertex graph are marked by a '*' to indicate that we did not use the standard values of *SETLIM* = 63 and *CANDNUM* = 50 on this graph, but instead set these parameters to 32 and 10, respectively.

Table V
Running Times Required to Obtain Given Colorings of $G_{n,0.1}$ Random Graphs

	Penalty Functional Annealing		Kempe Chain Annealing		Fixed-K Annealing			Successive Augmentation	
Colors	Hours	[TF, SF]	Hours	[TF, SF]	Hours	[TF, SF]	(Trials)	Hours	Algorithm
125-Vertex, $p = 0.1$ Random Graph ($D = 0.0950$, $LB = 5$)									
6	0.1	[0.5, 0.25]	0.0	[1, 0.5]	0.0	[0.5, 0.25]	(7/10)	0.0	RLF[med, best]
5	2.9	[2, 8]	4.8	[4, 16]	0.2	[1, 16]	(4/10)	0.0	XRLF[1, 125]
250-Vertex, $p = 0.1$ Random Graph ($D = 0.1034$, $LB = 7$)									
10	—	—	—	—	—	—	—	0.0	RLF[median]
9	0.2	[1, 0.5]	0.5	[1, 1]	0.0	[1, 0.5]	(9/10)	1.8	RLF[best:45%]
8	(36.9)	[4, 16]	(25.6)	[4, 16]	2.6	[4, 16]	(5/10)	(40.3)	XRLF[1280, 125]
500-Vertex, $p = 0.1$ Random Graph ($D = 0.0999$, $LB = 11$)									
15	—	—	—	—	—	—	—	0.1	RLF[median]
14	2.0	[1, 1]	1.6	[0.5, 0.5]	0.1	[1, 0.5]	(10/10)	11.8	RLF[best:18%]
13	24.2	[2, 16]	68.6	[2, 16]	1.0	[2, 2]	(8/10)	16.5	XRLF[320, 100]
1,000-Vertex, $p = 0.1$ Random Graph ($D = 0.0994$, $LB = 19$)									
24	3.2	[1, 0.5]	7.0	[0.5, 0.5]	0.3	[0.5, 0.5]	(10/10)	0.8	RLF[med, best]
23	13.0	[1, 2]	21.0	[1, 1]	0.6	[1, 0.5]	(6/6)	1.2	XRLF[5, 100]
22	37.0	[2, 4]	124.6	[1, 8]	4.1	[2, 2]	(4/4)	35.1	XRLF[160, 100]
21	(101.0)	[2, 16]	(281.9)	[1, 16]	36.1	[2, 16]	(2/2)	(137.0)	XRLF[640, 100]

large a value as 63 for *SETLIM*, settling instead for *SETLIM* = 20. We also discovered that we had to increase the starting temperature for penalty function annealing from 10 to 30, and for Kempe chain annealing from 5 to 10, in order for the initial acceptance ratio to reach the 30% level. (The starting temperature of 2 remained sufficient for fixed-K annealing.)

Note that here, with even bigger color classes, Kempe chain annealing falls behind penalty function annealing. Moreover, as with the graphs of the previous section, both are dominated by fixed-K annealing, as is XRLF.

Table VI summarizes the results of experiments with $G_{n,0.9}$ graphs for our standard values of n. We only consider two of the three annealing approaches in detail here. Given the trends in running times indicated by our results for $p = 0.1$ and $p = 0.5$, it seemed highly unlikely that penalty function annealing would be competitive with Kempe chain annealing when $p = 0.9$. (As a test case, we ran both on the $G_{500,0.9}$ graph. The penalty function approach required 27 hours to find a 132-coloring, whereas Kempe chain annealing found a 131-coloring in just 3.6 hours.) As in the $p = 0.5$ case, we used starting temperatures of 5.0 and 2.0 for Kempe chain and fixed-K annealing, respectively.

For these graphs it is possible to run XRLF in the exhaustive mode even for $n = 1{,}000$, although this may not always be the best choice. (In particular, a 232-coloring of the 1,000-vertex graph was obtained more quickly with *SETLIM* < 1,000, as indicated in the table.) Once again, both annealing and XRLF can find substantially better colorings than traditional successive augmentation heuristics like RLF and DSATUR (and the former again dominates the latter). In contrast to the $G_{n,0.5}$ graphs, however, Kempe chain annealing substantially outperforms both XRLF and fixed-K annealing on all the graphs with $n \geqslant 250$. Fixed-K annealing outperforms XRLF on all graphs with $n \leqslant 500$, but XRLF seems to have caught it by $n = 1{,}000$.

2.4.5. Geometrically Defined Graphs

In this, our final section of results, we consider how the various heuristics behave on graphs in which there is built-in structure (but not built-in colorings). In particular, we consider the "geometrical" graphs of Part I, and their complements. A random geometrical graph $U_{n,d}$ is generated as follows. First, pick $2n$ independent numbers uniformly from the interval (0, 1), and view these as the coordinates of n points in the unit square. These points represent vertices, and we place an edge between two vertices if and only if their (Euclidean) distance is d or less. Table VII summarizes our results for three examples of such graphs, all with $n = 500$: a $U_{500,0.1}$ graph, a $U_{500,0.5}$ graph, and the complement of a $U_{500,0.1}$ graph (denoted $\overline{U}_{500,0.1}$). The densities of these graphs were 0.0285, 0.4718 and 0.9721, respectively. (Experiments with second examples of each type of graph, having slightly different densities, yield qualitatively similar results.)

Again, we drop penalty function annealing from the comparison, and use starting temperatures of 5.0 and 2.0 for Kempe chain and fixed-K annealing, respectively. The story is once again mixed. Although fixed-K annealing is the winner for the $d = 0.1$ graph, it is outperformed by Kempe chain annealing for $d = 0.5$, and what

Table VI
Running Times Needed to Obtain Given Colorings for $G_{n,0.9}$ Random Graphs[a]

	Kempe Chain Annealing		Fixed-K Annealing			Successive Augmentation	
	Hours	[TF, SF]	Hours	[*TF*, *SF*]	(Trials)	Hours	Algorithm
			125-Vertex, $p = 0.9$ Random Graph ($D = 0.8982$, $LB = 40$)				
50	—	—	—	—	—	0.0	RLF[median]
48	—	—	—	—	—	0.1	RLF[best:48%]
45	0.1	[0.25, 0.1]	0.0	[1, 1]	(8/10)	0.0	XRLF[ex, 0]
44	6.6	[2, 8]	0.3	[1, 8]	(7/10)	1.7	XRLF[ex, 80]
43	(46.9)	[8, 16]	(9.3)	[8, 64]	(0/3)	—	—
			250-Vertex, $p = 0.9$ Random Graph ($D = 0.8963$, $LB = 70$)				
84	—	—	—	—	—	0.0	RLF[median]
82	—	—	—	—	—	0.5	RLF[best:11%]
76	—	—	—	—	—	0.0	XRLF[ex, 60]
75	0.3	[0.25, 0.1]	0.6	[2, 2]	(8/10)	(184.7)	XRLF[ex, 80]
74	—	—	1.3	[2, 4]	(5/10)	—	—
73	0.8	[0.5, 0.25]	5.0	[2, 16]	(3/4)	—	—
72	20.0	[2, 4]	(48.5)	[8, 64]	(0/1)	—	
71	(70.8)	[2, 16]	—	—	—	—	—
			500-Vertex, $p = 0.9$ Random Graph ($D = 0.9013$, $LB = 122$)				
155	—	—	—	—	—	0.0	RLF[median]
152	—	—	—	—	—	3.2	RLF[best:6%]
134	1.1	[0.25, 0.1]	2.7	[2, 2]	(7/10)	0.3	XRLF[ex, 0]
133	—	—	4.8	[2, 4]	(8/8)	0.3	XRLF[ex, 60]
132	—	—	9.6	[4, 4]	(2/3)	1.5	XRLF[ex, 70]
131	3.6	[0.5, 0.25]	9.6	[4, 4]	(3/3)	(77.8)	XRLF[ex, 80]
130	6.5	[1, 0.25]	21.6	[2, 16]	(2/6)	—	—
129	15.2	1, 0.5]	(80.2)	[8, 16]	(0/1)	—	—
128	29.1	[1, 1]	—	—	—	—	
			1,000-Vertex, $p = 0.9$ Random Graph ($D = 0.8998$, $LB = 217$)				
283	—	—	—	—	—	0.2	RLF[median]
276	—	—	—	—	—	21.5	RLF[best:1%]
238	10.0	[0.5, 0.1]	38.7	[2, 8]	(3/3)	—	—
237	—	—	—	—	—	—	
236	—	—	78.7	[2, 16]	(4/4)	—	—
235	—	—	88.9	[2, 16]	(1/7)	0.2	XRLF[5, 0]*
234	—	—	84.6	[2, 16]	(1/4)	3.0	XRLF[ex, 0]
233	36.3	[0.5, 0.2]	(172.5)	[4, 16]	(0/2)	4.9	XRLF[ex, 60]
232	—	—	—	—	—	11.0	XRLF[20, 70]*
231	—	—	—	—	—	(277.5)	XRLF[ex, 80]
230	44.7	[1, 0.25]	—	—	—	—	—
229	—	—	—	—	—	—	—
228	122.4	[1, 1]	—	—	—	—	—
227	—	—	—	—	—	—	—
226	350 +	[2, 2]	—	—	—	—	—

[a]The 232- and 235-colorings of the 1,000-vertex graph using XRLF were obtained with *SETLIM* = 250, *CANDNUM* = 50, and *TRIALNUM* as specified. The Kempe chain run that found the 226-coloring was terminated by a computer crash rather than by convergence. If the run had survived until convergence, it might have taken much more time than the 350 hour actually used. (It might also have found a better coloring.)

is more surprising, both are beaten substantially by DSATUR, a successive augmentation heuristic that finished well behind RLF on all our nongeometric graphs. Performing 100 runs of this heuristic took only 1.3 hours and the best coloring found was 3 colors better than we could find in 50 or more hours using either annealing technique or XRLF. Indeed, one in five runs of DSATUR uses fewer colors than any of the other techniques. (The percentages we quote here for DSATUR are based on a set of 1,000 runs, rather than the standard 100, and so should be fairly robust for this graph.)

Our final graph, the complement of a $d = 0.1$ graph, also shows some anomalies. Here DSATUR again outperforms XRLF, but is itself beaten by ordinary RLF.

Table VII
Running Times Needed to Obtain Given Colorings for Various Geometric Graphs[a]

	Kempe Chain Annealing		Fixed-K Annealing			XRLF		Successive Augmentation	
Colors	Hours	[*TF*, *SF*]	Hours	[*TF*, *SF*]	(Trials)	Hours	[*TN*, *EL*]	Hours	Algorithm
				500-Vertex, $d = 0.1$ Random Geometric Graph					
13	—	—	—	—	—	0.0	[1, 0]*	0.0	DSAT[med.]
12	0.8	[0.25, 0.25]	0.0	[0.5, 0.5]	(10/10)	(7.1)	[50, 0]*	1.2	DSAT[best:29%]
				500-Vertex, $d = 0.5$ Random Geometric Graph					
132	—	—	1.4	[1, 2]	(5/10)	—	—	0.8	RLF[med.]
131	—	—	4.2	[2, 4]	(6/10)	—	—	7.5	RLF[best:1%]
130	3.8	[1, 0.5]	8.1	[2, 8]	(5/10)	0.0	[1, 0]*	—	—
129	7.4	[1, 1]	17.8	[2, 16]	(1/6)	—	—	0.0	DSAT[med.]
128	20.7*	[2, 2]	32.4	[4, 16]	(2/10)	—	—	—	—
127	72.5*	[2, 8]	(113.3)	[4, 64]	(0/2)	2.3	[1, 0]*	—	—
126	(126.5)	[2, 16]	—	—	—	(59.8)	[20, 0]*	—	—
125	—	—	—	—	—	—	—	—	—
124	—	—	—	—	—	—	—	1.3	DSAT[best:1%]
				Complement of a 500-Vertex, $d = 0.1$ Random Geometric Graph					
95	—	—	—	—	—	0.1	[ex, 180]	—	—
94	—	—	—	—	—	0.3	[ex, 210]	—	—
93	—	—	—	—	—	8.7	[ex, 270]	0.0	DSAT[med.]
92	—	—	—	—	—	0.5	[ex, 280]	—	—
91	—	—	—	—	—	18.8	[ex, 290]	—	—
90	4.0	[0.5, 0.5]	—	—	—	1.9	[ex, 300]	0.0	RLF[med.]
89	5.0	[1, 0.5]	—	—	—	(100 +)	[ex, 315]	1.3	DSAT[best:4%]
88	12.3	[1, 1]	—	—	—	—	—	1.5	RLF[best:4%]
87	(239.8)	[2, 16]	0.0	[0.5, 0.5]	(10/10)	—	—		
86	—	—	0.0	[1, 1]	(10/10)	—	—	—	—
85	—	—	0.0	[2, 2]	(10/10)	—	—		
84	—	—	(75.3)	[4, 64]	(0/1)	—	—		

[a]The first five entries in the XRLF column are marked by "*"s because the parameters *SETLIM* and *CANDNUM* had to be varied to obtain the best results, although our format only allows us to specify *TRIALNUM* and *EXHAUSTLIM*. These entries were derived using the following (*SETLIM*, *TRIALNUM*, *CANDNUM*) combinations: (20, 1, 50), (30, 40, 50), (63, 1, 1), (250, 1, 1), and (250, 20, 50). The final XRLF run for the third graph had not yet terminated after 100 hours, at which point it was killed.

The annealing implementations reassert themselves, however, with the fixed-K approach again coming out on top. Once the correct values for *SIZEFACTOR* and *TEMPFACTOR* were chosen, it took only 30 seconds per run for 85-colorings. (Much more time was spent finding the correct parameter values, however. When we tried to 85-color the graph with *SIZEFACTOR* set to 1 instead of 2, no legal coloring was found and a typical run took an hour or more.) Another running time anomaly is evident from the results for XRLF. Here, because of the density of the graph, we could exhaustively search for the best independent set at each step, and switch over to exhaustive coloring when the number of uncolored vertices was still quite large. Our running times, however, do not monotonically increase with *EXACTLIM*, the parameter controling the switchover, but instead gyrate wildly. (Most likely, this is due to the wide variance in the running times of our exhaustive coloring routine, as seen in Figure 6.)

2.5. Commentary

The experiments we report in Section 2.4 do not allow us to identify a best graph coloring heuristic. Indeed, they reinforce the notion that there is no best heuristic. Table VIII displays the winners for each of the graphs we studied (except the cooked graphs, for which all three annealing algorithms were in the same ballpark, and pulled away from XRLF as soon as n reached 250). For each graph, we name the heuristic with the best performance, along with a runner-up if the competition is close. In judging performance for a given graph, we rank the algorithms first according to the best coloring they found. If this is a tie, we then consider the time the algorithms took to find this best coloring, penalizing

Table VIII
Algorithms Providing the Best Performance for Each of the Random Graphs $G_{n,p}$ and Geometric Graphs $U_{n,d}$ Covered in Our Study[a]

Graph Type	Number of Vertices			
	125	250	500	1,000
$G_{n,0.1}$	XRLF	Fixed!	Fixed!	Fixed!
$G_{n,0.5}$	Fixed, Kempe	XRLF, Kempe	Kempe, XRLF	XRLF!
$G_{n,0.9}$	Fixed	Kempe!	Kempe!	Kempe!
$U_{n,0.1}$	—	—	Fixed, DSAT	—
$U_{n,0.5}$	—	—	DSAT!	—
$\overline{U}_{n,0.1}$	—	—	Fixed!	—

[a]Close runners up are also listed. Runaway winners are annotated with an exclamation point.

fixed-K annealing by a (somewhat arbitrary) factor of 3 to account for the extra overhead it must incur in choosing K. If the contest is considered a runaway, we add an exclamation point. Note that the balance tends to shift from fixed-K annealing to Kempe chain annealing as the graphs get denser (although this effect is masked in the cases where XRLF and DSAT win).

At present, we have only tentative explanations of why a given approach dominates one class of graphs and not another. One factor no doubt is the question of whether the data structures of our implementation are optimized for sparse or dense graphs. (Recall that our Kempe chain implementation is most efficient when the color classes are small, which is likely to happen with dense graphs.)

Another important factor may be the "nature" of the good colorings for the graphs in question. Penalty function and Kempe chain annealing both use a cost function that rewards colorings in which the color class sizes are skewed: better a large and a small class than two equal sized ones. (XRLF has the same bias, given the greedy way in which it operates.) Fixed-K annealing, on the other hand, is neutral as to the sizes of the color classes it constructs, and so might be expected to construct more balanced colorings. Thus, for graphs in which good colorings of the latter type predominate, the fixed-K approach may well outperform the other two methods. In particular, this appears to have been the case with the final graph we studied, the complement of a $U_{n,0.1}$ random geometric graph. Here, the colorings that had the best costs under the Kempe chain formulation were definitely not the best colorings (94-colorings were found that cost less than 88-colorings). Consequently, even though this was a very dense graph on which we might have expected Kempe chain annealing to excel, fixed-K annealing could in seconds find colorings that were substantially better than anything seen by the Kempe chain algorithm in hundreds of hours.

In considering which of the new algorithms is best in which situation, we should not lose sight of the more fundamental implication of our results: as a class, these new randomized search algorithms (including XRLF) offer the potential for substantial improvement over traditional successive augmentation heuristics. When sufficient running time is available, they are usually to be preferred over the option of performing multiple iterations of a traditional heuristic, with the advantage increasing as more running time becomes available. Moreover, the running times of 100 hours and more that characterize the extremes of our experiments are not normally necessary if all one wants to do is outperform the traditional heuristics. On our instance of $G_{1,000,0.5}$, XRLF took only 10 minutes on a slow computer to improve by 8 colors over the best solution we ever found using traditional heuristics.

The approaches we study here of course do not exhaust the possibilities for computationally intensive randomized search. For instance, there is the "tabu" search technique of Glover (1989), which has been applied to graph coloring by Hertz and de Werra (1987). As with simulated annealing, this is a randomized modification of local optimization that allows uphill moves. Here, however, the basic principle is closer to that used in the Kernighan and Lin (1970) graph partitioning heuristic and the Lin and Kernighan (1973) traveling salesman heuristic, studied in Parts I and III of this paper, respectively. Given a solution, one randomly samples r neighbors, and moves to the best one, even if that means going uphill, unless that move is on the current "tabu" list. In the implementation of Hertz and de Werra, which is based on the fixed-K neighborhood structure, a move that changes the color of vertex v from i to j is considered tabu if v was colored j at any time during the last 7 moves. No tabu move can be made except in the following situation: The current cost is c, the new cost would be $c' < c$, and at no time in the past has a

move been made that improved a solution of cost c to one of cost as good as c'.

According to Hertz and de Werra, this technique (augmented with special purpose routines that may be of some use in jumping to a legal coloring at the end of the process) outperforms the original fixed-K annealing implementation of Chams, Hertz and de Werra. In our own limited experiments with tabu search, we have seen some speed-up on small instances and for easy colorings, but no general dominance. (This may be because we failed to tune the tabu parameters properly, or it may be because, as indicated in Section 2.4.1, our fixed-K annealing implementation seems to be significantly faster than that of Chams, Hertz and de Werra.) There is clearly much room for further investigation, both with these algorithms and alternatives, such as the hybrids suggested in Chams, Hertz and de Werra (1987) and Hertz and de Werra (1987), or entirely new annealing implementations. (One such new implementation has been proposed in Morgenstern (1989), with promising results: For some $G_{1,000,0.5}$ random graphs it finds 84-colorings.)

A final issue to be discussed here is the appropriate methodology for organizing the multiple runs that seem necessary if one is to get the best results possible for a given new graph from a given algorithm in a given amount of time. For penalty function annealing, Kempe chain annealing, and XRLF, one would presumably start with a short run and then adjust the parameters on each successive run so as to double the running time until no further improvement occurs or the available time is used up. Assuming that the last run provides the best results, only about half the overall time will have been wasted on preliminary work. This was essentially the procedure used here, although we have not fully investigated the question of *which* parameters to adjust when there are choices, and it sometimes seemed to make a difference. (As we mentioned, this is especially the case with XRLF.) For fixed-K annealing, a similar approach can be taken, only now one must also decide when and how far to decrease or increase K (which is why we imposed a factor-of-3 run-time penalty on fixed-K annealing when ranking the algorithms for Table VIII). One possibility is to start with a high value of K and a short running time. Thereafter, if the run is successful, try again with the same run-time parameters and reduce K by 1; if not, try again with K fixed and the running time doubled. Under this methodology, significantly more than half the time may be spent on preliminary runs, but the time spent on such runs should still be manageable.

Our experiments also raise questions about the methodologies used for starting and terminating runs; we shall have more to say about these generic issues in Section 4.

3. NUMBER PARTITIONING

In this section, we consider the application of simulated annealing to the number partitioning problem described in the Introduction. For an instance $A = (a_1, a_2, \ldots, a_n)$ of this problem, a feasible solution is a partition of A, i.e., a pair of disjoint sets A_1 and A_2 whose union is all of A. In contrast to the situation with graph partitioning in Part I, there is no requirement that the cardinality of the sets be equal; *all* partitions are feasible solutions. The *cost* of such a partition is $|\sum_{a\in A_1} a - \sum_{a\in A_2} a|$, and the goal is to find a partition of minimum cost.

We make no claims about the practical significance of this NP-hard problem, although perfect solutions (ones with cost 0) might have code-breaking implications (Shamir 1979). We have chosen it mainly for the extremely wide range of locally optimal solution values that its instances can have (as measured in terms of the ratio between the bset and the worst: see below), and because of the challenges it presents to simulated annealing.

The major challenge is that of devising a suitable and effective neighborhood structure. We shall argue that the natural analogs and generalizations of the structures for graph partitioning and graph coloring have serious limitations, and then show experimentally that the simulated annealing procedure does not have enough power to overcome these drawbacks. This does not imply that there is *no* way of successfully adapting simulated annealing to this problem, but at present we can think of no better alternatives than the ones we consider.

3.1. Neighborhood Structures

The "natural" neighborhood structures referred to above form a series, $SW_1, SW_2, \ldots$. In the neighborhood graph SW_k, there is an edge between solutions (A_1, A_2) and (B_1, B_2) if and only if A_1 can be obtained from A_2 by "swapping" k or fewer elements, i.e., $|A_1 - B_1| + |B_1 - A_1| \leqslant k$. We shall refer to SW_k as the *k-swap neighborhood*. (Our annealing implementation for graph partitioning in Part I extends the definition of solution to include all partitions and then uses the 1-swap neighborhood graph.)

The limitations of these neighborhoods are illustrated (and emphasized) when we consider instances consisting of random numbers drawn independently from a uniform distribution over [0, 1]. Let I_n be the random variable representing an n-element instance of this type and $OPT(I_n)$ represents the optimum solution value for I_n.

Karmarkar et al. (1986) have shown that the expected value of the optimal solution value $OPT(I_n)$ is $O(\sqrt{n}/2^n)$, i.e., exponentially small. In contrast, the expected value of the smallest cost difference between neighboring solutions under neighborhood SW_k is about $1/n^k$, only polynomially small. Thus, any reasonable solution will be a local optimum of the neighborhood structure and will be buried in a deep "valley" of the solution space, i.e., all its neighbors will have values that are worse by very high multiplicative factors. Moreover, the most frequent such local optima will themselves be worse than the best by very high multiplicative factors. Thus, a local optimization algorithm, if started from a random partition, is almost certain to stop at a relatively bad solution.

Can simulated annealing do any better? Given the fact that annealing allows occasional uphill moves and runs for a long time, we would expect a typical annealing run to visit many distinct local optima, and so the best solution it sees should most likely be better than the average solution found by local optimization. But would it be the case (as it was for graph partitioning) that this best is better than what could be obtained by simply spending the same amount of time performing multiple runs of local optimization from random starts? The "mountainous" nature of the solution space raises doubts.

Moreover, it will not be enough merely to improve slightly on local optimization. This is because there exists an efficient algorithm, not based on local optimization or neighborhood structures at all, that should, at least asymptotically, outperform any local optimization algorithm based on a neighborhood SW_k for some fixed k. This is the "differencing" algorithm of Karmarkar and Karp.

3.2. The Competition

The differencing algorithm runs in $O(n \log n)$ time. It works by creating a tree structure with the elements of A as vertices, and then forming a partition by 2-coloring the tree and letting A_i be the set of elements with color i for $i \in \{1, 2\}$. (Such a coloring is unique and constructible in linear time.) The tree is constructed as follows.

We begin with a vertex for each element, labeled by the value of that element and declared to be "live." We then repeatedly perform the following operations until there is but a single live vertex: 1) Find the two live vertices u and v with the largest labels (ties are broken arbitrarily), and assume $label(u) \geq label(v)$. 2) Add an edge between u and v, declare v to be "dead," and set $label(u) = label(u) - label(v)$. (This operation essentially makes the decision to put u and v on opposite sides of the partition, postponing for the time being the decision as to which sides those are to be.)

It is easy to prove inductively that at any point in the construction we will have constructed a forest in which each tree contains exactly one live vertex, and the label of that vertex is precisely the value of the partition induced by that tree (the difference between the sums of the two sets that we obtain by 2-coloring that tree). Thus, the value of the eventual partition formed is simply the label of the final live vertex.

For random instances of the type we have been discussing, the expected value of this final label is thought to be $O(1/n^{\log n})$, which is asymptotically smaller than the expected smallest move size for any of the neighborhoods SW_k. (The $O(1/n^{\log n})$ has not actually been proved for the differencing algorithm, but rather for a variant specially designed to simplify the probabilistic analysis (Karmarkar and Karp). There is no apparent reason, however, why this variant should be better than, or even as good as, the original.) Although $O(1/n^{\log n})$ is still far larger than the expected optimum, it offers formidable competition to other approaches, and simulated annealing would have to improve substantially on local optimization to be in the running. Can it do so?

3.3. Implementation Details

To investigate this question, we construct implementations based on both the 1-swap and 2-swap neighborhood structures. Note that, if all the annealing parameters of our generic algorithm in Figure 1 are fixed, the latter implementation will take much more time per temperature, as its neighborhood size is $n + n(n-1)/2 = (n^2 + n)/2$ versus simply n for the SW_1 neighborhood. The neighborhood size for SW_k, $k > 2$ would analogously have been $\Omega(n^k)$ and we abandon all those neighborhood structures as computationally infeasible.

Among the problem-specific subroutines used in our implementation only the INITIAL_SOLN( ) and NEXT_CHANGE( ) routines merit detailed discussion. (The others are determined by our choice of neighborhood structure and the details of the problem itself.) For our initial solution, we pick a random partition by independently "flipping a fair coin" for each a_i to decide the set to which it is assigned. To pick a random neighbor under SW_1, we simply choose a random element of A and move it from its current set (A_1 or A_2) to the other set. Rather than choose a new random element each time NEXT_CHANGE( ) is called, however, we initially choose a random permutation of A, and then at each call simply choose the next element in the permutation, until the permutation is used up. Every

$|A|$ moves we rescramble the permutation and start over. This approach was mentioned in Part I and was observed to yield a more efficient use of time. It also helps ensure that the annealing process will end up with a solution that is truly locally optimal (if there is an improving move, it must be encountered sometime in the next $2n$ trials). An analogous process is used for the SW_2 case, only now we work from a permutation of all 1- and 2-element subsets X of A.

Since we were mainly interested in deriving rough order-of-magnitude estimates of tradeoffs between running time and the quality of the annealing solutions found, we did not do extensive experiments to optimize the parameters of the generic algorithm, but merely adopt reasonable values based on the lessons learned from our experiments with graph partitioning in Part I. (We did find it necessary to modify the generic termination condition, however, due to the anomalous way that annealing behaves for this problem; see the next section.) In particular, we set $INITPROB = 0.5$ and $TEMPFACTOR = 0.9$, and adjust the length of time spent in the annealing process by varying $SIZEFACTOR$. (For these experiments we kept $CUTOFF = SIZEFACTOR$; i.e., cutoffs were not used.)

The remaining detail to be filled in is the method for selecting the starting temperature. As no one temperature seemed to work equally well for all n, we chose to use an adaptive method. To explain this, we should say a little bit about how our experiments were performed. For each instance and value of $SIZEFACTOR$ considered, we performed 10 annealing runs, all with the same starting temperature. This common starting temperature was based on the average uphill move encountered when calling NEXT_CHANGE( ) N times (where N was the neighborhood size) for each of 10 randomly chosen initial solutions produced by INIT_SOLN( ). The initial temperature was chosen so that the probability of accepting this average uphill move was $INITPROB = 0.5$. Such a technique is simpler but somewhat less accurate than the "trial run" technique used for selecting starting temperatures in Part I. It tends to result in higher initial temperatures, and hence, somewhat longer running times. Fortunately, our results did not depend on the fine details of the running times, as we shall see.

3.4. Experimental Results

All our experiments concern random instances of the type discussed in Section 2.1. In order that rounding effects not obscure the quality of the solutions generated by the Karmarkar-Karp algorithm, each input number was generated in multiprecision form, with 36 decimal digits to the right of the decimal point, and multiprecision arithmetic was used throughout.

Figure 9 presents a "time exposure" of an individual annealing run on a random 100-element instance, using the SW_2 neighborhood structure and $SIZEFACTOR = 16$. (The generic termination conditions were turned off and the time exposure was run until visual evidence indicated that "freezing" had set in.) The x-axis of the plot measures time, or more precisely, the number of calls to NEXT_CHANGE( ) divided by the neighborhood size $N = 5{,}050$. The y-axis gives the value of the objective function on a logarithmic scale. The points in the plot represent the current solution value, as sampled once every 5,050 steps. Also depicted is a horizontal line with the y-coordinate equal to the value of the solution found by the Karmarkar-Karp algorithm.

Note that although the best solution encountered was better than the Karmarkar-Karp solution, the value to which the process converged was substantially worse, and indeed was little better than the average value seen during the last quarter of the cooling schedule. This is in contrast to standard annealing time exposures like those for graph coloring in the previous section, where the process converges essentially to the best value seen. Since our implementation outputs the best solution seen rather than the last, this is not a fatal defect, although it does indicate that the neighborhood structure is having a rather striking effect on the annealing process.

These anomalies will also confuse our generic termination test, which was predicated on the assumption that "freezing" began when one stopped seeing improvement in the current "champion" solution. As suggested by Figure 9, the time at which the final champion

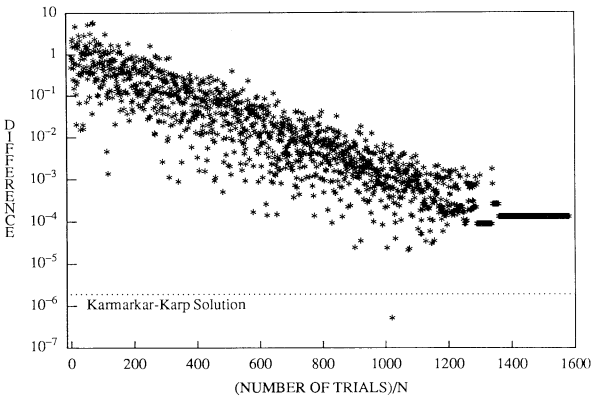

Figure 9. The evolution of the solution value for a random 100-element instance under simulated annealing with the 2-swap neighborhood structure, compared to the solution found using the Karmarkar-Karp algorithm (dotted line). (Note that one isolated annealing data point falls below the dotted line.)

appears may be a random phenomenon, only tangentially related to the convergence of the annealing process. Since for this study we were interested in the final frozen value as well as the champion, we modified the termination conditions in the experiments reported below as follows: To halt, we require that the acceptance ratio be less than $MINPERCENT = 1\%$ and that the solution value remain unchanged during each of the last 10 temperatures (or more precisely, that the values reported at the end of each of those temperatures all be the same).

With this change, we ran a suite of experiments on a 200-element random instance, and a 500-element random instance, summarized in Figures 10–13. Our first experiment concerned the 200-element instance and the 1-swap neighborhood. We performed 10 trials each with *SIZEFACTOR* taking on values equal to the powers of 2 running from 2 to 2,048. Figure 10 depicts the final and best solutions found for each run, marked by "o"s and "+"s, respectively, and plotted as a function of running time. For comparison purposes, the value of the solution found by the Karmarkar-Karp algorithm is once again represented by a horizontal dotted line.

In contrast to the behavior of annealing on graph partitioning and graph coloring, here the final annealed solution values do not appreciably improve as running time increases, but remain in the vicinity of the expected smallest move for this neighborhood, i.e., $1/200 = 0.005$, far above the Karmarkar-Karp solution. To be more precise, the average of $\log_{10}(\textit{difference})$ over all final solutions found (regardless of running time) is -2.27, whereas $\log_{10}(1/200) = -2.30$. Interestingly, the smallest possible move for this instance is uncharacteristically large, slightly bigger than 0.03, yielding

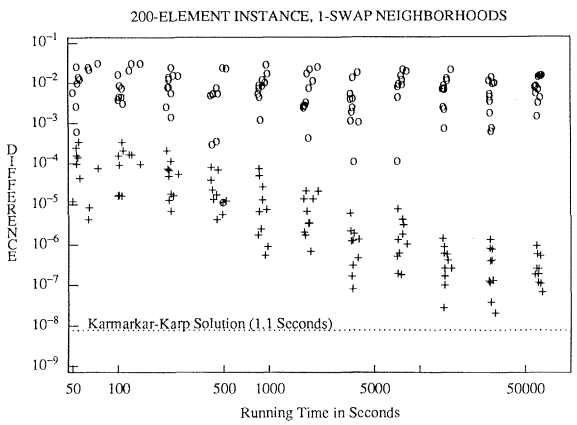

Figure 10. Final and best solutions (o's and +'s, respectively) found by 1-swap annealing for a random 200-element number partitioning instance, as a function of running time as SIZEFACTOR increases from 2 to 2,048.

$\log_{10}(0.03) = -1.52$, and 1-swap local optimization only has -1.80 as its average value for $\log_{10}(\textit{difference})$. Thus, although time spent on annealing in excess of 50 seconds seems wasted if one is interested only in final solutions, that first 50 seconds seems to have been worth something.

The *best* solutions tell a different story: these improved steadily with increased running times, approaching the solution value obtained by the Karmarkar-Karp algorithm. Note, however, that in this range we are spending well over 10,000 times the 1.1 seconds required by Karmarkar and Karp, which is enough time for over 100,000 runs of 1-swap local optimization (ignoring input time, which can be amortized, 10 such runs can be performed in a second). Figure 11 compares our annealing results with those obtained by spending an equivalent amount of time performing local optimization from random starts. For each value $J = 500, 1{,}000, \ldots, 512{,}000$, we perform 10 independent sets of J runs of local optimization, and plot the best solution in that subset versus the overall running time for the group (as estimated from our figure for the average time per run). These points, marked by 'o's, were then combined with the data points for annealing bests from Figure 10.

Note that across the board local optimization does just as well as annealing, if not better. Moreover, even if we could speed up our annealing implementation by a factor of 4, the resulting comparison (obtained by shifting the local optimization data points two steps to the right) would still be about equal. When we turn to the 2-swap neighborhood, or larger instances under the 1-swap

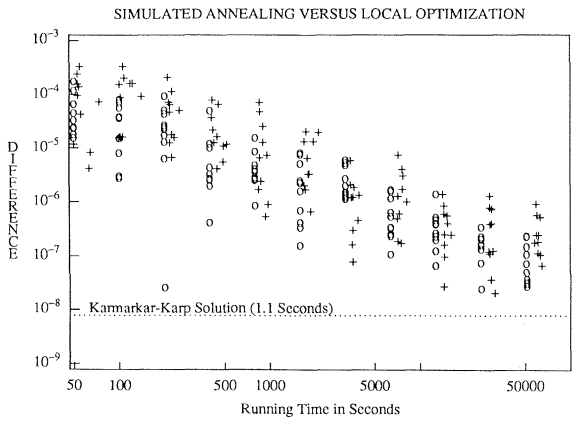

Figure 11. Comparison of best solutions found on annealing runs (+'s) with best solutions found by performing multiple starts of local optimization for an equivalent amount of time (o's). (Results are for the 200-element instance of Figure 10, with both algorithms using the 1-swap neighborhood.)

neighborhood, the comparison is no longer close, and multiple start local optimization substantially outperforms annealing, as can be seen in Figures 12 and 13.

Figure 12 shows the results for the 2-swap neighborhood and the 200-element instance of Figures 10 and 11. The best solutions found by annealing during 10 runs each for *SIZEFACTOR* = 0.25, 0.5, 1, 2, 4 and 8 are plotted, along with the best solutions found during 10 trials each of 150, 300, 600, 1,200, 2,400 and 4,800 local optimization runs. (For the 2-swap neighborhood, local optimization takes about 6.5 seconds per run, so 150 runs take roughly 1,000 seconds.) Many data points fall below the line representing the Karmarkar-Karp solution, although most of them come from local optimization. The annealing bests appear to be only slightly better on average than those obtained in equivalent time using the 1-swap neighborhood, as shown in Figure 10. (Although not depicted here, the final values found by annealing were again relatively independent of running time, and slightly better than those obtained by local optimization. Here the average of $\log_{10}(\textit{difference})$ for all annealing runs was -4.76, compared to -4.60 for local optimization. Both these values are substantially better than those we obtain using the 1-swap neighborhood structure, and presumably reflect the fact that much smaller moves are possible with the SW_2, on the order of $1/200^2$ instead of $1/200$; note that $\log_{10}(1/200^2) = -4.53$.)

Figure 13 shows the results for a 500-element instance and the 1-swap neighborhood. Here annealing was run with values of *SIZEFACTOR* going up by factors of 2

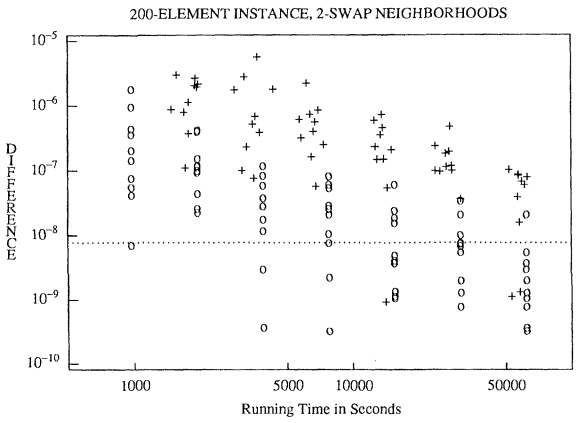

Figure 12. Comparison for the 2-swap neighborhood of best solutions found on annealing runs (+'s) with best solutions found by performing multiple starts of local optimization for an equivalent amount of time (o's). (Results are again for the 200-element random instance of Figure 10, and the dotted line represents the Karmarkar-Karp solution.)

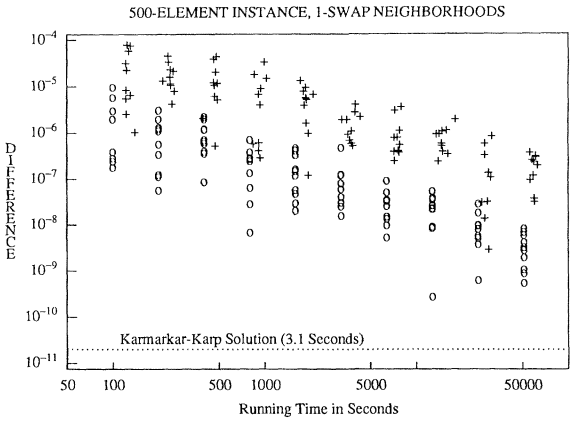

Figure 13. Comparison for the 1-swap neighborhood and a 500-element random instance of best solutions found on annealing runs (+'s) with best solutions found by performing multiple starts of local optimization for an equivalent amount of time (o's).

from 1–512. (The final values found were again relatively independent of running time, with the average of $\log_{10}(\textit{difference})$ for all annealing runs being -3.57, compared to -3.04 for local optimization and -2.7 for $\log_{10}(1/500)$. For this instance, the smallest possible move was substantially smaller than expected.)

Once again, local optimization substantially outperforms annealing on a time-equalized basis. Also, as expected, both annealing and local optimization are much further away from the Karmarkar-Karp solution value than they were in the 200-element case. This is true even if we allow for a linear increase in running time with instance size, as happens with annealing when we compare results for a fixed value of *SIZEFACTOR*. For example, when *SIZEFACTOR* = 512, the median annealing best is only 40 times larger than the Karmarkar-Karp solution when $n = 200$, whereas it is roughly 10,000 times larger when $n = 500$. For local optimization and equivalent running times, the corresponding ratios are roughly 40 and 150—still growing, but not quite so rapidly.

We performed limited experiments using the 2-swap neighborhood on the 500-element instance, but, within the 100,000 second time bound (approximately 30 hours) neither 2-swap annealing nor 2-swap local optimization did as well as our 1-swap results. (The quadratic growth rate of the 2-swap neighborhood seems to have begun to take its toll; one run of 2-swap local optimization takes 40 seconds on average, versus 0.25 seconds for 1-swap local optimization.) Thus, although we could approach and even surpass the performance of the Karmarkar-Karp algorithm when $n = 200$, given large but feasible

amounts of running time, our luck seems to have run out by the time $n = 500$.

We did not perform experiments for values of $n > 500$, but the trends are already obvious and conform to our expectations: Annealing (and local optimization, to a slightly lesser extent) will be even more substantially outclassed by Karmarkar Karp as n continues to increase. Note that typically values for $\log_{10}$ (*Karmarkar-Karp*) are less than: -13 for $n = 1{,}000$ (in 6.5 seconds), -16 for $n = 2{,}000$ (in 13.6 seconds), and -24 for $n = 10{,}000$ (in 75.8 seconds). Thus, number partitioning, at least for the types of random instances we have been considering, illustrates the limitations of simulated annealing as a general technique. When the solution space is sufficiently mountainous, annealing's advantage over straightforward multiple start local optimization can be lost entirely. Moreover, other approaches, not tied to the concept of navigating around a solution space, may be able to outperform it substantially.

There remains the question of whether some other neighborhood structure for the problem, perhaps using different notions of solution and cost, might prove more amenable to annealing. We do not rule out this possibility, although at present we see not reasonable alternatives. The natural idea of modifying the annealing implementation by replacing *difference* as the objective function by log(*difference*) appears to be of little help, based on limited experiments. We leave the investigation of additional possibilities to future researchers.

4. CONCLUSION

In this paper, we consider implementations of simulated annealing for two problems that had previously not been thought accessible to local optimization and its variants: graph coloring and number partitioning. Our graph coloring results, as summarized in Section 2.5, were generally positive for simulated annealing, assuming one can tolerate the large computation times involved. The results for number partitioning were, as expected, decidedly negative, with annealing substantially outperformed by the much faster Karmarkar-Karp algorithm, and even beaten (on a time-equalized basis) by multiple start local optimization.

This all fits in with the view expressed in Part I of this paper (Johnson et al. 1989), that annealing is a potentially valuable tool but in no ways a panacea. Part III (Johnson et al. 1990) will conclude this series with an exploration of how well simulated annealing does against the more traditional competition on perhaps the most famous combinatorial optimization problem of them all, and the one for which it was originally touted by Cerny (1985) and Kirkpatrick, Gelatt and Vecchi (1983): the traveling salesman problem.

In performing our final experiments for that paper shall take into account several lessons learned from the experiments reported here. First, due to difficulties we encountered, it has become clear that both our starting and terminating procedures need revision.

Our current termination tests ask whether *FREEZE_LIM* consecutive temperatures have occurred in which: a) the acceptance ratio was below *MINPERCENT*, and b) no improvement in the best solution seen has taken place. For several types of instances encountered, we had to make major changes in the termination parameters simply because an abundance of 0-cost moves kept the acceptance frequency high, even though no further improvement in cost was occurring. Thus, the termination condition should probably be altered so that only the rate at which *uphill* moves are accepted is relevant (a very simple modification).

We also found ourselves regularly having to choose starting temperatures in an ad hoc manner because the generic methods we had devised for this (using either trial runs or multiple calls to NEXT_CHANGE) were not sufficiently robust. We suspect that, for most problems, starting temperatures can be determined using simple problem-specific formulas (analytically or empirically derived) that depend only on the desired initial acceptance ratio and a few easily computable parameters of the instance. For instance, $|V|$ and $|E|$ might well suffice in the case of graph coloring. Thus, there is likely to be a problem-specific INITIAL_TEMP routine in our future implementations.

A final observation is that the running-time/quality-of-solution tradeoff inherent in most annealing implementations may well extend far beyond the standard limits of acceptable running time. In our graph coloring experiments, we saw positive results come out of runs that took a week or more of continuous computing. That this may be of more than academic interest follows from the rapid rate at which the price of computer cycles is declining. That compute-week could be almost free if it were spent on one of the idle personal computers that now decorate many offices, or it could be an overnight background run on one of the much faster machines becoming more widely available. For problems in which the economic value of finding improved solutions is substantial, this is a thought to keep in mind.

REFERENCES

AARTS, E. H. L., AND J. H. M. KORST. 1989. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, Chichester, U.K.

BERGER, B., AND J. ROMPEL. 1990. A Better Performance Guarantee for Approximate Graph Coloring. *Algorithmica* **5**, 459–466.

BOLLOBÁS, B., AND A. THOMASON. 1985. Random Graphs of Small Order. *Ann. Discrete Math.* **28**, 47–97.

BRÉLAZ, D. 1979. New Methods to Color Vertices of a Graph. *Comm. ACM* **22**, 251–256.

BROUWER, A. E., J. B. SHEARER, N. J. A. SLOANE AND W. D. SMITH. 1990. A New Table of Constant Weight Codes. *IEEE Trans. Inf. Theory* **36**, 1334–1380.

CERNY, V., 1985. A Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *J. Optimiz. Theory Appl.* **45**, 41–51.

CHAMS, M., A. HERTZ AND D. DE WERRA. 1987. Some Experiments With Simulated Annealing for Coloring Graphs. *Eur. J. Opns. Res.* **32**, 260–266.

COLLINS, N. E., R. W. EGLESE AND B. L. GOLDEN. 1988. Simulated Annealing: An Annotated Bibliography. *Am. J. Math. Mgmt. Sci.* **8**, 205–307.

DE WERRA, D. 1985. An Introduction to Timetabling. *Eur. J. Opns. Res.* **19**, 151–162.

GAREY, M. R., AND D. S. JOHNSON. 1976. The Complexity of Near-Optimal Graph Coloring. *J. Assoc. Comput. Mach.* **23**, 43–49.

GAREY, M. R., AND D. S. JOHNSON. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco.

GLOVER, F. 1989. Tabu Search, Part I. *ORSA J. Comput.* **1**, 190–206.

GRIMMET, G. R., AND C. J. H. MCDIARMID. 1975. On Colouring Random Graphs. *Math. Proc. Cambridge Philos. Soc.* **77**, 313–324.

HERTZ, A., AND D. DE WERRA. 1987. Using Tabu Search Techniques for Graph Coloring. *Computing* **39**, 345–351.

JOHNSON, D. S. 1974. Worst-Case Behavior of Graph Coloring Algorithms. In *Proceedings 5th Southeastern Conference on Combinatorics, Graph Theory, and Computing*. Utilitas Mathematica Publishing, Winnipeg, 513–527.

JOHNSON, D. S., C. R. ARAGON, L. A. MCGEOCH AND C. SCHEVON. 1989. Optimization by Simulated Annealing: An Experimental Evaluation, Part I, Graph Partitioning. *Opns. Res.* **37**, 865–892.

JOHRI, A., AND D. W. MATULA. 1982. Probabilistic Bounds and Heuristic Algorithms for Coloring Large Random Graphs. Technical Report, Southern Methodist University, Dallas, Texas.

KARMARKAR, N., AND R. M. KARP. 1982. The Differencing Method of Set Partitioning. Report No. UCB/CSD 82/113, Computer Science Division, University of California, Berkeley.

KARMARKAR, N., R. M. KARP, G. S. LUEKER AND A. M. ODLYZKO. 1986. Probabilistic Analysis of Optimum Partitioning. *J. Appl. Prob.* **23**, 626–645.

KERNIGHAN, B. W., AND S. LIN. 1970. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Syst. Tech. J.* **49**, 291–307.

KIRKPATRICK, S., C. D. GELATT AND M. P. VECCHI. 13 MAY 1983. Optimization by Simulated Annealing. *Science* **220**, 671–680.

LEIGHTON, F. T. 1979. A Graph Coloring Algorithm for Large Scheduling Problems. *J. Res. Natl. Bur. Standards* **84**, 489–506.

LIN, S., AND B. W. KERNIGHAN. 1973. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Opns. Res.* **21**, 498–516.

MATULA, D. W., G. MARBLE AND J. D. ISAACSON. 1972. Graph Coloring Algorithms. In *Graph Theory and Computing*, R. C. Read (ed.). Academic Press, New York.

MORGENSTERN, C. 1989. Algorithms for General Graph Coloring. Doctoral Dissertation, Department of Computer Science, University of New Mexico, Albuquerque.

MORGENSTERN, C., AND H. SHAPIRO. 1986. Chromatic Number Approximation Using Simulated Annealing. Unpublished manuscript.

OPSUT, R. J., AND F. S. ROBERTS. 1981. On the Fleet Maintenance, Mobile Radio Frequency, Task Assignment, and Traffic Phasing Problems. In *The Theory and Applications of Graphs*, G. Chartrand, Y. Alavi, D. L. Goldsmith, L. Lesniak-Foster and D. R. Lick (eds.). John Wiley & Sons, New York, 479–492.

SHAMIR, A. 1979. On the Cryptocomplexity of Knapsack Systems. In *Proceedings 11th Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, New York, 118–129.

THOMASON, A. 1987. Private communication.

VAN LAARHOVEN, P. J. M. AND E. H. L. AARTS. 1987. *Simulated Annealing: Theory and Practice*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

VECCHI, M. P., AND S. KIRKPATRICK. 1983. Global Wiring by Simulated Annealing. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* **CAD-2**, 215–222.

WELSH, D. J. A., AND M. B. POWELL. 1967. An Upper Bound on the Chromatic Number of a Graph and Its Application to Timetabling Problems. *Comput. J.* **10**, 85–86.

WIGDERSON, A. 1983. Improving the Performance Guarantee of Approximate Graph Coloring. *J. Assoc. Comput. Mach.* **30**, 729–735.