OPTIMIZATION BY SIMULATED ANNEALING: AN EXPERIMENTAL EVALUATION; PART I, GRAPH 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 1988; revision received January 1989; accepted February 1989)

In this and two companion papers, we report on an extended empirical study of the *simulated annealing* approach to combinatorial optimization proposed by S. Kirkpatrick et al. That study investigated how best to adapt simulated annealing to particular problems and compared its performance to that of more traditional algorithms. This paper (Part I) discusses annealing and our parameterized generic implementation of it, describes how we adapted this generic algorithm to the graph partitioning problem, and reports how well it compared to standard algorithms like the Kernighan-Lin algorithm. (For sparse random graphs, it tended to outperform Kernighan-Lin as the number of vertices become large, even when its much greater running time was taken into account. It did not perform nearly so well, however, on graphs generated with a built-in geometric structure.) We also discuss how we went about optimizing our implementation, and describe the effects of changing the various annealing parameters or varying the basic annealing algorithm itself.

A new approach to the approximate solution of difficult combinatorial optimization problems recently has been proposed by Kirkpatrick, Gelatt and Vecchi (1983), and independently by Cerny (1985). This *simulated annealing* approach is based on ideas from statistical mechanics and motivated by an analogy to the behavior of physical systems in the presence of a heat bath. The nonphysicist, however, can view it simply as an enhanced version of the familiar technique of *local optimization* or *iterative improvement*, in which an initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution. Simulated annealing randomizes this procedure in a way that allows for occasional *uphill moves* (changes that worsen the solution), in an attempt to reduce the probability of becoming stuck in a poor but locally optimal solution. As with local search, simulated annealing can be adapted readily to new problems (even in the absence of deep insight into the problems themselves) and, because of its apparent ability to avoid poor local optima, it offers hope of obtaining significantly better results.

These observations, together with the intellectual appeal of the underlying physical analogy, have inspired articles in the popular scientific press (*Science 82*, 1982 and *Physics Today* 1982) as well as attempts to apply the approach to a variety of problems, in areas as diverse as VLSI design (Jepsen and Gelatt 1983, Kirkpatrick, Gelatt and Vecchi 1983, Vecchi and Kirkpatrick 1983, Rowan and Hennessy 1985), pattern recognition (Geman and Geman 1984, Hinton, Sejnowski and Ackley 1984) and code generation (El Gamal et al. 1987), often with substantial success. (See van Laarhoven and Aarts 1987 and Collins, Eglese and Golden 1988 for more up-to-date and extensive bibliographies of applications.) Many of

Subject classifications: Networks/graphs, heuristics: algorithms for graph partitioning. Simulation, applications: optimization by simulated annealing.

0030-364X/89/3706-0865 $01.25

the practical applications of annealing, however, have been in complicated problem domains, where previous algorithms either did not exist or performed quite poorly. In this paper and its two companions, we investigate the performance of simulated annealing in more competitive arenas, in the hope of obtaining a better view of the ultimate value and limitations of the approach.

The arena for this paper is the problem of partitioning the vertices of a graph into two equal size sets to minimize the number of edges with endpoints in both sets. This application was first proposed by Kirkpatrick, Gelatt and Vecchi, but was not extensively studied there. (Subsequently, Kirkpatrick 1984 went into the problem in more detail, but still did not deal adequately with the competition.)

Our paper is organized as follows. In Section 1, we introduce the graph partitioning problem and use it to illustrate the simulated annealing approach. We also sketch the physical analogy on which annealing is based, and discuss some of the reasons for optimism (and for skepticism) concerning it. Section 2 presents the details of our implementation of simulated annealing, describing a parameterized, *generic* annealing algorithm that calls problem-specific subroutines, and hence, can be used in a variety of problem domains.

Sections 3 through 6 present the results of our experiments with simulated annealing on the graph partitioning problem. Comparisons between annealing and its rivals are made difficult by the fact that the performance of annealing depends on the particular *annealing schedule* chosen and on other, more problem-specific parameters. Methodological questions also arise because annealing and its main competitors are *randomized* algorithms (and, hence, can give a variety of answers for the same instance) and because they have running times that differ by factors as large as 1,000 on our test instances. Thus, if comparisons are to be convincing and fair, they must be based on large numbers of independent runs of the algorithms, and we cannot simply compare the average cutsizes found. (In the time it takes to perform one run of the slower algorithm, one could perform many runs of the faster one and take the best solution found.)

Section 3 describes the problem-specific details of our implementation of annealing for graph partitioning. It then introduces two general types of test graphs, and summarizes the results of our comparisons between annealing, local optimization, and an algorithm due to Kernighan-Lin (1970) that has been the long-reigning champion for this problem. Annealing almost always outperformed local optimization, and for sparse random graphs it tended to outperform Kernighan-Lin as the number of vertices became large. For a class of random graphs with built-in geometric structure, however, Kernighan-Lin won the comparisons by a substantial margin. Thus, simulated annealing's success can best be described as mixed.

Section 4 describes the experiments by which we optimized the annealing parameters used to generate the results reported in Section 3. Section 5 investigates the effectiveness of various modifications and alternatives to the basic annealing algorithm. Section 6 discusses some of the other algorithms that have been proposed for graph partitioning, and considers how these might factor into our comparisons. We conclude in Section 7 with a summary of our observations about the value of simulated annealing for the graph partitioning problem, and with a list of lessons learned that may well be applicable to implementations of simulated annealing for other combinatorial optimization problems.

In the two companion papers to follow, we will report on our attempts to apply these lessons to three other well studied problems: Graph Coloring and Number Partitioning (Johnson et al. 1990a), and the Traveling Salesman Problem (Johnson et al. 1990b).

1. SIMULATED ANNEALING: THE BASIC CONCEPTS

1.1. Local Optimization

To understand simulated annealing, one must first understand local optimization. A combinatorial optimization problem can be specified by identifying a set of *solutions* together with a *cost function* that assigns a numerical value to each solution. An *optimal* solution is a solution with the minimum possible cost (there may be more than one such solution). Given an arbitrary solution to such a problem, local optimization attempts to improve on that solution by a series of incremental, local changes. To define a local optimization algorithm, one first specifies a method for perturbing solutions to obtain different ones. The set of solutions that can be obtained in one such step from a given solution A is called the *neighborhood* of A. The algorithm then performs the simple loop shown in Figure 1, with the specific methods for choosing S and S' left as implementation details.

Although S need not be an optimal solution when the loop is finally exited, it will be *locally optimal* in that none of its neighbors has a lower cost. The hope is that locally optimal will be good enough.

To illustrate these concepts, let us consider the *graph partitioning* problem that is to be the topic of Section

1. Get an initial solution S.
2. While there is an untested neighbor of S do the following.
   2.1 Let S' be an untested neighbor of S.
   2.2 If *cost* $(S') <$ *cost* (S), set $S = S'$.
3. Return S.

Figure 1. Local optimization.

3. In this problem, we are given a graph $G = (V, E)$, where V is a set of *vertices* (with $|V|$ even) and E is a set of pairs of vertices or *edges*. The *solutions* are partitions of V into equal sized sets. The *cost* of a partition is its *cutsize*, that is, the number of edges in E that have endpoints in both halves of the partition. We will have more to say about this problem in Section 3, but for now it is easy to specify a natural local optimization algorithm for it. Simply take the neighbors of a partition $\Pi = \{V_1 \cup V_2\}$ to be all those partitions obtainable from Π by exchanging one element of V_1 with one element of V_2.

For two reasons, graph partitioning is typical of the problems to which one might wish to apply local optimization. First, it is easy to find solutions, perturb them into other solutions, and evaluate the costs of such solutions. Thus, the individual steps of the iterative improvement loop are inexpensive. Second, like most interesting combinatorial optimization problems, graph partitioning is NP-complete (Garey, Johnson and Stockmeyer 1976, Garey and Johnson 1979). Thus, finding an *optimal* solution is presumably much more difficult than finding *some* solution, and one may be willing to settle for a solution that is merely *good enough*.

Unfortunately, there is a third way in which graph partitioning is typical: the solutions found by local optimization normally are *not* good enough. One can be locally optimal with respect to the given neighborhood structure and still be unacceptably distant from the globally optimal solution value. For example, Figure 2 shows a locally optimal partition with cutsize 4 for a graph that has an optimal cutsize of 0. It is clear that this small example can be generalized to arbitrarily bad ones.

1.2. Simulated Annealing

It is within this context that the simulated annealing approach was developed by Kirkpatrick, Gelatt and Vecchi. The difficulty with local optimization is that it has no way to *back out* of unattractive local optima. We never move to a new solution unless the direction is *downhill*, that is, to a better value of the cost function.

Simulated annealing is an approach that attempts to avoid entrapment in poor local optima by allowing an occasional *uphill* move. This is done under the influence of a random number generator and a control parameter called the *temperature*. As typically implemented, the simulated annealing approach involves a pair of nested loops and two additional parameters, a *cooling ratio* r, $0 < r < 1$, and an integer *temperature length* L (see Figure 3). In Step 3 of the algorithm, the term *frozen* refers to a state in which no further improvement in *cost*(S) seems likely.

The heart of this procedure is the loop at Step 3.1. Note that $e^{-\Delta/T}$ will be a number in the interval (0, 1) when Δ and T are positive, and rightfully can be interpreted as a probability that depends on Δ and T. The probability that an uphill move of size Δ will be accepted diminishes as the temperature declines, and, for a fixed temperature T, small uphill moves have higher probabilities of acceptance than large ones. This particular method of operation is motivated by a physical analogy, best described in terms of the physics of crystal growth. We shall discuss this analogy in the next section.

1.3. A Physical Analogy With Reservations

To grow a crystal, one starts by heating the raw materials to a molten state. The temperature of this *crystal melt* is then reduced until the crystal structure is *frozen* in. If the cooling is done very quickly (say, by dropping the external temperature immediately to absolute zero), bad things happen. In particular, widespread irregularities are locked into the crystal structure and the trapped energy level is much higher than in a perfectly structured crystal. This *rapid quenching* process can be viewed as analogous to local optimization. The states of the physical system correspond to the solutions of a combinatorial optimization problem; the energy of a state corresponds to the cost of a

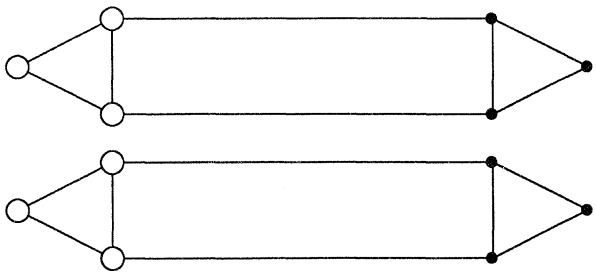

Figure 2. Bad but locally optimal partition with respect to pairwise interchange. (The dark and light vertices form the two halves of the partition.)

1. Get an initial solution S.
2. Get an initial temperature $T > 0$.
3. While not yet *frozen* do the following.
   - 3.1 Perform the following loop L times.
     - 3.1.1 Pick a random neighbor S' of S.
     - 3.1.2 Let $\Delta = cost\ (S') - cost\ (S)$.
     - 3.1.3 If $\Delta \leq 0$ (downhill move), Set $S = S'$.
     - 3.1.4 If $\Delta > 0$ (uphill move), Set $S = S'$ with probability $e^{-\Delta/T}$.
   - 3.2 Set $T = rT$ (reduce temperature).
4. Return S.

Figure 3. Simulated annealing.

solution, and the minimum energy or *ground* state corresponds to an optimal solution (see Figure 4). When the external temperature is absolute zero, no state transition can go to a state of higher energy. Thus, as in local optimization, uphill moves are prohibited and the consequences may be unfortunate.

When crystals are grown in practice, the danger of bad local optima is avoided because the temperature is lowered in a much more gradual way, by a process that Kirkpatrick, Gelatt and Vecchi call "careful annealing." In this process, the temperature descends slowly through a series of levels, each held long enough for the crystal melt to reach equilibrium at that temperature. As long as the temperature is nonzero, uphill moves remain possible. By keeping the temperature from getting too far ahead of the current equilibrium energy level, we can hope to avoid local optima until we are relatively close to the ground state.

Simulated annealing is the algorithmic counterpart to this physical annealing process, using the well known *Metropolis algorithm* as its inner loop. The Metropolis algorithm (Metropolis et al. 1953) was developed in the late 1940's for use in Monte Carlo simulations of such situations as the behavior of gases in the presence of an external heat bath at a fixed temperature (here the energies of the individual gas molecules are presumed to jump randomly from level to level in line with the computed probabilities). The name *simulated annealing* thus refers to the use of this simulation technique in conjunction with an *annealing schedule* of declining temperatures.

PHYSICAL SYSTEM	OPTIMIZATION PROBLEM
State	Feasible Solution
Energy	Cost
Ground State	Optimal Solution
Rapid Quenching	Local Search
Careful Annealing	Simulated Annealing

Figure 4. The analogy.

The simulated annealing approach was first developed by physicists, who used it with success on the Ising spin glass problem (Kirkpatrick, Gelatt and Vecchi), a combinatorial optimization problem where the solutions actually *are* states (in an idealized model of a physical system), and the cost function *is* the amount of (magnetic) energy in a state. In such an application, it was natural to associate such physical notions as *specific heat* and *phase transitions* with the simulated annealing process, thus, further elaborating the analogy with physical annealing. In proposing that the approach be applied to more traditional combinatorial optimization problems, Kirkpatrick, Gelatt and Vecchi and other authors (e.g., Bonomi and Lutton 1984, 1986, White 1984) have continued to speak of the operation of the algorithm in these physical terms.

Many researchers, however, including the authors of the current paper, are skeptical about the relevance of the details of the analogy to the actual performance of simulated annealing algorithms in practice. As a consequence of our doubts, we have chosen to view the parameterized algorithm described in Figure 3 simply as a procedure to be optimized and tested, free from any underlying assumptions about what the parameters *mean*. (We have not, however, gone so far as to abandon such standard terms as *temperature*.) Suggestions for optimizing the performance of simulated annealing that are based on the analogy have been tested, but only on an equal footing with other promising ideas.

1.4. Mathematical Results, With Reservations

In addition to the support that the simulated annealing approach gains from the physical analogy upon which it is based, there are more rigorous mathematical justifications for the approach, as seen, for instance, in Geman and Geman (1984), Anily and Federgruen (1985), Gelfand and Mitter (1985), Gidas (1985), Lundy and Mees (1986) and Mitra, Romeo and Sangiovanni-Vincentelli (1986). These formalize the physical notion of *equilibrium* mathematically as the equilibrium distribution of a Markov chain, and show that there are cooling schedules that yield limiting equilibrium distributions, over the space of all solutions, in which, essentially, all the probability is concentrated on the optimal solutions.

Unfortunately, these mathematical results provide little hope that the limiting distributions can be reached quickly. The one paper that has explicitly estimated such convergence times (Sasaki and Hajek 1988) concludes that they are exponential even for a very simple problem. Thus, these results do not seem

to provide much direct practical guidance for the real-world situation in which one must stop far short of the limiting distribution, settling for what are hoped to be near-optimal, rather than optimal solutions. The mathematical results do, however, provide intuitive support to the suggestion that slower cooling rates (and, hence, longer running times) may lead to better solutions, a suggestion that we shall examine in some detail.

1.5. Claims and Questions

Although simulated annealing has already proved its economic value in practical domains, such as those mentioned in the Introduction, one may still ask if it is truly as good a general approach as suggested by its first proponents. Like local optimization, it is widely applicable, even to problems one does not understand very well. Moreover, annealing apparently yields better solutions than local optimization, so more of these applications should prove fruitful. However, there are certain areas of potential difficulties for the approach.

First is the question of running time. Many researchers have observed that simulated annealing needs large amounts of running time to perform well, and this may push it out of the range of feasible approaches for some applications. Second is the question of adaptability. There are many problems for which local optimization is an especially poor heuristic, and even if one is prepared to devote large amounts of running time to simulated annealing, it is not clear that the improvement will be enough to yield good results. Underlying both these potential drawbacks is the fundamental question of competition.

Local search is not the only way to approach combinatorial optimization problems. Indeed, for some problems it is hopelessly outclassed by a more constructive technique one might call *successive augmentation*. In this approach, an initially empty structure is successively augmented until it becomes a solution. This, in particular, is the way that many of the efficiently solvable optimization problems, such as the Minimum Spanning Tree Problem and the Assignment Problem, are solved. Successive augmentation is also the design principle for many common heuristics that find near-optimal solutions.

Furthermore, even when local optimization is the method of choice, there are often other ways to improve on it besides simulated annealing, either by sophisticated backtracking techniques, or simply by running the local optimization algorithm many times from different starting points and taking the best solution found.

The intent of the experiments to be reported in this paper and its companions has been to subject simulated annealing to rigorous competitive testing in domains where sophisticated alternatives already exist, to obtain a more complete view of its robustness and strength.

2. FILLING IN THE DETAILS

The first problem faced by someone preparing to use or test simulated annealing is that the procedure is more an *approach* than a specific algorithm. Even if we abide by the basic outline sketched in Figure 3, we still must make a variety of choices for the values of the parameters and the meanings of the undefined terms. The choices fall into two classes: those that are problem-specific and those that are generic to the annealing process (see Figure 5).

We include the definitions of *solution* and *cost* in the list of choices even though they are presumably specified in the optimization problem we are trying to solve. Improved performance often can be obtained by modifying these definitions: the graph partitioning problem covered in this paper offers one example, as does the graph coloring problem that will be covered in Part II. Typically, the solution space is enlarged and penalty terms are added to the *cost* to make the *nonfeasible* solutions less attractive. (In these cases, we use the term *feasible solution* to characterize those solutions that are legal solutions to the original problem.)

Given all these choices, we face a dilemma in evaluating simulated annealing. Although experiments are capable of demonstrating that the approach performs well, it is impossible for them to prove that it performs poorly. Defenders of simulated annealing can always say that we made the wrong implementation choices. In such a case, the best we can hope is that our experiments are sufficiently extensive to make the existence of good parameter choices seem unlikely, at

PROBLEM-SPECIFIC
1. What is a *solution*?
2. What are the *neighbors* of a solution?
3. What is the *cost* of a solution?
4. How do we determine an *initial* solution?

GENERIC
1. How do we determine an *initial temperature*?
2. How do we determine the *cooling ratio r*?
3. How do we determine the *temperature length L*?
4. How do we know when we are *frozen*?

Figure 5. Choices to be made in implementing simulated annealing.

least in the absence of firm experimental evidence that such choices exist.

To provide a uniform framework for our experiments, we divide our implementations into two parts. The first part is a *generic* simulated annealing program, common to all our implementations. The second part consists of subroutines called by the generic program which are implemented separately for each problem domain. These subroutines, and the standard names we have established for them, are summarized in Figure 6. The subroutines share common data structures and variables that are unseen by the generic part of the implementation. It is easy to adapt our annealing code to new problems, given that only the problem-specific subroutines need be changed.

The generic part of our algorithm is heavily parameterized, to allow for experiments with a variety of factors that relate to the annealing process itself. These parameters are described in Figure 7. Because of space limitations, we have not included the complete generic code, but its functioning is fully determined by the information in Figures 3, 6, and 7, with the exception of our method for obtaining a starting temperature, given a value for *INITPROB*, which we shall discuss in Section 3.4.2.

Although the generic algorithm served as the basis for most of our experiments, we also performed limited tests on variants of the basic scheme. For example, we investigated the effects of changing the way temperatures are reduced, of allowing the number of trials per temperature to vary from one temperature to the next, and even of replacing the $e^{-\Delta/T}$ of the basic loop by a different function. The results of these experiments will be reported in Section 6.

READ_INSTANCE()
Reads instance and sets up appropriate data structures; returns the expected neighborhood size N for a solution (to be used in determining the initial temperature and the number L of trials per temperature).

INITIAL_SOLUTION()
Constructs an initial solution S_0 and returns *cost* (S_0), setting the locally-stored variable S equal to S_0 and the locally-stored variable c^* to some trivial upper bound on the optimal feasible solution cost.

PROPOSE_CHANGE()
Chooses a random neighbor S' of the current solution S and returns the difference *cost* (S') − *cost* (S), saving S' for possible use by the next subroutine.

CHANGE_SOLUTION()
Replaces S by S' in local memory, updating data structures as appropriate. If S' is a feasible solution and *cost* (S') is better than c^*, then sets c^* = *cost* (S') and sets the locally stored *champion* solution S^* to S'.

FINAL_SOLUTION()
Modifies the current solution S to obtain a feasible solution S''. (S'' = S if S is already feasible). If *cost* (S'') ≤ c^*, outputs S''; otherwise outputs S^*.

Figure 6. Problem-specific subroutines.

ITERNUM
The number of annealing runs to be performed with this set of parameters.

INITPROB
Used in determining an initial temperature for the current set of runs. Based on an abbreviated trial annealing run, a temperature is found at which the fraction of accepted moves is approximately *INITPROB*, and this is used as the starting temperature (if the parameter *STARTTEMP* is set, this is taken as the starting temperature, and the trial run is omitted).

TEMPFACTOR
This is a descriptive name for the cooling ratio r of Figure 3.

SIZEFACTOR
We set the temperature length L to be *N*SIZEFACTOR*, where N is the expected neighborhood size. We hope to be able to handle a range of instance sizes with a fixed value for *SIZEFACTOR*; temperature length will remain proportional to the number of neighbors no matter what the instance size.

MINPERCENT
This is used in testing whether the annealing run is *frozen* (and hence, should be terminated). A counter is maintained that is incremented by one each time a temperature is completed for which the percentage of accepted moves is *MINPERCENT* or less, and is reset to 0 each time a solution is found that is better than the previous champion. If the counter ever reaches 5, we declare the process to be *frozen*.

Figure 7. Generic parameters and their uses.

3. GRAPH PARTITIONING

The graph partitioning problem described in Section 2 has been the subject of much research over the years, because of its applications to circuit design and because, in its simplicity, it appeals to researchers as a test bed for algorithmic ideas. It was proved NP-complete by Garey, Johnson and Stockmeyer (1976), but even before that researchers had become convinced of its intractability, and hence, concentrated on heuristics, that is, algorithms for finding good but not necessarily optimal solutions.

For the last decade and a half, the recognized benchmark among heuristics has been the algorithm of Kernighan and Lin (1970), commonly called the Kernighan-Lin algorithm. This algorithm is a very sophisticated improvement on the basic local search procedure described in Section 2.1, involving an iterated backtracking procedure that typically

finds significantly better partitions. (For more details, see Section 7.) Moreover, if implemented using ideas from Fiduccia and Mattheyses (1982), the Kernighan-Lin algorithm runs very quickly in practice (Dunlop and Kernighan 1985). Thus, it represents a potent competitor for simulated annealing, and one that was ignored by Kirkpatrick, Gelatt and Vecchi when they first proposed using simulated annealing for graph partitioning. (This omission was partially rectified in Kirkpatrick (1984), where limited experiments with an inefficient implementation of Kernighan-Lin are reported.)

This section is organized as follows. In 3.1, we discuss the problem-specific details of our implementation of simulated annealing for graph partitioning and in 3.2 we describe the types of instances on which our experiments were performed. In 3.3, we present the results of our comparisons of local optimization, the Kernighan-Lin algorithm, and simulated annealing. Our annealing implementation generally outperforms the local optimization scheme on which it is based, even if relative running times are taken into account. The comparison with Kernighan-Lin is more problematic, and depends on the type of graph tested.

3.1. Problem-Specific Details

Although the neighborhood structure for graph partitioning described in Section 2.1 has the advantage of simplicity, it turns out that better performance can be obtained through indirection. We shall follow Kirkpatrick, Gelatt and Vecchi in adopting the following new definitions of *solution*, *neighbor*, and *cost*.

Recall that in the graph partitioning problem we are given a graph $G = (V, E)$ and are asked to find that partition $V = V_1 \cup V_2$ of V into equal sized sets that minimizes the number of edges that have endpoints in different sets. For our annealing scheme, a *solution* will be *any* partition $V = V_1 \cup V_2$ of the vertex set (not just a partition into equal sized sets). Two partitions will be *neighbors* if one can be obtained from the other by moving a single vertex from one of its sets to the other (rather than by exchanging *two* vertices, as in Section 2.1). To be specific, if (V_1, V_2) is a partition and $v \in V_1$, then $(V_1 - \{v\}, V_2 \cup \{v\})$ and (V_1, V_2) are neighbors. The *cost* of a partition (V_1, V_2) is defined to be

$$c(V_1, V_2) = |\{\{u, v\} \in E : u \in V_1 \ \& \ v \in V_2\}| + \alpha(|V_1| - |V_2|)^2$$

where $|X|$ is the number of elements in set X and α is a parameter called the *imbalance factor*. Note that although this scheme allows infeasible partitions to be solutions, it penalizes them according to the square of the imbalance. Consequently, at low temperatures the solutions tend to be almost perfectly balanced. This *penalty function* approach is common to implementations of simulated annealing, and is often effective, perhaps because the extra solutions that are allowed provide new *escape routes* out of local optima. In the case of graph partitioning, there is an extra benefit. Although this scheme allows more solutions than the original one, it has smaller neighborhoods (n neighbors versus $n^2/4$). Our experiments on this and other problems indicate that, under normal cooling rates such as $r = 0.95$, temperature lengths that are significantly smaller than the neighborhood size tend to give poor results. Thus, a smaller neighborhood size may well allow a shorter running time, a definite advantage if all other factors are equal.

Two final problem-specific details are our method for choosing an initial solution and our method for turning a nonfeasible final solution into a feasible one. Initial solutions are obtained by generating a random partition (for each vertex we flip an unbiased coin to determine whether it should go in V_1 or V_2). If the final solution remains unbalanced, we use a greedy heuristic to put it into balance. The heuristic repeats the following operation until the two sets of the partition are the same size: Find a vertex in the larger set that can be moved to the opposite set with the least increase in the cutsize, and move it. We output the best feasible solution found, be it this possibly modified final solution or some earlier feasible solution encountered along the way. This completes the description of our simulated annealing algorithm, modulo the specification of the individual parameters, which we shall provide shortly.

3.2. The Test Beds

Our explorations of the algorithm, its parameters, and its competitors will take place within two general classes of randomly generated graphs. The first type of graph is the standard *random graph*, defined in terms of two parameters, n and p. The parameter n specifies the number of vertices in the graph; the parameter p, $0 < p < 1$, specifies the probability that any given pair of vertices constitutes an edge. (We make the decision independently for each pair.) Note that the expected average vertex degree in the random graph $G_{n,p}$ is $p(n-1)$. We shall usually choose p so that this expectation is small, say less than 20, as most interesting applications involve graphs with a low average degree, and because such graphs are better for distinguishing the performance of different heuristics than more dense ones. (For some theoretical results

about the expected minimum cutsizes for random graphs, see Bui 1983.)

Our second class of instances is based on a non-standard type of random graph, one that may be closer to real applications than the standard one, in that the graphs of this new type will have by definition inherent structure and clustering. An additional advantage is that they lend themselves to two-dimensional depiction, although they tend to be highly nonplanar. They again have two parameters, this time denoted by n and d. The random *geometric graph* $U_{n,d}$ has n vertices and 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 the vertices; we place an edge between two vertices if and only if their (Euclidean) distance is d or less. (See Figure 8 for an example.) Note that for points not too close to the boundary the expected average degree will be approximately $n\pi d^2$.

Although neither of these classes is likely to arise in a typical application, they provide the basis for repeatable experiments, and, it is hoped, constitute a broad enough spectrum to yield insights into the general performance of the algorithms.

3.3. Experimental Results

As a result of the extensive testing reported in Section 5, we settled on the following values for the five parameters in our annealing implementation: $\alpha = 0.05$, $INITPROB = 0.4$, $TEMPFACTOR = 0.95$, $SIZEFACTOR = 16$, and $MINPERCENT = 2$. We compared this particular implementation (hereafter referred to as Annealing with a capital A) to the Kernighan-Lin algorithm (hereafter referred to as the K-L algorithm), and a local optimization algorithm (referred to as Local Opt) based on the same neighborhood structure as our annealing algorithm, with the same rebalancing heuristic used for patching up locally optimal solutions that were out-of-balance. (Experiments showed that this local optimization approach yielded distinctly better average cutsizes than the one based on the pairwise interchange neighborhood discussed in Section 2.1, without a substantial increase in running time.) All computations were performed on VAX 11-750 computers with floating point accelerators and 3 or 4 megabytes of main memory (enough memory so that our programs could run without delays due to paging), running under the UNIX operating system (Version 8). (VAX is a trademark of the Digital Equipment Corporation; UNIX is a trademark of AT&T Bell Laboratories.) The programs were written in the C programming language.

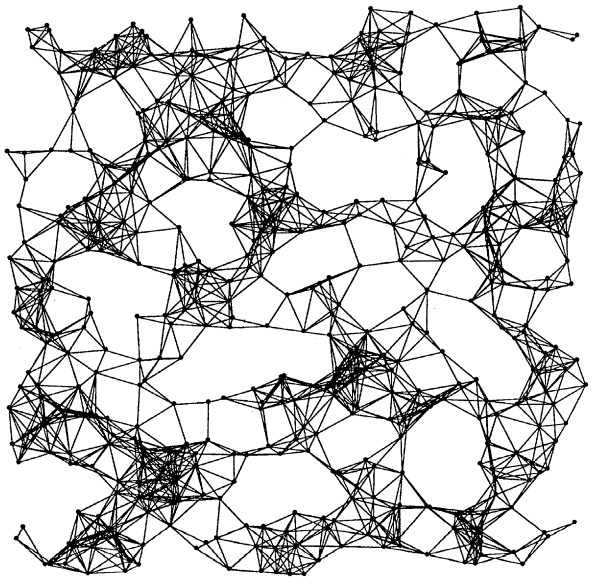

Figure 8. A geometric graph with $n = 500$ and $n\pi d^2 = 10$.

The evaluation of our experiments is complicated by the fact that we are dealing with *randomized* algorithms, that is, algorithms that do not always yield the same answer on the same input. (Although only the simulated annealing algorithm calls its random number generator *during* its operation, *all* the algorithms are implemented to start from an initial random partition.) Moreover, results can differ substantially from run to run, making comparisons between algorithms less straightforward.

Consider Figure 9, in which histograms of the cuts found in 1,000 runs each of Annealing, Local Opt, and K-L are presented. The instance in question was a random graph with $n = 500$ and $p = 0.01$. This particular graph was used as a benchmark in many of our experiments, and we shall refer to it as G_{500} in the future. (It has 1,196 edges for an average degree of 4.784, slightly less than the expected figure of 4.99.) The histograms for Annealing and Local Opt both can be displayed on the same axis because the worst cut found in 1,000 runs of Annealing was substantially better (by a standard deviation or so) than the *best* cut found during 1,000 runs of Local Opt. This disparity more than balances the differences in running time: Even though the average running time for Local Opt was only a second compared to roughly 6 *minutes* for Annealing, one could not expect to improve on Annealing simply by spending an equivalent time doing multiple runs of Local Opt, as some critics suggested might be the case. Indeed, the best cut found in 3.6 million runs of Local Opt (which took roughly the same 600 hours as did our 1,000 runs of

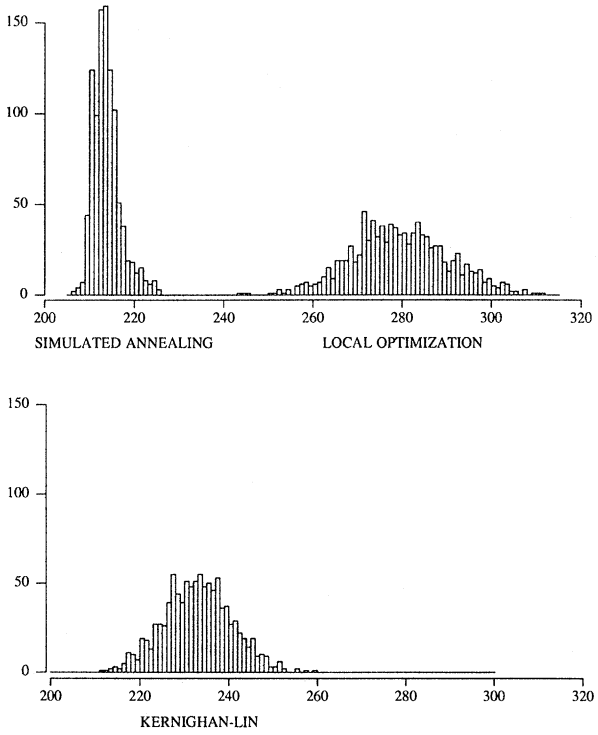

Figure 9. Histograms of solution values found for graph G_{500} during 1,000 runs each of Annealing, Local Opt and Kernighan-Lin. (The X-axis corresponds to cutsize and the Y-axis to the number of times each cutsize was encountered in the sample.)

Annealing) was 232, compared to 225 for the worst of the annealing runs. One can, thus, conclude that this simulated annealing implementation is intrinsically more powerful than the local optimization heuristic on which it is based, even when running time is taken into account.

Somewhat less conclusive is the relative performance of Annealing and the sophisticated K-L algorithm. Here the histograms would overlap if they were placed on the same axis, although the median and other order statistics for Annealing all improve on the corresponding statistics for K-L. However, once again, Annealing is by far the slower of the two algorithms, this time by a factor of roughly 100 (K-L had an average running time of 3.7 seconds on G_{500}). Thus, ideally we should compare the best of 100 runs of K-L versus one run of Annealing, or the best of $100k$ runs versus the best of k.

Fortunately, there is a more efficient way to obtain an estimate of the expected best of k runs than simply to repeatedly perform sets of k runs and compute the average of the bests. We perform some number $m \gg k$ runs, and then compute the expected best of a random sample of k of these particular m runs, chosen without replacement. (This can be done by arranging the m results in order from best to worst and then looping through them, noting that, for $1 < j \leq m - k + 1$, the probability that the jth best value in the overall sample is the best in a subsample of size k is $k/(m - j + 1)$ times the probability that none of the earlier values was the best.) The reliability of such an estimate on the best of k runs, of course, decreases rapidly as k approaches m, and we will usually cite the relevant values of m and k so that readers who wish to assess the confidence intervals for our results can do so. We have not done so here as we are not interested in the precise values obtained from any particular experiment, but rather in trends that show up across groups of related experiments.

Table I shows our estimates for the expected best of k runs of Annealing, k runs of K-L, and $100k$ runs of K-L for various values of k, based on the 1,000 runs of Annealing charted in Figure 9 plus a sample of 10,000 runs of K-L. Annealing clearly dominates K-L if running time is not taken into account, and still wins when running time *is* taken into account, although the margin of victory is much less impressive (but note that the margin increases as k increases). The best cut ever found for this graph was one of size 206, seen once in the 1,000 Annealing runs.

To put these results in perspective, we performed similar, though less extensive, experiments with random graphs that were generated using different choices for n and p. We considered all possible combinations of a value of n from {124, 250, 500, 1,000} with a value of np (approximately the expected average degree) from {2.5, 5, 10, 20}. We only experimented with one graph of each type, and, as noted above, the overall *pattern* of results is much more significant than any individual entry. Individual variability among graphs generated with the same parameters can be substantial: the graph with $n = 500$ and $p = 0.01$ used in these experiments was significantly denser than

Table I
Comparison of Annealing and Kernighan-Lin on G_{500}

k	Anneal (Best of k)	K-L (Best of k)	K-L (Best of $100k$)
1	213.32	232.29	214.33
2	211.66	227.92	213.19
5	210.27	223.30	212.03
10	209.53	220.49	211.38
25	208.76	217.51	210.81
50	208.20	215.75	210.50
100	207.59	214.33	210.00

G_{500}, which was generated using the same parameters. It had an average degree of 4.892 versus 4.784 for G_{500}, and the best cut found for it was of size 219 versus 206. Thus, one cannot expect to be able to replicate our experiments exactly by independently generating graphs using the same parameters. The cutsizes found, the extent of the lead of one algorithm over another, and even their rank order, may vary from graph to graph. We expect, however, the same general trends to be observable. (For the record, the actual average degrees for the 16 new graphs are: 2.403, 5.129, 10.000, 20.500 for the 124-vertex graphs; 2.648, 4.896, 10.264, 19.368 for the 250-vertex graphs; 2.500, 4.892, 9.420, 20.480 for the 500-vertex graphs; and 2.544, 4.992, 10.128, 20.214 for the 1,000-vertex graphs.)

The results of our experiments are summarized in Tables II–V. We performed 20 runs of Annealing for each graph, as well as 2,000 runs each of Local Opt and K-L. Table II gives the best cuts ever found for each of the 16 graphs, which may or not be the *optimal* cuts. Table III reports the estimated means for all the algorithms, expressed as a percentage above the best cut found. Note that Annealing is a clear winner in these comparisons, which do not take running time into account.

Once again, however, Annealing dominates the other algorithms in amount of time required, as shown in Table IV. The times listed do not include the time needed to read in the graph and create the initial data structures, as these are the same for all the algorithms, independently of the number of runs performed, and were, in any case, much smaller than the time for a single run. The times for Annealing also do not include the time used in performing a trial run to determine an initial temperature. This is substantially less than that required for a full run; moreover, we expect that in practice an appropriate starting temperature would be known in advance—from experience with similar instances—or could be determined analytically as a simple function of the numbers of vertices and edges in the graph, a line of research we leave to the interested reader.

Table II
Best Cuts Found for 16 Random Graphs

	Expected Average Degree			
|V|	2.5	5.0	10.0	20.0
124	13	63	178	449
250	29	114	357	828
500	52	219	628	1,744
1,000	102	451	1,367	3,389

Table III
Average Algorithmic Results for 16 Random Graphs (Percent Above Best Cut Ever Found)

	Expected Average Degree				
|V|	2.5	5.0	10.0	20.0	Algorithm
124	87.8	24.1	9.5	5.6	Local Opt
	18.7	6.5	3.1	1.9	K-L
	4.2	1.9	0.6	0.2	Annealing
250	101.4	26.5	11.0	5.5	Local Opt
	21.9	8.6	4.3	1.9	K-L
	10.2	1.8	0.8	0.4	Annealing
500	102.3	32.9	12.5	5.8	Local Opt
	23.4	11.5	4.4	2.4	K-L
	10.0	2.2	0.9	0.5	Annealing
1,000	106.8	31.2	12.5	6.3	Local Opt
	22.5	10.8	4.8	2.7	K-L
	7.4	2.0	0.7	0.4	Annealing

A comment about the running times in Table IV is in order. As might be expected, the running times for Kernighan-Lin and Local Opt increase if the number of vertices increases or the density (number of edges) increases. The behavior of Annealing is somewhat anomalous, however. For a fixed number of vertices, the running time does not increase monotonically with density, but instead goes through an initial decline as the average degree increases from 2.5 to 5. This can be explained by a more detailed look at the way Annealing spends its time. The amount of time *per temperature* increases monotonically with density. The *number* of temperature reductions needed, however, declines as density increases, that is, freezing sets in earlier for the denser graphs. The interaction between these two phenomena accounts for the nonmonotonicity in total running time.

Table V gives results better equalized for running time. Instead of using a single run of Annealing as our standard for comparison, we use the procedure that runs Annealing 5 times and takes the best result. As can be seen, this yields significantly better results, and is the recommended way to use annealing in practice, assuming enough running time is available. For each of the other algorithms, the number of runs corresponding to 5 Anneals was obtained separately for each graph, based on the running times reported in Table IV. Observe that once again Annealing's advantage is substantially reduced when running times are taken into account. Indeed, it is actually beaten by Kernighan-Lin on most of the 124- and 250-vertex graphs, on the sparsest 500-vertex graph, and by Local Opt on the densest 124-vertex graph. Annealing does, however, appear to be pulling away as the number of vertices increases. It is the overall winner for the

Table IV
Average Algorithmic Running Times in Seconds for 16 Random Graphs

	Expected Average Degree				
|V|	2.5	5.0	10.0	20.0	Algorithm
124	0.1	0.2	0.3	0.8	Local Opt
	0.8	1.0	1.4	2.6	K-L
	85.4	82.8	78.1	104.8	Annealing
250	0.3	0.4	0.8	1.3	Local Opt
	1.5	2.0	2.9	4.6	K-L
	190.6	163.7	186.8	223.3	Annealing
500	0.6	0.9	1.5	3.2	Local Opt
	2.8	3.8	5.7	11.4	K-L
	379.8	308.9	341.5	432.9	Annealing
1,000	2.4	3.8	6.9	14.1	Local Opt
	7.0	8.5	14.9	27.5	K-L
	729.9	661.2	734.5	853.7	Annealing

densest 250-vertex graph, the three densest 500-vertex graphs, all four 1,000-vertex graphs, and for each density its lead increases with graph size. Note, however, that by choosing to use the best of 5 runs for Annealing as our standard of comparison, rather than a single run, we have shifted the balance slightly to Annealing's advantage (assuming the results reported in Table I are typical). Indeed, had we compared the expected cut for a *single* Annealing run to an equivalent number of K-L runs on the 1,000-vertex, average degree 2.5 graph, K-L would outperform Annealing rather than lose to it by a small amount as it does here.

Our next series of experiments concerned geometric graphs. We considered just 8 graphs this time, four with 500 vertices and four with 1,000. For these we chose values of d so that the expected degrees $n\pi d^2 \in \{5, 10, 20, 40\}$. (The actual average degrees were 5.128, 9.420, 18.196, and 35.172 for the 500-vertex graphs and 4.788, 9.392, 18.678, and 36.030 for the 1,000-vertex graphs.) We performed 20 runs of Annealing and 2,000 runs each of Local Opt and K-L on each of the 8 graphs. The best cuts ever found for these graphs were 4, 26, 178, and 412 for the 500-vertex graphs, and 3, 39, 222, and 737 for the 1,000-vertex graphs. None of these were found by Annealing; all but the 4, 3, and 39 were found by K-L. The latter three were found by a special hybrid algorithm that takes into account the geometry of these graphs, and will be discussed in more detail in Section 6. Table VI summarizes the average running times of the algorithms. Table VII estimates the expected best cut encountered in 5 runs of Annealing or a time-equivalent number of runs of K-L or Local Opt. Note that for geometric graphs. On the sparsest graphs, none of the algorithms is particularly good, but K-L substantially outperforms Annealing.

Table V
Estimated Performance of Algorithms When Equalized for Running Times (Percent Above Best Cut Ever Found)[a]

	Expected Average Degree				
|V|	2.5	5.0	10.0	20.0	Algorithm
124	7.7	1.6	0.1	0.0	Locals
	0.0	0.0	0.0	0.0	K-L's
	0.0	0.4	0.1	0.2	5 Anneals[a]
250	31.0	5.3	2.1	1.4	Locals
	0.0	0.3	0.2	0.1	K-L's
	1.8	0.6	0.3	0.0	5 Anneals
500	59.6	13.2	4.5	2.5	Locals
	1.7	2.6	0.8	0.6	K-L's
	5.7	0.8	0.2	0.2	5 Anneals
1,000	72.4	19.9	7.6	3.7	Locals
	3.7	3.6	1.5	0.9	K-L's
	3.2	0.8	0.2	0.1	5 Anneals

[a] The best of 5 runs of Annealing is compared with the best obtainable in the same overall running time by performing multiple runs of Local Opt and Kernighan-Lin.

Table VI
Average Algorithmic Running Times in Seconds for 8 Geometric Graphs

	$n\pi d^2$				
|V|	5	10	20	40	Algorithm
500	1.0	1.6	3.2	7.2	Local Opt
	3.4	4.8	7.4	11.1	K-L
	293.3	306.3	287.2	209.9	Annealing
1,000	2.2	3.7	7.2	18.0	Local Opt
	7.6	11.9	18.9	28.7	K-L
	539.3	563.7	548.7	1038.2	Annealing

Annealing's poorer relative performance here may well be traceable to the fact that the topography of the solution spaces for geometric graphs differs sharply from that of random graphs. Here local optima may be far away from each other in terms of the length of the shortest chain of neighbors that must be traversed in transforming one to the other. Thus, Annealing is much more likely to be trapped in bad local optima. As evidence for the different nature of the solution spaces for geometric graphs, consider Figure 10, which shows a histogram of cutsizes found by K-L for a geometric graph with $n = 500$, $n\pi d^2 = 20$. Compared to Figure 9, which is typical of the histograms one generates for standard random graphs, the histogram of Figure 10 is distinctly more spiky and less bell shaped, and has a much larger ratio of best to worst cut encountered. The corresponding histogram for

Table VII
Estimated Performance of Algorithms When Equalized for Running Times (Percent Above Best Cut Ever Found)

	$n\pi d^2$				
|V|	5	10	20	40	Algorithm
500	647.5	169.6	11.3	0.0	Locals
	184.5	3.3	0.0	0.0	K-L's
	271.6	70.6	11.3	15.5	5 Anneals
1,000	3217.2	442.7	82.5	6.4	Locals
	908.9	44.4	1.3	0.0	K-L's
	1095.0	137.2	15.7	7.2	5 Anneals

Local Opt is slightly more bell shaped, but continues to be spiky with a large ratio of best to worst cut found. (For lower values of d, the histograms look much more like those for random graphs, but retain some of their spikiness. For higher values of d, the solution values begin to be separated by large gaps.)

In Sections 4 and 5, we consider the tradeoffs involved in our annealing implementation, and examine whether altering the implementation might enable us to obtain better results for annealing than those summarized in Tables V and VII.

4. OPTIMIZING THE PARAMETER SETTINGS

In this section, we will describe the experiments that led us to the standard parameter settings used in the above experiments. In attempting to optimize our annealing implementation, we faced the same sorts of questions that any potential annealer must address. We do not claim that our conclusions will be applicable to all annealing implementations; they may not even be precisely applicable to graph partitioning if the graphs are substantially larger or different in character from those we studied. (The main experiments were performed on our standard graph G_{500}, with secondary runs performed on a small selection of other types of graphs to provide a form of validation.) As there were too many parameters for us to investigate all possible combinations of values, we studied just one or two factors at a time, in hope of isolating their effects.

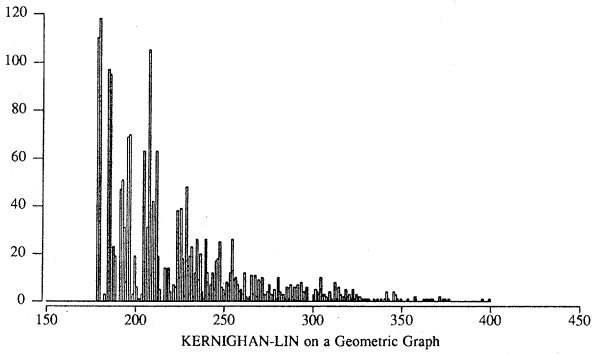

Figure 10. Histogram of solution values found for a geometric graph with $n = 500$, $n\pi d^2 = 20$ during 2,000 runs of K-L. (The X-axis corresponds to cutsize and the Y-axis to the number of times each cutsize was encountered in the sample.)

Despite the limited nature of our experiments, they may be useful in suggesting what questions to investigate in optimizing other annealing implementations, and we have used them as a guide in adapting annealing to the three problems covered in Parts II and III.

4.1. The Imbalance Factor

First, let us examine the effect of varying our one problem-specific parameter, the imbalance factor α. Figure 11 shows, for each of a set of possible values for α, the cutsizes found by 20 runs of our annealing algorithm on G_{500}. In these runs, the other annealing parameters were set to the standard values specified in the previous section. The results are represented by *box plots* (McGill, Tukey and Desarbo 1978), constructed by the AT&T Bell Laboratories statistical graphics package *S*, as were the histograms in the preceding section. Each box delimits the middle two quartiles of the results (the middle line is the median). The *whiskers* above and below the box go to the farthest values that are within distance (1.5) * (quartile length) of the box boundaries. Values beyond the whiskers are individually plotted.

Observe that there is a broad *safe* range of equivalently good values, and our chosen value of $\alpha = 0.05$ falls within that range. Similar results were obtained for a denser 500-vertex random graph and a sparser 1,000-vertex random graph, as well as for several geometric graphs. For all graphs, large values of α lead to comparatively poor results. This provides support for our earlier claim that annealing benefits from the availability of out-of-balance partitions as *solutions*. The main effect of large values of α is to discourage such partitions. As this figure hints, the algorithm also performs poorly for small values of α. For such α, the algorithm is likely to stop (i.e., freeze) with a partition that is far out-of-balance, and our greedy rebalancing heuristic is not at its best in such situations.

Unexpectedly, the choice of α also has an effect on the running time of the algorithm. See Figure 12, where running time is plotted as a function of α for our runs on graph G_{500}. Note that the average running

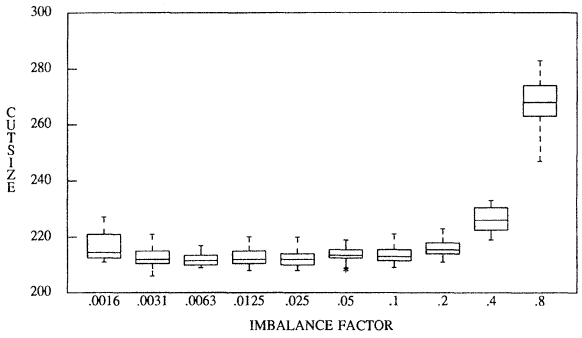

Figure 11. Effect of imbalance factor α on cutsize found for G_{500}.

time first increases, then decreases as α increases. The same behavior occurs for the other graphs we tested, although the point at which the running time attains its maximum varies. This appears to be a complex phenomenon, and several factors seem to be involved. First, lower values of α should make escaping from local optima easier, and hence, quicker. On the other hand, an interaction between α and our freezing criterion may prolong the cooling process. As α decreases, so does the cost of the cheapest uphill move, that is, a move that increases the imbalance from 0 to 2 but does not change the number of edges in the cut. Since moves of this type are presumably always available, lowering their cost may lower the temperature needed for the solution to become frozen, given that we don't consider ourselves frozen until fewer than 2% of the moves are accepted. This effect is, in turn, partially balanced by the fact that the *initial* temperature—chosen to be such that 40% of the moves are accepted—also declines with α.

In choosing a standard value for α, we restricted attention to those values that lay in the safe ranges for all the graphs tested with respect to cutsizes found and attempted to find a value that minimized running

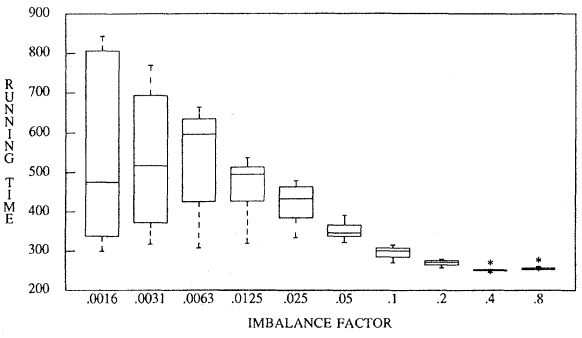

Figure 12. Effect of imbalance factor α on running time in seconds for G_{500}.

time among such candidates. No candidate yielded a simultaneous minimum for all the graphs, but our value of $\alpha = 0.05$ was a reasonable compromise.

4.2. Parameters for Initialization and Termination

As we have seen, the choice of α has an indirect effect on the annealing schedule. It is mainly, however, the generic parameters of our annealing implementation that affect the ranges of temperatures considered, the rate at which the temperature is lowered, and the time spent at each temperature. Let us first concentrate on the range of temperatures, and do so by taking a more detailed look at the operation of the algorithm for an expanded temperature range.

Figure 13 presents a *time exposure* of an annealing run on our standard random graph G_{500}. The standard

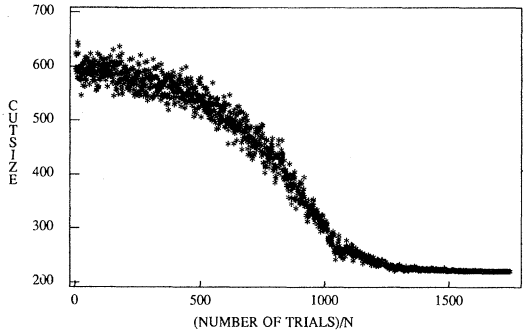

Figure 13. The evolution of the solution value during annealing on G_{500}. (Time increases, and hence, temperature decreases along the X-axis. The Y-axis measures the current solution value, that is, the number of edges in the cut plus the imbalance penalty.)

parameters were used with the exception of INITPROB, which was increased to 0.90, and MINPERCENT, which was dropped to 1%. During the run, the solution value was sampled each $N = 500$ trials (i.e., 16 times per temperature), and these values are plotted as a function of the time at which they were encountered (i.e., the number of trials so far, expressed as a multiple of 500). This means that temperature *decreases* from left to right. It is clear from this picture that little progress is made at the end of the schedule (there is no change at all in the last 100 samples). Moreover, the value of the time spent at the beginning of the schedule is also questionable. For the first 200 or so samples the cutsizes can barely be distinguished from those of totally random partitions (1,000 randomly generated partitions for this graph had a mean cutsize of about 599 and ranged from 549 to 665).

Furthermore, although the curve in Figure 13 begins to slope downward at about the 200th sample, a view behind the scenes suggests that serious progress does not begin until much later. Figure 14 depicts a second annealing run, where instead of reporting the cutsize for each sample, we report the cutsize obtained when Local Opt is applied to that sample (a much lower figure). Comparing Figure 14 with the histogram for Local Opt in Figure 9, we note that the values do not fall significantly below what might be expected from random start Local Opt until about the 700th sample, and are only beginning to edge their way down when the acceptance rate drops to 40% (the dotted line at sample 750).

There still remains the question of whether the time spent at high temperatures might somehow be laying necessary, but hidden, groundwork for what follows. Figure 15 addresses this issue, comparing a much shorter annealing run, using our standard values of INITPROB = 0.4 and MINPERCENT = 2%, with the tail of the run depicted in Figure 13. (All samples from Figure 13 that represent temperatures higher than the initial temperature for the INITPROB = 0.4 run were deleted.) Note the marked similarity between the two plots, even to the size of the cuts found (214 and 215, respectively, well within the range of variability for the algorithm).

All this suggests that the abbreviated schedule imposed by our standard parameters can yield results as good as those for the extended schedule, while using less than half the running time. More extensive experiments support this view: Figures 16 and 17 present box plots of cutsize and running time for a series of runs on G_{500}. We performed 20 runs for each of nine

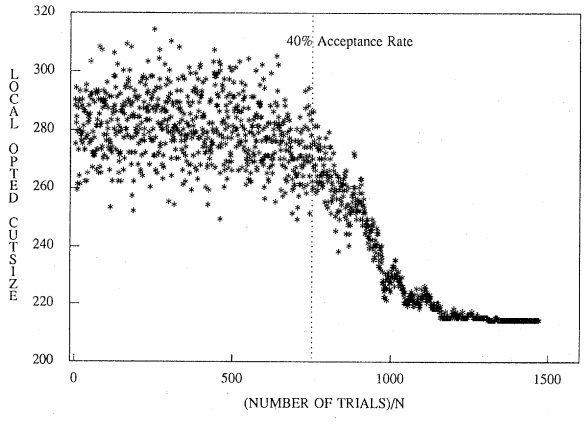

Figure 14. The evolution of Local-Opt(S) during annealing on G_{500}, where S is the current solution and Local-Opt(S) is the cutsize obtained by applying Local Opt to S.

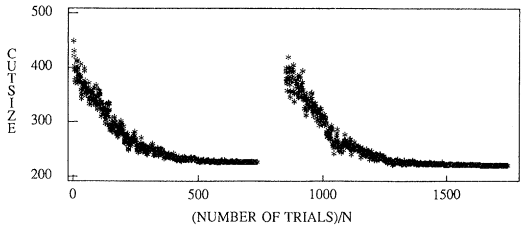

Figure 15. Comparison of the tail of an annealing run where INITPROB = 0.4 and MINPERCENT = 2% with an entire annealing run where INITPROB = 0.9 and MINPERCENT = 1%.

different values of INITPROB, from 0.1 to 0.9. Given the inherent variability of the algorithm, all values of INITPROB from 0.2 (or 0.3) to 0.9 seem to be roughly equivalent in the quality of cutsize they deliver. Running time, however, clearly increases with INITPROB. Similar results were obtained for the other graphs mentioned in Section 4.1, and our choice of INITPROB = 0.4 was again based on an attempt to reduce running time as much as possible without sacrificing solution quality. Analogous experiments led to our choice of MINPERCENT = 2%.

4.3. TEMPFACTOR and SIZEFACTOR

The remaining generic parameters are TEMPFACTOR and SIZEFACTOR, which together control how much time is taken in cooling from a given starting temperature to a given final one. Tables VIII and IX illustrate an investigation of these factors for our standard random graph G_{500}. (Similar results also were obtained for geometric graphs.) We fix all parameters except the two in question at their standard settings, in addition we fix the starting temperature at 1.3, a typical value for this graph when INITPROB = 0.4. We then let TEMPFACTOR and SIZEFACTOR take on various combinations of values from {0.4401, 0.6634, 0.8145, 0.9025, 0.9500, 0.9747, 0.9873} and {0.5, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1,024}, respectively. Note that increasing either value to the next larger should be expected to approximately double the running time, all other factors being equal. (The increases in TEMPFACTOR amount to replacing the current value by its square root, and hence, yield schedules in which twice as many temperatures are encountered in a given range.) The averages presented in Tables VIII and IX are based on 20 annealing runs for each combination of values.

Table VIII shows the effects of the parameters on running time. Our running time prediction was only approximately correct. Whereas doubling the time

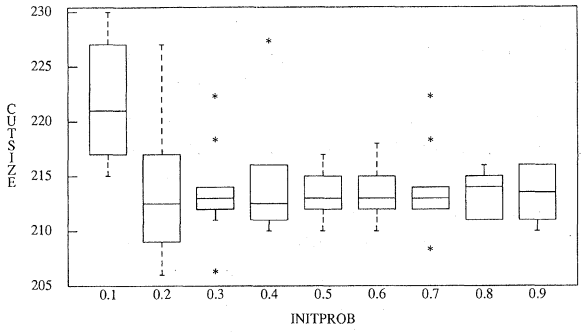

Figure 16. Effect of INITPROB on cutsize found for G_{500}.

spent per temperature seems to double the running time, it appears that halving the cooling rate only multiplies it by a factor of roughly 1.85. Thus, the (TEMPFACTOR, SIZEFACTOR) pair (0.6634, 128), which spends 8 times longer per temperature than (0.95, 16) but makes only one temperature drop for every 8 made by the latter, ends up taking almost twice as long to freeze. Two factors seem to be working here. Compare Figure 18, which provides a time exposure of an annealing run with the pair (0.6634, 128), with the left half of Figure 14, which does the same thing for (0.95, 16). First, we observe that our stopping criterion, which requires 5 consecutive temperatures without an improvement, causes the frozen *tail* of the time exposure to be much longer in the (0.6634, 128) case. Second, although this appears to be a more minor effect, the onset of freezing seems to have been delayed somewhat, perhaps because so little progress is made while the temperature is fixed.

Table IX shows the average cut found for each combination of the parameters. Unsurprisingly, increasing the running time tends to yield better solutions, although beyond a certain point further increases do not seem cost effective. A second observation is that increasing TEMPFACTOR to its square root appears to have the same effect on quality of solution as doubling SIZEFACTOR even though it does not add as much to the running time. (The values along the diagonals in Table IX remain fairly constant.) Thus, increasing TEMPFACTOR seems to be the preferred method for improving the annealing results by adding running time to the schedule.

To test this hypothesis in more detail, we performed 300 runs of Annealing on G_{500} with SIZEFACTOR = 1 and TEMPFACTOR = 0.99678 (approximately $0.95^{1/16}$). The average running time was 10% better than for our standard parameters, and the average cutsize was 213.8, only slightly worse than the 213.3 average for the standard parameters. It is not clear whether this slight degradation is of great statistical significance, but there is a plausible reason why the cutsize might be worse: Once the acceptance rate drops below MINPERCENT, our stopping criterion will

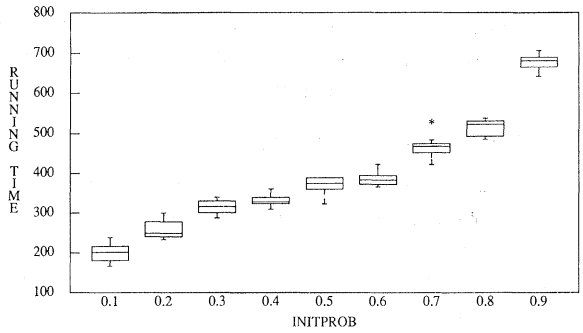

Figure 17. Effect of INITPROB on running time in seconds for G_{500}.

Table VIII
Dependence of Running Time on SIZEFACTOR and TEMPFACTOR for G_{500} (Number of Trials/N)

SIZE FACTOR	TEMPFACTOR						
	0.4401	0.6634	0.8145	0.9025	0.9500	0.9747	0.9873
0.25	—	—	—	—	—	—	37
0.5	—	—	—	—	—	78	45
1	—	—	—	—	49	86	164
2	—	—	—	53	97	178	323
4	—	—	65	109	190	332	662
8	—	88	130	209	361	682	1,317
16	130	174	261	411	734	1,336	2,602
32	208	336	490	820	1,459	2,854	—
64	416	691	992	1,600	2,874	—	—
128	1,024	1,331	1,971	3,264	—	—	—
256	2,048	2,688	3,994	—	—	—	—
512	4,096	5,427	—	—	—	—	—
1,024	8,192	—	—	—	—	—	—

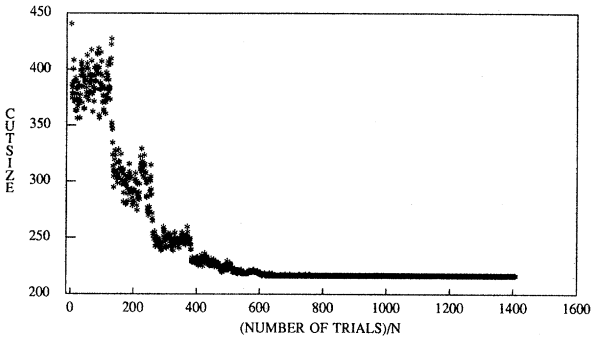

Figure 18. Evolution of cutsize when TEMPFACTOR = 0.6634 and SIZEFACTOR = 128.

terminate a run if there are roughly $5 \mid V \mid \cdot$ SIZEFACTOR consecutive trials without an improvement, and the smaller SIZEFACTOR is, the higher the probability that this might occur prematurely. For this paper, we have chosen to stick with our standard parameter values, rather than risk solution degradation for only a small improvement in running time. In practice, however, one might prefer to gamble on the speedup. In Section 6 we shall discuss how such speedups (in conjunction with others) might effect the basic comparisons of Section 4.

4.4. A Final Time/Quality Tradeoff

The final generic parameter to be investigated is ITERNUM, the number of iterations to be performed. In previous sections, we saw that we could improve on the results of a single run of annealing by performing multiple runs and taking the best solution found. The results just reported indicate that we can also obtain improved cuts by allowing an individual run to take more time. The question naturally arises: what is the best way to allocate a given amount of computation time between the number of runs and time per run? In simplest terms, suppose that we are currently taking the best of k runs. Could we get better results by performing $2k$ runs with the parameters set so that each run takes half as long, or should we perhaps perform $k/2$ runs, each taking twice as long?

There does not appear to be a simple answer. The choice may vary, depending on the total time available. For small time limits, it appears that it is best to spend all the time on a single run. Consider our standard random graph G_{500}. Experiments indicate that if we allow annealing 3 seconds (say setting SIZEFACTOR = 1 and TEMPFACTOR = 0.05), the best we can hope for is that Annealing approximate Local Opt in the quality of solution found. Thus, if we only have 6 minutes available, the histograms of Figure 9 indicate that it is better to spend that time on a single 6-minute annealing run rather than on 120 runs of 3 seconds each. Suppose, however, that we have 12 minutes or 24 hours. Does a point of diminishing returns ever set in, or is it always better to spend all the time in one run?

For a partial answer, see Table X, which summarizes an experiment performed on the geometric graph of Figure 8. Fixing SIZEFACTOR at 1 and the starting temperature at a typical value corresponding to INITPROB = 0.4, we ran a sequence of annealing runs for various values of TEMPFACTOR (1,024 for TEMPFACTOR = 0.95, 512 for 0.9747, 256 for 0.9873, 128 for 0.99358, 64 for 0.99678, 32 for 0.99839, and 16 for 0.99920, where each value of TEMPFACTOR is approximately the square root of its predecessor). Normalizing the running times so that the TEMPFACTOR = 0.99920 runs averaged 100 units, we compared the quality of solutions expected for each value of TEMPFACTOR under various time bounds, assuming that as many runs as

Table IX
Dependence of Average Cutsize Found on SIZEFACTOR and TEMPFACTOR

SIZE FACTOR	TEMPFACTOR						
	0.4401	0.6634	0.8145	0.9025	0.9500	0.9747	0.9873
0.25	—	—	—	—	—	—	234.8
0.5	—	—	—	—	—	222.1	230.9
1	—	—	—	—	235.0	225.2	221.4
2	—	—	—	230.1	224.9	220.1	217.2
4	—	—	232.4	225.1	220.3	216.0	214.1
8	—	229.9	223.8	218.7	215.2	214.2	212.4
16	228.1	223.3	219.6	215.0	213.6	210.8	209.7
32	229.5	220.8	215.3	214.7	211.9	211.0	—
64	219.6	217.0	212.9	211.0	211.4	—	—
128	216.1	213.4	212.3	211.9	—	—	—
256	216.2	212.0	211.3	—	—	—	—
512	215.2	212.6	—	—	—	—	—
1,024	210.6	—	—	—	—	—	—

Table X
Results of Experiment Comparing the Effect of Allowing More Time Per Run Versus Performing More Runs for the Geometric Graph of Figure 8 (Cutsize as a Function of Time Available)

TEMP-FACTOR	NORMALIZED RUNNING TIME	TIME AVAILABLE 25	50	100	200	400
0.99920	100.0	—	—	51.1	42.7	35.5
0.99893	53.0	—	55.8	45.8	37.8	32.1
0.99678	28.5	64.6	53.2	43.5	36.6	30.8
0.99358	15.4	61.5	55.6	46.2	37.0	30.7
0.98730	8.6	64.6	55.3	47.1	40.0	33.7
0.97470	4.9	70.2	62.1	54.6	47.0	40.1
0.95000	2.9	71.6	63.8	56.3	49.6	43.5

possible within the bound were made and the best result taken. (The actual number of runs used was the integer nearest to *(TIME LIMIT)/(RUNNING TIME)*.) Note that a point of diminishing returns sets in at around TEMPFACTOR = 0.99678. It is better to run once at this TEMPFACTOR than a proportionately increased number of shorter runs at lower TEMPFACTORs, but it is also better to run 2 or 4 times at this value than a proportionately decreased number of longer runs at higher TEMPFACTORs.

It is interesting to note that, by analogy with the results displayed in Tables VIII and IX, the (SIZEFACTOR,TEMPFACTOR) pair (1, 0.99678) should correspond roughly to the (16, 0.95) pair we used in our experiments in the quality of solutions found. Similar experiments with SIZEFACTOR = 16 and TEMPFACTOR = 0.95, 0.9747, and 0.9873 supported the conclusion that multiple runs for the (16, 0.95) pair were to be preferred over fewer longer runs. Analogous experiments with our standard random graph G_{500} were less conclusive, however, showing no statistically significant difference between the three TEMPFACTORs when running time was taken into account. Thus, the tradeoff between ITERNUM and TEMPFACTOR appears to be another variable that can differ from application to application, and possibly, from instance to instance.

5. MODIFYING THE GENERIC ALGORITHM

In choosing the standard values for our parameters, we attempted to select values that yielded the quickest running time without sacrificing the quality of the solutions found. We were, however, limited in our alternatives by the fact that we were operating within the framework of our generic algorithm, which was designed for constructing prototype implementations rather than a final product. In this section, we consider some of the options available to us if we are prepared to modify the generic structure, and examine whether they offer the possibility of improved performance.

5.1. Cutoffs

One commonly used speedup option is the *cutoff*, included in Kirkpatrick's original implementations. Cutoffs are designed to remove unneeded trials from the beginning of the schedule. On the assumption that it is the number of moves accepted (rather than the number of trials) that is important, the processing at a given temperature is terminated early if a certain threshold of accepted moves is passed. To be specific, we proceed at a given temperature until *either* SIZEFACTOR $*$ N moves have been tried or CUTOFF $*$ SIZEFACTOR $*$ N moves have been accepted. This approach was orginally proposed in the context of annealing runs that started at high temperatures (INITPROB $\geq$ 0.9 rather than the INITPROB = 0.4 of our standard implementation). It is, thus, natural to compare the effect of using high starting temperatures and cutoffs versus the effect of simply starting at a lower temperature.

Figure 19 plots the (running time, cutsize) pairs for annealing runs made using both approaches. The points marked by $*$s were obtained from 10 runs each with CUTOFF = 1.0 (i.e., no cutoff), and with INITPROB $\in$ {0.1, 0.2, ..., 0.9} (the same data used to generate Figures 16 and 17). The points marked by 0s came from 20 runs each with INITPROB = 0.95 and CUTOFF $\in$ {0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625}.

There appears to be little in the figure to distinguish the two approaches. If there is any correlation present, it seems to be between running time and cutsize, no matter which method for reducing running time is used. More extensive experimentation might reveal some subtle distinctions, but tentatively we conclude that the two approaches are about equally effective.

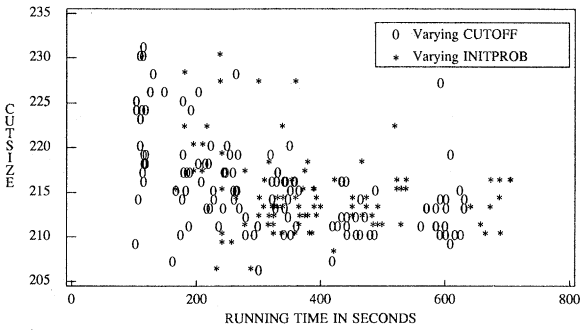

Figure 19. Comparison of the effects of cutoffs versus lowering INITPROB.

In particular, using INITPROB = 0.95 and CUTOFF = 0.125 is roughly equivalent in running time and quality of solution to our standard choice of INITPROB = 0.4 and CUTOFF = 1.0. Reducing CUTOFF below 0.125 has the same effect as reducing INITPROB below 0.4: running time is saved but the average solution quality deteriorates. Limited experimentation also indicated that little was to be gained by using cutoffs in conjunction with INITPROB = 0.4.

5.2. Rejection-Free Annealing

In the previous section, we investigated the effect of shortcuts at the beginning of the annealing schedule. Alternatively, one might attempt to remove unnecessary computations from the *end* of the schedule. At low temperatures almost all our time is spent in considering moves that we end up rejecting. Viewing this as wasted time, Green and Supowit (1986) propose that we reorganize the computation as follows. Compute for each possible move the probability p_i that it would be accepted if chosen. Let the sum of all these probabilities be P. We construct a probability distribution over the set of all moves where the probability of move i is p_i/P, select a move randomly according to this distribution, and accept it automatically. Green and Supowit show that this is equivalent to doing annealing in the ordinary way. Moreover, for the generalization of graph partitioning that they consider, the probabilities can be updated efficiently. Consequently, the procedure runs more quickly than the ordinary method as soon as the percentage of acceptances drops below some cutoff (11–13% in Green and Supowit's experiments). Although we did not investigate this approach ourselves, it appears that for low values of INITPROB, a time savings of up to 30% might be obtainable. Note, however, that this approach may not be generally applicable, as the efficient updating of the p_i's depends in large part on the nature of the cost function used.

5.3. Adaptive Cooling Schedules

Although much of the time spent by Annealing at very high and very low temperatures seems unproductive, we saw in Section 4.3 that the amount of time spent at temperatures *between* these limits had a large effect on the quality of solution obtained. The following alternative method for increasing this time was suggested by Kirkpatrick, Gelatt and Vecchi. Based on their reasoning about the physical analogy, they proposed spending more time at those temperatures where the current average solution value is dropping rapidly, arguing that more time was needed at such temperatures to reach equilibrium. To investigate this, we temporarily modified our generic algorithm to apportion its time amongst temperatures *adaptively*. At each temperature we repeat the following loop.

1. Run for SIZEFACTOR $*$ N trials, observing the best solution seen and the average solution value.
2. If either of these is better than the corresponding values for the previous SIZEFACTOR $*$ N trials, repeat.
   Otherwise, lower the temperature in the standard way.

This adaptive technique approximately tripled the running time for a given set of parameters. Its effect on solutions could not, however, be distinguished from that of tripling the running time by changing SIZEFACTOR or TEMPFACTOR. See Figure 20 for a display of solutions found using the adaptive method (indicated by 0s) and the standard method (indicated by $*$s) as a function of running time. SIZEFACTOR was fixed at 16 and each method was run for the range of values of TEMPFACTOR used in Tables VIII and IX, with the smallest value omitted in the nonadaptive case and the largest value omitted in the adaptive case. All other parameters were set to their standard values, with the starting temperature fixed at 1.3. Ten trials were performed for each value.

Note how solution quality correlates much more strongly with running time than with method (adaptive or nonadaptive). On the basis of these observations and similar ones for geometric graphs, we saw no need to add the extra complication of adaptive annealing schedules to our algorithm. Further studies of adaptive cooling may, however, be warranted, both for this and other problems. The approach we took, although easy to implement, is rather simple-minded. More sophisticated adaptive cooling schedules have been proposed recently in which the cooling rate is adjusted based on the standard deviation of the

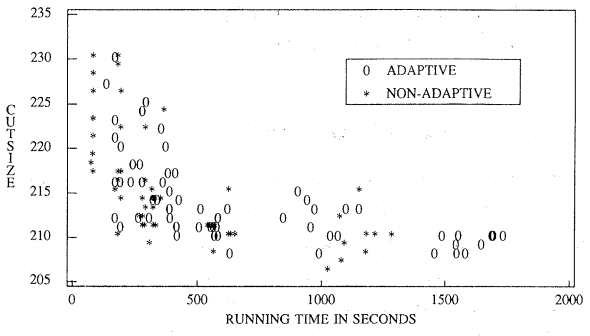

Figure 20. Results for adaptive and nonadaptive schedules as a function of running time.

solution values seen at the current temperature. Promising initial results for such approaches have been reported by Aarts and Van Laarhoven (1984), Lam and Delosme (1986), and Huang, Romeo, and Sangiovanni-Vincentelli (1986).

5.4. Alternative Cooling Schedules

Returning briefly to Figure 13, we note that some authors have suggested that the shape of this *cooling curve*, in particular the points at which its slope undergoes a major change, may in some way reflect a process analogous to the phase transitions that H_2O undergoes when it is cooled from a gaseous to a solid state. For these experiments, however, there seems to be a much better explanation for the shape of the curve. First note, as displayed in Figure 21, that there is a direct correlation between cutsize and the percentage of moves currently accepted when the cutsize is encountered, with cutsize improving essentially linearly as the probability of acceptance goes down. Since the percentage of acceptance presumably is determined by the temperature, it is natural to ask how our cooling schedule affects the probability of acceptance. Figure 22 displays the probability $P(t)$ that an uphill move of size 1 will be accepted when the temperature is $T_0(0.95)^t$, where T_0 is chosen so that $P(1) = 0.99$. Note how similar this curve is to that in Figure 13.

A natural question to ask is how important is the nature of this curve? What about other possible methods of temperature reduction, and the curves they inspire? To investigate this question, we considered three proposed alternatives.

The first is what we shall call *linear probability cooling*. We first fix INITPROB and the number C of temperature reductions that we wish to consider. Next we choose a starting temperature T_0 at which approximately INITPROB of the moves are accepted, letting

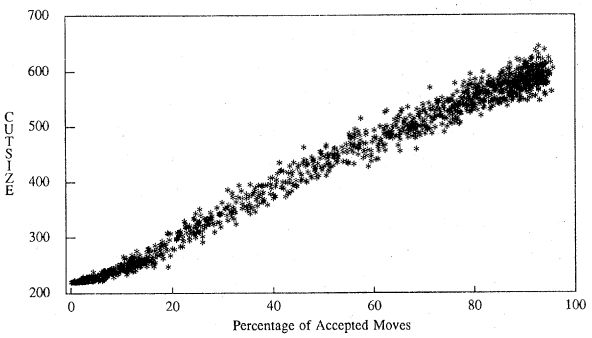

Figure 21. Correlation between cutsize and percentage of acceptance for the annealing run depicted in Figure 13.

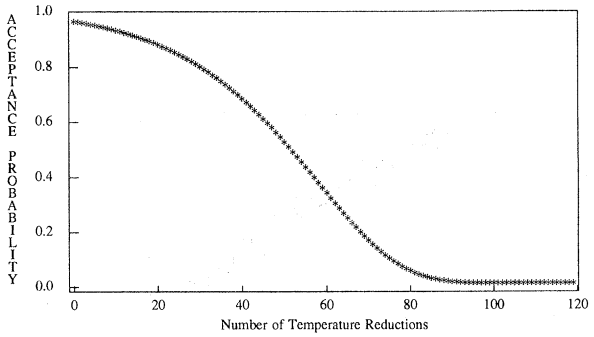

Figure 22. Probability that a given uphill move will be accepted as a function of the number of times the temperature has been lowered under geometric cooling.

Δ be the size of an uphill move accepted with probability INITPROB at temperature T_0. The ith temperature T_i is then chosen so that the probability p_i that an uphill move of size Δ will be accepted equals $((C - i)C) * INITPROB$. (This is guaranteed by setting $T_i = -\Delta/(\ln((C - i)C) * INITPROB)$, or more precisely the minimum of this figure and 0, since the solution may still be improving at temperature T_{100}, in which case, we will need additional temperatures to ensure freezing according to our standard criterion of five temperatures without improvement.)

This cooling method is intriguing in that it yields time exposures that approximate straight lines (see Figure 23), further confirming our hypothesis that the shapes of such time exposure curves are determined by the cooling schedule. Based on limited experiments, it appears that this cooling technique is far more sensitive to its starting temperature than is geometric cooling. For $T_0 = 1.3$ (corresponding to INITPROB = 0.4 for the graph G_{500}), we were unable to distinguish the results for this cooling method from those for the geometric method when the parameters were adjusted to equalize running time. In particular, for $C = 100$ and SIZEFACTOR = 8, the running time for linear probability cooling was roughly the same as that for geometric cooling under our standard parameters (346 seconds versus 330), while averaging a cutsize of 213.5 over 50 runs, compared to 213.3 for the standard parameters. However, if we set $T_0 = 11.3$ (corresponding to INITPROB = 0.9), the average cut increased significantly (to 220.4), even if we doubled C to take account of the fact that we were starting at a higher temperature. Recall that under geometric cooling, increasing INITPROB above 0.4, although it led to increased running time, had no significant effect on cutsizes found.

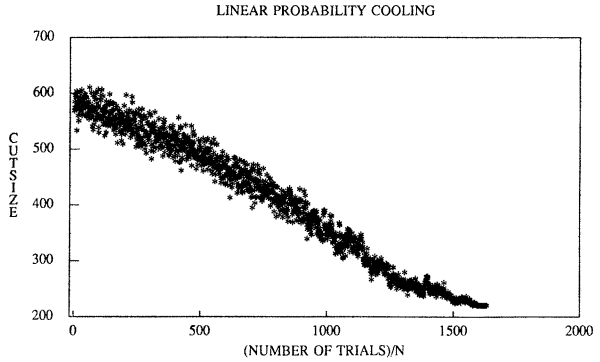

Figure 23. Time exposure for linear probability cooling.

The second cooling method we shall call *linear temperature cooling*, and it has apparently been used by some experimenters (e.g., Golden and Skiscim 1986). In this method we choose C and derive a starting temperature T_0 from INITPROB as before. The ith temperature T_i is then simply $((C - i)/C) * T_0$. Time exposures for this cooling method (see Figure 24) again reflect the curve of acceptance probability values. This technique proved to be equally sensitive to the choice of T_0. For our standard instance G_{500}, setting $C = 100$, SIZEFACTOR = 8, and $T_0 = 1.3$ again yielded an average cutsize over 50 runs of 213.5 (again using approximately the same time as geometric cooling), whereas increasing T_0 to 11.3 and C to 200 yielded an average cutsize of 219.9. Because of this lack of robustness, it appears that neither linear probability cooling nor linear temperature cooling is to be preferred to geometric cooling.

The final alternative we considered was suggested by the mathematical proofs of convergence for simulated annealing that were discussed in Section 1.4. In the papers cited there, it is shown that if one changes the temperature at every *step*, letting the temperature

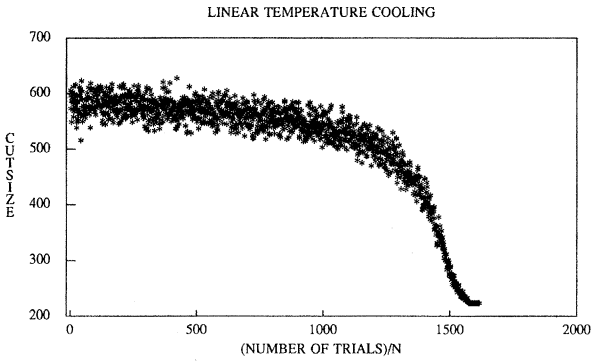

Figure 24. Time exposure for linear temperature cooling.

at step i be $C/(1 + \log(i))$ for come constant C, and if one is willing to wait long enough, then the process will almost surely converge to an *optimal* solution. The catch is that waiting long enough may well mean waiting longer than it takes to find an optimal solution by exhaustive search. It is, thus, unlikely that such results have anything to tell us about annealing as it can be implemented in practice.

Nevertheless, for completeness, we performed limited experiments with such *logarithmic cooling* schedules, executing logarithmic cooling runs of 1,000,000 steps for G_{500} and various values of C. This seems to give logarithmic cooling the benefit of the doubt, because a typical geometric cooling run on G_{500} using the standard parameters took about 400,000 steps. Note, however, that under logarithmic cooling the temperature only drops by about 5% during the last half of the schedule; indeed, after the first 0.1% of the schedule it drops only by a factor of 2. Thus, the choice of C seems to be crucial. Too high a value of C will cause the annealing run to finish at too high a temperature for the solution to be frozen in the normal sense. Too low a value will result in almost all the time spent at near-zero temperatures, thus yielding results that are little better than Local Opt. For G_{500}, the best value of C we could find was one that yielded a schedule for which the last 80% of the time was spent with an acceptance ratio between 2 and 3%, and for which the average cutsize found (over 20 runs) was 219.7, significantly above the 213.3 that our standard method averaged while using less than half the time.

5.5. Choosing Moves According to Random Permutations

In our description of the basic annealing process in Figure 3, the neighboring solution S' to be tested is simply chosen randomly from the set of all neighbors of the current solution. In terms of graph partitioning, we randomly choose a vertex as a candidate for moving from one side of the partition to the other, independently of all previous choices. Although this has the appeal of simplicity, there are reasons to think it might be inefficient. Suppose, for instance, that just one of the N vertices in the graph will yield an acceptable new solution if moved. Then there is a nonnegligible probability that we may have to perform substantially more than N trials before we encounter that special vertex. Thus, it has been suggested that, instead of picking our moves independently at each iteration, we should introduce enough dependence so that each vertex is chosen once in each successive block of N moves. This can be done while still maintaining a high degree of randomness, simply by choosing a

random permutation of the vertices at the beginning of each block, and using that permutation to generate the next N moves.

Using this modification, but otherwise making no changes from the standard parameters and starting time, we performed 100 annealing runs on our standard geometric graph G_{500}. The average running time was essentially unchanged, but the average cutsize found was 212.5, as compared to 213.3 for 1,000 runs of the standard algorithm. This seems significant because none of the 10 groups of 100 runs that made up that 1,000 yielded an average better than 212.9. In view of the results summarized in Table IX, the reduction in cutsize obtained by using permutations to generate moves seems to be almost as much as one would obtain by doubling the running time and staying with the standard method. This conclusion was further supported by experiments with the geometric graph of Figure 8. Here the average cutsize found during 100 trials dropped from 22.6 for the standard method to 20.4 using permutation generation, without a significant change in running time. If we used permutation generation but reduced the value of SIZEFACTOR from 16 to 8, the average cutsize went back up to 22.8 but we only used half the time of the standard implementation. Based on these limited experiments, it seems that move generation by permutation is a promising approach.

5.6. Using Better-Than-Random Starting Solutions

Many authors have suggested the possibility of using *better-than-random* starting solutions, and lower than normal starting temperatures, to obtain better results, or at least better running times. We studied this possibility, and our results indicate that the source of the better-than-random solutions may be of crucial significance.

We first studied the effect on annealing of using partitions generated by K-L, which are far better than random, as our starting solutions. We revised our Initial_Solution routine to return the result of K-L running on a random partition, rather than a random partition itself. (Since K-L works only on balanced partitions, we actually started from a random balanced partition, as we normally did for K-L, rather than a general random partition, as we normally did for annealing.) We then ran experiments analogous to those in Figures 16 and 17 to find an appropriate, presumably lower than normal, value of INITPROB. Surprisingly, it turned out that the boxplots for these good starts looked just like the boxplots for random starts in Figures 16 and 17. The best values of INITPROB were still 0.3 or 0.4 for our standard random graph G_{500}, and so no significant speedup could be expected. Furthermore, for G_{500} the average cutsize found over 100 runs was exactly the same as the average for random starts. There also seems to be no significant benefit to using K-L starts on the 16 standard random graphs covered by Tables II–V, although this conclusion is based only on 20 runs with and without the K-L starts for each graph. (We also did not attempt to determine optimal values of INITPROB for each graph separately, but stuck with the value of 0.4 indicated by our experiments with G_{500}.)

The situation for geometric graphs was somewhat better. Although, once again, the optimal value of INITPROB was around 0.4 and so no speedup could be obtained, the use of starting solutions generated by K-L yields better solutions on average, as seen in Table XI. As shown in the table, however, for these geometric graphs there proved to be an even better starting solution generator, a simple heuristic we shall call the *Line* Algorithm. This heuristic is tailored especially to the geometric nature of these instances. Recall that a geometric graph is based on a model in which vertices correspond to points in the unit square, and there is an edge between two vertices if and only if the vertices are sufficiently close to each other. In terms of this underlying model, one might expect a good partition to divide the unit square into two regions, and the size of the cut to be roughly proportional to the length of the boundary between the two

Table XI
Average Cutsizes Found for Line, K-L, and Annealing (the Latter Two With and Without Good Starting Solutions Compared to Best Cuts Ever Found[a]

|V|	$n\pi d^2$ 5	10	20	40	Algorithm
500	4	26	178	412	(Best Found)
	38.1	89.4	328.4	627.8	Line
	36.1	89.7	221.0	436.3	K-L
	12.5	45.0	200.8	442.8	Line + K-L
	21.1	65.8	247.0	680.3	Anneal
	22.2	52.4	192.4	507.4	K-L + Anneal
	12.9	48.6	217.0	501.1	Line + Anneal
1000	3	39	222	737	(Best Found)
	47.2	123.7	381.8	1128.7	Line
	70.7	157.5	316.7	857.8	K-L
	15.7	64.6	272.0	825.5	Line + K-L
	41.2	120.4	355.3	963.8	Anneal
	38.1	99.9	295.0	833.8	K-L + Anneal
	15.8	54.7	262.5	839.6	Line + Anneal

[a] These figures do not take running time into account, and thus, overestimate the relative efficacy of annealing.

regions. Since the shortest such boundary is a straight line, this suggests looking at those partitions that are formed by straight lines.

The Line Algorithm is given as input the coordinates of the points p_i, $1 \leq i \leq n$, in the model for a geometric graph and proceeds as follows.

1. Pick a random angle θ, $0 \leq \theta \leq \pi$.
2. For each point p_i, compute the y-intercept y_i of the straight line through p_i with a slope equal to $\tan(\theta)$.
3. Sort the points according to their values of y_i, and form a partition between the vertices corresponding to the first $n/2$ points and those corresponding to the last $n/2$.

Compared to our more sophisticated algorithms, the running time for the $O(n \log n)$ Line Algorithm was negligible, e.g., 1.5 seconds for the 1,000-vertex, expected degree 5 graph. (Note that by using a linear-time, median-finding algorithm instead of the sorting in Step 3, we could reduce the overall time to $O(n)$, although for the size of problems we consider here, the time saving is not worth the programming effort.)

Table XI shows the average cuts found for our eight standard geometric graphs by 20 runs of Annealing starting from random partitions, K-L partitions, and Line partitions. For comparison purposes, it also shows the best cuts ever found by any method and the averages of 1,000 runs of Line, 2,000 runs of K-L, and 1,000 runs of K-L from a Line start. Note that Line by itself is as good or better than K-L for the sparser graphs, but that the true value of Line appears to be as an initial partition generator. Even more intriguing than the averages reported in the table are the cutsizes of the *best* partitions found by the various algorithms for the 1,000-vertex graph with $n\pi d^2 = 5$: The best partition ever found by random start K-L has cutsize 26 (compared to 30 for random start Annealing), the best Line partition found had cutsize 19, and both Line + K-L and Line + Annealing found cuts of size 3. Moreover, with Line starts, it was possible to begin Annealing at a very low temperature (INITPROB = 0.025) without losing on solution quality, and so substantial speedups (as big as a factor of 5) were possible. (The combination of Line with K-L was also slightly faster than K-L by itself, no doubt because fewer K-L iterations were needed, given the good start.)

Note that the results in Table XI do not take running time into account. If one equalizes for running time, Line beats Annealing (usually substantially) on all but the 500-vertex graph with $n\pi d^2 = 20$, and beats K-L on all the graphs with $n\pi d^2 \leq 10$. Moreover, Line + K-L passes Line + Annealing on the two graphs where it does not already beat it *without* accounting for running time. (The especially good value for K-L + Annealing on the 500-vertex, $n\pi d^2 = 20$ graph appears to be a statistical anomaly, arising from the fact that the starting solutions generated by K-L for these 20 runs had a much better than normal percentage of very good cutsizes.)

We conclude from these experiments that there is value to be gained by using good starting solutions, but that the nature of the starting solutions can make a crucial difference. It is especially helpful if the starting solutions are in some sense *orthogonal* to the kinds of solutions generated by annealing, as is the case with the Line solutions for geometric graphs, which make use of geometric insights into the nature of the instances that are not directly available to a general algorithm like annealing that must work for *all* instances. (One might hypothesize that the reason that K-L starting solutions were also helpful for geometric graphs is that the detailed operation of K-L is adapted to take advantage of the local structure of geometric graphs in ways that annealing is less likely to find. See Section 7 for a brief description of how K-L works.) Moreover, good starts may be equally or more useful when used with approaches other than annealing.

The above results mirror practical experience we had with certain real-life instances. The real-life instances came from a related problem, that of *hypergraph* partitioning. In a hypergraph the *edges* are *sets* of vertices, not just pairs, and the cutsize for a partition $V = V_1 \cup V_2$ is the number of edges that contain vertices from both V_1 and V_2. A scheme for designing "standard-cell VLSI circuits," developed at AT&T Bell Laboratories and described by Dunlop and Kernighan, performs cell layout by repeated calls to a hypergraph partitioning algorithm. Traditionally the K-L algorithm has been used for this (it was originally defined in general hypergraph terms). On real circuits, it gives its best results when started from a partition provided by the circuit designers or a slightly randomized variant thereof. Such starting partitions were significantly better than the partitions typically found by K-L when it was started from a purely random partition. They made use of instance-specific *inside information*, just as the Line starting partitions did for our geometric instances of ordinary graph partitioning.

The same behavior was observed with an implementation of annealing for hypergraph partitioning. In a victory for our generic approach, this implementation was obtained from our graph partitioning implementation in a few hours by just making some minor changes to the problem-specific routines. If

started from a random partition, annealing was not competitive with the designers; starting from *their* partition, with a low initial temperature, it made substantial improvements. On certain graphs the improvements were more substantial than those made by K-L, on others they were less. Overall, the competition between the two algorithms for this application was inconclusive, so long as one did not take running time into account.

As a final comment on our experiments with good starting solutions, we note that they also indicate a further dimension to the superiority of Annealing over K-L when running time is ignored. For the graph G_{500}, the solutions found by Annealing averaged 9% better than the initial K-L solutions when INITPROB = 0.4. They were still averaging 1% better than the initial K-L solutions when INITPROB was set to 0.025, which reduced the running time of Annealing by a factor of 4. In contrast, when we performed limited experiments in which K-L was started from the final solution found by Annealing, K-L never yielded an improvement.

5.7. Approximate Exponentiation

In looking for other savings, a natural candidate for streamlining is the exponentiation $e^{-\Delta/T}$ that takes place once each time through the inner loop of the code. On a VAX 11-750, even with a floating point accelerator, this is an expensive operation. Under our standard parameters, Annealing will perform approximately 400,000 such exponentiations in handling G_{500}, and these will take almost one third of the running time. It thus seems appealing to use some other function than $e^{-\Delta/T}$ to determine the probability of acceptance. Although there are mathematical motivations for using the exponential, they only apply in certain asymptotic senses (e.g., see Anily and Federgruen, and Mitra, Romeo and Sangiovanni-Vincentelli). There is no a priori reason why some other, simpler to compute function might not serve just as well or better in the context of the algorithm as actually used. One appealing possibility is the function $1 - \Delta/T$, which involves just one division and at least *approximates* the exponential. It takes less than $1/25$ as much time to compute on our system, thus presumably offering about a 33% speedup. On the basis of 300 runs with this function and our standard parameters, we can confirm the speedup, and notice no significant difference in the quality of the solution (an average cutsize of 213.2 versus 213.3 for $e^{-\Delta/T}$).

We did not investigate this approximation further however, as an equivalent speedup can be obtained by an alternative and better approximation to $e^{-\Delta/T}$. This better approximation uses the following table lookup scheme. First note that the ratio between the smallest uphill move that has a nonnegligible chance of rejection and the largest uphill move that has a nonnegligible chance of acceptance is no more than 1,000 or so (an uphill move of size $T/200$ has an acceptance probability 0.9950 whereas one of size $5T$ has an acceptance probability 0.0067). Thus, to obtain the value of $e^{-\Delta/T}$ to within a half percent or so, all we need do is round $200\Delta/T$ down to the nearest integer, and use that as an index into a table of precomputed exponentials (if the index exceeds 1,000, we automatically reject). Implementing this latter scheme saved $1/3$ the running time, and had no apparent effect on quality of solution. We have used it in all our subsequent experiments with Annealing on other problems, choosing it over the linear approximation so that we could still claim to be analyzing what is essentially the standard annealing approach.

Had we used this approximation in our graph partitioning experiments, it would skew the results slightly more in Annealing's favor, but not enough to upset our main conclusions, even if we combine it with the two other major potential speedups uncovered in this study. Table XII shows the reduction in running time obtained by: 1) using table lookup exponentiation, 2) doing more generation by random permutation while halving the temperature length, as suggested in Section 6.5, and 3) combining a further reduction in the temperature length (SIZEFACTOR = 1) with a corresponding decrease in the cooling rate (TEMPFACTOR = 0.99358 = $(0.95)^{1/8}$) for *smoother* cooling, as suggested in Section 5.3. Five runs of this modified Annealing algorithm were performed for each of the 16 random graphs in our ensemble, and Table XII reports the ratios of the resulting average running times to the averages for our original implementation, as reported in Table IV.

Note that the running time was reduced by a factor of at least two in all cases, with significantly more improvement as the graphs became sparser and/or

Table XII
Average Speedups Using Approximate Exponentiation, Permutation Generation, and Smoother Cooling (Running Time Ratios)

	Expected Average Degree			
|V|	2.5	5.0	10.0	20.0
124	0.29	0.28	0.38	0.40
250	0.31	0.36	0.35	0.41
500	0.34	0.39	0.41	0.47
1000	0.34	0.37	0.39	0.49

Table XIII
Comparison of K-L and C-K-L With Sped-up Annealing (Percent Above Best Cut Ever Found)

	Expected Average Degree				
|V|	2.5	5.0	10.0	20.0	Algorithm
124	0.0	0.0	0.0	0.0	K-L's
	0.0	0.1	0.0	0.0	C-K-L's
	0.0	0.4	0.1	0.2	5 Anneals
250	0.1	0.9	0.5	0.2	K-L's
	0.0	0.4	0.6	0.2	C-K-L's
	1.8	0.6	0.3	0.0	5 Anneals
500	4.0	3.3	1.1	0.7	K-L's
	1.9	2.2	1.2	0.8	C-K-L's
	5.7	0.8	0.2	0.2	5 Anneals
1,000	5.2	4.5	1.8	1.0	K-L's
	2.0	3.5	1.6	1.1	C-K-L's
	3.2	0.8	0.2	0.1	5 Anneals

smaller. The running time reductions for our eight geometric graphs were similar, with ratios ranging from 0.33 to 0.45 in all but one case (the ratio for the 500-vertex geometric graph with $n\pi d^2 = 40$ was 0.76). These running time savings were obtained with no appreciable loss in solution quality: the average cut-sizes were roughly the same for both implementations. These speedups for both types of graphs alter the time-equalized comparison of Annealing and K-L reported in Tables V and VII, as fewer runs of K-L could be performed in the time it takes to do 5 anneals. The typical change, however, involves only a minor increase in K-L's expected excess over the best cutsize found, and K-L still has a significant lead over Annealing for all the geometric graphs and for the random 250- and 500-vertex graphs with expected degree 2.5. (To see the effect on random graphs, compare Table XIII with Table V.) Moreover, if we are willing to go to such efforts to optimize our annealing implementation, we should also consider attempts to improve on K-L by more traditional means. We do this in the next section.

6. MORE ON THE COMPETITORS

Simulated annealing is not the only challenger to the Kernighan-Lin graph partitioning throne. Alternative algorithms for graph and hypergraph partitioning recently have been proposed by a variety of researchers, including Fiduccia and Mattheyses (1982), Goldberg and Burstein (1983), Bui et al. (1984), Goldberg and Gardner (1984), Krishnamurthy (1984), Bui, Leighton and Heigham (1986), and Frankle and Karp (1986). Some of this work in fact has been stimulated by the reported success of annealing on certain graph partitioning problems, researchers having concluded that the true message in this relative success is not that annealing is good, but that K-L is a much poorer algorithm than previously thought.

We have done limited tests of two of the most promising approaches. The first is based on the Fiduccia-Mattheyses algorithm, a variant of K-L. The K-L algorithm operates only on balanced partitions, and is based on a repeated operation of finding the best pair of as-yet-unmoved vertices (one from V_1 and one from V_2) to interchange (best in the sense that they maximize the decrease in the cut, or if this is impossible, minimize the increase). If this is done for a total of $|V|/2$ interchanges, one ends up with the original partition, except that V_1 and V_2 are reversed. One then can take the best of the $|V|/2$ partitions seen along the way as the starting point for another pass, continuing until a pass yields no improvement. For a fuller description, see Kernighan and Lin.

Fiduccia and Matheyses (F-M) proposed to speed up the process by picking just the best *single* vertex to move at each step. This reduces the number of possibilities from $|V|^2$ to $|V|$ and, with the proper data structures (adjacency lists, buckets, etc.) can reduce the total worst case running time (per pass) to $O(|V| + |E|)$ from what looks like $\Omega(|V|^3)$ for K-L. In practice, this speedup is illusory, as K-L runs in time $O(|V| + |E|)$ per pass in practice when implemented with the same proper data structures, and the two algorithms had comparable running times in our limited tests. Nor were we able to get F-M to outperform K-L in the quality of solutions found. F-M was actually slightly worse under the standard implementation in which vertices are chosen alternately from V_1 and from V_2, to ensure that every other partition encountered is in balance. If instead we choose to move the best vertex in *either* V_1 or V_2, and use the *imbalance squared* penalty function of our Annealing algorithm, F-M improved to parity with K-L, but no better. (As remarked in Section 4.3, local optimization based on this penalty function is substantially better than local optimization based on pairwise interchanges: The average of 1,000 runs of the former on G_{500} was 276 versus 290 for the latter.)

The second approach has been embodied in algorithms due to Goldberg and Burstein and to Bui, Leighton, and Heigham, and involves coalescing vertices to form a smaller graph, and applying K-L to this. Based on our implementation of both algorithms, the Bui, Leighton, and Heigham algorithm seems to be superior and can offer a significant improvement over basic K-L. In this algorithm, one first finds a maximal matching on the vertices of the graph, and

forms a new graph by coalescing the endpoints of each edge in the matching (or all but one of them, if the number of edges in the matching is not divisible by 4). The result is a graph with an even number of vertices, upon which K-L is performed. The resulting partition is expanded by uncoalescing the matched vertices, and, if necessary, modified by random shifts so that it becomes a balanced partition of the original graph. This is then used as the starting solution for a run of K-L on the entire graph, the result of which is the output of the algorithm. We shall refer to this algorithm as *coalesced Kernighan-Lin* and abbreviate it as C-K-L.

Despite the two calls to K-L, the running time of C-K-L is less than twice that of K-L, ranging between 1.1 and 1.9 times that for K-L by itself on our test graphs (the first call to K-L is on a smaller graph, and the second is from a good starting partition). The excess over K-L's running time tends to go up as the density of the graph increases. Taking this increased running time into account, however, C-K-L outperforms basic K-L on all our geometric test graphs and on the sparser of our random ones. It did not outperform Line + K-L on the geometric graphs, however. Table XIII is the analog of Table V for our test bed of random graphs. Both K-L and C-K-L are compared to our original estimate for the best of 5 anneals, with the time equalization taking into account the speedups for Annealing reported in Table XII. The K-L data are derived from our original suite of 2,000 runs per graph; data for C-K-L are based on 1,000 runs per graph.

Note that C-K-L dominates our sped-up Annealing implementation on all the graphs with expected average degree 2.5 (except the smallest, where all three algorithms are tied). In comparison, K-L loses out on the 1,000-vertex graph of this type, even when compared to the slower Annealing implementation, as in Table V. Annealing still seems to be pulling away, however, as the graphs become larger and denser.

Finally, all three algorithms (K-L, C-K-L, and Annealing) can be beaten badly on special classes of graphs. We have seen the efficacy of the Line Algorithm for geometric graphs. Bui et al. report on an approach based on network flow that almost surely finds the optimal cut in certain regular graphs with unique optimal cuts. Neither Annealing nor K-L matches its performance on such graphs. For especially sparse graphs, another possibility suggests itself. Such graphs may not be connected, and it is thus possible that some collection of connected components might contain a total of exactly $|V|/2$ vertices, yielding a perfect cut. Theoretically this is unlikely unless the graph is *very* sparse; normally there should still be one *monster* component that contains most of the vertices, and this was indeed the case for all the test graphs studied. We were, however, able to generate a 500-vertex geometric graph with this property by taking $d = 0.05$ (expected average degree slightly less than 4). This graph had an optimum cutsize of 0 that was found by using a connected components algorithm with an $O(n^2)$ dynamic programming algorithm for solving the resulting subset sum problem. Neither K-L nor Annealing, however, ever found such a cut, despite thousands of runs of the former and hundreds of the latter.

7. CONCLUSIONS

In light of the above, simulated annealing seems to be a competitive approach to the graph partitioning problem. For certain types of random graphs, it appears to beat such traditional heuristics as Kernighan-Lin, as well as more recent improvements thereon, even when running time is taken into account. It was substantially outclassed on other types of graphs, however. Generalizing from the results we observed for random and geometric graphs, it appears that if the graph is particularly sparse or has some local structure, it may well be better to spend an equivalent amount of time performing multiple runs of K-L or C-K-L, or using heuristics specially tuned to the instances at hand.

In addition to evaluating annealing's performance on the graph partitioning problem, our experiments may also provide some preliminary insight into how best to adapt our generic annealing algorithm to other problems. In particular, we offer the following observations.

Observation 1. To get the best results, long annealing runs must be allowed.

Observation 2. Of the various ways to increase the length of an annealing run, adding time to the beginning or end of the schedule does not seem to be as effective as adding it uniformly throughout the schedule. The latter can be accomplished by increasing TEMPFACTOR, increasing SIZEFACTOR, or using adaptive temperature reduction. It is not clear which of these methods is to be preferred, although a TEMPFACTOR increase seems to yield a slight running time advantage in our implementation.

Observation 3. It may not be necessary to spend much time at very high temperatures (ones where almost all moves are accepted). One can reduce the time spent

at such temperatures by using cutoffs, or simply by starting at a lower temperature. It is not clear if it makes a difference which technique is used, so long as the value of the cutoff/starting temperature is properly chosen. For this, experimentation may be required.

Observation 4. Simple minded adaptive scheduling appears to yield no improvement beyond that to be expected due to the increase in overall running time it provides. We do not rule out the possibility that more sophisticated adaptive schedules or schedules hand-tuned to particular types of instances might be more effective, especially if instances exhibit evidence of the "phase transitions" alluded to by Kirkpatrick, Gelatt and Vecchi. No such transitions were evident in our graph partitioning instances, however. For these, the shape of a time exposure of current solution values seems to be determined mostly by the curve of declining move probabilities, with no unexplained irregularities that an adaptive scheduler might attempt to exploit.

Observation 5. There seems no reason to replace the standard geometric cooling method by any of the *nonadaptive* alternatives we have examined (logarithmic cooling, linear temperature cooling, etc.).

Observation 6. It appears that better solutions can be found subject to a given bound on running time, if one does not simply generate candidate moves one at a time, independently, but instead uses random permutations to generate sequences of N successive moves without repetition.

Observation 7. Even with long runs, there can still be a large variation in the quality of solutions found by different runs. However, up to a certain point, it seems to be better to perform one long run than to take the best of a time-equivalent collection of shorter runs.

Observation 8. There can be an advantage to starting at a good solution rather than a randomly generated one (an advantage in quality of solution, running time, or both), but this depends strongly on the nature of the *good* solution. Starting solutions that take advantage of some special structure in the instance at hand seem to be preferable to those obtained by general heuristics.

Observation 9. Replacing the computation of the exponential $e^{-\Delta/T}$ with a table lookup approximation seems to be a simple way to speed up the algorithm without degrading its performance.

Observation 10. The best values of the annealing parameters may depend not only on the problem being solved, but also on the type and size of instance at hand. One must beware of interactions between the generic parameters, and between these and any problem specific parameters that may exist in the implementation. Given this warning, however, the generic parameter values we derived for our graph partitioning implementation seem like a good starting point, assuming they result in feasible running times.

Observation 11. In adapting annealing to a particular problem, it may pay to expand the definition of solution. One can allow violations of some of the basic constraints of the problem definition, so long as a penalty for the violation is included in the cost function. This allows for a smoother solution space in which local optima are easier to escape. The smaller the penalty, the smoother the space, and surprisingly small penalties may still be enough to ensure that final solutions are legal, or close to it.

Although based on the study of a single application of annealing, these observations have been supported by our subsequent work on other applications. In particular, they will be used and elaborated on in the two companion papers (Johnson et al. 1990a, b), which report on our experiments adapting simulated annealing to graph coloring, number partitioning and the traveling salesman problem.

As a service to readers who would like to replicate or improve upon our graph partitioning experiments and desire a common basis for comparison, we are prepared, for a limited time, to make electronic copies available of the graphs used as instances in this study. Interested readers should contact the first author (electronic mail address: dsj@research.att.com).

ACKNOWLEDGMENT

The authors thank Phil Anderson for providing the initial stimulation for this study, Scott Kirkpatrick for his help in getting us started, Jon Bentley, Mike Garey, Mark Goldberg, Martin Grötschel, Tom Leighton and John Tukey for insightful discussions along the way, and the Referees for suggestions that helped improve the final presentation.

REFERENCES

AARTS, E. H. L., AND P. J. M. VAN LAARHOVEN. 1985. A New Polynomial-Time Cooling Schedule. In *Proc. IEEE Int. Conf. on CAD* (*ICCAD 85*), pp. 206–208, Santa Clara, Calif.

ANILY, S., AND A. FEDERGRUEN. 1985. Simulated Annealing Methods With General Acceptance Probabilities. Preprint. Graduate School of Business, Columbia University, New York.

BONOMI, E., AND J.-L. LUTTON. 1984. The *N*-City Travelling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm. *SIAM Review* **26,** 551–568.

BUI, T. 1983. On Bisecting Random Graphs. Report No. MIT/LCS/TR-287, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Mass.

BUI, T., S. CHAUDHURI, T. LEIGHTON AND M. SIPSER. 1984. Graph Bisection Algorithms With Good Average Case Behavior. In *Proceedings 25th Ann. Symp. on Foundations of Computer Science*, 181–192. Los Angeles, Calif.

BUI, T., T. LEIGHTON AND C. HEIGHAM. 1986. Private Communication.

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

COLLINS, N. E., R. W. EGLESE AND B. L. GOLDEN. 1988. Simulated Annealing: An Annotated Bibliography. Report No. 88-019, College of Business and Management, University of Maryland, College Park, Md.

DUNLOP, A. E., AND B. W. KERNIGHAN. 1985. A Procedure for Placement of Standard-Cell VLSI Circuits. *IEEE Trans. Computer-Aided Design* **4,** 92–98.

EL GAMAL, A. A., L. A. HEMACHANDRA, I. SHPERLING AND V. K. WEI. 1987. Using Simulated Annealing to Design Good Codes. *IEEE Trans. Inform. Theory* **33,** 116–123.

FIDUCCIA, C. M., AND R. M. MATTHEYSES. 1982. A Linear-Time Heuristic for Improving Network Partitions. In *Proc. 19th Design Automation Conference*, pp. 175–181, Las Vegas, N.M.

FRANKLE, J., AND R. M. KARP. 1986. Circuit Placements and Cost Bounds by Eigenvector Decomposition. In *Proc. IEEE Int. Conf. on CAD (ICCAD 86)*, pp. 414–417, Santa Clara, Calif.

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

GAREY, M. R., D. S. JOHNSON AND L. STOCKMEYER. 1976. Some Simplified NP-Complete Graph Problems. *Theor. Comput. Sci.* **1,** 237–267.

GELFAND, S. B., AND S. K. MITTER. 1985. Analysis of Simulated Annealing for Optimization. In *Proc. 24th Conf. on Decision and Control*, 779–786, Ft. Lauderdale, Fla.

GEMAN, S., AND D. GEMAN. 1984. Stochastic Relaxation, Gibbs Distribution, and the Bayesian Restoration of Images. *IEEE Proc. Pattern Analysis and Machine Intelligence* **PAMI-6,** 721–741.

GIDAS, B. 1985. Non-Stationary Markov Chains and Convergence of the Annealing Algorithm. *J. Statis. Phys.* **39,** 73–131.

GOLDBERG, M. K., AND M. BURSTEIN. 1983. Heuristic Improvement Technique for Bisection of VLSI Networks. In *Proc. IEEE International Conf. on Computer Design*, 122–125, Port Chester, N.Y.

GOLDBERG, M. K., AND R. D. GARDNER. 1984. Computational Experiments: Graph Bisecting. Unpublished Manuscript.

GOLDEN, B. L., AND C. C. SKISCIM. 1986. Using Simulated Annealing to Solve Routing and Location Problems. *Naval. Res. Logist. Quart.* **33,** 261–279.

GREEN, J. M., AND K. J. SUPOWIT. 1986. Simulated Annealing Without Rejecting Moves. *IEEE Trans. Computer-Aided Design* **CAD-5,** 221–228.

HINTON, G. E., T. J. SEJNOWSKI AND D. H. ACKLEY. 1984. Boltzmann Machines: Constraint Satisfaction Networks That Learn. Report No. CMU-CS-84-119, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa.

HUANG, M. D., F. ROMEO AND A. SANGIOVANNI-VINCENTELLI. 1986. An Efficient General Cooling Schedule for Simulated Annealing. In *Proc. IEEE Int. Conf. on CAD (ICCAD 86)*, 381–384, Santa Clara, Calif.

JEPSEN, D. W., AND C. D. GELATT, JR. 1983. Macro Placement by Monte Carlo Annealing. In *Proc. International Conf. on Computer Design*, 495–498, Port Chester, N.Y.

JOHNSON, D. S., C. R. ARAGON, L. A. MCGEOCH AND C. SCHEVON. 1990a. Optimization by Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partitioning). *Opns. Res.* (to appear).

JOHNSON, D. S., C. R. ARAGON, L. A. MCGEOCH AND C. SCHEVON. 1990b. Optimization by Simulated Annealing: An Experimental Evaluation, Part III (The Traveling Salesman Problem). (In Preparation.)

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

KIRKPATRICK, S. 1984. Optimization by Simulated Annealing: Quantitative Studies. *J. Statis. Phys.* **34,** 975–986.

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

KRISHNAMURTHY, B. 1984. An Improved Min-Cut Algorithm for Partitioning VLSI Networks. *IEEE Trans. Computers* **C-33,** 438–446.

LAM, J., AND J. M. DELOSME. 1986. Logic Minimization Using Simulated Annealing. In *Proc. IEEE Int. Conf. on CAD (ICCAD 86)*, 348–351, Santa Clara, Calif.

LUNDY, M., AND A. MEES. 1986. Convergence of the Annealing Algorithm. *Math. Prog.* **34,** 111–124.

Math of a Salesman. 1982. *Science* **3:9,** 7–8 (November).

MCGILL, R., J. W. TUKEY AND W. A. DESARBO. 1978. Variations on Box Plots. *Am. Stat.* **32:1,** 12–16.

METROPOLIS, W., A. ROSENBLUTH, M. ROSENBLUTH, A. TELLER AND E. TELLER. 1953. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **21,** 1087–1092.

MITRA, D., F. ROMEO AND A. SANGIOVANNI-VINCENTELLI. 1986. Convergence and Finite-Time Behavior of Simulated Annealing. *J. Advan. Appl. Prob.* **18,** 747–771.

NAHAR, S., S. SAHNI AND E. SHRAGOWITZ. 1985. Experiments with Simulated Annealing. In *Proceedings 22nd Design Automation Conference*, 748–752, Las Vegas, N.M.

ROWEN, C., AND J. L. HENNESSY. 1985. SWAMI: A Flexible Logic Implementation System. In *Proceedings 22nd Design Automation Conference*, 169–175, Las Vegas, N.M.

SASAKI, G. H., AND B. HAJEK. 1988. The Time Complexity of Maximum Matching by Simulated Annealing. *J. Assoc. Comput. Mach.* **35,** 387–403.

Statistical Mechanics Algorithm for Monte Carlo Optimization. 1982. *Physics Today* **35:5,** 17–19 (May).

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* **CAD-2,** 215–222.

WHITE, S. R. 1984. Concepts of Scale in Simulated Annealing. In *Proc. International Conf. on Computer Design*, 646–651, Port Chester, N.Y.