Local Search for the Retrieval Layout Problem

Lenwood Heath and Joseph W. Lavinus

TR 93-28

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

August 27, 1993

Submitted to OR SPEKTRUM, 1993.

Local Search for the Retrieval Layout Problem*

Lenwood S. Heath
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061–0106, USA
FAX: (703) 231–6075

Joseph W. Lavinus
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903–2442, USA
FAX: (804) 982–2114

Running Title:
Retrieval Layout Problem

*The authors gratefully acknowledge the support of National Science Foundation grant IRI–9116991.

Abstract

An information graph is a data representation for object-oriented databases in which each object is a vertex and each relationship between objects is an edge. The retrieval layout problem is to arrange the storage of an information graph on a physical medium so as to use storage efficiently and to allow rapid navigation along paths in the graph. This paper describes an empirical study of the performance of various local search heuristics for the retrieval layout problem. The heuristics examined are local optimization, simulated annealing, tabu search, and genetic algorithms. In addition, the hierarchical hybrid approach is introduced, in which the problem is recursively decomposed into small pieces, to which local search heuristics are then applied.

Zusammenfassung

Ein Informationsgraph ist eine Datenrepräsentation für objektorientierte Datenbanken, in der jedes Objekt als Knoten und jedes Verhältnis zwischen Objekten als Kante dargestellt wird. Das Problem des Wiederauffindens besteht darin, wie man den Informationsgraphen auf einem physischen Medium speichert, damit der Platz gut ausgenulzt wird und der Graph schnell durchgesucht werden kann. Diese Arbeit berichtet über die Ergebnisse einer empirischen Untersuchung der Heuristiken verschiedener lokalen Suchverfahren. Untersucht werden lokale Optimierung, simulierte Abkühlung, Tabu Search, und genetische Algorithmen. Zusätzlich wird das hierarchische Mischverfahren vorgestellt, in dem das Problem rekursiv in kleinere Stücke zerteilt wird, woran dann lokale Suchheuristiken angewandt werden.

Key words: local search, simulated annealing, tabu search, genetic algorithms, graph layout.

Schlüsselwörter: lokales Suchverfahren, simulierte Abkühlung, Tabu Search, genetische Algorithmen, Graphenauslegung.

1 Introduction

Heath and Lavinus [13] present the *information graph* as a representation for an object-oriented database (see also Parunak [18]). Such a graph is an undirected graph in which the vertices hold pieces of information (objects) and an edge represents a relationship between the objects stored in its endpoints. Each vertex in an information graph has an associated size sufficient to store the representation of the corresponding object. We imagine the database stored on a linear medium with random access slower than sequential access; examples of such media include hard-disk drives and CD-ROMs. Since these media cannot provide efficient random retrieval of individual small objects, the objects are stored in fixed-size *pages;* each page typically holds a number of objects.

The key problem that arises is the assignment of vertices in the information graph to pages of the medium. This assignment should strive for space-efficient storage of the information graph while supporting time-efficient retrieval of its contents. The resulting optimization problem is the *retrieval layout problem*: find a layout of the vertices (an assignment of vertices to pages) of an information graph that achieves space efficiency (few pages are used) and time efficiency (retrieval of neighboring vertices is frequently within the current page or along short edges). The requirement of space efficiency alone is the classic bin packing problem [15]. The additional requirement of time efficiency raises a significant new optimization issue.

An objective function to measure the quality of a layout with respect to retrieval layout must reflect both requirements. Lavinus [17] considers several candidate objective functions to identify ones that are good predictors of space and time efficiency. He finds two useful objective functions: number of edges between pages and total length of edges between pages. He finds that each objective function has strengths and weaknesses but that typically the two are in agreement. Hence, in this paper we consider only one objective function: number of edges between pages.

The optimization problem that results from this choice of objective function is, not surprisingly, NP-complete (see Garey and Johnson [10], page 209). Hence we turn to the consideration of heuristics for the retrieval layout problem, including four local search heuristics: local optimization, simulated annealing, tabu search, and genetic algorithms. In addition, we test a hierarchical hybrid approach consisting of recursive decomposition combined with more expensive heuristics such as simulated annealing and tabu search. Finally, for purposes of comparison, we include a non-local search heuristic called connectivity traversal, devised by Lavinus [17] for the retrieval layout prob-

lem; this is a fast heuristic that consistently produces good solutions. We report on our empirical evaluation of these heuristics by testing with various input graphs, including actual databases and two classes of randomly generated graphs. We are able to draw conclusions about the suitability of each heuristic for the retrieval layout problem for both moderate and large information graphs.

Since real information graphs are very large, we test these heuristics on large graphs. Previous experimentation with graph layout heuristics has been limited to graphs with 1,000 or fewer vertices [17]. We assume that the graphs are fairly sparse, i.e., have low average degree, as this is a typical characteristic of actual information graphs that we have encountered.

The remainder of the paper is organized as follows. Section 2 introduces some notation and formally defines the retrieval layout problem. Section 3 briefly describes each of the heuristics. Section 4 describes the input graphs used, details the experiments performed, and reports the results of these experiments. Finally, Section 5 offers conclusions.

2 Definitions

An *information graph* is an undirected graph $G = (V, \rho, E)$ where

- V is the set of vertices;
- The function $\rho : V \rightarrow \mathbb{Z}$ gives the *size* of a vertex in bytes; and
- E is the set of edges.

The *size* of a subset of V is the sum of the sizes of its vertices. The *page size* is a constant P that is at least as large as the size of any vertex. A *layout of* G is a function $\lambda : V \rightarrow \{1, 2, \ldots, |V|\}$

subject to the constraint that

$$\sum_{\lambda(i)=j} \rho(i) \leq P,$$

for $1 \leq j \leq |V|$. The *page utilization*

$$R(\lambda) = |\{j \mid \lambda^{-1}(j) \neq \emptyset\}|$$

is the number of non-empty pages used by λ. Without loss of generality, we may assume that λ assigns at least one vertex to page 1 and assigns no vertex to a page numbered greater than $R(\lambda)$. The *length* of an edge (u, v) is simply:

$$\Delta_\lambda(u, v) = |\lambda(v) - \lambda(u)|,$$

the distance between the pages of its endpoints. With respect to the retrieval layout problem, the cheapest edge is one that has both of its endpoints in the same page; that is, one with length 0.

The study by Lavinus [17] determines that two objective functions on layouts that correlate with retrieval efficiency are number of interpage edges (the non-cheap edges) and total edge length. In this paper, we consider only the first objective function. The *number of interpage edges* in a layout λ is

$$\mathcal{NI}(\lambda) = |\{(u, v) \in E \mid \Delta_\lambda(u, v) \neq 0\}|.$$

We also call $\mathcal{NI}(\lambda)$ the *objective value* of λ.

3 Heuristics

This section briefly describes the heuristics we examine. The first four heuristics—local optimization, simulated annealing, tabu search, and genetic algorithms—are pure local search algorithms. Each of these starts from the view that any layout can be modified in a small number of simple ways to obtain new layouts that are its *neighbors*. The first three heuristics always remember a single current layout and examine only the neighbors of the current layout in one step; if a heuristic chooses to make a neighbor the new current layout, it performs a *move* to the neighbor. By contrast, a genetic algorithm maintains a population of layouts and each generation of the algorithm involves calculating a new population. The remaining two heuristics are not local search algorithms. Connectivity traversal is a sophisticated kind of greedy algorithm, while hierarchical hybrid is a hybrid approach based on recursive decomposition. Further exposition and implementation details for each of these heuristics can be found in Lavinus [17].

Local optimization. Local optimization is the simplest local search heuristic; its aim is to move steadily downhill to some local minimum of the objective function. Initially some random layout is chosen as the current layout. At each step, the heuristic moves to the neighbor of the current layout with the smallest objective value. In our implementation, each move takes a vertex from its current page and places it in a different page, provided there is space in the new page for the vertex. The effect of any move on the objective function $\mathcal{NI}$ can be computed in constant time, and thus a move can be made in constant time.

Simulated annealing. Simulated annealing [21] is a well-known local search heuristic designed to avoid entrapment in globally poor local optima by the use of the notion of *temperature*. At any step, simulated annealing examines only one neighbor of the current layout and decides on one of two bases whether to take that move. First, as in local optimization, improving moves are always taken. Second, moves that increase the objective value of the layout may also be taken with probability directly dependent on the temperature and inversely dependent on the magnitude of the move. The temperature is slowly decreased, and thus the solution converges to some locally optimal value, which is typically a much better solution than one produced by local optimization. Our implementation is similar to that of Johnson, Aragon, McGeoch, and Schevon [16].

Tabu search. Tabu search [11] relies on two mechanisms: a *tabu list* of forbidden moves and *aggressiveness*. Aggressiveness means that rather than examining one randomly selected neighbor at each step, as simulated annealing does, tabu search examines all neighbors of the current layout and selects the move, among those not on the tabu list, that results in the best objective value. This move may not result in a decrease in objective value if the current move is a local minimum or if all moves that would improve the objective value are tabu. The tabu list, which consists of the last t moves made for some constant t, prevents the heuristic from cycling in and out of a local optimum. The heuristic allows a tabu move to override its tabu status if it results in a layout with lower objective value than any seen so far. Tabu search halts when a certain number of moves have elapsed without finding a new best layout or when the total number of moves exceeds a constant m. Thus, the worst-case time complexity of tabu search is $O(m(R(\lambda) \cdot |E| + t))$ (though in practice this bound is quite pessimistic).

Genetic algorithms. A genetic algorithm [12] maintains a current generation of several layouts, which are mated with one another by means of a crossover operator such that solutions with lower objective values are more likely to survive to the next generation. We implement genetic algorithms for the retrieval layout problem using two different crossover operators: *partially mapped crossover (PMX)* [8] and *order crossover (OX)* [20].

Connectivity traversal. Connectivity traversal is a linear-time heuristic designed specifically for the retrieval layout problem [17]. Connectivity traversal fills pages 1 through $R(\lambda)$ one at a time. The initial vertex assigned to page 1 is chosen uniformly at random, and subsequent vertices are assigned to pages as pages 1 through $R(\lambda)$ are filled. At each step, the next vertex chosen is the one that is most heavily connected to those vertices already assigned to the current page.

Hierarchical hybrid. Hierarchical hybrid is a heuristic technique that combines some other heuristic with recursive decomposition. A recursive decomposition heuristic partitions the vertex set of an information graph into two subsets of roughly equal size, and then recursively partitions the subgraphs induced by these subsets in the same manner, until some threshold on vertex set size is reached. A typical recursive decomposition heuristic aims to cut as few edges at each step of the recursion as possible. We rely on the recursive decomposition heuristic of Fiduccia and Mattheyses [9] to produce good recursive decompositions. If the threshold for stopping the recursion is so small that each vertex set fits in a single page, we obtain a pure heuristic for the retrieval layout problem. However, this heuristic is consistently outperformed by connectivity traversal [17]. Improved performance can sometimes be obtained by setting the threshold to a small multiple of

P and then applying one of the previous heuristics to each of the subgraphs induced by the vertex subsets. This is our hierarchical hybrid approach.

4 Experiments

Since local search heuristics have unpredictable running times and no guarantee on solution quality, we use empirical testing to estimate the time complexity and effectiveness of these heuristics. We first specify the inputs we selected for this testing. We then describe the progression of tests we performed and report the results.

4.1 Inputs

As inputs, we use an actual database as well as two information graphs selected from certain classes of randomly generated graphs. The standard class of random graphs [2] does not accurately model real information graphs and also has optimal layouts that are not much better than random layouts [14]. However, two other classes of randomly generated graphs—geometric graphs [16] and gravitational graphs [17]—more accurately model actual information graphs.

An element of the class $U_{n,d}$ of n-vertex, distance-d *geometric graphs* is generated as follows. Each vertex is a point chosen uniformly at random in the unit square. An edge exists between two points u and v if $\mathrm{L}_2(u,v) < d$, where L_2 is the Euclidean distance between points. A more sophisticated model is obtained if each vertex v is additionally assigned a random *mass* $m(v)$ drawn from a lognormal distribution [1]. An element of the class $M_{n,d}$ of n-vertex *gravitational graphs* has an edge between two vertices u and v if $\mathrm{L}_2(u,v)/\max\{m(u),m(v)\} < d$. In a gravitational graph,

there tend to be a few vertices of large mass and correspondingly high degree, which seems to be a characteristic of real information graphs.

For our testing, we selected three large information graphs as inputs. The first graph is a bibliographic database drawn from *Communications of the ACM*, which contains 6,093 vertices and 65,438 edges; this graph is henceforth referred to as *CACM*. The size of vertices in CACM ranges between 5 and 270 bytes. The other two graphs are a geometric graph from $U_{10000,0.01}$ and a gravitational graph from $M_{10000,0.01}$. The size of each vertex in these two graphs is 1. For all testing, the page size is fixed at $P = 8192$, which is a typical page size found in current object-oriented databases [3, 4].

4.2 Heuristic Performance

First-cut testing. One heuristic *dominates* another heuristic on a particular input if it is both faster and produces a better layout for that input. In initial or first-cut testing, we run each of six pure heuristics (including two genetic algorithms) on each of the three input graphs to determine whether any heuristic is consistently dominated by some other heuristic. Figures 1 and 2 plot the performance of each heuristic versus its running time on the three first-cut graphs. The heuristics are represented by the single-letter mnemonics listed in Table 1. Each local search heuristic is initialized with a random layout. Each heuristic is run at least 10 times, and beyond that, a sufficient number of times to reduce the variances in objective value and running time to less than 1% of the mean. The dashed line near the top of each plot indicates the value of $\mathcal{NI}$ for a random layout, that is, the value we can obtain essentially for free. The testing platform is a DECstation$^{\text{TM}}$ 5000/200 with 40Mb of main memory.

Heuristic	Mnemonic	Heuristic	Mnemonic
Connectivity Traversal	C	Local optimization	L
Simulated annealing	S	Tabu search	T
Genetic algorithm (PMX)	G	Genetic algorithm (OX)	O

Table 1: The mnemonics used for each heuristic.

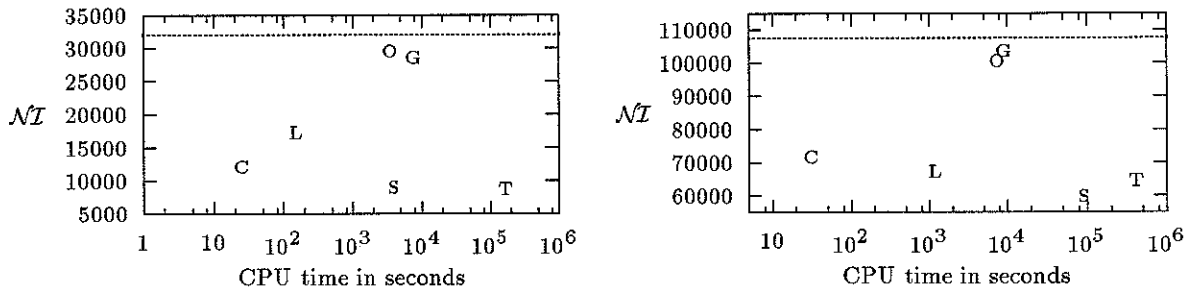

Figure 1: $\mathcal{NI}$ versus running time for all heuristics on CACM (left) and $M_{10000,0.01}$ (right).

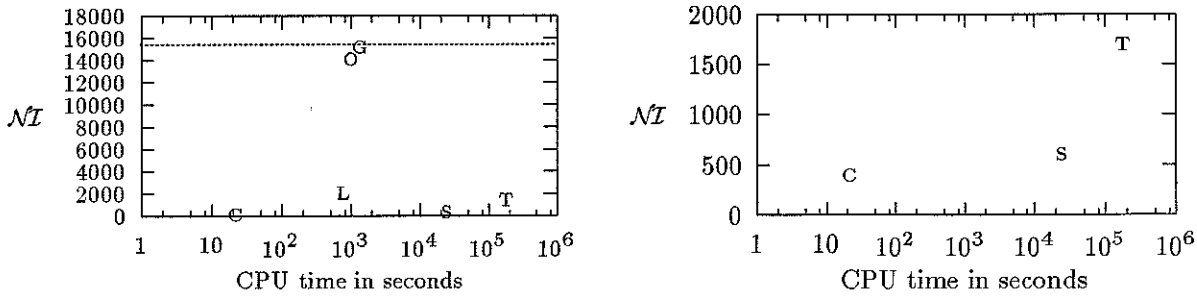

Figure 2: $\mathcal{NI}$ versus running time for all heuristics on $U_{10000,0.01}$ (left), and enlargement of the lower portion thereof (right).

As a result of these first-cut tests, we eliminate from further consideration the genetic algorithms (G and O), as each is strongly dominated by connectivity traversal on all three inputs. This is not surprising; genetic algorithms often do not work well on permutation problems such as retrieval layout, since it is difficult to devise crossover operators that preserve the structure of the parents in any meaningful way [5]. More complex crossover operators can work well for some permutation problems [6, 7], but devising such crossover operators is quite difficult.

The performance of local optimization varies greatly with input; it is clearly dominated by connectivity traversal on CACM and $U_{10000,0.01}$ but gives a better solution than connectivity traversal on $M_{10000,0.01}$, albeit after a much longer time. Local optimization possesses one advantage over other local search heuristics: its running time is strongly correlated with the amount of layout improvement it achieves.

Simulated annealing performs as expected: it produces high-quality layouts but requires large amounts of computing time. Pure simulated annealing is not a feasible approach for large information graphs, but we show later that it has value as part of the hierarchical hybrid approach.

Tabu search consumes even more computing time than simulated annealing, and in two of the three first-cut tests, produces results no better than connectivity traversal. We consider it further within the hierarchical hybrid approach.

In addition, the performance of simulated annealing and tabu search can be improved by initializing them with a good solution rather than a random solution; further discussion of this technique appears below.

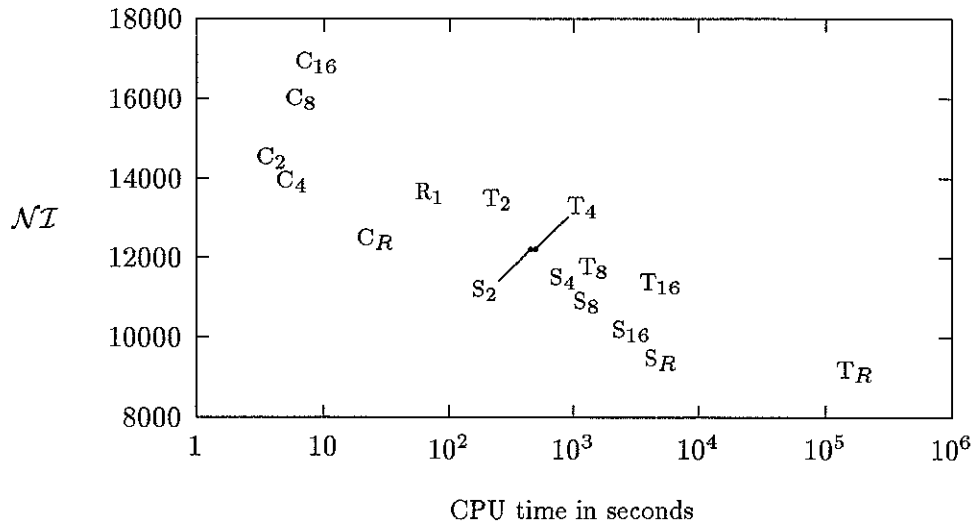

Figure 3: Results of hierarchical hybrid for various levels of decomposition (see text).

Hierarchical hybrid testing. We now consider in detail the hierarchical hybrid approach of recursive decomposition into subgraphs that are sufficiently small to handle with more expensive heuristics. Specifically, the approach taken here is to run recursive decomposition until each subgraph has size less than or equal to kP, where k is a small constant. Simulated annealing, tabu search, or connectivity traversal is then run on each resulting subgraph, and the results combined to form a complete layout. Figure 3 shows the results of testing simulated annealing, tabu search, and connectivity traversal on the CACM graph, recursively decomposed with k taking values 2, 4, 8, and 16. In the figure, S_k denotes simulated annealing applied to subgraphs of size kP; similarly, T_k and C_k denote the same for tabu search and connectivity traversal, respectively. In Figure 3, R_1 is the result of pure recursive decomposition, and S_R, T_R, and C_R are the results of applying the corresponding heuristic to the entire graph with no recursive decomposition. These results indicate that hierarchical hybrid is a viable approach that often outperforms connectivity traversal for

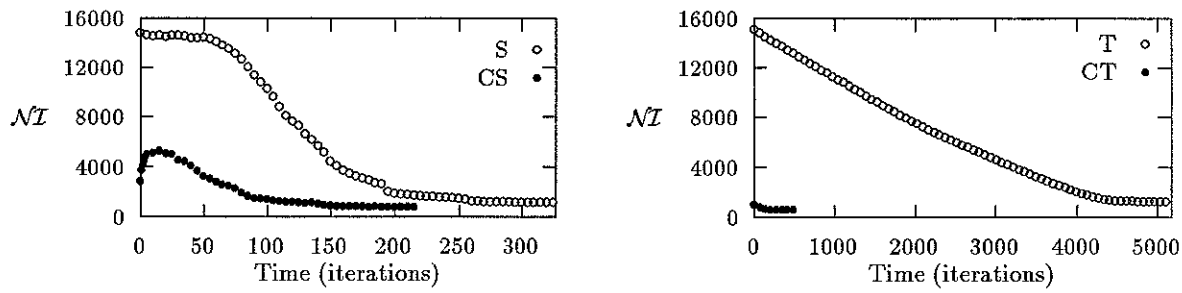

Figure 4: Objective value versus time for simulated annealing (left) and tabu search (right). Input is from $U_{5000,0.02}$.

graphs of this size. However, hierarchical hybrid is too slow to be practical for any but small values of k. For larger graphs—those with hundreds of thousands of vertices—hierarchical hybrid cannot be run in a feasible length of time with k greater than 4 [17]; however, when $k \leq 4$, hierarchical hybrid is consistently dominated by connectivity traversal.

A good initial layout. Another approach we examine is *jump-starting* a local search heuristic by giving it a good initial layout produced by connectivity traversal rather than a random one. This technique substantially reduces the running time and improves the solution quality of both simulated annealing and tabu search. Figure 4 shows solution quality as a function of time for simulated annealing and tabu search, each initialized with a random layout (S and T) or with a layout produced by connectivity traversal (CS and CT).

To jump-start simulated annealing, the initial temperature must be lower than when the heuristic is initialized with a random layout. We chose this temperature by trial and error, and the slight increase in objective value seen in the first 50 iterations in Figure 4(a) indicates that the starting temperature is still a bit too high. Recent work by Varanelli and Cohoon [22] and Rose, Klebsch,

and Wolf [19] investigates more rigorous methods for determining starting temperature in a jump-starting scheme; it would be interesting to apply such techniques to the retrieval layout problem. Nonetheless, the jump-started version runs in less time (26 minutes instead of 35) and produces a better solutions ($\mathcal{NI} = 867$ instead of $\mathcal{NI} = 1233$) than the version initialized with a random layout.

Tabu search seems, in a sense, better suited to this jump-starting scheme than simulated annealing, as it has no notion of temperature—the heuristic may simply be initialized with a good layout rather than a random one, with no other modification required. As with simulated annealing, the jump-started version runs far faster (73 minutes instead of roughly 9 hours) and produces far better solutions ($\mathcal{NI} = 646$ instead of $\mathcal{NI} = 1317$) than the version initialized with a random layout. Note that the jump-started tabu search heuristic produces the best layout seen in these tests, but that it requires more time than even the non-jump-started version of simulated annealing.

To gain some further insights into the performance of the heuristics, we tested them on six classes of 1000-vertex graphs. The results are summarized in Table 2; each value is averaged over 10 runs on each of 10 instances of the specified graph class. The row labeled "Quality" is the average ratio of layouts produced by the heuristics to random layouts. Where the mnemonic for a local search heuristic appears alone, the heuristic is initialized with a random layout; where the mnemonic appears preceded by "C," the heuristic is jump-started, i.e. initialized with a layout produced by connectivity traversal. For the most part, these results confirm those seen in the previous tests. With one exception ($U_{1000,0.03}$), simulated annealing consistently outperforms tabu search in both running time and solution quality, even when tabu search is jump-started. Further

Graph class	$U_{1000,0.03}$ (average degree = 5.6)						
Algorithm	C	L	CL	S	CS	T	CT
Quality	0.0247	0.1270	0.0095	0.0246	0.0213	0.0653	0.0087
Runtime (s)	0.4	3.8	1.3	75.7	57.4	355.7	50.2
Graph class	$U_{1000,0.05}$ (average degree = 15.1)						
Algorithm	C	L	CL	S	CS	T	CT
Quality	0.0862	0.2075	0.0664	0.0820	0.0514	0.1146	0.0589
Runtime (s)	0.2	8.6	3.4	118.6	81.7	388.9	135.3
Graph class	$U_{1000,0.07}$ (average degree = 29.0)						
Algorithm	C	L	CL	S	CS	T	CT
Quality	0.1449	0.2541	0.1210	0.1430	0.1056	0.1213	0.1100
Runtime (s)	0.5	14.4	8.6	125.6	71.5	621.0	328.7
Graph class	$M_{1000,0.01}$ (average degree = 7.0)						
Algorithm	C	L	CL	S	CS	T	CT
Quality	0.2225	0.3296	0.1549	0.1240	0.1179	0.2026	0.1393
Runtime (s)	0.3	3.4	2.1	79.3	58.7	222.8	55.0
Graph class	$M_{1000,0.015}$ (average degree = 17.1)						
Algorithm	C	L	CL	S	CS	T	CT
Quality	0.3112	0.3814	0.2451	0.2022	0.1994	0.2334	0.2217
Runtime (s)	0.3	5.1	3.3	111.0	75.9	332.4	134.8
Graph class	$M_{1000,0.02}$ (average degree = 28.0)						
Algorithm	C	L	CL	S	CS	T	CT
Quality	0.4540	0.4952	0.3646	0.3176	0.3164	0.3378	0.3310
Runtime (s)	0.5	7.1	5.2	126.0	74.3	423.4	184.8

Table 2: Heuristic performance on six classes of 1000-vertex graphs (see text).

study is required to more precisely determine the tradeoffs between simulated annealing and tabu search.

5 Conclusions

This paper reports a first examination of the performance of local search heuristics on the retrieval layout problem. Our initial conclusions are:

- Genetic algorithms do not produce good solutions to this problem;
- Simulated annealing and tabu search are too expensive to use in their pure forms;
- For a good solution found quickly, connectivity traversal should always be run first;
- Local optimization can be run second and often produces an improved solution, especially when initialized with a good layout produced by connectivity traversal;
- A hierarchical hybrid approach using simulated annealing can give a good layout in significantly less time than pure simulated annealing; and
- Our best heuristic for finding a good layout is to run connectivity traversal to obtain an initial layout for a run of simulated annealing or tabu search.

As with all studies of local search heuristics, better results might be obtained using different parameters—for example, a different cooling schedule for simulated annealing or different tabu list management techniques for tabu search. However, it is apparent that these heuristics cannot be naïvely applied to very large graphs, as their running time is prohibitive. Two approaches that

do appear promising are hierarchical hybrid using simulated annealing, and jump-started simulated annealing or tabu search. In particular, these approaches provide tremendous flexibility in finding high-quality solutions in a feasible amount of time.

References

[1] L. J. Bain and M. Engelhardt, *Introduction to Probability and Mathematical Statistics*, Duxbury Press, Belmont, 1992.

[2] B. Bollobás, *Random Graphs*, Academic Press, Orlando, Florida, 1985.

[3] Q. F. Chen, *An Object-Oriented Database System for Efficient Information Retrieval Applications*, PhD thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, 1992.

[4] Q. F. Chen and E. A. Fox, *LEND System Component Description*, Virginia Polytechnic Institute and State University, 1991.

[5] J. P. Cohoon, 1992. Personal communication.

[6] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. Richards, *Distributed genetic algorithms for the floorplan design problem*, IEEE Transactions on Computer-Aided Design, 10 (1991), pp. 483–492.

[7] J. P. Cohoon and W. D. Paris, *Genetic placement*, IEEE Transactions on Computer-Aided Design, CAD–6 (1987), pp. 956–964.

[8] P. J. Denning, *Genetic algorithms*, American Scientist, 80 (1992), pp. 12–14.

[9] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in Proceedings of the 19th IEEE Design Automation Conference, 1982, pp. 175–181.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.

[11] F. Glover, *Tabu search—part I*, ORSA Journal on Computing, 1 (1989), pp. 190–206.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts, 1989.

[13] L. S. Heath and J. W. Lavinus, *Heuristics for laying out information graphs.* Submitted, 1993.

[14] ——, *Optimal and random partitions of random graphs.* Submitted; available as Technical Report TR 93-24, Department of Computer Science, Virginia Tech, 1993.

[15] D. S. Johnson, *Fast algorithms for bin packing*, Journal of Computer and System Sciences, 8 (1974), pp. 272–314.

[16] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, *Optimization by simulated annealing; part I, graph partitioning*, Operations Research, 37 (1989), pp. 865–892.

[17] J. W. Lavinus, *Heuristics for laying out information graphs*, Master's thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, 1992. Available as Technical Report ST92–01.

[18] H. V. D. Parunak, *Ordering the information graph*, in Hypertext/Hypermedia Handbook, E. Berk and J. Devlin, eds., McGraw-Hill, 1991.

[19] J. S. Rose, W. Klebsch, and J. Wolf, *Temperature measurement and equilibrium dynamics of simulated annealing placement*, IEEE Transactions on Computer-Aided Design, 9 (1990), pp. 253–259.

[20] K. Shahookar and P. Mazumder, *A genetic approach to standard cell placement using meta-genetic parameter optimization*, IEEE Transactions on Computer-Aided Design, 9 (1990), pp. 500–511.

[21] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel, Boston, 1987.

[22] J. M. Varanelli and J. P. Cohoon, *Two-stage simulated annealing*, in Proceedings of the 4th ACM/SIGDA Physical Design Workshop, 1993, pp. 1–10.