Simulated N-Body: New Particle Physics-Based Heuristics for a Euclidean Location-Allocation Problem

Rahul Simha* Weidong Cai† Valentin Spitkovsky‡

September 29, 1999

Abstract

The general facility location problem and its variants, including most location-allocation and P-median problems, are known to be NP-hard combinatorial optimization problems. Consequently, there is now a substantial body of literature on heuristic algorithms for a variety of location problems, among which can be found several versions of the well-known simulated annealing algorithm. This paper presents an optimization paradigm that, like simulated annealing, is based on a particle physics analogy but is markedly different from simulated annealing. Two heuristics based on this paradigm are presented and compared to simulated annealing for a capacitated facility location problem on Euclidean graphs. Experimental results based on randomly generated graphs suggest that one of the heuristics outperforms simulated annealing both in cost minimization as well as execution time. The particular version of location problem considered here, a location-allocation problem, involves determining locations and associated regions for a fixed number of facilities when the region sizes are given. Intended applications of this work include location problems with congestion costs as well as graph and network partitioning problems.

Keywords: Facility location, P-median, location-allocation, simulated annealing, graph partitioning.

1 Introduction

Research on the general facility location problem, including several versions of the problem, has now grown to encompass scores of papers, several books and a journal devoted to location problems [9, 17, 19]. As evidenced in the literature, exact solutions for the vast majority of location problems remain computationally intractable and, consequently, a great deal of research has focused on

*Author for correspondence. Department of Computer Science, College of William & Mary, Williamsburg, VA 23187. simha@cs.wm.edu.

†Open System Software Division, Hewlett-Packard, MS 47 LA 1, 19447 Pruneridge Avenue, Cupertino, CA 94086. Email: wcai@cup.hp.com.

‡Electrical Engg. and Computer Science Dept., Massachusetts Institute of Technology, Boston, MA. val@mit.edu.

developing and analysing heuristics for a variety of location problems. Some of these heuristics exploit the structure of specific variations of the problem and others, such as simulated annealing, are general search heuristics applied to individual location problems. The heuristics in this paper have physics-based origins similar to those of simulated annealing but are quite different in form and are designed to exploit the structure of the particular location problem considered here.

We consider a discrete capacitated location-allocation problem on an n-node Euclidean graph in which k facilities are to be placed at graph vertices so that each graph node is "served" by a facility. Furthermore, m_i, the capacity of facility i is defined here as the number of nodes it is to serve; the k capacities $m_1, \ldots, m_k$ are given as input to the problem. The problem is to decide where to place the facilities (the location problem) and to decide which facility each node is assigned to for service (the allocation problem). The cost function we consider is of the P-median type, a sum of the distances between nodes and their facilities. Note that unlike other capacitated location problems in which m_i is simply an upper bound, our problem requires that exactly m_i nodes be assigned to facility i. This variation is inspired by location problems with heterogeneous servers that consider congestion costs [3, 4, 5, 19] in which, to balance congestion loads, each server is assigned a number of nodes that it must serve. Other applications include graph partitioning problems for load balancing of parallel computations and network partitioning problems [7].

For the location problem studied in this paper, we present two heuristics that, like simulated annealing, are based on an analogy drawn from particle physics. Loosely speaking, while the metaphor for simulated annealing equivalences the search process with the use of annealing to strengthen alloys, the metaphor we use equivalences search with finding an approximately stable solution for an n-particle (n-body) system with interacting forces. Hence, the name *simulated n-body*. The key idea is to let particles represent facilities with mass proportional to the capacities and to let the particles "jostle around" and find a low energy configuration, in the hope that the configuration so found leads to a good solution to the original location-allocation problem. We devise two heuristics that differ in the manner by which the energy or potential for a candidate configuration is computed, and also consider faster versions that prune the search space albeit at some sacrifice of solution quality.

Experiments on graph sizes up to 1000 nodes indicate that one of our heuristics significantly outperforms our own annealing implementation both in computation time and quality of solution. We recognize, however, that the strength of our heuristics – their exploitation of the particular structure of the problem – is also a drawback in that it may be difficult to apply to other location problems. Nonetheless, we believe problems with similar structure (load balancing, graph partitioning) could use the ideas underlying the algorithms.

Location problems have been surveyed extensively in several books and journal articles [9, 10, 17, 19, 24, 27]. In particular, the book on discrete location theory [19] has an extensive bibliography on location-allocation problems [8, 12] and capacitated location problems. The allocation part of a location-allocation problem is sometimes referred to as districting. Indeed, we use a districting method similar to the one in [14] and concentrate on the location part of the problem. The application of simulated annealing to location problems is described in [6, 18, 22, 23]. A particularly relevant reference is [18] in which simulated annealing is used on Euclidean location problems. A recent paper describing tabu search on location-allocation is [12]. Finally, a problem similar to ours but with a P-center cost is considered in [16].

Most location problems, except for a few with specific graph structures, are NP-hard. Searches for optimal solutions scale unfavourably with the size of problem instances. For example, it takes more than 1 hour of CPU time on a Convex 210 machine to solve a 30-node, 5-facility location problem to optimality, as reported in [21]. Accordingly, considerable effort has been expended on the development of fast heuristic methods, including tuning general-purpose search algorithms such as simulated annealing. However, these general-purpose algorithms often do not exploit the structure of the problem, in contrast with the algorithms presented in this paper.

This paper is organized as follows. Section 2 contains a mathematical formulation of our problem. Section 3 is the core of this paper. We will discuss in great detail the two simulated n-body heuristics. In Section 4 we discuss our implementation of simulated annealing. Experimental results are given in Section 5 and concluding remarks in Section 6.

2 Problem Formulation

Our capacitated location-allocation problem has the following components:

- $G(V, E)$ – a connected Euclidean graph on $|V| = n$ vertices. The length or weight of an edge is the Euclidean distance between the endpoints of the edge.
- $N(u)$ – the degree of graph node u (the number of edges incident at u).
- $d(u, v)$ – the length of the shortest path from node u to node v in the graph G.
- k – the number of facilities.
- m_i – the number of nodes that is to be assigned to facility i, $i = 1, 2, \ldots, k$. Note that $\sum_{i=1}^{k} m_i = n$.
- $\delta : \{1, \ldots, k\} \times \{1, \ldots, n\} \rightarrow \{0, 1\}$ – an allocation of nodes to facilities. For facility i and graph node j, $\delta_{ij} = 1$ if and only if node j is assigned to facility i; $\delta_{ij} = 0$ otherwise. Thus, we have the constraints: (1) $\forall j : \sum_i \delta_{ij} = 1$ (a node is assigned to only one facility), (2) $\forall i : \sum_j \delta_{ij} = m_i$ (a facility is assigned exactly as many nodes as specified).
- $L : \{1, \ldots, k\} \rightarrow V$ is the function that determines the location of each facility: $L(i) \in V$ is the location of facility i. Note that facilities are constrained to be located on graph nodes, an assumption common to most discrete location problems.

- $f(L, \delta)$ – the cost function for a particular location L and allocation δ:

$$f(L, \delta) = \sum_{i=1}^{k} \sum_{j=1}^{n} d(L(i), j)\delta_{ij}.$$

We are now in a position to state the problem: we wish to find locations for the facilities (L) and an allocation of nodes to facilities (δ) to minimize f.

Note that an allocation of nodes to facilities after they are located results in districts or regions where a region for facility i is the collection of nodes assigned to facility i: $\{j \in V : \delta_{ij} = 1\}$. Observe that we have not restricted the regions to be contiguous; that is, the subgraph of G corresponding to a region need not be connected. There appears to be no practical reason to impose such a restriction. Nonetheless, solutions typically produce contiguous regions for Euclidean graphs, since by swapping nodes that violate contiguity one often obtains a better solution. Thus, if the entire Euclidean graph along with its faces is viewed as a piece of cardboard, the problem can loosely be described as one of cutting up the board into "chunks" of given sizes. In particular, for a regular mesh graph, the sizes correspond to the actual weights (masses) of the pieces. This observation is the intuition behind our heuristics, a presentation of which follows next.

3 Heuristics

We begin describing the heuristic algorithms in this paper by outlining the intuition behind them. Our heuristics were first inspired by considering the problem for regular mesh graphs. In this case, a rectangular grid of nodes is to be partitioned into k pieces when the piece sizes are given. If the grid is drawn on a piece of cardboard, the problem is to cut the board into smaller pieces of given mass and to do so while minimizing the cost function. Roughly, the cost function amounts to the average (or sum) of the moments of inertia [26] of the pieces. Thus, one would like circular pieces as much as possible – something that is difficult to achieve with a grid. Nonetheless, if one could find a good solution, then the centers of mass of these pieces would be ideal locations for the facilities.

By viewing each facility as a particle with mass proportional to its given region size, and by allowing the particles to "jostle" for space in an environment consisting of the grid and some interacting (repelling) forces, it is hoped that large (heavy) particles would keep a distance from smaller (lighter) particles. If the system stabilizes so that the potential energy is minimal, then perhaps the particle locations could serve as locations for the facilities on the graph. The idea here is that a large particle would, by forcing other particles away, carve out enough space for itself; a small particle, on the other hand, is likely to find a corner or an interstitial space between the regions of large particles – a property that encourages efficient "packing." At this time, we do not forsee using any attracting forces between particles.

Thus, we have transformed the location problem into an N-body problem (or k-body in this case) [2, 11, 25] in which a stable solution (a solution with minimal potential) for the N-body problem is associated with a solution to the location part of the location-allocation problem. (The allocation part will be solved using a standard heuristic [14]). Unfortunately, while the N-body metaphor is appealing, a direct simulation of the N-body problem often results in unstable or bifurcating systems with no single stationary solution.

To make the method practical, we take two further steps. First, the Euclidean plane is discretized by returning to the graph structure and allowing particles to only assume positions on graph nodes. Second, we use a process of aggregation and decomposition akin to dynamic programming to speed up the computation. The latter process is described as follows. Initially, all but one particle is aggregated into one "giant" particle, leaving us with one giant particle and one regular particle. Searching quadratically, all possible locations are considered for these two particles in an attempt to minimize potential energy. When the optimal locations are found, the location for the regular particle is fixed. Then, the giant particle is decomposed by extracting another regular particle from it, and the process is repeated. Thus, a quadratic-cost search is performed for each facility, leading to a computation cost of $O(kn^2)$. Since it takes $O(k)$ time to compute the potential at each step, the overall time is $O(kn)^2$.

Two types of heuristics are presented, distinguished by the potential function used in each. As indicated above, particle locations are evaluated using a potential function and thus, different potential functions can result in different solutions. The first heuristic, which we call the *Boundary Heuristic*, uses a potential function that consists of two terms, one a sum of two-particle interactions (mass-product divided by the distance) and the second, a sum of terms describing the interaction between particles and a boundary enclosing the graph. The intuition is, we would like particles to stay away from each other and also from the "walls" enclosing the graph. The second heuristic, which we call the *Connectivity Heuristic* uses the topological structure of the given graph: the particle interaction terms are scaled by the degree of the graph vertex at which the placement is being considered. The intuition here is that a vertex should be favored if its degree is high, since a facility at that location should be able to reach many nodes by a direct edge. (A similar idea is used in [15] for the graph partitioning problem).

Once a location is computed, the graph nodes need to be allocated to the facilities. We use a simple heuristic for this purpose. Initially, each node picks the nearest facility. This will result in some regions with excess and others with fewer nodes than required. Regions with fewer nodes (starved regions) then grab nodes from regions with more nodes (rich regions). To minimize the increase of the cost caused by such grabbing, starved regions grab the nodes that are closest to them in the rich regions.

3.1 Boundary Heuristic

By way of introduction, consider a simple potential function with only particle interactions. For a particular configuration of particles where $L(i)$ denotes the location of particle i, the contribution to the potential by particle i can be written as:

$$\sum_{j \neq i} \frac{m_i m_j}{d(L(i), L(j))}$$

where m_i and m_j are the "masses" of particles i and j (actually, the region sizes) and d is a suitable distance measure. Note that d can be a generic distance function, although we have used Euclidean distance. Since a lower potential is achieved by separating the particles (especially the heavy ones), particles can accumulate along the boundary. To prevent this, we define a repelling force from a "wall" enclosing the graph. One possible enclosing boundary is the bounding rectangle for the graph (the smallest rectangle with sides parallel to the axes that encloses the graph). In this case, we have four segments in the boundary. Let W_i, $i = 1, \ldots, 4$ denote the four segments and let $D(u, W_i)$ denote the distance of vertex u from segment W_i. Then, the potential for particle i can be written as:

$$\sum_{j \neq i} \frac{m_i m_j}{d(L(i), L(j))} + \gamma \sum_{j=1}^{4} \frac{m_i}{d(L(i), W_j)}$$

where γ is a weighting factor selected to weight the relative importance of boundary effects. Thus, the potential for a particular configuration, L, of particles is

$$\text{Potential } = \frac{1}{2} \sum_{i=1}^{k} \sum_{j \neq i} \frac{m_i m_j}{d(L(i), L(j))} + \gamma \sum_{i=1}^{k} \sum_{j=1}^{4} \frac{m_i}{d(L(i), W_j)} \tag{1}$$

We now describe the algorithm. As explained earlier, a decomposition technique is used so that locations for the k particles are determined in k iterations with each iteration used for a single particle. The particles are sorted in order of mass. The smallest particle is considered in the first iteration along with an aggregate consisting of all the other particles. This giant particle and the smallest one are placed in all possible combinations to discover the placement with the least potential. (The potential is computed using the aggregate mass for the giant particle). When the best configuration is found, the location of the smallest particle is fixed.

In the second iteration, the next smallest particle is extracted from the aggregate and both the aggregate (giant) particle and this next smallest particle are tried in different locations to find the least potential. Note that even though the first particle (the smallest overall) has been located, it is used in the potential computation since to ignore its presence as a particle would be ignore its effect as a facility.

In this manner, at each iteration a particle is removed from the aggregate and the two are placed in all possible combinations to select one with the least potential. The potential at each step is computed using all particles available at the step, including those whose assignment has been made.

The pseudocode for the algorithm is given below. The array P[ ] is used to store masses and locations of the different particles. The aggregate particle's mass and location are maintained in variables aggregate.mass and aggregate.center.

Algorithm: Boundary Method

```
Compute all-pairs shortest paths for graph G.
Sort particles so that P[1].mass,...,P[k].mass are decreasing;
Set aggregate.mass = P[1].mass + ... + P[k].mass;
for j from k downto 1
   min ← MAXFLOAT;
   aggregate.mass = aggregate.mass - P[j].mass;
   for aggregate.center from 1 to n
      for P[j].center from 1 to n
         if aggregate.center ≠ P[j].center
            // Compute potential for this configuration
            // First, the potential due to the two particles
            p1 ← (P[j].mass ∗ aggregate.mass) / Dist (P[m].center, aggregate.center);
            for m from j+1 to k // Next, the potential due to assigned particles
               p1 ← p1 + (P[m].mass ∗ P[j].mass) / Dist(P[m].center, P[j].center) +
               (P[m].mass ∗ aggregate.mass) / Dist(P[m].center, aggregate.center);
            Compute boundary effects potential, p2, similarly;
            potential ← p1 + γ∗p2;
            if potential < min
               min ← potential;
               facility[j].center = P[j].center
            endif
         endif
      endfor // P[j].center from 1 to n.
      P[j].center ← facility[j].center;
endfor // Outermost for-loop
```

Note that the above computation of potential in the inner for-loops does not include the potential due to interactions between particles whose locations have already been assigned. This is because their contribution to the potential will cancel on both sides of a comparison – it is only the contribution due to the new particles that need to be compared. Also, observe that the particles are sorted in decreasing order and thus are considered smallest-first in the iterative process. Experimentally, the decreasing order turned out to be better than an increasing order. We hypothesize that it is important for the aggregate particle to be as large as possible initially so that small particles can be pushed into "corners" as efficiently as possible.

3.2 Connectivity Heuristic

While the boundary heuristic was conceived for uniform rectangular grids and works well on these grids, it does not perform as well on general graphs. It is easy to see why it works for rectangular grids. As mentioned earlier, particles need to repel each other and the four boundaries of the rectangular grid. Because of the grid uniformity, the "space" around a particle is roughly proportional to the number of graph nodes contained within it. Stated differently, the center is at an ideal location because it can reach its assigned graph nodes as quickly as possible (with paths as short as possible). However, this property does not hold for general graphs. A random graph can have dense clusters of nodes as well as sparsely populated areas. Nonetheless, we would like to select locations so that the property of being "centrally located" within a region holds as much as possible.

The *connectivity heuristic* modifies the potential function so that nodes with high degree are favored over nodes with lower degree in selecting locations. The intuition is, path lengths from high-degree nodes to nearby vertices are likely to be short. We ignore boundary effects since high degree nodes may be found anywhere, including on the boundary. The potential for two particles is given by

$$\frac{m_i \cdot m_j}{d(L(i), L(j)) \cdot N(i) \cdot N(j)}, \tag{2}$$

where $N(i)$ and $N(j)$ are the number of neighbors of the facilities i and j. The potential for a complete configuration is therefore,

$$Potential = \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} \frac{m_i \cdot m_j}{d(s_i, s_j) \cdot N(i) \cdot N(j)}. \tag{3}$$

Note that the degree is factored in by increasing the potential for low-degree nodes (inverse proportionality).

Except for the calculation of the potential in each iteration, the pseudocode for the connectivity heuristic is identical to that of the boundary heuristic. In both heuristics, the facilities are first located and then an allocation of nodes to facilities is performed using the method described earlier. We explore next the possibility of interleaving allocation steps with location steps.

3.3 Interleaved location and allocation

Unlike the current method of first determining locations and then allocating nodes, facilities may also be allocated their regions at the time their location is fixed in the N-body iterative process. This interleaving of allocation with location seeks to reduce the search space. Consider the first facility to be located (in the first iteration of the boundary heuristic, for example). If its region is determined, the nodes in the region can be avoided in the search for future facility locations, thereby reducing the search space. Currently, the process of location allows most graph nodes to be searched. Thus, by reducing the search space we can hope for a speed-up in computation, albeit at the cost of producing somewhat poorer solutions. Our experimental results (the "interleaved" columns in tables 1 and 3) show that, indeed, this variation is faster and also produces solutions with higher cost than the non-interleaved versions. We believe the increase in both speed and cost can be attributed to the reduction in search space: since fewer solutions are considered this approach is faster; at the same time, allocation decisions are made early, prior to determining even the location of some facilities, resulting in potentially negative consequences. Note that interleaving may be applied to both the boundary and connectivity heuristics.

4 Simulated Annealing

We use the performance of simulated annealing as a benchmark for our heuristics. The algorithm is not described here given the ample literature on this subject (see [1], for example).

There are probably two ways of running the annealing algorithm for the purpose of comparison with our heuristics. One, execute the algorithm until the acceptance probability is very small and no further (greedy) progress is made and then compare both execution time and quality of solution. Or two, run the heuristic while noting its execution time and then run annealing for the same amount of time, after which the quality of solution can be compared. We opt for the second approach because it is more straightforward: only one quantity, the quality of solution, needs to be compared. Furthermore, our experiments reveal that the acceptance probability is very small at the end of the annealing runs – thus, continued execution of the annealing algorithm does not produce any improvement. In comparing annealing to all of our heuristics, we select the longest running time among the heuristics to use as the clock time for annealing, thereby giving annealing its best shot.

An annealing algorithm is determined by the choice of a neighborhood function and a cooling schedule. We first describe the cooling schedule. As numerous studies have revealed [1], the temperature schedule used in annealing greatly impacts the performance of the algorithm. Consequently, some experimentation with cooling schedules was needed to get annealing to perform well on our problem. We use three different schedules and for each we follow these rules of thumb based on the presentation in [1]: first, the initial temperature is selected so that the acceptance probability is close to unity; second, when the acceptance probability is very small (say, of the order of 10^{-6}), the algorithm is terminated since there is at that time only a negligible chance to climb out of local optima without incurring a severe computational cost.

The following three cooling schedules were used separately. For an explantion of these schedules, the reader is referred to [1]. Let t_k denote the temperature at the k-th step.

1. *Linear cooling*: $t_{k+1} \leftarrow t_k - r$, where r is the reduction rate and is computed initially. One way to compute r is to guess how many iterations will be needed. For example, suppose we want to use 100,000 iterations. Then, with an initial acceptance probability of unity and a final acceptance probability of 10^{-6}, r can be computed by the number of subtractions needed (100,000) to get from unity to 10^{-6}.

2. *Polynomial cooling*: $t_{k+1} \leftarrow \frac{t_k}{1+\nu t_k}$, where ν is a small constant.
3. *Exponential cooling*: $t_{k+1} \leftarrow \lambda t_k$, where λ is a number close to but less than unity. For example, $\lambda = 0.9999$.

While none of these schedules is known to result in theoretical convergence, our experiments indicate that the cooling schedule that does theoretically converge, the *inverse-log* schedule, cools too slowly for appreciable change.

For a neighborhood function, the following scheme was used. First, an initial configuration of regions is randomly selected given the region sizes. Facility locations are then selected as the nodes closest to the geometric center of each region. At each subsequent step, a new configuration is created by randomly selecting two regions for a "swap" and selecting two nodes randomly in the region for a swap. To compute the new cost, a new configuration is generated as follows: the nodes are swapped and the cost is computed based on the current locations of the facilities. If the new cost is lower, the new configuraton is the next configuration and the geometric center of each new region is taken as the new location of the facilities. If the new cost is not lower, the new configuration is accepted as the next configuration based on the acceptance probability computed using the two costs, the temperature and the constants used in the algorithm. In all other cases, the new configuration is the current configuration. The final cost is taken to be the lowest cost found among all iterations.

While the general structure of the annealing algorithm provides endless opportunities to experiment with both neighborhood functions, temperature schedules, memory and aspiration criteria, such exhaustive experimentation somewhat defeats the purpose of using these algorithms. Better neighborhood functions, for example, can often be obtained by analysis of the problem structure. However, effort spent on analysing the problem structure is just as well spent on devising special-purpose heuristics for the problem, the approach taken in this paper.

5 Experimental Results

We tested our algorithms and compared them to simulated annealing on randomly generated Euclidean graphs. The following method was used to generate planar graphs (since we expect the intended applications, geographic problems, to respect planarity for the most part). For an n-node problem, n points are generated uniformly in the unit square. As each point is generated, edges are randomly generated to existing nodes. Edges that intersect existing edges are rejected. At least one edge is generated so that the graph is always connected.

Graph sizes were taken in the range 100 to 1000 nodes. For each size, average costs and execution times were estimated from the results of 10 randomly-generated graphs per size (10 runs per sample mean) run on a Pentium machine (200MHz) with the Linux operating system. We make no claim about statistical validity: long execution times prohibited the generation of enough data for meaningful confidence intervals. For each graph, 10 facilities were used. We used two types of region sizes: in one set of data, the region sizes were identical; in the second, the region sizes were non-uniform. For the non-uniform region sizes for a 100-node graph, the following region sizes were used: 5,5,5,5,5,5,15,15,15,25. These sizes were scaled appropriately for the different graph sizes. The parameter γ reflecting the relative weight of the boundary effects was taken to be 0.75 after some experimentation with small graph sizes.

As mentioned earlier, simulated annealing was run for a fixed time for each graph size (different for each graph size), with the time taken as the longest execution time among the heuristics for the graph size.

Tables 1 and 2 compare our heuristics with simulated annealing for the uniform region-size case. Cost and execution times are given for both interleaved and non-interleaved versions of our heuristics. Up to six decimal places of accuracy are shown. For each version of annealing, the corresponding longest time was taken from Table 1 and used as the execution time. The resulting costs are shown in Table 2. For example, consider the size-400 row in tables 1 and 3. The longest time is 1129.4, taken by the non-interleaved version of the connectivity algorithm. This number is rounded up to 1130 and each version of the annealing algorithm for size-400 is run for this time (for each run). The average is shown in the size-400 row of tables 2 and 4.

The following observations may be made from the tables:

- The connectivity algorithm in both its versions performs the best, both in execution time and quality of solution.
- The interleaved version of each heuristic executes much faster than the corresponding non-interleaved version, but results in higher costs.
- The difference between the interleaved and non-interleaved versions is sharper for the connectivity heuristic.
- Simulated annealing produces solutions as good as the boundary algorithm (sometimes, slightly better) while taking the longest time.

These results for uniform sizes are confirmed by the results below for the non-uniform data set summarized in tables 3 and 4.

6 Conclusions

This paper presented heuristic algorithms for a special case of the general facility location problem. This special case involved fixed region or district sizes, the Euclidean measure and a sum cost function. The algorithms were based on principles in particle physics similar to those that inspired simulated annealing. One of the algorithms was shown to perform better than an implementation of simulated annealing both in execution time and quality of solution produced. The potential for an improved heuristic exists when our heuristics may be used as the basis for a tabu search procedure.

Acknowledgements: The authors would like to thank Prof. Rex Kincaid, Department of Mathematics, College of William and Mary, and the area editor for several valuable comments regarding this paper.

References

[1] Aarts, E. and Korst, J. (1989) *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons Ltd.

[2] Appel, A.W. (1985) An efficient program for many-body simulation. *SIAM J. Sci. Stat. Comput.*, Vol. 6, 1985, pp. 85-103.

[3] Batta, R. and Berman, O. (1989) A location model for a facility operating as an M/G/k queue. *Networks*, Vol. 19, pp. 717-728.

[4] Berman, O. and Mandowsky, R.R. (1986). Location-allocation on congested networks. *Euro. J. Oper. Res.*, Vol. 26, No. 2, pp. 238-250.

[5] Berman, O. and Larson, R.C. (1982). Optimal 2-facility network districting in the presence of of queueing. *Transportation Science*, Vol. 19, No. 3, pp. 207-216.

[6] Burkard, R.E. and Rendl, R. (1984). A thermodynamically motivated simulation procedure for combinatorial optimization problems. *Euro. J. Oper. Res.*, Vol. 17, pp. 169-174.

[7] Choi, H-A., Narahari, B. and Simha, R. (1997). Algorithms for mapping task graphs to a network of hetergeneous workstations. *Proc. ADCOMP 97*, 1997.

[8] Cooper, L. (1964) Heuristic methods for location-allocation problems. *SIAM Review*, **6**, 37-53.

[9] Daskin, M.S. (1995). *Network and Discrete Location: Models, Algorithms and Applications.* John Wiley & Sons, Inc.

[10] Francis, R.L., and Goldstein, J.M. (1974). Location theory: a selective bibliography. *Operations Research*, Vol. 22, pp. 400-410.

[11] Hockney, R.W. and Eastwood, J.W. *Computer simulation using particles.* McGraw-Hill, NY, 1981.

[12] Houck, C.R., Joines, J.A. and Kay, M.G. (1996) Comparison of genetic algorithms, random restart and two-opt switching for solving large location-allocation problems. *Computers & operations research*, Vol. 23, No. 6, pp. 587.

[13] Jacobsen, S.K. (1982) Heuristics for the capacitated plant location model. *European Journal of Operational Research*, **12**, 253-261.

[14] Keeney, R.L. (1972) A method of districting among facilities. *Operations Research*, **20**(3), 613-618.

[15] Kernighan, B.W. and Lin, S. (1970) An efficient heuristic procedure for partitioning graphs. *Bell Sys. Tech. J.*, **49**, pp. 291-307.

[16] Khuller, S. and Sussman, Y. (1996) The capacitated k-center problem. *Lecture Notes in Computer Science*, 1136, Springer-Verlag.

[17] Love, R.L., Morris, J.G., and Wesolowsky, G.O. (1988). *Facilities Location: Models and Methods.* North-Holland.

[18] Maffioli, F. and Righini, G. (1994) An annealing approach to multi-facility location problems in Euclidean space. *Location Science*, Vol. 2, No. 4, 205-222.

[19] Mirchandani, P.B. and Francis, R.L. (1990) *Discrete Location Theory.* John Wiley & Sons, Inc.

[20] Rolland, E., Schilling, D.A. and Current, J.R. (1996) An efficient tabu search procedure for the p-median problem. *Euro. J. Op. Res.*, Vol. 96, 1996, pp. 329-342.

[21] Rosing, K.E. (1991) Towards the solution of the (Generalized) Multi-Weber Problem. *Environment and Planning, Series B*, **18**, 347-360.

[22] Sharpe, R. and Marksjo, B.S. (1985). Facility layout optimization using the Metropolis algorithm. *Environment and Planning B*, Vol. 12, pp. 443-453.

[23] Sharpe, R., Marksjo, B.S., Mitchell, J.R. and Crawford, J.R. (1985). An interactive model for the layout of buildings. *Applied Mathematics and Modelling*, Vol. 9, pp. 207-214.

[24] Tansel, B.C., Francis, R.L. and Lowe, T.J. (1983). Location on networks: a survey, *Management Science*, Vol. 29, pp. 482-511.

[25] Warren, M.S. and Salmon, J.K. (1992) Astrophysical N-body simulation using hierarchical tree data structures. *Proc. Supercomputing 92*, 1992, pp. 570-576.

[26] Wilson, J.D. (1994) *College physics.* Prentice-Hall, NJ, 1994.

[27] Wong, R.T. (1985). Location and network design. In *Combinatorial Optimization: Annotated Bibliographies*, (O'Eigeartaigh et al, eds), John Wiley and Sons.

	Boundary heuristic				Connectivity heuristic			
	Non-Interleaved		Interleaved		Non-Interleaved		Interleaved	
Size	Cost	Time	Cost	Time	Cost	Time	Cost	Time
100	4883.1	62.5	4908.98	23.8	2560.6	63.4	3073.5	20.2
200	24514.5	265.3	25417.7	96.1	11063.9	268.5	14050.8	81.1
300	57165.5	619.5	59674.7	220.8	24510.7	626.4	32751.1	188.3
400	110085.0	1109.4	115872.0	397.6	44469.4	1128.6	57512.9	338.5
500	182543.0	1746.6	187472.0	624.8	71249.1	1775.2	86017.5	532.1
600	262052.0	2529.9	275311.0	904.5	97235.3	2575.5	125331.0	773.9
700	355635.0	3459.0	405515.0	1296.1	130229.0	3545.4	143716.0	1109.8
800	487707.0	4536.3	513456.0	1617.6	180347.0	4655.3	230040.0	1381.3
900	611871.0	5761.6	645013.0	2051.9	223667.0	5892.1	275729.0	1752.3
1000	769950.0	7194.3	814731.0	2539.8	274915.0	7277.5	343413.0	2170.1

Table 1: Boundary vs. Connectivity – uniform region sizes

Size	Cost(linear)	Cost(Exp.)	Cost(Poly.)	Time
100	6162.6	6102.4	6074.7	64
200	28021.3	28032.2	27786.0	269
300	64416.5	64166.4	63383.8	627
400	117060.0	116548.0	115801.0	1130
500	185430.0	187394.0	185163.0	1778
600	276324.0	273668.0	271197.0	2578
700	362409.0	361404.0	358974.0	3550
800	498260.0	497034.0	493136.0	4656
900	643863.0	640995.0	638047.0	5893
1000	712010.0	712714.0	705873.0	7285

Table 2: Simulated annealing – uniform region sizes

	Boundary heuristic				Connectivity heuristic			
	Non-Interleaved		Interleaved		Non-Interleaved		Interleaved	
Size	Cost	Time	Cost	Time	Cost	Time	Cost	Time
100	4446.6	62.3	5007.7	37.9	2608.0	63.5	3558.5	35.3
200	23121.1	264.6	26574.3	153.0	11489.4	268.0	16469.2	142.6
300	54720.0	614.1	62642.0	351.6	25867.8	625.3	37755.3	330.0
400	101810.0	1108.5	121607.0	631.2	45965.7	1129.4	64140.1	591.3
500	166949.0	1744.9	208861.0	995.3	69456.8	1777.2	94633.7	924.7
600	249829.0	2526.7	293355.0	1432.9	136775.0	3549.2	138909.0	1339.7
700	339227.0	3456.0	379076.0	1955.7	136775.0	3549.2	201526.0	1866.6
800	480839.0	4548.2	524906.0	2561.3	188191.0	4632.7	263751.0	2398.9
900	592117.0	5775.6	687192.0	3246.7	230083.0	5883.5	328913.0	3059.5
1000	716625.0	7161.4	823649.0	4014.2	279095.0	7284.6	394604.0	3767.0

Table 3: Boundary vs. Connectivity – nonuniform region sizes

Size	Cost(linear)	Cost(Exp.)	Cost(Poly.)	Time
100	6165.6	6119.6	6023.6	64
200	27632.5	28121.4	28524.7	269
300	64545.5	64242.9	64020.8	627
400	116284.0	116438.0	115509.0	1130
500	187607.0	186620.0	187307.0	1778
600	275336.0	276194.0	272704.0	2578
700	361038.0	360183.0	357330.0	3550
800	497722.0	497861.0	493354.0	4656
900	640139.0	643816.0	638615.0	5893
1000	714565.0	709709.0	708933.0	7285

Table 4: Simulated annealing – nonuniform region sizes