Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

SIMULATED ANNEALING AND RESOURCE LOCATION IN COMPUTER NETWORKS

G. Anandalingam
Department of Systems
University of Pennsylvania
Philadelphia, PA 19104

ABSTRACT
This paper examines the problem of locating resources such as databases, controllers, and data processors on a computer network. The integer programming problem in location variables y and interconnection variables x is solved using two simulated annealing algorithms. Pure simulated annealing for this problem has complexity $O(2^{N^2+N})$. In hybrid simulated annealing (HSA), for fixed y, the problem in x becomes a special case of the transporation problem; the worst case solution for HSA is $O(N.2^N)$. In parallel simulated annealing (PSA), a decomposition procedure yields knapsack problems for x when $y = 1$, and $x = 0$ when $y = 0$; this is solved in $O(N^k 2^N)$. Numerical results show that the computation time taken by the simulated annealing algorithms are comparable to a Lagrangian relaxation procedure, and the solutions are on the average within 8 % of a lower bound.

1. INTRODUCTION

The goal of topological design of computer networks is to achieve a specified performance at minimum cost. Since the full problem is intractable, topological optimization is decomposed into separate components including the design of the backbone of the network, local access network including the location of concentrators, and the distribution of network resources such as data bases, packet controllers, and fault indicators (i.e. alarms). Each of these individual problems is also difficult and complex, and there is fairly intensive research at present in deriving solution techniques for them.

Another important problem in designing computer networks is that of locating resources such as databases, controllers, and data processors (i.e. computers) so as to make them available to all customer sites at a minimum cost. Simultaneous treatment of computer and database location has been analyzed by Chen and Akoka [1980], Fisher and Hochbaum [1980], Gavish and Pirkul [1986], and Morgan and Levin [1977] among others. We call such an optimization problem the *Computer Resource Location Problem* (CRLP).

We present two algorithms based on simulated annealing for solving the CRLP. Simulated annealing, introduced by Kirkpatrick et al. [1983], has gained popularity in recent years because it is a new technique and has been applied to the solution of such problems as VLSI design, circuit placement, scheduling, and the famous travelling salesman problem in operations research (see the bibiliography in the report by Collins et al. [1987]). To our knowledge, this paper is the first one that has attempted to solve computer network design problems using simulated annealing.

The main drawback with simulated annealing (SA) is that it usually takes enormous amount of computer time to solve optimization problems. In this paper, we have used some heuristics for speeding up the process. In the first one, we transform the CRLP in such a way that part of it is easily solvable using straightforward linear programming techniques. Thus only some variables need to be generated randomly. In the second one, we parallelize the optimization problem, and use a nested simulated annealing algorithm to solve it.

In order to compare the performance of these algorithms, we also provide a *Lagrangian relaxation* algorithm, based on Mirzaian's [1985] method. We solved 25 randomly generated problems using all three algorithms for networks of upto 15 nodes. Results show that, on the average, both algorithms based on simulated annealing performed quite well. On the average, the computation time was comparable to the Lagrangian relaxation algorithm, and the optimal solutions were within 8 % of a lower bound.

This paper is organized as follows: In the next section, we provide a general formulation of the computer resource location problem (CRLP), and examine its complexity. Section 3 presents the main parameters of the simulated annealing algorithms. In Sections 4 and 5 we present the modified simulated annealing algorithms that we propose and examines the issue of complexity. In Section 6, we present the Lagrangian relaxation algorithm. Section 7 contains the results of numerical examples.

2. COMPUTER RESOURCE LOCATION PROBLEM

We begin with an existing network of nodes and interconnecting paths. In computer networks, paths consist of communication links, such as telephone lines, satellite links or microwave links. The nodes can represent a message switch, satellite earth station or a microwave relay station, etc., and provide basic communication services. We seek to place resources at various nodes on the existing network. We intend to determine the optimum number of resources and the optimum interconnections between users and the resources in the network.

When a resource is connected to a user, there is a cost incurred on the path used between the entities to provide the service. This cost will be referred as the path weight. The interpretation of the path weight depends on the application and the optimization criteria. For example, if one of the objectives is to minimize the total of communication costs for messages then the path weight might represent the communication cost per message on the path. Other examples of the path weight are delay on the path, and reliability of the path.

A cost is also incurred in locating a resource at a given node. This cost will be referred as the entity weight for that location. The interpretation of the entity weight also depends on the application and the optimization criteria. If one of the objectives is to minimize the total investment in the resource being installed, the entity weight might represent the cost incurred in buying, installing, and maintaining the resource. Nodal processing delay, and reliability of a node are other examples of the entity weight.

The Computer Resource Location Problem (CRLP) associated with the hierarchical structure described above is stated as follows:

Given: Candidate locations for the resources in the network, the path weights, the entity weights for every candidate location, and the service capacities of entities at each node.

Obtain: The optimal number of resources and their locations on the network, and the optimal interconnections between users and resources.

To minimize: The total cost due to path and entity weights.

Subject to: Capacity constraints.

Mathematically the CRLP is stated as,

P1
Choose x_{ij} and y_j to
Minimize:

$$Z = \sum_{j \in I} \sum_{i \in I} c_{ij} x_{ij} + \sum_{j \in I} f_j y_j \quad (1)$$

Subject to:

$$\sum_{j \in I} x_{ij} = 1, \quad \forall i \in I \quad (2)$$

$$\sum_{i \in I} x_{ij} \leq s_j y_j, \quad \forall j \in I \quad (3)$$

$$x_{ij} - y_j \leq 0, \quad \forall i \in I, j \in I \quad (4)$$

$$x_{ij}, y_j \in (0,1), \quad \forall i \in I, j \in I \quad (5)$$

Where,

Z: the total cost.

L: the number of levels in the hierarchy (given).

I: set of user locations and candidate locations for the resources.

i: an index which represents the location of the users.

j: an index which represents the candidate locations for the resources.

c_{ij}: path weight between i and j.

f_j: entity weight at location j.

s_j: capacity of resource at location j.

x_{ij}: assignment variable which is equal to 1 if the user at location i is connected to resource at location j, and is 0 otherwise.

y_j: location variable which is equal to 1 if a resource is placed at the candidate location j, and is 0 otherwise.

The first term on the right hand side of Eq.(1) represents the cost due to the interconnection of users to resources, and the second term represents costs due to the location of resources. Constraint set (2) ensures that user is adequately served by the resources. Note that we have allowed each user to be only served by one resource. This is frequently referred to as the 'star-star' configuration. Constraint set (3) ensures that the total service demanded from a resource by all users does not exceed its capacity. Constraint set (4) ensures that users are not assigned to a location where a resource does not exist.

3. SIMULATED ANNEALING

Simulated annealing (SA) is motivated by the behavior of mechanical systems with a very large number of degrees of freedom. According to the general principles of physics, any such system can be coaxed into a minimum energy level by a slow annealing process. Kirkpatrick et al. [1983] showed that a fruitful connection existed between simulated annealing and combinatorial optimization involving search for global extrema of a function that had many local extrema. In optimization, simulated annealing works as follows: At relatively high *temperatures* (a parameter of the procedure), many solutions are accepted, even if they are bad. This allows the search to discover the gross features of the problem domain. Successively lower temperatures identify more details, and the solutions become more localized. Ultimately, SA yields a very good solution on a very good mode of the problem domain. We will now present a formal representation of simulated annealing.

We are concerned with the minimization of a function Z: X $\times$ Y $\rightarrow$ R, where X = $[x_{ij} : x_{ij}$ is integer $\forall i \in I, j \in I]$ and Y = $[y_j : y_j \in (0,1), \forall j \in I]$. Let w = $(x_{ij}, y_j) \in$ X $\times$ Y be a particular feasible solution to the problem, and assume that for each w there is a set N(w) $\subset$ X $\times$ Y, which we call the set of neighbors of w. Assume that there is a transition probability matrix P such that P(w, w') > 0 if and only if w' $\in$ N(w). We will denote Z = Z(w). Also let T_1, T_2,..... be a sequence of strictly positive numbers (the temperature), such that

$$T_1 \geq T_2 \geq .... \tag{6}$$

and

$$Lim_{k \rightarrow \infty} T_k = 0 \tag{7}$$

The simulated annealing algorithm, with current solution w, works as follows:

```
0. Let the best solution w* = w.
1. Generate w' ∈ N(w).
2. Set w* to w' with probability
       p = Min [1, exp((Z - Z')/T_k)],
       and leave it unchanged with
       probability 1-p.
3. Replace w with w',
       and repeat steps 1 to 3 until
       solution unchanged for M iterations.
4. Replace T_k with T_{k+1},
       and repeat Steps 1 to 4, until T_k ≈ 0.
```

Note that in Step 2, if Z' < Z, p = 1, and w' is chosen to be the next value of w* for sure. Also conversely if Z' > Z, p< 1, for most values of T_k. The exception is the case when $T_k >>|$ Z - Z' |, which gives exp (Z - Z')/T$k \approx$ 1, and even if Z' > Z. In this case, w' would be chosen to be the next value of w, even though it might give a worse value of Z; i.e. initially, nearly all w are candidates for the best solution. As the temperature decreases, the worse solutions get selected with lower probability.

Hajek [1988], Lundy and Mees [1986] and Mitra et al. [1986] have proved that the simulated annealing algorithm converges asymptotically. The main issue is to obtain a *cooling schedule* $\alpha(T)$, i.e. the trajectory of the temperature reduction denoted by (6) and (7), that will guarantee asymptotic optimality.

We chose a cooling schedule that had the form of an exponential decay:

$$T_k = T_1 e^{-\alpha k} \tag{8}$$

where T_1 is the initial starting temperature, and α is a constant. Note that the cooling schedule given by equation (8) satisfies equations (6) and (7), and the conditions of Hajek's [1988] theorem proving the convergence of simulated annealing. Note that this cooling schedule has more attractive properties than the ones suggested by Hajek [1988] and Lundy and Mees [1986]: it takes longer to find the gross features of the problem domain, and does not linger too long once it has found the mode in which the global optimal solution lies.

Next, we have to decide the value of T_1, the initial temperature. Here there is a trade-off between choosing a value of T_1 so high that *all* w'$\in$ N(w) would be candidates for w in Step 2, thus considerably slowing down the convergence of the algorithm, versus having a T_1 so low that only w' for which Z' < Z are chosen to be the next value of w, where the gross features of the solution space X $\times$ Y are missed leading to local rather than global optima. The latter case also amounts to reducing simulated annealing to a straightforward *steepest descent* algorithm. We chose the initial temperature as follows:

$$T_1 = Min_{x \in X} Max_{y \in Y} Z \tag{9}$$

Note that equation (9) gives the least upper bound of the hierarchical facility location-allocation problem.

Finally, although $Lim_{i \rightarrow \infty} T_i = 0$ guarantees convergence, it may take exponentially long to obtain a result. Thus the choice of T_f, the final temperature determines the final error in the solution. If the total number of possible solutions to problem P1 is W, (note only one of which is the global optima), the sensitivity of optimal solution (i.e. the allowed error bound) is δ, and the error probability is γ, then we need $\gamma/(1-\gamma) \geq (W-1)e^{-\delta/T_f}$ (Lundy and Mees, 1986). This gives us

$$T_f \quad \leq \quad \delta/(log(W-1) - log\gamma) \tag{10}$$

We use equation (10) to set the final temperature for the simulation.

4. HSA: HYBRID SIMULATED ANNEALING

The first algorithm proposed in this paper makes use of a special property of the CRLP and combines linear programming with the simulated annealing technique to solve the CRLP very efficiently. When the location variables (y's) are known and fixed, we can drop constraint (4) in problem P1. This reduces P1 to a special case of the *transportation* problem. Since the constraint matrix of the transportation problem is unimodular, when the right-hand-side s_j are all integers, a straightforward *linear* programming algorithm gives integer (0 or 1) solutions for the x's, even when the integer constraint on the x's are dropped.

Computational experience has shown that a straightforward application of the simulated annealing technique, where both x's and y's are chosen randomly, results in very poor performance. By transforming the CRLP as described above, only the y's need to be generated randomly and, for each y, the optimal x's can be obtained using linear programming techniques. The number of x's is much larger than the number of y's. Thus, our transformation reduces the computation time considerably. We deal with the complexity of HSA in the next subsection.

Algorithm 1

We will now present the algorithm. In order to keep the description concise, we will adopt the following notation:

Z : the total cost as defined by Eq.(1).

Z^*: the total cost of the current optimal solution.

I: set of candidate locations for the resources. Note that these are the same nodes where the users are located.

K^l: subset of I selected for location of resources at the l-th iteration. [i.e. $y_j = 1, \forall j \in K^l$].

n: cardinality of the set I, i.e, the number of candidate locations for the resources.

k^l: cardinality of the set K^l, i.e, the number of resources selected in the l-th iteration.

m: the minimum number of resources required to serve all the k users.

Statement of the algorithm:

```
1. Initialize: set $Z^* = \infty$, temperature, $T=T_1$ using Eq. (9)
            set $l=0$, and $K^0$.
2. $l = l + 1$
            Randomly generate $K^l = N(K^{l-1})$
            Such that,
            $K^l \subset I$
            $m \le k^l \le n$
            $\sum_{j \in K} s_j \ge k^l$
3. For fixed $y_j$, Do The Following:
            Solve the LP,
            $Z = Min \sum_{j \in K^l} \sum_{i \in I} c_{ij} x_{ij}$
            Subject to:
            $\sum_{j \in K^l} x_{ij} = 1, \quad \forall i \in I$
            $\sum_{i \in I} x_{ij} \le s_j y_j, \quad \forall j \in I$
4. Compute $\Delta Z = Z - Z^*$
            If $\Delta Z < 0$, Then $Z^* \leftarrow Z$
            and store current solution.
            Else with Prob = $exp(-\Delta Z/T)$,
            $Z^* \leftarrow Z$ and store current solution.
                If solution unchanged after
                N iterations, go to Step 5.
                Else, go to Step 2.
5. Lower Temperature, $T \leftarrow \alpha(T)$, using Eq. (8).
            If $T > T_f$ (Eq. 10), go to Step 2.
            Else, stop and record optimal solution.
```

Complexity

The first thing to note is that HSA is guaranteed to converge since the cooling schedule (equation 8) we have used satisfies Hajek's [1988] theorem of convergence. Secondly, by our choice of the final temperature (equation 10), we have specified that the final solution will have an error of δ or less with probability γ.

If the cardinality of the set I is N, there are N^2 x-variables and N y-variables, each of which can take a value of either 0 or 1. A total enumeration of the integer programming problem would require 2^{N^2+N} computations. Even a partial enumeration technique like branch-and-bound, or the Lagrangian relaxation technique would have a worst case performance of $O(2^{N^2+N})$ (Parker and Rardin, 1988). Mitra et al. [1986] have shown that the worst case convergence time of *pure* simulated annealing for an optimization problem with n 0-1 parameters is $O(2^{cn})$ where c is some constant. Thus in the problem of resource location in data networks, had we used pure SA we would expect a computational complexity of $O(2^{c(N^2+N)})$. However, we only use simulated annealing for the N y- variables. The x_{ij} for simulated y_j are obtained by solving a linear program in N^2 variables. Decades of experience with the simplex method shows that *observed* behavior of a linear program with m constraints is O(m) (Murthy, 1976). Thus since the number of constraints m = 2N for the problem in Step 3 of Algorithm 1, the *observed* complexity of the hybrid simulated annealing algorithm should be $O(N.2^N)$ which is much smaller than the worst case performance of either pure SA or a Lagragian relaxation technique.

The worst case performance of the simplex method is $O(2^n)$. However, recent theoretical research using probability distributions on linear programming data has shown that the *expected* performance of the simplex method ranges from $O((1/n + 1/(m-n+1))^{-1})$ to $O((1/n + 1/(m-n+1) - 1/m)^{-1})$, where n and m are the number of variables and constraints respectively. (See for instance, Parker and Rardin [1988] and Smale [1983]). Thus, the *expected* complexity of HSA is also $O(N.2^N)$, and we are justified in arguing that the complexity of the hybrid simulated annealing method we propose in this paper would be orders of magnitude less than a partial enumeration technique like Lagrangian relaxation for solving the hierarchical facility location-allocation problem.

5. PSA: PARALLEL SIMULATED ANNEALING

In the second method, we parallelize the simulated annealing procedure and obtain an algorithm that has complexity $O(N^k.2^N)$ where k is a constant. We first use a Lagrangian technique to relax constraint (2) in problem P1. Thus the objective function becomes:

$$ZR(\mu) = \sum_{j\in I}\sum_{i\in I} c_{ij}x_{ij} + \sum_{j\in I} f_j y_j + \sum_{i\in I}\mu_i(\sum_{j\in I} x_{ij} - 1) \tag{11}$$

where $\mu = (\mu_1, ..., \mu_N)$ are the Lagrangian multipliers associated with the N constraints (2). Note that we can rewrite (11) as:

$$ZR(\mu) = \sum_{j\in I}\{\sum_{i\in I}(c_{ij} + \mu_i)x_{ij} + f_j y_j\} - \sum_{i\in I}\mu_i \tag{12}$$

From equation (12) we see that P1 can be written as a set of I problems in j where the j-th problem, for fixed μ, is given by:

P2Rj

Minimize w.r.t. x

$$ZR_j(\mu) = \sum_{i\in I}(c_{ij} + \mu_i)x_{ij} + f_j y_j - \sum_{i\in I}\mu_i \tag{13}$$

Subject to:

$$\sum_{i\in I} x_{ij} \le s_j y_j \tag{14}$$

$$x_{ij} - y_j \le 0, \quad \forall i \in I \tag{15}$$

$$x_{ij}, y_j \in (0,1), \quad \forall i \in I \tag{16}$$

Note that $ZR(\mu) = \sum_{j\in I} ZR_j(\mu)$.

The dual of this problem is given by

P2Dj

Maximize w.r.t. μ

$$ZD_j = ZR_j(\mu_i) \tag{17}$$

Subject to:

$$\mu_i \ge 0 \tag{18}$$

Note that, when $y_j = 0$ for some j, $x_{ij} = 0, \forall i \in I$. Conversely, when $y_j = 1$ for some j, equation (15) can be dropped since it is superfluous due to (16), and problem P2Rj belongs to the class of 'knapsack problems'. The following results apply:

Theorem 1:

Suppose $\mu_i \ge 0$, $\forall i \in I$, and $\mu = (\mu_1, ...\mu_N)$. Also, for fixed y, let Z^*, ZR^*, ZD^* be the optimal solutions of P1, P2R, and P2D respectively. (Here P2R is the combined solution from all the individual P2Rj, and P2D is the same). Then

$$ZR^* \le Z^*$$
$$ZD^* \le Z^* \tag{19}$$

Proof: Modification of proof of Theorem 5.13 in Parker and Rardin [1988] applies.

Theorem 2:

If $x^*_{ij}, \forall i \in I$ solves P2Rj for fixed y_j, and some $\hat{\mu} \ge 0$, and both (i) $\sum_{j\in I} x^*_{ij} = 1$, $\forall i \in I$, and (ii) $\sum_{i\in I}\hat{\mu}_i(\sum_{j\in I} x^*_{ij} - 1) = 0$ hold, then x^*_{ij} solves P1 for that y_j.

Proof: Modification of proof of Theorem 5.14 in Parker and Rardin [1988] applies.

Thus for some value of μ we can obtain at least the lower bound for the problem P1 (by Theorem 1). Also, by changing μ, and resolving P2Rj, $\forall j$, we can obtain the optimal solution of P1 if the conditions of Theorem 2 are satisfied. We need a mechanism for finding the optimal μ's for P2Dj.

One way to change the μ's is by using a subgradient technique (Parker and Rardin, 1988). For fixed μ, $ZR^*(\mu)$ will give the lower bound of the problem; i.e. $ZR^*(\mu) = Z_{LB}$. Suppose we find *feasible* x_{ij}'s such that $\sum_{j\in I} x_{ij} = 1, \forall i \in I$, then we will get $Z_{UB}(\mu)$. We use the following mechanism to change the μ's:

$$w_i^t = r^t(Z_{UB}(\mu_i) - Z_{LB}(\mu_i))/(\sum_{j\in I} x_{ij} - 1)^2 \tag{20}$$

$$\mu_i^{t+1} = Max\{0, \mu_i^t + w_i^t(\sum_{j\in I} x_{ij} - 1)\} \tag{21}$$

where r^t is some parameter such that $r^t \le 2$, $Lim_{t\to\infty} r^t = 0$, and $Lim_{T\to\infty}\sum_{t=1}^{T} = \infty$.

The problem now is solved in two parts: First, we will find the optimal value of the x_{ij}'s for all $y_j = 1$. This involves using the subgradient procedure given in equations (20) and (21). We will call this 'Algorithm 2a', and the results of this procedure are stored. Next we will randomly generate the y_j's and use the stored values from Algorithm 2a to evaluate the objective function Z which is then examined for goodness using the traditional simulated annealing mechanism. We will call this 'Algorithm 2b'.

Algorithm 2a

```
1. Initialize: Pick any $\mu_i^t \ge 0$, $\forall i \in I$, and
               set $t = 1$, and $Z^0 = -\infty$.
               Set $y_j = 1$, $\forall j \in I$

2. Solve P2Rj, $\forall j \in I$, and get $x_{ij}^{t*}$, $\forall i \in I$,
               and $ZD(\mu^t) = \sum_{j\in I} ZR_j(\mu^t)$.

3. If $\sum_{j\in I} x_{ij}^{t*} = 1$,
               and $\sum_{i\in I}\mu_i^t(\sum_{j\in I} x_{ij}^{t*} - 1) = 0$,
               STOP; $x_{ij}^{t*}$ solves P1.
```

```
4. If $Z^{t-1} < ZD(\mu^t)$, let $Z^t = ZD(\mu^t)$.
       Otherwise $Z^t \leftarrow Z^{t-1}$.
5. Compute new point $\mu_i^{t+1}$, $\forall i \in I$
       using (20) and (21)
       Replace $t \leftarrow t+1$, and return to Step 2.
```

Theorem 3:
Algorithm 2a either stops at Step 3 with $ZD^* = Z^*$, or generates a sequence $\{Z^t\}$ satisfying $Lim_{t\to\infty} Z^t = ZD^*$.

Now we will proceed to the simulated annealing part of the algorithm. We will use the same notation as in Algorithm 1.

Algorithm 2b

```
1. Initialize: set $Z^* = \infty$, temperature, $T=T_1$ using Eq. (9)
       set $l=0$, and $K^0$.
2. $l = l + 1$
       Randomly generate $K^l = N(K^{l-1})$
       Such that,
       $K^l \subset I$
       $m \leq k^l \leq n$
       $\sum_{j \in K} s_j \geq k^l$
3. For fixed $y_j \in K^l$, obtain $x_{ij}^{t*}$, $\forall i \in I$
       from Algorithm 2a.
       For $y_j \notin K^l$, set $x_{ij}^{t*} = 0$.
4. Compute $\Delta Z = Z - Z^*$
       If $\Delta Z < 0$, Then $Z^* \leftarrow Z$
           and store current solution.
       Else with Prob = $exp(-\Delta Z/T)$,
           $Z^* \leftarrow Z$ and store current solution.
       If solution unchanged in N iterations,
           go to Step 5.
       Else, go to Step 2.
5. Lower Temperature, $T \leftarrow \alpha(T)$, using Eq. (8).
       If $T > T_f$ (Eq. 11), go to Step 2.
       Else, stop and record optimal solution.
```

Complexity

In order to analyze the complexity of this problem, recall that problem P2Rj is a Knapsack problem in x_{ij}'s whenever the y_j's are 1. Since there are N, x_{ij}'s for each fixed y_j, the knapsack problem is solved using dynamic programming techniques in O(N) (Denardo, 1982). When the $y_j = 0$, $x_{ij} = 0$ also. In order to generate x values from each subproblem P2Rj, j=1,...,J, for $y_j = 1$, $\forall j$, it will take O(N^2M) iterations where the subgradient technique takes M steps. Usually $M \leq N^k$ where k is some positive integer, and $k << \infty$. Suppose we store all these results, then we will have all the information that is necessary to run the entire simulated annealing algorithm. In the worst case, 2^N possible values of y need to be checked. Thus the overall complexity of the problem is O($N^k 2^N$), where usually $k < 3$.

6. A LAGRANGIAN RELAXATION ALGORITHM

In order to compare the computational performance of the greedy heuristic and the algorithm based on simulated annealing presented in the last two sections to a more traditional operations research technique, we develop a Lagrangian relaxation algorithm for the hierarchical facility location- allocation problem. To our knowledge, the method developed in this section is different from others in the computer resource location literature. (See for instance, the paper by Gavish and Pirkul [1986] and references therein). It is similar to the algorithm developed by Mirzaian [1985] for the location of concentrators in a communications network. The Lagrangian relaxation technique will also provide a lower bound for the problem which will enable us to compare the *goodness* of our approximate algorithms by estimating how close they get to this lower bound.

Recall problem P1 given by equations (1) to (5). We will relax equations (2) and (3) by incorporating them into the objective function with the associated Lagragian multipliers λ_i, (i $\in I$), and γ_j, (j $\in I$). Then, for all values of λ and γ, we need to find x_{ij} and y_j to solve the problem:

P3

$$Z(\lambda,\gamma) = Min_{x,y} \sum_{j\in I}\sum_{i\in I}(c_{ij} - \lambda_i + \gamma_j)x_{ij} + \sum_{j\in I}(f_j - \gamma_j s_j)y_j + \sum_{i\in I}\lambda_i \qquad (22)$$

Subject to:

$$x_{ij} - y_j \leq 0, \quad \forall i \in I, j \in I \qquad (23)$$

$$x_{ij}, y_j \in (0,1), \quad \forall i \in I, j \in I \qquad (24)$$

Note that, except for the integrality conditions, there is only one constraint in problem P3. The theorem below characterizes the optimal solution of the problem, for *given values* of λ and γ.

Theorem 4:
Let J(j) = [i $\in I$: $(c_{ij} - \lambda_i + \gamma_j) < 0, \forall j \in I$], and let $\Omega_{ij} = f_j - \gamma_j s_j + \sum_{i\in J(j)}(c_{ij} - \lambda_i + \gamma_j)$. An optimal solution of P3 is:

$$y_j = \begin{cases} 1 & \text{if } \Omega_{ij} \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

$$x_{ij} = \begin{cases} y_j & \text{if } i \in J(j) \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

Proof: See Appendix

The next issue is to find the optimal λ and γ. Let ZL be the objective function value of the linear relaxation of problem P1; i.e. ZL = Z given by equation (1) where the x's and y's are linear. Given Theorem 1, and noting that minimizing the primal is equivalent to maximizing the dual in *linear programs*, i.e. $ZL = \text{Max}_{\lambda,\gamma}\ Z(\lambda, \gamma)$, we get the following result:

Theorem 5:
The optimal solution of the linear relaxation of the hierarchical resource location problem is $ZL^* = Z(\lambda^*, \gamma^*)$ where λ^* and γ^* are the optimal solutions of P4 given by:
P4

$$ZL = Max_{\lambda,\gamma} \quad -\sum_{j \in I} \eta_j + \sum_{i \in I} \lambda_i \tag{27}$$

$$\lambda_i - \gamma_j - \beta_{ij} \leq c_{ij} \tag{28}$$

$$-\eta_j + \sum_{i \in I} \beta_{ij} \leq d_j \tag{29}$$

$$\lambda_i, \gamma_j, \eta_j, \beta_{ij} \geq 0, \quad i \in I, j \in I \tag{30}$$

Proof: See Appendix

Clearly since we are dealing with *integer programs*, ZL obtained from solving problem P4 (given by equation 27) will be different from the optimal solution for the entire problem Z, given by equation (1). This 'duality gap' exists for all integer programming problems. It should be noted that we can use problem P4 to obtain the *lower bound* for the problem, which is the value of ZL at the optimum. We will estimate the distance of the solutions of the approximate algorithms to this lower bound. The Lagrangian procedure outlined above can be modified with a subgradient technique to obtain a near optimal solution as opposed to a lower bound. We have not done this because our main reason for using the technique was to analyze the performance of the approximate algorithms. It should be noted that much less computation time would be required for obtaining the lower bound.

7. COMPUTATIONAL RESULTS

HSA and PSA are coded in C, with LINDO used for the transportation (linear programming) part of HSA. The Lagrangian relaxation, was also coded in C. The computational performance of all algorithms was investigated using an IBM-PC-AT with a math coprocessor for data networks with 5, 10, and 20 nodes. For each size of network, we generated 25 problems, and for each problem, we estimated the amount of cpu time each algorithm took to satisfy the relevant stopping criterion, recorded the lowest objective function value achieved by each approximate algorithm, and estimated how close it was to the lower bound produced by the Lagrangian Relaxation technique for the same problem.

The test problems were generated randomly as follows: The link weights for links between every pair of nodes in the network are picked randomly from a uniform distribution. Dijkstra's shortest path algorithm was used to find the shortest path between every pair of nodes in the fully connected network. The total path weight between each pair of nodes was obtained by summing the weights of the links on the shortest path between the two nodes. The entity weights were also selected randomly from a uniform distribution. All the nodes in the network were treated as candidate locations for every level.

The results of the three methods are summarized in Table 1. For networks with 5 to 10 nodes, HSA was within 6 percent of the lower bound, and for the 15 node data network, it was within 10 percent. These are very good results. The time taken by the LR was 50 percent lower than that of the HSA algorithm for the small network, and was around 15 percent lower for networks with 10-20 nodes. However, it should be noted that the LR only produces a lower bound. If it were combined with a subgradient technique to obtain near optimal solutions, the computation time could be much longer.

The results for PSA have not been completed as yet.

8. CONCLUDING REMARKS

In this paper we formulated the problem of locating resources in a computer network as an integer programming problem, and presented two algorithms based on simulated annealing for solving it. The approximate algorithms both had advantages over a traditional operations research technique based on Lagrangian relaxation. HSA obtained solutions very close to the lower bound, although the computational time was quite large. The Lagrangian relaxation algorithm took a considerable amount of time to compute the lower bound.

What we have shown is that a nontraditional search technique such as simulated annealing could be adapted to the solution of an important problem in computer communications network design. Given the advent of cheap fast desktop computing capabilities, there is a strong argument for using the computer to perform intelligent search for solutions. Also, since design is generally not an on-line activity, the long computation time required by these nontraditional search techniques is not really a problem. There is a much greater need for obtaining solutions that are close to being optimal.

9. APPENDIX

Proof of Theorem 4

We will provide a logical proof. Note that all d_j's and a_j's are strictly positive. Thus if all $(c_{ij} - \lambda_i + \gamma_j)$ are also strictly positive, then the solution is trivial, and for optimality all x_{ij}'s and y_j's are set to zero. This trivial solution is given by the Theorem.

Suppose now that $(c_{ij} - \lambda_i + \gamma_j) \leq 0$ at some $i^* \in I, j^* \in I$. $Z(\lambda, \gamma)$ is minimized if the user at i^* is allocated to the facility at j^*, provided that the location cost $d_j - \gamma_j a_j$ at j^* does not exceed the allocation cost. Continuing this line of argument, many users can be serviced by the facility located at one node (say j^*), so long as the location cost does not exceed the sum of allocation costs. This provides the condition for the y_j. Clearly, only users with negative allocation cost will be serviced by a facility located at any of the nodes. This gives the condition for the x_{ij}'s. □

Proof of Theorem 5

Given Theorem 1, and noting that $ZL = \text{Max}_{\lambda,\gamma}\, Z(\lambda, \gamma)$, and letting $f_j = d_j - \gamma_j s_j$, we get that

P5

$$ZL = Max_{\lambda,\gamma} \sum_{j\in I} Min[0, d_j + \sum_{i\in J(j)} (c_{ij} - \lambda_i + \gamma_j)] + \sum_{i\in I} \lambda_i \tag{31}$$

We can rewrite this as

$$Z = Max_{\lambda,\gamma} \quad -\sum_{j\in I} \eta_j + \sum_{i\in I} \lambda_i \tag{32}$$

$$\eta_j \geq -f_j - \sum_{i\in J(j)} (c_{ij} - \lambda_i + \gamma_j)] \tag{33}$$

$$\lambda_i, \gamma_j, \eta_j \geq 0, \qquad i \in I, j \in I \tag{34}$$

We can rewrite the second term on the right-hand-side of equation as follows:

$$-\sum_{i\in J(j)} (c_{ij} - \lambda_i + \gamma_j)] = Max \sum_{i\in I} (\lambda_i - c_{ij} - \gamma_j) w_{ij} \tag{35}$$

$$s.t. \qquad 0 \leq w_{ij} \leq 1 \tag{36}$$

The dual of the above problem is

$$Min \sum_{i\in I} \beta_{ij} \tag{37}$$

$$s.t. \qquad \beta_{ij} \geq \lambda_i - c_{ij} - \gamma_j \tag{38}$$

$$\beta_{ij} \geq 0 \tag{39}$$

The above is true for all $j \in I$. Incorporating equations (37)-(39) into equations (32)-(34), we can rewrite problem P5 as that given in Theorem 5. □

10. REFERENCES

1. Chen, P. and J. Akoka [1980] 'Optimal Design of Distributed Information Systems', *IEEE Transactions on Computers*, vol. C-29, December, pp 617-625.
2. Collins, W. E., R. W. Eglese, and B. L. Golden [1987] Simulated Annealing An Annotated Bibiliography, Working Paper Series no. Ms/S 87-02, College of Business and Management, University of Maryland, College Park, 62 pages.
3. Denardo, E. V. [1982] *Dynamic Programming: Models and Applications*, (Englewood-Cliffs, NJ: Prentice-Hall).
4. Fisher, M. L. and D. S. Hochbaum [1980] 'Database Location in Computer Networks', *Journal of the Association of Computer Machinery*, vol. 27, no. 4, pp 718-735.
5. Gavish, B. and H. Pirkul [1986] 'Computer and Database Location in Distributed Computer Systems', *IEEE Transactions on Computers*, vol. C-35, July, pp 583-590.
6. Hajek, B. [1988], 'Cooling Schedules for Optimal Annealing', *Mathematics of Operations Research*, vol. 13, no. 2, pp 311-321.
7. Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi [1983] "Optimization By Simulated Annealing", *Science*, vol. 220, May, pp 671-680.
8. Lundy, M. and A. Mees [1986], 'Convergence of an Annealing Algorithm', *Mathematical Programming*, vol. 34, pp 111-124.
9. Mirzaian, A. [1985] 'Lagrangian Relaxation for the Star-Star Concentrator Location Problem: Approximation Algorithm and Bounds', *Networks*, vol. 15, pp 1-20.
10. Mitra D., F. Romeo, and A. S. Vincentelli [1986] 'Convergence and Finite Time Behavior of Simulated Annealing', *Advances in Applied Probability*, vol. 18, Sept., pp 747-771.
11. Morgan, H. L. and K. D. Levin, [1977] 'Optimal Program and Data Locations in Computer Networks', *Communications of the Association of Computer Machinery*, vol. 20, no. 5, pp 315-321.
12. Murthy, K. G. [1976] *Linear and Combinatorial Programming*, (New York: Wiley).
13. Parker, R. G. and R. L. Rardin [1988] *Discrete Optimization*, (San Diego: Academic Press).
14. Smale, S. [1983] 'On the Average Number of Steps in the Simplex Method of Linear Programming', *Mathematical Programming*, vol. 27, pp 241-262.

ACKNOWLEDGEMENTS

Supported partially through Unisys Corporation grant 5-24535. The research assistance of Ashok Vernekar at the initial stages of this project is gratefully acknowledged.

AUTHOR'S BIOGRAPHY

G. ANANDALINGAM is an assistant professor of Systems Engineering at the University of Pennsylvania. Previous to this he was an assistant professor of Systems Engineering at the University of Virginia. He received a B. A., M. A. degree in electrical sciences from Cambridge University (England) in 1975, and S. M. and Ph. D. degrees in systems engineering from Harvard University in 1977 and 1981 respectively. His current research interests include computer communications networks, hierarchical optimization, and applied statistics.

Table 1
Results of Numerical Analysis

Problem Size (nodes)	Algorithm	Per Cent Superior*	Distance from LB+	Computation Time Ave. cpu sec.
5	Parallel SA			
5	Hybrid SA	40%	5.3%	66.2
5	Lagrangian Relaxation	-	-	33.8
10	Parallel SA			
10	Hybrid SA	68%	5.3%	900.0
10	Lagrangian Relaxation	-	-	776.7
20	Parallel SA			
20	Hybrid SA	60%	8.1%	1900.6
20	Lagrangian Relaxation	-	-	1691.7

* The algorithm produced a solution with a lower objective function value for this percentage of the problems. Percentages do not add up to 100% because for some problems, the solutions were identical.
+ LB = Lower Bound