NETWORK CONFIGURATION AND MACHINE LAYOUT IN FIXED-PATH MATERIAL HANDLING SYSTEMS

Khaled S. Al-Sultan
Department of Systems Engineering
King Fahd University
of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

Yavuz A. Bozer
Department of Industrial & Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 92-6
January 1992

NETWORK CONFIGURATION AND MACHINE LAYOUT IN FIXED-PATH MATERIAL HANDLING SYSTEMS†

Khaled S. Al-Sultan‡

Department of Systems Engineering
King Fahd University
of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

Yavuz A. Bozer

Department of
Industrial and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109, USA

ABSTRACT

In this paper we address a difficult combinatorial problem that arises in designing fixed-path material handling systems, where handling occurs over a pre-defined and fixed route which connects various "sites" that are represented by a unique set of pick up and deposit points. Examples of such systems include power-and-free conveyors, monorails, in-floor towline conveyors, and automated guided vehicle systems. The objective of this study is to determine an efficient system design by simultaneously considering the configuration of the fixed path and the layout, i.e., the processor (or machine) assigned to each site. Past studies in this area have focused either on optimizing the path configuration for a given layout or on optimizing the layout for a simple path configuration. In this study we show how simulated annealing can be applied to obtain good solutions to the problem when both the path configuration and the processor locations are determined concurrently. Since the above two problems are closely coupled, significant savings can be achieved by considering them simultaneously.

†This study was partially supported by Dr. Bozer's Presidential Young Investigator Award under NSF Grant DMC-8858562.

‡This study was conducted while Dr. Al-Sultan was at the Department of Industrial and Operations Engineering, University of Michigan. His stay was supported by the King Fahd University of Petroleum and Minerals.

1. INTRODUCTION

A fixed–path material handling system is characterized by a "rail" or "guidepath" which defines a fixed route that a handling "device" has to follow in transporting a load from one point to another. Power–and–free conveyors, monorails, in-floor towline conveyors, and automated guided vehicle (AGV) systems represent well-known examples of fixed–path material handling systems. (The interested reader may refer to Tompkins and White [25, Ch. 6], for an excellent review of fixed–path material handling systems and others.) Due to the structure of the main aisles/columns and the nature of the handling devices, in most (if not all) fixed–path material handling systems, the devices follow a rectilinear path (i.e., the l_1 –norm) and the flow is unidirectional. One exception to the latter is bidirectional AGVs. However, mostly due to congestion and delays that result from path contention in bidirectional systems, a majority of current AGV systems use unidirectional guidepaths.

Each *processor* in the facility must be assigned to a unique site which is represented by a unique pick up and deposit (P/D) point. A processor may represent a single machine, a group of machines, or a "cell." (We implicitly assume that the processors have similar space requirements.) Following a unidirectional fixed path, a device picks up a load at the pick up point of a site and transports it to the deposit point of the appropriate site. A problem often encountered by an engineer who must design a fixed-path material handling system is the layout of the processors (i.e., assigning a processor to each site) and the configuration of the fixed path. Of course, the location of the sites along the fixed path, and the throughput capacity of the system (as a function of the number of devices) are also important design variables.

Using simulated annealing, in this study we address a basic design step which must precede throughput evaluation. Assuming that the site locations are given and fixed, and that the overall structure of the fixed path (i.e., an *undirected network*) is also given and fixed, we determine the assignment of processors to sites (i.e., the layout) and the configuration of the network simultaneously. (By the network configuration, here we are referring to the direction assigned to each edge in the network.) Past studies in this area have focused either on optimizing the path configuration for a given layout or on optimizing the layout for a simple path configuration. In this study we show how simulated annealing can be used when both the network configuration and the layout are determined concurrently. Since the above two problems are closely coupled, significant savings can be achieved by considering them simultaneously. Of course, alternative site locations and alternative undirected networks, where the latter is obtained by adding or removing certain path segments or "shortcuts," can also be evaluated by systematically varying the input and rerunning the proposed algorithm.

The paper is organized in six sections. In the next section, we define some additional terms and formally state the problem. In section 3, we present the literature review. In section 4, we develop and present a simulated annealing approach to the problem along with appropriate notation. In section 5 we evaluate the effectiveness of our algorithm through numerical experiments. For two of the example problems presented in section 5, we compare the solution obtained from simulated annealing against the optimum. In section 6 we state our conclusions and suggest directions for future research in this area.

2. PROBLEM DESCRIPTION

Consider the undirected network shown in Figure 1a, where there are two types of nodes. The first type represents a node which has been identified as a potential site for a processor. We will refer to such nodes as "site nodes." The second type of node is defined simply to designate turning or intersection points in the network; we will refer to such nodes as "network nodes." Assuming that there are four processors which must be assigned to four sites, in Figure 1a nodes 1 through 4 are site nodes, while the remaining nodes are network nodes. In general, we will use n to designate the number of nodes in the network (including the site nodes), and m ($m \leq n$) to designate the number of sites (or site nodes), which is assumed to be equal to the number of processors. Also, we let M designate the set of site nodes in a network. For example, in Figure 1a, $n = 9$, $m = 4$, and $M = \{1, 2, 3, 4\}$.

The objective is to obtain a solution (i.e., a network configuration and a layout) that minimizes the product of the flows between the processors and the distance over which these flows occur. Let $f_{k\ell}$ denote the flow from processor k to processor ℓ. (We assume the flow data is given.) Let d_{ij} denote the *shortest path* from site i to site j. Note that the d_{ij} values can be obtained only after the network configuration has been determined. The objective is to minimize $\sum_k \sum_\ell \sum_{i \in M} \sum_{j \in M} f_{k\ell} d_{ij}$ while assigning exactly one processor to each site and vice versa.

An example solution for the problem is shown in Figure 1b, where the arcs have been numbered 1 through 11. Note that each arc has been assigned a direction and each processor (enclosed by a triangle) has been assigned to a site. We will refer to such a solution as a *network-layout configuration.* A *network configuration* alone refers to a directed network, while a *layout configuration* alone refers to a particular assignment of processors to sites. An unqualified reference to a "configuration" implies a "network-layout configuration."

The above problem is a difficult combinatorial problem and it is doubtful that it can be formulated in closed form, primarily due to the shortest path problems involved. Even if one were to formulate the above problem in closed form, obtaining exact solutions to the resulting model would be quite unlikely. Note that, if a network configuration is given, the above problem becomes a Quadratic Assignment Problem (QAP). The QAP is NP-hard (Garey and Johnson [8]) and generally very time consuming to solve if there are 15 or more sites. Hence, the problem addressed in this paper, that is, determining the optimum network-layout configuration, is also NP-hard. Note that the distance values used as inputs for the QAP are decision variables in our problem.

A remaining assumption we make for the study is concerned with the above d_{ij} values obtained from a network configuration. We assume that these values are obtained simply by solving a shortest path problem over the current network configuration with given arc lengths. Since we do not consider "timing" issues and congestion that may potentially develop in certain segments of the network, d_{ij} does not necessarily represent the *shortest time* route from site i to site j. Although selecting the shortest distance versus the shortest time route is an important issue, it is too detailed to be considered at the early planning stage where the proposed algorithm will be used.

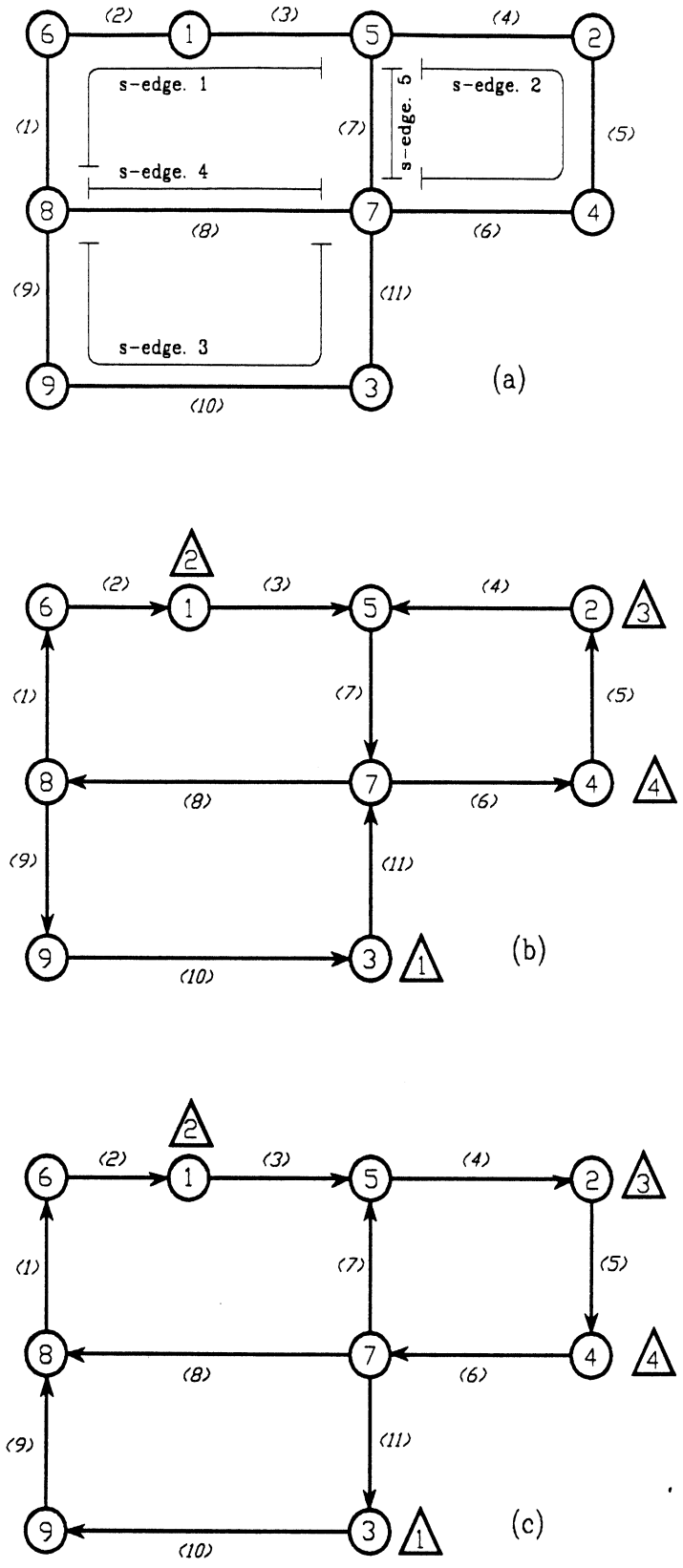


Figure 1. Network and Layout Configurations.

3. LITERATURE REVIEW

As remarked earlier, past studies have focused either on optimizing the path configuration for a fixed layout or on optimizing the layout for a simple path configuration. A number of studies of the first type have been reported in the literature. These studies have been performed in the context of *unidirectional* AGV systems where the optimum guidepath configuration is determined by assigning a direction to each edge in a given undirected network. Gaskins and Tanchoco [9] formulate the problem as a zero–one integer programming problem where the objective is to minimize loaded vehicle travel. The authors assume that the layout and the location of all the P/D points are given and fixed. The flow matrix, which is expressed as a from–to chart, is also supplied by the user. The above information is then "translated" into a node–arc network, where the nodes represent corners, intersections, and P/D points while the arcs represent possible routes of travel for the AGVs.

Gaskins and Tanchoco [9] demonstrate the model and show that good results are obtained for a 22–node problem. However, according to Evans, Wilhelm, and Usher [7], the model is "very cumbersome to apply since the integer program formulation becomes very large even for relatively small problems." Also, Bakkalbaşı and McGinnis [3] report that the model may generate infeasible or non-optimal solutions.

An alternative formulation to the AGV guidepath optimization problem is presented by Kaspi and Tanchoco [13], who emphasize that their methodology is equally applicable to "flow path" design in "fixed–path material handling systems." The model size still increases quite rapidly with the problem size. According to the authors, a problem with 30 nodes (ten of which are P/D points) requires more than 10,000 variables. As a solution procedure, the authors propose a branch–and–bound algorithm with "depth–search first and backtracking." The algorithm is applied to an example problem with 23 nodes (nine of which are P/D points) and 33 undirected edges. The example problem and the corresponding solution obtained in [13] are discussed later in section 5.

Another paper on guidepath optimization in AGV systems is presented by Goetz and Egbelu [10], who "build on the contributions" of Gaskins and Tanchoco [9]. The two principal differences between [9] and [10] is that, in [10] the P/D point locations can be altered to improve system performance, and a heuristic procedure is used to reduce the resulting problem size. The department (or processor) locations, however, remain fixed. That is, the layout problem is not considered in the model.

A few studies have addressed the layout problem for a given simple path configuration. Heragu and Kusiak [12] present heuristic approaches for the case where the machines are arranged in a circular or straight-line fashion. They also consider the case where the machines are "clustered." The authors assume that the site locations are not known *a priori* since they depend on the type of machine assigned to each site. Such may be the case if the machines vary considerably in size and they are "tightly packed" with little or no distance between them. Using dynamic programming, Picard and Queyranne [21] present an exact and "fast" procedure for the straight-line case, which is also known as the one-dimensional space allocation problem.

Relative to the machine sizes, if there is non-negligible distance between the machines (due to aisles, maintenance areas, other handling or storage requirements, etc.) and/or

the machines are similar in size, then site locations can be determined *a priori* regardless of the type of machine ultimately assigned to each site. In this case the problem becomes the QAP. Bozer and Rim [5], Kiran, Unal, and Karabati [14], Rim and Bozer [22], Leung [18], Kouvelis and Kim [16] have studied the case where the path is a simple closed-loop with no shortcuts, i.e., the circular layout problem. In each case the overall objective is to find the "best" layout around the loop. The above studies differ in the particular type of objective function used and the direction of flow around the loop (which can be unidirectional or bidirectional). Given predetermined site locations, Kouvelis and Chiang [17] study the straight-line single-row layout problem. Their objective is to determine the layout which minimizes the total backtracking distance of the material handling device. With predetermined site locations, the straight-line layout problem can also be solved with the DP algorithm developed by Picard and Queyranne [21].

4. SIMULATED ANNEALING

In this section, we present a simulated annealing algorithm to solve the problem defined in section 2. Although simulated annealing was first discussed as a concept by Metropolis et al. [20], it was only recently that Kirkpatrick, Gelatt, and Vecchi [15] and Cerny [6] independently established an analogy between the annealing process in thermodynamics and optimization algorithms. (For a brief discussion of this analogy, the reader may refer to Al-Sultan [2] and Lundy and Mees [19]). Since its introduction, simulated annealing has been successfully applied to solve difficult combinatorial problems such as the Traveling Salesman Problem (Cerny [6]), cluster analysis (Al-Sultan [2]), the QAP (Wilhelm and Ward [26]), and the Multiple Choice Knapsack problem (Al-Sultan [1]).

Combinatorial optimization problems typically possess many local minima. Descent methods usually converge to one of these locally optimal points and terminate there since there is no mechanism that forces the algorithm out of a local optimum. One of the strengths of simulated annealing that makes it suitable for tackling such difficult combinatorial problems is that, unlike descent methods, it generally does not terminate at the first local minimum encountered. In fact, with simulated annealing, at any point during the execution of the algorithm, the probability of making an "uphill" move is always positive. Thus, the algorithm can move out of a local minimum, and given sufficient time, one would expect to obtain a globally optimal (or near-optimal) solution.

Convergence of the annealing algorithm is discussed by Romeo and Sangiovanni-Vincentelli [23] and Lundy and Mees [19]. Hajek [11] provides a set of necessary and sufficient conditions for the annealing algorithm to converge to a global minimum. However, these convergence proofs are only of theoretical interest, and of little practical value, since the design of the algorithm requires the selection of certain values for a set of parameters (which is highly problem dependent) as well as some knowledge about the curvatures of the functions involved (which is normally not available *a priori*). Therefore, an annealing algorithm should be viewed as a heuristic approach. As with many other heuristic procedures, with very few exceptions, there are no guarantees on the absolute quality of the resulting solutions.

The proposed algorithm starts with an arbitrary network–layout configuration which we refer to as the *current solution.* At each iteration, we use the current network con-

figuration to generate a new one. As defined below, if the new network configuration is infeasible, i.e., it is not strongly connected, we simply generate another one. Once a feasible network configuration is obtained, we generate a layout configuration and compute the cost of this *trial network-layout configuration.* (We implicitly assume that all layout configurations are feasible.) If the cost of the trial solution is less than that of the current solution, we accept the trial solution as the current solution and update the current configuration accordingly. Otherwise, we accept the trial solution (as the current solution) with a certain non-zero probability, which is controlled by a parameter known as the "temperature." The algorithm usually starts with a high initial temperature, say, T_0, which reflects a high probability of accepting trial solutions that are worse than the current solution.

Throughout the algorithm we keep track of the best solution and update it as necessary. If the best solution does not decrease in value within, say, NI iterations, we reduce the current temperature by a factor of α $(0 < \alpha < 1)$, which corresponds to reducing the above probability. We continue in this fashion until the current temperature reaches a low final temperature, say, T_f, at which point the system is said to have reached a "freezing stage" in annealing terminology. To formally present the above scheme, we need to define the following terms:

1. A Network Configuration:

Recall that a network configuration is a directed network. For ease of exposition, we will adopt the following convention: if a horizontal (vertical) arc points to the right (up), it is assigned a value of one. Otherwise, it is assigned a value of zero. For example, using this convention, the network configuration shown in Figure 1b can be represented by the one dimensional array {1,1,1,0,1,1,0,0,0,1,1}, where the first position in the array corresponds to arc 1, the second position to arc 2, and so on. The above array, which defines a network configuration, will be designated by N.

2. Checking Network Feasibility:

A newly generated network configuration may or may not be feasible. We say that a network configuration is feasible if it is *strongly connected*, i.e., each node can be reached from all the other nodes in the network. Given a network configuration, we use the following routine to check for feasibility:

Routine FEAS:

a. Forward Pass:

0. Initialization: Let S denote an array of size n, where $S(q)$ corresponds to network node q. Set $S(q) = 0$ for $q = 1, 2, \ldots, n$.
1. Set $S(1) = 1$. Flag node 1 as "active" and all the other nodes as "inactive."
2. Select an active node from S and label it as node r. If there are no active nodes in S, **STOP.** Otherwise, for each arc originating at node r and terminating at node q, set $S(q) = 1$ and flag node q as active, provided that node q has not been already fathomed.
3. Fathom node r. Go to 2.

When the above algorithm stops, if all the elements of S are equal to one, we invoke the "backward pass" described below. If there are any elements of S that are equal to zero, we declare the network configuration infeasible and stop the feasibility checking procedure. Note that any node q for which $S(q) = 0$ is a node causing the infeasibility.

b. Backward Pass:

This routine is identical to the forward pass, except that in step 2 we search for each arc originating at node q and terminating at node r. When the backward pass stops, if all the elements of S are equal to one, we declare the network configuration feasible; otherwise, we declare it infeasible. As before, if the network configuration is infeasible, any node q for which $S(q) = 0$ is a node causing the infeasibility. For more information on strong connectivity in networks, the reader may refer to Tarjan [24], among others.

Remarks:

1. Some network configurations may be *feasible* even if they are *not strongly connected.* Note that, for a network configuration to be feasible, a path from site node i to site node j is required only if there is non-zero flow from the processor located at site i to that located at site j. For example, although the network configuration shown in Figure 2 is not strongly connected, it will be feasible if f_{12} and f_{32} are both equal to zero. (Note that f_{21} and f_{23} need not be equal to zero.) Nevertheless, we require strong connectivity to maintain flexibility in accommodating possible future changes in the flow patterns.

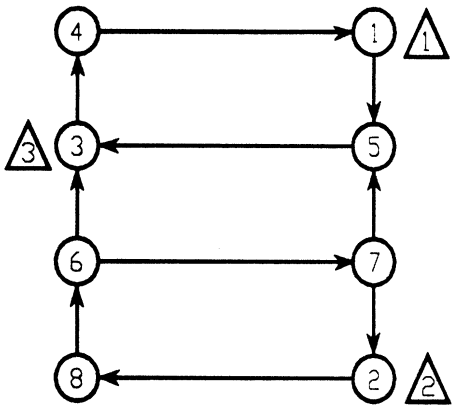


Figure 2. Network Feasibility and Strong Connectivity.

2. If a network configuration contains at least one incoming arc and at least one outgoing arc at each node, it is not necessarily feasible. For example, in the network configuration shown above in Figure 2, each node has at least one incoming arc and one outgoing arc. However, if f_{12} or f_{32} is greater than zero, the network configuration will not be feasible.

3. For computational reasons, we define a *super-edge* of the network as a set of edges that must have the same direction in order for any network configuration to be feasible. For example, as shown in Figure 1a, edges 1, 2, and 3 constitute one super-edge. In any feasible

network configuration, all the edges in this super-edge must have the same direction. In Figure 1b, note that the super-edge in question has a "clockwise" direction, while super-edge 2 (which represents arcs 4, 5, and 6) has a "counterclockwise" direction. Once a specific direction is assigned to a super-edge, we will refer to it as a *super-arc*. Obviously, if an edge does not belong to any super-edge, it is considered a super-edge by itself. (See, for example, edge 7 in Figure 1a.)

3. Layout Configuration:

Recall that, given a set of sites and an equal number of processors, a layout configuration defines an assignment of processors to sites. The result is a vector of cardinality m, which we denote by L. For example, in Figure 1b, L can be represented by the following one dimensional array: {2,3,1,4}, where the first site has been assigned processor 2, the second site has been assigned processor 3, and so on. Since the original flow matrix is given in terms of flow between the processors (rather than the sites), we use L to map this flow matrix into flow between sites. That is, we let $F(L) = (f_{ij}(L))$, where $f_{ij}(L)$ represents the flow from the processor located at site i to the processor located at site j according to layout configuration L.

4. Shortest Path Routine:

Given a feasible network configuration, N, the shortest path routine (SHPT) computes the shortest paths between all pairs of site nodes in the network. The output of this routine is a distance matrix. Since this distance matrix obviously depends on N, we denote it by $D(N) = (d_{ij}(N))$. Since computing shortest paths is a standard algorithm in network programming, we will not discuss it here. The interested reader may refer to Bazaraa and Jarvis [4], or any other text on networks.

5. Computing the Cost of a Network–Layout Configuration:

Given a feasible network configuration, N, the SHPT routine can be used to compute $D(N)$. Also, recall that for a given layout configuration, L, the flow matrix is denoted by $F(L)$. Hence, the cost of a network–layout configuration, say, $f(N, L)$, can be expressed as follows

$$f(N,L) = \sum_{i \in M} \sum_{j \in M} f_{ij}(L)\, d_{ij}(N). \tag{1}$$

We assume that the value of the above objective function is computed by the cost function (COFN) routine.

6. Neighbors of a Network Configuration:

There is no exact mathematical definition of a neighbor of a given network configuration. However, we loosely define a neighbor to be a network configuration that is somehow related to the current one. Since there is no unique strategy for generating a neighbor, we propose the following routine which we denote by NGB1:

1. Map the given network configuration into one consisting of only super-arcs.
2. Given a scalar $0 < p^N < 1$ (a parameter), for each super-arc u draw a random number $p_u \sim U(0,1)$. If $p_u < p^N$, retain the current direction of super-arc u; otherwise, reverse its direction.

The above routine is not only simple but it preserves the super-arc structure of the network and when it is used many times, it is likely to cover most of the state space (i.e., possible network configurations). As an example, consider the network configuration shown earlier in Figure 1b. Mapping it into one of super-arcs translates the network array into the array $\{1,0,0,0,0\}$, where a one represents a clockwise direction, and a zero represents a counterclockwise direction if the super-arc contains more than one arc; otherwise, we follow the same convention stated earlier (i.e., one is up or right, zero is down or left). Suppose we set $p^N = 0.6$ and randomly draw the p_u values as follows: 0.2, 0.8, 0.7, 0.4, 0.9, for $u = 1,\ldots,5$, respectively. The resulting neighbor would be given by $\{1,1,1,0,1\}$, which translates into the following array in terms of the original arcs: $\{1,1,1,1,0,0,1,0,1,0,0\}$. The resulting neighbor for the network configuration is shown in Figure 1c.

Note that NGB1 does not take into account the feasibility of the neighbor. Therefore, it is possible that the neighbor we obtain from NGB1 is infeasible. This, of course, will be detected by the FEAS routine discussed earlier. The value of p^N clearly affects the "shakeup" generated in the current network configuration; the smaller the value of p^N, the more "shakeup" is likely to result; i.e., more arcs will have their directions reversed in the new network configuration.

7. Neighbor of a Layout Configuration:

As it is the case for a network configuration, there is no exact mathematical definition of a neighbor of a given layout configuration. Therefore, as before, we loosely define a neighbor to be a layout configuration that is somehow related to the current one. We propose the following routine, say, NGB2, to generate a neighbor from a given layout configuration:

1. Given a scalar $0 < p^L < 1$ (a parameter), set $j = 1$ and draw a random number $p_j \sim U(0,1)$.
2. If $p_j \geq p^L$, interchange the processors located at sites j and $j+1$; otherwise, do nothing.
3. Set $j = j+1$. If $j = n$, **STOP.** Otherwise, repeat steps 1 and 2 with the new j value.

The relationship between p^N and the "shakeup" of the network configuration discussed above also applies to p^L and the "shakeup" of the layout configuration. That is, the smaller the value of p^L, the more likely that more pairs of processors will exchange locations.

THE ALGORITHM:

Initialization Step:

Select values for the parameters p^N, p^L, α, NI, an initial temperature, T_0, and a final temperature, T_f. Next select an arbitrary network-layout configuration, (N, L). Set $(N_t, L_t) = (N_c, L_c) = (N, L)$, where t designates the "trial" configuration and c designates the "current" configuration. Use the FEAS routine to check if the network configuration, N, is feasible. If N is feasible, use the SHPT routine to compute $D(N)$ and the COFN routine to compute $f(N, L)$; set $f_t = f_c = f_b = f(N, L)$ and $(N_b, L_b) = (N, L)$, where b designates the "best" solution. If N is not feasible, set $f_t = f_c = f_b = \infty$. In either case, set $T = T_0$, $j = 0$, and go to the main step.

Main Procedure:

1. Given the current network configuration, N_c, use the NGB1 routine to generate a neighbor, i.e., a new network configuration, N_t.
2. Use the FEAS routine to check if N_t is feasible. If it is infeasible go to 1; otherwise, use the SHPT routine to compute the distance matrix $D(N_t)$, and go to 3.
3. Given the current layout configuration, L_c, use the NGB2 routine to generate a neighbor, i.e., a trial layout configuration, L_t.
4. Use the COFN routine to compute $f_t = f(N_t, L_t)$. If $f_t < f_b$, set $N_c = N_b = N_t$, $L_c = L_b = L_t$, $f_c = f_c = f_t$, $j = 0$, and go to 1; otherwise, go to 5.
5. If $f_t < f_c$, set $N_c = N_t$, $L_c = L_t$, $f_c = f_t$, and go to 7; otherwise, go to 6.
6. Draw a random number $p \sim U(0, 1)$. If $p \leq \exp[(f_c - f_t)/T]$, set $N_c = N_t$, $L_c = L_t$, and $f_c = f_t$.
7. If $j < NI$, set $j = j + 1$, and go to step 1; otherwise, set $T = \alpha T$. If $T < T_f$, **STOP**; N_b and L_b represent the best solution with a cost of f_b. Otherwise, set $j = 0$ and go to 1.

5. COMPUTATIONAL EXPERIENCE

The algorithm described in the previous section was slightly modified as follows: the maximum number of temperature reductions allowed, say, TR, is specified by the user ahead of time; if the number of temperature reductions reaches TR before the current temperature reaches T_f, then the algorithm stops. Imposing an upper limit on the number of temperature reductions is common in simulated annealing.

As one might expect, our preliminary computational results with the above algorithm clearly indicated that a substantial fraction of the runtime is devoted to generating feasible network configurations, i.e., steps 1 and 2 of the main procedure. To reduce the computational burden associated with generating feasible network configurations, we made two minor adjustments. First, we modified step 2 as follows:

2a. Use the FEAS routine to check if N_t is forward feasible. If N_t is forward feasible, then go to 2c. Otherwise, identify a node, say, node q, for which $S(q) = 0$ $(1 \leq q \leq n)$ and reverse the direction of one of the arcs associated with that node. Go to 2b.

2b. Use the FEAS routine to check if N_t is forward feasible. If N_t is forward feasible, go to 2c; otherwise, go to 1.

2c. Use the FEAS routine to check if N_t is backward feasible. If N_t is backward feasible, then use the SHPT routine to compute the distance matrix $D(N_t)$, and go to 3. Otherwise, identify one node for which $S(q) = 0$ $(1 \leq q \leq n)$ and reverse the direction of one of the arcs associated with that node. Go to 2d.

2d. Use the FEAS routine to check if N_t is backward feasible. If N_t is backward feasible, go to 2e; otherwise, go to 1.

2e. Use the FEAS routine to check if N_t is forward feasible. If N_t is forward feasible, then use the SHPT routine to compute the distance matrix $D(N_t)$, and go to 3. Otherwise, go to 1.

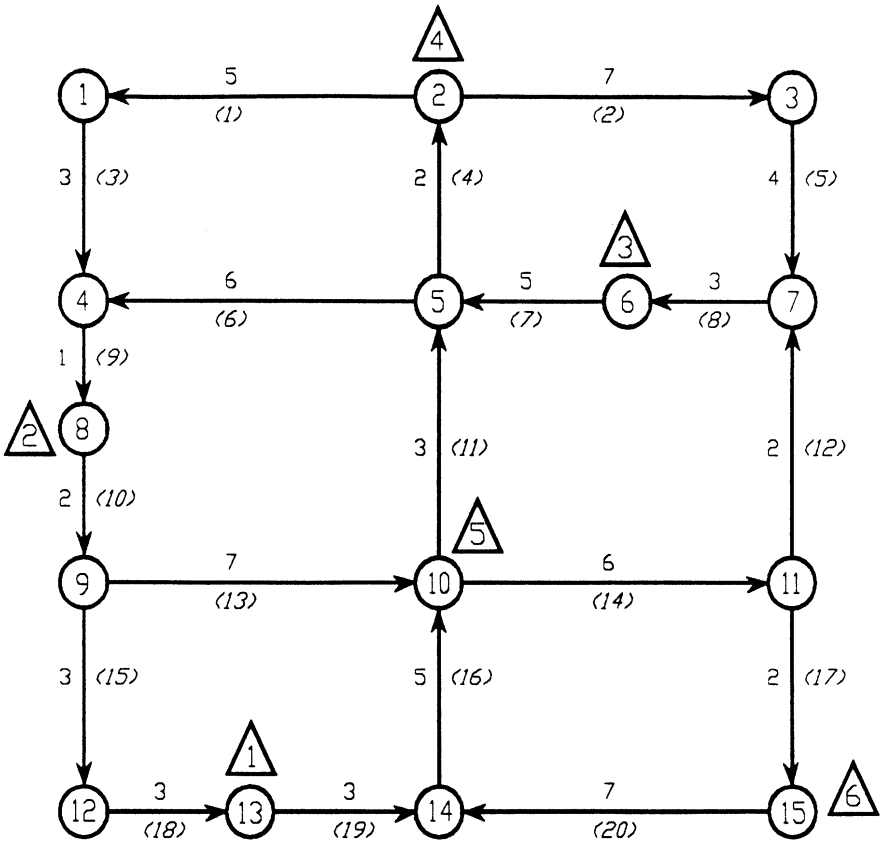
Note that step 2e is required because we need to reevaluate forward feasibility if the direction of an arc is reversed in step 2c.

The second adjustment we made is based on the fact that generating a trial layout configuration is straightforward since any layout configuration is assumed to be feasible. Thus, it is computationally desirable to generate, say, 50 trial solutions by generating 50 trial layout configurations from a single (feasible) network configuration. Of course, when 50 trial solutions are generated from a network configuration, the algorithm attempts to generate a new trial network configuration, i.e., a network neighbor as described in steps 1 and 2. Once a network neighbor is obtained, the next 50 trial solutions are generated again by varying only the layout configuration, and so on. Whether the network configuration is fixed or not, the iteration counter (designated by j in the main procedure) is incremented with each trial solution evaluated.

The above algorithm was applied to three example problems. The network and the flow data for the first problem are shown in Figure 3, where there are 6 site nodes and 9 network nodes. (Momentarily disregard the direction assigned to each edge and the station located at each site node.) The distance (or travel time) associated with each edge is also shown in Figure 3, where the numbers in parenthesis denote the edge number; for brevity, we will not show the super-edge numbers. Likewise, the network and the flow data for the second and third problems are shown in Figures 4 and 5, respectively. (The third problem was taken from [13].) In Figure 5, note that edges 1, 2, 3, 4, 30, and 33 involve a turn without the corresponding network node. Rather than modify the original problem by adding a node for each turn, we simply modeled the above edges as super-edges. Recall that with super-arcs a 1 indicates a clockwise direction while a 0 indicates the opposite.

The results obtained for all three problems are shown in Table 1, where 50 or 100 layout configurations have been generated from each network configuration. Each entry in the Table shows three results: the objective function value of the best solution obtained, the time at which the best solution was obtained, and the total runtime. (All the runtimes are expressed in seconds and are based on a 25 MHz 80386 DOS-based computer with a 80387 numeric coprocessor.)

As shown in Table 1a, the overall best objective for the first problem is $z = 1,157$ units which is obtained in 62.28 seconds. The corresponding network–layout configuration is shown in Figure 3. For ease of exposition, the sites are numbered in increasing order of the node number. For example, site 1 is at node 2, site 2 is at node 6, site 3 is at node 8, and so on. Thus, for the best solution we have $L = \{4, 3, 2, 5, 1, 6\}$. Using a brute-force approach, we determined that the configuration shown in Figure 3 is also an optimal

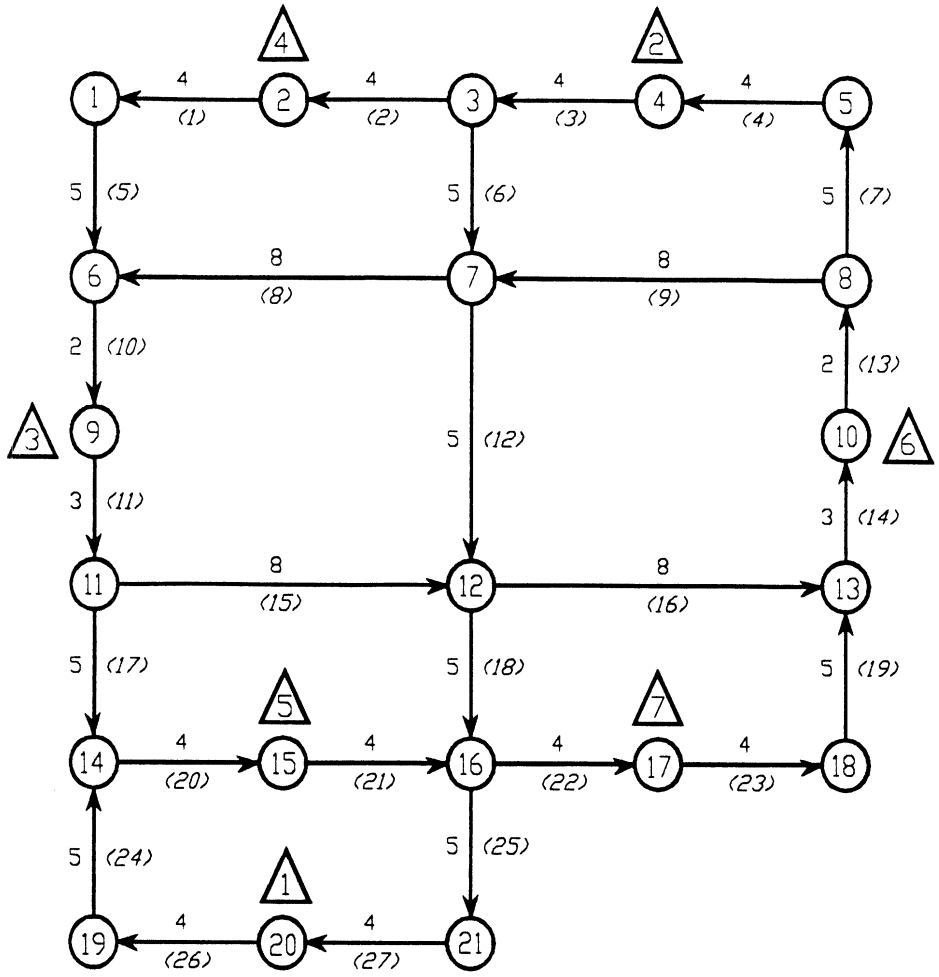


(a) Network-Layout Configuration.

	1	2	3	4	5	6
1	0	2	3	5	3	0
2	1	0	2	1	4	1
3	0	2	0	12	1	2
4	2	7	0	0	2	0
5	3	8	8	14	0	9
6	0	1	2	0	9	0

(b) The Flow Data (From-To Chart).

Figure 3. Data for First Example Problem.

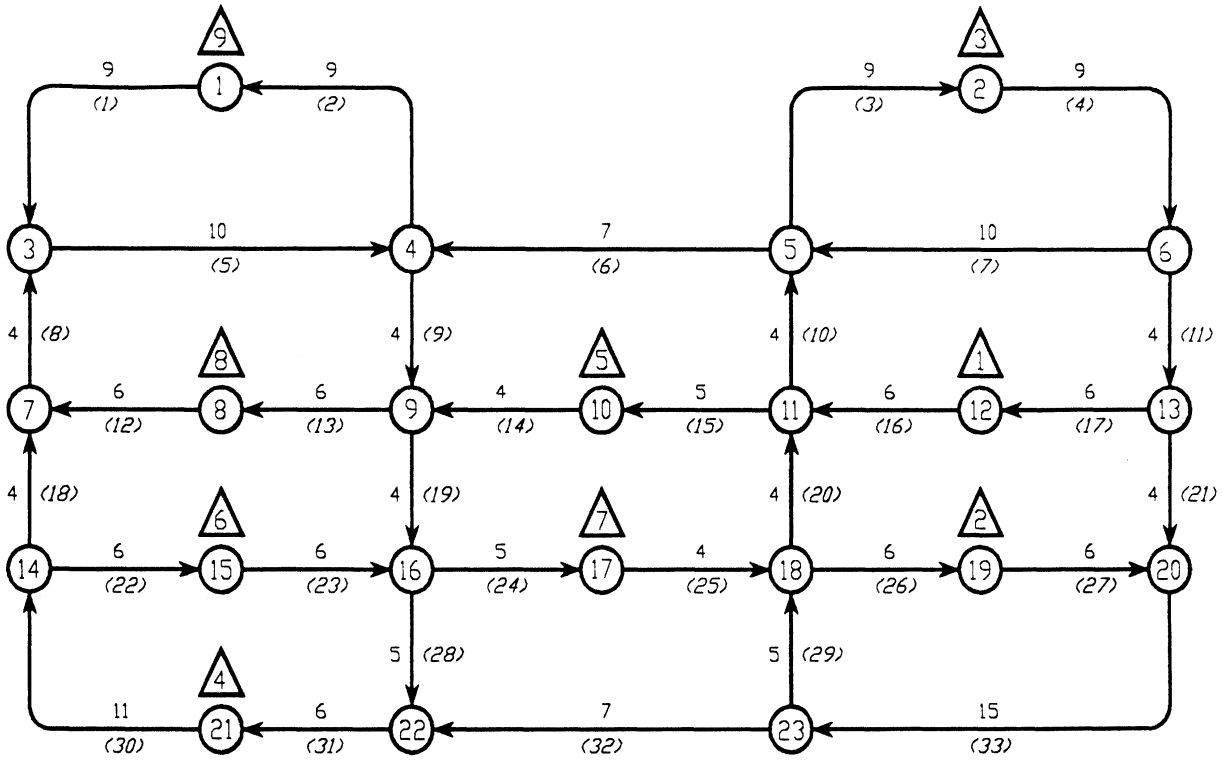


(a) Network-Layout Configuration.

	1	2	3	4	5	6	7
1	0	2	3	0	9	2	0
2	0	0	12	0	0	4	5
3	3	2	0	0	14	8	1
4	0	3	9	0	2	2	5
5	4	0	0	0	0	3	1
6	2	11	3	0	12	0	8
7	1	7	1	0	0	11	0

(b) The Flow Data (From-To Chart).

Figure 4. Data for Second Example Problem.



(a) Network-Layout Configuration.

	1	2	3	4	5	6	7	8	9
1	0	0	20	20	50	0	0	0	30
2	0	0	0	0	0	0	0	0	0
3	0	0	0	5	0	10	0	5	0
4	0	0	0	0	0	10	5	10	10
5	0	30	0	0	0	0	25	15	0
6	0	10	0	10	0	0	0	0	0
7	0	30	0	0	20	0	0	0	0
8	0	30	0	0	0	0	0	0	0
9	0	0	0	0	0	0	20	10	0

(b) The Flow Data (From-To Chart).

Figure 5. Data for Third Example Problem. (Taken from [13]).

TABLE 1. SIMULATED ANNEALING RESULTS.

(a) First Example Problem.

	TR: 200 PL, PN: 0.5		LAYOUTS: NI: 50 200			TR: 200 PL, PN: 0.5		LAYOUTS: NI: 100 400		
BEST OBJ/ BESTTIME/ TOTAL TIME	To: 10	60	100	200	AVERAGE	To: 10	60	100	200	AVERAGE
ALPHA: 0.5	**1373**	**1382**	**1357**	**1325**	**1359**	**1357**	**1228**	**1353**	**1257**	**1299**
	0.77	6.26	2.14	7.69		15.44	5.88	15.77	19.72	
	6.54	10.93	12.02	14.83		15.49	21.64	21.53	25.11	
0.7	**1345**	**1343**	**1259**	**1286**	**1308**	**1227**	**1257**	**1340**	**1295**	**1280**
	8.90	16.43	20.43	19.56		4.39	2.19	17.80	42.02	
	12.03	19.39	20.54	26.64		20.49	33.39	39.22	44.77	
0.8	**1335**	**1294**	**1230**	**1367**	**1307**	**1337**	**1330**	**1157**	**1343**	**1292**
	11.65	4.06	18.18	30.21		29.55	9.50	20.92	10.93	
	17.47	28.67	32.24	37.24		35.04	51.25	62.28	65.41	
AVERAGE	**1351**	**1340**	**1282**	**1326**	**1325**	**1307**	**1272**	**1283**	**1298**	**1290**

(b) Second Example Problem.

	TR: 200 PL, PN: 0.5		LAYOUTS: NI: 50 200			TR: 200 PL, PN: 0.5		LAYOUTS: NI: 100 400		
BEST OBJ/ BESTTIME/ TOTAL TIME	To: 10	60	100	200	AVERAGE	To: 10	60	100	200	AVERAGE
ALPHA: 0.5	**3591**	**3470**	**3795**	**3580**	**3609**	**3884**	**3493**	**3619**	**3642**	**3660**
	5.93	2.85	5.66	6.65		6.70	25.37	19.28	28.67	
	13.63	11.20	16.76	16.86		18.67	28.89	25.32	32.40	
0.7	**3614**	**3507**	**3482**	**3533**	**3534**	**3402**	**3404**	**3626**	**3480**	**3478**
	15.77	13.95	25.21	15.87		24.77	2.86	1.21	8.57	
	19.56	24.66	26.86	29.39		28.17	39.33	40.31	49.87	
0.8	**3712**	**3602**	**3648**	**3486**	**3612**	**3601**	**3479**	**3503**	**3472**	**3514**
	18.02	15.99	31.80	28.39		11.92	53.99	43.44	73.32	
	23.12	36.36	43.50	46.19		36.09	63.44	68.27	79.53	
AVERAGE	**3639**	**3526**	**3642**	**3533**	**3585**	**3629**	**3459**	**3583**	**3531**	**3550**

(c) Third Example Problem.

	TR: 200 PL, PN: 0.5		LAYOUTS: NI: 50 200			TR: 200 PL, PN: 0.5		LAYOUTS: NI: 100 400		
BEST OBJ/ BESTTIME/ TOTAL TIME	To: 10	60	100	200	AVERAGE	To: 10	60	100	200	AVERAGE
ALPHA: 0.5	**10000**	**10040**	**9280**	**10290**	**9903**	**9820**	**10010**	**9590**	**10310**	**9933**
	9.23	17.30	0.77	26.04		14.11	5.99	21.37	42.02	
	20.82	23.72	26.31	32.79		23.56	31.48	33.95	47.13	
0.7	**10130**	**9920**	**10130**	**10230**	**10103**	**9490**	**9810**	**9750**	**9570**	**9655**
	12.41	40.21	32.68	43.94		7.42	19.06	20.16	39.93	
	26.80	48.89	49.65	54.27		35.10	63.72	69.21	72.44	
0.8	**10370**	**9600**	**9740**	**9310**	**9755**	**9630**	**9400**	**9160**	**9790**	**9495**
	9.51	5.06	71.89	45.64		38.12	82.94	26.69	85.36	
	41.69	66.63	80.46	89.42		55.42	95.63	107.27	120.13	
AVERAGE	**10167**	**9853**	**9717**	**9943**	**9920**	**9647**	**9740**	**9500**	**9890**	**9694**

configuration; i.e., $z^* = 1,157$ units for the first problem. Furthermore, in Table 1a, the worst solution ($z = 1,382$ units) is about 20% above optimal, and 10 solutions (out of 24) are within approximately 10% of the optimal solution.

A fundamentally different, two-stage heuristic approach one may use to determine an alternative network–layout configuration can be outlined as follows: in the first stage, given the distance (or travel time) for each (undirected) edge in the network, we determine the shortest path between all the site nodes. (This is analogous to bidirectional travel with no path contention.) Using these distance values as input, we solve the layout configuration problem as a QAP. In the second stage, we determine the optimal network configuration for the layout configuration obtained from the first stage. The above procedure is a heuristic procedure only because the problem is solved in two stages. Otherwise, the solution obtained at each stage is obviously an optimal one.

Applying the above "alternate heuristic" to the first problem, we obtained $z = 1,254$ units for which $L = \{4,3,1,5,2,6\}$ and the same network configuration shown in Figure 3a. This solution is approximately 8% above optimal. We wish to emphasize that we are presenting the alternate heuristic and the results obtained from it only as a benchmark; i.e., we do not intend to perform a direct and rigorous comparison between the alternate heuristic (and its possible variations) and the simulated annealing algorithm. Also, the alternate heuristic was implemented on a mainframe computer because we wanted to obtain exact solutions at both stages.

The results obtained for the second problem are shown in Table 1b. The overall best objective is $z = 3,402$ units which is obtained in 28.17 seconds. The corresponding layout and network configurations are given as follows: $L = \{1,4,6,2,5,3,7\}$ and $N = \{1,1,0,0,1,0,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0\}$. The optimal configuration for the second problem (which was obtained by brute-force) is shown in Figure 4a for which $L = \{4,2,3,6,5,7,1\}$ and $z^* = 3,207$. The worst solution obtained in Table 1b ($z = 3,884$) is about 21% above optimal, and 13 out of 24 solutions are within approximately 10% of the optimal solution. For the second problem, the alternate heuristic generates the following results: $z = 3,343$ units (which is about 4% above optimal), $L = \{1,5,3,6,2,7,4\}$ and $N = \{1,1,1,1,1,1,0,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1\}$. Note that the best simulated annealing solution is about 1.8% above the solution obtained from the alternate heuristic.

Similarly, the results obtained for the third problem are shown in Table 1c. The overall best objective is $z = 9,160$ units which is obtained in 107.27 seconds. Due to its size, we were not able to obtain the optimal configuration for the third problem. With the alternate heuristic, however, we obtained $z = 8,960$ units, $L = \{3,6,4,8,2,1,5,7,9\}$ and

$$N = \{1,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,0,0,1,0,1\}.$$

Thus, the best simulated annealing solution is about 2.2% above the alternate heuristic solution. In [13], with a *fixed* layout configuration given by $L = \{1,8,2,9,7,3,5,6,4\}$, the authors obtain

$$N = \{1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0\}$$

and $z = 10,170$ units (which is approximately 11% above the best solution obtained with simulated annealing). We emphasize that the result obtained in [13] is cited only to show the impact of not varying the layout while configuring the network.

To further investigate the impact of optimizing only the network configuration while keeping the layout fixed, we generated three *arbitrary* layouts for the first example problem (shown earlier in Figure 3) and determined the optimal network configuration for each layout configuration. The three layout configurations are given by $L = \{2,3,4,5,6,1\}$, $L = \{6,5,4,3,2,1\}$, and $L = \{1,2,3,4,5,6\}$ for which we obtained $z = 1,347$, $z = 1,467$, and $z = 1,482$ units, respectively. Since $z^* = 1,157$ units for problem 1, we empirically observe that optimizing the network configuration alone leads to solutions that are 16% to 28% above optimal. Repeating the same type of experiment for the second example problem (which was shown earlier in Figure 4) we obtained solutions that were 13% to 19% above optimal. Hence, combined with the results we obtained from the alternate heuristic (which uses a QAP-based layout configuration), our empirical results suggest that the layout configuration plays an important role in determining overall solution quality, and that it is generally better to configure both the layout and the network concurrently even if it forces one to solve the problem heuristically.

As far as heuristic solutions are concerned, we note that the results obtained from the alternate heuristic are slightly better than those obtained from simulated annealing for the second and third example problems. However, we also note that the longest PC-based runtime observed in Table 1 is only 65.41, 79.53, and 120.13 seconds for the three problems, respectively. The network–layout configuration problem is an important design problem with significant cost implications. Coupled with the fact that such design problems are not solved frequently, it is clear that longer runs than those shown in Table 1 are justified. Of course, one of the advantages of simulated annealing is that, one could simply change the parameter values shown in Table 1 to obtain longer runs. However, this is likely but not guaranteed to improve the quality of the resulting solutions.

In order to allow longer runs than those shown in Table 1 with guaranteed improvements in solution quality, we made the following simple but important observation: given a fixed layout configuration and a feasible network configuration, another feasible network configuration can be obtained simply by reversing the direction of each arc in the original network configuration. Unless the flow matrix is symmetric, the new network configuration represents an alternative and potentially better solution. Hence, we changed the original algorithm such that a "mirror image" of the current network configuration is maintained and evaluated along with the original one. To ensure that the annealing process follows the same path, we simply maintained the "mirror solution" in the "background;" that is, we did not let it affect any of the decisions made during the annealing process. (We forced the process to follow the same path only to guarantee that the results obtained from the revised algorithm will be no worse than those obtained in Table 1.)

The results obtained from the revised algorithm are shown in Table 2. It is clearly observed that the "average" performance of the algorithm has improved (particularly for the third problem). Of course, since the optimal solution was obtained for the first problem, there is no further improvement. The best solution for the second problem, however, decreases from $z = 3,402$ units (in Table 1b) to $z = 3,389$ units (in Table 2b). This new solution is about 5.7% above optimal and only 1.4% above the solution obtained from the alternate heuristic. It is obtained with $L = \{6,7,2,4,3,5,1\}$ and $N = \{0,0,0,0,0,1,1,0,0,0,0,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1\}$. Furthermore, 16 out

TABLE 2. SIMULATED ANNEALING RESULTS WITH NETWORK REVERSAL.

(a) First Example Problem.

	TR: 200 PL, PN: 0.5		LAYOUTS: NI: 50 200			TR: 200 PL, PN: 0.5		LAYOUTS: NI: 100 400		
BEST OBJ/ IMPRV %/ BESTTIME/ TOTAL TIME	To: 10	60	100	200	AVERAGE	To: 10	60	100	200	AVERAGE
ALPHA: 0.5	**1373**	**1246**	**1318**	**1297**	**1309**	**1337**	**1228**	**1346**	**1257**	**1292**
	0.0%	9.8%	2.9%	2.1%		1.5%	0.0%	0.5%	0.0%	
	1.32	4.39	18.73	2.75		5.71	9.99	31.08	33.34	
	10.77	17.85	19.82	24.39		26.20	36.80	36.69	42.51	
0.7	**1345**	**1267**	**1259**	**1286**	**1289**	**1227**	**1257**	**1263**	**1246**	**1248**
	0.0%	5.7%	0.0%	0.0%		0.0%	0.0%	5.8%	3.8%	
	14.50	3.18	33.45	31.92		7.36	3.57	47.08	25.93	
	19.61	31.86	33.66	43.56		34.71	56.58	66.41	76.02	
0.8	**1299**	**1294**	**1230**	**1273**	**1274**	**1284**	**1258**	**1157**	**1267**	**1242**
	2.7%	0.0%	0.0%	6.9%		4.0%	5.4%	0.0%	5.7%	
	5.65	6.76	29.71	42.40		23.56	18.24	35.37	22.19	
	28.83	46.96	52.95	61.07		59.54	86.95	105.51	110.84	
AVERAGE	**1339**	**1269**	**1269**	**1285**	**1291**	**1283**	**1248**	**1255**	**1257**	**1261**

(b) Second Example Problem.

	TR: 200 PL, PN: 0.5		LAYOUTS: NI: 50 200			TR: 200 PL, PN: 0.5		LAYOUTS: NI: 100 400		
BEST OBJ/ IMPRV %/ BESTTIME/ TOTAL TIME	To: 10	60	100	200	AVERAGE	To: 10	60	100	200	AVERAGE
ALPHA: 0.5	**3523**	**3470**	**3649**	**3580**	**3556**	**3884**	**3493**	**3600**	**3642**	**3655**
	1.9%	0.0%	3.9%	0.0%		0.0%	0.0%	0.5%	0.0%	
	5.71	4.23	8.68	10.10		10.88	40.87	3.63	45.86	
	20.38	17.30	25.27	24.93		30.43	46.53	40.65	51.85	
0.7	**3535**	**3507**	**3482**	**3479**	**3501**	**3402**	**3404**	**3626**	**3480**	**3478**
	2.2%	0.0%	0.0%	1.5%		0.0%	0.0%	0.0%	0.0%	
	23.40	20.71	37.95	0.66		39.16	4.62	2.04	13.46	
	28.94	37.02	40.53	43.88		44.87	63.22	65.14	80.52	
0.8	**3680**	**3599**	**3509**	**3482**	**3568**	**3545**	**3389**	**3404**	**3472**	**3453**
	0.9%	0.1%	3.8%	0.1%		1.6%	2.6%	2.8%	0.0%	
	18.35	2.63	35.60	39.93		18.95	38.66	89.53	118.75	
	35.27	55.69	64.76	70.19		58.22	102.32	110.29	128.74	
AVERAGE	**3579**	**3525**	**3547**	**3514**	**3541**	**3610**	**3429**	**3543**	**3531**	**3528**

(c) Third Example Problem.

	TR: 200 PL, PN: 0.5		LAYOUTS: NI: 50 200			TR: 200 PL, PN: 0.5		LAYOUTS: NI: 100 400		
BEST OBJ/ IMPRV %/ BESTTIME/ TOTAL TIME	To: 10	60	100	200	AVERAGE	To: 10	60	100	200	AVERAGE
ALPHA: 0.5	**10000**	**10040**	**9280**	**10060**	**9845**	**9820**	**9840**	**9590**	**9760**	**9753**
	0.0%	0.0%	0.0%	2.2%		0.0%	1.7%	0.0%	5.3%	
	11.42	22.35	1.04	25.65		21.15	9.89	31.26	19.33	
	25.92	30.76	33.01	41.63		34.71	44.38	49.82	67.07	
0.7	**9730**	**9240**	**9790**	**9760**	**9630**	**9490**	**9720**	**9480**	**9540**	**9558**
	4.0%	6.9%	3.4%	4.6%		0.0%	0.9%	2.8%	0.3%	
	33.01	3.18	32.24	50.42		11.42	46.41	27.57	97.10	
	34.49	31.86	64.59	70.91		52.34	90.63	98.81	105.45	
0.8	**10370**	**9490**	**8740**	**9310**	**9478**	**9630**	**9180**	**9010**	**9790**	**9403**
	0.0%	1.2%	10.3%	0.0%		0.0%	2.3%	1.6%	0.0%	
	12.36	23.84	30.15	58.39		54.93	69.05	141.05	120.50	
	52.90	85.85	103.31	115.24		80.03	137.43	148.80	170.59	
AVERAGE	**10033**	**9590**	**9270**	**9710**	**9651**	**9647**	**9580**	**9360**	**9697**	**9571**

of 24 solutions are within approximately 10% of the optimal solution.

Similar improvements are obtained for the third example problem. The results are shown in Table 2c, where the best solution decreases from $z = 9,160$ units (in Table 1c) to $z = 8,740$ units for which $L = \{9,3,8,5,1,6,7,2,4\}$ and the network configuration is shown in Figure 5a. Note that the best simulated annealing solution is actually better than the solution of $z = 8,960$ units obtained from the alternate heuristic. We also note that the solution obtained in [13] for a fixed layout (i.e., $z = 10,170$ units) is now approximately 16% above the best simulated annealing solution.

To maintain the layout problem as general as possible, throughout the study we assumed that any processor can be located at any site. If space is limited at some sites, or certain processors have special requirements, one may wish to limit the site(s) at which a particular processor may be located. With minor modifications, our algorithm can accommodate such constraints imposed on the layout.

6. CONCLUSIONS AND FUTURE RESEARCH

The numerical results obtained in the study are summarized in Table 3 where we show the best solutions obtained from simulated annealing for each example problem along with the alternate heuristic solution and the optimal solution. (Recall that the alternate heuristic was implemented on a mainframe computer since the two subproblems involved were solved optimally.) Considering the quality of the solutions we obtained and the PC-based runtimes reported in section 5, the algorithm presented here suggests that simulated annealing is an effective heuristic to solve the network–layout configuration problem which is a difficult combinatorial problem that arises in the design of fixed-path material handling systems.

The model we used for the study can be enhanced in two primary directions. First, one may wish to include empty device travel in the objective. In the model presented here, we aimed at minimizing only the product of the flows and distances which is equivalent to minimizing loaded device travel. In some systems, such as AGV systems, empty device travel plays an important role since a device may be forced to travel empty a non-negligible fraction of the time. If the frequency of empty trips from one site node to another is estimated, one may incorporate these trips as "flows" into the data and adopt the same objective used here.

The second enhancement is concerned with the layout aspect of the problem. In this study, we treated the layout problem as a machine assignment problem where a set of "processors" are assigned to a set of predetermined sites. At a higher level, however, the "layout problem" involves departments instead of machines and the objective is to develop an efficient "block layout," where each department is represented as an entity with a unique, non-zero area requirement. In such cases, the concurrent development of the block layout and the network configuration appears to be a challenging and promising problem to pursue. The exact number and location of the P/D points would also be part of the network configuration problem.

Table 3. Summary of Computational Results.

	Problem 1	Problem 2	Problem 3
Simul. Anneal.			
Original Algo.	1157	3402	9160
Revised Algo.	1157	3389	8740
Alternate Heur.*	1254	3343	8960
Optimum	1157	3207	??

(*) The alternate heuristic is based on a sequential optimization procedure. The results shown were obtained on a mainframe computer by first optimizing the layout configuration (by solving a Quadratic Assignment Problem where the number of "sites" is equal to the number of site nodes in the network) and then optimizing the network configuration (by brute force). The latter problem can also be formulated as a large-scale Integer Programming Problem (see [13]).

BIBLIOGRAPHY

[1] Al-Sultan, K. S., "A Simulated Annealing Algorithm for The Multiple Choice Knapsack Problem," *Presented at the ORSA/TIMS Joint Spring Meeting*, Nashville, TN, May 1991

[2] Al-Sultan, K. S., "Efficient Global Algorithms for Hard and Fuzzy Clustering," Unpublished M.S. Thesis, King Fahd University of Petroleum and Minerals, Saudi Arabia, 1987.

[3] Bakkalbaşı, Ö. and McGinnis, L. F., *Personal Communication.*

[4] Bazaraa, M. S. and Jarvis, J. J., *Linear Programming and Network Flows*, John Wiley & Sons, 1977, New York.

[5] Bozer, Y. A. and Rim, S., "Exact Solution Procedures for the Circular Layout Problem," Technical Report 89–33, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI 48109.

[6] Cerny, V., "Thermodynamical Approach to Traveling Salesman Problem: An Efficient Simulation Algorithm," *Journal of Optimization Theory and Applications*, Vol. 45, No. 1, 1985, pp. 41–51.

[7] Evans, G. W., Wilhelm, M. R. and Usher, J. S., "A Modular Algorithm for Pre-Simulation Design of Automatic Guided Vehicle Systems," *Proceedings of the 1990 Material Handling Research Colloquium*, June 19–21, Hebron, Kentucky, pp. 333–340.

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

[9] Gaskins, R. J. and Tanchoco, J. M. A., "Flow Path Design for Automated Guided Vehicle Systems," *International Journal of Production Research*, Vol. 25, No. 5, 1987, pp. 667–676.

[10] Goetz, W. G., Jr. and Egbelu, P. J., "Guide Path Design and Location of Pick-Up/Drop-Off Points for an Automated Guided Vehicle System," *International Journal of Production Research*, Vol. 28, No. 5, 1990, pp. 927–942.

[11] Hajek, B., "Cooling Schedules for Optimal Annealing," *Mathematics of Operations Research*, Vol. 13, 1988, pp. 311–329.

[12] Heragu, S. S. and Kusiak, A., "Machine Layout Problem in Flexible Manufacturing Systems," *Operations Research*, Vol 36, No. 2, 1988, pp. 258–268.

[13] Kaspi, M. and Tanchoco, J. M. A., "Optimal Flow Path Design of Unidirectional AGV Systems," the *International Journal of Production Research*, Vol. 28, No. 6, 1990, pp. 1023–1030.

[14] Kiran, A. S., Unal, A. T., and Karabati, S., "A Location Problem on Unicyclic Networks: Balanced Case," Working Paper 89–11, Department of Industrial and Systems Engineering, University of Southern California, Los Angeles.

[15] Kirkpatrick, S., Gelatt, C. and Vecchi, M., "Optimization by Simulated Annealing," *Science*, Vol. 20, No. 4598, 1983, pp. 671–680.

[16] Kouvelis, P. and Kim, M. W., "Unidirectional Loop Network Layout Problem in Automated Manufacturing Systems," to appear in *Operations Research.*

[17] Kouvelis, P. and Chiang, W. C., "Linear Single-Row Machine Layout in Automated Manufacturing Systems," Working Paper 89/90–4–3, Department of Management, The University of Texas at Austin.

[18] Leung, J., "A Graph-Theoretic Heuristic for Designing Loop-Layout Manufacturing Systems," Technical Report 90–12, Department of Operations Research, Yale University, New Haven, CT 06511.

[19] Lundy, M. and Mees, A., "Convergence of an Annealing Algorithm," *Mathematical Programming*, Vol. 34, No. 1, 1986, pp. 111-124.

[20] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N. and Teller, A. H., "Equation of State Calculation by Fast Computing Machines," *The Journal of Chemical Physics*, Vol. 21, 1953, pp. 1087–1092.

[21] Picard, J. C. and Queyranne, M., "On the One-Dimensional Space Allocation Problem," *Operations Research*, Vol. 29, No. 2, 1981, pp. 371–391.

[22] Rim, S. and Bozer, Y. A., "The Bidirectional Circular Layout Problem," Working Paper, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI 48109.

[23] Romeo, F. and Sangiovanni-Vincentelli, A., "Probabilistic Hill Climbing Algorithms: Properties and Applications," Memorandum No. UCB/ERL M84/34, UC Berkeley, March 1984.

[24] Tarjan, R., "Depth–First Search and Linear Graph Algorithms," *SIAM Jour. Comput.*, Vol. 1, No. 2, 1972, pp. 146 - 160.

[25] Tompkins, J. A. and White, J. A., *Facilities Planning*, John Wiley & Sons, 1984.

[26] Wilhelm, M. R. and Ward, T. L., "Solving Quadratic Assignment Problems by Simulated Annealing," *IIE Transactions*, Vol. 19, No. 1, 1987, pp. 107–119.