

A neural network approach for a robot task sequencing problem

O. Maimon^a, D. Braha^{1,b,*}, V. Seth^c

^aDepartment of Industrial Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel

^bDepartment of Industrial Engineering, Ben-Gurion University, P.O. Box 653, Beer-Sheva 84105, Israel

^cDepartment of Manufacturing Engineering, Boston University, Boston, MA 02215 USA

Received 3 December 1999; accepted 10 March 2000

Abstract

This paper presents a neural network approach with successful implementation for the robot task-sequencing problem. The problem addresses the sequencing of tasks comprising loading and unloading of parts into and from the machines by a material-handling robot. The performance criterion is to minimize a weighted objective of the total robot travel time for a set of tasks and the tardiness of the tasks being sequenced. A three-phased parallel implementation of the neural network algorithm on Thinking Machine's CM-5 parallel computer is also presented which resulted in a dramatic increase in the speed of finding solutions. To evaluate the performance of the neural network approach, a branch-and-bound method and a heuristic procedure have been developed for the problem. The neural network method is shown to give good results and is especially useful for solving large problems on a parallel-computing platform. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: Robot task sequencing; Asymmetric traveling salesman Problem; Recurrent neural networks; Parallel and distributed computation; Automated storage and retrieval

1. Introduction

1.1. Background and justification

The problem under consideration is that of a robot task sequencing in a Flexible Manufacturing System (FMS). We consider an FMS with multiple machines, multiple storage buffers, and a single free-ranging robot transporting parts in-between machines and also between machines and buffers. Every few minutes the system is examined and a set of move requests (tasks) is generated. The set of tasks at each time instant is assumed to be known in advance and the method of generating these tasks is not treated here. Thus, the robot can be considered to have a set of tasks to carry out at each planning point. Each task is generated by the need to either load a part on a machine or unload a part from the machine. The tasks to be sequenced might also be required to meet some precedence constraints. For example, consider a situation where the task might be to unload a part from a particular machine and another task in the same set might be to load that machine with another part. Thus, we need to

ensure that the machine is unloaded first before it is loaded with a new part.

The problem is to sequence the tasks, with the objective of minimizing some performance measure such as the total travel time of the robot. Apart from the total travel time of the robot, other local performance measures can be considered for this problem. One such performance measure is the amount of time a task has been waiting for to be completed by the robot. Alternatively, the completion time of a task could also be modeled as a constraint to ensure that a task is completed before a certain time. An example of it is in wafer fabrication facilities, where time windows for the tasks to be carried out are a consideration. The local objectives of minimizing the total travel time of the robot and the total tardiness of the tasks support the global objective of maximizing system's throughput.

To illustrate the robot task-sequencing problem consider the material flow in an automated printed circuit board assembling environment, which employs the surface mount technology [1]. The surface mount technology involves mainly the process of placing a surface-mount component onto the board in the required position. Assembly of a surface-mount device involves positioning the component on the board, which has previously had the solder paste applied. Thus, the robot has to transport the boards in-between insertion machines and also between

* Corresponding author. Tel.: + 972-7-472-255; fax: + 972-7-472-958.

E-mail address: brahad@bgumail.bgu.ac.il (D. Braha).

¹ Also at the New England Complex Systems Institute, 24 Mt. Auburn St., Cambridge, MA 02135, USA.

insertion machines and buffers at the right time before the adhesive becomes dry (in which case the board has to be discarded).

The basic robot task-sequencing problem described here can also be ascribed to other manufacturing environments that include automated material handling devices (e.g. free-ranging automated guided vehicles) or Automated Storage and Retrieval Systems [2]. Another situation where the proposed model may be tried is in robotic assembly [3]. Consider a multi-robot assembly cell that has to perform some given tasks. Each task (e.g. “assemble parts”) can be broken down to a set of partially ordered operations (assembly tree). The partial ordering results from the parts’ process plans and the robotic cell configuration [4,5]. Each task can be performed in a number of alternative sequences of operations. Each operation is further broken down to detailed robot movements that include sensory interaction. This last process eventually results in robot programs for the task. The multi-robot assembly planning problem is to determine the optimal sequence of motions and operations for each robot such that all the given tasks are executed with a total minimum time [3]. As shown in Ref. [3], the basic robot task-sequencing problem considered here is one of the important modules in solving the multi-robot assembly planning problem.

1.2. The proposed approach

The robot task-sequencing problem has been modeled in some respects comparable to the Traveling Salesman Problem (henceforth referred to as the TSP). The problem is different from the typical TSP in the sense that the distances being considered are not symmetric due to the nature of the problem. In addition, the problem is modeled with some additional constraints as discussed below. Therefore, the problem is more akin to the Asymmetric Traveling Salesman Problem (henceforth referred to as the ATSP) with additional constraints. The ATSP is computationally difficult (i.e. it belongs to the class of NP-complete problems, see Ref. [6]). In particular, this means that that CPU time required to solve the ATSP problem, based on known algorithms, grows exponentially with the “size” of the problem. At this point of time, no polynomial time algorithms are capable of solving NP-complete problems, and it is unlikely that polynomial time algorithms will be developed for these problems. Thus, although the problem may be mathematically formulated, the solution complexity is intractable. For small problems (mostly less than 20 tasks) optimal seeking techniques, such as branch-and-bound type solutions can be used. Larger problems call for heuristic solutions. In this paper, the solution has been attempted using Hopfield Neural Networks [7]. Hopfield’s model has been proven successful in rapidly generating good solutions to NP-complete problems, and is readily amenable to parallel and distributed computation. Parallelism results in a

large reduction in the computation time, a property required in a real-time decision making environment.

To fully assess the benefits of the neural network scheme, we undertook full-scale experimental studies. The experimental studies were performed both on serial and parallel machines, as opposed to most current neural network research that employs the conventional serial machine. To evaluate the performance of the proposed neural network approach, a branch-and-bound method and a heuristic procedure have been developed for the problem under consideration,² and the solutions are compared with those obtained by the neural net method. The computational results show that the proposed neural network scheme yields efficient solutions within a small computation time, and is very effective when compared to the branch-and-bound and heuristic procedures.

1.3. Related literature

In this section, research on some relevant issues in material handling task-sequencing are briefly discussed with respect to several themes.

1.3.1. Operations research procedures

As mentioned in Section 1.2, the robot task sequencing is modeled as a combinatorial optimization problem comparable to the ATSP for which several operations research procedures were applied.

- *Branch-and-bound algorithms.* Carpeneto and Toth [8] developed branch-and-bound algorithm for the ATSP based on the subtour elimination approach or the Hungarian algorithm. They discussed a new selection procedure for the subtour to be split and the ordering of the arcs in the selected subtour. They also presented computational results for random problems with various ranges of coefficients of the distance matrix. Similar approach was presented by Syslo et al. [9]. It is based on the Hungarian algorithm and reduces the original distance matrix till an optimal solution is obtained. It is shown that the execution time is strongly dependent on the problem instance. It also grows with the size of the network. They also showed that it would be quite expensive to obtain an exact solution of the ATSP using the branch-and-bound algorithm for networks with nodes greater than 40. Miller and Pekny [10] have surveyed recent methods like branch-and-bound, Genetic Algorithms (GA) and Simulated Annealing for solving large TSP problems. A branch-and-bound algorithm is presented for the ATSP along with computational results. It is shown to perform well for some classes of problems. Balas and Christofides [11] have developed a branch-and-bound procedure based on sub-tour elimination, which is shown to give good results. The approach can be adapted to the

² For continuity, we present the heuristic and branch-and-bound algorithms in Appendix B.

symmetric TSP. Although branch-and-bound can be used to find the optimal solution of the problem, it tends to get computationally intractable for large and complex problems.

- *Cutting plane approach.* Ascheuer et al. [12] have presented a Cutting Plane approach to the Sequential Ordering Problem (SOP). This problem is similar to our problem and has applications in FMS and finds minimum cost paths subject to precedence constraints. They outline a Linear Programming framework for efficient solutions for the SOP and discuss polynomial time separation algorithms for obtaining the solution.
- *Task-sequencing with special structure.* In Ref. [2] the problem of order-picking in a rectangular warehouse that contains crossovers only at the ends of aisles is addressed by Ratliff and Rosenthal. It is shown to be a solvable case of the TSP. An algorithm is presented for picking an order in minimum time. The computational time required is linear in the number of aisles. In Ref. [13] a near-minimum task-planning algorithm for fruit harvesting robots having to pick fruits at N given locations is presented. The sequence of motions for the robot was obtained by solving the TSP using the geodesic distance along the robot manipulator's inertia space. The algorithm helps in selecting the most efficient robot design to perform a sequence of tasks at N locations.

1.3.2. Heuristics

For small task-sequencing problems the aforementioned techniques can be used. Larger problems call for heuristic solutions. The heuristic approaches that are often applied to material handling task-sequencing problems include the closest insertion algorithm [14], the nearest-neighbor rule [15] and the First-Come-First-Served dispatching rule [15–17]. The closest insertion algorithm repeatedly selects a task from those unassigned, which causes the smallest increase in the length of the sequence. By the nearest-neighbor rule, the robot travels to the nearest pickup point (e.g. a machine or a storage place) from its current position. The First-Come-First-Served dispatching rule serves the tasks in chronological order, and is easily adapted for quasi-real-time robot task-sequencing problems. The above heuristics were shown to give good results and are quite fast.

Another set of heuristics for the task-sequencing problem includes GA. GAs have been quite successfully applied to optimization problems like the ATSP. Jog et al. [18] present a survey on this topic. Grefenstette et al. [19], Storer et al. [20] and Tate et al. [21], also discuss about using GAs to solve the ATSP. Chen and Tseng [22] present a GA approach for solving the problem of a robot path planning.

1.3.3. Neural network approaches

As the neural network approach for the robot task-sequencing problem under consideration has a unique structure; we have not been able to trace any direct references in

the literature. However, a few papers on Hopfield neural network models³ for the TSP and ATSP were examined.

Hopfield and Tank [23,24] showed that highly interconnected networks of non-linear analog neurons are extremely effective in solving the TSP. Wilson and Pawley [25] tried a number of variations of the Hopfield and Tank algorithm. An improvement in the ability of the net to generate valid tours was obtained by modifying the initialization procedure. The initialization is based on the rationale that cities on opposite sides of the two-dimensional Euclidean square probably should be on opposite sides of the tour [26]. The starting activity of each unit is biased to reflect this fact. Cities far from the center of the square received a stronger bias than those near the middle. Szu [27] developed a modified Hopfield net for solving the TSP. In addition to improving the energy function, Szu uses continuous activities, but binary output signals. That is, the output function is the "hard limiter" or the unit step function, rather than the differentiable sigmoid function used by Hopfield and Tank.⁴ The architecture that Szu uses is the same as that for the Hopfield–Tank models, however, the activities of the units are updated simultaneously rather than sequentially. He performs n^2 such updates and then tests for a valid tour. The simultaneous update of the units results in faster solutions. A determination of the parameters used in the Hopfield–Tank formulation was carried out by Hegde, Sweet and Levy [28] and it was observed that these parameters need to be carefully selected and tuned in order to give good solutions. Relationships between parameters and the size of the problem were studied. It was shown that it becomes harder to solve the TSP involving larger number of cities. Aiyer et al. [29] analyzed the behavior of the Hopfield model both as a Content Addressable Memory (CAM) and as a method for solving the TSP. An analysis of why the network can frequently converge to an invalid solution when applied to the TSP was done. Analytic expressions were derived for the constant terms used in constructing the TSP energy function. With these expressions, the network can be made robust, and can reliably solve the TSP. They analyzed the eigenvalues of the connection matrix and gave a comprehensive description of the way in which a Hopfield network solves the TSP. They used a modified differential equation and employed a piecewise linear function (instead of using a hyperbolic tangent output function). Abe [30] examines how to suppress the *spurious stable states*⁵ of the Hopfield neural networks. Assuming that the output function of neurons is monotonic and differentiable at any interior point in the output range, the condition that a vertex of a hypercube becomes a local minimum

³ There has been progress in the application of the Kohonen self-organizing network to the TSP as well [37,38].

⁴ This step function is also called the McCulloch–Pitts input/output function.

⁵ States in which convergence is achieved without meeting the constraints.

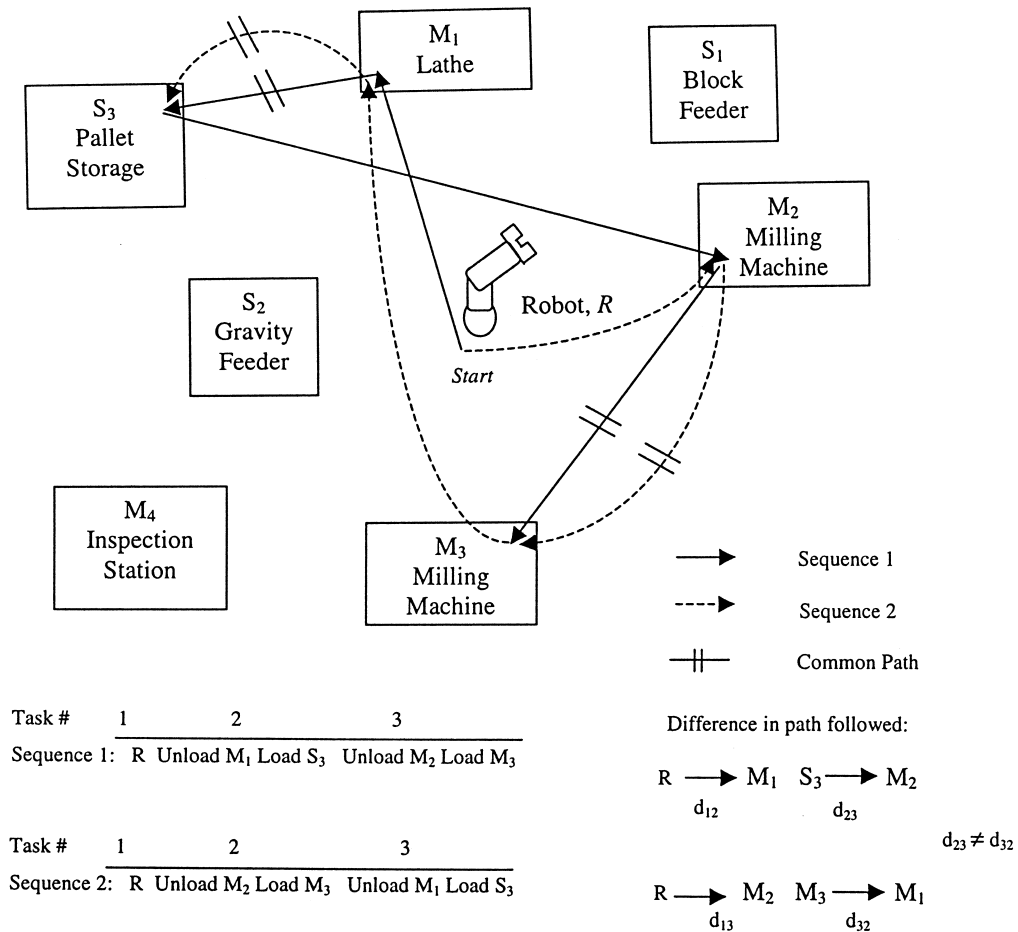


Fig. 1. Schematic diagram of the system.

of the Hopfield network and a monotonic convergence region to that minimum are clarified. Based on this, a method to analyze and suppress the spurious states in the networks was found, and was shown to be applicable for the original Hopfield energy function. In a recent work, Abe [31] derive stability conditions for local minima and their convergence regions for the Hopfield neural networks when the diagonal elements of the coefficient matrix are all non-zero. For the TSP, the ranges of weight values in the energy function and the range of values of the diagonal elements is clarified so that the feasible solutions become stable and the infeasible solutions become unstable. By gradually decreasing diagonal elements, quality of solutions is drastically improved compared with that of zero diagonal elements. As a result they show that non-zero diagonal elements can lead to better performance.

An efficient neural network algorithm for solving the Multiple Traveling Salesman Problem (MTSP) was developed by Wacholder et al. [32]. A new transformation of the MTSP to the standard TSP was introduced. An interesting concept was that the model associated with the problem used the Basic Differential Multiplier Method (BDMM), which evaluates Lagrange multipliers simultaneously with

problem's state variables, as a solution procedure. As a result a constrained optimization problem can be converted to an unconstrained optimization problem by introducing Lagrange multipliers. A gradient ascent on these multipliers forces the solution to meet the constraints and shows convergence to valid solutions for a wide variety of parameters. A study of how the contributions to the total energy are transferred between the various components while the total energy gradually decreases to its final local minima helped in formulation of the problem and improving it.

1.4. Paper's organization

In Section 2 the mathematical formulation of our robot task sequencing problem is presented. The solution methodology for the formulation using the Hopfield neural net is discussed in Section 3. Section 4 consists of presentation of results from the neural network solution methodology along with a comparison of its performance with respect to a branch-and-bound procedure and the closest insertion heuristic. Conclusions and future research directions are discussed in Section 5.

2. Robot task sequencing problem

As introduced in Section 1, the robot task-sequencing problem consists of sequencing a set of tasks with the objective of minimizing a weighted objective of the total robot travel time for a set of tasks and the tardiness of the tasks being sequenced keeping in mind precedence constraints.

Underlying the formulation are the following assumptions: (1) the robot can carry only one part at a time. (2) Multiple storage locations are considered. (3) The parts to be transported from the machines have been fully processed. In other words, the processing times are not considered in our model. (4) The robot at the start of the sequence is at some arbitrary position. This is because the robot may have performed a task previously, and is still at the position where it ended that task. A central or fixed starting position for the robot is not considered, in order to make the problem more general. (5) The travel time for the robot between any two points, in which either one of the points can be a machine or a storage place, is known. This is denoted by an asymmetric matrix D .

We now illustrate the sequencing problem by the simple example shown in Fig. 1. This will show that the matrix of travel times to be optimized that appear in the objective function will be asymmetric. A first dummy task is introduced in the sequence in which the robot does not perform any loading or unloading operation. This dummy task is used for convenience in formulation of the problem. Let us suppose that the first task is to unload machine M_1 and take the part to storage buffer S_3 , and the second task to be loading machine M_3 with the part from machine M_2 . These along with our first dummy task will comprise the set of tasks for this system. Let T be the set of tasks to be performed by the robot. Let the number of tasks to be performed in T be n . We denote each task except the dummy task by $T_{a,b}$ where ‘a’ refers to the origin point for the task and ‘b’ refers to the destination point. Let the dummy task be denoted by O . Now for this problem $n = 3$, where Task 1 $\stackrel{\text{def}}{=} O$; Task 2 $\stackrel{\text{def}}{=} T_{M_1,S_3}$; and Task 3 $\stackrel{\text{def}}{=} T_{M_2,M_3}$.

Now we can consider two sequences for tasks: (1) Task 2 is carried out before Task 3; and (2) Task 3 precedes Task 2. The paths corresponding to these two sequences are shown in Fig. 1. Obviously the travel time for the robot for the two paths will be different. However, we can see that there is a common travel segment for both sequences. This is the portion of each path where the part is being carried by the robot from one location to another. This part of the robot travel is necessary and cannot be minimized. Hence, the travel to be minimized is the travel of the robot in between the tasks, i.e. the travel from the end of one task to the start of the next one. We represent this part of the robot travel by $d_{k,l}$, i.e. $d_{k,l} \cup$ travel time from the location where task k ends to the location where task l begins. As the above example shows, in general $d_{k,l} \neq d_{l,k}$. Thus the $n \times n$ matrix $[d_{k,l}]$ is generally asymmetric. Task 1 is the dummy task and is

needed to represent the travel from the starting position of the robot, R , to the beginning of Task 2.

To formulate the robot task-sequencing problem, we define the following binary decision variables $x_{i,j}$:

$$x_{i,j} = \begin{cases} 1 & \text{if task } i \text{ occupies position } j \text{ in the sequence} \\ 0 & \text{otherwise} \end{cases}$$

Thus, a solution to the problem is specified once $x_{i,j}$ ’s is determined. For n tasks, it will be an instance of a $n \times n$ permutation matrix, with the rows representing tasks and columns representing their positions in the sequence. Hence, we have to ensure that each row and column contains only a single ‘1’ and the other entries are ‘0’ (so that each task is performed once and only once).

Let us again consider the system in Fig. 1. We suppose that machine M_3 had a part that had to be transported to buffer S_3 as one of the robot’s tasks. In this case, we have to ensure that machine M_3 is unloaded before it can be loaded with the part from machine M_2 . This brings about a precedence relationship among tasks. Let π_i denote the precedence task for task i , with $\pi_i = 0$ if task i has no precedence.

We now consider additional constraints concerning due dates. Let D_i be the due time for task $i \ i = 1, 2, \dots, n$. We include a due time constraint for task i , namely that it should be completed by time D_i , if possible. We consider this as a *soft* constraint; i.e. solutions that do not meet this constraint are considered.

The problem formulation is given below:

$$\text{minimize } \sum_{i=1}^n \sum_{\substack{k=1 \\ k \neq i}}^n \sum_{j=1}^{n-1} d_{i,k} x_{i,j} x_{k,j+1}, \tag{1}$$

$$\text{subject to } x_{1,1} = 1, \tag{2}$$

$$\sum_{i=1}^n x_{i,j} = 1 \quad j = 1, \dots, n, \tag{3}$$

$$\sum_{j=1}^n x_{i,j} = 1 \quad i = 1, \dots, n, \tag{4}$$

$$\sum_{k=1}^{j-1} x_{\pi_i,k} \geq x_{i,j} \quad \forall \pi_i \neq 0, \quad i = 2, \dots, n; \quad j = 2, \dots, n, \tag{5}$$

$$D_i \geq \sum_{j=2}^n \left[x_{i,j} \left(\sum_{l=2}^j \sum_{m=1}^n \sum_{\substack{q=1 \\ q \neq m}}^n d_{m,q} x_{m,l-1} x_{q,l} \right) \right] \quad i = 1, \dots, n, \tag{6}$$

where the objective function represents the total travel time to be minimized. This is different from the TSP objective function because a closed tour is not desired; in other words

it is assumed that the robot stays at the position it is in after completing the last task in the sequence.

Constraint (2) ensures that the “dummy” task is always the first task in the sequence. Constraints (3) and (4) ensure that each task is performed exactly once (as in the TSP). Constraint (5) ensures precedence requirements. Constraint (6) requires that the tasks be completed by their due times. Constraints (5) and (6) are uniquely presented for the robot task-sequencing problem. In this paper, we consider constraint (6) a “soft” constraint. Thus, we can model it as an objective so that the robot task-sequencing problem now becomes a multiple-objective function problem with the second objective as:

$$\text{minimize } \sum_{i=2}^n \left(\max \left\{ 0, \sum_{j=2}^n \left[x_{i,j} \left(\sum_{l=2}^j \sum_{m=1}^n \sum_{\substack{q=1 \\ q \neq m}}^n d_{m,q} x_{m,l-1} x_{q,l} \right) \right] - D_i \right\} \right), \quad (7)$$

where we minimize the total tardiness of the tasks. The weight to be given to this objective compared to the minimization of travel time can be adjusted in the neural network approach to obtain desired results (see Section 4).

3. Neural network solution methodology

3.1. The hopfield-tank model

We now discuss the Hopfield neural network methodology for the formulated problem. An optimization problem can be mapped on to a Hopfield neural network by constructing appropriate recurrent neural network and energy function for the network and interpreting the network outputs as the solution of the optimization problem. The energy function is constructed such that network’s state that achieves the lowest energy value corresponds to the optimal (or near optimal) solution of the optimization problem. This network’s state is often referred as the most stable state of the network. The Hopfield network is a continuous deterministic model, while the robot task-sequencing problem requires a discrete solution. This means embedding a discrete problem in a continuous decision space where the neural computation operates. In this research, we borrow the Hopfield–Tank model [23] concept to show how a continuous decision space is associated with our problem, and how to solve our problem through highly interconnected neural nets.

It is known that the equations of motion for a network with symmetric connections always lead to the convergence of stable states [33,34]. However, our formulation of the robot task-sequencing problem involves an asymmetric time matrix that will result in asymmetric connections in the network representing it. Based on experimental results, Hopfield states that complicated dynamics can occur for asymmetric connections in the network, which can lead to the possibility that a minimum state will not be stable and

will be replaced in time by another minimum [34]. However, it was shown by simulation experiments that even though the probability of errors increases the algorithm continues to generate stable minima in case of asymmetric connections. Thus, while the symmetry of the network is essential to the mathematics, a pragmatic view indicates that even asymmetric networks might work. This has been confirmed by the computational study in Section 4.

For the robot task-sequencing problem with n tasks, a $n \times n$ task-position matrix and a total of n^2 neurons are needed. Thus, neuron (i, j) will be represented by U_{ij} , where i denotes a task and j its position in a tour. The state of neuron U_{ij} will be represented by $u_{ij} \in R$. The output of the neuron U_{ij} is obtained from its state through an activation function:

$$v_{ij} = g(u_{ij}),$$

where the form of $g(\cdot)$ shall be discussed later. The value of $v_{ij} \in [0, 1]$ expresses the tendency of task i to be located in position j in a sequence. A final solution to the problem is obtained whenever the neuron outputs converge to a stable steady state in time of the form:

$$\lim_{t \rightarrow \infty} v_{ij}(t) = \begin{cases} 1 & \text{task } i \text{ is in position } j \\ 0 & \text{otherwise} \end{cases} \quad \forall i, j = 1, 2, \dots, n.$$

3.2. Energy function formulation

As shown in Refs. [33,34], Hopfield nets work on an energy minimization approach. Thus, the first task at hand is to describe the energy function whose minimization corresponds to solving the robot task-sequencing problem. This function consists of three parts: a term for the total robot travel time, a term for the tardiness of tasks, and a term for the precedence constraints. All energy terms are chosen so as to ensure that the states corresponding to the desired permutation matrix are favored.

The energy function which describes the total travel time of the sequence is denoted by E_t and is given as follows:

$$E_t = \frac{1}{2} \sum_{i=1}^n \sum_{\substack{k=1 \\ k \neq i}}^n \sum_{j=1}^{n-1} d_{i,k} v_{i,j} v_{k,j+1}. \quad (8)$$

The energy function which represents the tardiness of the tasks is called E_d and is given by:

$$E_d = \sum_{i=2}^n \left(\max \left\{ 0, \sum_{j=2}^n \left[v_{i,j} \left(\sum_{l=2}^j \sum_{m=1}^n \sum_{\substack{q=1 \\ q \neq m}}^n d_{m,q} v_{m,l-1} v_{q,l} \right) \right] - D_i \right\} \right). \quad (9)$$

The constraint, $x_{11} = 1$, namely that the dummy task should always be the first task in any sequence, leads to the fact that the neuron in position $(1,1)$, i.e. U_{11} will always “fire” or be “on”. Thus the output of neuron U_{11} will always be 1, which

is represented by the condition:

$$v_{11} = 1. \tag{10}$$

Constraints (3) and (4) which ensure that each task is performed exactly once, can be represented by the following energy functions:

$$E_1 = \frac{1}{2} \sum_{i=1}^n \left(\sum_{j=1}^n v_{ij} - 1 \right)^2, \tag{11}$$

$$E_2 = \frac{1}{2} \sum_{j=1}^n \left(\sum_{i=1}^n v_{ij} - 1 \right)^2. \tag{12}$$

E_1 is minimized (would attain a zero value) only when exactly one neuron fires in each row, and E_2 is minimized (would attain a zero value) only when exactly one neuron fires in each column.

For the precedence relationships (constraint 5), we have developed the following energy function:

$$E_3 = \sum_{i=1}^n \sum_{k=1}^{n-1} \left(v_{ik} \sum_{j=k+1}^n v_{\pi_i, j} \right). \tag{13}$$

Minimizing E_3 to zero ensures that the predecessor of task i (π_i) is carried out before this task.

We also considered an alternative formulation for the second and third constraints by using the following energy functions:

$$E'_1 = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \sum_{\substack{k=1 \\ k \neq j}}^n v_{ij} v_{ik},$$

$$E'_2 = \frac{1}{2} \sum_{j=1}^n \sum_{i=1}^n \sum_{\substack{k=1 \\ k \neq i}}^n v_{ij} v_{kj},$$

$$E'_3 = \frac{1}{2} \left(\sum_{i=1}^n \sum_{j=1}^n v_{ij} - n \right)^2.$$

E'_1 is minimized (would attain a zero value) only when one neuron fires in each row, and E'_2 is minimized (would attain a zero value) only when one neuron fires in each column. However, these two together cannot prevent a situation where none of the neurons fires, or ensure that n neurons will necessarily fire. Minimizing E'_3 ensures that any feasible solution to the problem contains n entries of value 1 in the entire permutation matrix (when $E'_3 = 0$). This formulation has been observed to work less well for these constraints.

In the next two sections, we consider two solution schemes for the problem, which improve the original procedure of the Hopfield–Tank model: (1) the Basic Differential Multiplier Method approach [32,35]; and (2) the Penalty function approach [36].

3.3. Basic Differential Multiplier Method approach

A constrained optimization problem can be converted into an unconstrained one by introducing Lagrange multipliers, λ_α , and minimizing the resulting energy function. We can represent our problem as:

$$\text{minimise}_{v \in \mathcal{V}} E = E_t + E_d + \sum_{\alpha=1}^3 \lambda_\alpha E_\alpha \tag{14}$$

with condition (10), where E_t , E_d and E_α ($\alpha = 1, 2, 3$) are defined in Eqs. (8)–(13). Based on the above energy function, we obtain a system of ordinary differential equations, each of which describes the time evolution of a neuron (each neuron has the same circuit motion function):

$$\frac{du_{ij}}{dt} = \left(-\frac{u_{ij}}{\tau} - \frac{\delta E_t}{\delta v_{ij}} - \frac{\delta E_d}{\delta v_{ij}} - \sum_{\alpha=1}^3 \lambda_\alpha \frac{\delta E_\alpha}{\delta v_{ij}} \right) \forall i, j = 1, 2, \dots, n, \tag{15}$$

$$\frac{d\lambda_\alpha}{dt} = +E_\alpha \quad \forall \alpha = 1, 2, 3; \tag{16}$$

with condition (10). Notice that the damping term, $-u_{ij}/\tau$ (τ is a constant), suggested by Hopfield and Tank [23] is used. Also, to ensure convergence to a stable equilibrium point on the state space boundaries (i.e. $v_{ij} \in \{0, 1\}$, $\forall i, j = 1, \dots, n$), the plus sign is chosen in the right-side of Eq. (16) as suggested by Platt and Barr [35]. This method stabilizes the BDMM and results in increasing penalty terms (i.e. λ_α 's) which forces the solution to fulfill the constraints.

The solution of the dynamic model was carried out using the first-order Euler numerical method:

$$u_{ij}^{\text{new}} = u_{ij}^{\text{old}} + \Delta t \left[\frac{du_{ij}}{dt} \right]^{\text{old}}, \tag{17}$$

$$\lambda_\alpha^{\text{new}} = \lambda_\alpha^{\text{old}} + \Delta t E_\alpha^{\text{old}}. \tag{18}$$

The “new” and “old” designate current and previous time step values, respectively. The initial values of the Lagrange multipliers, $\lambda_\alpha(0)$, can be chosen at random but should be positive as the selection of negative values would cause the equations longer to converge to a valid solution. This is because $\lambda_\alpha(0)$'s affect the speed with which the solution is obtained. An analysis of the various energy terms helps in deciding the values to be assigned to $\lambda_\alpha(0)$'s (see Section 4.1).

3.4. Penalty function approach

In this case the constrained optimization problem is reformulated as an unconstrained one by incorporating the constraints into the objective function as a penalty term. The penalty term $P(v)$ is formulated as follows:

$$P(v) = E_1 + E_2 + E_3 \tag{19}$$

and the resulting problem is now

$$\text{minimize}_{v \in \mathcal{V}} E = E_t + E_d + \lambda P(v), \quad (20)$$

where λ is the penalty parameter. Based on the above energy function, we again obtain a system of ordinary differential equations:

$$\frac{du_{ij}}{dt} = \left(-\frac{u_{ij}}{\tau} - \frac{\delta E_t}{\delta v_{ij}} - \frac{\delta E_d}{\delta v_{ij}} - \lambda \frac{\delta P(v)}{\delta v_{ij}} \right) \quad \forall i, j = 1, 2, \dots, n, \quad (21)$$

$$c_p \frac{d\lambda}{dt} = [P(v)]^{-1}, \quad (22)$$

where c_p (≈ 1) is a positive capacitance parameter and τ is typically 1. The system described by the above equations is again discretized based on the first-order Euler formula. The BDMM approach utilizes Lagrange multipliers for each constraint to force a feasible solution. The Penalty function approach uses a single penalty parameter for all the constraints and increments its value through the inverse function $[P(v)]^{-1}$ in Eq. (22). This approach was observed to give better results for the robot task-sequencing problem than the BDMM approach in terms of the quality of solutions obtained (see Section 4.2).

For performing the updates on the neural network units as given in Eqs. (15) and (21), we need to evaluate the partial derivatives of the energy terms. They are given in Appendix A.

3.5. Serial neural network algorithm

The stopping condition in Step 7 is given by either a fixed number of iterations or when all the constraints are satisfied. The algorithm steps are:

1. Initialize activation of all units (i.e. initialize u_{ij}). Initialize Δt to a small value.
2. While the stopping condition is false, do Steps 3–7.
3. Perform Steps 4–6 n^2 times.
4. Choose a unit at random.
5. Update the activity on the selected unit:

$$u_{ij}(\text{new}) = u_{ij}(\text{old}) + \Delta t \left[\frac{du_{ij}}{dt} \right].$$

6. Apply output function: $v_{ij} = g(u_{ij})$
7. Check stopping condition.

The choice of the initialization in Step 1 and the activation function of Step 6 was carried out after running trial experiments to determine which combination gives the best results. The following piece-wise linear activation function

[29] and initialization of $u_{ij}(0)$ [30,36] gave the best results:

$$v_{ij} = \begin{cases} 0 & \text{if } u_{ij} \leq -0.5 \\ u_{ij} + 0.5 & \text{if } -0.5 \leq u_{ij} \leq +0.5, \\ 1 & \text{if } u_{ij} \geq +0.5 \end{cases}$$

where $u_{ij}(0)$ is a uniformly distributed random variable in $[0,1]$.

3.6. Parallel neural network algorithm

It was observed that the algorithm was computationally intensive and hence the use of an algorithm which could be implemented in a parallel fashion on Boston University's *CM-5 Connection Machine* was explored. The CM-5 consists of 64 parallel processors partitioned into three separate groups called *partition managers*, consisting of 16, 16 and 32 processors. For our simulations we used the *Illiac* partition manager consisting of 16 processors. A parallel algorithm was developed in three phases.

3.6.1. Phase I

In an attempt to decrease the computation time, the serial code was converted into a parallel code in C^* using parallel variables for arrays. This would make *array-based* operations easier to perform.

3.6.2. Phase II

Each of CM-5's processors can be made to generate a solution separately. This can be carried out by running a copy of the simulation code on each of the processors or nodes. Thus all the processors work concurrently to give multiple solutions.

3.6.3. Phase III

In the serial algorithm presented in Section 3.5, the units are chosen at random to update. Thus during each iteration, each unit has, on an average, one opportunity to update its activity level. Also, the updating of the activities of the units is performed sequentially. It may be desirable to ensure that each unit does update its activity in every iteration. Also, the units could update their output simultaneously rather than sequentially. Each of the Steps 1, 3, and 4 below is performed simultaneously for all units. This is carried out by efficiently paralleling the update function instead of paralleling independent statements in the code, which utilizes the multiple vector units in the processors more efficiently. The algorithm in this case is as follows:

1. Initialize activation of all units. Initialize Δt to a small value.
2. While the stopping condition is false, do Steps 3–5.

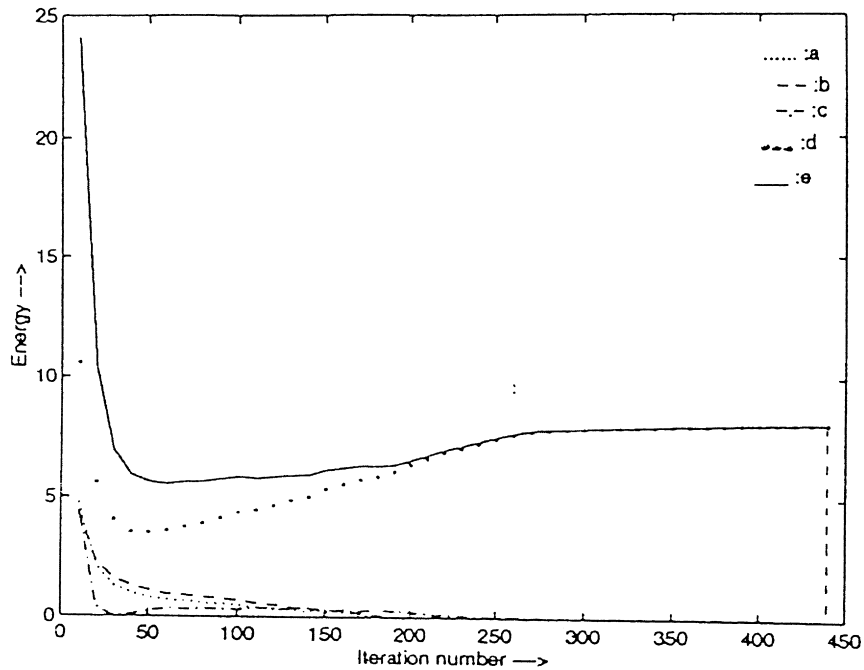


Fig. 2. Energy variation versus number of iterations for a 25-task problem when $\lambda_1 = 50, \lambda_2 = 50, \lambda_3 = 50, a = E_1, b = E_2, c = E_3, d = E_4, e = E_5$.

3. Update activity on every unit:

$$u_{ij}(\text{new}) = u_{ij}(\text{old}) + \Delta t \left[\frac{du_{ij}}{dt} \right].$$

4. Apply output function on all units: $v_{ij} = g(u_{ij})$.

5. Check stopping condition.

4. Performance of various algorithms

In this section, we present the performance results and comparative performance of the various neural network algorithms discussed in Section 3. We also demonstrate their superiority over a heuristic (the ‘closest insertion’ heuristic) and a branch-and-bound algorithm that were developed for the robot task-sequencing problem. For

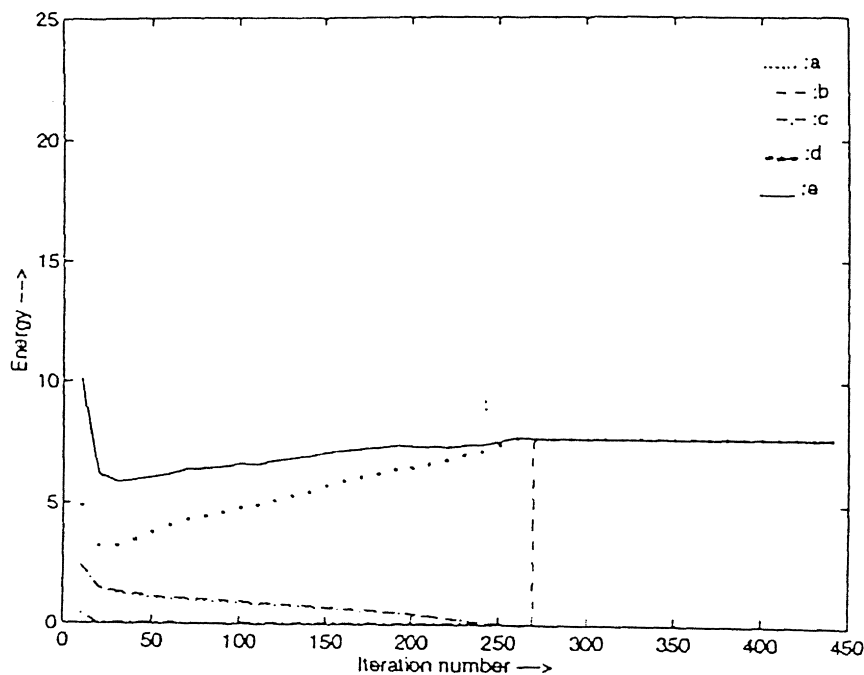


Fig. 3. Energy variation versus number of iterations for a 25-task problem when $\lambda_1 = 50, \lambda_2 = 50, \lambda_3 = 50, a = E_1, b = E_2, c = E_3, d = E_4, e = E_5$.

Table 1
Comparison of results with weighted objectives

A	B	Average path length	Average tardiness	Maximum path length	% Solutions with minimum path length
1	1	230.88	0.408	195	50.0
5	1	202.54	0.530	195	73.3
1	5	267.84	0.309	195	27.0

clarity these algorithms are presented in Appendix B. We conducted the experiments to address the following questions:

1. What are the optimal values of the parameters involved in the neural network procedures?
2. How does the BDMM compare with the Penalty function procedure?
3. How does the serial neural network algorithm compare with the parallel implementation?
4. How does the neural network approach compare with the closest insertion and branch-and-bound procedures?

4.1. What are the optimal values of the parameters involved in the neural network procedures?

A very significant step in the implementation of the artificial neural networks is the selection of the initial parameters. These values are important for the speed of convergence of the network to a valid tour. For example, the penalty parameters λ_α ($\alpha = 1, 2, 3$) in the BDMM approach (see Section 3.3) have to be given initial values $\lambda_\alpha(0)$. There is no optimal method to select these parameters. The decision process can be aided by studying how the contributions to the total energy are transferred between the various components while the total energy, E , decreases to its minimum value. In general, we found that it is important to choose “good” initial values for the parameters by running several simulations and adjusting them. Such energy function analysis also gives information on the constraints, which are more difficult to satisfy or are responsible for the delay in convergence and hence deserve more attention. This is demonstrated for a 25-task problem using the BDMM approach discussed in Section 3.3 for the following energy function:

$$\text{minimize } E = E_t + \lambda_1 E_1 + \lambda_2 E_2 + \lambda_3 E_3,$$

Table 2
Simulation results from the serial neural network

Problem size (# of tasks)	10	15	20	25	50
% Solutions with minimum path length (over the randomly generated instances, where each instance undergoes 1008 simulations)	50.2	48.4	35.0	20.5	11.2

where λ_α ($\alpha = 1, 2, 3$) have to be given initial values $\lambda_\alpha(0)$. Fig. 2 shows the plots for various energy components with a set of equal $\lambda_\alpha(0)$'s. E_3 takes the maximum number of iterations (450) to fall to a value of zero while the other energy functions, namely E_1 and E_2 reach a zero value much quicker (200 iterations). In order to force E_3 decrease faster, we increase the initial value of $\lambda_3(t)$, i.e. $\lambda_3(0)$. The subsequent plot is shown in Fig. 3 and the convergence is observed to take place much faster, i.e. in fewer iterations, so that all the E_α s go down to zero almost simultaneously. Thus it is important to choose “good” values for the initial values of parameters by running several simulations and adjusting them. Such energy function analysis also gives information on the constraints that are more difficult to satisfy or are responsible for the delay in convergence and hence deserve more attention.

Recall that our formulation consists of two objectives: the total travel time objective function and the tardiness objective function. It may be desirable to give more weight to the former as compared to the latter. This can be easily accomplished with the energy function formulation. We can multiply the corresponding energy terms with constants corresponding to their relative weights. In the simulations, we found that the neural network approach was very effective in obtaining results with the weighted objectives. To illustrate, consider the following problem formulation,

$$\text{minimize}_{v \in V} E = AE_t + BE_d + \lambda P(v),$$

where E_t , E_d and $P(v)$ are the terms from the penalty function approach (see Section 3.4) and A and B are constants. The following results were observed from 1000 solutions for a randomly generated 25-task problem:

We observe that different objectives can be emphasized by giving a higher weight to their corresponding energy terms (Table 1).

4.2. How does the BDMM compare with the Penalty function procedure?

We applied both the BDMM and Penalty function algorithms to different problem sizes. The number of tasks (n) the robot has to sequence gives the size of a problem (between 10 and 50 in our simulations). The number of tasks is often correlated with the number of machines in the FMS. For each problem size, multiple problem instances using randomly generated distances were used.⁶ The results for each randomly generated problem instance were obtained after running 1008 independent simulations in an attempt to generate many solutions. Each simulation was run with a maximum of 1000 iterations. The stopping condition for both algorithms was chosen to be either a fixed number of 1000 iterations, or when a feasible solution is obtained. The value of the time step Δt for the first order

⁶ In order to examine the sensitivity of the task sequencing methods to the system layout, the input distances were generated from system layouts derived from industrial scenarios.

Table 3
Number of simulations required for at least one minimum solution, with 99% confidence

Problem size (# of tasks)	The probability of obtaining an optimal solution in a single simulation run (p)	No. of simulations required for at least one minimum solution (L)
10	0.502	7
15	0.484	7
20	0.350	11
25	0.205	21
50	0.112	39

Euler formula was typically between 0.01 and 0.1, and the initial value of the penalty parameter $\lambda(0)$ was kept at 1. The following observations are made:

1. Both the BDMM and the Penalty function approaches gave almost 100% feasible solutions.
2. The Penalty function approach gave better results than the BDMM approach in terms of the quality of solutions obtained (i.e. % deviation from optimality).
3. The Penalty function approach gave more feasible solutions when each unit gets updated instead of updating the units randomly.

Table 2 gives the summary of results obtained by the neural network approach using the Penalty function method. The optimal path lengths for the randomly generated problem instances were found using a branch-and-bound method (see Appendix B) whose performance is discussed in detail later. We observe the following:

1. A solution was obtained within a small computation time, typically within 50–200 iterations.
2. It can be observed from Table 2 that an optimum solution is always obtained by our neural network approach. However, since our network involves asymmetric connections, the percentage of solutions (over the randomly generated instances, where each instance undergoes 1008 simulations) that are optimal (i.e. with minimum path length) decreases with increasing problem size. The average quality of solutions obtained also decreases with increasing problem size.
3. In order to analyze how the asymmetric time matrix in the task-sequencing problem affects the quality of the

Table 4
Comparison of the serial neural network algorithm with the CM-5 phase III parallelization with respect to the average CPU time in minutes for 1 simulation (100 iterations)

Problem size (# of tasks)	Serial machine	CM-5 Phase III parallelization
10	2.0	0.0050
15	115.61	0.0083
20	894.18	0.0137
25	>2000	0.0952
50	>2000	0.5473

final solutions, let us denote by p the probability of obtaining an optimal solution in a single simulation run with a maximum of 1000 iterations. We estimate p as the percentage of optimal solutions obtained through our randomly generated instances, where each instance undergoes 1008 simulations (see second column of Table 2). We can model the probability distribution for the number of minimum solutions obtained for a fixed number of simulations by a Binomial distribution. Thus, the probability for obtaining at least one optimal solution in L simulations is given by $1 - (1 - p)^L$, and the number of simulations required to obtain at least one optimal solution with a certain confidence level (say r) is given by $L = \lceil \log(1 - r) / \log(1 - p) \rceil$ where $\lceil x \rceil$ is the smallest integer greater than or equal to x . The third column (after being rounded off to the next higher integer) in Table 3 shows the number of simulations required to obtain at least one optimal solution with a 99% confidence level (i.e. $r = 0.99$). We observe that although the probability of obtaining an optimal solution in a single simulation run decreases with increasing the problem size, the number of simulations required to obtain at least one optimal solution is very reasonable.

4.3. How does the serial neural network algorithm compare with the parallel implementation?

Simulations have been conducted on the *Connection Machine* to explore the three phase parallel implementation introduced in Section 3.6. The following conclusions are derived:

1. The computation times of the Phase I parallelization that uses C^* to write the code are even longer than those for the serial machine (*Sun-Sparc*). This is because of the CM-5's overhead, which is large especially for small problems. Hence, the time saving benefits can be achieved only for very large size problems.
2. Phase II that generates multiple solutions simultaneously gave a big reduction in the run times as compared to the Phase I parallelization. As the simulations were performed on the *Illiac* partition manager, which has 16 processors, the increase in speed from Phase I to Phase II is observed to be almost exactly 16 times (i.e. a proportionate increase in speed).
3. Phase III consists of updating the activation of all neurons simultaneously. The parallel nature of the neural networks is exploited here and the activations of the neurons are updated in a parallel fashion. The results are shown in Table 4 and it can be observed that the run times for Phase III are almost negligible as compared to the serial machine.

4.4. How does the neural network approach compare with the closest insertion procedure?

The 'closest insertion' procedure (see Appendix B) is one

Table 5

Quality of solutions obtained for different problem sizes with the closest insertion algorithm

Problem size (# of tasks)	10	15	20	25
% deviation from optimal = $100 \times (x - y)/y$	13.4	107.1	114.5	115.5

of the most prevalent and intuitive heuristics for minimizing the total robot travel time [14], and is utilized as a benchmark to examine other heuristics. We conclude the following:

1. The closest insertion algorithm gave feasible and fast solutions to any size problem. However, the quality of solutions obtained for different size problems in terms of the travel time of the robot were much worse than those obtained by the neural network method as illustrated in Table 5.
2. The quality of solutions obtained by the closest insertion algorithm dropped further with increase in the number of precedence constraints in the problem, while the quality of solutions with the neural network method was not significantly affected by these constraints.
3. The percentage of solutions generated by the neural network algorithm were almost always (90–100%) better than the ones generated by the closest insertion algorithm.

4.5. How does the neural network approach compare with a branch-and-bound procedure?

A branch-and-bound method was used to generate optimal solutions and also provide a basis for comparison of the different algorithms. Simulations were run on a serial (Sun-Sparc) machine. The following observations were made:

1. The behavior of the branch-and-bound algorithm was found to be highly dependent on the problem size. The number of sub-problems as well as the running times was observed to increase immensely with the increase in the problem size.
2. For each problem size, the running time of the branch-and-bound algorithm increased rapidly with the increase in the number of precedence constraints, while multiple precedence constraints were easily handled by the neural network algorithm and did not affect the run time significantly (see Table 6).

Table 6

Running times of the branch-and-bound algorithm for different problem sizes and multiple precedence constraints

Problem size (# of tasks)	Approx. running time in minutes		
	One precedence constraint	Two precedence constraints	Three precedence constraints
10	<1	<1	<1–1.5
15	<1–14.6	<1–24.8	3.6–46.7
20	1.2–36.7	1.3–83.2	2.5–96.3
25	1.5–66.2	3.9–151.2	6.9–223.2
50	5.1–359.2	15.4–667.5	37.8–>900

Table 7

Comparison of the branch-and-bound algorithm with the CM-5 phase III parallelization in respect to CPU time in minutes (with three precedence constraints)

Problem size (# of tasks)	Branch-and-bound approach	CM-5 Phase III parallelization (99% confidence level)
10	<1–1.5	0.0350
15	3.6–46.7	0.0581
20	2.5–96.3	0.1507
25	6.9–223.2	1.9992
50	37.8–900	21.3447

3. The CPU times of the branch-and-bound and parallel neural network (Phase III parallelization) approaches are compared in Table 7. The run times for the neural network approach have been calculated by multiplying the number of simulations required to obtain an optimal solution with 99% confidence (data extracted from the third column of Table 3) with the average time required for 1 simulation using Phase III parallelization (data extracted from the third column of Table 4). For example, for a 25-task problem the run time for the neural network approach is given by $0.0952 \times 21 = 1.9992$. We observe that the run times for Phase III are almost negligible as compared to the branch-and-bound algorithm. In addition, the increase in running time of the branch-and-bound procedure with precedence constraints could be substantial, especially with bigger problems.

5. Summary

In this paper, we have presented a new methodology for the quasi-real-time robot task-sequencing problem. The problem encountered needs to be solved repeatedly in a real time environment, and its solution complexity is intractable (NP-complete). The proposed neural network approach has been proven successful in coping with both these impediments as summarized in the following: (1) the computational results show that the neural network schemes yield efficient solutions within a small computation time, and is effective when compared to branch-and-bound and heuristic procedures; thus, the work presented here supports the conclusion that the neural network model offers a worthy

method for large scale FMS. (2) Hopfield neural network approach was successfully implemented for asymmetric networks. The approach was shown to be feasible and practically useful for such networks. (3) Parallel nature of the neural network was exploited to implement simulations on a parallel computing platform (the Connection Machine CM-5), which was shown to be very useful especially for large problems. This results in a large reduction in computation time, a property required in a real-time decision making environment.

A theoretical investigation into the properties of asymmetric neural networks is very desirable. This paper demonstrated the feasibility of using such networks for practical applications such as the robot task-sequencing problem. However, developing alternative neural network approaches and an understanding of the dynamics of such networks is very essential to improve the solution techniques.

Appendix A. Evaluating partial derivatives of the energy terms

Let $[]_{i \neq k}$ ($[]_{i \neq k}$) denote that the term inside the square brackets is present only for $i \neq k$ (only for $i \neq k$).

$$\frac{\delta E_t}{\delta v_{ij}} = \left[\sum_{\substack{k=1 \\ k \neq i}}^n d_{ik} v_{k,j+1} \right]_{j \neq n} + \left[\sum_{\substack{k=1 \\ k \neq i}}^n d_{ki} v_{k,j-1} \right]_{j \neq 1}. \quad (A1)$$

In order to evaluate the partial derivatives ($\delta E_d / \delta v_{ij}$), we first define

$$S_i \equiv \sum_{j=2}^n \left[v_{i,j} \left(\sum_{l=2}^j \sum_{m=1}^n \sum_{\substack{q=1 \\ q \neq m}}^n d_{m,q} v_{m,l-1} v_{q,l} \right) \right] - D_i,$$

and let $\text{sgn}(S_i) = 0$ if $S_i \leq 0$ and $\text{sgn}(S_i) = 1$ otherwise. Then,

$$\begin{aligned} \frac{\delta E_d}{\delta v_{ij}} = & \text{sgn}(S_i) \left(\sum_{l=2}^j \sum_{m=1}^n \sum_{\substack{q=1 \\ q \neq m}}^n d_{m,q} v_{m,l-1} v_{q,l} \right) \\ & + \sum_{z=2}^n \left\{ \text{sgn}(S_z) \left[\sum_{k=j+1}^n v_{zk} \left(\sum_{\substack{q=1 \\ q \neq i}}^n d_{i,q} v_{q,j+1} \right) \right. \right. \\ & \left. \left. + \sum_{\substack{k=j \\ m=1 \\ m \neq i}}^n v_{zk} \left(\sum_{m=1}^n d_{m,i} v_{m,j-1} \right) \right] \right\}, \quad (A2) \end{aligned}$$

$$\frac{\delta E_1}{\delta v_{ij}} = \sum_{k=1}^n v_{ik} - 1, \quad (A3)$$

$$\frac{\delta E_2}{\delta v_{ij}} = \sum_{k=1}^n v_{kj} - 1, \quad (A4)$$

$$\frac{\delta E_3}{\delta v_{ij}} = \left[\sum_{k=j+1}^n v_{\pi k} \right]_{l=i} + \left[\sum_{k=1}^{j-1} v_{lk} \right]_{i=\pi}. \quad (A5)$$

Appendix B. A heuristic and a branch and bound algorithm for the robot task-sequencing problem

B.1. Branch-and-bound approach

Branch-and-Bound is a technique for solving combinatorial optimization problems using exhaustive but intelligent enumeration. It is based on a tree search where each node can be *branched* into several nodes at each step. The branching is based on a heuristic method that helps reduce the effort in searching for an optimal solution. After branching, a *lower bound* on the cost of each new node is calculated, and the search is continued on the node with the smallest lower bound. The procedure continues until a *Hamiltonian cycle* (described later) is obtained. The search is continued only on those nodes that have a lower bound smaller than the best solution obtained so far. The search ends when all the nodes are processed.

B.1.1. Reduction

The Branch-and-Bound method used here is based on the one outlined by Syslo et al. [9] for the TSP. Recall the asymmetric travel time matrix for our problem where each element d_{ij} represents the time from task i to task j . Let n be the total number of tasks to be sequenced. A Hamiltonian cycle (cycle of length n) will contain exactly one element from each row and column of this matrix. Unlike a TSP, for our problem it is not desirable for the sequence to be a closed tour (i.e. the robot returning to its initial position). Therefore the matrix is modified such that the *cost* or the time to go from any task to the first task is 0, i.e. the first column of the matrix contains all zeroes. This makes the total distance for a closed sequence the same as the cost for the one where the robot does not return to the starting task at the end of the sequence. If any constant q is subtracted from any row or column, the total distance of any tour is reduced by q . Therefore, the optimal sequence still remains the same. If a subtraction is carried out from the rows and columns of the matrix such that each row and each column contains at least one zero, the total amount subtracted will be a lower bound on the distance obtained from any solution. This process is called *reduction*. Thus the travel time matrix is reduced by subtracting the lowest entries in each row from their corresponding rows. Then the lowest entry of each column is subtracted from that column. Let the total time obtained from the above subtractions be a ; a is a lower bound on the time of all solutions for the problem.

B.1.2. Branching

The branching will be based on the path selected to split the solution space. Let (i, j) represent the assignment of task i before task j in the sequence. One set will contain solutions that exclude these assignments, while the other one will contain all solutions with this assignment. If the assignment $i - j$ is excluded, the corresponding entry in the travel time matrix, d_{ij} , is changed to $+\infty$. The resulting time matrix is

then reduced again if it is so required, and a new lower bound b is obtained for all solutions that exclude assignment $i-j$. The other set contains all solutions that include the assignment $i-j$, so the i th row and the j th column must be deleted from the reduced time matrix, because this assignment is fixed. The result will be a $(n-1) \times (n-1)$ matrix. Also, since all solutions in this set use the assignment $i-j$, $j-i$ can no longer be used and hence d_{ji} is set to $+\infty$.

The assignment $i-j$ (branching) is selected amongst all assignments such that it would result in the largest lower bound of the new node. This is carried out by choosing the zero in the matrix which when changed to infinity will allow the most to be subtracted from its row and column. This rule is used for branching since it is preferred to look for the solution along the left branches rather than the right ones. This is because the left branches reduce the dimension of the problem while the right ones do not.

It must be kept in mind that the precedence requirements need to be incorporated into this method. This is carried out by checking that the assignment $i-j$ does not violate any precedence relationship. If it does then the set of all solutions which contain the path (i, j) are infeasible and no further branching on that set is required.

This reduction and branching procedure is carried out until all tasks have been assigned. Let the feasible solution so obtained have the total robot travel time of p . All nodes in the search tree with a lower bound greater than p can be rejected or *fathomed*. The ones which have bounds lower than p are expanded further and this procedure is continued until all nodes have either lead to feasible solutions or have been fathomed. The solution with the lowest distance for a feasible sequence is the optimal solution.

B.2. Closest insertion algorithm

We use the Closest Insertion Algorithm presented by Askin and Standridge [14] to obtain a good solution for the robot task-sequencing problem. We start the algorithm by selecting task 1 as the first task; $n-1$ stages are then carried out, with a new task being added at each stage. A partial sequence is always maintained, which grows by one task at each stage. At each stage a task is selected from those unassigned which causes the smallest increase in the length of the sequence. However, the precedence relationships also have to be maintained. This is carried out by choosing from only those unassigned tasks that already have their predecessors in the partial sequence.

Let $T_p = \{t_1, t_2, \dots, t_l\}$ denote the partial sequence of tasks at any stage, where t_j is the j th task in the sequence. Let T_a be the set of schedulable tasks at that stage. T_a excludes any task that does not have its predecessor in T_p . For each unassigned task j , $c(j)$ denotes the task in the partial sequence that is “closest to j ”. This is found by looking at the matrix for the travel times between tasks and choosing the task that is closest to j .

Algorithm

1. Initialize.

$l = 1$. $T_p = \{1\}$. $T_a = \{2, \dots, n\}$. $c(j) = 1$, $j = 2, \dots, n$.

2. In the travel time matrix set incremental times to $+\infty$ for all positions before a predecessor. This ensures that a task can never be placed before its predecessor.

3. Select new task.

Find $j^* = \operatorname{argmin}_{j \in T_a} \{d_{j,c(j)}, \text{ or } d_{c(j),j}\}$. Set $l = l + 1$.

Insert j^* , update $c(j)$. $T_a = T_a - \{j^*\}$.

Find task $i^* \in T_p$ such that $i^* = \operatorname{argmin}_{i \in T_p} \{d_{i,j^*} + d_{j^*,i+1} - d_{i,i+1}\}$.

Update $T_p = \{t_1, \dots, i^*, j^*, i^* + 1, \dots, t_l\}$. If $l < n$, go to Step 2.

Step 3 results in the possibility of tasks being added in the middle of a current partial sequence.

References

- [1] Coombs CF. Printed circuits handbook. New York: McGraw-Hill, 1988.
- [2] Ratliff HD, Rosenthal AS. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Operations Research 1983;31(3):507–21.
- [3] Maimon O. The robot task-sequencing planning problem. IEEE Transactions on Robotics and Automation 1990;6(6):760–5.
- [4] De Fazio TL, Whitney DE. Simplified generation of all mechanical assembly sequences. IEEE Journal of Robotics and Automation 1987;RA-3 (6).
- [5] Kapitanovski A, Maimon O. Robot programming system for assembly: conceptual graph-based approach. Journal of Intelligent and Robotic Systems 1993;8:35–62.
- [6] Garey MR, Johnson DS. Computers and intractability: a guide to the NP-completeness. San Francisco: Freeman, 1979.
- [7] Fausett L. Fundamentals of neural networks: architectures, algorithms, and applications. Englewood Cliffs, NJ: Prentice Hall, 1994.
- [8] Carpaneto G, Toth P. Some new branching and bounding criteria for the asymmetric traveling salesman problem. Management Science 1980;26(7):736–43.
- [9] Syslo MM, Deo N, Kowalik JS. Discrete optimization algorithms: with pascal programs. Englewood Cliffs, NJ: Prentice-Hall, 1983.
- [10] Miller DL, Pekny JF. Exact solution of large asymmetric traveling salesman problems. Science 1991;251:754–61.
- [11] Balas E, Christofides N. A restricted lagrangean approach to the traveling salesman problem. Mathematical Programming 1981;21:19–46.
- [12] Ascheuer N, Escudero LF, Grottschel M, Stoer M. A Cutting plane approach to the sequential ordering problem (with application to job scheduling in manufacturing). SIAM Journal of Optimization 1993;3(1):25–42.
- [13] Edan Y, Flash T, Peiper UM, Shmulevich I, Sarig Y. Near-minimum-time task planning for fruit-picking robots. IEEE Transactions on Robotics and Automation 1991;7(1):48–55.
- [14] Askin RG, Standridge CR. Modeling and analysis of manufacturing systems. New York: Wiley, 1993.
- [15] Han M-H, McGinnis LF, Shieh JS, White JA. On sequencing retrievals in an automated storage/retrieval system. IIE Transactions 1987;19:56–66.
- [16] Chow W-M. An analysis of automated storage and retrieval in manufacturing assembly lines. IIE Transactions 1986;6:204–13.
- [17] Egbelu PJ, Tanchoco JMA. Characterization of automated vehicle

- dispatching rules. *International Journal of Production Research* 1984;22:359–74.
- [18] Jog P, Jung YS, Gucht DV. Parallel genetic algorithms applied to the travelling salesman problem. *SIAM Journal on Optimization* 1991;5:15–29.
- [19] Grefenstette JJ, Gopal R, Rosamita B, Van Gucht D. Genetic algorithms with the travelling salesman problem. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*. Pittsburgh: Carnegie-Mellon University, 1985.
- [20] Storer RH, Wu SD, Vaccari R. New search spaces for sequencing problems with applications to job shop scheduling. *Management Science* 1992;38:1495–509.
- [21] Tate DM, Tunasar C, Smith AE. Genetically improved presequences for euclidean travelling salesman problems. *Mathematical and Computer Modeling* 1994;20(2):135–43.
- [22] Chen C-J, Tseng C-S. The path and location planning of workpieces by genetic algorithms. *Journal of Intelligent Manufacturing* 1996;7:69–76.
- [23] Hopfield JJ, Tank DW. Neural computation of decisions in optimization problems. *Biological Cybernetics* 1985;52:141–52.
- [24] Hopfield JJ, Tank DW. Computing with neural circuits: a model. *Science* 1986;8:625–33.
- [25] Wilson V, Pawley GS. On the stability of the tsp problem algorithm of hopfield and tank. *Biological Cybernetics* 1988;58:63–70.
- [26] Durbin R, Wilshaw DJ. An analog approach to the traveling salesman problem using an elastic net method. *Nature* 1987;326:689–91.
- [27] Szu H. Fast TSP Algorithm based on binary neuron output and analog neuron input using the zero-diagonal interconnect matrix and necessary and sufficient constraints of the permutation matrix. *Proceedings of the ICNN-88* 1988:259–66.
- [28] Hegde SU, Sweet JL, Levy WB. Determination of parameters in a hopfield/tank computational network. *Proceedings of the ICNN 1998*;II:291–8.
- [29] Aiyer VB, Niranjan M, Fallside F. A theoretical investigation into the performance of the Hopfield model. *IEEE Transactions on Neural Networks* 1990;1(2).
- [30] Abe S. Global convergence and suppression of spurious states of the Hopfield neural networks. *IEEE Transactions on Circuit and Systems-I: Fundamental Theory and Applications* 1993;20(4).
- [31] Abe S. Global Convergence of the Hopfield Neural Network with Non-zero Diagonal Elements. Submitted for publication.
- [32] Wacholder E, Han J, Mann RC. A neural network algorithm for the multiple traveling salesman problem. *Biological Cybernetics* 1989;61:11–19.
- [33] Cohen MA, Grossberg S. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man and Cybernetics* 1983;SMC-13:815–26.
- [34] Hopfield JJ. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science* 1984;81:3088–92.
- [35] Platt JC, Barr AH. Constrained differential optimization. *Proceedings of the IEEE, NIPS Conference*, 1987.
- [36] Wang J. A parallel distributed processor for the quadratic assignment problem. *Proceedings of the INNC 90* 1990;1:278–81.
- [37] Angeniol B, de La Croix Vaubois G, Le Texier JY. Self-organizing feature maps and the traveling salesman problem. *Neural Network* 1988;1:289–93.
- [38] Fort JC. Solving a combinatorial problem via self-organizing process: an application of the Kohonen algorithm to the traveling salesman problem. *Biological Cybernetics* 1988;59:33–40.