Balancing a U-Shaped Assembly Line by applying Nested Partitions Method

By

Nikhil V. Bhagwat

A Creative Component submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Industrial Engineering

Program of Study Committee:
Dr. Sigurdur Olafsson (Major Professor)
Dr. John Jackman
Dr. Brian Mennecke

Iowa State University
Ames, Iowa
2005

Acknowledgements

I would sincerely like to thank **Dr. Sigurdur Olafsson** (Associate Professor, Department of Industrial and Manufacturing Systems Engineering) for guiding me through this study. I would also like to thank **Dr. John Jackman** (Associate Professor, Department of Industrial and Manufacturing Systems Engineering) and **Dr. Brian Mennecke** (Associate Professor, Management Information Systems) for being on my graduation committee. I would like to take this opportunity to thank my family members and friends for being supportive and patient.

TABLE OF CONTENTS

1. Introduction

An assembly line consists of a several stations performing various tasks repeatedly on items moving along the line. The first assembly line was developed by Henry Ford in 1915. Since then, they have been widely used in various industries to achieve higher productivity in mass production environments.

Line balancing is the process of assigning sets of tasks to stations to accomplish an assembly work. The duration to perform a task is called the *task time*. The sum of the times of tasks assigned to a station is called the *station time*. *Precedence constraints* specify the permissible sequences of the tasks. This sequence can be influenced by several technological and managerial considerations. Thus, a simple assembly line balancing problem (SALBP) can be defined as *"given a finite set of tasks, each having a task time, and a set of precedence constraints which specify the permissible orderings of the tasks, the problem is to assign the tasks to an ordered sequence of stations with a pre-specified cycle time such that the precedence constraints are satisfied and some performance measure is optimized"* [2]. Typically, the performance measure for a SALBP would be minimizing the number of stations m, given a cycle time c.

1.1 Definition of UALBP

Although a vast amount of research has dealt with SALBP, the straight assembly lines may have several disadvantages associated with them. Some of these include the boredom of work requiring low-level skilled operators, the inflexibility of the production system concerning failures and varying demand etc. These usually result in low-motivation for operators, recurring quality problems and large inventories due to inflexible output rates.

To overcome these drawbacks, modern production methodologies e.g. the Just-in-time principle and flexible manufacturing technology, have been incorporated into assembly line production by several manufacturing facilities. Such modern assembly lines are often organized in shape of U-line (Figure 1).

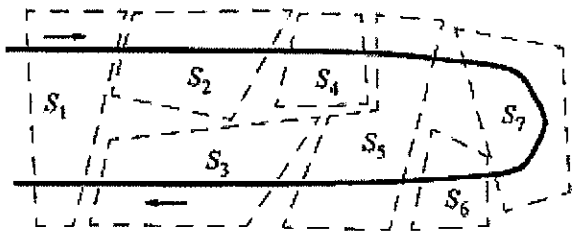

Figure1: U-shaped Assembly line 1

(Taken from ULINO: Optimally balancing U-Shaped JIT assembly lines, Scholl & Klein, 1999)

"Both ends of the line are close together forming a rather narrow *U*" [4]. Stations may be positioned along the two segments of the line facing each other simultaneously. For example, station 1 works at the beginning and at the end of the line, i.e. it may perform the first and the last tasks of a product unit. Operators are involved in different aspects of the production process, thus broadening their skills. Stations are closer together. This helps in having a better visibility of the entire production process and also helps in enhancing communication between the operators. Furthermore, helping each other in cases of bottlenecks and sharing machine capacities is facilitated. As a result of this, workers acquire multiple skills thereby leading to higher motivation, improved quality of products and increased flexibility.

"The U-line assembly balancing problem (UALBP) is an extension of SALBP with respect to the precedence constraints" [4]. In SALBP, all predecessors tasks (direct and indirect) of a task *j* performed at a station *k* must be assigned to one of the stations 1... *k*. A task can share a station *k* with an indirect predecessor or successor task only if all intermediate tasks defining this precedence relationship are also in station *k*. In UALBP, each task, in principle, can share a station with any of its predecessors and/or any of its successors. However, all predecessors and/or successors of a task *j* performed at a station *k* must be assigned to one of the stations 1...*k*. "Due to these relaxed precedence constraints, the optimal line efficiency of a SALBP instance is a lower bound on the optimal line efficiency of the corresponding UALBP instance" [4]. In several cases, a higher efficiency is possible with UALBP

instance. Note that increasing the line efficiency has the further positive effect of smoothing the levels of station utilization i.e. the load distribution amongst stations is more even.

1.2 The Nested Partitions Method

Finding a single optimal solution among a finite set of alternatives is a fairly common problem one encounters. This problem is hard to solve. In most cases, the number of alternatives is quite large and only a handful of them can be considered. Besides, these problems lack rich structure that can be utilized in identifying an optimal solution. As a result, heuristic algorithms are used such as Genetic Algorithm, Tabu Search etc. [7]. Most of these problems have multiple local optima, some of which may have extremely poor performance as compared to the global optimum solution. This makes it inappropriate to apply only local optimization techniques. A popular and effective approach to escape local optima is to use randomization. **Nested Partitions** is a randomized method for global optimization [1]. The motivation for this method is that some parts of the feasible region may be most likely to contain the global optima. Hence it is efficient to concentrate the computational effort in these regions. The Nested Partitions Method takes a global perspective and combines global and local search techniques. This method is also motivated by solving problems that have a finite feasible region (referred to as combinatorial optimization problems).

1.3 Objective

The primary objective of this study was to see if near-optimal solutions could be obtained in a reasonable amount of time by applying a meta-heuristic such as the Nested Partitions Method to the problem of Balancing U-Shaped Assembly Lines (UALBP) wherein the task times are deterministic.

2. Illustration of UALBP

"When standardized products have to be produced in large quantities, assembly lines are the appropriate production systems" [4]. An assembly line typically consists of a sequence of m work stations through which the product units proceed. Each station performs a subset of the n tasks (operations) necessary for manufacturing the products. Due to the movement of the line, each product unit remains at a station for a fixed amount of time referred to as the cycle time c. In conventional assembly lines, stations are arranged one after the other in the form of a straight line. Each product unit proceeds along this line and visits a station only once (except if it has to be re-worked). However, for reasons stated above, U-shaped assembly lines have started replacing the conventional straight assembly lines.

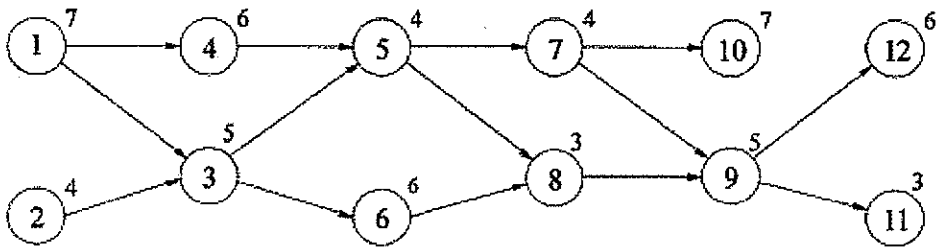

Figure2: Precedence Graph

(Taken from ULINO: Optimally balancing U-Shaped JIT assembly lines, Scholl & Klein, 1999)

The figure above shows an example of a precedence network with $n = 12$ tasks wherein tasks are represented as nodes, precedence relations as arcs and task times as node weights. Let j denote tasks required to manufacture a single product unit and t_j indicate the deterministic task time for j^{th} task. An arc (i, j) means that task i should be completed before task j can be started. Each task can be assigned to exactly one station. The sets of tasks S_k assigned to station $k = 1 \ldots m$ are called station loads. Let the sum of all task times be designated by t_{sum}. The objective usually consists of maximizing the line efficiency given by:

$$E = \frac{t_{sum}}{(m * c)} * 100\ \%$$

Assume that the cycle time for the above problem is $c = 10$. The figure below illustrates an optimal solution for UALBP with six stations (stations have been numbered consecutively from left to right).

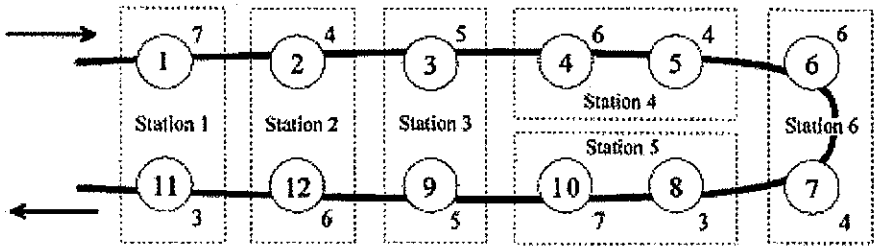

Figure3: Solution to the example problem for c = 10

(Taken from ULINO: Optimally balancing U-Shaped JIT assembly lines, Scholl & Klein, 1999)

Let us examine station 3 which performs the tasks 3 and 9, i.e. $S_3 = \{3, 9\}$. Task 3 is performed whenever a product unit crosses the station for the first time (from left to right) after its predecessors have been performed in stations 1 and 2 respectively. When the product unit returns to station 3 (from right to left), all predecessors of task 9 have already been performed (in stations 1-6) and task 9 can be performed. Following this, the successors of 9 are executed in stations 2 and 1 respectively.
An optimal solution to the corresponding SALBP instance requires seven stations: $S_1 = \{1\}$, $S_2 = \{22, 4\}$, $S_3 = \{3, 5\}$, $S_4 = \{6, 7\}$, $S_5 = \{8, 10\}$, $S_6 = \{9, 11\}$, $S_7 = \{12\}$. Due to $t_{sum} = 60$, the efficiency of the U-shaped line is 100% whereas that of the straight line is only 85.7%.

3. Nested Partitions Method

Nested Partitions method deals with problems of the following form:

$$\theta^* \in \arg \min_{\theta \in \Theta} f(\theta)$$

Where Θ is the finite solution set and f: $\Theta \rightarrow$ R is the performance function to be optimized [1].

This problem is mathematically trivial in the sense that all we need to do is enumerate and compare all the points to find the one with the best performance. However, the huge number of alternatives makes this approach infeasible. Some problems have a rich special structure that can be exploited to find an optimal solution without checking all the alternatives, but many problems lack such structure. For these problems, the only known method to guarantee an optimal solution is to check all the alternatives.

In the Nested Partitions method; in each iteration of the algorithm, we assume that we have a region, i.e. a subset of Θ that is considered *most promising*. We then partition this promising region into M sub-regions and aggregate the entire surrounding region into one region. In each iteration, we therefore look at $M + 1$ disjoint subsets of the feasible region. Each of these $M + 1$ regions is sampled using some random sampling scheme, and the performance function values of the randomly selected samples are used to calculate the promising index for each region. This index determines which region is the most promising region for the next iteration. If one of the sub-regions is found to be the best, this region becomes the most promising region. If the surrounding region is found to be the best, the algorithm backtracks to a larger region containing the old most promising region. The new most promising region is then partitioned and sampled in a similar fashion.

In the first iteration, nothing is assumed to be known about where good solutions are. Hence we use the entire feasible region Θ as the most promising region. Since the surrounding region is empty, we sample only from M region in the first iteration where Θ is considered as the most promising region. It is also clear that Θ is finite,

eventually there will be regions that will be singletons, that is, contain single point. Such regions are the regions of maximum depth and generally talk about the depth of a region. Such regions yield the complete solutions and can be considered as terminating points for the algorithm. This is defined iteratively in the obvious manner with Θ having depth of 0 and so forth.

Assume that a partitioning scheme has been fixed. This means that it has been decided how many sub-regions there will be, and what rule is followed in partitioning any given region into M regions. Let us call a region constructed using the partitioning scheme as a *valid region* given the fixed partition. If a valid region σ is formed by partitioning a valid region η, then σ is called the sub-region of region η, and region η is called *super-region* of region σ.

Let us look at the following illustration. Consider a feasible region consisting of eight points $\eta_0 = \Theta = \{1, 2, 3, 4, 5, 6\ 7, 8\}$ and that in each iteration, we partition the current most promising region into $M = 2$ disjoint sets as shown in the figure below.

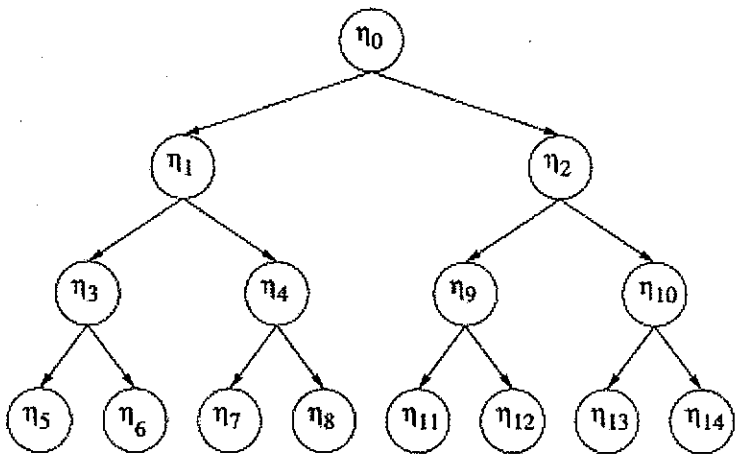

Figure4: Example of partition generated by applying NP method
(Taken from Nested Partitions Method for Global Optimization: Shi & Olafsson, 2000)

In the first iteration, the current most promising region η_0 is partitioned into two sub-regions $\eta_1 = \{1, 2, 3, 4\}$ and $\eta_2 = \{5, 6, 7, 8\}$. Both η_1 and η_2 are sampled and the promising index of each region is estimated. Assume that the estimated promising index of η_1 is better than that of region η_2. We then select η_1 as the most promising

region in the second iteration and further partition it into two sub-regions $\eta_3 = \{1, 2\}$ and $\eta_4 = \{3, 4\}$. In the second iteration, η_3, η_4 and their surrounding region η_2 are sampled. Assume that the estimated promising index of region η_3 is the best. We then select η_3 to be the most promising region in the third iteration and partition η_3 further into two sub-regions $\eta_5 = \{1\}$ and $\eta_6 = \{2\}$. Thus, in the third iteration, η_5, η_6 and their surrounding region $\eta_0 \setminus (\eta_5 \cup \eta_6)$ are sampled and their most promising index values are estimated. If the estimated promising index of the surrounding region is the best, we backtrack to a larger region containing η_3. In this case, that would either be η_0 or η_1.

4. Proposed Method

To meet the objectives of this study, a metaheuristic algorithm was required to be applied to the UALBP. Nested Partitions is known to have worked well for several practical problems such as the scheduling of tasks in a single machine environment, scheduling parallel manufacturing systems with flexible resources [10] etc. Hence, Nested Partitions method was chosen. As a benchmark for evaluation, Branch and Bound method was also applied to the same problem. This section elaborates the approach used for both the algorithms. To evaluate the performance of the two algorithms, they were programmed in C#.Net and several tests were conducted. The results of these tests, as well as detailed analysis of the results are presented in the next section. Both these algorithms are based on the following assumptions:

- Each UALBP has a well-defined precedence relationship.
- The task times are deterministic.
- The objective function is: Determining the number of stations required to perform all the designated tasks for a given cycle time. The smaller the number, the better is the performance of the algorithms.
- Backtracking (if required) is permitted.
- The maximum task time is less that or equal to the cycle time.

4.1 Branch and Bound Method as applied to UALBP

Branch and Bound procedures are enumeration schemes. In these schemes, certain schedules or classes of schedules are discarded by showing that the objective values of all these schedules are higher than a provable lower bound and this lower bound is higher than (or equal to) the value of the objective of a schedule obtained earlier. The Branch and Bound procedure for UALBP has been constructed as follows: The branching process is based on the fact that complete solutions are developed starting from the beginning of the solution [7]. There is a single node at level 0, which is at the top of the tree. At this node, none of the tasks have been assigned to any stations. There will be n branches going down n nodes at level 1 where n stands for the number of tasks that can be assigned to current stations / new stations (if assignment to current station is not possible). By *tasks that can be assigned to stations* we mean

those tasks which have already had all their predecessor (direct or indirect) / successor (direct or indirect) tasks assigned to one of the stations: *1* ... *k* where *k* is the current station. For level 1, since there is no station opened so far, *k* will be *1*. Each node at this level will correspond to a partial solution with a specific task in the first station. There are several arcs emanating from each node at level 1. The precedence relationship determines the number of tasks that become available for assignment, after the assignment of the task at level 1. It could be different for different nodes at the same level. At level 2, the first two tasks have been assigned to stations. The task assignment to a station is based on the assumption that there is sufficient time available at the station after the initial assignment. The available station time is computed as follows:

Available station time = cycle time – (sum of all the tasks assigned to the station)

A task will be assigned to a station only if the available time for the station is at least equal to the task time; else, a new station will be created (opened) and the task will be assigned to the new station. This task assignment will continue until all the tasks have been assigned to stations. The depth of the tree formed thus, will always be equal to the number of tasks in the problem.

A *Depth first* approach has been used for branching; i.e. one branch is traversed until a complete solution is obtained. Remaining solutions (branches) are obtained by backtracking. The solution obtained by traversing the first branch of the tree is initially considered as the *bounding solution*. During the branching process, a complete solution represented by a branch is compared for quality with the *bounding solution*. If the solution of the current branch has fewer stations as compared to the bounding solution, the current branch's solution becomes the new *bounding solution*; else the branch is terminated (bound).

Consider the following illustration consisting of four tasks and the precedence relationship as shown. For the sake of simplicity, all the four tasks are assumed to have the same time value viz. *10*. Let the cycle time *c* be 20.

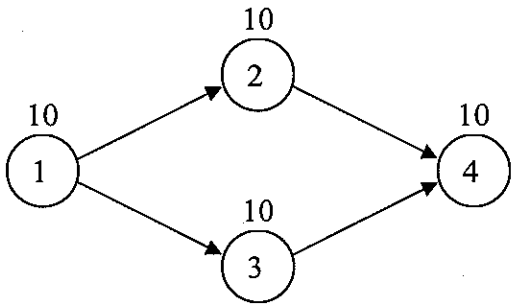

Figure5: Precedence graph for the illustration under consideration

A complete solution set for the above problem using the Branch and Bound procedure is shown below:

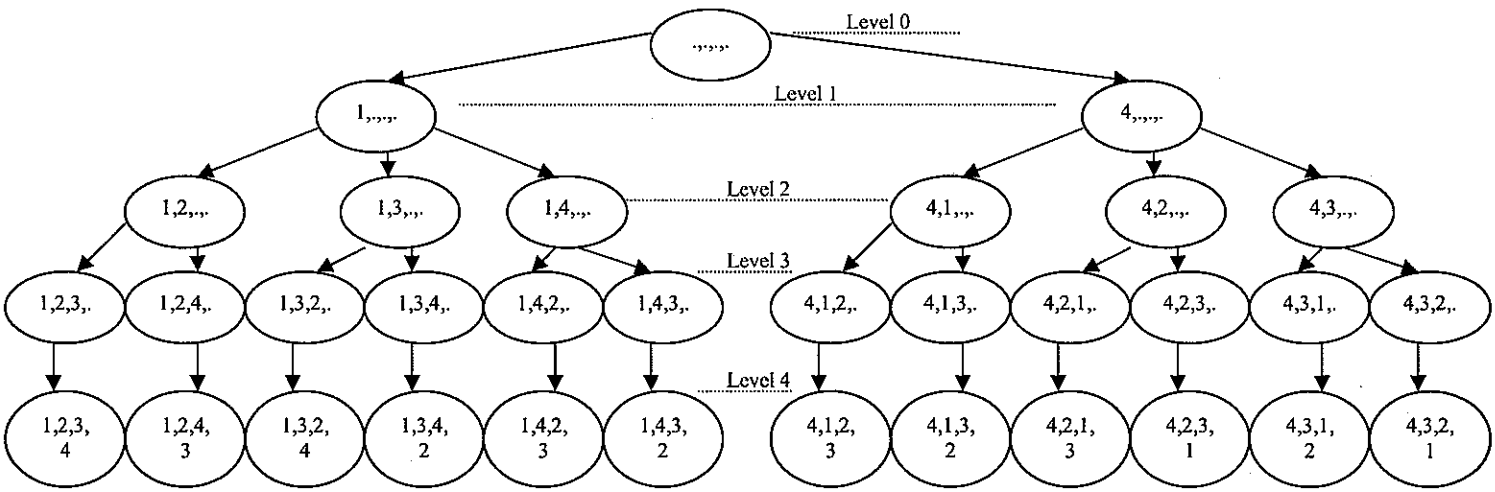

Figure6: Solution set for the illustration with cycle time = 20

At the beginning, i.e. when we are at *level 0*, none of the tasks have been assigned to any stations. At this point in time, task *1* and task *4* are available for making assignments to stations as they do not have any unassigned predecessors or successors respectively. Since we do not have any stations, we begin the branching process by opening a new station *1* and assigning task *1* to this station (level *1*). We traverse down this branch until we have a complete solution. Having assigned task *1* to station *1*, we now have tasks *2, 3* and *4* available for the next assignment (tasks *2* and *3* are

available for assignment as their predecessor task *1* has been assigned). Next, we try assigning task *2* (level 2). The available time for station *1* is computed as follows:

Available time = Cycle time – Sum (All tasks assigned to the station) i.e.
Available time = 20 – 10 = 10

Since the time for task *2* is 10 (which is <= available station time), we assign task *2* to station *1* itself i.e. we do not open a new station for assigning task *2*. Traversing down the branch, we are now left with tasks *3* and *4* for assignment. We assign task *3* next (level 3). Since the available time for station *1* is zero, we open a new station *2* and assign task *3* to this new station. We are no left with task *4*. This task can be assigned to station *2* as well since its available time (10) is equal to the time of task *4* (level 4). We are left with no more tasks. Thus we have a complete solution. The value of the objective function of this solution is equal to the number of stations in the solution i.e. *2*. We treat the first solution as the *bounding solution.*

We now begin backtracking (un-assigning tasks in the reverse order in which they were assigned) in search of a better solution. From level *4*, we un-assign task *4* (the last assigned task) and move to level *3*. At this level, we check if we have any tasks (other than *4*) for making assignments. Since we do not have any such tasks, we backtrack to level *2* by un-assigning task *3* as well. Having unassigned task *3*, the value of the *partial solution* becomes equal to *1* (station *2* is discarded as it does not have any tasks assigned to it). We again check if we have any tasks (other than task *3* which was previously assigned at this level) for making assignments. Task *4* is the only task that is available. However, for assigning task *4*, we have to open a new station (station *2*) as station *1* does not have any available time. This would mean that a complete solution obtained by making the assignment of task *4* to the new station *2* would have a objective function value of at least *2* which is not better than the value of the *bounding solution* i.e. *2*. Hence, we disregard this branch and continue backtracking by un-assigning task *2*. Once again, we check for the availability of tasks (other than task *2*). Tasks *3* and *4* are both available for assignment. We choose

task *3* and assign it to station *1* since it the available time for station is equal to the time of task *3*. This partial solution's value is still under the bounding solution's value. So we continue with assignments. Next, we have tasks *2* and *4* available for assignment. We chose task *2*. At this point in time, we find that we have to assign task *2* to a new station *2* since the available time for station *1* is zero. If we make this assignment, then we are bound to end up with a complete solution having the objective function's value of at least *2* which is not better than the value of that of the *bounding solution*. As a result of this, we disregard this branch and continue backtracking in order to seek a better solution.

The same procedure is continued until we exhaust all the available alternatives. If during this backtracking, we encounter a (complete) solution having the objective function's value lesser than that of the *bounding solution*, that solution becomes the new bounding solution and the older solution is disregarded. At the end of all the iterations, the bounding solution is considered as the final (best) solution.

4.2 Nested Partitions Method as applied to UALBP

Nested Partitions method is similar to Branch & Bound and Beam search methods. Random sampling and local search techniques are used to find which node is the most promising of all. At any given point in time, only one node (Beam Width = 1) [1] is retained with a possibility of backtracking (if necessary). The implementation details are given below.

The algorithm is based on the fact that complete solutions are developed starting from the beginning of the solution. There is a single node at the top. At this node, none of the tasks have been assigned to any stations. We begin by partitioning the available region (most promising region) into several regions on the basis of availability of tasks for assignment. All those tasks that have their predecessors / successors already assigned are considered available. Referring to the illustration in figure 5, we have tasks *1* and *4* available for assignment. Hence, we can partition the current most promising region into two regions viz. region *1* with task *1* and region *2* with task *4* as shown below.

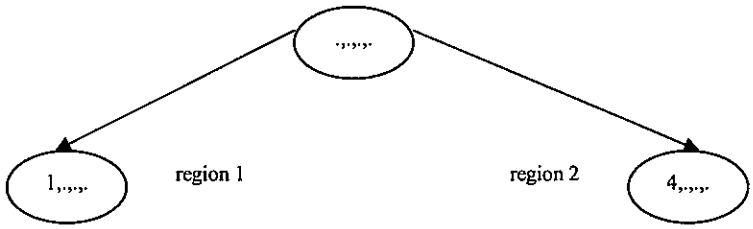

Figure7: Regions to be considered for the first iteration

At this point in time, we do not have a neighboring region. To proceed with task assignment, one of these two regions has to be picked as the next *most promising region*. To do so, we have to compare the values of the *performance indices* of the two regions. The region with a better performance index is considered as the next *most promising region* and the task associated with that region is assigned to a station subject to the availability of sufficient time at the station (i.e. available time is at least equal to the task time), else a new station is opened.
For the purpose of this study, the value of the performance index of a region is computed as follows:

From each region, some N samples are obtained randomly. In reality, these samples are *complete solutions* that are obtained by assigning the task associated with that region to an existing / new station (depending on the availability of sufficient time at the station). For obtaining complete solutions, subsequent assignments are made by choosing tasks randomly from the group of available tasks. This process of random selection and assignment of tasks to existing / new stations continues until all task assignments are completed. The number of stations required for each of the N sample solutions is determined. The least value amongst these is chosen as the *performance index* of that region. Similarly, the performance indices of other regions are obtained. The values of the performance indices of all these regions are compared and the region with the least value is chosen as the next *most promising region* as mentioned earlier.

There are a several ways in which the number of samples N for a region can be determined. For the purpose of this study, Rinott's *two-stage ranking and selection*

procedure [8] has been used. In this method, initially, a few random samples (complete solutions) are considered from the region n_0. The number of these initial samples can be chosen randomly. Then the variance S is computed for these samples. Using the variance value and a few other parameters, the actual number of samples required for the region N is determined. If the actual number exceeds the initial number of samples, then a few more samples (equal to the difference) are taken from the region. The number N is computed as follows:

$$N = \max\left\{n_0, \frac{h^2 S^2}{e^2}\right\}$$

Where e is the indifference zone (The amount by which the difference between the values of two solutions in a region is considered insignificant); and h is a constant that is determined by n_0 and the minimum probability P^* of correct selection. For the purpose of this study, the following table of constants was used (Note that k is the number of regions for an iteration) [11].

P^*	0.90	0.90	0.95	0.95
N_0	20	40	20	40
k = 2	1.896	1.852	2.453	2.386
k = 3	2.342	2.283	2.872	2.786
k = 4	2.583	2.514	3.101	3.003
k = 5	2.747	2.669	3.258	3.150
k = 6	2.870	2.785	3.377	3.260
k = 7	2.969	2.878	3.472	3.349
k = 8	3.051	2.954	3.551	3.422
k = 9	3.121	3.019	3.619	3.484
k >= 10	3.182	3.182	3.679	3.539

Table1: Values of constant *h*

Coming back to the illustration, we have two regions viz. region *1* associated with task *1* and region *2* associated with task *4*. Using the sampling procedure described above, we determine the performance indices of the two regions. The region having a

better performance index (*lower* in the case of this study) is selected. Assume that the performance index of region *1* is lower than that of region *2*. Region *1* thus becomes the *most promising region*. All the other regions (region *2* in this case) become a part of the *neighboring region*. The task associated with region *1* i.e. task *1* is assigned to a new station *1* since there are no stations in the beginning.

The most promising region i.e. region *1* is now partitioned. The number of regions is equal to the number of tasks that have become available for assignment (since the preceding / succeeding task/s have been assigned). Having assigned task *1* to station *1*, tasks *2*, *3* and *4* are available for subsequent assignments. Thus the number of regions under the most promising region is *3* as shown below. However, the total number of regions at this point in time also include the neighboring region *1* associated with task *4*. Thus, we have 4 regions for the second iteration.

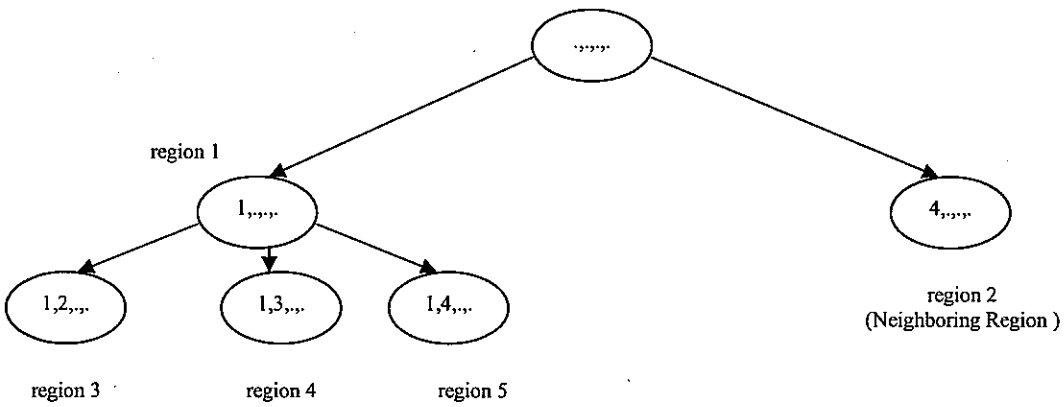

Figure8: Regions to be considered for the second iteration

For each of these 4 regions, we compute the values of their respective performance indices and compare these. Assume that region *4* (task *3* followed by task *1*) has the least value of the performance index. Thus this region becomes the next *most promising region* and the task associated with this region (task *3*) is also assigned to a station. Since the available time for station *1* is 10 (which is equal to the time for task *3*), it is assigned to station *1*.

For the next iteration, region *4* is partitioned into its sub-regions. The tasks associated with these sub-regions are *2* and *3* (since these tasks have all their predecessors / successors assigned to stations). The region *3*, region *5* and region *2* combined together form the new *neighboring region*. This process continues until there are no more tasks available for assignment to stations. This will result in a complete solution consisting of all the tasks assigned to some or the other station. During the execution of this method, if the *neighboring region* is found to have a better (lower) performance index, the algorithm backtracks to the previous *most promising region* and the last assigned task is un-assigned from its station. If the station does not have any more assigned tasks left, the station is removed from the solution as well.

5. Test results and Analysis

An experiment was conducted to compare the Nested Partitions method and the Branch & Bound method for the UALBP. In another experiment, the results obtained for different input parameters for the Nested Partitions method were compared for the UALBP. The objective function for the two algorithms was determining the number of stations required for a given problem for a given cycle time. In order to get fair comparison, both the algorithms were coded using *C#* language in *Microsoft.NET 1.1* framework. In both the cases, the same method was implemented for storing precedence relationships. All the tests were performed on an Intel Pentium 4 processor running on *Microsoft Windows XP* operating system with 512 MB RAM. For the purpose of conducting these tests, the following data sets were used:

- Talbot data set (Talbot, 1986) comprising of 64 instances with number of tasks ranging from 8 to 111.
- Hoffmann data set (Hoffmann, (1990, 1992)) comprising of 50 instances with number of tasks ranging from 30 to 111
- Scholl data set (Scholl, 1993) comprising of 168 instances with number of tasks ranging from 25 to 297

Details of the combined data sets are given below (*comp* indicates if the dataset was used in comparison of Nested Partitions and Branch & Bound):

Dataset	No. of Tasks	T_{min}	T_{max}	T_{sum}	OS	TV	DIV	CONV
Arcus1	83	233	3691	75707	59.09	15.84	0.73	0.73
Arcus2	111	10	5689	150399	40.38	568.90	0.63	0.63
Barthold2	148	1	83	4234	25.80	83.00	0.74	0.72
Bowman (comp)	8	3	17	75	75.00	5.67	0.88	0.80
Buxey	29	1	25	324	50.74	25.00	0.74	0.78
Gunther	35	1	40	483	59.50	40.00	0.78	0.76
Hahn	53	40	1775	14026	83.82	44.38	0.63	0.63
Jackson (comp)	11	1	7	46	58.18	7.00	0.77	0.77
Jaeschke (comp)	9	1	6	37	83.33	6.00	0.73	0.73
Kilbridge	45	3	55	552	44.55	18.33	0.67	0.69
Lutz1	32	100	1400	14140	83.47	14.00	0.76	0.82
Lutz2	89	1	10	485	77.55	10.00	0.75	0.75
Lutz3	89	1	74	1644	77.55	74.00	0.75	0.75
Mansoor (comp)	11	2	45	185	60.00	22.50	0.79	0.91
Mertens (comp)	7	1	6	29	52.38	6.00	1.00	0.78
Mitchell	21	1	13	105	70.95	13.00	0.74	0.70
Roszieg	25	1	13	125	71.67	13.00	0.74	0.69
Wee-Mag	75	2	27	1499	22.67	13.50	0.85	0.62

Table2: Input task data sets and their measures

The following measures were calculated for the data sets:

- Number of tasks (n)
- Minimum task time (T_{min})

- Maximum task time (T_{max})
- Sum of tasks times (T_{sum})
- Order strength (OV) = number of all precedence relations / (n * (n-1))
- Task time variability ratio (TV) = Tmax / Tmin
- Degree of divergence of precedence graph (DIV)
- Degree of convergence of precedence graph (CONV)

For executing the *Branch and Bound* algorithm, cycle time *c* was used as the input parameter. For executing *Nested Partitions* method, cycle time *c*, indifference zone *e* initial number of samples for each iteration n_0 and the minimum probability of correct selection P^* were used as the input parameters.

The following output parameters were measured for the two heuristics:

- Number of stations (N_s)
- Computing time (CT) in milliseconds
- Balance Efficiency (E)
- Percentage relative deviation from optimality (R %)

5.1 Test results

The ULINO algorithm was run against the datasets for obtaining the optimum number of stations N^*. Table *3* summarizes the comparative results obtained by running the two algorithms on the datasets mentioned in the table (i.e. results of experiment 1). Table *4* summarizes the comparative results obtained by running the Nested Partitions method for different sets of input parameters (i.e. results of experiment 2).

Dataset	T_{sum}	c	N^*	Branch & Bound			Nested Partitions ($P^* = 0.90$, $N_0 = 20$, $e = 1$)							
							Average Values				Standard Deviation Values			
				CT	N_s	E	CT	N_s	E	R	CT	N_s	E	R
Bowman	75	20	4	31.3	4	93.8	29.7	4	94	0	10.96	0	0.06	0
Jackson	46	7	7	1140.6	7	93.9	64.1	8	88	7	10.95	0.5	5.99	7
		10	5	781.3	5	92	67.2	5	92	0	25.23	0	0	0
		13	4	703.1	4	88.5	54.7	4	89	0	10.47	0	0.15	0
		14	4	1250	4	82.1	57.8	4	82	0	9.98	0	0.03	0
		21	3	1750	3	73	56.3	3	73	0	10.35	0	0	0
Jaeschke	37	6	8	93.8	8	77.1	40.6	8	77	0	10.37	0	0.03	0
		18	3	78.1	3	68.5	31.3	3	69	0	12.11	0	0.15	0
Mansoor	185	48	4	343.8	4	96.4	56.3	4	96	0	14.32	0	0.12	0
		62	3	234.4	3	99.5	53.1	4	77	30	10.35	0.3	7.51	9.9
		94	2	62.5	2	98.4	46.9	2	98	0	12.08	0	0.12	0
Mertens	29	6	6	46.9	6	80.6	23.4	6	81	0	10.51	0	0.12	0
		7	5	31.3	5	82.9	155	5	83	0	380.78	0	0.03	0
		8	5	46.9	5	72.5	25	5	73	0	14.34	0	0.15	0
		10	3	31.3	3	96.7	127	3	97	0	327.65	0	0.09	0
		15	2	15.6	2	96.7	20.3	2	97	0	14.07	0	0.09	0
		18	2	31.3	2	80.6	20.3	2	81	0	10.03	0	0.12	0

Table3: Comparative results for the two algorithms

Dataset	c	N^*	Nested Partitions ($P^*=0.90; N_0=20; e=1$)				Nested Partitions ($P^*=0.90; N_0=40; e=1$)				Nested Partitions ($P^*=0.95; N_0=20; e=1$)				Nested Partitions ($P^*=0.95; N_0=40; e=1$)			
			CT	N_t	E	R	CT	N_t	E	R	CT	N_t	E	R	CT	N_t	E	R
Bowman	20	4	46.9	4	93.8	0	62.5	4	93.8	0	31.3	4	93.8	0	46.9	4	93.8	0
Jackson	7	7	93.8	8	82.1	14	109.4	7	93.8	0	62.5	8	82.1	14	125	7	93.8	0
	10	5	140.6	5	92	0	109.4	5	92	0	62.5	5	92	0	109.4	5	92	0
	13	4	78.1	4	88.5	0	93.8	4	88.5	0	46.9	4	88.5	0	93.8	4	88.5	0
	14	4	78.1	4	82.1	0	109.4	4	82.1	0	46.9	4	82.1	0	109.4	4	82.1	0
	21	3	78.1	3	73	0	109.4	3	73	0	46.9	3	73	0	93.8	3	73	0
Jaeschke	6	8	62.5	8	77.1	0	62.5	8	77	0	46.9	8	77	0	62.5	8	77	0
	18	3	62.5	3	68.5	0	62.5	3	68.5	0	31.3	3	68.5	0	46.9	3	68.5	0
Mansoor	48	4	93.8	4	96.4	0	62.5	4	96.4	0	62.5	4	96.4	0	93.8	4	96.4	0
	62	3	78.1	4	74.6	33	93.8	4	74.6	33	62.5	4	74.6	33	93.8	4	74.6	33
	94	2	62.5	2	98.4	0	78.1	2	98.4	0	46.9	2	98.4	0	93.8	2	98.4	0
Mertens	6	6	46.9	6	80.6	0	78.1	6	80.6	0	31.3	6	80.6	0	46.9	6	80.6	0
	7	5	1296.9	5	82.9	0	62.5	5	82.9	0	31.3	5	82.9	0	62.5	5	82.9	0
	8	5	62.5	5	72.5	0	46.9	5	72.5	0	15.6	5	72.5	0	140.6	5	72.5	0
	10	3	1109.4	3	96.7	0	46.9	3	96.7	0	31.3	3	96.7	0	31.3	3	96.7	0
	15	2	62.5	2	96.7	0	31.3	2	96.7	0	15.6	2	96.7	0	31.3	2	96.7	0
	18	2	46.9	2	80.6	0	31.3	2	80.6	0	15.6	2	80.6	0	31.3	2	80.6	0
Mitchell	14	8	515.6	9	83.3	12	625	8	93.8	0	468.8	9	83.3	12	562.5	8	93.8	0
	15	8	13953.1	8	87.5	0	703.1	8	87.5	0	328.1	8	87.5	0	687.5	8	87.5	0
	21	5	359.4	5	100	0	625	5	100	0	343.8	5	100	0	609.4	5	100	0
	26	5	343.8	5	80.8	0	656.25	5	80.8	0	328.1	5	80.8	0	656.3	5	80.8	0
	35	3	296.9	3	100	0	656.25	3	100	0	312.5	3	100	0	578.1	3	100	0
	39	3	343.8	3	89.7	0	625	3	89.7	0	328.1	3	89.7	0	625	3	89.7	0
Roszieg	14	9	1015.6	10	89.3	11	1890	10	89.3	11	859.4	10	89.3	11	2343.8	10	89.3	11
	16	8	921.9	8	97.7	0	1781.3	8	97.7	0	921.9	9	86.8	12	1671.9	8	97.7	0
	18	7	906.3	8	86.8	14	1593.8	8	86.8	14	890.6	8	86.8	14	1875	8	86.8	14
	25	5	812.5	5	100	0	1578.1	5	100	0	921.9	6	83.3	20	1484.4	5	100	0
	32	4	921.9	4	97.7	0	1687.5	4	97.7	0	828.1	4	97.7	0	1671.9	4	97.7	0
Arcus1	5048	16	28937.5	17	88.2	6	142125	17	88.2	6	36109.4	17	88.2	6	50218.8	16	93.7	0
	5853	13	26875	14	92.4	7	54593.8	14	92.4	7	27953.1	14	92.4	7	55562.5	14	92.4	7
	6842	12	26984.4	12	92.2	0	53750	12	92.2	0	26859.4	12	92.2	0	55562.5	12	92.2	0
	7571	11	27468.8	11	90.9	0	53921.9	11	90.9	0	26843.8	11	90.9	0	54234.4	11	90.9	0
	8412	10	27078.1	10	90	0	55156.3	10	90	0	27171.9	10	90	0	54281.3	10	90	0
	8898	9	27359.4	9	94.5	0	55171.9	9	94.5	0	30203.1	9	94.5	0	55046.9	9	94.5	0
	10816	7	27171.9	8	87.5	14	54750	8	87.5	14	26843.8	8	87.5	14	54046.9	8	87.5	14
Arcus2	5755	27	234218.8	30	87.1	11	472515.6	30	87.1	11	274968.8	30	87.1	11	460546.9	31	84.3	14
	8847	18	295406.3	19	89.5	5	526031.3	19	89.5	5	313171.9	19	89.5	5	246281.3	19	89.5	5
	10027	15	129796.9	17	88.2	13	321187.5	16	93.7	6	126906.3	17	88.2	13	494218.8	16	93.7	6
	10743	14	216187.5	16	87.5	14	520234.4	16	87.5	14	217828.1	15	93.3	7	496046.9	16	87.5	14
	11378	14	207140.6	14	94.4	0	591437.8	14	94.4	0	138828.1	15	88.1	7	623937.5	15	88.1	7
	17067	9	138312.5	9	97.9	0	529187.5	9	97.9	0	122656.3	10	88.1	11	409171.9	9	97.9	0

Table4: Results for the Nested Partitions method for different input parameters

5.2 Analysis of results

Table 3 shows that the quality of the solution obtained by applying Nested Partitions method is almost same as that obtained by applying the Branch and Bound method (which has provided optimum solution in all the test cases). *Computation Time* comparisons show that the Nested Partitions method is a lot faster than the Branch and Bound method, barring a few cases for relatively smaller problems. For bigger problems, the Branch and Bound method could not complete the execution for a sufficiently long period of time (in some cases, results were not obtained for as long as 24 hours after which the execution was stopped.) For the same problems, the Nested Partitions method generated good solutions in a very short amount of time. The influence of the various input parameters on the quality of the solutions and computation times have been summarized in the following graphs.

5.2.1 Graphical comparison of Branch & Bound and Nested Partitions method

Figures 9, 10 and 11 compare the Branch & Bound method and the Nested Partitions method.

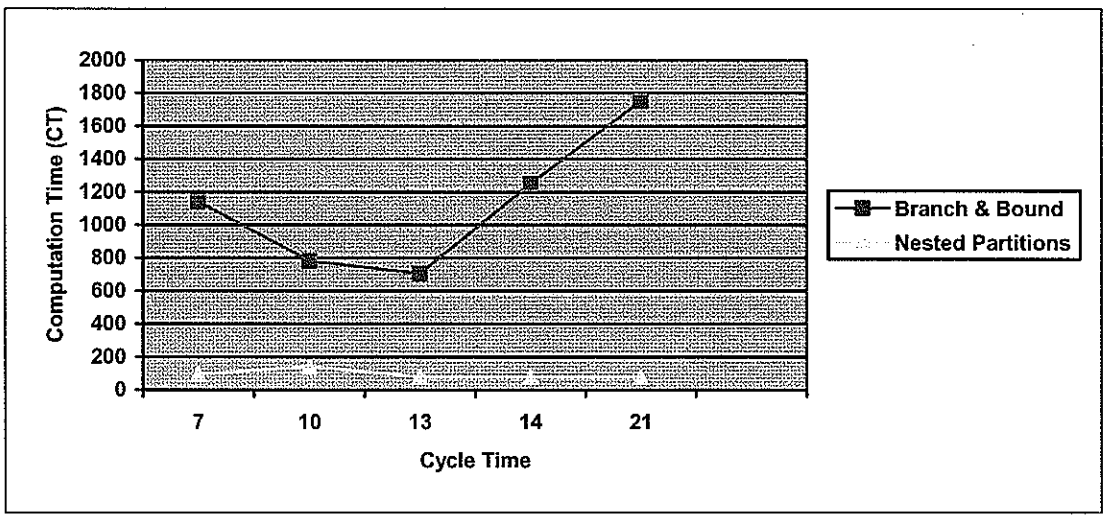

Figure9: Influence of cycle time on computation time for Jackson's data set

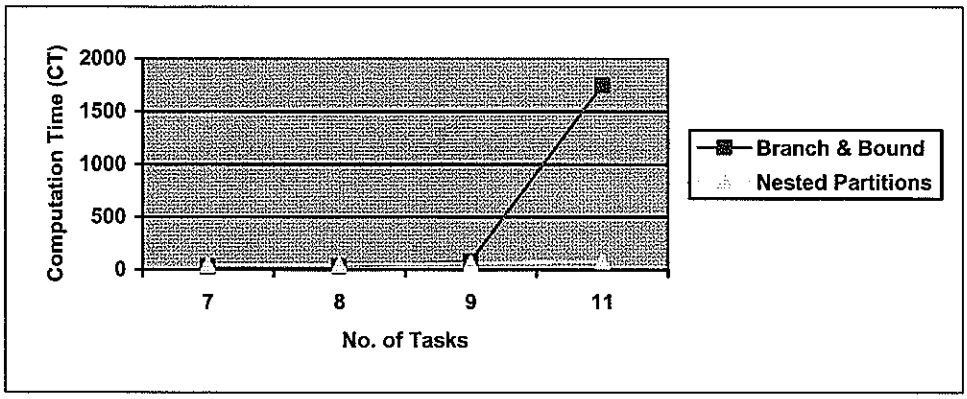

Figure10: Influence of No. of tasks on computation time

In the graph above, results for only a few instances of smaller data sets have been plotted. Concrete results for larger data sets for Branch & Bound method could not be obtained as the computation time very well exceeded the computation time for Nested Partitions method for the same problems (thereby outperforming the Branch & Bound method).

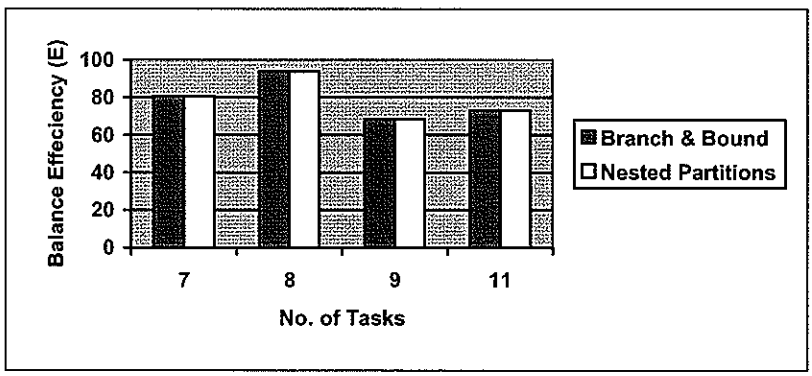

Figure11: Influence of No. of tasks on Balance Efficiency

The balance efficiency for both the algorithms is same; the reason being that the solutions obtained for both these algorithms are almost same and are almost always optimal for smaller data sets.

5.2.2 Graphical comparison of Nested Partitions method for different sets of input parameters

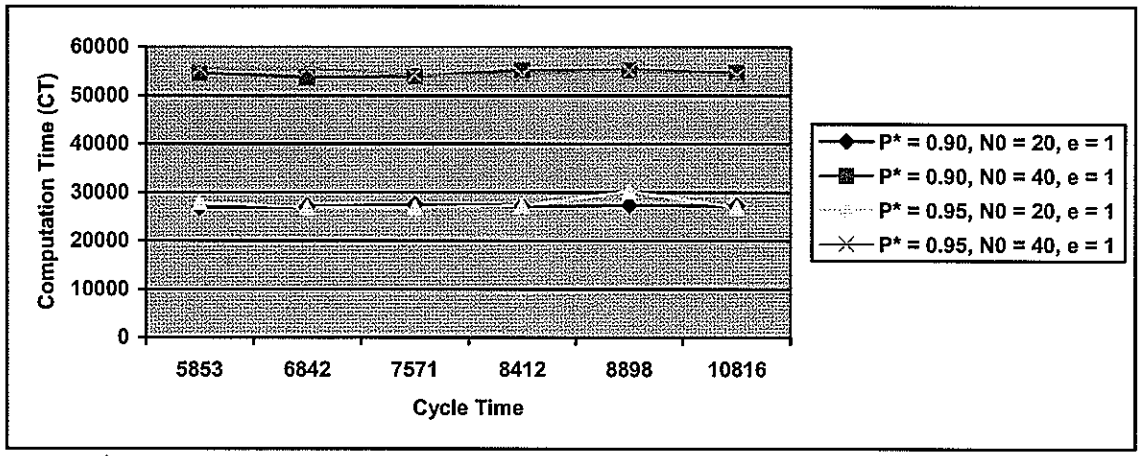

Figure12: Influence of Cycle Time on Computation time for Arcus1 data set

As is evident from the graph above, the computation time remains more or less constant even if the cycle time varies for the four sets of input parameters. It is more influenced by the number of tasks as is shown below.

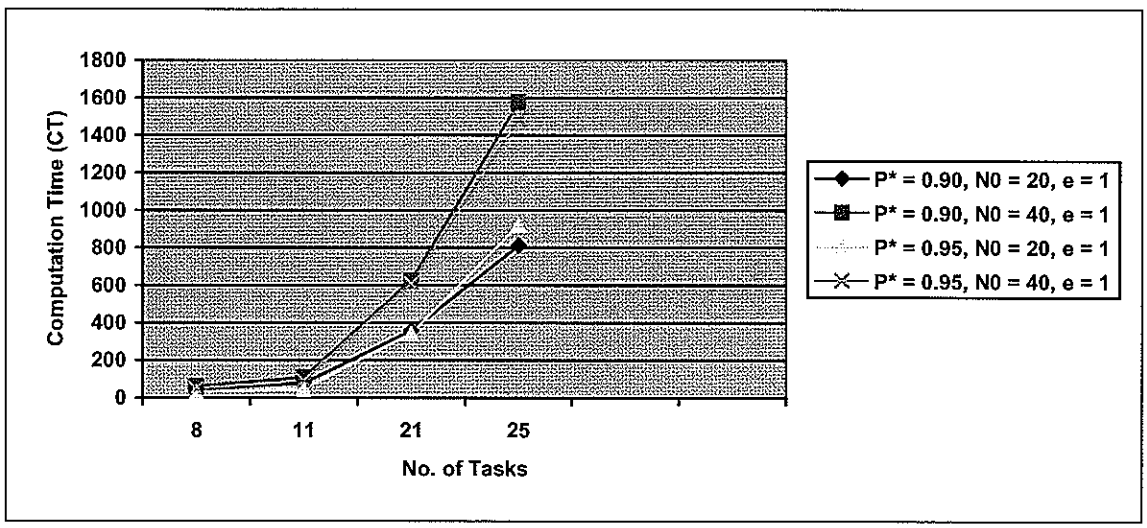

Figure13: Influence of No. of tasks on Computation time

The reason behind this is that as the number of tasks increases, the time taken to generate the random samples for each region also increases. This is due to the fact that a random sample is a solution in itself which requires assignment of tasks to

various stations. The time taken to assign tasks to stations is directly proportional to the number of tasks. Thus, if the number of tasks increases, the time taken to generate random samples for the different regions also increases.

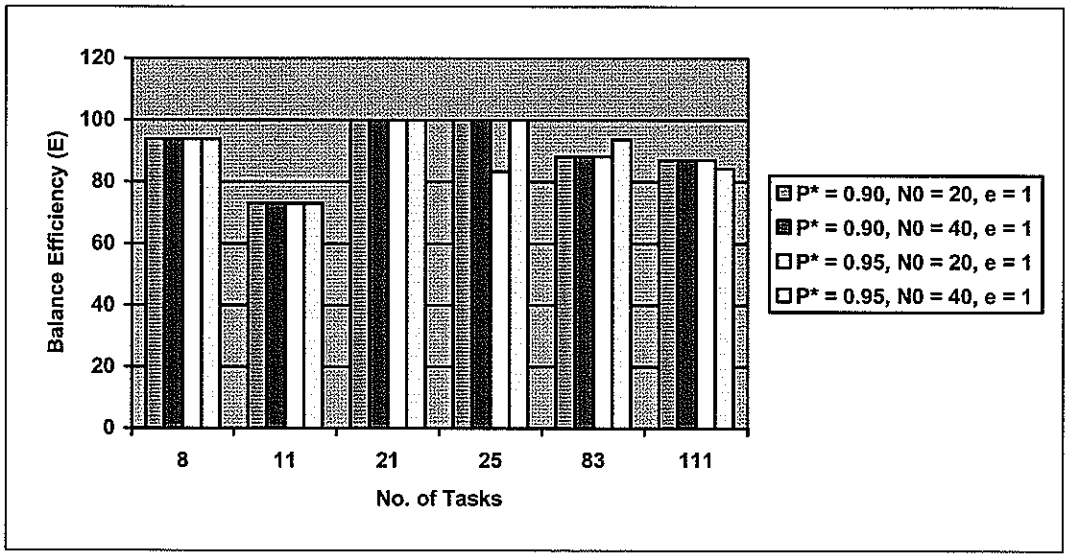

Figure14: Influence of No. of tasks on Balance Efficiency

The balance efficiencies are almost always same for all the four sets of parameters meaning that the solutions in all 4 cases are very close. The probable reason for this could be that we are considering a very high initial number of random samples in the iterations (more than what is actually required). The following data reveals this trend.

Data set	P^*	Cycle Time	Task ID	No. of Regions	Initial Sample Size	Variance	Constant Value (K)	Total No. Of Samples Required When indifference zone is equal to	
								1	0.5
Arcus1	0.95	5048	1	2	20	0.2631	2.453	2	8
Arcus1	0.95	5048	1	2	40	0.9743	2.386	6	24
Arcus1	0.95	5048	83	2	20	0.4210	2.453	3	12
Arcus1	0.95	5048	83	2	40	0.7692	2.386	5	20

Data set	P*	Cycle Time	Task ID	No. of Regions	Initial Sample Size	Variance	Constant Value (K)	Total No. Of Samples Required When indifference zone is equal to	
								1	0.5
Arcus2	0.90	10027	1	2	20	0.9473	1.896	4	16
Arcus2	0.90	10027	1	2	40	1.3846	1.852	5	20
Arcus2	0.90	10027	111	2	20	0.7894	1.896	3	12
Arcus2	0.90	10027	111	2	40	0.8717	1.852	3	12
Arcus2	0.95	10027	1	2	20	1.2631	2.453	8	32
Arcus2	0.95	10027	1	2	40	0.9743	2.386	6	24
Arcus2	0.95	10027	111	2	20	0.8947	2.453	6	24
Arcus2	0.95	10027	111	2	40	1.4871	2.386	9	36
Jackson	0.90	7	1	2	20	0.6315	1.896	3	12
Jackson	0.90	7	1	2	40	0.5897	1.852	3	12
Jackson	0.90	7	11	2	20	0.2105	1.896	1	4
Jackson	0.90	7	11	2	40	0.9230	1.852	4	16
Jackson	0.95	7	1	2	20	0.4736	2.453	3	12
Jackson	0.95	7	1	2	40	0.6153	2.386	4	16
Jackson	0.95	7	11	2	20	0.7894	2.453	5	20
Jackson	0.95	7	11	2	40	0.9487	2.386	6	24

Table5: No. of samples required for various regions

In many cases, it was also observed that for a given data set, if only the initial number of samples was varied, the quality of the solution improved marginally. The reason behind this is that the actual number of samples required (for a given indifference zone) was found out to be far less than the initial number of samples in each case (20 and 40). Hence, in both these cases, there was no need to consider any additional samples. We know that larger is the number of random samples, lesser is the

probability of getting a poorer solution. Hence, we end up getting a better solution in some test cases when the initial sample size is 40 as compared to 20.

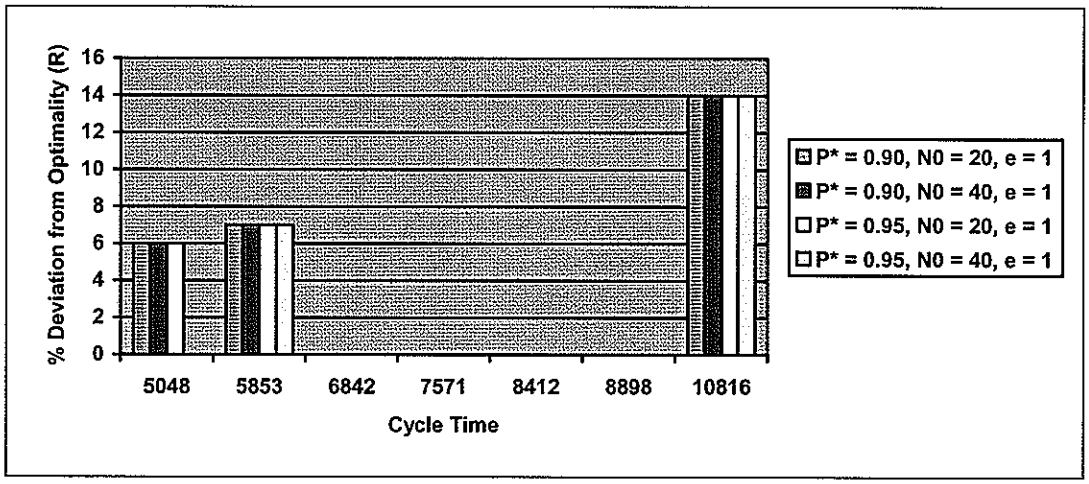

Figure15: Influence of Cycle time on % Deviation from optimality

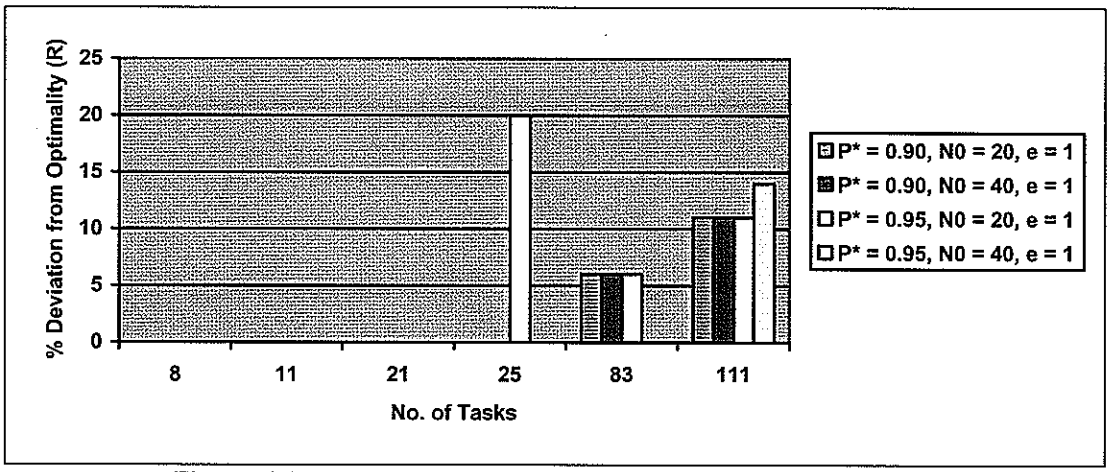

Figure16: Influence of No. of tasks on % Deviation from optimality

As is evident from the graph above, for smaller data sets, the solutions are almost always near optimal. However, for larger data sets, the solutions start deviating from the optimal value (although, the deviation is not very large).

6. Conclusions and further research

In this study, we applied the Nested Partitions method to a U-line balancing problem and conducted experiments to evaluate the application. From the results, it is quite evident that the Nested Partitions method provided near optimal solutions (optimal in some cases). Besides, the execution time is quite short as compared to the Branch and Bound algorithm. However, for larger data sets, the algorithm took significantly longer times for execution. One of the reasons could be the way in which the random samples are generated. In the present study, a random sample is a solution in itself which requires assignment of tasks to various stations. The time taken to assign tasks to stations is directly proportional to the number of tasks. Thus, if the number of tasks increases, the time taken to generate random samples for the different regions also increases.

The performance index for the Nested Partitions method in the present study was the number of stations in the random solutions (samples) generated. The total idle time for the samples can be used as another performance index. ULINO method is known to have used a combination of bounds to come up with good solutions. This approach of combining different performance indices can be used to evaluate the random samples and obtain even better solutions.

Here, we used deterministic time values for the tasks. In industries where majority of tasks are performed manually, the stochastic version of the problem could be of vital importance.

Experimenting with different objective functions (No. of stations was used in this study) could be of some significance to some industries where in the cost associated with creation of a new station is not the same. For such industries, the results obtained by using the present approach will not be of much value. Labor costs, task incompletion costs or a combination of those can be effectively used as alternate objective functions.

7. Bibliography

1. L. Shi and S. Olafsson, May-June 2000. Nested Partitions Method for Global Optimization, *Operations Research*, Volume 48, No. 3, 390-407
2. E. Erel, I. Sabuncuoglu and H. Sekerci, April 2005. Stochastic Assembly Line Balancing using Beam Search, *International Journal of Production Research*, Volume 43, No. 7, 1411-1426
3. A. Scholl and R. Klein, Fall 1997. SALOME: A Bidirectional Branch-and-Bound Procedure for Assembly Line Balancing, *INFORMS Journal on Computing*, Volume 9, No.4, 319-334
4. A. Scholl and R. Klein, 1999. ULINO: Optimally Balancing U-shaped JIT Assembly Lines, *International Journal of Production Research*, Volume 37, No. 4, 721-736
5. R. Johnson, February 1998. Optimally Balancing Large Assembly Lines with 'Fable', *Management Science*, Volume 34, No. 2, 240-253
6. T. Hoffmann, January 1992. Eureka: A Hybrid System for Assembly Line Balancing, *Management Science*, Volume 38, No. 1, 39-47
7. M. Pinedo and X. Chao, 1999. *Operations Scheduling with Applications in Manufacturing and Services*, Irwin/McGraw-Hill, International Editions 1999
8. S. Olafsson and N. Gopinath, 2000. Optimal Selection Probability in the Two-Stage Nested Partitions Method for Simulation-Based Optimization, *Proceedings of the 2000 Winter Simulation Conference.*
9. Y. Rinott, 1978. On two-stage selection procedures and related probability-in-equalities, *Communications in Statistics*, A7: 799-811
10. S. Olafsson and L. Shi, November 1998. A method for scheduling in parallel manufacturing systems with flexible resources, *IIE Transactions* 2000, Volume 32, No. 135-146.
11. A. Law and W. Kelton, *Simulation Modeling and Analysis*, Third edition, 2000. McGraw-Hill Publications, 576-577