Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp25-30)

Sampling-Based Tasks Scheduling in Dynamic Grid Environment

Kun Gao[1,2], Meiqun Liu[3], Kexiong Chen[2], Ning Zhou[3], Jiaxun Chen[1]
[1]Information Science and Technology College, Donghua University, P.R.C
[2]Aviation University of Air Force, P.R.C
[3]Administration of Radio Film and Television of Jilin Province, P.R.C

Abstract: -In this paper, we propose a new solution for data mining task scheduling in Grid environment. First, we propose a sample-based application run time evaluation. Then, we propose a cost model for predicting the data transfer time on Grid. Finally, according the priori estimation of the application response time and the data transfer time, we propose the method for tasks scheduling in grid environment.

Key-Words: - Sampling, Scheduling, Prediction, Grids

1 Introduction

Data mining is the process of autonomously extracting useful information or knowledge from large data stores or sets. Due to the distributed nature of input data, both computational and data intensive, the Grid is a natural platform for deploying a high performance data mining service. Data mining application is typical irregular problems [1], where patterns of computations and communications are unstructured and/or changing dynamically. The performance costs of many DM tools depend not only on the size of data, but also on the specific mining parameters provided by the user. Consider for example the Association Rule Mining (ARM) analysis: its complexity not only depends on the size of the input dataset, but also on the user-provided support and confidence thresholds. Moreover, the correlation between the items present in the various transactions of a dataset largely influences the number and the maximal length of the rules found by an ARM tool. Therefore, it becomes difficult to predict in advance either the computational and input/output costs, or the size of the output data.

In order to deal with these issues, we first propose to exploit sampling as a method to acquire preventive knowledge about the rough execution costs of DM jobs. In order to predicting the data transfer time, we then propose the cost model for data mining task on Grid. Finally, according the priori estimation of the application response time and the data transfer time, we propose the method for tasks scheduling in grid environment. The paper is organized as follows. Section 2 introduces some related works. Section 3 presents the method that evaluates data mining task run time. Section 4 presents the cost model for predicting the data transfer time. Section 5 presents the solution for scheduling data mining tasks on Grid. Finally concludes this paper.

2 Related Works

Early work in the parallel computing area proposed using similarity templates of application characteristics to identify similar tasks in a history. A similarity template is a set of attributes that we use to compare applications in order to determine if they are similar. Thus, for histories recorded from parallel computer workloads, one set of researchers selected the queue name as the characteristic to determine similarity [2]. They considered that applications assigned to the same queue were similar. In other work [3], researchers used several templates for the same history, including user, application name, number of nodes, and age.

Manually selecting similarity templates had the following limitations:

- Identifying the characteristics that best determine similarity isn't always possible.
- It's not generic: Although a particular set of characteristics might be appropriate for one domain, it's not always applicable to other domains.

In this paper, we develop a rough sets based technique to address the problem of automatically selecting characteristics that best define similarity in Grid environment.

3 sample-based application run time evaluation

3.1 Principle of Computation Times Estimation

Rough sets theory as a mathematical tool to deal with uncertainty in data provides us with a sound theoretical basis to determine the properties that define similarity. Rough sets operate entirely on the basis of the data that is available in the history and require no external additional information. The history represents an information system

in which the objects are the previous applications whose runtimes and other properties have been recorded. The attributes in the information system are these applications' properties. The decision attribute is the application runtime, and the other recorded properties constitute the condition attributes. This history model intuitively facilitates reasoning about the recorded properties so as to identify the dependency between the recorded attributes and the runtime. So, we can concretize similarity in terms of the condition attributes that are relevant and significant in determining the runtime. Thus, the set of attributes that have a strong dependency relation with the runtime can form a good similarity template. Having cast the problem of application runtime as a rough information system, we can examine the fundamental concepts that are applicable in determining the similarity template.

The objective of similarity templates in application runtime estimation is to identify a set of characteristics on the basis of which we can compare applications. We could try identical matching, i.e. if n characteristics are recorded in the history, two applications are similar if they are identical with respect to all n properties. However, this considerably limits our ability to find similar applications because not all recorded properties are necessarily relevant in determining the runtime. Such an approach could also lead to errors, as applications that have important similarities might be considered dissimilar even if they differed in a characteristic that had little bearing on the runtime.

A similarity template should consist of the most important set of attributes that determine the runtime without any superfluous attributes. A reduct consists of the minimal set of condition attributes that have the same discerning power as the entire information system. In other words, the similarity template is equivalent to a reduct that includes the most significant attributes. Finding a reduct is similar to feature selection problem. All reducts of a dataset can be found by constructing a kind of discernibility function from the dataset and simplifying it. Unfortunately, it has been shown that finding minimal reduct or all reducts are both NP-hard problems. Some heuristics algorithms have been proposed. Hu gave an algorithm using significant of attribute as heuristics [4]. Starzyk used strong equivalence to simplify discernibility function [5]. Some algorithms using genetic algorithm have been also proposed. However, there are no universal solutions. It's still an open problem in rough set theory.

Rough sets theory has highly suitable and appropriate constructs for identifying the properties that best define similarity for estimating application runtime. A similarity template must include attributes that significantly affect the runtime and eliminate those that don't. This ensures that the criteria with which we compare applications for similarity have a significant bearing on determining runtime. Consequently, applications that have the same characteristics with respect to these criteria will have similar runtimes.

In this paper, we propose a feature ranking mechanism which can be used in fast heuristic reduct computation. Based on the mechanism, a sampling method for finding approximate reduct is presented.

3.2 Approximate Reduct Algorithm

In this section, we first recall necessary rough set notions [6] used in this section, then introduces the feature ranking mechanism, and finally present the sampling approximate reduct algorithm.

3.3 Related Rough Set Concepts

Definition 1 (information system) An information system is an ordered pair $S=(U, A \cup \{d\})$, where U is a non-empty, finite set called the universe, A is a non-empty, finite set of conditional attributes, d is a decision attribute. $A \cap \{d\} = \Phi$. The elements of the universe are called objects or instances.

Information system contains knowledge about a set of objects in term of a predefined set of attributes. The set of objects is called concept in rough set theory. In order to represent or approximate these concepts, an equivalence relation is defined. The equivalence classes of the equivalence relation, which are the minimal blocks of the information system, can be used to approximate these concepts.

Definition 2(Indiscernibility relation) Let $S=(U,A \cup \{d\})$ be an information system, every subset $B \subseteq A$ defines an equivalence relation $IND(B)$,called an indiscernibility relation,defined as $IND(B)=\{(x,y) \in U \times U: a(x)=a(y)$ for every $a \in B\}$.

Definition 3 (Positive region) Given an information system $S= (U,A \cup \{d\})$, let $X \subseteq U$ be a set of objects and $B \subseteq A$ a selected set of attributes. The lower approximation of X with respect to B is $B_*(X)=\{x \in U:[x]_B \subseteq X\}$. The upper approximation of X with respect to B is $B^*(X)=\{x \in U: [x]_B \cap X \neq \Phi\}$. The positive region of decision d with respect to B is $POS_B(d)= \cup \{B_*(X):X \in U/IND(d)\}$

The positive region of decision attribute with respect to B represents approximate quantity of B. Not all attributes are necessary while preserving the approximate quantity of the original information system. Reduct is the minimal set of attribute preserving approximate quantity.

Definition 4 (Reduct) An attribute a is dispensable in $B \subseteq A$ if $POS_B(d)= POS_{B-\{a\}}(d)$. A reduct of B is a set of attributes $B' \subseteq B$ such that all attributes $a \in B-B'$ are dispensable, and $POS_B(d)= POS_{B'}(d)$.

There are usually many reducts in an information system. In fact, one can show that the number of reducts of an information system may be up to $C^{|A|/2}_{|A|}$. In order to find reducts, discernibility matrix and discernibility function are introduced.

Definition 5 (discernibility matrix) The discernibility matrix of an information system is a symmetric $|U| \times |U|$ matrix with entries c_{ij} defined as $\{a \in A | a(x_i) \neq a(x_j)\}$ if $d(x_i) \neq d(x_j)$, Φ otherwise. A discernibility function can be constructed from discernibility matrix by or-ing all

attributes in c_{ij} and then and-ing all of them together. After simplifying the discernibility function using absorption law, the set of all prime implicants determines the set of all reducts of the information system.

3.4 The feature ranking mechanism

In a discernibility matrix, every entry represents a set of attributes discerning two objects. As we know, if an entry consists of only one attribute, the unique attribute must be a member of core, i.e., it has higher significance. By generalizing the idea, we recognize that shorter entry is more significant than longer ones. To make the difference computable, a weight $w(a)$ is assigned to every attribute a. The weight is initialized to zero and recomputed according the following formula when a new entry C is arrived:

$$w(a_i)=w(a_i)+|A|/|C| \ a_i \in C$$

where $|A|$ is the cardinality of attribute set A of the information system. The formula embodies the following idea:

- The more times an attribute appears in the discernibility, the more important the attribute might be.
- The shorter the entry is, the more important the attributes in the entry might be.

For example, let $w(a_1)=3$, $w(a_3)=4$, the system has 10 attributes in total, and the new entry is $\{a_1,a_3\}$. Then weights after this entry can be computed: $w(a_1)=3+10/2=8$; $w(a_3)=4+10/2=9$.

3.5 Sampling approximate reduct Algorithm

The approximate reduct algorithm is based on the feature ranking mechanism and rough set concept. Generally, constructing a discernibility matrix requires a lot of memory and $O(|A||U|^2)$ time. In the worst case, a discernibility matrix can have up to $|U|(|U|-1)/2$ entries and thus take up huge amount of memory. When a dataset becomes so large that it cannot fit in memory, computing discernibility matrix is very difficult because lots of disk I/O are needed. And the computing time becomes unbearable.

Sampling is an efficient way for large datasets. Due to the nature of sampling, it is impossible to find real reduct. However, approximate reduct with low error can be found rapidly. As we know, approximate reducts can be very useful in classification, for they are typically robust than regular reducts.

Sampling can find satisfactory approximate reduct in short time. However, how to do it efficiently? Our method consists of two phases. Phase I is called generating phase. In this phase, we select several samples of original dataset and count frequency of every attribute using discernibility matrix. These frequencies are sorted for later use. Phase II is called testing phase. In this phase, more samples are used together with frequencies counted in phase I to produce a good approximate reduct. Fig.1 presents the sampling approximate reduct algorithm written in pseudo-code. The algorithm is designed according to the principle given in previous subsection.

In phase I it seems natural to generate all reducts for one sample, and then gather most stable one. For large set of attributes, even a small number of instances, computing all reducts is a computationally impossible task.

We compute weight of attributes using the feature ranking method in the process of generating discernibility matrix for every sample. After all samples are processed, weights of every attribute are summed. A sorting procedure is then employed on the weights. This ends Phase I.

After phase I, an ordered list of attributes is obtained. The goal of phase II is to determine which attribute should be included in the reduct. More samples are sampled to do this job. First, discernibility matrix of a sample is computed. The attribute list is tested against the matrix. If result is a super-reduct (i.e., intersection with every entry of the matrix are not empty), the attribute with lowest weight is dropped from the list. Above process is repeated until the test fails. Last non-failed attribute list is referred as candidate reduct.

The candidate reduct is then tested against another sample. If the test fails, the candidate reduct (i.e., the attribute list) falls back by one attribute. The process is repeated until candidate reduct passes the test. Repeat the process to rest of the samples. At last we get an approximate reduct that passes test for all testing samples.

This phase ensure that the resulting approximate reduct has enough supports.

Input: sample times γ, sample rate ε, information system $S=(U, A \cup \{d\})$;
Output: an attribute set *Red*;
Phase I: ranking the attribute by sampling.
Red=Φ, AttrList=Φ, $w(a_i)=0$, $weight(a_i)=0$, for i=1,…|A|.
for k=1 to γ do
  S'=sample(S, ε);
  Generating discernibility matrix for S' and computing $weight(a_i)$ for every $a_i \in A$
  $w(a_i)= w(a_i)+weight(a_i)$ for i=1,..,n.
endfor
AttrList=Sorting A according to $w(a_i)$;
Phase II: test the ranked attribute list by sampling.
S'=Sample(S, ε);
Generating discernibility matrix M for S'.
While (!Test(AttrList, M)) remove the attribute with lowest weight from AttrList;
  Add the last removed attribute back to AttrList;
  for k=2 to γ do
    S'=Sample(S, ε);
    Generating discernbility matrix M for S'.
    While (!Test(AttrList, M)) Add back the last removed attribute back to AttrList;
  endfor
Red=AttrList;
return Red

Figure 1 Sampling for approximate reduct algorithm

Sample(*S*, ε) samples S with sampling rate ε. *w(a)* sums up frequency computing by *weight(a)* which is explained before. Test(*AttrList*, *M*) tests whether *AttrList*

is a super-reduct of discernibility matrix *M*. If so, it returns true.

Using the reduct, We can predict the data mining application run time. Detail information see[7,8].

4 Cost Model

In the following cost model we assume that each input dataset is initially stored on at least a single machine mh, while the knowledge model extracted must be moved to a machine mk. Due to decisions taken by the scheduler, datasets may be replicated onto other machines, or partitioned among the machines composing a cluster.

Sequential execution. Dataset D_i is stored on a single machine m_h. Task t_i is sequentially executed on machine m_j, and its execution time is e_{ij}. The knowledge model extracted $|a_i(D_i)|$ must be returned to machine m_k. We have to consider the communications needed to move D_i from m_h to m_j , and those to move the results to m_k. Of course, the relative communication costs involved in dataset movements are zeroed if either h = j or j = k. The total execution time is thus:

$$E_{ij} = |D_i| / b_{hj} + e_{ij} + |\alpha_i(D_i)| / b_{jk}$$

Parallel execution. Task t_i is executed in parallel on a cluster c_{lJ} , with an execution time of e_{iJ}. In general, we have also to consider the communications needed to move and partition D_i from machine m_h to cluster c_{lJ} , and to return the results $|a_i(D_i)|$ to machine m_k. Of course, the relative communication costs are zeroed if the dataset is already distributed, and is allocated on the machines of c_{lJ}. The total execution time is thus:

$$E_{iJ} = \sum_{m_t^J \in cl_J} \frac{|D_i| / |cl_J|}{b_{ht}} + e_{iJ} + \sum_{m_t^J \in cl_J} \frac{|\alpha_i(D_i)| / |cl_J|}{b_{tk}}$$

Finally, consider that the parallel algorithm we are considering requires coallocation and coscheduling of all the machines of the cluster. A different model of performance should be used if we adopted a more asynchronous distributed DM algorithm, where first independent computations are performed on distinct dataset partitions, and then the various results of distributed mining analysis are collected and combined to obtain the final results.

To optimize scheduling, our mapper has to forecast the completion time of tasks. To this end, the mapper has also to consider the tasks that were previously scheduled, and that are still queued or running. Therefore, in the following we analyze the actual completion time of a task for the sequential case. A similar analysis could be done for the parallel case. Let C_{ij} be the wall-clock time at which all communications and sequential computation involved in the execution of t_i on machine m_j complete. To derive C_{ij} we need to define the starting times of communications and computation on the basis of the ready times of interconnection links and machines. Let s_{hj} be the starting time of the communication needed to move D_i from m_h to m_j , s_j the starting time of the sequential execution of task t_i on m_j, finally, s_{jk} the starting time of the communication needed to move $a_i(D_i)$ from m_j to m_k. From the above definitions:

$$C_{ij} = (s_{hj} + \frac{|D_i|}{b_{hj}}) + \delta_1 + e_{ij} + \delta_2 + \frac{|\alpha_i(D_i)|}{b_{jk}} = s_{hj} + E_{hj} + \delta_1 + \delta_2$$

Where $d_1 = s_j - (s_{hj} + \frac{|D_i|}{b_{hj}}) \geq 0, d_2 = s_{jk} - (s_j + e_{ij}) \geq 0$

If $m_{\bar{j}}$ is the specific machine chosen by our scheduling algorithm for executing a task t_i, where T is the set of all the tasks to be scheduled, we define $C_i = C_{\bar{ij}}$. The makespan for the complete scheduling is thus defined as $\max_{t_i \Xi T}(C_i)$, and its minimization roughly corresponds to the maximization of the system thoughput.

5 task scheduling

The cost model represent the response time for mining a dataset at different available servers. The cost formula for a given server is dependent on the location of the server (i.e. whether the server is located at the client's site or at the service provider's site), since this determines the adopted mining strategy , the location of the dataset, the estimated computation time at the server and the waiting time at the server. Thus, having developed the estimates and the cost matrix for DDM tasks, the question of task allocation for minimising the response time becomes a Generalised Assignment Problem (GAP)[9], which is a well-known class of NP-complete problems.

Generalised Assignments Problems have been widely studied in the literature and are described as the task of optimally assigning n tasks to m processors (or n jobs to m agents). Given the profit p_{ij} and the amount of resource w_{ij} for the assignment of task j to processor i and the total resource ci available for each processor I, the objective is to assign the tasks such that the total profit P is maximised as follows:

$$\text{Maximise } P = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \qquad (1)$$

The above maximisation is subject to the following constraints:

1. $\sum_{i=1}^{m} w_{ij}x_{ij} \leq c_i, i \in M = \{1,2,...,m\}$
2. $\sum_{i=1}^{m} x_{ij} = 1, j \in N = \{1,2,...,n\}$
3. $x_{ij} = 0$ or $1, i \in M, j \in N$ where

$$x_{ij} = \begin{cases} 1 & \text{if task j is assigned to i} \\ 0 & \text{otherwise} \end{cases}$$

The first constraint specifies that the amount of resource required for assigning a task to a processor must be less than the total resource available for that processor, the second constraint specifies that a task is assigned to exactly one processor and the third constraint specifies the values of x_{ij}. The minimisation version of the problem is expressed in terms of the cost c_{ij} of assigning a task j to a processor i and the objective is to minimise the overall cost C as follows:

$$Minimise\ C = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

The minimisation process is subject to the same constraints as the maximisation process. Generalised Assignment Problems are known to be NP-complete and do not necessarily have a feasible solution owing to constraint (2) above. However, it has been shown in [9] that an equivalent and feasible solution is possible by relaxing or eliminating this constraint. Several exact and approximate algorithms have been proposed in the literature for GAP problems including [10]. Exact solutions involve variations of branch-and-bound algorithms [9]. We note that given the typical size of a cost matrix for a DDM task in grid environment is not a high-dimensional matrix, it does not preclude exact algorithms. However, to preserve generality to cope with larger matrices we have adopted an approximate solution, which is good enough and is not as computationally expensive as an exact solution.

In this paper, we apply an approximate technique that has two phases to determine the task allocation strategy from the cost model. We first use the polynomial-time algorithm proposed by Martello and Toth [10] to determine the initial allocation of tasks and then apply a Tabu search to improve the initial allocation by exchanges. In the following discussion, we formally represent the requirement of determining the minimum response time strategy from the cost matrices in GAP terms and present our application of an initial allocation search followed by a local search to identify the appropriate task allocation strategy.

Given a mxn cost matrix CM, the elements $cm_{ij} \in CM$, where i=1, 2, …, m and j=1, 2, …, n represent the composite response time (having considered both the communication and computation times) for mining a dataset originally located at server S_j at server S_i. Therefore in GAP terms the servers represent "processors" and the datasets represent "tasks". The "cost" of processing a task (i.e. the dataset) j at a processor (i.e. the server) i is represented as the element cm_{ij} in the cost matrix CM. The cost matrix enumerates this cost with respect to all the processors that are available including those at the client's site and at the service provider's site. The selection of the minimum response strategy from cost matrix can be expressed using equation 1.

We relax the constraint $\sum_{i=1}^{m} w_{ij}x_{ij} \leq c_i, i \in M = \{1,2,...,m\}$ that is imposed on traditional GAP problems, since the variables represented as "amount of resource required for assigning a task to a processor" and the "total resource available for that processor" are not relevant in our computation. The a priori estimates of the response time form the "cost" in the cost models, thereby eliminating the need to include separate "resources required" and "total resources available" criteria into the equation. For example, the cost elements specify the estimated time to process a task at a server. The case of the server being currently unavailable is modelled using the "wait time" variable (W), which in turn is derived from the estimates for the tasks that have been currently assigned to that processor. The question is how to determine the minimum time to perform the requested task given the current computational resources that are available.

Having formulated our task allocation problem in GAP terms, we now discuss our approach to determining the minimum response time strategy from the cost matrix. We first apply the polynomial-time, approximate algorithm developed by [10] to perform an initial allocation of datasets to servers. For the sake of completeness we include an overview of the algorithm [10] adopted in this dissertation. The algorithm operates by first computing a measure of desirability (f_{ij}) for assigning a dataset j to server i, where j=1, 2, …, n and i=1, 2, …, m and uses this measure to determine the initial allocations. The steps of the algorithm are as follows:

1. Let f_{ij} be the measure of desirability for assigning dataset j to server i, where j=1, 2, …, n and i=1, 2, …, m.

2. Since the the cost $cm_{ij} \in CM$ represent the estimated response times and the objective is to minimise the response time, the desirability of

allocating a dataset to a server increases as the estimated response time decreases. It is evident that there is an inverse relationship between the cost elements of the matrix and the desirability factor. Therefore, we compute the desirability function for the elements of the cost matrix as $f_{ij} = 1/cm_{ij}$ (note: $cm_{ij} >= 0$, since the estimated response time can never be zero or negative).

3. The allocation is performed by iteratively considering all the unassigned datasets, and determining the dataset j* that has the maximum difference between the largest f_{ij} and the second largest f_{ij} ($i \Xi \{1, 2, .., m$).

4. The dataset j* is then assigned to the server for which f_{ij*} is maximum.

At the end of the initial task allocation phase, each dataset is assigned to a server that is optimal in that it has the minimum response time for that dataset. However, at this stage the optimisation is local to that dataset and this solution may not represent a global minimisation of response time. For example, two or more datasets might be assigned to a particular server since that provides the best response time for those datasets. While individually each dataset has been assigned to the server that provides the minimum response time, it is possible that having the datasets in question assigned to the same server does not result in the overall minimisation of the response time. Therefore, in some instances a sub-optimal allocation for individual datasets may lead to further reduction in the overall response time. The second phase of the task allocation strategy applies a Tabu search to improve the initial task allocation through local exchanges. The Tabu search that we use belongs to the class of strict Tabu searches known as Reverse Elimination Tabu searches [11].

6 conclusion

In summary, we have developed reduct algorithm and a cost model for a priori estimation of the communication and computation response time in grid environment. A sampling method for finding approximate reduct is presented. The approximate reduct algorithm is based on a feature ranking mechanism. We have discussed the task allocation strategy for minimising the response time using the estimates. The estimation technique presented in this paper is generic and can be applied to others optimization problems.

References:

[1] J. Darlington, M. Ghanem, Y. Guo, and H. W. To. Performance models for co-ordinating parallel data classification. In Proc. of the Seventh International Parallel Computing Workshop, 1997.

[2] A.B. Downey , "Predicting Queue Times on Space-Sharing Parallel Computers,"*Proc. 11th Int'l ParallelProcessing Symp.* (IPPS 97), IEEE CS Press, 1997.

[3] R. Gibbons , "A Historical Application Profiler for Use by Parallel Schedulers,"*Job Scheduling Strategies for Parallel Processing ,* LNCS 1291, Springer-Verlag, 1997.

[4] X.Hu, Knowledge discovery in databases: An attribute-oriented rough set approach, Ph.D thesis, Regina university, 1995.

[5] J.Starzyk, D.E.Nelson, K.Sturtz, Reduct generation in information systems, Bulletin of international rough set society, volume 3, 1998.

[6] S.K.Pal, A.Skowron, Rough Fuzzy Hybridization-A new trend in decision-making, Springer, 1999.

[7] Kun Gao, Kexiong Chen, Meiqun Liu, Jiaxun Chen, Rough Set Based Data Mining Tasks Scheduling on Knowledge Grid, Lecture Notes in Computer Science, Volume 3528, May 2005, Pages 150 - 155

[8] Kun Gao, Youquan Ji, Meiqun Liu, Jiaxun Chen, Rough Set Based Computation Times Estimation on Knowledge Grid, Lecture Notes in Computer Science, Volume 3470, July 2005, Pages 557 – 566.

[9] Martello, S., and Toth, P., (1990), "Knapsack Problems – Algorithms and Computer Implementations", John Wiley and Sons Ltd, England, UK.

[10] Martello, S., and Toth, P., (1981), "An Algorithm for the Generalised Assignment Problem", Operational Research'81, Amsterdam, pp. 589-603.

[11] Glover, F., (1990), "*Tabu Search – Part II*", ORSA Journal on Computing, Vol. 2, No. 1, pp. 4-32.