A Parallel Branch-and-Bound Method for Cluster Analysis

By: Lakshmi S. Iyer and J.E. Aronson.

Iyer, L.S. and J.E. Aronson. (1999) "A Parallel Branch-and-Bound Method for Cluster Analysis," *Annals of Operations Research* Vol. 90, 65-86.

Abstract:

Cluster analysis is a generic term coined for procedures that are used objectively to group entities based on their similarities and differences. The primary objective of these procedures is to group n items into K mutually exclusive clusters so that items within each cluster are relatively homogeneous in nature while the clusters themselves are distinct. In this research, we have developed, implemented and tested an asynchronous, dynamic parallel branchand-bound algorithm to solve the clustering problem. In the developmental environment, several processes (tasks) work independently on various subproblems generated by the branch-and-bound procedure. This parallel algorithm can solve very large-scale, optimal clustering problems in a reasonable amount of wall-clock time. Linear and superlinear speedups are obtained. Thus, solutions to real-world, complex clustering problems, which could not be solved due to the lack of efficient parallel algorithms, can now be attempted.

Keywords: integer programming | combinatorial optimization | parallel optimization | cluster analysis

Article:

1. Introduction

The primary aim of parallel processing is to solve very large problems in less time. One of the major concerns of the field of combinatorial optimization is to solve real-world problems (typically large in size) efficiently and in less time. Technological improvements have stimulated the application and design of parallel optimization algorithms. Clustering problems are highly combinatorial in nature. Computation times increase tremendously with slight increases in problem size. Often heuristics are applied to solve these problems because, in attempting to find an optimal solution, a mathematical programming model of even small problems becomes a large NP-complete combinatorial problem.

Here, we describe the development, implementation and computational testing of a parallel cluster analysis algorithm[1]). Our primary motivation is the need for an efficient algorithm that is fast and cost-effective to solve cluster analysis problems. In addition to this, we are interested in developing a general parallel branch-and-bound procedure that can be applied to cluster analyses. The use of parallel processing can not only speed up the solution process, but also increase the problem size for which an optimum can be found in a “reasonable” time frame. Parallel processing is an appropriate strategy for branch-and-bound algorithms as different branches of the tree can be independently evaluated by different processors. In the field of combinatorial optimization, parallel branch-and-bound algorithms have been found to yield solutions faster than serial methods.

Various systems are available to facilitate the development and implementation of parallel algorithms. The Parallel Virtual Machine (PVM) is one such system that helps link heterogenous computers and support the communication between various processors [33,34]. In general, PVM can start more than once process (task) on any processor and also helps communication and synchronization among the processes.

We have developed a dynamic, asynchronous parallel branch-and-bound algorithm to solve cluster analysis problems on a PVM platform in the Unix environment. Generating subproblems dynamically as opposed to statically has been found to yield not only better load balancing, but also increased processor utilization. Since some parts of the branch-and-bound tree are fathomed sooner than others, it would be difficult to use synchronized communication among processes. Thus, the asynchronous communication mode is more appropriate for our method, which ensures that all processors are active most of the time. Past research also supports the use of asynchronous communication [4,14,26]. We expect our parallel algorithm to have a significant impact on the solvability of large-scale clustering problems. Thus, solutions to real-world, complex clustering problems, which could not be solved due to a lack of efficient parallel algorithms, can now be attempted.

We next present a brief description of the optimal cluster analysis problem of interest, along with its purpose and applications. We then describe the mathematical model of the optimal cluster analysis model and algorithm on which the solution methodology of the parallel algorithm is based. Finally, we provide a detailed description of the parallel method itself, implementation issues, computational results and conclusions.

2. Cluster analysis

Classifying objects is an inherent property of human conceptual activity. It is also a fundamental process in any scientific inquiry. The technique of cluster analysis has been applied for these purposes for many years. Most of the early applications were in biology, where the term taxonomy is more commonly used. Until the widespread use of computers, the field of clustering was more of an art than a scientific endeavor. The complexity of clustering methods is known to increase tremendously with an increase in problem size. However, radical advances in computer technology allow us to handle the many complex computations necessary for applying clustering techniques. These have not only lead to increased applications, but also to advances in solution methodologies.

Cluster analysis is a general term for describing procedures that are used to classify a set of items (objects, entities, or elements) into a specified number of groups (clusters). Clustering techniques have been used for various purposes, some of which are data exploration model fitting, hypothesis testing, data reduction, prediction, and pattern recognition. Applications of cluster analysis can be found in various disciplines, such as artificial intelligence, botany, economics, flexible manufacturing systems, information systems design, marketing, psychology, sociology, and zoology.

There is an abundance of applications of cluster analysis to real-world problems. Some interesting examples include analyzing large engineering records collections [18], measuring welfare and quality of life across countries [16], management of cutting tools in flexible manufacturing systems [8], group cell formation in manufacturing [24], clustering as a quality management tool [32], identifying the structure and content of human decision making [1], mapping consumers' cognitive structures [17], information systems design [2,20,21,23], grouping executive information benefits [19], seating guests at a wedding [15], vehicle routing, production scheduling and sampling [30], and income tax bracket determination [27].

In the past couple of decades, both mathematicians and statisticians have been developing formal models to solve clustering problems. However, since cluster analysis is well established in statistics, very few mathematical programming approaches have been applied. But, most statistical methods only attempt to lower the sum of interactions between each observation and its group's mean or median. In contrast, mathematical programming techniques can also account for total group interaction (pairwise and independent of a mean or median), group size and "weight" limitations, and precedence relations. These factors are critical in such applications as information systems design, vehicle routing, production scheduling and others.

Although there are several methods to solve clustering problems, there is no unique classification of the solution methods. This is considered to be a pitfall in this field. However, due to the works of Cormack [7], Punj and Stewart [28] and Everitt [11], clustering techniques are broadly classified as hierarchical, optimization, density search, clumping and other techniques. Optimization techniques allow "relocation" of items during the clustering process, improving from an initial solution to optimality. The number of clusters must be decided a priori, although some methods allow for changes during analysis. Differences in optimization techniques exist due to the different methods used to obtain a starting solution and different clustering criteria used for obtaining an optimal solution [11]. Except for optimization methods, all the other solution methods listed above do not guarantee an optimal solution.

Mulvey and Crowder [27] use a subgradient method coupled with a simple search procedure for solving the clustering problem. However, their method does not yield an exact optimum. Although heuristics seem to be efficient for solving very large problems, the need to obtain optimal solutions for problems such as designing effective information systems [21,22] make heuristics less attractive. Klein and Aronson [21] have developed a mixed-integer programming model to find an optimal solution for clustering problems. Their method is based on the implicit enumeration method of Balas [2]. The next section provides the details of their mathematical model and serial solution methodology.

3. The optimal cluster analysis model and serial algorithm

The MIP model (GROUPS) for cluster analysis describes the grouping of n items into K mutually exclusive groups. There is an interaction cost associated with any two items belonging to the same group. This model tries to minimize the total sum of all pairwise interaction among the items in each group [2,21]. It also accounts for precedence relations and group size and weight limits. Depending on the application type, a data matrix represents either a similarity or dissimilarity (distance) measure of the items to be clustered. Since the objective of the MIP model is a minimizing function, the data matrix consists of dissimilarity measures between pairs of items. The following notation is used in describing the model:

dij the positive distance (or dissimilarity measure) between item i and item j,
Xik a zero–one variable representing whether item i is in group k (=1) or not (=0),
Yij a zero–one variable representing whether item i and item j are in the same group (=1) or not (=0),
K the desired number of groups, and
n the number of items.

In this paper, the terms items, entities, or elements are used interchangeably. Also, the terms groups or clusters are used interchangeably, unless otherwise indicated. The mathematical formulation for the cluster analysis problems may be stated as

(GROUPS)

Minimize $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} Y_{ij}$

subject to $Y_{ij} \geq X_{ik} + X_{jk} - 1$, for each *(i, j)* pair $\in$ each group

$$\text{i.e., for } i = 1,\ldots,n-1,\ j = i+1,\ldots,n,\ k = 1,\ldots,K, \quad (2)$$

$$\sum_{k=1}^{K} X_{ik} = 1 \qquad \text{for each item } i = 1,\ldots,n, \quad (3)$$

$$X_{ik} = 0,1, \qquad \text{for } i = 1,\ldots,n,\ k = 1,\ldots,K, \quad (4)$$

$$Y_{ij} \geq 0, \qquad \text{for } i = 1,\ldots,n-1,\ j = i+1,\ldots,n. \quad (5)$$

The objective function (1) minimizes all pairs of distances within groups. The constraint set (2) relates group membership to the objective function. Equation (3) represents the constraint set which ensures the assignment of each item to one group only. As the objective function drives the Yij to its minimum or maximum, integrality conditions in constraint set (5) are unnecessary. Further, since dij is positive, the upper bound of 1 for Yij can be omitted. When all distances are strictly positive, the optimal number of groups K* = K; however, if any distances are negative, then we can obtain K* < K. To handle negative distances, two set of constraints are added to the model, along with a modification to constraint set (5). A continuous variable Zij is added to the objective function to prevent the problem from being unbounded. The two new constraints and modified (5) are

$$
\begin{array}{lll}
Z_{ij} \geq (X_{ik} + X_{jk})/2, & \text{for each } (i, j) \text{ pair } \in \text{each group}, & \\
 & \text{i.e., for } i = 1, \ldots, n-1,\ j = i+1, \ldots, n,\ k = 1, \ldots, K, & (6) \\
Y_{ij} \leq Z_{ij}, & \text{for } i = 1, \ldots, n-1,\ j = i+1, \ldots, n, & (7) \\
Y_{ij} = 0, 1, & \text{for } i = 1, \ldots n-1,\ j + i + 1, \ldots, n. & (8)
\end{array}
$$

Hence, the objective (1) is now changed to

$$\text{Minimize} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (d_{ij} Y_{ij} + M Z_{ij}), \qquad (9)$$

where M is a large number. The Yij variables, which were formerly continuous, are now restricted to be 0–1. When the MIP model is now solved, the optimal number of groups K* , 1 £ K * £ K, will automatically be chosen.

An inherent problem in using a mathematical programming approach to solve the cluster analysis problem is its combinatorial nature. For any set of K optimal clusters, K! equivalent solutions need to be evaluated. To facilitate the pruning of nodes in the branch-and-bound tree, implicit constraints are added to reduce the number of nodes to be evaluated dramatically. The Level 1 implicit constraints are

$$\sum_{k=1}^{i} X_{ik} = 1, \quad \text{for } i = 1, \ldots, K-1. \qquad (10)$$

These constraints replace the first K – 1 constraints in (3). Basically, item 1 is placed in group 1; item 2 can then be placed in group 1 or 2; item 3 in group 1, 2 or 3; and so on up to item K – 1.

More restrictions are added to the model by using Level 2 implicit constraints. That is, if both items 1 and 2 are placed in group 1, then item 3 need not be placed in group 3, item 4 in group 4, etc., up to item K. These constraints are expressed as

$$X_{11} + X_{21} + X_{ii} \leq 2, \quad \text{for } i = 1, \ldots, K. \qquad (11)$$

These constraints are only valid for the case of no precedence or group limits. They dramatically reduce the subproblems evaluated in the tree. In their test results, Aronson and Klein [2] indicate that the implicit constraints reduced computation time by 80% by automatically pruning the search tree.

The solution algorithm developed for this model by Aronson and Klein [2] utilizes multiple branches from each node in the tree. Basically, the enumeration algorithm explores all combinations of the binary variables to obtain an optimal solution. In combinatorial problems, it would be computationally expensive to completely search the branch-and-bound tree. Hence, to prune the branches of the tree which may not yield a better incumbent solution, various fathoming strategies are used. Apart from implicit constraints, Aronson and Klein [2] compute lower bounds (Bp) on the interactions of the remaining observations to be classified to fathom nodes.

The serial solution algorithm uses a depth-first search strategy. In the branchand-bound tree structure, the depth of the tree represents the observation under consideration, while the branches represent group membership. Thus, for the clustering problem, in contrast to binary branching as in Balas's algorithm [3], we have multiple branching, where the maximum number of possible branches at each node is K, the number of groups (i.e., each branch at depth p represents an assignment to a unique group for item p). The depth of the tree is at most n, the number of items to be clustered. A sample tree structure for grouping 3 items into 2 clusters is shown in figure 1.

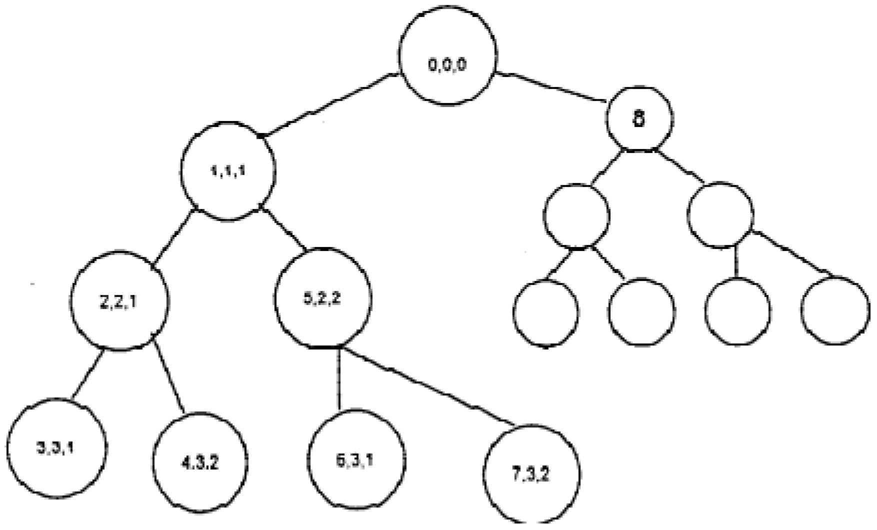


Figure 1. Tree structure for grouping 3 items into 2 groups. In each node are the node number, its depth in the tree (= item number), and the group membership assignment. Small nodes are implicitly enumerated (i.e., automatically fathomed).

Next, we present some additional notation used to describe the serial algorithm:

p : the current depth in the tree,
m : group membership indicator (from 1 to K),
B_p : bound on unassigned observations at depth p, n(m) : the number of observations in group m,
q : the number of groups with no membership,
SDIJ(p) : the objective value at node p,
Z : the objective value of the incumbent, and
X_{il} : the incumbent solution values of all X_{ik}.

The serial clustering algorithm (GROUPS)

The serial clustering algorithm (GROUPS)

```
Step 0. Initialization. $p = 0$. $n(m) = 0$ for $m = 1,\dots,K$, $q = K$, $Z = \infty$. $X_{ik} = 0$ for all
        $i$, $k$. $Y_{ij} = 0$ for all $i, j$. $index$ = TRUE.
Step 1. While ($index$)
        { /* Increment search depth, i.e., branch left */.
        Set $p = p + 1$. $m = 1$. $n(m) = n(m) + 1$. If $n(m) = 1$, set $q = q - 1$. $X_{pm} = 1$.
        $Y_{pj} = 1$ and $Y_{jp} = 1$ for all $j$ where $X_{jm} = 1$.
        /* Reasonability test, i.e., check if the number of items left to be assigned is
        less than number of groups with no membership. */
        While ($n - p \geq q$)
            { IF $Z > SDIJ(p)$ /* Modification of original serial method. Check objec-
            tive value before bound calculation */
                {calculate $B_p$. IF $Z > SDIJ(p) + B_p$
                    { IF $p \neq n$ EXIT LOOP
                    set $X_{Ik} = X_{ik}$ for all $i$, $k$; and set $Z = SDIJ(p)$}
            }
        While ($m = K$ or $n(m) = 1$ and $n(m + 1) = 0$)
        { /* Fathom: depth retraction. */
            $X_{pm} = 0$. $Y_{pj} = 0$ and $Y_{jp} = 0$ for all $j$ where $X_{jm} = 1$. $n(m) = n(m) - 1$.
            $p = p - 1$.
            IF ($n(m) = 0$) set $q = q + 1$. $m = j$ where $X_{pj} = 1$.
            IF $p \leq 0$, set $index$ = FALSE. EXIT LOOP /* optimal solution found */
        }
            /* Fathom: branch right on group. */
            set $X_{pm} = 0$. $Y_{pj} = 0$ and $Y_{jp} = 0$ for all $j$, where $X_{jm} = 1$. $n(m) = n(m) - 1$.
            $m = m + 1$. $n(m) = n(m) + 1$. If $n(m) = 1$, set $q = q - 1$. Set $X_{pm} = 1$. $Y_{pj} = 1$
            and $Y_{jp} = 1$ for all $j$ where $X_{jm} = 1$}
```

End of the serial algorithm

The serial algorithm stated above is a slightly modified form of the original version. In the modified verison, we calculate the lower bound at a node only if its current objective value is better than the global upper bound. This saved almost 20% of the solution CPU time. We use the CPU time obtained from this improved serial solution method (GROUPS) for evaluating the parallel algorithm.

4. The parallel branch-and-bound algorithm Branch-and-bound has been a popular solution tool for solving NP-complete combinatorial problems [9,13,25,31] for many years. Various strategies have been employed for tree exploration in the branch-and-bound method. Choosing a particular strategy depends on the number of processors available, the communication parameters (shared and distributed), the number of processes that can be run on a process, and the characteristics of the problem [6,13].

In our method, we use a Master–Slave configuration applicable on the Parallel Virtual Machine (PVM) in a UNIX environment. The Master process activates or spawns the Slave processes and coordinates information interchange among the processes. Apart from maintaining the processor busy and idle queues, the Master process also controls the job queue and decides how to distribute the work among the Slave processes. Thus, the Master process makes the key decisions. The Slave processes perform the actual problem solving, evaluating and fathoming the nodes in the tree. The Slave processes keep track of the local and global incumbent solution objective values.

First, the Master process spawns all Slave processes so as to initialize all processes at the same time, to exploit the advantages of parallelism to the fullest. The Master process assigns the busy Slaves to the busy process queue, while the idle ones are assigned to an idle process queue. Subproblems that are waiting to be assigned to idle Slaves are put in a job wait queue. Initially, the Master process assigns item 1 to group 1 and then introduces a dichotomy in the tree. That is, item 2 can now be placed in group 1 or 2, thus generating two subproblems which are assigned to two idle Slave processes, if available; otherwise, they are placed in a job wait queue.

Each of these Slave processes works independently on its part of the subproblem. A Slave process may send part of its tree to the Master process whose responsibility is to assign jobs to idle processes. The generation of subproblems by a Slave process is done dynamically. Once it reaches a specified depth (NDEPTH), it spins off the right branches to the Master process for assignment to other idle Slave processes, while continuing to work on the left branch much like the serial algorithm does. Subproblems are always generated from the top of the tree to ensure that all Slaves have enough work as they proceed deeper in the tree. Whenever a Slave process finds a new incumbent, it not only updates its local incumbent, but also broadcasts the objective value to all other Slave processes. Each Slave checks for a global upper bound (GUBND) once every NCHECK nodes are evaluated. The value of NCHECK is increased as we proceed deeper in the tree. If a Slave process has finished evaluating its subproblem, it broadcasts an idle message to the Master process that it needs another job.

The Master process then assigns a job from the job wait queue. However, if the job wait queue is empty, it assigns the Slave process to the process idle queue for later use. The parallel branch-and-bound methodology terminates if all the Slave processes are idle.

Therefore, there is communication not only between the Master and Slave processes, but also among the Slave processes. The Master process receives the following three types of messages from a Slave process:

(1) a new subproblem message (message type = Job),
(2) an idle message, if the Slave is done working on a subproblem (message type = Idle), and
(3) a DEAD message along with its solution to ensure that all Slaves are done and dead before the Master terminates (message type = Dead).

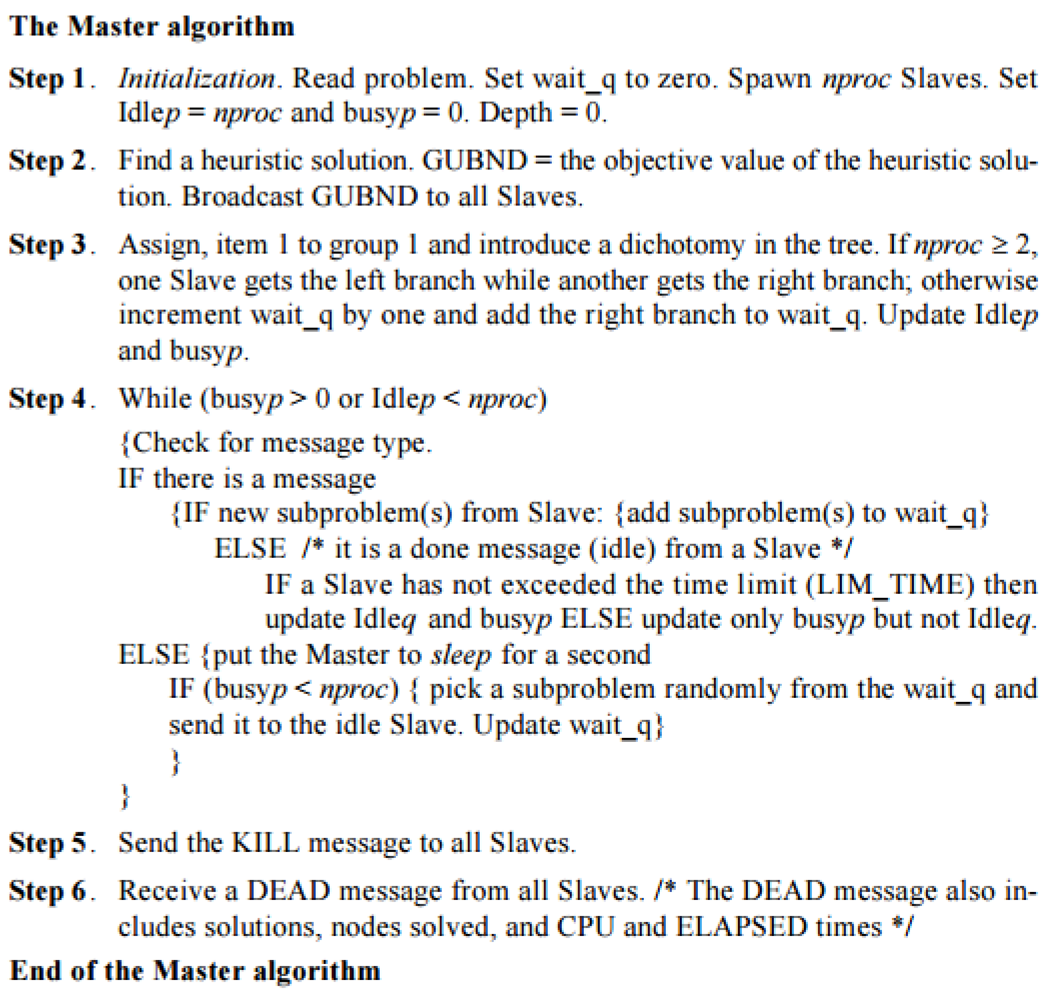
Similarly, a Slave process can receive the following types of messages either from the Master process or another Slave process:

(1) a new subproblem to be assigned to the Slave process from the Master process (message type = New Job),
(2) an initial upper bound (GUBND) from the Master process (message type = GUBND),
(3) a new global upper bound (GUBND) from another Slave process during the optimization procedure (message type = New GUB), and
(4) a KILL message, from Master Process when all Slaves are done (message type = KILL).

A depth-and-breadth-first combined strategy is used to explore the branch-andbound tree. As mentioned earlier, a Slave process always explores the left branch of the subproblem (depth-first approach). The right branch (or branches) are assigned to idle Slave Processes from the idle process queue (breadth-first approach) until a certain depth (BDEPTH) is reached to ensure that all Slave processes get enough work to do. After this depth is reached, the Slave process works like a serial algorithm using the depth-first approach to avoid communication overload. Thus, subproblems are explored both depth-wise and breadth-wise.

To formally present the parallel branch-and-bound algorithm for the clustering problem (PGROUPS), we divide the algorithm into two parts: the Master algorithm and the Slave algorithm. The communication between the two processes is handled by PVM. Only one copy of the Master process is active at a given time, while several copies of Slave processes may be running simultaneously. Some additional notation used in the description of the Master algorithm follows:

busyp : number of busy Slave processes,
Idlep : number of idle Slave processes,
nproc : number of Slaves the Slave processes spawned,
wait_q : job wait queue,
Nodes : total number of nodes solved by each Slave process.



The Master algorithm

```
Step 1. Initialization. Read problem. Set wait_q to zero. Spawn nproc Slaves. Set
        Idlep = nproc and busyp = 0. Depth = 0.
Step 2. Find a heuristic solution. GUBND = the objective value of the heuristic solu-
        tion. Broadcast GUBND to all Slaves.
Step 3. Assign, item 1 to group 1 and introduce a dichotomy in the tree. If nproc ≥ 2,
        one Slave gets the left branch while another gets the right branch; otherwise
        increment wait_q by one and add the right branch to wait_q. Update Idlep
        and busyp.
Step 4. While (busyp > 0 or Idlep < nproc)
        {Check for message type.
        IF there is a message
           {IF new subproblem(s) from Slave: {add subproblem(s) to wait_q}
              ELSE /* it is a done message (idle) from a Slave */
                 IF a Slave has not exceeded the time limit (LIM_TIME) then
                 update Idleq and busyp ELSE update only busyp but not Idleq.
        ELSE {put the Master to sleep for a second
           IF (busyp < nproc) { pick a subproblem randomly from the wait_q and
           send it to the idle Slave. Update wait_q}
           }
        }
Step 5. Send the KILL message to all Slaves.
Step 6. Receive a DEAD message from all Slaves. /* The DEAD message also in-
        cludes solutions, nodes solved, and CPU and ELAPSED times */
```

End of the Master algorithm

The Slave and Master algorithms are intertwined via the communication system. Not only can a Slave send messages to and receive messages from the Master process, it can also send messages to other Slave processes. We next present some notation used in the description of the Slave algorithm:

checkdep : Depth at which to stop backtracking.
Depth : Depth at which the subproblem starts.
nextbdep : Next depth at which the Slave process needs to spin off subproblems.
NODES : Total number of nodes evaluated by the Slave process.
NITR : An iteration counter that keeps track of the number of times the search process reached NDEPTH.
Maxdep : Depth at which the Slave process stops spinning off new subproblems.

The Slave algorithm

Step 1. *Initialization*. Read problem. Set NODES, checkdep, nextbdep, NITR = 0. Set parameters Maxdep and Bdepth. NDEPTH = INT($n/2$) + 1.

Step 2. Receive GUBND from the Master.

Step 3. Call CPU and ELAPSED timers.

Step 4. Check for message from the Master. While (message ≠ KILL)

```
{Obtain tree info, Depth, and obj_val(Depth – 1), checkdep = Depth,
nextbdep = checkdep + 1. index = TRUE.
While (index)
    { IF (assignment is feasible)
    {Calculate the objective value at the current depth.
IF (NCHECK nodes have been evaluated)
    {check for a message from another Slave regarding the new GUBND.
IF one is found, then update GUBND}
    IF ((objective value at the current depth + lower bound) ≤ GUBND)
    {IF (Depth ≠ n) THEN
    {IF (Depth = NDEPTH) {NITR = NITR + 1
    IF (((Depth = Bdepth) or (Depth = NDEPTH and NITR ≤ 1)) and
    (nextbdep ≤ Maxdep)) THEN
    {increment checkdep and nextbdep.
    Send right branches to the Master and continue to work on the left-
    most branch}
        ELSE {Do not send subproblems. Branch left}
    }
}
New incumbent found. IF (incumbent objective value < Global_UB)
{store the solution, update Global_UB and broadcast it to all Slaves.
    IF not the right most branch, THEN branch right
ELSE
    IF limit time not exceeded execute the fathom routine to back the
    last item out of stack
IF time limit exceeded
```

```
            {index = FALSE, send process_id, Idle message, LIM_TIME message
            to the Master}
            }
            ELSE Fathom node}
            ELSE Fathom node}
            IF (Depth (checkdep)
                {IF (time limit exceeded)
                {index = false}
                Send process_id, Idle message, LIM_TIME message to the Master}
            ELSE
            {execute the fathom routine to back the last item out of stack}
            }
        }
Step 5. /* KILL message from the Master */ Collect CPU and ELAPSED times. Send
        DEAD message, Slave solution, NODES, and ELAPSED time to the Master.

End of the Slave algorithm
```

5. Implementation parameters for the parallel algorithm

We implemented the parallel branch-and-bound algorithm on the IBM PowerParallel System (SP-2) at University Computing & Network Services at the University of Georgia using the xlf (f77) FORTRAN compiler under the AIX Version 3.2 (Unix) operating system. Our programming environment is the Parallel Virtual Machine (PVM). PVM function calls not only allow the creation or termination of Slave processes, but also aid in the communication among processes. Some of the parameters that are critical for a proper implementation of the parallel algorithm are: communication, distribution of work and synchronicity.

5.1. Communication

In our branch-and-bound algorithm, the Slave processes and Master process must communicate to share data. To reduce the communication complexity, the Slave processes perform the fathoming operations (actual problem solving), while the Master process performs bookkeeping operations, i.e., maintains the job wait queue and process idle or busy queues. In asynchronous communication, when a Slave process finds a new incumbent solution, it transmits the objective value to all other active Slave processes without waiting for the other Slave processes to finish their work. When each active Slave process receives the new incumbent objective value, it updates its global upper bound, which enhances the pruning operation. This not only maintains the efficiency of the solution procedure, but also minimizes both communication complexity and contention. The incumbent solution is not communicated to any process, since it is not needed in the actual problem solving. Also, the Master process does not need to receive the new global incumbent objective values. These further reduce communication contention.

5.2. Distribution of work Unsolved problems are kept in a job wait queue until they are allocated to idle Slave processes. There are various strategies for treating a work pool (a location where generated subproblems are stored). One of the methods is to use a single central pool that is managed by the Master process [5,13]. An alternate approach is the use of multiple pools, where there are several memory locations where processes can find and store their units of work [10,13].

We use the single central pool strategy as it is relatively easy to keep track of active subproblems that are waiting to be assigned to a Slave process. The Master process maintains this central pool, to which it adds the new subproblems generated by a Slave process. When a Slave process becomes idle, the Master process assigns a subproblem to it. If there are no subproblems in the central pool, the Slave process is added to the process idle queue.

5.3. Synchronicity

A synchronous algorithm divides work into phases such that within each phase, processes perform their jobs independently, while communication occurs only between phases. Our method is asynchronous in nature to avoid the inherent idle time caused by synchronous methods. Prior research indicates that the nondeterministic nature of asynchronous algorithms yield better results as compared to synchronous algorithms [14,29].

6. Implementation issues

The University of Georgia's IBM SP-2 is a dedicated machine with eight processors (nodes) of which only four are available. In our parallel algorithm, the Slave processes do most of the work. Preliminary test results indicated that the Master process is active for only about 0.1% of the time. PVM facilitates running a Master process and one Slave process on a single node of the IBM SP-2 when the ip mode (Internet Protocol (IP)) is used for communication. Alternatively, the user mode (which has a low-latency protocol) allows for only one process per node. The bandwidth (speed) is the same (40 MB s) as in user mode, but the startup time for each communication (latency) is higher than that of the user mode. Although communication latency is slightly higher in the case of the ip mode, we gain a usable processor by activating the Master process and a Slave process on the same processor. Hence, we adopted the ip mode (high latency) for all our tests.

To make the solution time comparisons more meaningful, both GROUPS and PGROUPS were implemented on the IBM SP-2. For GROUPS, only one processor is used, while in PGROUPS, the number of processors is always equal to the number of Slave processes. In PGROUPS, the Master process and the first Slave process are activated on the same physical processor, while all other Slave processes are activated on unique physical processors. Because of the way the operating system time shares the work in a processor, we found it necessary to put the Master process to sleep for one second (see step 4 of the Master algorithm). This keeps the Slave process that shares the processor with the Master process running and the load balancing even.

The CPU time to find and verify optimality is measured for both GROUPS as well as for PGROUPS. The CPU time is congruous to wall-clock time on the IBM SP- 2, since it is a dedicated computing environment. In addition to CPU time, we also tally the total number of branch-and-bound nodes evaluated by each Slave process during the optimization process. We use this to test load balancing in the parallel tests. Since our algorithm differs only in the optimization part, we do not measure the CPU time needed to read the problem data. In PGROUPS, all processes read the problem data at the same time during initialization.

We used real-world problem data as well as randomly generated clustering problems to test our algorithm. We used uniform distributions (with a random number seed, LSEED) for the randomly generated problems. Since the time limit on the IBM SP-2 is 6 CPU hours (21,600 CPU seconds), we considered problems that required 6 CPU hours or less in serial for our evaluation. We included code to detect the condition for a graceful exit rather than a system imposed termination. Also, we did not consider problems where GROUPS required less than 150 CPU seconds to obtain an optimum, since its parallel counterpart involves some communication and the overall time would be too small to show the effectiveness of PGROUPS. Our real-world problem data was from an industry survey of grouping the benefits of executive information systems [19]. Table 1 shows the specifications of the problem sets (both randomly generated and real-world problem data) used to evaluate PGROUPS. The problem_id for randomly generated data sets begin with the letter R, the first number indicates the number of items and the second number indicates the number of groups (i.e., R-22-5 indicates it is a randomly generated problem set for which 22 items are grouped into 5 clusters). Likewise, in the problem_id for real-world problem sets, the text indicates the base problem, while the two numbers indicate the number of items and the groups respectively (i.e., EIS-23-5 indicates it is the data associated with grouping 23 EIS benefits into 5 clusters).

Within the Master Process, we use another random number seed, NSEED, to set the random number generator used to select a subproblem to be assigned to an idle Slave process from the central pool of jobs. Prior research [14] and preliminary test results indicate that randomizing the subproblems increases the likelihood of obtaining an optimal solution early as compared to using the last-in-first-out (LIFO) or first-in-first-out (FIFO) strategy.

Table 1

Specifications of problem sets used for evaluating PGROUPS.

Problem_id	Items	GROUPS	LSEED	Cost range
R-22-5	22	5	196719	1-100
R-23-5A	23	5	196719	1-100
R-22-6	22	6	372917	1-100
R-23-6	23	6	372917	1-100
R-23-5B	23	5	2719	1-100
R-25-5	25	5	196719	1-100
EIS-23-4	23	4		
EIS-23-5	23	5		
EIS-23-6	23	6		
EIS-23-7	23	7		

The initial setup of GROUPS and PGROUPS are similar. That is, after the input data are read, an initial feasible heuristic solution based on an improvement heuristic algorithm [2] is found. For problems with no side constraints, the following heuristic was developed to obtain a good initial incumbent:

Improved heuristic

```
Step 1. Initialization. Set $L_k = n/K$, the group limit for $k = 1,\ldots, K$ groups. If the ratio
        $n/K$ is not an integer then increment $L_k$ by 1, starting with group 1 until
        $\sum L_k = n$. Set $k = 1$, $c = 1$.
Step 2. While ($k \leq K$)
            { While (group $k$ is NOT full and more items are available)
                {Place item $c$ into group $k$
                Select item $c'$ as the closest one to item $c$
                Set $c = c'$ }
            $k = k + 1$
        set $c$ = 1st available item}
```

In most cases, this heuristic improved the initial solution objective value by at least 20%. The global upper bound is assigned the objective value of this heuristic solution. This improvement combined with the modification in GROUPS decreased the overall solution CPU time by 40%. Once an optimum is found and verified, the procedure terminates. As mentioned earlier, since the two methods vary only in the optimization procedure, the solution time is determined from the start of the optimization step until all necessary nodes have been explored and an optimum is found and verified. Thus, the solution time of PGROUPS is the time from the beginning of the optimization process until all Slave processes have gone idle after having explored all necessary nodes and confirmed its optimality. In both GROUPS and PGROUPS, the procedure terminates with a warning message when the maximum time of 6 CPU hours (21,600 seconds) is exceeded, or if it encounters an execution error.

7. Computational results

Due to the dedicated nature of the IBM SP-2, there was no significant difference in the CPU times obtained for GROUPS, when the test problems were repeated. Hence, we only report the CPU times obtained from one run for each test problem. As for PGROUPS, there were slight differences because of the randomness in subproblem selection. In table 2 we present the CPU times obtained using GROUPS, and PGROUPS on one, two, three and four processors (P1, P2, P3, P4) for all ten problems. Table 3 provides a summary of the total nodes evaluated by GROUPS and PGROUPS for one, two, three and four processors (P1, P2, P3, P4). For P2, P3 and P4, we also present the total nodes evaluated by each of the Slave processes.

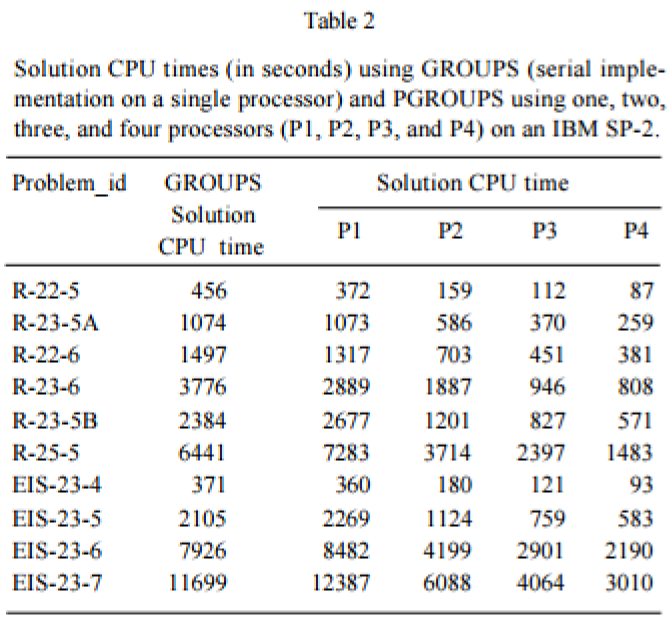


Table 2

Solution CPU times (in seconds) using GROUPS (serial implementation on a single processor) and PGROUPS using one, two, three, and four processors (P1, P2, P3, and P4) on an IBM SP-2.

Problem_id	GROUPS Solution CPU time	Solution CPU time			
		P1	P2	P3	P4
R-22-5	456	372	159	112	87
R-23-5A	1074	1073	586	370	259
R-22-6	1497	1317	703	451	381
R-23-6	3776	2889	1887	946	808
R-23-5B	2384	2677	1201	827	571
R-25-5	6441	7283	3714	2397	1483
EIS-23-4	371	360	180	121	93
EIS-23-5	2105	2269	1124	759	583
EIS-23-6	7926	8482	4199	2901	2190
EIS-23-7	11699	12387	6088	4064	3010

The efficacy of the parallel algorithm is measured using the following metrics: speedup and load balancing (i.e., the proportion of work done by each Slave process is the same). We determine both absolute and relative speedup of the parallel algorithm. Absolute speedups for P1, P2, P3 and P4 are calculated by dividing the serial solution time by the parallel solution time; relative speedups for P2, P3 and P4 are determined by dividing the P1 solution CPU time by the solution CPU times for P2, P3 and P4, respectively. For half the test problems, PGROUPS with only one Slave process (P1) was faster than GROUPS. The mean of all 10 runs indicates a 2.8% improvement. On the one hand, one could argue that P1 is the best serial algorithm since it runs one processor. On the other hand, two processes are required. Rather than belabor the point, we report both the absolute and relative speedups in this paper.

Table 3

Number of branch-and-bound tree nodes evaluated using GROUPS (serial implementation on a single processor) and PGROUPS using one, two, three, and four processors (P1, P2, P3, and P4).

Problem_id	GROUPS	P1	P2		
		Slave 1	Slave 1	Slave 2	Total
R-22-5	49037754	3547368	15117621	15769643	30887264
R-23-5A	99485266	91625417	50408992	49624287	100033279
R-22-6	124932382	146243100	734707562	81026609	154497371
R-23-6	285847155	292558866	192278051	187750042	380028093
R-23-5B	235617845	246322129	109561824	113705466	223267290
R-25-5	527010231	553097312	283761779	281177181	564938960
EIS-23-4	22436148	22436098	11246218	11359998	22606216
EIS-23-5	208852347	208852287	103042467	107178837	210221304
EIS-23-6	966142668	966142607	474617206	494728126	969345332
EIS-23-7	1646158759	1659969380	810947920	840234278	1651182198

Problem_id	GROUPS	P1	P3			
		Slave 1	Slave 1	Slave 2	Slave 3	Total
R-22-5	49037754	3547368	10653376	10778688	10415015	3184707900
R-23-5A	99485266	91625417	31806843	31413733	32035303	95255879
R-22-6	124932382	146243100	49626117	48513823	49373813	147513753
R-23-6	285847155	292558866	93186347	99750644	95795354	288732345
R-23-5B	235617845	246322129	73319241	77901487	77660844	228881572
R-25-5	527010231	553097312	175916475	181402331	19224568	549564674
EIS-23-4	22436148	22436098	7485692	7573392	7686209	22745293
EIS-23-5	208852347	208852287	68712758	71044486	71747799	211505043
EIS-23-6	966142668	966142607	32702727	323815776	321656914	972499960
EIS-23-7	1646158759	1659969380	5244665929	561109027	565320608	1651095564

Problem_id	GROUPS	P1	P4				
		Slave 1	Slave 1	Slave 2	Slave 3	Slave 4	Total
R-22-5	49037754	3547368	8015171	8724760	7815284	7697469	32252684
R-23-5A	99485266	91625417	21385522	22813513	22724856	21640662	885645533
R-22-6	124932382	146243100	41882951	38053141	37423018	37423018	155539041
R-23-6	285847155	292558866	80478219	84036889	78536738	84114491	327166339
R-23-5B	235617845	246322129	49484846	50935102	54544226	53760482	208724656
R-25-5	527010231	553097312	110358157	116625044	110647746	117637394	455268341
EIS-23-4	22436148	22436098	3538901	5673607	6038596	5830362	22901466
EIS-23-5	208852347	208852287	53162079	52045473	54092710	53506735	212806997
EIS-23-6	966142668	966142607	228116130	259895309	242603630	245049630	975664699
EIS-23-7	1646158759	1659969380	406609613	407383956	439274031	402999310	16562669190

Table 4

Summary of absolute speedups (GROUPS/PGROUPS).

Problem_id	GROUPS	P1	P2	P3	P4
R-22-5	1	1.226	2.868	4.071	5.241
R-23-5A	1	1.001	1.833	2.903	4.147
R-22-6	1	1.137	2.132	3.319	3.929
R-23-6	1	1.307	2.001	3.992	4.673
R-23-5B	1	0.891	1.985	2.883	4.175
R-25-5	1	0.884	1.734	2.687	4.343
EIS-23-4	1	1.031	2.061	3.066	3.989
EIS-23-5	1	0.928	1.873	2.773	3.611
EIS-23-6	1	0.934	1.888	2.732	3.619
EIS-23-7	1	0.944	1.922	2.879	3.887
Mean	1	1.028	2.030	3.131	4.162
Standard deviation	0.0	0.147	0.316	0.508	0.497
Maximum	1	1.307	2.868	4.071	5.241
Minimum	1	0.884	1.734	2.687	3.611

In determining the absolute speedup, we use the CPU time for the sequential algorithm. We also present the relative speedup to evaluate the performance of PGROUPS with respect to P1. Table 4 summarizes the absolute speedups for PGROUPS, while table 5 provides the relative speedups for PGROUPS.

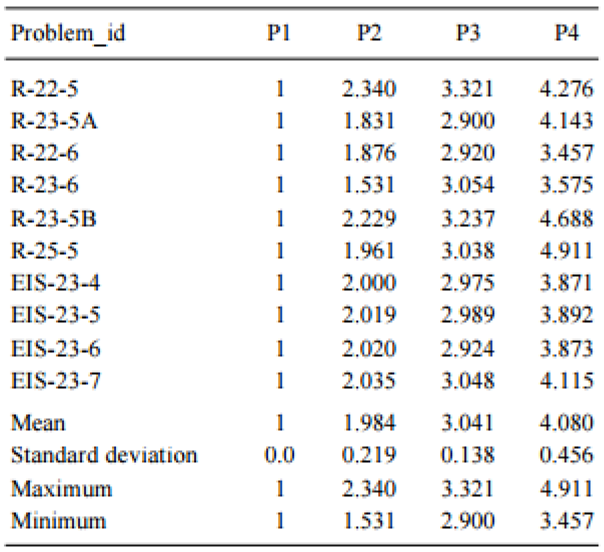
To determine if equal work load is obtained by the Slave processes for P2, P3 and P4, we divide the number of nodes evaluated by each of the Slave processes by the total number of nodes evaluated by PGROUPS in each case. Table 6 provides a summary of load balancing for PGROUPS. On average, each Slave process has performed an (approximately) equal share of the optimization work.

8. Summary and conclusions

We have developed, implemented and tested a dynamic, asynchronous parallel algorithm (PGROUPS) to execute a branch-and-bound methodology to solve cluster analysis problems on a PVM platform. The cluster analysis problem defines the grouping of n items into K groups, where items in each group are highly homogenous in nature while the clusters themselves are distinct. Clustering problems are highly combinatorial in nature and, hence, optimization methods take a long time to obtain an optimum for even medium-sized problems.

Table 5

Summary of relative speedups (P1/PGROUPS).



Problem_id	P1	P2	P3	P4
R-22-5	1	2.340	3.321	4.276
R-23-5A	1	1.831	2.900	4.143
R-22-6	1	1.876	2.920	3.457
R-23-6	1	1.531	3.054	3.575
R-23-5B	1	2.229	3.237	4.688
R-25-5	1	1.961	3.038	4.911
EIS-23-4	1	2.000	2.975	3.871
EIS-23-5	1	2.019	2.989	3.892
EIS-23-6	1	2.020	2.924	3.873
EIS-23-7	1	2.035	3.048	4.115
Mean	1	1.984	3.041	4.080
Standard deviation	0.0	0.219	0.138	0.456
Maximum	1	2.340	3.321	4.911
Minimum	1	1.531	2.900	3.457

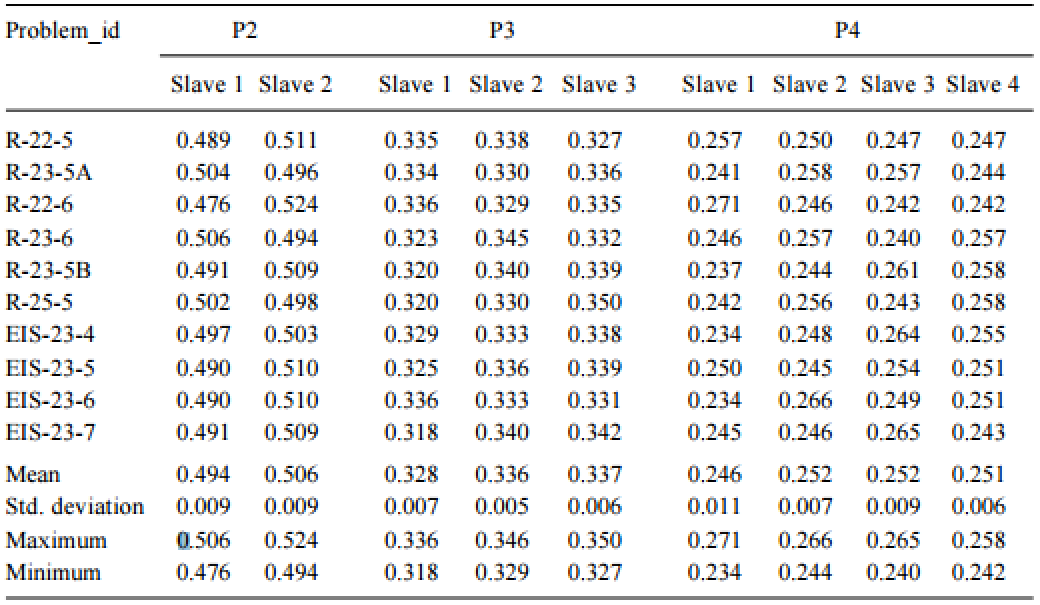
Our solution methodology is based on the serial branch-and-bound method (GROUPS) developed by Aronson and Klein [2]. However, before developing the parallel method, we improved their serial method in two ways. First, we modified the heuristic to get a better initial solution for pure clustering problems (i.e., with no side constraints). Secondly, we improved the bound fathoms by not calculating the lower bounds unless the objective value at the current node was better than the global upper bound. These improvements improved the solution CPU time of the serial GROUPS by almost 40%.

PGROUPS was implemented on the IBM Scalably PowerParallel System (SP-2), a distributed memory multiprocessor, using the Internet protocol communication switch (high latency). PGROUPS was evaluated on a set of ten problems (both randomly generated and real-world problem data) using up to four processors. For half of the test problems, we obtained superlinear speedups. On average, we obtain superlinear speedups (both absolute and relative) for PGROUPS. In addition, PGROUPS also achieves even load balancing among the Slave processes (for P1, P2, P3 and P4), thus fully realizing the advantages of parallelism.

Restriction of the SP-2 configuration to run only one process per node, using the low latency communication switch, was one of the limitations in testing PGROUPS. This increased the communication penalty slightly, but we gained the full use of a processor from the high latency switch. Another limitation is the 6 hour time limit for using the IBM SP-2 which curtailed our ability to solve larger problems using GROUPS. Thus, our sample is somewhat limited.

Table 6

Summary of load balancing (proportion of nodes evaluated by each Slave process/total nodes evaluated by PGROUPS) for P2, P3, and P4.



Problem_id	P2		P3			P4			
	Slave 1	Slave 2	Slave 1	Slave 2	Slave 3	Slave 1	Slave 2	Slave 3	Slave 4
R-22-5	0.489	0.511	0.335	0.338	0.327	0.257	0.250	0.247	0.247
R-23-5A	0.504	0.496	0.334	0.330	0.336	0.241	0.258	0.257	0.244
R-22-6	0.476	0.524	0.336	0.329	0.335	0.271	0.246	0.242	0.242
R-23-6	0.506	0.494	0.323	0.345	0.332	0.246	0.257	0.240	0.257
R-23-5B	0.491	0.509	0.320	0.340	0.339	0.237	0.244	0.261	0.258
R-25-5	0.502	0.498	0.320	0.330	0.350	0.242	0.256	0.243	0.258
EIS-23-4	0.497	0.503	0.329	0.333	0.338	0.234	0.248	0.264	0.255
EIS-23-5	0.490	0.510	0.325	0.336	0.339	0.250	0.245	0.254	0.251
EIS-23-6	0.490	0.510	0.336	0.333	0.331	0.234	0.266	0.249	0.251
EIS-23-7	0.491	0.509	0.318	0.340	0.342	0.245	0.246	0.265	0.243
Mean	0.494	0.506	0.328	0.336	0.337	0.246	0.252	0.252	0.251
Std. deviation	0.009	0.009	0.007	0.005	0.006	0.011	0.007	0.009	0.006
Maximum	0.506	0.524	0.336	0.346	0.350	0.271	0.266	0.265	0.258
Minimum	0.476	0.494	0.318	0.329	0.327	0.234	0.244	0.240	0.242

Our test results indicate that PGROUPS has the potential to solve large, complex cluster analysis problems efficiently. We were limited in our testing to at most four processors and we recognize that to generalize our results further testing on a largescale, multiple instruction, multiple data computer, such as an IBM SP-2 with 512 processors, is warranted.

Acknowledgement

Jay E. Aronson was partially supported by a Fellowship from The Terry and Selig Foundations.

References

[1] S.T. Allison, A.M.R. Jordan and C.E. Yeatts, A cluster-analytic approach toward identifying the structure and content of human decision making, Human Relations 45(1992)49–73.

[2] J.E. Aronson and G. Klein, A clustering algorithm for computer-assisted process organization, Decision Sciences 20(1989)730–745.

[3] E. Balas, An additive algorithm for solving linear programs with zero–one variables, Operations Research 13(1965)517–546.

[4] D.P. Bertsekas and DA. Castañon, Parallel asynchronous Hungarian methods for the assignment problem, ORSA Journal on Computing 5(1993)261–274.

[5] A. de Bruin, A.H.G. Rinnooy Kan and H.W.J.M. Trienekens, A simulation tool for the performance evaluation of parallel branch-and-bound algorithms, Mathematical Programming 42(1988)245– 271.

[6] J. Clausen and J.L. Träff, Implementation of parallel branch-and-bound algorithms: Experiences with graph partitioning problems, Annals of Operations Research 33(1991)331–349.

[7] R.M. Cormack, A review of classification, Journal of the Royal Statistical Society, Series A 134 (1971)321–367.

[8] R.B.R. DeSouza and R. Bell, A tool cluster based strategy for the management of cutting tools in flexible manufacturing systems, Journal of Operations Management 10(1991)73–91.

[9] S. Dowaji and C. Roucairol, Load balancing strategy and priority of tasks in distributed environments, Rapport de Recherche (Research Report) RR-94 32, Laboratoire PRiSM, Université de Versailles, Saint-Quentin-en-Yvelines, Versailles, France, October, 1994.

[10] J. Eckstein, How much communication does parallel branch and bound need?, in: Proceedings of the Parallel Optimization Colloquium POC-96, Université de Versailles, Versailles, France, March, 1996, pp. 59–68.

[11] B. Everitt, Cluster Analysis, 2nd ed., Halsted Press, Division of Wiley, New York, NY, 1980.

[12] A. Ferrenberg, University Computing & Network Services, The University of Georgia, Athens, GA, private communication, April, 1996. [13] B. Gendron and T.G. Crainic, Parallel branch-and-bound algorithms: Survey and synthesis, Operations Research 42(1994)1042–1066.

[14] B. Gupta, A parallel multiperiod assignment algorithm, Doctoral Dissertation, Terry College of Business Administration, University of Georgia, Athens, GA, 1995.

[15] K.M. Hilmer and J.E. Aronson, A visual front end to a cluster analysis algorithm, Technical Report, The University of Georgia, Athens, GA, 1993.

[16] J.G. Hirschberg, E. Maasoumi and D.J. Slottje, Cluster analysis for measuring welfare and quality of life across countries, Journal of Econometrics 50(1991)131–150.

[17] G.P. Hodgkinson, J. Padmore and A.E. Tomes, Mapping consumers' cognitive structures: A comparison of similarity trees with multidimensional scaling and cluster analysis, European Journal of Marketing 25(7)(1991)41–60.

[18] A.S. Homayoun, The use of cluster analysis in analyzing large engineering records collection, Records Management Quarterly (October 1984)22–25.

[19] S.L. Iyer and J.E. Aronson, Grouping EIS benefits: An optimal clustering approach, Proceedings of the Americas Conference on Information Systems, Pittsburgh, PA, August, 1995.

[20] J. Karimi, An automated software design methodology using CAPO, Journal of Management Information Systems 3(3)(1986)71–100.

[21] G. Klein and J.E. Aronson, Optimal clustering: A model and method, Naval Research Logistics 38(1991)447–461.

[22] G. Klein, P.O. Beck and B.R. Konsynski, Computer aided process structuring via mixed integer programming, Decision Sciences 19(1988)750–761. [

23] G. Klein, D.B. Tesch, J.E. Aronson and P.O. Beck, A computer-aided process structuring methodology, Working Paper, Louisiana Tech University, Ruston, LA, 1995.

[24] S. Kulkarni, D.F. Rogers, and J.E. Aronson, An optimal clustering model for cellular manufacturing, Conference Presentation, INFORMS, Washington, DC, May, 1996.

[25] E.L. Lawler and D.E. Wood, Branch-and-bound algorithms: A survey, Operations Research 14(1966) 699–719. [26] J. Mohan, Experience with two parallel programs solving the travelling salesman problem, in: Proceedings of the 1983 International Conference on Parallel Processing, 1983, pp. 191–193.

[27] J. Mulvey and H. Crowder, Cluster analysis: An application of Lagrangian relaxation, Management Science 25(1979)329–340. [

28] G. Punj and D.W. Stewart, Cluster analysis in marketing research: Review and suggestions for application, Journal of Marketing Research 20(1983)134–148.

[29] M.J. Quinn, Analysis and implementation of branch-and-bound algorithms on a hypercube multicomputer, IEEE Transacitons on Computers 39(1990)384–387.

[30] H. Romesburg, Cluster Analysis for Researchers, Lifetime Learning Publications. Belmont, CA, 1984.

[31] C. Roucairol, Parallel processing for difficult combinatorial optimization problems, European Journal of Operations Research 92(1996)573–590.

[32] A.W. Spisak, Cluster analysis as a quality management tool, Quality Progress 25(12)(1992)33–38.

[33] V.S. Sunderam, PVM: A framework for parallel distributed computing, Concurrency, Practice and Experience 2(4)(1990)315–339.

[34] V.S. Sunderam, and G.A. Geist, The PVM system: Supercomputer-level concurrent computation on a network of IBM RISC System 6000 POWERstations, Reprinted from Scientific Excellence In Supercomputing: The 1990 IBM Contest Prize Papers, vol. 2, eds. K. Billingsley, H. Brown and E. Derohanes, The Baldwin Press, The University of Georgia, Athens, GA, 1992, pp. 779–804.