# Control Schemes in a Generalized Utility for Parallel Branch-and-Bound Algorithms

Yuji Shinano*, Kenichi Harada and Ryuichi Hirabayashi
Department of Management Science
Science University of Tokyo
1-3, Kagurazaka, Shinjuku-ku, Tokyo 162, Japan
shinano@ms.kagu.sut.ac.jp

## Abstract

*Branch-and-bound algorithms are general methods applicable to various combinatorial optimization problems and parallelization is one of the most hopeful methods to improve these algorithms. Parallel branch-and-bound algorithm implementations can be divided in two types based on whether a central or a distributed control scheme is used. Central control schemes have reduced scalability because of bottleneck problems frequently encountered. In order to solve problem cases that cannot be solved with sequential branch-and-bound algorithm, distributed control schemes are necessary. However, compared to central control schemes, higher efficiency is not always achieved through the use of a distributed control scheme. In this paper, a mixed control scheme is proposed, changing between the two different types of control schemes during execution. In addition, a dynamic load balancing strategy is applied in the distributed control scheme. Performance evaluation for three different cases is carried out: central, distributed, and mixed control scheme. Several TSP instances from the TSPLIB are experimentally solved, using up to 101 workstations. The results of these experiments show that the mixed control scheme is one of the most promising control schemes and furthermore, the hybrid selection rule, which was introduced in our previous work, has an advantage in parallel branch-and-bound algorithms.*

## 1. Introduction

*Branch-and-bound algorithms* are general methods applicable to various combinatorial optimization problems that belong to the class of NP-hard problems. These algorithms are search-based techniques that enumerate the entire solution space implicitly. Parallelization is the most appropriate method for accelerating the enumeration process. Since the algorithms usually need much time to evaluate a subproblem, we implement *high-level parallelism* of such algorithms, in which case all the existing subproblems are parallelized simultaneously provided that an adequate number of processors is available. Even though there are several criteria for classifying parallel branch-and-bound algorithms [1, 4, 9, 14], the most useful criterion is the search tree management. The search tree is managed with a single subproblem pool in the central control case(*central control scheme*) or with multiple subproblem pools in the distributed control case(*distributed control scheme*). Parallelization with a single subproblem pool usually achieves higher efficiency until a large number of processors are used [3, 7].

In a previous work[13], a generalized system for parallel branch-and-bound algorithms was developed (*PUBB:Parallelization Utility for Branch-and-Bound algorithms*). This was an implementation of the central control scheme with a single subproblem pool. Computational experiments with PUBB showed the robustness of the hybrid selection rule in the central control scheme, but the distributed control scheme was not tested. The system has been rebuilt to accommodate distributed control with multiple subproblem pools this time. Moreover, *mixed control scheme* is implemented in this paper, which incorporates both the central and the distributed control schemes during execution. In this paper, a new dynamic load balancing strategy is also applied to the renewed PUBB. Load balancing strategies based on the *diffusion* or the *dimension exchange* methods perform load balancing operations iteratively[2, 8, 10, 15]. However, the load balancing strategy proposed here performs the operation only when it is necessary. That is to say, a processor with few tasks to perform, or with many tasks that exceed its capacity, invokes the load balancing process. Naturally this kind of load balancing has poor performance in terms of “quality”

*Research Fellow of the Japan Society for the Promotion of Science.

as described in [8]( Load balancing in parallel branch-and-bound algorithms can generate additional workload. Therefore, when applying this strategy the operator has to worry about not only the workload distribution, but also how to decrease the workload level itself. An application of the load balancing strategy which does not introduce much workload and distributes the tasks to the processors efficiently, leads to high "quality"). It is shown in this paper, that "quality" can be enhanced with the appropriate use of selection rules and control schemes.

## 2. Renewed Parallelization Utility for Branch-and-Bound algorithms

The renewed PUBB is a completely redesigned algorithm and not an extension of the old implementation of [13]. All program codes are written in C++ and use the PVM (Parallel Virtual Machine) library. PUBB has three *running modes* with respect to the control schemes used. The central control scheme is applied in the "Master-Slave(MS)" mode, the distributed control scheme is applied in the "Fully-Distributed(FD)" mode and the mixed control scheme is applied in the "Master-Slave to Fully-Distributed(MStoFD)" mode.

### 2.1. Composition of the utility: the basic tasks

The renewed PUBB is composed of three tasks in the PVM, which are presented in Figure 1.

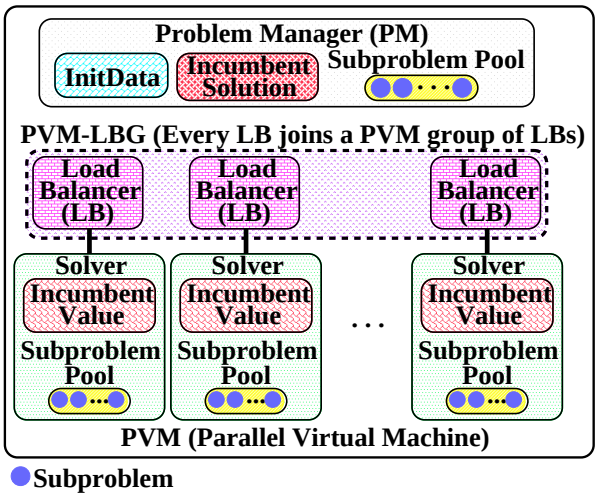


**Figure 1. The composition of tasks**

**Problem Manager(PM):** This task maintains throughout the execution the instance data and the incumbent solution for the target problem. In the MS mode, it has a subproblem pool and once it is initialized, it behaves like an "iterative server", handling requests from other tasks.

**Load Balancer(LB):** This task spawns the corresponding **Solver**. In the FD mode, it regulates the load balancing in cooperation with other **LB**s. It also behaves like an "iterative server", when it is initialized. All **LB**s join a PVM group called *PVM-LBG*.

**Solver:** This task and the **LB** have a one to one correspondence and both run on the same workstation. Subproblems are evaluated in the Solver. In PUBB, the "evaluation" can be defined as follows: (1)obtain a subproblem, (2)compute a bounding value, (3)check whether the subproblem can be eliminated or not, (4)select branching variables and generate new subproblems if necessary, and (5)remove the obtained subproblem. A local subproblem pool is available, only in the FD mode.

### 2.2. Implementation of selection rules

The data structure of a subproblem pool is presented in Figure 2. It consists of three index heaps: (a)the selection ordered heap which is ordered by the *selection criterion value*, (b)the best bound ordered heap and (c)the worst bound ordered heap. Each heap element has only pointers. Basic heap operations such as *insert* or *remove* are performed based on the value of the subproblem data. Using this data structure, a subproblem can be inserted into or removed from the pool by either selection criterion value, best bound, or worst bound order on a set of size $n$ in $O(\mathbf{lg}n)$ time. The pruning procedure removes subproblems from the subproblem pool by worst bound order as long as the bounding value of the subproblem on the top of the worst bound heap is worse than the incumbent value. This operation may take quite some time, but usually not many improved solutions are found in the algorithms.

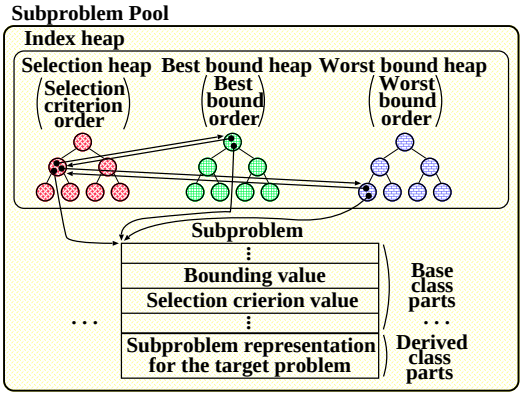


**Figure 2. Data structure of a subproblem pool**

The PUBB supports *depth first*, *breadth first*, *best bound first*, *hybrid* and any *heuristic value ordered* selection rules. The hybrid selection rule was proposed in [13] and Figure 3 shows how this rule is applied. Initially, depth first selection is applied until the search is "terminated", because

a feasible solution is found or it is certain that no improved solution can be yielded. This is then followed by the application of the best bound first selection rule. The depth first selection rule is then performed till the next "termination" and the iterations follow on. In the MS mode, the selection is performed on only one global subproblem pool. On the other hand, in the FD mode, the selection is performed on the local subproblem pool in each **Solver**.

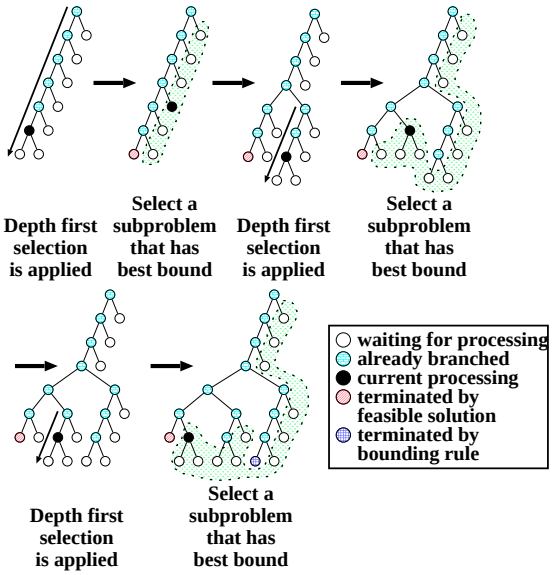


**Figure 3. Hybrid selection rule**

## 2.3. Initialization and interaction among tasks

### 2.3.1 "Master-Slave(MS)" mode

At first, a Problem Manager(**PM**) has to be started. The **PM** reads the problem instance data and recognizes the target problem type(minimization or maximization) and the selection rule. Subsequently, an initial solution and a root problem are obtained by calling the user defined functions. Initialization of the **PM** ends when the root problem is put into the subproblem pool. Then, the **PM** waits for requests.

When a Load Balancer(**LB**) starts, it spawns a **Solver**. The **PM** is notified by the **Solver** on the completion of the initialization via the corresponding **LB**. Then the **PM** checks for available **Solver**s and manages their operation. After the initialization, the **PM** behaves as the "master" and the **Solver**s are the "slaves". As long as the subproblem pool of the **PM** is not empty and more than one idle **Solver** exist, the **PM** selects subproblems and assigns them to the **Solver**s. When the **Solver** receives a subproblem, it evaluates it. If an improved solution is found, the **Solver** sends it to the **PM** and broadcasts the objective value to the PVM-LBG. Each **LB** receives the objective value, and sends it to the corresponding **Solver**. If new subproblems are generated, they are also sent to the **PM**.

### 2.3.2 "Fully-Distributed(FD)" mode

The initialization sequence is almost the same as the one in the MS mode. The only differences are in the following two points: (1)the **PM** does not have a subproblem pool and a root problem is just maintained by the **PM** and when the first **Solver** is available, the problem is assigned to this **Solver** as a part of the initialization data, and (2)the **PM** does not manage available **Solver**s.

In the **Solver**, a sequential branch-and-bound algorithm is operated obtaining subproblems from a local subproblem pool and putting newly generated subproblems into it. In order to avoid imbalance of workload among the **Solver**s, each **Solver** may send or receive subproblems at the end of each evaluation. When the evaluation is completed, the **Solver** sends its own local pool states (the number of subproblems, their best bound values etc.) to the corresponding **LB** and checks if any messages have arrived or not. Upon receiving a subproblem, the **Solver** puts it into the local pool and sends an acknowledgment message to the sender. After, receiving an instruction message from the corresponding **LB** to send a subproblem to a specified **Solver**, the **Solver** follows the instruction. Note that the **Solver** does not care about the origin or destination decided by the corresponding **LB**. This relation between the **Solver** and the corresponding **LB** described above, is graphically depicted in Figure 4.

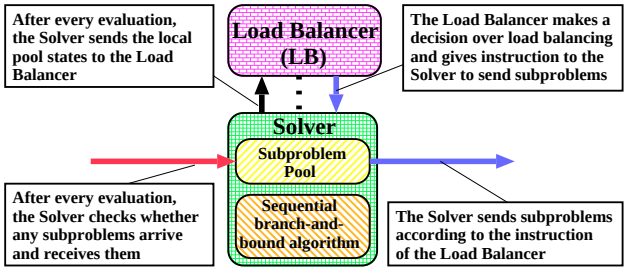


**Figure 4. Interaction between the Solver and the Load Balancer**

When a **LB** recognizes that the number of subproblems in the corresponding **Solver** is less than the *threshold value* specified as a parameter, the **LB** searches for a *partner* which is the **LB** whose corresponding **Solver** is the partner in the pairwise load balancing. Since each **LB** knows the pool states of the corresponding **Solver**, it can search for a partner in cooperation with other **LB**s. Therefore, during the search process, the underlying evaluation in the **Solver**s are performed concurrently. Although the search is initiated usually from a **LB** of a receiver side, it is started from one of a sender side, when the subproblem pool of a **Solver** is filled up to its capacity.

The **LB** which starts searching for a partner, will be called *initiator*. An initiator broadcasts a request for a partner. When the **LB**s receive this request, each of them replies

with its states immediately. The initiator considers the group of **LB**s replies of which were communicated to the initiator, as a cluster called the *load balancing group(LBG)*. The **LB**s replies sent to the initiator are checked after every iteration, or one second later if no subproblem exists in the **Solver**. Therefore, the LBG extends with time. An **LB** is selected as a partner from the group of **LB**s in the LBG,if the following conditions are met: (1)The state of the corresponding **LB** is *FREE*, which means that the **LB** is not an initiator and the corresponding **Solver** is not performing pairwise load balancing, (2)The number of subproblems stored in the **Solver** is greater than the average number of subproblems in the LBG. These procedures mentioned above are performed on each **LB** independently. The case where the procedures on several **LB**s are performed asynchronously is presented in Figure 5. This strategy usually searches for a near partner, where the distance concept is defined in terms of the response time. Subproblems keep being transferred until the number of subproblems in the **Solver** managed by the initiator is equal to the average number of subproblems in the LBG.

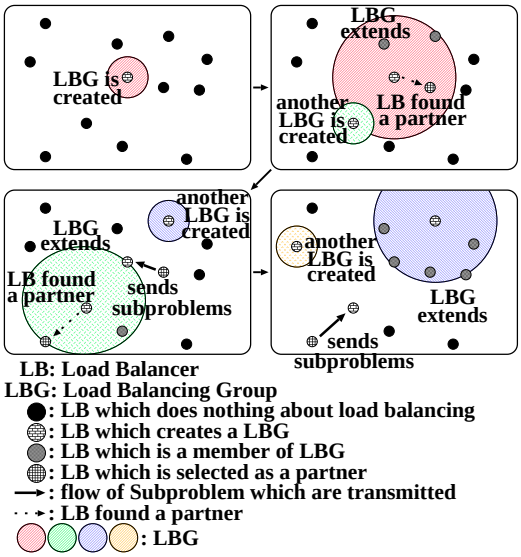


**Figure 5. Searching procedure for partner**

The **Solver** can assume two different states: "idle"(no subproblems) or "full" (the subproblem pool in the **Solver** is filled up to its capacity) or it can also be canceled. The **PM** is notified on any change in the **Solver** state. When the **PM** detects that the states of all **Solver** are "idle" or "full", the execution is terminated.

#### 2.3.3 "Master-Slave to Fully-Distributed(MStoFD)" mode

The PUBB starts execution with the same initialization sequence and internal architecture as that in the MS mode. When the number of subproblems maintained by the **PM** becomes greater than a certain value specified as a parameter, the internal architecture is modified to the one in the FD mode. As soon as this condition is detected, the **PM** broadcasts the "change architecture" message to PVM-LBG and each **Solver** receives the message via the corresponding **LB**. Upon receiving the message, the **Solver** sends an acknowledgment message to the **PM** and starts placing recently generated subproblems into the local pool. When the **PM** recognizes that all the **Solver** states have been changed, it selects a subproblem from the pool using best bound order and distributes it to a different **Solver** one by one using the "round robin" scheme. Note that while these procedures are performed, the **Solver**s keep evaluating subproblems.

## 3. Performance evaluation

Computational experiments on the TSP were conducted and the results are presented here. We used the algorithm for solving the TSP developed by M.Held and R.Karp and based the bounding rule on the minimum 1-tree relaxation and sub-gradient method [5, 6]. Though many powerful algorithms have been proposed for the TSP, this method is enough to evaluate the performance of the algorithm in parallelizations. In these experiments, problem cases from TSPLIB[12] were used.

The workstations operated were an IBM RS/6000 Model 25T with a PowerPC601 at 66MHz and 64M bytes memory. The maximum number of networked workstations used was 101 and only one **Solver** was ran on each machine. The **PM** was hosted by a workstation where no **Solver** and no other processes ran. Therefore, the number of the workstations used for the experiments was one more than the number of **Solver**s. No other user process ran on any workstation during the experiments.

### 3.1. Preliminary experiments

In order to show the behavior of the algorithm in solving the TSP and to investigate the parallelization effectiveness, the computational results of the sequential algorithm are also presented here. We solved ten problem cases using the sequential algorithm and applying the best bound first and the hybrid selection rules, because these rules for the algorithm showed good performance in our previous work. Table 1 shows the computing time for each problem case. The ending parts of the problem case names indicate the number of TSP cities.

The application of the hybrid selection rule was not always faster than that of the best bound first selection. However, the mean evaluation time of each subproblem was short when the hybrid selection rule was applied, because each evaluation used knowledge produced by the previous

evaluation in the depth first phase. This is an advantage of the hybrid selection rule.

**Table 1. Computing time in the sequential algorithm (sec)**

| Prob. | Best Bound Selection | | | | Hybrid Selection | | | |
|---|---|---|---|---|---|---|---|---|
| | Comp. Time | Ev.Time of a Subproblem Mean(S.D.) | Max | Min | Comp. Time | Ev.Time of a Subproblem Mean(S.D.) | Max | Min |
| berlin52 | 343.19 | 2.37(1.44) | 5.64 | 0.32 | 32.70 | 2.05(1.89) | 5.88 | 0.36 |
| dantzig42 | 16605.20 | 1.35(0.63) | 3.76 | 0.01 | 6539.66 | 0.93(0.85) | 3.78 | 0.02 |
| eil101 | 4859.47 | 6.20(3.55) | 21.39 | 0.57 | 5850.59 | 5.54(4.20) | 21.75 | 0.17 |
| eil51 | 717.64 | 1.32(0.78) | 4.82 | 0.07 | 1120.83 | 1.32(0.84) | 5.30 | 0.03 |
| eil76 | 447.13 | 4.43(0.86) | 8.68 | 1.89 | 618.43 | 3.37(2.58) | 11.72 | 0.11 |
| gr48 | 26152.50 | 1.221(0.95) | 4.73 | 0.06 | 23605.86 | 1.16(0.98) | 4.91 | 0.02 |
| rat99 | 687.98 | 6.73(1.07) | 8.72 | 1.82 | 2027.10 | 6.01(2.95) | 15.43 | 0.15 |
| rd100 | 50463.03 | 7.98(5.62) | 22.16 | 0.18 | 36573.59 | 6.90(5.68) | 22.17 | 0.16 |
| st70 | 14209.28 | 3.598(0.85) | 9.55 | 0.39 | 15245.44 | 2.80(2.37) | 10.28 | 0.06 |
| swiss42 | 23.07 | 1.13(0.75) | 2.87 | 0.29 | 23.13 | 0.96(0.92) | 3.48 | 0.06 |

### 3.2. Performance comparison among several schemes

In the following results, we demonstrate the performance of each control scheme. We experimented only four cases: dantzig42, gr48, rd100 and st70 in parallel with 2,3,5,7,10,20,40,70 and 100 **Solver**s. Other cases are generally solved fast, do not have to be solved in parallel. Experiments were conducted with three control schemes: central(MS), distributed(FD) and mixed(MStoFD). In the MStoFD mode, the control scheme was changed when the number of subproblems maintained by the **PM** exceeded over the following number:$(\text{threshold value} + 1) \times (\text{the number of } \textbf{Solver}\text{s})$. This setting of parameter causes no initial imbalance among **Solver**s turn in the distributed control. The best first and the hybrid selection rules were also applied. Five runs were repeated with the same parameter specifications in the same computing environment. Figures 6, 8, 10 and 12 computing time vs. the number of **Solver**s used. Figures 7, 9, 11 and 13 the number of subproblems that were evaluated vs. the number of **Solver**s used. The results of the use of sequential algorithms are also plotted in the figures, for comparison reasons. All the results except the trials with the sequential algorithm are plotted using the mean value of five repeated runs. The shapes of plots and lines in Figures 7, 9, 11 and 13 are the same as those in Figures 6, 8, 10 and 12. *LinearSpeedup* plots are drawn using the following function: $LinearSpeedup(x) = (\text{Computing time in the sequential algorithm})/x$

The application of the hybrid selection rule often showed better performance than that of the best bound first selection. This performance is observed not only in the short computing time but also in the small number of evaluated subproblems. In addition, in the parallelization with many **Solver**s, the performance seems to have plenty of advantages. The reason is that, since a depth-first search is used in the hybrid selection rule, solutions which can improve the incumbent value are more likely to be found earlier compared to the best bound first selection rule. Hence, the number of redundant subproblems evaluated with the hybrid rule will be less than the one in the best bound first rule. Moreover, a parallel application of the hybrid selection rule seems to enhance the advantages.

The central control in the MS mode often showed good performance within applications of the same selection rule. Especially, when the hybrid selection rule was used, superlinear speedup was achieved. The distributed control case in the FD mode with the best bound first selection rule showed that our load balancing strategy is poor in terms of "quality", because it increased the number of evaluated subproblems very much in gr48 and st70. However, substituting the best bound first with the hybrid selection rule improved the situation. Moreover, the mixed control in the MStoFD mode improved the situation even more. In the setting of the parameter in the MStoFD mode, the control scheme was changed in the beginning of the execution. It is likely that imposing central control in the beginning of the algorithms is important, because the performance of the poor load balance case in the FD mode was greatly improved in the MStoFD mode (especially the results of gr48 and st70).

Table 2 shows the efficiency vs. the number of **Solver**s used. *Efficiency* can be defined as: $Efficiency(x) =$ ( Computing time in sequential ) / ( ( Computing time in parallel with $x$ **Solver**s ) $\times x$ ). The problem cases are arranged in order of the number of subproblems evaluated by the sequential algorithms with each selection rule (the number is indicated under the problem case name). It is not clear, but the advantages of FDtoMS mode is observed with increasing the number and with adding **Solver**s used.

**Table 2. Efficiency vs. # of Solvers used**

| Prob. | Mode | #=1 | #=2 | #=3 | #=5 | #=7 | #=10 | #=20 | #=40 | #=70 | #=100 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| st70 | FD | 1.00 | 0.76 | 1.14 | 0.96 | 0.72 | 0.59 | 0.15 | 0.13 | 0.08 | 0.06 |
| [Best] | MS-FD | 1.00 | 0.77 | 1.14 | 0.81 | 0.66 | 0.48 | 0.44 | 0.33 | 0.27 | 0.32 |
| (3923) | MS | 1.00 | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 | 0.96 | 0.91 | 0.85 | 0.74 |
| rd100 | FD | 1.00 | 0.77 | 1.20 | 1.09 | 1.00 | 0.92 | 0.94 | 0.76 | 0.53 | 0.38 |
| [Hybrid] | MS-FD | 1.00 | 0.98 | 1.54 | 0.75 | 0.74 | 1.07 | 1.01 | 0.92 | 0.78 | 0.57 |
| (5269) | MS | 1.00 | 1.24 | 1.14 | 0.80 | 0.79 | 1.22 | 1.06 | 1.15 | 0.99 | 1.03 |
| st70 | FD | 1.00 | 0.95 | 0.93 | 0.90 | 0.85 | 0.88 | 0.69 | 0.59 | 0.36 | 0.30 |
| [Hybrid] | MS-FD | 1.00 | 0.81 | 0.75 | 0.94 | 0.77 | 1.01 | 0.77 | 0.77 | 0.61 | 0.34 |
| (5418) | MS | 1.00 | 0.80 | 0.76 | 0.88 | 0.73 | 0.72 | 0.90 | 1.00 | 1.27 | 0.93 |
| rd100 | FD | 1.00 | 0.89 | 0.72 | 0.94 | 0.85 | 0.86 | 0.69 | 0.66 | 0.43 | 0.37 |
| [Best] | MS-FD | 1.00 | 1.09 | 0.93 | 1.08 | 0.64 | 0.85 | 0.68 | 0.70 | 0.66 | 0.50 |
| (6280) | MS | 1.00 | 1.00 | 1.00 | 0.99 | 0.99 | 0.99 | 0.97 | 0.96 | 0.89 | 0.86 |
| dantzig42 | FD | 1.00 | 0.80 | 0.68 | 0.64 | 0.58 | 0.60 | 0.65 | 0.53 | 0.37 | 0.18 |
| [Hybrid] | MS-FD | 1.00 | 0.86 | 0.91 | 0.69 | 0.76 | 0.79 | 0.66 | 0.55 | 0.43 | 0.28 |
| (6966) | MS | 1.00 | 1.08 | 0.80 | 0.77 | 0.85 | 0.78 | 0.82 | 0.64 | 0.57 | 0.36 |
| dantzig42 | FD | 1.00 | 0.85 | 0.65 | 0.74 | 0.77 | 0.60 | 0.67 | 0.56 | 0.52 | 0.34 |
| [Best] | MS-FD | 1.00 | 0.85 | 0.88 | 0.83 | 0.80 | 0.75 | 0.78 | 0.79 | 0.70 | 0.50 |
| (12217) | MS | 1.00 | 0.99 | 0.98 | 0.98 | 0.98 | 0.96 | 0.95 | 0.91 | 0.61 | 0.44 |
| gr48 | FD | 1.00 | 0.92 | 0.82 | 0.81 | 0.97 | 0.93 | 0.98 | 0.82 | 0.74 | 0.51 |
| [Hybrid] | MS-FD | 1.00 | 1.12 | 0.82 | 0.83 | 0.71 | 0.81 | 0.86 | 1.00 | 0.85 | 0.66 |
| (20292) | MS | 1.00 | 0.94 | 1.08 | 1.08 | 0.92 | 1.00 | 1.03 | 1.07 | 0.89 | 0.71 |
| gr48 | FD | 1.00 | 1.07 | 0.84 | 0.74 | 0.57 | 0.64 | 0.50 | 0.40 | 0.33 | 0.34 |
| [Best] | MS-FD | 1.00 | 0.90 | 1.02 | 0.99 | 0.77 | 0.81 | 0.85 | 0.76 | 0.65 | 0.60 |
| (21281) | MS | 1.00 | 0.99 | 0.99 | 0.99 | 0.98 | 0.99 | 0.98 | 0.94 | 0.70 | 0.50 |

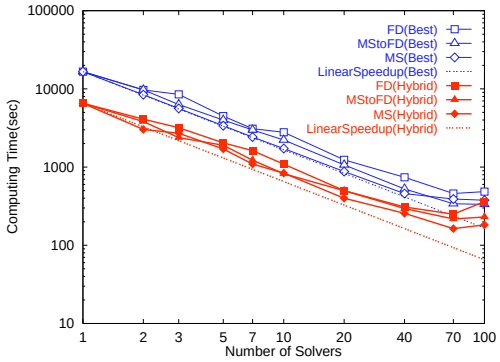


**Figure 6. Time results for dantzig42**

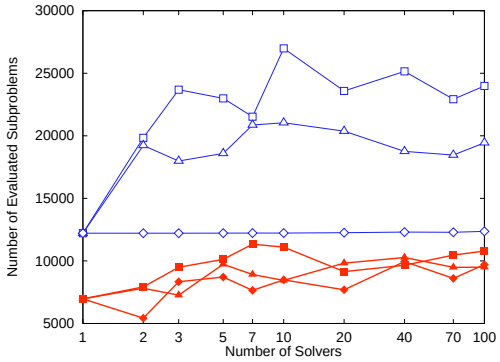


**Figure 7. Number results for dantzig42**

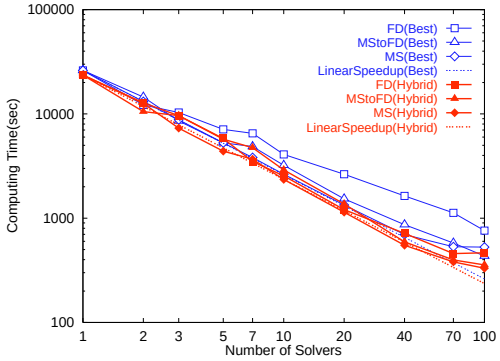


**Figure 8. Time results for gr48**

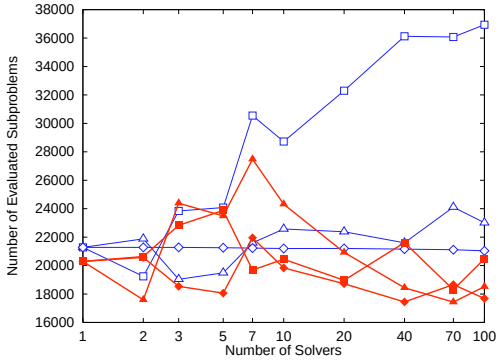


**Figure 9. Number results for gr48**

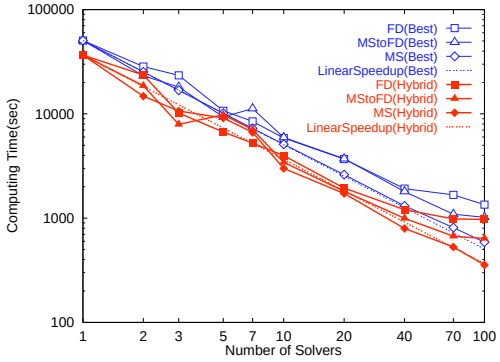


**Figure 10. Time results for rd100**

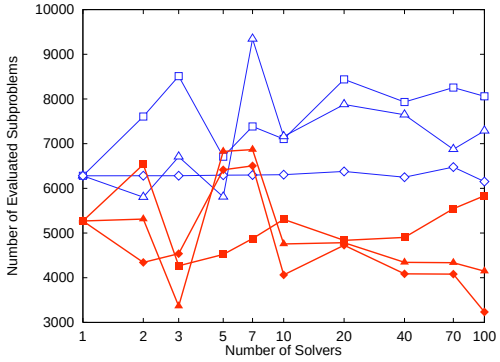


**Figure 11. Number results for rd100**

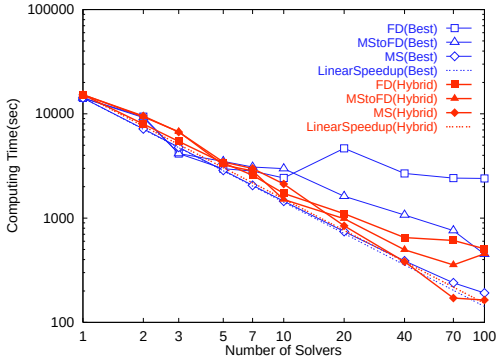


**Figure 12. Time results for st70**

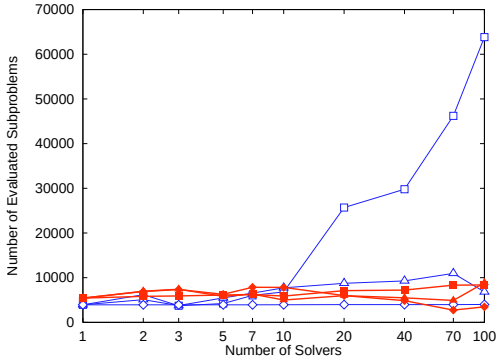


**Figure 13. Number results for st70**

## 4. Concluding remarks

In this paper, the mixed control scheme was proposed and examined and a load balancing strategy was implemented on the renewed PUBB. The most important result presented is the power of the hybrid selection rule in the real world cases. It is known that the number of subproblems visited in a sequential branch-and-bound algorithm is minimized when the best bound first selection is applied. Therefore, implementations have often been applied to the best bound selection rule[7, 8, 11]. However, the idea is based on the assumption that the evaluation of each subproblem takes up the same computing time. Our results showed that the evaluation time varies significantly(see Table 1) in practice. Based on the experimental facts, the advantages of the hybrid selection rule, both in the central control and in the distributed control scheme, were shown.

The results presented in this paper do not show that mixed control in the MStoFD mode is better than central control in the MS mode. However, it is important to know that the instances solved in the parallel algorithm can also be solved in the sequential one. These cases were selected for a comparison between the sequential and the parallel. The central control scheme has its memory bottleneck. Therefore, an instance which cannot be solved in a sequential algorithm because of the memory bottleneck problem, often can neither be solved in parallel when the maximum available subproblem pool size is fixed. In order to increase the solvable instances, the distributed control schemes are especially useful in practice. For the limitation of computing environment, we could not show the the advantages of mixed control in the MStoFD mode. However the efficiency analysis provided in Table 2 shows its potential.

In general, a proper selection of the control scheme is determined by the characteristics of the solving algorithm for the target problem and the scale of the parallel computing environment used. When the evaluation workload for a subproblem is very low, the access convergence bottleneck in the central control scheme becomes a serious problem. In this case, the use of the distributed control scheme has to be attempted. If the evaluation workload for a subproblem is large enough and the algorithm is applied on a few processors, the central control scheme seems to be the most appropriate. Therefore, several control schemes are necessary in a generalized system in order to have a wide application.

## Acknowledgments

Our special thanks are due to Yiannis Anagnostakis for reading the entire text in its original form. Research support from the Information Processing Center at the Science University of Tokyo is highly acknowledged by the authors. The authors are grateful to the anonymous referees for careful reading of the manuscript and helpful comments.

## References

[1] R. Corrêa. A parallel formulation for general branch-and-bound algorithms. In A.Ferreira and J.Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems*, volume 980 of *Lecture Notes in Computer Science*, pages 395–409. Springer-Verlag, 1995.

[2] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Jornal of Parallel and Distributed Computing*, 7:279–301, 1989.

[3] J. Eckstein. Control strategies for parallel branch-and-bound. In *Proceedings of Super-computing '94*, pages 41–48, 1994.

[4] G. Gendron and T. G. Crainic. Parallel branch-and-bound algorithms:survey and synthesis. *Operations Research*, 42(6):1042–1066, 1994.

[5] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[6] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Partii. *Mathematical Programming*, 1:6–25, 1971.

[7] R. Lüling and B. Monien. Two strategies for solving the vertex cover problem on transputer network. In M.Raynal and J.C-.Bermond, editors, *Distributed Algorithms*, Lecture Notes in Computer Science, pages 160–170, 1989.

[8] R. Lüling and B. Monien. Load balancing for distributed branch & bound algorithms. In *Proceedings of the 6th International Parallel Processing Symposium*, pages 543–549, 1992.

[9] G. P. McKeown, V. J. Rayward-Smith, and S. A. Rush. Parallel branch-and-bound. In L.Kronsjö and D.Shumsheruddin, editor, *Advances in Parallel Algorithms*, Advanced topics in computer science, pages 111–150, 1992.

[10] P. M. Paradalos and G. P. Rodgers. Parallel branch and bound algorithms for quadratic zero-one programs on the hypercube architecture. *Annals of Operations Research*, 22:271–292, 1990.

[11] M. J. Quinn. Analysis and implementation of branch-and-bound algorithms on a hypercube multicomputer. *IEEE Transactions on Computers*, 39(3):384–387, 1990.

[12] G. Reinelt. Tsplib - a traveling salesman problem library. In *ORSA Journal on Computing*, volume 3, pages 376–384, 1991.

[13] Y. Shinano, M. Higaki, and R. Hirabayashi. A generalized utility for parallel branch and bound algorithms. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, pages 392–401, San Antonio, Texas, 1995.

[14] H. W. J. M. Trienekens. *Parallel Branch and Bound Algorithms*. Phd thesis, Erasmus Universiteit, Rotterdam, 1990.

[15] S. Tschöke, R. Lüling, and B. Monien. Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network. In *Proceedings of the 9th International Parallel Processing Symposium*, Santa Barbara, California, 1995(to appear).