A Parallel Branch-and-Bound Method for the Traveling Salesman Problem and Its Implementation on a Network of PCs

Noritaka Shigei[1], Mitsunari Okumura[2] and Hiromi Miyajima[1]
[1]Department of Electrical and Electronics Engineering,
Kagoshima University, Japan
E-mail: shigei@eee.kagoshima-u.ac.jp
[2]Department of Computer Science, Shimane University, Matsue, Japan

Abstract: This study presents a parallel branch-and-bound (PBAB) method for traveling salesman problem (TSP). The PBAB method adopts intermediate form of central control and distributed control in terms of the lightness of the master process's role. Compared with fully distributed control, the control scheme involves less concentration of communication on the master. Moreover, in order to reduce the influence of communication, the worker is composed of a computation thread and a communication thread. The multithreadness realizes the almost blocking free communications on the master. We implement the proposed PBAB method on a network of PCs, which consists of one master and up to 16 workers. We experiment five TSP instances. The results shows that the efficiency increases with the problem size.

1. Introduction

The traveling salesman problem (TSP) is the most popular NP-hard problems[2]. Parallel branch-and-bound (PBAB) methods have been considered to reduce the solution time[5], [4]. The key to achieve good efficiency in the PBAB methods is dynamical load balancing. In general, the dynamical load balancing schemes are classified into two types: *central control* and *distributed control*. In the central control schemes, a master process (shortly called master) knows the states of every worker processes (shortly called workers), and assigns a subproblem to idle processes. The processing of the master, however, is a bottleneck when the number of processors is large. On the other hand, in the distributed control schemes, every processors cooperatively assign a subproblem to idle processors. A processor in general is allowed to cooperate with the limited processors, because the non-limited cooperation makes a network bottleneck. Tschöke et al. [5] considered distributed control, and Shinano et al. [4] considered hybrid control schemes.

In order to achieve high efficiency in the PBAB methods, it is important to reduce the following three factors: (1) *idleness of processors*, (2) *communication overhead* and (3) *search overhead*. However, among the three factors, there exist tradeoffs. Reducing *idleness of processors* and *search overhead* involves increasing *communication overhead*, because the reduction requires frequent communication between the processors.

This study presents a PBAB method that reduces *communication overhead* independent of *idleness of processors* and *search overhead*. The PBAB method adopts an intermediate form of central control and distributed control in terms of the lightness of the master's role. The main roles of the master are (1) memorizing the states of the workers, (2) directing to divide subproblems, and (3) supervising the current upper bound through all subproblems. After directing a division, the division process is progressed between a pair of workers. Compared with fully distributed control, the control scheme involves less concentration of communication on the master. Moreover, in order to reduce the influence of communication, the worker processors use a computation thread and a communication thread. The multithreadness realizes the almost blocking free communications on the master. We implement the proposed PBAB method on a network of PCs, which consists of one master and up to 16 workers. We experiment five TSP instances. The results shows that the efficiency increases with the problem size. Further, for an instance, the proposed method achieves the better performance than Shinano's methods[4].

2. Preliminaries

2.1 Traveling Salesman Problem

The traveling salesman problem (TSP) is a problem that finds the shortest path visiting all the N cities just once. This paper considers the symmetrical TSP, in which $c_{ij} = c_{ji}$, where c_{ij} is the cost of edge from i-th city to j-th city. Let $M = \frac{N(N-2)}{2}$. Let c_e be the cost of e-th edge ($e \in \{1, 2, \cdots M\}$). The symmetrical TSP P is to find a $(0, 1)$ vector $X = (x_1, x_2, \cdots, x_N)$ such that:

$$\begin{aligned} \min \quad & \sum_{e=1}^{M}\sum_{e=1}^{M} c_e x_e \\ \text{s.t.} \quad & \sum_{e\in\delta(i)} x_e = 2 \quad \forall i \in \{1, 2, \cdots, N\} \\ & \sum_{e\in\delta(S)} x_e \geq 2 \quad \forall S \subset \{1, 2, \cdots, N\} \\ & x_e \in \{0, 1\} \quad \forall e \in \{1, 2, \cdots, N\}, \end{aligned}$$

where $x_e = 1$ means that the e-th edge is selected, and $\delta(S)$ is the set of selected edges which connect S and $\{1, 2, \cdots, N\} - S$.

2.2 Branch-and-Bound Method

Branch-and-bound (BAB) methods have been widely used for solving the TSP. The BAB algorithm contains two main procedures: *branching* and *bounding*. In branching procedure, a problem is divided into several subproblems. In bounding procedure, a lower bound for each subproblem is calculated, and the subproblems whose lower bound is higher than the current upper bound are discard.

```
SBAB (problem $P$) {
    $\boldsymbol{P} \leftarrow \{P\}$;
    $u \leftarrow \infty$;
    while ($\boldsymbol{P} \neq \emptyset$) {
        Calculate a lower bound $l_p$ for a problem $p \in \boldsymbol{P}$;
        $\boldsymbol{P} \leftarrow \boldsymbol{P} - p$;
        if ($l_p < u$) {
            if ($l_p$ is the cost of a tour) {
                /* bounding */
                $u \leftarrow l_p$;
                for all $p \in \boldsymbol{P}$ st $l_p \geq u, \boldsymbol{P} \leftarrow \boldsymbol{P} - p$;
            } else {
                /* branching */
                Select an edge $e \in E - (R \cup F)$, where $E$ is
                the set of all edges and $p = P(R, F)$;
                $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup \{P(R \cup e, F), P(R, F \cup e)\}$;
            }
        }
    }
    The tour for upper bound $u$ is the solution;
}
```

Figure 1. Sequential BAB

Let $P(R, F)$ be a subproblem of a problem P or a subproblem. R is a set of required edges, and F is a set of forbidden edges. In a feasible solution $X = (x_1, \cdots, x_N)$ of $P(R, F)$, for any $e \in R$; $x_e = 1$, and for any $e \in F$; $x_e = 0$.

Our parallel BAB (PBAB) method presented in the next section is a parallelization of a sequential BAB (SBAB) algorithm based on 1-tree relaxation[1]. In the BAB algorithm, a lower bound l_p for a subproblem p is obtained by calculating a minimum 1-tree t for p. When t is a tour, l_p is the upper bound for p. A pseudo-code for the sequential BAB (SBAB) algorithm is shown in Fig. 1.

3. Parallel BAB Method

The proposed PBAB method uses one *master* process and more than one *worker* processes. At most one process runs on a processing node. We assume that the method runs on a distributed memory parallel computers based on message passing communication such as network of PCs.

3.1 Master Process

The main roles of the master process are (1) memorizing the states of the workers, (2) directing to divide subproblems, and (3) supervising the current upper bound through all subproblems. The master operates as follows:

Master:

1. The master memorizes the states of the workers, which are "*idle* or *active*", "the number of branchings" and "the lower bound". The states are periodically informed from the workers.
2. When an idle worker requests a subproblem, according to the memorized states the master selects an *active* worker which will divide a subproblem. Then the master requests the selected worker to divide a subproblem to the idle worker.

```
computation_thread() {
    $\boldsymbol{P} \leftarrow \emptyset$;
    $u \leftarrow \infty$;
    $rq \leftarrow$false;
    while (true) {
        if ($\boldsymbol{P} = \emptyset$) {
            Inform request for subproblem to master;
            if (a subproblem $p$ is received) {
                Inform reception to master;
                $\boldsymbol{P} \leftarrow \boldsymbol{P} + p$;
            } else if (termination is informed) {
                Return acknowledgement to master;
                Terminate worker process;
            }
        }
        Calculate a lower bound $l_p$ for a subproblem $p \in \boldsymbol{P}$;
        if ($l_p$ is the cost of tour) {
            Inform $l_p$ to master as an upper bound;
        } else {
            /* branching */
            Select an edge $e \in E - (R \cup F)$, where $E$ is
            the set of all edges and $p = P(R, F)$;
            $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup \{P(R \cup e, F), P(R, F \cup e)\}$;
        }
        /* bounding */
        for all $p \in \boldsymbol{P}$ st $l_p \geq u, \boldsymbol{P} \leftarrow \boldsymbol{P} - p$;
        if ($rq$ =true and $\boldsymbol{P} \neq \emptyset$) {
            Send a subproblem $p \in \boldsymbol{P}$;
            $\boldsymbol{P} \leftarrow \boldsymbol{P} - p$;
            $rq \leftarrow$ false;
        }
        Inform the number of branchings and a lower bound;
    }
}
```

Figure 2. Computation thread

3. When a worker informs a new upper bound, the master informs the upper bound to every workers.
4. When all workers become *idle*, the master notifies termination of the task to them.

Since the master process can communicate with at most one worker process at a time, multiple accesses from workers may cause congestion. However, our master process exchanges less amount of messages with worker processes than a typical master process. The master process has no subproblem, but all the subproblems are distributed over all the worker processes. After the master directs a division, the division process is progressed between a pair of workers. Therefore, the length of messages treated by the master is not proportional to the problem size N. It may reduces congestion on the master process.

3.2 Worker Process

The master may send a division request and a new upper bound to a worker. Checking periodically whether the master try to send messages or not can degrade the performance of PBAB methods. A long checking period leaves

```
communication_thread() {
  while (true) {
    if (upper bound $u_{new}$ is received)
      $u \leftarrow u_{new}$;
    else if (work division is re-
quested)
      $rq \leftarrow$true;
  }
}
```

Figure 3. Communication thread

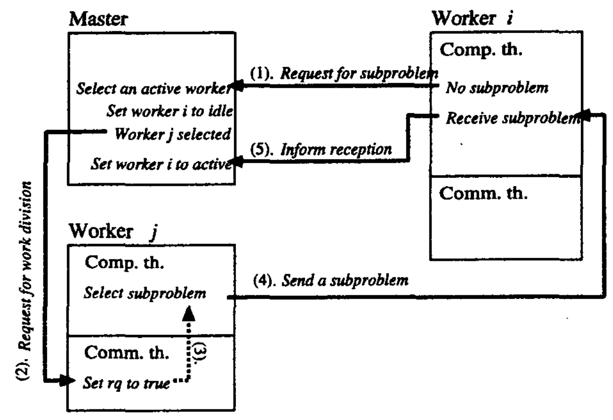

Figure 4. A work transferring process: worker i requests a subproblem and receives a subproblem from worker j.

workers idle for a long time, and a short checking period increases overhead. To overcome this difficulty, the worker process is composed of two types of thread: *computation thread* and *communication thread*. The computation thread mainly solves a subproblem using a BAB algorithm. The communication thread devotes to receiving messages from the master. Therefore the master sends the messages in almost blocking free. The messages received by the communication thread are passed to the computation thread via shared memory. Pseudo-codes for the two threads are shown in Fig.2 and Fig.3. Fig.4 shows the simplest work transferring process. If unlike the case of Fig.4 worker j becomes idle before completing the work division (worker j requests a subproblem), the master re-requests another active worker to divide a subproblem.

3.3 Division of Subproblems

As the calculation progresses, the number of subproblems increases. If a small subproblem is transferred to an idle worker, the worker will soon finish solving the subproblem. From the other viewpoint, in such a case, it is expected to find a new upper bound and to reduce the search overhead. It seems to exist a tradeoff. From the observation, we consider dividing a subproblem from a fixed branching level L ($0 \le L \le 1$). Level $L = 0$ means a first branching level, and level $L = 1$ means a last branching level. In the next section, effective branching levels are experimentally investigated.

Table 1. The computation times on a sequential computer

Name	# of cities	Time
RD50	50	47.61 sec
RD70	70	562.77 sec
ST70	70	258.57 sec
RD90	90	4067.80 sec
RD100	100	2350.48 sec

4. Experiments and Observations

We implement the PBAB method on a network of PCs by using a thread-safe MPI. The network of PCs consists of 1 master and up to 16 workers, each of which comprises single Intel 450MHz Celeron processor. The PCs are connected via a First Ethernet switch. We experiment five TSP instances: rd50, rd70, rd90, st70 and rd100. A number in the instance names indicates the number of cities. Instances rd50, rd70 and rd90 are original random instances. Instances st70 and rd100 are from the TSPLIB[3]. Their computation times on a sequential computer are summarized in Table 1. Table 1 implies that the order in difficulty is RD90, RD100, RD70, ST70 and RD50.

Table 2 and Figure 5 show the experiment results in terms of the speedup. The speedup is *(the parallel solution time)/(the sequential solution time)*. Table 2 shows the results for our original instances rd50, rd70 and rd90. According to the results, the best branching level L_{best} depends on the problem, and it seems to be $0 \le L_{best} \le 0.3$. The efficiency of our method increases with the instance size. Moreover, for rd90, our method achieves super-linear speedup. In Figure 5, our method and Shinano's method are compared for instances st70 and rd100. For rd100, which is more difficult than st70, our method outperforms Shinano's one. From the experiment results, we can find that the efficiency of our method increases with the difficulty of the problem.

5. Conclusion

This study presents a PBAB method for TSP and implements the method on a network of PCs. The PBAB method adopts an intermediate form of central control and distributed control in terms of the lightness of the master's role. The worker process is composed of two threads. It realizes that the master sends the messages in almost blocking free. The performed simulation results show that the efficiency of our method increases with the difficulty of the problem.

References

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

[2] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy-Kan, and D.B. Shmoy. *The traveling salesman problem.* John Wiley & Sons, Chichester, 1985.

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

[4] Y. Shinano, K. Harada, and R. Hirabayashi. Control

Table 2. The speedup for rd50, rd70 and rd90.

Instance Name	L	Number of Workers							
		2	4	6	8	10	12	14	16
rd50	0	2.10	3.43	4.27	4.75	5.14	5.58	5.69	5.91
	0.1	1.79	2.77	3.77	4.18	4.49	4.86	5.19	5.36
	0.3	1.87	4.63	5.78	6.86	7.67	7.90	7.68	9.10
	0.6	1.58	2.84	3.91	4.56	4.54	5.67	5.47	6.41
rd70	0	2.60	4.34	6.77	8.17	9.49	10.21	11.96	12.90
	0.1	2.23	4.08	5.56	8.01	9.69	10.71	11.50	12.31
	0.3	3.01	5.26	8.14	9.67	10.63	11.56	12.16	12.52
	0.6	2.21	4.07	6.28	7.78	9.86	11.03	10.32	10.69
rd90	0	5.86	11.56	17.73	23.56	29.24	32.34	34.97	37.31
	0.1	2.25	10.73	19.57	25.05	29.16	32.19	35.50	38.96
	0.3	3.43	9.06	13.33	18.19	22.25	25.31	29.36	30.93
	0.6	2.85	6.66	10.78	11.92	16.83	20.52	23.61	27.87

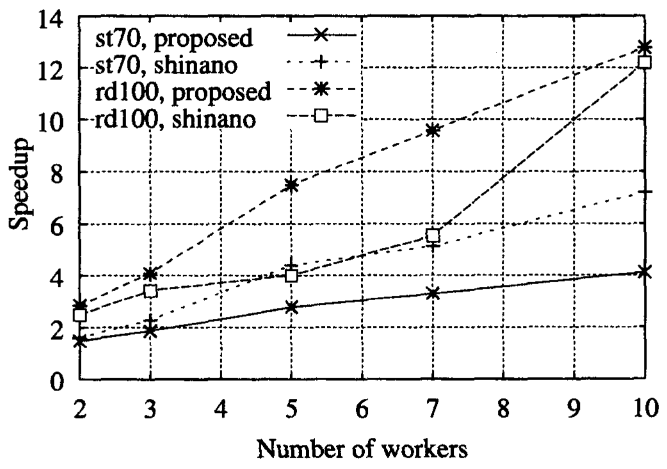

Figure 5. The speedup for st70 and rd100.

schemes in a generalized utility for parallel branch-and-bound algorithms. In *Proc. of IPPS'97*, 1997.

[5] S. Tschöke, R. Lüling, and B. Monien. Solving the traveling salesman problem with a parallel branch-and-bound algorithm on a 1024 processor network. In *Proc. of IPPS'95*, 1995.