A Comparison of Multiprocessor Scheduling Algorithms without Communication Costs

ReaKook Hwang* and Mitsuo Gen**

* Graduate School of Information, Production & Systems, Waseda University

rkhwnag@ruri.waseda.jp

** Graduate School of Information, Production & Systems, Waseda University

gen@ruri.waseda.jp

Abstract - **Efficient assignment and scheduling for tasks to multiprocessors is one of the key issues in the effective utilization of multiprocessor systems. This problem is well known as an NP-hard problem and many heuristic methods for finding a optimal or suboptimal schedule reported.**

This paper addresses the problem of scheduling to multiprocessors represented as directed acyclic task graph (DAG) without communication costs to fully connected multiprocessors. We propose integrated algorithm, called pGA/SPF (Priority-based Genetic Algorithm with Shortest Processor First mapping method), where a priority-based genetic algorithm is improved with the introduction of some knowledge about the scheduling problem represented by the used of crossover and mutation genetic operators. And new mapping method, shortest processor first method, assigns the selected task to a processor that can minimize a task schedule efficiently. A comparison of proposed algorithm with previous reported scheduling algorithms is carried out. The proposed algorithm generates same or even better solutions than the previous algorithms in terms of the completion times of resulting schedules.

I. INTRODUCTION

The impressive proliferation in the use of multiprocessor systems these days in a great variety of applications is the result of many breakthroughs over the last two decades. These developments of multiprocessor systems are being used for several applications, including fluid flow, weather modeling, database systems, real-time, and image processing. The data for these applications can be distributed evenly on the processors of multiprocessor systems and maximum benefits from these systems can be obtained by employing and efficient task assignment and scheduling strategy.

Moreover, multiprocessor systems are increasingly being used to meet the high performance and intense computation needs of today's applications. To efficiently execute a program on a multiprocessor system, it is essential to solve a minimum execution time of multiprocessor scheduling problem [1]~[3], which determines how to assign a set of tasks to processors and in what order these tasks should be executed to obtain the minimum execution time.

The multiprocessor scheduling problem considered in this paper is based the deterministic model, that is, the execution time between tasks are represented and the directed acyclic graph (DAG) represents the precedence relations of the tasks of a multiprocessor system. Such problems are, however, extremely difficult to solve and are generally intractable; that is, it is well known that relaxed or simplified sub-problems constructed from the original scheduling problem by imposing a variety of restricting conditions still fall into the class of NP-hard problems and we made some strong assumptions that communication costs between task nodes are not considered.

Many heuristic based methods and approaches to the task scheduling problem have been proposed [4]~[6]. One of the major set of heuristics for task scheduling on multiprocessors is based on list scheduling [7]~[9]. It has been reported in [7][8] that the critical path list scheduling heuristic is within 5% of the optimal solution 90% of the time when the communication cost is ignored, while in the worst case any list scheduling is within 50% of the optimal solution.

Recently, an evolutionary approaches have been developed to solve the problem and GA-based approach can find better near optimal solution than list scheduling in most case [10]~[12]. A GA is a guided random search method where elements (called population) are randomly combined until some termination condition is achieved. In these GA-based scheduling problems, Hou *et al.* [1], Gen *et al.* [11] and Wang *et al.*[9] proposed pure genetic algorithms whose main difference lays in the way the individuals are code. Wang uses a dimensional matrix to code a schedule, while Hou et al. and Gen et al. proposed a coding based on strings. In all algorithms, no knowledge about the problem is taken into account, and the search is accomplished entirely at random.

In this paper, we demonstrated the impact of integrating knowledge- heuristic mapping method as Shortest Processor First (SPF) with priority-based genetic algorithm for encoding method into multiprocessor scheduling. We proposed algorithm, called pGA/SPF (Priority-based Genetic Algorithm/Shortest Processor First mapping method), where algorithm used the encoding method from Gen's [11] approach and heuristic mapping method that efficiently assign tasks to processors refers to the lowest absolute value of difference between the schedule length on the processor and the earliest start time of the task node.

We will compare our pGA/SPF with the ETF, HLFET and MCP algorithms which belong to the bounded number of processors scheduling algorithms to evaluate the effectiveness of algorithm by using simple DAG. In additionally, instead of testing our pGA/SPF algorithm with randomly generated instance, as in [1], we preferred to use as benchmark test for some relatively large graph: the Newton-Euler inverse dynamics equations task graph for the Stanford manipulator [2]. We will compare with other previous GA approaches by using this Stanford manipulator task graph.

II. Multiprocessor Scheduling Problems

In this paper, we use a well-accepted model of multiprocessor system and parallel programs. The system consists of m identical processors, $m > 1$, which are fully connected with each other via a reliable network. Each processor has its own memory, and can execute at most one task at a time and task preemption is not allowed. While computing, we made some strong assumptions that communication costs between task nodes are not considered. An example of a DAG consisting of 10 tasks is shown in Fig. 2 and a fully connected multiprocessor systems consisting of two processors ($m = 2$). However, the start node s and terminal node t are dummy node. Table 1 is data set of example DAG that includes the processing time and set of predecessors of each task.

We formulate the problem of multiprocessor scheduling that can be stated as finding a scheduling for a general DAG as shown in Fig. 2. to be executed on a multiprocessor system so that schedule length can be minimized. The multiprocessor system with m processors is to assign the computation tasks to processors in such a way that precedence relations are maintained and that all tasks are completed in the shortest possible time as given time chart (Fig. 1) with mathematical formulation:

$$\begin{aligned}
\min \quad & \{\max_j \{c_j x_{ij}\}\} \\
\text{s. t.} \quad & c_k - p_k \geq c_j, \quad T_j \prec T_k \\
& \sum_{j=1}^{n} p_j x_{ij} \leq t_{\max}, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, \ldots, n \\
& x_{ij} = 0 \text{ or } 1, \quad i = 1, \ldots, m, \; j = 1, \ldots, n
\end{aligned}$$

where

$$x_{ij} = \begin{cases} 1, & \text{if task } T_j \text{ is assigned to processor } P_i \\ 0, & \text{otherwise} \end{cases}$$

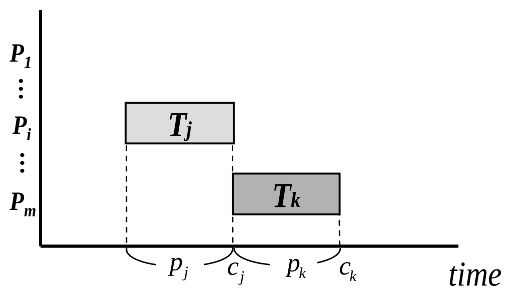


Fig. 1. Time Chart for DAG

And where $t_{\max} = \max_i \{t_i\}$, c_j is the completion time of task T_j, p_j the processing time of task T_j, ti the time required to process all tasks assigned to processor P_i, $Pred_j$ the set of predecessors of task T_j and $\prec$ represents a precedence relation; a precedence relation between tasks, $T_j \prec T_k$ means that T_j precedes T_k. We assume that the communication system is contention free and permits the overlap of communication started only after all dates have been received from predecessors. And duplication of the same task is not allowed.

III. pGA/SPF Approach

In this section, we present the Priority-based Genetic Algorithm (pGA) and Shortest Processor First (SPF) mapping method for multiprocessor scheduling problem. We demonstrate the operation of the proposed pGA/SPF using the simple DAG as shown in Fig 2.

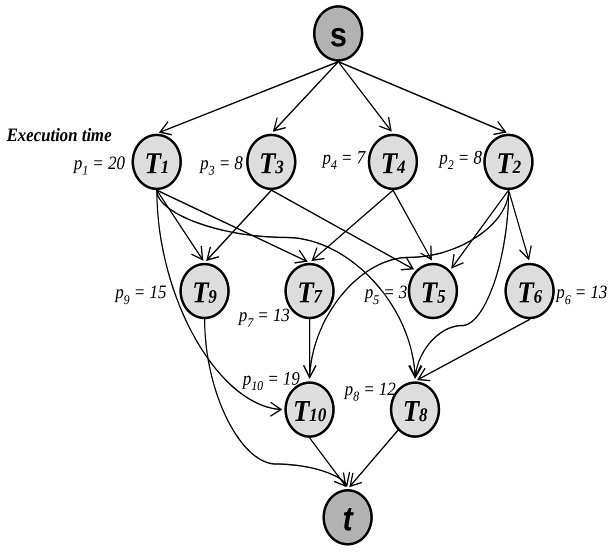
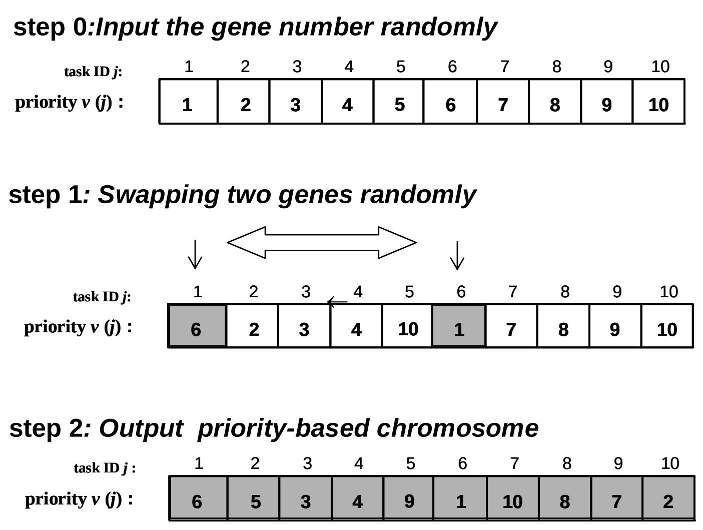


Fig. 2. A directed acyclic graph (DAG) with 10 tasks

Table 1. Data set of DAG

T_j	s	1	2	3	4	5	6	7	8	9	10	t
p_j	**0**	**20**	**8**	**8**	**7**	**3**	**13**	**13**	**12**	**15**	**19**	**0**
Pred$_j$	{ }	{s}	{s}	{s}	{s}	{2,3,4}	{2}	{1,4}	{1,2,6}	{1,3}	{1,2,7}	{8,9,10}



```
procedure 1: priority-based encoding
              ( initial chromosome )
input: number of processors $m$,
       number of tasks $n$
output: chromosome $[v(j)]$
begin
    for $j = 1$ to $n$                         // step 0
    $v(j)$    random (1, $n$);
    for $j = 1$ to $\lfloor n/2 \rfloor$       // step 1
        $j \leftarrow$ random (1, $n$);
        $k \leftarrow$ random (1, $n$);
            if $j \neq k$ then
                swap $\{v(j), v(k)\}$;
        Output the chromosome $[v(j)]$ // step 2
end
```

Fig. 3. An example priority-based encoding procedure

A. Priority-based Genetic Algorithm (pGA)

The scheduling problem can be thought of as consisting of two parts: the assignment of tasks to processors and task execution ordering within a processor. At list scheduling heuristic solves both problems at once. In a GA approach, how to encode a schedule for a DAG is a critical step. Special difficulty arises because: 1) a schedule contains variable number of nodes, and 2) a random sequence of edges usually does not correspond to a schedule. To cope with such difficulties, Gen et al. [9][10] adopted an indirect approach: Gen et al. proposed priority-based encoding method that the position of a gene was used to represent a task node and the value of the gene was used to represent the priority of the task node for constructing a schedule among candidates. As proposed encoding method, first randomly generate initial chromosome from procedure 1 of Fig. 3. Each position of chromosome is called a gene. Each gene will be used the priority of node in DAG. This encoding method is easily verified that any permutation of the encoding corresponds to the schedules, so that most existing genetic operators can easily be applied to the encoding.

Suppose we want to assign n tasks to m processors by using above chromosome (Fig. 3). We use the simple DAG as Fig. 1, at the beginning, we try to find a node for the position next to node s. Node T_1, T_2, T_3, and T_4 are eligible for the next position, which can be suitable for next start node. Here we check their priority that are 6, 5, 3 and 4 respectively. Then the task T_1 has the highest priority of 6 and is put into the schedule S. Then next possible nodes are T_2, T_3 and T_4. They have 5, 3 and 4 priority respectively, and then we put T_2 into schedule S. And we repeat these steps until we obtain complete schedule $S = \{T_1, T_2, T_4, T_7, T_3, T_5, T_9, T_{10}, T_6, T_8\}$ as decoding procedure 2 (Fig. 5).

B. Shortest Processor First (SPF) with Mapping

Next step, we assign the selected task T_j to processor P_i from the above completed schedule which corresponds to lowest absolute value of difference between the current schedule length on the processor and the earliest start time of the task node as shown the procedure 3. If the processor exists, our algorithm assigns the selected task T_j to the processor P_i with the smallest value of $a_i = |l_i - e_k|$. Otherwise it assigns the selected task T_j to the processor that allows the earliest execution, using the non insertion approach.

Procedure 3 describes the task assignment procedure. And where e_j is the earliest start time of task T_j, l_i the schedule length of processor P_i and a_i is the absolute value of difference between the earliest start time of task T_j and the schedule length on the processor P_i.

task ID j :	1	2	3	4	5	6	7	8	9	10
priority v (j) :	6	5	7	4	9	8	10	1	3	2

Fig. 4. A priority-based encoding chromosome

C. Evaluation Function

The calculation of the evaluation function is quite simple. First, the Gantt chart of each string is calculated. The length of each processor string is measured to find the total finishing time of the schedule. The evaluation function used for our algorithm is based on the makespan (F_{max}) of schedule.

```
procedure 2: decoding (one schedule growth)
input: number of tasks $n$, chromosome $[v(j)]$,
       set of task nodes $\bar{S}$
output: schedule $S$
begin
    $\bar{S} \leftarrow \emptyset$, $S \leftarrow \emptyset$;          //step 0
    $n \leftarrow 0$, $j \leftarrow 0$;
    while ( $j \leqq n$ ) do                  //step 1
        $\bar{S} \leftarrow \bar{S} \cup \text{Pred}_j$;
        $j^* \leftarrow \text{argmax}\{ v(j) | j \in \bar{S}\}$;
        $\bar{S} \leftarrow \bar{S} \setminus j^*$;
        $S \leftarrow \bar{S} \cup j^*$;
        $j^* \leftarrow j$;
    Output the $S$                            //step 2
end
```

```
procedure 3: decoding ( assigning tasks )
input: processing time $p_k$ for each task, schedule $S$,
       set of predecessors $\text{Pred}_j$
output: makespan $F$
begin
    $c_j \leftarrow 0,\ j = 1,2, \ldots, n$                    //step 0
    $l_i \leftarrow 0,\ i = 1,2, \ldots, m$
    $k \leftarrow 0$;
    for $r = 1$ to n                                 //step 1
        $k \leftarrow S[r]$
        $e_k \leftarrow \max \{c_j | j \in \text{Pred}_k\}$;
        $i^* \leftarrow \text{argmin} \{a_i | a_i = |l_i - e_k|, i = 1,2, \ldots, m\}$;
        $c_k \leftarrow l_{i^*} + p_k$;
        $l_{i^*} \leftarrow c_k$;
    makespan $F \leftarrow \max\{ c_k,\ k = 1,2, \ldots, n \}$ // step 2
end
```

Fig. 5. Decoding procedures

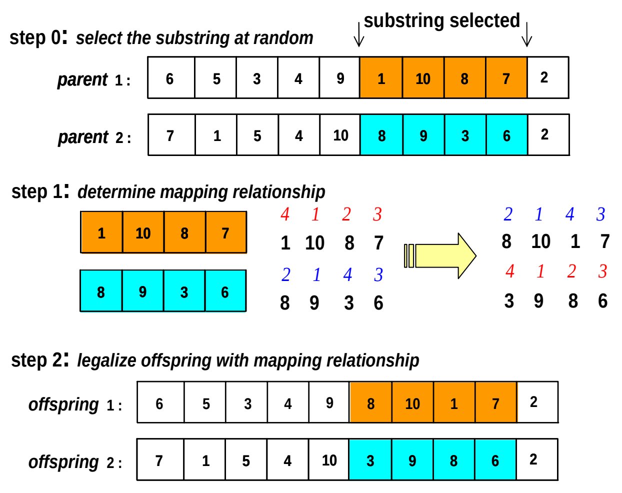
We convert the minimization problem to maximization problem, that is, the used evaluation function is as follow:

$$eval(v_k) = 1/F^k_{\max}, \qquad i = 1,\ldots, popSize$$

where $F^k_{\max}$: the makespan of the k-th chromosome

D. *Crossover*

Here the position-based crossover operator by the weight mapping crossover (WMX) that we proposed. It can be viewed as two-point crossover of binary string and remapping by order of different binary string as shown in Fig. 6.

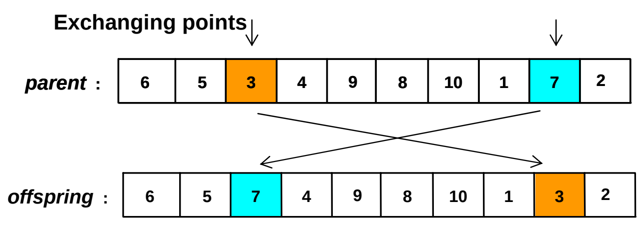


step 0: select the substring at random
substring selected
parent 1 : 6 5 3 4 9 1 10 8 7 2
parent 2 : 7 1 5 4 10 8 9 3 6 2
step 1: determine mapping relationship
1 10 8 7
8 9 3 6
4 1 2 3
1 10 8 7
2 1 4 3
8 9 3 6
2 1 4 3
8 10 1 7
4 1 2 3
3 9 8 6
step 2: legalize offspring with mapping relationship
offspring 1 : 6 5 3 4 9 8 10 1 7 2
offspring 2 : 7 1 5 4 10 3 9 8 6 2

Fig. 6. Weight Mapping Crossover (WMX)

E. *Mutation*

We proposed the swap mutation operator, in which two positions are selected at random and their contents are swapped as shown in Fig. 7.



Exchanging points
parent : 6 5 3 4 9 8 10 1 7 2
offspring : 6 5 7 4 9 8 10 1 3 2

Fig. 7. Swap Mutation operator

IV. Numerical Examples and Analyses

A. *Example 1 (DAG Fig. 2)*

Our proposed pGA/SPF algorithm, as mentioned above, creates a schedule the task graph onto the multiprocessor system with two processors. We present the performance results of the proposed pGA/SPF algorithm and compare with the HLFET (High Levels First with Estimated Times) by Adam et al. [8], ETF (Earliest Test First) by Hwang et al. [7] and MCP (Modified Critical Path) by Wu et al. [13] algorithms as shown in Fig 8. The HLFET algorithm schedule the nodes in the same schedule as the MCP algorithm (makespan = 64 time unit) and ETF algorithm schedule the worse (makespan = 67 time unit) than other algorithms.

As a result, our proposed algorithm performed batter than other three algorithms (makespan = 59 time unit).

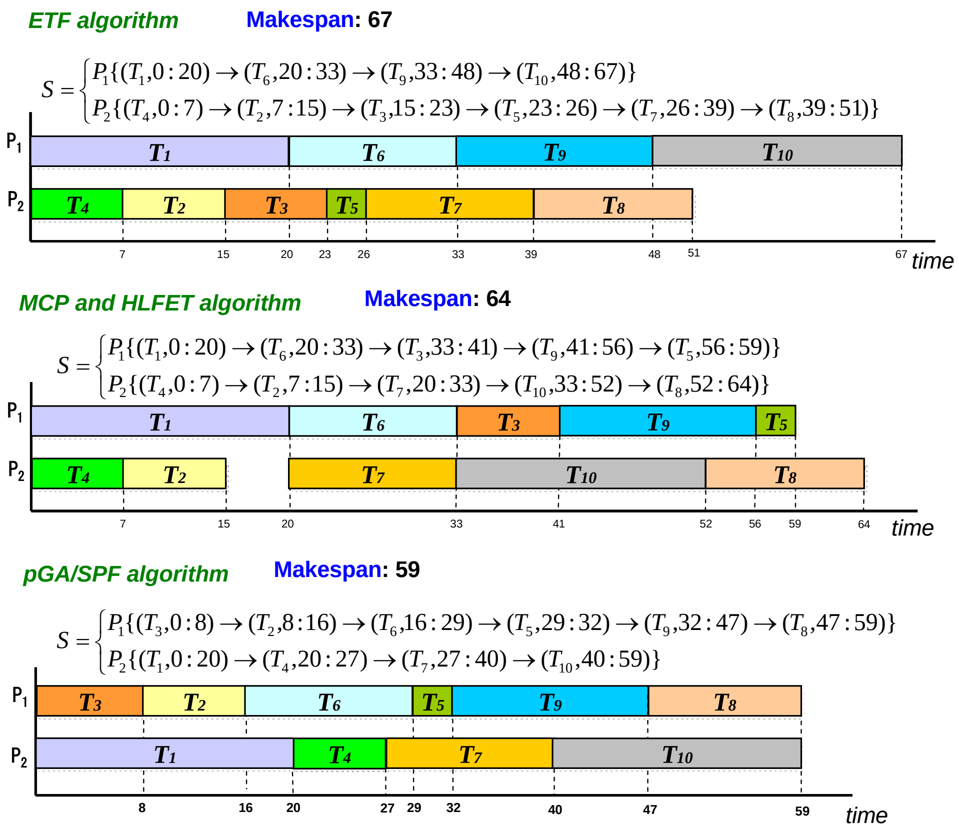


Fig. 8. The schedule Gantt chart of the task graph on Fig 2 generated by ETF, MCP, HLFET and pGA/SPF algorithms.

Table 2. Comparative Results for the Stanford Manipulator Task Graph

Number of Processors	Optimal Solution	Best Solution (time unit)			Optimal solution/GA approaches (%)		
		Hou *et al.*'s GA (1)	T-G's GA (14)	pGA/SPF	Hou *et al.*'s GA (1)	T-G's GA (14)	pGA/SPF
2	**1242**	1249	1247	**1246**	99.4	99.6	**99.7**
3	**879**	938	903	**888**	93.7	97.3	**99.0**
4	**659**	774	759	**724**	85.1	86.8	**91.0**
5	**586**	679	671	**642**	86.3	87.3	**91.3**
6	**573**	627	628	**605**	91.4	91.2	**94.7**
10	**570**	590	570	**570**	96.6	100.0	**100.0**

B. *Example 2 (Stanford Manipulator Graph)*

In a numerical experiment, we use the data of the Newton-Euler inverse dynamics equations task graph for the Stanford manipulator [2] as a large-scale of benchmark test. The Stanford manipulator task graph consists of 88 tasks. In Fig. 10, it does not need take nodes 1 and 90 into consideration because they are additional dummy nodes. We used same parameter of former GA approaches- Hou *et al.*'s GA [1] and T-G's GA [14] (Tsujimura and Gen)- to compare pGA/SPF with them. The proposed pGA/SPF used the following parameters throughout the simulations:

Population size:	*popSize* = 30
Maximum generation:	*maxGen* = 2000
Crossover probability:	$p_C = 0.7$
Mutation probability:	$p_M = 0.3$

The comparative results for various numbers of processors are summarized in Table 2. We present the performance results of the proposed pGA/SPF algorithm and compare with the Hou *et al.*'s GA and T-G's approaches as shown in Fig 9. pGA/SPF presents the ranging from 0.0 to 10% greater than the optimal solutions.

V. Conclusion

In This paper presents pGA/SPF scheduling algorithms which can schedule the directed acyclic graph (DAG) with precedence constraints between each task. The pGA/SPF algorithm schedules the tasks and it is suitable for graphs with arbitrary computation and without communication costs, and applicable to homogeneous fully connected processors. The performances of the proposed pGA/SPF algorithm have been observed by comparing with well-known heuristic list scheduling algorithms and other existing genetic algorithm approaches in terms of the schedule length. As a result, it is confirmed that the proposed pGA/SPF can provide good solutions for simple multiprocessor scheduling problems (Fig. 2) and improve task graph scheduling without significantly increasing the scheduling time. In addition, results on the second example showed that the completion time can be reduced more efficiently by using the proposed algorithm as compared with existing approaches.

In the future, we intent to extend our algorithms to schedule both the tasks and the messages for task graphs with arbitrary computation and communication costs.

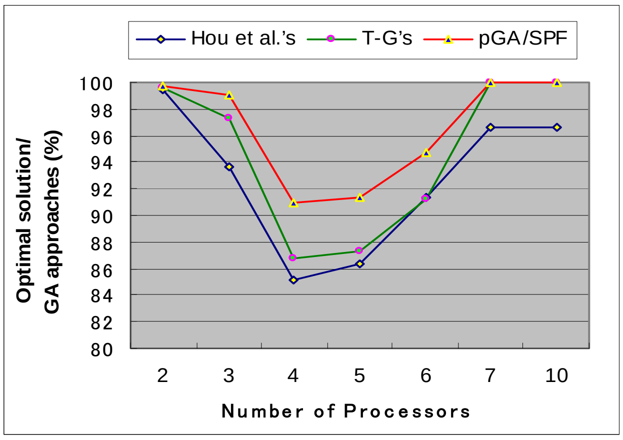


Fig. 9. Number of processors versus Best Solution

REFERENCES

[1] E. Hou, N. Ansari and H. Ren : "A Genetic Algorithm for Multiprocessor Scheduling", *IEEE Trans. on Parallel and Distributed System*, Vol.5, No.2, pp.113-120 (1994)

[2] H. Kasahara and S. Narita : "Parallel Processing of Robot-Arm Control Computation on a Multimicroprocessor System", *IEEE Trans. of Robotics and Automation*, Vol.Ra-1, No.2, pp.686-701 (1985)

[3] T. Tsuchiya, T. Osada and T. Kikuno : "A new heuristic algorithm based on GAs for multiprocessor scheduling with task duplication", *Proc. of Algorithms and Architectures for Parallel Processing*, Vol.10, No.12, pp.295-308 (1997)

[4] A. Gerasoulis amd T. Yang : "On the granularity and clustering of directed acyclic task graphs", *IEEE Trans. on Parallel and Distributed Systems*, Vol.4, No.6, pp.686-701 (1996)

[5] M.A. Palis and J.C. Lieu : "Task clustering and scheduling for distributed memory parallel architectures", *IEEE Trans. on Parallel and Distributed Systems*, Vol.7, No.1, pp.46-55 (1996)

[6] S. H. Woo, S. B. Yang, S. D. Kim and T. D. Han : "Task scheduling in distributed computing systems with a genetic algorithm", *Proc, of High Performance Computing on the Information Superhighway*, HPC Asia '97, pp.301-305 (1997)

[7] T. Tsuchiya, T. Osada and T. Kikuno : "A new heuristic algorithm based on GAs for multiprocessor scheduling with task duplication", *Proc. of Algorithms and Architectures for Parallel Processing*, Vol.10, No.12, pp.295-308 (1997)

[8] S. Darbha and D.P. Agrawal : "Optimal scheduling algorithm for distributed-memory machines", *IEEE Trans. on Parallel and Distributed Systems*, Vol.9, No.1, pp.87-95 (1998)

[9] M.A. Palis and J. C. Lieu : "A new heuristic for scheduling parallel programs on multiprocessor", *Proc. of International Conference on Parallel Architectures and Compilation Techniques*, Vol.2, No.18, pp.358-365 (1998)

[10] M. Gen and R. Cheng : *Genetic Algorithms & Engineering Design*, John Wiley & Sons, New York, (1997)

[11] M. Gen and R. Cheng : *Genetic Algorithm and Engineering Optimization*, John Wiley and Sons, New York, (2000)

[12] R.K. Hwang and M. Gen : "Task Scheduling in Parallel and Distributed Systems Using Priority-based Genetic Algorithm", *Proc. 33rd International Conference on Computer and Industrial Engineering*, Korea, pp.686-701 (2004)

[13] M.-Y. Wu and D. D. Gajski : "Hypertool: A programming aid for message-passing systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol.1, No.3, pp.330-343 (1990)

[14] Tsujimura, Y. and M. Gen : "Genetic algorithms for solving multiprocessor scheduling problems", *Simulated Evolution and Learning* , Springer-Verlag, Heidelberg, pp.106-115 (1995).

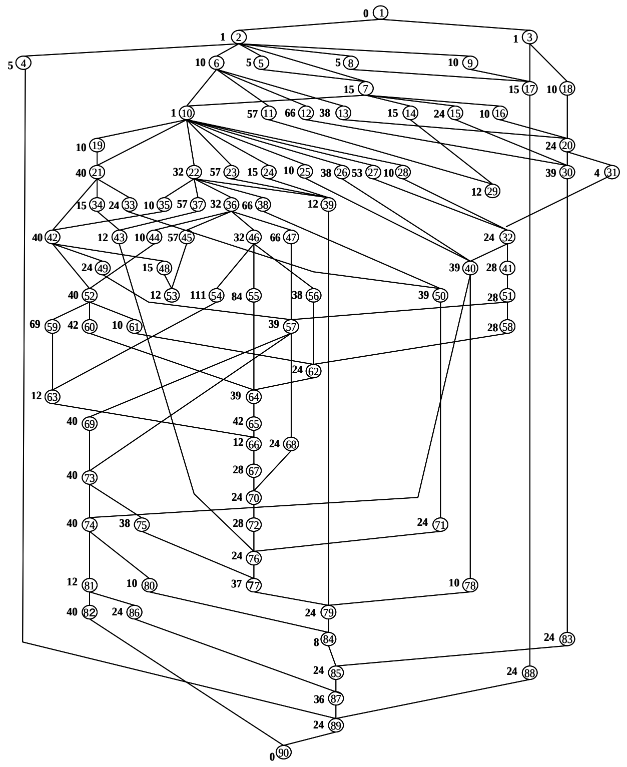


Fig. 10. Stanford manipulator with 88 tasks