© The Author 2006. Published by Oxford University Press on behalf of The British Computer Society. All rights reserved.
For Permissions, please email: journals.permissions@oxfordjournals.org
Advance Access published on June 15, 2006 doi:10.1093/comjnl/bxl030

Static Task Scheduling with a Unified Objective on Time and Resource Domains

BETUL DEMIROZ AND HALUK RAHMI TOPCUOGLU*

Department of Computer Engineering, Marmara University, 34722, Istanbul, Turkey
**Corresponding author: haluk@eng.marmara.edu.tr*

Task scheduling for parallel and distributed systems is an NP-complete problem, which is well documented and studied in the literature. A large set of proposed heuristics for this problem mainly target to minimize the completion time or the schedule length of the output schedule for a given task graph. An additional objective, which is not much studied, is the minimization of number of processors allocated for the schedule. These two objectives are both conflicting and complementary, where the former is on the time domain targeting to improve task utilization and the latter is on the resource domain targeting to improve processor utilization. In this paper, we unify these two objectives with a weighting scheme that allows to personalize the importance of the objectives. In this paper, we present a new genetic search framework for task scheduling problem by considering the new objective. The performance of our genetic algorithm is compared with the scheduling algorithms in the literature that consider the heterogeneous processors. The results of the synthetic benchmarks and task graphs that are extracted from well-known applications clearly show that our genetic algorithm-based framework outperforms the related work with respect to normalized cost values, for various task graph characteristics.

Keywords: Task scheduling, genetic algorithms, heuristics, parallel computing, task graphs

Received 11 July 2005; revised 25 January 2006

1. INTRODUCTION

Scheduling is in general an important problem in several areas such as computer systems, manufacturing, process control and operations research. Efficient scheduling tasks of an application on available processors are a critical issue for achieving high performance in parallel and distributed systems. The general task scheduling problem includes the problem of assigning the tasks of an application to suitable processors and the problem of ordering task executions on each processor so that task precedence requirements are satisfied. When the characteristics of an application, which include execution times of the tasks, communication times between the tasks and the task dependencies, are known *a priori*, they are represented with a *static* model. For static scheduling, an application is represented by a weighted directed acyclic graph (DAG), also called macro-dataflow graph, in which the nodes represent application tasks, and the edges represent inter-task data dependencies.

The task scheduling problem is NP-complete in its general case [1] as well as in some restricted cases [2], such as scheduling tasks with one or two time units to two processors, or scheduling unit-time tasks to an arbitrary number of processors. There are only a few known polynomial-time scheduling algorithms [3, 4] that were generated by placing restrictions on the input task graph. Considerable research efforts for this problem caused a very large set of proposed heuristics that is mainly for systems with homogeneous processors. A few recent studies consider the heterogeneous systems [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. Some of the recent studies for heterogeneous systems consider multiple phases of hybridizing different techniques or strategies [15, 16, 17].

Each of the proposed heuristics maps the tasks onto processors and orders their executions on each processor so that task-precedence requirements are satisfied. The performance of any heuristic on a given graph is usually measured in terms of quality of the output schedule, which is equal to the schedule length or the overall completion time. In addition to the schedule length (or the makespan) minimization, a second objective, which is not as much addressed as in the first one, is the minimization of the number of processors allocated for the output schedule.

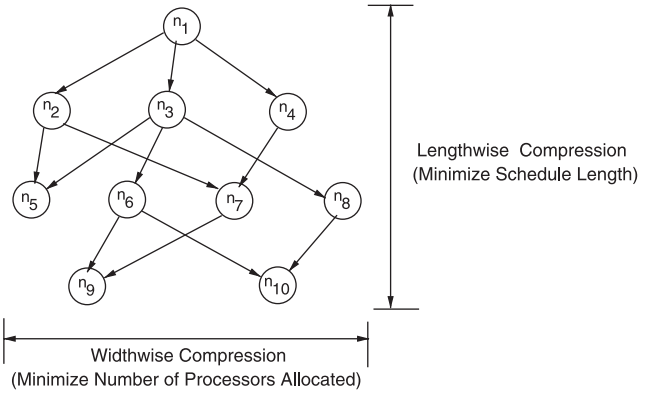


FIGURE 1. Two objectives of the task scheduling problem [18].

The *width* of a task graph is the maximum number of tasks on a level, which is basically the maximum number of tasks that can be executed concurrently. As can be seen from Figure 1, the minimization of the makespan requires the lengthwise compression, and the minimization of the number of the processors allocated requires widthwise compression. There may be tradeoffs between these objectives; i.e. using larger number of processors can possibly result in better quality of schedule. The schedule length minimization is for application utilization and the other one is for resource utilization.

Task scheduling problem with multiple objectives of the schedule length and the processor usage minimizations is not much studied in the literature. Ge and Yun [18] presented a study by forming a 2D scheduling window where 'pressure' is applied in both dimensions by utilizing constrained resource planning (CRP) paradigm. The CRP paradigm in general provides two domain-independent guiding strategies for the identification of tasks and the selection of the solutions: the *most-constrained strategy* and the *least-impact strategy*. Based on the first strategy, the most constrained task is the one that has the least flexibility for delay. The second strategy selects the solution of the current task by minimizing the impact to the other tasks. The performance is measured by a 2D cost function, which is computed by multiplying the schedule length with the number of processors allocated. The cost function of CRP-based method [18] is limited, since minimization of the schedule length and minimization of the number of processors are two conflicting objectives with different features. Additionally, the normalization of these two terms with respect to their lower bounds is not considered in the CRP-based method; and the cost function does not consider a 'weighting scheme' for personalizing the importance of those two objectives. Moreover, the CRP-based method considers only homogeneous processors as part of the 2D compression process.

In this paper, we first unify the two objectives on time and resource dimensions with a weighting scheme that includes the normalized schedule lengths and the normalized processor usages. Then, the general task scheduling problem with the unified objective is formulated using a genetic algorithm (GA). The performance of our GA-based method is compared with five task scheduling heuristics for various graph characteristics. The results from the experimental study show that our proposed heuristic outperforms the other heuristics on various graph characteristics including the graph size, communication-to-computation ratio (CCR) and the graph structure.

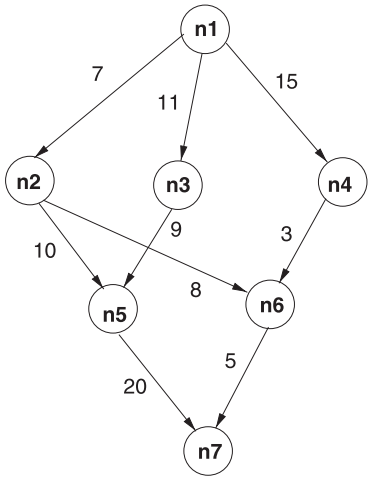
The rest of this paper is structured as follows. In the next section, we define the task scheduling problem with the related terminology and we propose a new unified objective. Our GA-based framework and its details are presented in Section 3. We present a short summary of related work in Section 4. Section 5 gives a comparison study of our algorithms with the related work for various graph attributes, which is based on randomly generated task graphs and the task graphs of real applications. We state the conclusions in Section 6.

2. PROBLEM DEFINITION

In static scheduling, an *application* is represented by a DAG, $G = (V, E)$, where V is the set of v tasks and E is the set of e edges between the tasks. Each edge $(i, j) \in E$ represents the precedence constraint where the task n_i should complete its execution before the task n_j starts. There are research efforts to replace DAG with new models such as TTIG [19, 20] representations for iterative parallel applications, which are out of scope of this paper.

Target computing environment consists of a set Q of q heterogeneous processors connected in a fully connected topology in which all inter-processor communications are performed without contention. The estimated execution time (i.e. the computation cost) to complete task, the n_i on the processor p_j, is represented by $w_{i,j}$. The communication cost of the edge (i, k), i.e. the expected communication time for transferring data from the task n_i (scheduled on p_m) to the task n_k (scheduled on p_n) is represented by $c_{i,j}$. When both n_i and n_k are scheduled on the same processor, $c_{i,k}$ is assumed to be zero since the intra-processor communication cost is negligible. A seven-node task graph with the computation costs and average communication costs is given in Figure 2. Two sample schedules of this graph are given in Figure 3, which require schedule lengths of 50 and 47 by allocating 2 and 3 processors respectively.

Before presenting the objective function, it is necessary to define the EST and the EFT attributes. EST(n_i, p_j) and EFT(n_i, p_j) are the *earliest* execution start time and the *earliest* execution finish time of the task n_i on the processor p_j respectively. They are computed recursively starting from



Computation Costs

Task	P1	P2	P3
n1	8	11	9
n2	10	5	12
n3	11	13	15
n4	10	3	9
n5	12	13	11
n6	13	12	8
n7	7	5	9

FIGURE 2. A 7-node task graph with the computation and the communication costs.

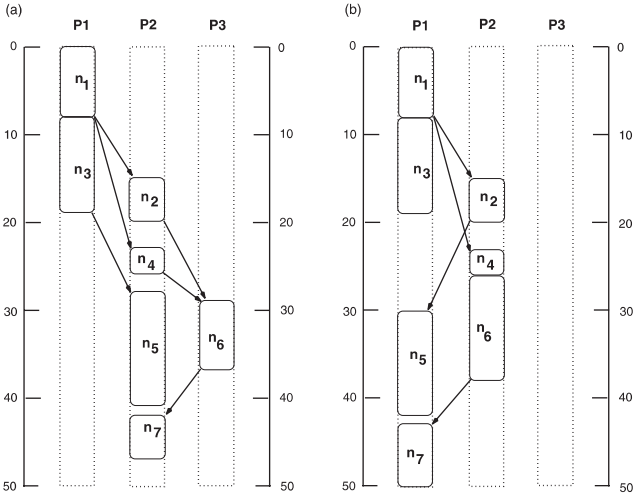


FIGURE 3. Two output schedules of the task graph given in Figure 2.

the entry task.

$$\mathrm{EST}(n_i, p_j) = \max\{ready_{proc}(p_j), ready_{data}(n_i, p_j)\} \quad (1)$$

$$\mathrm{EFT}(n_i, p_j) = w_{i,j} + EST(n_i, p_j). \quad (2)$$

Here, $ready_{data}$ (n_i, P_j) is the time when all data needed by the task n_i have arrived at the processor p_j, which is defined by

$$ready_{data}(n_i, p_j) = \max_{n_m \in pred(n_i)} (\mathrm{AFT}(n_m) + c_{m,i}), \quad (3)$$

where the AFT (n_m) is the actual finish time of the n_m on its scheduled processor and *pred* (n_i) is the set of the immediate predecessor tasks of n_i. In Equation (1), $ready_{proc}$ (p_j) is the earliest time at which the processor p_j is ready for task execution. For a non-insertion-based scheduling policy, if n_k is the last scheduled task on p_j, then $ready_{proc}p_j$ is equal to actual finish time of the task n_k, which is represented by AFT(n_k). We consider the insertion-based scheduling, which targets a possible insertion of each task in the earliest idle time slot between two already-scheduled tasks on the same processor by preserving the precedence constraints.

The schedule length of an output schedule is basically the actual finish time of the exit task of a given graph. In our study, there are two objectives that should be addressed:

- minimization of the schedule length, which is an objective for the *time domain* and
- minimization of the number of the processors used, which is an objective for the *resource domain*.

These two objectives are combined with a weighting scheme that allows to personalize the importance of them to a particular situation. The new objective is to minimize the normalized cost value, NCost, of the output schedule, which is defined by

$$\mathrm{NCost} = w_1 \times \mathrm{NSL} + w_2 \times \mathrm{NPU} \quad (4)$$

In this equation, NSL is the normalized schedule length, NPU is the normalized processor usage and w_1, w_2 are the coefficients used for the importance of the objectives. In our experimental study, the coefficients are set with the predefined values to provide equal importance of the two objectives given in Equation (4).

Since a large set of task graphs with different properties is considered in the experiments, it is necessary to normalize the schedule length and the processor usage terms to their lower bounds. For an unscheduled DAG, if the computation cost of each task is set with the minimum value among the various processor alternatives, then the critical path (the longest path) of the graph is represented with $\mathrm{CP_{MIN}}$. The NSL is computed by

$$\mathrm{NSL} = \frac{\text{Schedule length}}{\sum_{n_i \in \mathrm{CP_{MIN}}} \min_{p_j \in Q}\{w_{i,j}\}}. \quad (5)$$

The denominator in Equation (5) (i.e. the lower bound of the schedule length) is the summation of minimum computation cost of each task. The NPU term is computed by

$$\mathrm{NPU} = \frac{\text{\#PEs used}}{\text{Width of graph}}, \quad (6)$$

where the numerator is the total number of processing elements (PEs) used in the output schedule. The width of a task graph is the maximum degree of concurrency, which is the maximum number of the tasks of a given graph that can be executed concurrently. The width of the graph in Figure 2 is equal to 3, by considering the tasks n_2, n_3, n_4. On the other hand, the width of the graph in Figure 4 is equal to 4, which is due to the tasks filled with the given pattern.

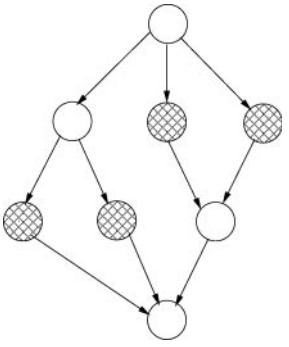


FIGURE 4. An 8-node task graph where the maximum degree of concurrency is equal to 4.

When the Dilworth's theorem [21, 22] is considered, the width of a graph is equal to the minimal number of chains (i.e. the paths) needed to cover the given task graph. A *path cover* of a task graph is a set P of vertex-disjoint paths such that every vertex in the task graph is included in exactly one path in P [23]. The paths may be of any length (including 0) and they may start and end anywhere in the graph. A *minimum path cover* of a task graph is a path cover containing the fewest possible paths.

The minimum path cover of a task graph (or the width term given in Equation (6)) is computed in polynomial time via reduction to the maximum flow problem [23]. In order to apply the maximum flow algorithm, a new graph $G'(V', E')$ is constructed from the task graph $G(V, E)$ by assuming $V = \{1, 2, \ldots, v\}$, where

$$V' = \{x_0, x_1, \ldots, x_v\} \bigcup \{y_0, y_1, \ldots, y_v\} \tag{7}$$

$$E' = \{(x_0, x_i) : i \in V\} \bigcup \{(y_i, y_0) : i \in V\} \bigcup \{(x_i, y_j) : (i, j) \in E\}. \tag{8}$$

After the maximum flow algorithm is executed on the new graph G' by considering unit edge capacities, the minimum path cover (so the width of the original graph G) is equal to $v - f$, where v is the number of tasks in the original graph G and f is the maximum flow in the new graph G'.

3. GA-BASED TASK SCHEDULING

GAs [24, 25, 26, 27] are general-purpose, stochastic search methods that use the principles inspired by natural selection and genetics. Each point in the well-defined search space of a given problem is called a *chromosome*. A GA operates by iteratively generating a population of chromosomes that are encoded of the candidate solutions. The quality of the solution represented by a chromosome is evaluated with a function called *fitness function*.

In order to reach an optimum solution, a GA typically uses three biologically inspired operators (*selection*, *crossover* and *mutation* operators) by applying the 'survival of the fittest' principle. The selection operator forms a new population of strings by selecting from the old population based on their fitness values. The crossover operator exchanges the portions of the selected chromosomes to generate two valid chromosomes by applying with a predefined probability. The mutation operator is an occasional random operator that gives an opportunity to reach parts of the search space which may not be reached by the crossover operator. GAs have been efficiently used in a wide variety of applications in the engineering, science and business fields [24, 28].

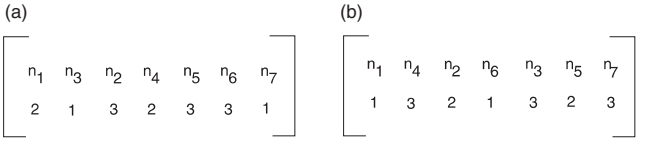


FIGURE 5. Two sample schedules for the task graph given in Figure 2.

Although there is a large number of GA-based approaches proposed in the literature for the task scheduling problem [10, 11, 29, 30, 31, 32, 33, 34, 35, 36], their common objective is to minimize the schedule length of the graph. The minimization of the processor usage is not considered in the related work. Additionally, many of them do not consider the heterogeneous environments [29, 31, 32, 34, 35].

As stated in [37], GA-based task scheduling methods can be classified into two main approaches: the *direct* and the *indirect* representations. For the direct representation, a given chromosome represents a single solution; and the solution represented with a string always yields the same makespan [29, 34, 35]. On the other hand, in an indirect representation, one or more decoding methods or algorithms are proposed in order to build the solutions by considering the given chromosome representation [32, 33, 38]. The direct representation is considered in our GA-based approach, which is presented in detail in the following subsections. Additionally, an indirect representation method [32, 33] is presented in Section 5, which is considered in the experimental study.

3.1. String representation

In our GA-based approach, each chromosome is represented by a string of length v denoted by *str*, where v is the number of tasks in the given graph. Each *gene* of a given string *str* is a tuple, such as $str(i) = (n_j, p_k)$ where the i-th location of the string includes the task n_j and it is assigned to the processor p_k. The *str* string is a topological sort of the given graph, where the total ordering of the tasks in *str* preserves the precedence constraints. Therefore, if the task graph has the edge (a, b), then n_a should come before n_b in *str*. Figure 5 has two valid strings (i.e. schedules) for the task graph given in Figure 2.

3.2. Initial population generation

A predefined number of strings are generated as part of the initial population generation step. The first phase of string generation is the topological order of the tasks given in a graph, which is done with the *blevel* ordering and the *ready list* methods. The *blevel* (the bottom level) of the task n_i is the length of the critical path (the longest path) from n_i to the exit task, including the computation cost of n_i. It is recursively defined by

$$blevel(n_i) = \bar{w}_i + \max_{n_j \in succ(n_i)} (\overline{c_{i,j}} + blevel(n_j)), \qquad (9)$$

where $succ(n_i)$ is the set of the immediate successors of n_i, $\overline{c_{i,j}}$ is the average communication cost of the edge (i, j) and $\bar{w}_i$ is the average computation cost of the task n_i. For the exit task, its blevel is equal to its average computation cost. The decreasing order of tasks with respect to blevel values provides the order of tasks in the string representation.

The *ready list* method orders tasks by randomly choosing a task among the tasks in the ready list, at each step. A task is inserted into the ready list when all of its immediate predecessors are already scheduled; and the task is deleted from the list when it is scheduled. After the topological sort is generated using blevel ordering and ready-list methods, each task in the string is randomly assigned to a processor in the system. In our experiments, the two methods are repeated for a predefined number of cases to generate strings with the different order of tasks and the different mappings, as well.

3.3. Genetic operators

The fundamental components that form the basis of GAs are the *selection operators* which allow better individuals to become parents of the next generation and the variation operators (*crossover*, and *mutation*) that create the necessary diversity [27].

Selection. We consider a deterministic tournament-based selection by setting the tournament size to 2, since this type of selection does not require global knowledge of the population, and it does not require the fitness values to be normalized. There is no need to reformulate the minimization objective to a maximization objective as in the roulette wheel method; therefore, the fitness value of an individual is equal to the normalized cost of the output schedule generated by the algorithm.

Crossover. We consider a single-point order crossover operator. First, it generates a crossover point and cuts the selected pair of strings into left and right parts. Then, it copies the portion on the left of the crossover point of the string from the first parent to the child. Then, the remaining genes in the form of tuples from the other parent are appended in the same order. The second offspring is generated with the same way. The order crossover operator always produces valid strings by preserving the precedence constraints [31].

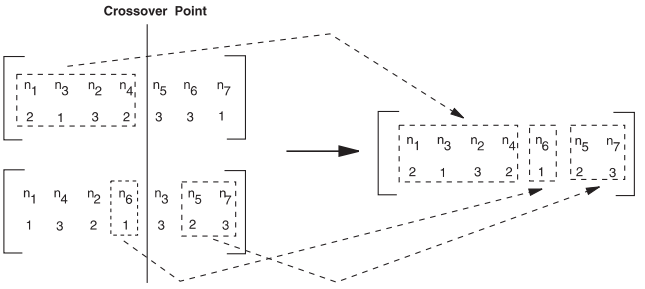


FIGURE 6. Applying crossover operator on two parent strings of the task graph given in Figure 2.

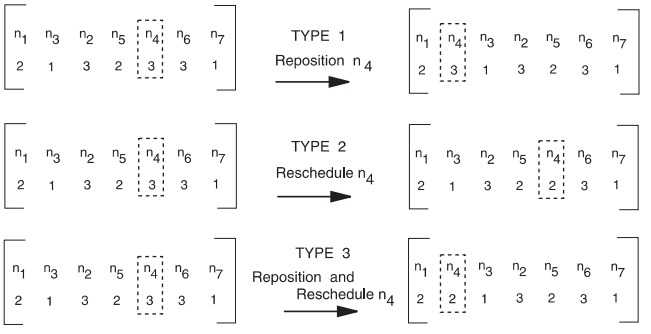


FIGURE 7. Three mutation operators applied on the selected task.

As an example, the left portion of the first parent given in Figure 6, $\{(n_1, 2), (n_3, 1), (n_2, 3), (n_4, 2)\}$, is passed directly to the child. For the remaining tasks (n_5, n_6 and n_7), their corresponding genes in the second parent are appended to the child by preserving their order given in the second parent. The second offspring was not drawn in Figure 6.

Mutation. Three mutation operators are considered in our experimental study.

- *Type* 1. This operator randomly repositions the selected task n_i on the given string anywhere between its first and last possible locations, without violating any data dependency constraints. The first possible location is set by adding one to the location of the parent of n_i which is the closest one to the end of the given string. The last possible location is set by subtracting one from the location of the child of n_i which is the closest one to the beginning of the string. Then, the task n_i is moved to a random position in the string within its free range, which is bordered by its first and last possible locations. When the 'type 1' mutation operator is considered on n_4 of the string given in Figure 7, the values of the first and the last possible locations of n_4 are equal to 2 and 5 respectively.
- *Type* 2. This type of mutation operator selects a task and it randomly modifies the assignment of the task. In

Figure 6, the 'type 2' operator maps the task n_4 to the processor p_2.

- *Type* 3. In this type, both the 'type 1' and the 'type 2' mutation operators are applied on the randomly selected task.

4. RELATED WORK

The task scheduling problem for heterogeneous processors with a unified objective on time and resource domains was not studied in the literature. Therefore, we consider only the algorithms that achieved the best results with respect to the time domain (i.e. the schedule length minimization) in our experiments. The heterogeneous-earliest-finish-time (HEFT) algorithm [5], the dynamic level scheduling (DLS) algorithm [6], the mapping heuristic (MH) [7] and the levelized-min time (LMT) algorithm [8] are selected for the comparison study due to their performance and their occurrences in various comparative studies in the literature. Additionally, a GA-based algorithm [32, 33] that uses the indirect representation is also considered in the comparison study to provide an objective testing for the effectiveness of the proposed study.

HEFT algorithm. This algorithm [5] sets computation and communication costs of the tasks with the mean values. The blevel values are computed, and the tasks are sorted in a list by non-increasing order of their blevel values. It selects the task with the highest blevel value at each step and it assigns this task to the processor which minimizes its earliest finish time with an insertion-based approach. The experiments presented in [5] show that the HEFT algorithm significantly surpassed the previous approaches in terms of both quality of schedules and the running time of generating schedules. For a dense graph when the number of edges is proportional to $O(v^2)$, the time complexity of the HEFT algorithm is on the order of $O(v^2 \times q)$, where v is the number of tasks and q is the number of processors.

DLS algorithm. In this algorithm [6], the computation cost of each task is set with the median value of the computation costs. At each step, this algorithm selects the (ready node, available processor) pair that maximizes the value of the *dynamic level*, which is equal to $\mathrm{DL}(n_i, p_j) = blevel^s(n_i) - \mathrm{EST}(n_i, p_j)$. The $blevel^s(n_i)$ term is the static blevel value of n_i, which is computed similar to Equation (9) without considering the communication costs. The EST term is the earliest start time of the given task and the processor pair (see Equation (1)). For the heterogeneous environments, a new term was added to represent the difference between the task's median execution time on all processors and its execution time on the current processor. The general DLS algorithm has an $O(v^3 \times q)$ time complexity.

MH algorithm. In this algorithm [7], the computation cost of a task on a processor is computed by the number of instructions to be executed in the task, divided by the speed of the processor. However, in setting the computation costs of tasks and the communication costs of edges before scheduling, a system with homogeneous processors is assumed. The heterogeneity comes into the picture during the scheduling process. This algorithm uses static blevel value to assign each task's priority. (The authors also experimented by adding the communication delay to the rank values.) The MH algorithm does not schedule a task to an idle time slot that is between two tasks already scheduled. The time complexity when contention is not considered is equal to $O(v^2 \times q)$ for v tasks and q processors.

LMT algorithm. It is a two-phase algorithm [8], where the first phase groups the tasks that can be executed in parallel by considering the levels of the tasks; and the second phase assigns each task to the fastest available processor. A task in a lower level has higher priority than a task in a higher level. Within the same level, the task with the highest computation cost has the highest priority. If the number of tasks in a level is greater than the number of available processors, fine-grained tasks are merged into coarse-grained tasks until the number of tasks is equal to the number of processors. Beginning from the largest task and using the decreasing order with respect to average computation costs, each task is assigned to a processor that minimizes the sum of the task's computation cost and the total communication costs with tasks in the previous levels. For a fully connected graph, the time complexity is equal to $O(v^2 \times q^2)$.

Indirect GA. A GA-based task scheduling algorithm by considering indirect chromosome representation was proposed by Ahmad and Dhodhi [32]. Based on the comparison study of the genotype representations given in [37], their algorithm was the most reliable one among a set of the direct and the indirect GA-based solutions, due to its lowest maximum deviation from the best solution found for various task graphs. The extended version of their algorithm for heterogeneous systems [33] is considered in our computational experiments, which is called as IGA (indirect GA) in the rest of the paper.

The chromosome representation of the IGA method has two parts. The first part of the chromosome indicates the availability of processors in the output schedule. The second part consists of a priority list, where each task of the given graph has a priority value. The priority values are used to generate the schedule by using the priority list scheduling. Separate variation operators (the single point crossover and the mutation operators) are applied to each part of the chromosome.

5. PERFORMANCE EVALUATION

In this section, we first present the results of the experiments for determining the values of various GA parameters. It is followed by experimental evaluation of our algorithm,

which is based on the two types of the task graphs: the synthetically generated task graphs with various characteristics, and the task graphs that capture the features of the selected well-known problems.

5.1. Experiments for identifying parameter settings

Our GA-based framework has four control parameters: the crossover rate (δ_C), the mutation rate (δ_M), the population size (P) and the number of generations (G). A fifth parameter is the mutation operator, which is chosen among the three types mentioned in Section 3.3. For each graph, the crossover rate is varied in the range between 0.1 and 1.0 with an increment of 0.1, and the mutation rate is varied in the range 0.05–0.5 with an increment of 0.05, which results in 100 different test cases. When the average normalized costs and the total number of cases achieving the best results are considered as the performance metrics, the crossover and the mutation rates are set to 0.8 and 0.35 in our experiments respectively.

The graphs of higher number of tasks generate higher number of different solutions. Therefore, population size can be varied linearly with respect to the number of tasks in the graphs. A set of experiments with different graph sizes are performed to identify the values of the population size (P) and the number of generations (G) by using the equations $P = k_1 \times v$ and $G = k_2 \times v$, where v is the number of tasks and (k_1, k_2) are coefficients that are varied with a predefined set $S = \{1, 3, 5, 10, 15\}$. Based on the results with various task graphs, a significant reduce is achieved in normalized cost values, when k_1 and k_2 are equal to 3. Therefore, both the population size and the number of generations are set with the equation $3 \times v$, in the subsequent experiments.

When the performance of the three mutation operators presented in Section 4.3 are compared, it is observed that the mutation based on rescheduling the tasks (the 'type 2' operator) outperforms the mutation based on repositioning the tasks (the 'type 1' operator). In our experiments, a set of individuals in the initial populations is generated by using the blevel order. Since the blevel order is the most appropriate attribute for generating the order of the tasks [5], repositioning tasks as in the 'type 1' mutation operator breaks this order and it may lead to an increase in the schedule length, which causes an increase in the normalized cost. Similarly, the 'type 3' operator does not perform as good as in the 'type 2' operator due to the fact that it includes repositioning the tasks.

5.2. Experimental evaluation based on synthetically generated task graphs

In our study, a graph generator module is implemented to produce benchmark graphs with various characteristics that depend on several input parameters. The experimental framework executes the graph generator to construct the task graphs, followed by the execution of the scheduling algorithms to generate the output schedules. Finally, it computes the performance metrics based on the schedules.

5.2.1. *Details of the graph generator*

The graph generator module requires the following input parameters that diversify the generated task graphs.

- Number of tasks in the graph (v).
- Shape coefficient of the graph (α). It is the *root* of the ratio of the average width (i.e. the number of tasks in a level) to the average height (i.e. the number of levels) of the graph. A dense graph (a shorter graph with high parallelism) can be generated by selecting $\alpha \gg 1.0$. Similarly, it generates a higher-depth graph with minimum parallelism capability, when $\alpha \ll 1.0$.
- Task heterogeneity range (ϕ_T). It represents the variation among the base computation cost of each task on the given reference machine. For each task n_i, its base computation cost on the reference machine B, which is represented by w_i^B, is randomly set using the uniform distribution in the range of [1, ϕ_T].
- Machine heterogeneity range (ϕ_M). It represents the variation among the computation costs of tasks across all processors. The heterogeneity factor of each processor P_k with respect to the reference machine for the given task n_i is set from the range [1, ϕ_M]. The heterogeneity factors are multiplied with the base computation costs to compute the actual computation costs on all processors.
- CCR value. It is the ratio of the average communication cost to the average computation cost for the given graph. If the CCR value is very low, it can be considered as a computation-intensive application; if it is very high, the given application is a communication-intensive one.
- Average fan-out degree of tasks in the given graph.
- Total number of processors in the system (p).

The processor-related parameters are also considered in the above list, which is due to the fact that the computation costs of tasks on all processors are set as part of the random graph generation. A set of predefined values is considered for each of these input parameters in our experimental study, which provides diverse DAGs with various characteristics.

5.2.2. *Performance results*

In our experiments, the values of the GA parameters are set with the best values presented in Section 5.1. Additionally, one of the string in the initial population is determined with the schedule generated by the HEFT algorithm, since the HEFT algorithm outperformed the other algorithms with respect to all the metrics given for the time domain [5].

The task graphs for the computational experiments can be classified into four different heterogeneity types with respect to the values of the task and the machine heterogeneity ranges, which are (i) the low-task, the low-machine heterogeneity

TABLE 1. The comparison of the algorithms with respect to the graph size for the LoLo and HiHi heterogeneity types

		LoLo Graph size					HiHi Graph size				
Algorithm	Metric	20	40	60	80	100	20	40	60	80	100
GA	NCost	0.70	0.59	0.61	0.67	0.72	0.83	0.66	0.69	0.74	0.79
	NSL	2.90	2.60	2.57	2.59	2.56	2.69	2.36	2.46	2.32	2.12
	PE	7.78	13.73	17.69	20.52	23.37	10.56	17.83	22.26	25.52	28.86
IGA	NCost	0.64	0.55	0.58	0.71	0.76	0.77	0.68	0.69	0.76	0.82
	NSL	3.39	3.40	3.56	3.79	4.00	3.16	3.31	3.81	4.08	4.40
	PE	5.72	8.91	11.12	16.73	17.83	8.58	14.79	15.67	17.60	18.81
HEFT	NCost	0.96	0.77	0.81	0.87	0.91	0.99	0.98	0.77	0.84	0.87
	NSL	1.74	1.74	1.76	1.71	1.73	1.19	1.19	1.17	1.16	1.17
	PE	15.20	25.12	31.58	34.87	36.91	16.51	36.34	32.59	35.96	37.62
DLS	NCost	0.98	0.79	0.83	0.88	0.92	1.00	0.76	0.79	0.85	0.88
	NSL	1.74	1.74	1.76	1.71	1.74	1.20	1.19	1.17	1.17	1.18
	PE	15.47	25.98	32.49	35.53	37.55	16.7	27.02	33.48	36.72	38.09
MH	NCost	0.98	0.78	0.81	0.87	0.91	1.00	0.74	0.77	0.84	0.87
	NSL	1.74	1.74	1.76	1.72	1.75	1.19	1.19	1.17	1.16	1.18
	PE	15.43	25.68	31.93	34.87	37.01	16.65	26.50	32.81	36.02	37.63
LMT	NCost	0.93	0.76	0.80	0.88	0.93	0.99	0.77	0.80	0.87	0.91
	NSL	1.81	1.81	1.93	1.91	1.98	1.20	1.22	1.40	1.44	1.59
	PE	14.35	24.72	30.55	34.23	36.38	16.50	27.58	33.29	36.51	37.98

(LoLo) type, (ii) the low-task, the high-machine heterogeneity (LoHi) type, (iii) the high-task, the low-machine heterogeneity (HiLo) type and (iv) the high-task, the high-machine heterogeneity (HiHi) type. The value of the task heterogeneity range is set to 100 and 2000 for low-task and high-task heterogeneity cases respectively. Similarly, the value of the machine heterogeneity range is set with 10 and 100.

The performance of the algorithms are compared with the different sets of graphs for each of the heterogeneity type. The computational experiments given in this subsection are classified according to the characteristics of the generated task graphs. We only present the results of selected heterogeneity types for each experiment, due to significant similarities.

Graph size. The first experiment compares the performance of the heuristics with respect to the *NCost*, the NSL and the number of processors allocated (PE), by varying the graph size in the given range and assigning fixed values for the other parameters. Specifically, the shape coefficient and the CCR values are set to 1.0; and the fan-out degree range is set to 5. The experimental results given in this paper are the average results over 100 different task graphs by setting the number of processors in the system to 40, if it is not specified otherwise.

We only consider the results of LoLo and the HiHi heterogeneity types, which are presented in Table 1. It should be noted that the values in the cells of the tables are the average of results of the 100 different task graphs. When Table 1 (LoLo) is examined, our GA-based solution and the IGA method outperform the other methods up to 34%, with respect to the NCost values. The performance of the IGA method is slightly better than that of our algorithm, when the small size graphs are considered. Our method outperforms the IGA method up to 5%, for large graphs. An increase in graph size may increase the number of tasks that are not on the critical path, when values of other parameters including the CCR, the shape parameter and the number of processors are fixed. Therefore, an increase in the graph size increases the NSL values for some of the algorithms.

The ranking of these algorithms with respect to the average processor usage is {*IGA*, *GA*, *LMT*, *HEFT*, *MH*, *DLS*}. For 100 nodes, the processor usage of non-GA methods is close to 40, which is the maximum number of processors in the experiments. On average, the non-GA methods allocate 139% more processors than the IGA method and 73% more processors than our method. Our method outperforms the IGA method by 37% with respect to average normalized schedule lengths. Additionally, the HEFT algorithm gives the best NSL values, which is consistent with the results given in their original paper [5].

For high-task, high-machine heterogeneity type (HiHi in Table 1), our method and the IGA method outperform the other methods up to 19% with respect to the NCost values. Our method outperforms the IGA method when $n \geq 40$,

TABLE 2. Comparison of the algorithms with respect to the CCR values for the LoLo and HiHi heterogeneity types

		LoLo CCR value						HiHi CCR value					
Algorithm	Metric	0.1	0.2	0.5	1.0	2.0	5.0	0.1	0.2	0.5	1.0	2.0	5.0
GA	NCost	0.68	0.67	0.68	0.73	0.80	1.05	0.80	0.78	0.81	0.82	0.83	0.90
	NSL	2.06	2.04	2.09	2.50	3.43	6.22	2.41	2.14	2.39	2.59	2.50	3.38
	PE	23.68	23.58	23.79	23.88	23.01	21.64	28.27	28.75	28.60	28.38	28.57	28.28
IGA	NCost	0.63	0.64	0.68	0.75	0.90	1.32	0.81	0.80	0.81	0.83	0.87	1.02
	NSL	3.00	3.15	3.45	3.98	5.30	8.99	4.15	4.04	4.29	4.33	4.55	5.88
	PE	16.61	16.42	16.75	17.77	18.46	21.43	19.72	20.09	19.32	20.00	20.61	21.65
HEFT	NCost	0.87	0.87	0.88	0.91	0.98	1.23	0.86	0.86	0.86	0.87	0.88	0.94
	NSL	1.07	1.12	1.32	1.73	2.65	5.60	1.04	1.05	1.10	1.17	1.34	2.06
	PE	38.03	37.81	37.47	36.91	36.01	33.53	37.84	37.81	37.74	37.62	37.41	36.93
DLS	NCost	0.87	0.88	0.89	0.92	1.00	1.26	0.87	0.87	0.87	0.88	0.89	0.96
	NSL	1.07	1.13	1.33	1.74	2.67	5.66	1.05	1.06	1.10	1.18	1.34	2.07
	PE	38.30	38.15	37.93	37.55	36.85	34.54	38.24	38.23	38.19	38.09	37.94	37.68
MH	NCost	0.87	0.87	0.88	0.91	0.99	1.26	0.86	0.86	0.86	0.87	0.88	0.95
	NSL	1.08	1.14	1.33	1.75	2.67	5.67	1.05	1.06	1.10	1.17	1.34	2.07
	PE	37.89	37.68	37.41	37.01	36.25	34.46	37.75	37.72	37.67	37.63	37.45	37.16
LMT	NCost	0.88	0.88	0.89	0.92	1.01	1.36	0.91	0.91	0.91	0.91	0.93	1.02
	NSL	1.20	1.25	1.48	1.98	3.10	6.84	1.46	1.47	1.51	1.59	1.77	2.73
	PE	37.94	37.62	37.10	36.38	35.22	33.64	38.28	38.25	38.13	37.98	37.76	37.26

where n is the number of nodes. The ranking of the algorithms with respect to average processor usage becomes {*IGA*, *GA*, *HEFT*, *MH*, *DLS*, *LMT*}.

In order to provide comparative study on running times, we consider only the graphs with 100 tasks (i.e. the maximum number of tasks in the experimental study). It is observed that the non-GA methods take significantly less time than the GA-based methods for various graph characteristics; and the IGA algorithm requires up to 133% more running time than our GA-based algorithm. As a specific example, if $\alpha = 10$ when $n = 100$, our GA-based algorithm takes 2.11 s on the average, the IGA algorithm takes 3.52 s, the DLS algorithm takes 0.023 s, the HEFT algorithm takes 0.002 s, the LMT algorithm takes 0.003 seconds and the MH algorithm takes 0.0013 s. The ranking of the algorithms with respect to running times is preserved for various graph sizes and characteristics.

CCR. This experiment aims to compare the performance of algorithms with respect to various CCR values by considering graphs with 100 nodes. The shape coefficient is equal to 1.0 and the fan-out degree range is equal to 5. The results of the LoLo and the HiHi heterogeneity types are presented in Table 2.

When the results of the heuristics are considered, average NSL values increase with an increase in CCR values. A higher CCR value increases the average communication cost in the graph, which causes an increase in the schedule length. Increase in the schedule length will increase the numerator part of the NSL given in Equation (5) but the lower bound does not change. Therefore, the NCost values increase with increase in NSL values.

For the LoLo heterogeneity type, the ranking of these algorithms with respect to the NCosts is {*GA*, *IGA*, *HEFT*, *MH*, *DLS*, *LMT*}. Our algorithm is better than the IGA algorithm by 7%, and it is better than the other methods up to 28%. For computation-intensive applications (i.e. when CCR = 0.1), our method has slightly worse performance than the IGA algorithm, with respect to the NCost values. For communication-intensive applications (i.e. when CCR = 0.5), it is better than the IGA algorithm by 26%, which is due to 44% lower NSL value. Additionally, a significant increase in CCR value decreases the processor usage for all algorithms except the IGA algorithm. For communication-intensive applications, algorithms tend to cluster the tasks into the fewer number of processors in order to decrease the inter-task communication costs.

For the HiHi heterogeneity type, there is not a significant difference in the normalized cost values of the algorithms for various CCRs; and our GA-based method gives the lowest NCost values. For a high machine heterogeneity case, it may not be possible to cluster the tasks into less number of processors, since our task model supports inconsistent matrices for the computation costs. In an inconsistent matrix, a processor may provide the highest computation costs for some of the tasks and the lowest computation costs for

TABLE 3. Comparison of the algorithms with respect to the shape coefficient of the graphs for the LoLo and HiHi heterogeneity types

		LoLo Shape coefficient					HiHi Shape coefficient				
Algorithm	Metric	0.1	0.2	1.0	5.0	10.0	0.1	0.2	1.0	5.0	10.0
GA	NCost	13.32	1.12	0.73	0.89	0.95	13.97	1.26	0.81	0.94	1.01
	NSL	3.90	2.61	2.58	3.12	3.33	9.57	2.63	2.40	2.74	3.30
	PE	16.16	20.10	23.53	29.17	30.99	16.27	24.71	28.79	33.36	33.86
IGA	NCost	9.97	0.88	0.86	1.20	1.32	13.76	1.27	0.97	1.41	1.57
	NSL	3.65	3.16	4.04	6.60	7.37	10.80	3.53	4.36	8.26	9.52
	PE	12.01	18.70	22.68	27.29	29.13	15.85	22.95	26.74	29.46	30.91
HEFT	NCost	28.01	1.24	0.91	0.97	0.97	29.31	1.59	0.87	0.91	0.92
	NSL	1.68	1.75	1.73	1.73	1.76	1.27	1.23	1.17	1.19	1.23
	PE	34.79	35.45	36.91	39.91	40.00	36.48	36.66	37.61	39.84	40.00
DLS	NCost	28.01	1.24	0.92	0.98	0.98	29.31	1.59	0.88	0.92	0.92
	NSL	1.68	1.75	1.74	1.78	1.83	1.27	1.23	1.18	1.22	1.24
	PE	34.79	35.58	37.54	40.00	40.00	36.48	36.79	38.09	40.00	40.00
MH	NCost	28.01	1.24	0.91	0.97	0.98	29.31	1.59	0.87	0.91	0.92
	NSL	1.68	1.76	1.75	1.77	1.83	1.28	1.23	1.18	1.19	1.23
	PE	34.79	35.47	37.01	39.93	40.00	36.48	36.63	37.63	39.73	40.00
LMT	NCost	28.01	1.22	0.92	1.11	1.16	29.31	1.60	0.92	1.21	1.31
	NSL	1.68	1.84	1.98	3.12	3.60	1.27	1.32	1.59	4.09	5.09
	PE	34.81	34.41	36.38	40.00	40.00	36.48	36.63	37.98	40.00	40.00

others. Additionally, there exits a significant difference in computation costs over various processors, if high machine heterogeneity is considered. The IGA method requires less number of processors than the other methods, which causes significantly higher NSL value.

Graph structures. This experiment compares the performance of the algorithms with respect to the five values of the shape coefficient, $\alpha = \{0.1, 0.2, 1.0, 5.0, 10.0\}$. When $\alpha = 0.1$, the graphs with higher-depth and the minimum parallelism degree are generated; whereas the last choice is for generating the graphs with lower-depth and the maximum parallelism degree. Our rationale for using graphs with various shapes is to avoid any bias that an algorithm may have toward a particular graph shape or structure. We consider graphs with 100 tasks, where the CCR value is 1.0. The fan-out degree range is equal to 5, and the number of processors in the system is equal to 40.

When the NSL values in Table 3 are examined, the highest values are observed when $\alpha = 0.1$. A sharp decrease occurs in the NSL values for the algorithms, with an increase in α. If the other parameters are set with fixed values by considering sufficient processors in the system, the graphs with the high-depth values will have more number of inter-processor communications due to the more number of levels. Therefore, the schedule length of each output will be significantly higher than its lower bound, which will cause higher NSL values for all algorithms. When the graphs with the maximum-parallelism degree are considered, there are more number of tasks that are not on the critical path. However, the number of inter-communications is limited due to the low-depths, when we consider sufficient processor availability.

If we consider the cases when $\alpha = 0.1$, the NCost values are significantly higher (NCost $\gg$ 1.0) than the other cases, which is due to high NPU values (see Equation (6)). Since α is the *root* of the ratio of the average width to the average height of a graph, setting α to 0.1 may generate graphs with close to 100 levels with a single node at each level. Our model supports inconsistent matrices of the computation costs. Therefore, different processors may give lowest computation costs for different tasks. Consequently, the number of processors allocated will be high, even if the graph has very low parallelism degree.

For the LoLo heterogeneity type, the GA-based methods significantly outperform the other methods with respect to the NCost values. An increase in the value of α increases the processor usage and decreases the NCost value, for all algorithms considered. Specifically, when $\alpha = 5.0$, each constructed graph includes a few levels of tasks, where each intermediate level (other than the entry and the exit levels) has high number of tasks. Therefore, when a graph is constructed with 100 tasks and $\alpha \geq 5.0$, the number of tasks at one or more intermediate levels is greater than 40, which is the maximum number of available processors in the experiments. Therefore, the values at the PE rows of Table 3 for

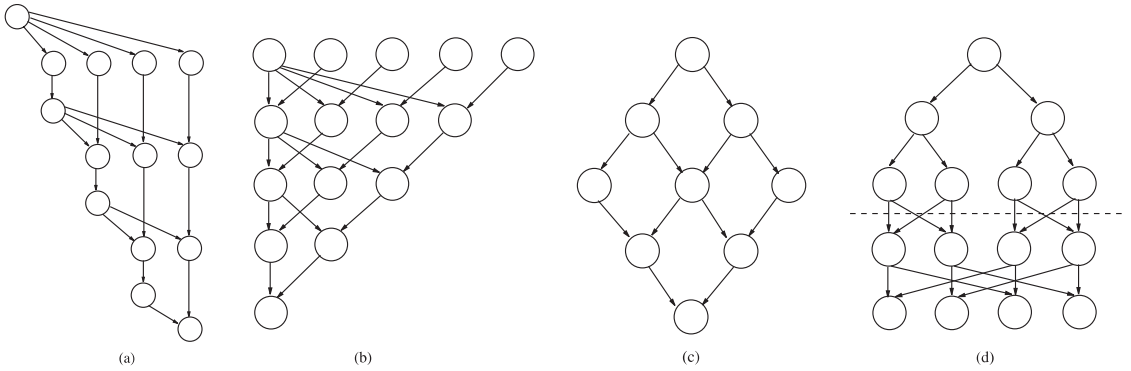


FIGURE 8. Sample task graphs of the selected applications. (a) Gauss elimination. (b) Gauss Jordan. (c) Laplace equation solver. (d) Fast Fourier transformation.

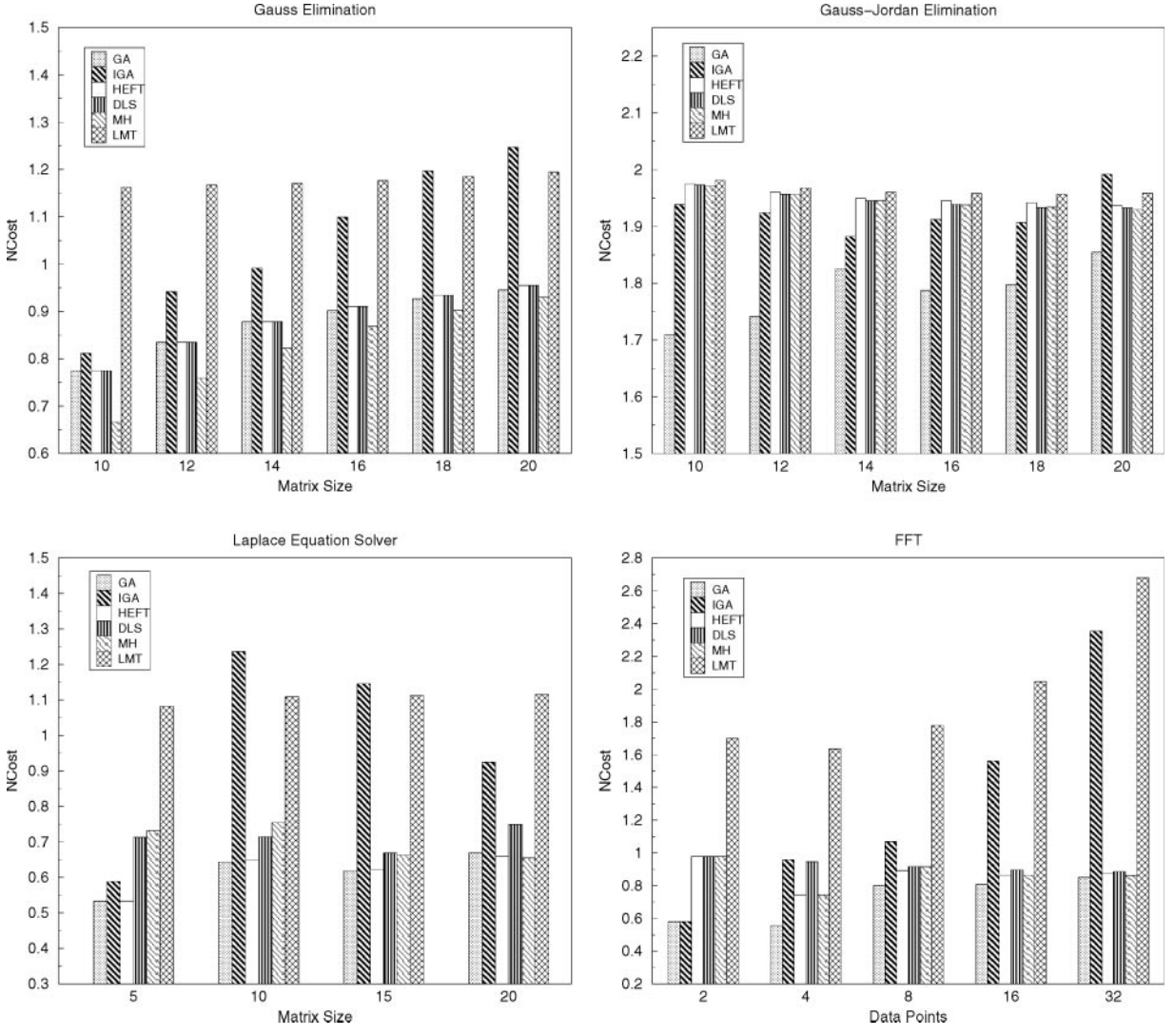


FIGURE 9. Performance of the algorithms for the task graphs of the selected applications.

non-GA algorithms are close to 40. The results of the HiHi heterogeneity type is consistent with the results of the LoLo heterogeneity type. The processor usages are higher than those of LoLo cases for all algorithms, which is due to the significant variance on computation costs of the tasks among all processors.

5.3. Experimental evaluation based on task graphs of well-known problems

In this section, we consider task graphs that capture the representations and features of the four well-known problems, which are the Gauss elimination [39, 40], the Gauss–Jordan elimination [41], the Laplace equation solver [39] and the fast Fourier transformation (FFT) [23, 42]. Since the structures of the graphs for the given applications are known, several graph parameters including α, number of tasks, fan-out degree are not considered during the graph generation phase. A new parameter, matrix size m, is considered in place of the number of tasks. The CCR value is set to 1.0 for all cases.

Figure 8a and b give the task graph of the Gauss elimination and the Gauss–Jordan algorithms respectively, when the dimension of the matrix is equal to 5. Similarly, Figure 8c gives the task graph of a parallel Laplace application based on the Gauss–Seidel algorithm. The task graph of a recursive, 1D FFT algorithm [23, 42] for the case of four data points is given in Figure 8d. This graph can be divided into two parts—the tasks above the dashed line are the recursive call tasks and the ones below the line are the butterfly operation tasks. For an input vector of size m, there are $2 \times m - 1$ recursive call tasks and $m \times \log_2 m$ butterfly operation tasks.

When the results presented in Figure 9 are examined, our method outperforms the other algorithms for all applications except the Gauss elimination. The IGA and the LMT algorithms generate significantly higher NCost values than the others, which is due to the higher processor usages for the LMT algorithm and the higher schedule lengths for the IGA algorithm. For the Gauss elimination case, the MH algorithm provides better results than our algorithm, when graphs with low matrix sizes are considered. It requires on the average of 32% less processor usage and 26% higher schedule length than our algorithm, which generates better Ncost values.

6. CONCLUSIONS AND FUTURE WORK

In this paper, a GA is presented for the task scheduling problem by considering a new objective of the cost minimization, which is generated by unifying the schedule length minimization and the processor usage minimization in a given weighting scheme. Based on the experimental study with both the randomly generated task graphs and the graphs of real applications, our GA-based method outperforms the other heuristics with respect to the normalized cost values for various graph characteristics.

It is planned to diversify the coefficients given in the normalized cost equation and to measure the effects of the two objectives on the performance of our algorithm. Another future work is to study the minimization of the normalized cost values by keeping the schedule length values in a predefined neighborhood of the corresponding lowest values. For this extension, it is planned to use constraint handling strategies within the GA framework.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees for their constructive suggestions in improving the quality of the paper. A preliminary work of this research [43] was presented at the Second International Conference on Advances in Information Systems, 2002.

REFERENCES

[1] Garey, M. R. and Johnson, D. S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., New York.

[2] Ullman, J. D. (1975) NP-Complete scheduling problems. *J. Comput. Syst. Sci.*, **10**, 384–393.

[3] Hu, T. C. (1961) Parallel sequencing and assembly line problems. *Operat. Res.*, **9**, 841–848.

[4] Coffman, E. G. (1976) *Computer and Job-Shop Scheduling Theory*, John Wiley and Sons, New York.

[5] Topcuoglu, H., Hariri, S. and Wu, M. (2002) Performance effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parall. Distrib. Syst.*, **13**, 260–274.

[6] Sih, G. C. and Lee, E. A. (1993) A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parall. Distrib. Syst.*, **4**, 175–186.

[7] El-Rewini, H. and Lewis, T. G. (1990) Scheduling parallel program tasks onto abitrary target machines. *J. Parall. Distrib. Comput.*, **9**, 138–153.

[8] Iverson, M., Ozguner, F. and Follen, G. (1995) Parallelizing existing applications in a distributed heterogeneous environment. In *Proc. Heterogeneous Computing Workshop*, Santa Barbara, CA, April 25, pp. 93–100. IEEE Computer Society, Los Alamitos.

[9] Maheswaran, M. and Siegel, H. J. (1998) A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Proc. Heterogeneous Computing Workshop*, Orlando, FL, March 30, pp. 57–69. IEEE Computer Society, Los Alamitos.

[10] Wang, L., Siegel, H. J. and Roychowdhury, V. P. (1996) A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments. In *Proc. Heterogeneous Computing Workshop*, Honolulu, HI, April 15–19, pp. 72–85. IEEE Computer Society, Los Alamitos.

[11] Singh, H. and Youssef, A. (1996) Mapping and scheduling heterogeneous task graphs using genetic algorithms. In *Proc.*

of Heterogeneous Computing Workshop, Honolulu, HI, April 15–19, pp. 86–97. IEEE Computer Society, Los Alamitos.

[12] Hagras, T. and Janecek, J. (2005) A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. *Parall. Comput.*, **31**, 653–670.

[13] Qin, X., Jiang, H. and Swanson, D. R. (2002) An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *Proc. Int. Conf. Parallel Processing*, Vancouver, Canada, August 18–21, pp. 360–368. IEEE Computer Society, Los Alamitos.

[14] Qin, X., Jiang, H., Zhu, Y. and Swanson, D. R. (2003) Dynamic load balancing for I/O-intensive tasks on heterogeneous clusters. In *Proc. Int. Conf. High Performance Computing*. Hyderabad, India, December 17–20, pp. 300–309. Springer-Verlag, Berlin.

[15] Boeres, C. and Lima, A. (2003) Hybrid task scheduling: integrating static and dynamic heuristics. In *Proc. 15th Symp. Computer Architecture and High Performance Computing, SBAC-PAD03*, Sao Paulo, Brazil, November 10–12, pp. 199–206. IEEE Computer Society, Los Alamitos.

[16] Sakellariou, R. and Zhao, H. (2004) A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Proc. Int. Parallel and Distributed Processing Symp., IPDPS 2004*, Santa Fe, NM, April 26–30, IEEE Computer Society, Los Alamitos.

[17] Liu, G. Q., Poh, K. L. and Xie, M. (2005) Iterative list scheduling for heterogeneous computing. *J. Parall. Distrib. Comput.*, **65**, 654–665.

[18] Ge, Y. and Yun, D. (1996) Simultaneous compression of makespan and number of processors using CRP. In *Proc. Int. Parallel Processing Symp.*, Honolulu, HI, April 15–19, pp. 332–338. IEEE Computer Society, Los Alamitos.

[19] Roig, C., Ripoll, A., Senar, M., Guirado, F. and Luque, E. (2002) A new model for static mapping of parallel applications with task and date parallelism. In *Proc. of Int. Parallel and Distributed Processing Symp., IPDPS 2002*, Fort Lauderdale, FL, April 15–19, pp. 78–85. IEEE Computer Society, Los Alamitos.

[20] Han, J. and Li, Q. (2004) A novel static task scheduling algorithm in distributed computing environment. In *Proc. Int. Parallel and Distributed Processing Symp., IPDPS 2004*, Santa Fe, NM, April 26–30. IEEE Computer Society, Los Alamitos.

[21] Dilworth, R. P. (1950) A decomposition theorem for partially ordered sets. *Ann. Math.*, **51**, 161–166.

[22] Abdeddaim, Y., Kerbaa, A. and Maler, O. (2003) Task graph scheduling using timed automata. In *Proc. Int. Parallel and Distributed Processing Sympo., IPDPS 2003*, Nice, France, April 22–26. IEEE Computer Society, Los Alamitos.

[23] Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2001) *Introduction to Algorithms,* Second Edition, The MIT Press, Cambridge.

[24] Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

[25] Mitchell, M. (1998) *An Introduction to Genetic Algorithms*. Paperback Edition, MIT Press, Cambridge.

[26] Michalewicz, Z. (1999) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin.

[27] Eiben, A. E. and Smith, J. E. (2003) *Introduction to Evolutionary Computing*, Springer-Verlag, Berlin.

[28] Back, T., Fogel, D. B. and Michalewicz, Z. (1997) *Handbook of Evolutionary Computation*, Physics Publishing and Oxford University Press, New York.

[29] Hou, E. S. H., Ansari, N. and Ren, H. (1994) A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parall. Distrib. Syst.*, **5**, 113–120.

[30] Shroff, P., Watson, D. W., Flann, N. S. and Freund, R. (1996) Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In *Proc. Heterogeneous Computing Workshop*, Honolulu, HI, April 15–19, pp. 98–104. IEEE Computer Society, Los Alamitos.

[31] Kwok, Y. and Ahmad, I. (1997) Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithms. *J. Parall. Distrib. Comput.*, **47**, 58–77.

[32] Ahmad, I. and Dhodhi, M. K. (1996) Multiprocessor scheduling in a genetic paradigm. *Parall. Comput.*, **22**, 395–406.

[33] Dhodhi, M. K., Ahmad, I. and Storer, R. (1995) SHEMUS: synthesis of heterogeneous multiprocessor systems. *Microprocess. Microsys.*, **19**, 311–319.

[34] Correa, R. C., Ferreira, A. and Rebreyend, P. (1996) Integrating list heuristics into genetic algorithms for multiprocessor scheduling. In *Proc. IEEE Symp. Parallel and Distributed Processing*, New Orleans, Louis, October, pp. 462–469, IEEE Computer Society, Los Alamitos.

[35] Correa, R. C., Ferreira, A. and Rebreyend, P. (1999) Scheduling multiprocessor tasks with genetic algorithms. *IEEE Trans. Parall. Distrib. Sys.*, **10**, 825–837.

[36] Wu, A. S., Yu, H., Jin, S., Lin, K. and Schiavone, G. (2004) An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Trans. Parall. Distrib. Syst.*, **15**, 824–834.

[37] Rebreyend, P., Sandnes, F. E. and Megson, G. M. (1998) *Static Multiprocessor Task Graph Scheduling in the Genetic Paradigm: A Comparison of Genotype Representations*, Ecole Normale Supérieure de Lyon, Laboratoire de l'Informatique du Parallélisme, Lyon, France, Research Report No. 98-25.

[38] Sandnes, F. E. and Megson, G. M. (1997) Improved static multiprocessor scheduling using cyclic task graphs: a genetic approach. In *Proc. PARCO 97*, Bonn, Germany, September 19–22, pp. 703–710. Elsevier, North-Holland, Amsterdam.

[39] Wu, M. and Gajski, D. (1990) Hypertool: a programming aid for message passing systems. *IEEE Trans. Parall. Distrib. Sys.*, **1**, 330–343.

[40] Cosnard, M., Marrakchi, M., Robert, Y. and Trystram, D. (1988) Parallel Gaussian elimination on an MIMD computer. *Parall. Comput.*, **6**, 275–295.

[41] Gerasoulis, A. and Yang, T. (1994) Performance bounds for parallelizing Gaussian-Elimination and Gauss-Jordan on message-passing machines. *Appl. Numer. Math. J.*, **16**, 283–297.

[42] Chung, Y. and Ranka, S. (1992) Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In *Proc. Supercomputing*, Minneapolis, MN, November 16–20, pp. 512–521. IEEE Computer Society, Los Alamitos.

[43] Topcuoglu, H. and Sevilmis, C. (2002) Task scheduling with conflicting objectives. In *Proc. Int. Conf. Advances in Information Systems, ADVIS 2002*, Izmir, Turkey, October 23–25, pp. 346–355. Springer-Verlag, Berlin.