CCF Transactions on High Performance Computing
https://doi.org/10.1007/s42514-024-00192-0

REGULAR PAPER

CPTF–a new heuristic based branch and bound algorithm for workflow scheduling in heterogeneous distributed computing systems

D. Sirisha[1] · S. Sambhu Prasad[1]

Received: 10 November 2023 / Accepted: 16 April 2024

Abstract
Computationally intensive applications embodied as workflows entail interdependent tasks that involve multifarious computation requirements and necessitate Heterogeneous Distributed Computing Systems (HDCS) to attain high performance. The scheduling of workflows on HDCS was demonstrated as an NP-Complete problem. In the current work, a new heuristic based Branch and Bound (BnB) technique namely Critical Path_finish Time First (CPTF) algorithm is proposed for workflow scheduling on HDCS to achieve the best solutions. The primary merits of CPTF algorithm are due to the bounding functions that are tight and of less complexity. The sharp bounding functions could precisely estimate the promise of each state and aid in pruning infeasible states. Thus, the search space size is reduced. The CPTF algorithm explores the most promising states in the search space and converges to the solution quickly. Therefore, high performance is achieved. The experimental results on random and scientific workflows reveal that CPTF algorithm could effectively exploit high potency of BnB technique in realizing better quality solutions against the widely referred heuristic scheduling algorithms. The results on the benchmark workflows show that CPTF algorithm has improved schedules for 89.36% of the cases.

Keywords Workflow scheduling · Task scheduling · Heuristics · Heterogeneous distributed computing systems · Branch and bound technique · Makespan

1 Introduction

Multifaceted applications that include modelling, extensive simulations, and experiments to research high energy physics, chemical reactions and structural biology consisting interdependent tasks are manifested as workflows. The computational complexity and heterogeneity in the processing requirements of the tasks demand heterogeneous distributed computing systems (HDCS) for multifarious computation needs. HDCS is expeditiously evolving as a significant enabling technology in contemporary computing propelled for realizing high performance economically. HDCS involves a diverse set of processors with varying capabilities and performance characteristics. Moreover, communicating tasks and data movement among processors can result in high communication overhead. Efficient scheduling is essential to minimize this overhead to enhance system performance (Ahmad and Bashir 2020; Khojasteh Toussi et al. 2022; Kung et al. 2022; Sirisha 2023; Topcuoglu et al. 2002; Sirisha and Prasad 2022).

Gary and Johnson (1979) proved that scheduling workflows is a well-acknowledged NP-Complete problem and is more challenging in HDCS due to the heterogeneity in the processors. Developing an effective approach for scheduling workflows on HDCS while minimizing the complexity is a challenging research problem. (Topcuoglu et al. 2002; Illavarasan and Thambidurai 2007). Workflow scheduling determines the sequence of execution of the tasks and decides the start time of each task on a processor. In the current work, a new Branch and Bound (BnB) technique based Critical Path_finish Time First (CPTF) algorithm is proposed for scheduling workflows. Generally, BnB technique is restricted to smaller problems. Nevertheless, emergence of HDCS has enabled BnB technique to handle large problems by efficiently exploring the parallelism of HDCS.

✉ D. Sirisha
sirishad998@gmail.com

S. Sambhu Prasad
drssp@gmail.com

[1] Nadimpalli Satyanarayana Raju Institute of Technology, Sontyam, Visakhapatnam, Andhra Pradesh, India

Enumerative techniques like the BnB technique are usually used to solve combinatorial optimization problems and arrive at optimum solutions. This is an effective method because, in most circumstances, it can produce solutions rather quickly without thoroughly examining the search space. However, worst cases may lead to the degeneration of search to the exhaustive enumeration of all possible cases. Jonsson and Shin (1997) opined that BnB approach can typically handle NP-hard problems effectively in average cases, but the worst-case time complexity is still exponential.

Kasahara and Narita (1985) and Fujita (2011) employed BnB technique to optimize the workflow schedules. The study on the available literature on BnB emphasizes that designing of efficient bounding functions is essential to overcome the main difficulty of minimizing the turnaround time, otherwise it may explore large search space in the worst case. Vempaty et al. (1992); Zhang and Korf (1991) studied that tight bounding functions can accurately estimate the promise of each state that may lead to the potential reduction in the search space size. Tight bounds abet in eliminating the subtrees that indicate inferior solutions. However, more computation are required at each state in order to design tight bounds. Another important consideration is the complexity of finding the bounds for every state which has adverse effect on turnaround time. However, by selecting the globally optimal state in the search space, search for an optimal solution can be speeded up to possibly minimize search space (Zhang and Korf 1991). Selecting the most promising state in the search space can result in the convergence of a solution rapidly with fewer states being explored. The possibility of arriving at the optimal solution may be less, but there might a good chance of quickly reaching the solution state. Considering these aspects, a heuristic based BnB technique viz., Critical Path_finish Time First Algorithm (CPTF) algorithm is proposed. The premise of CPTF algorithm is that tight bounds with less complexity can alleviate search space size. Thus, reduces the turnaround time.

The chief contributions of this work are mentioned below.

- A heuristic based BnB technique is proposed to generate better schedules.
- Lower and upper bounds are devised to evaluate the potential of each state generated in the search space.
- The proposed lower and upper bounds are observed to be simple, tight, and less complex.
- Best first search technique is adopted to select the globally best states in the search space.
- Simulation experiments are conducted on random and scientific workflows to validate the proposed approach against the widely referred scheduling algorithms.

The remaining paper is organized as follows. In Sect. 2, workflow scheduling problem on HDCS is detailed with the required terminology. Sect. 3 elaborates BnB technique. The literature on current topic is studied in Sect. 4. The proposed CPTF algorithm is illustrated in Sect. 5. The performance analysis of CPTF algorithm is discussed in Sect. 6. Section 7 summarizes the proposed work with an emphasis on the future scope.

2 The workflow scheduling problem

The workflow scheduling system constitutes a workflow model, an HDCS model, and a scheduling approach. The list of variables along with the notations used with descriptions are given in Table 1.

2.1 Workflow model

Basically, workflow W signifies a weighted Directed Acyclic Graph (DAG), W = <T, E> where T indicates a set $t_i \in$ T, $1 \le i \le n$ tasks and E is a set $e_{i,j} \in$ E of e directed edges between tasks t_i and t_j, t_i, $t_j \in$ T. Each edge $e_{i,j}$ between tasks t_i and t_j, $i \neq j$, enforces dependency constraint among them, consequently t_j can start its execution once its predecessor t_i is completed. An $n \times n$ matrix D denotes the dataflow time i.e., the time required to transfer data between the tasks, where each element in the matrix $d_{i,j}$ indicates the dataflow between the tasks t_i and t_j. A task is recognized as ready when its dependency constraints are satisfied. A task without any immediate predecessor is identified as a start task indicated as t_{start}, whereas a task without any immediate successor is a sink task denoted as t_{sink}. Usually, a workflow consists of a pair of start and sink tasks. In cases, when workflow comprises of numerous start and sink tasks, then these tasks are connected to pseudo start task and sink task having zero processing time with pseudo edges having zero dataflow times so that schedule length remains unaffected. For each task t_i, its immediate predecessor and successor are designated as $pred(t_i)$ and $succ(t_i)$, respectively. The actual finish time (AFT) of sink task of the workflow is defined as the schedule length, or makespan. Figure 1 depicts a sample workflow with six tasks.

2.2 HCDCS model

The HDCS model consists of a set $p_k \in$ P of m processors, $1 \le k \le m$. HDCS model is assumed as fully connected where interprocessor communications are performed without contention. It is supposed that computations and dataflows between the tasks are executed concurrently. Besides, task executions are not preemptive. The processing

Table 1 List of notations

Notation/variable	Definition
T	Set of tasks in the workflow
n	Number of tasks in a workflow
t_i	i^{th} task in the workflow
t_{start}	Start task of the workflow
t_{sink}	Sink task of the workflow
$w_{i,k}$	Processing time of task t_i on processor p_k
E	Set of edges in the workflow
e	Number of edges in the workflow
$e_{i,j}$	Directed edge connecting the tasks t_i and t_j
$d_{i,j}$	Dataflow time between the tasks t_i and t_j
P	Set of heterogeneous processors on HDCS
m	Number of processors on HDCS
p_k	k^{th} processor on HDCS
$succ(t_i)$	Set of immediate successors of task t_i
$pred(t_i)$	Set of immediate predecessors of task t_i
$EST(t_i,p_k)$	Earliest start time of task t_i on processor p_k
$EFT(t_i,p_k)$	Earliest finish time of task t_i on processor p_k
$AFT(t_i)$	Actual finish time of task t_i
$ready\_time(t_i)$	Time at which the task t_i is ready for execution
$avail\_time(p_k)$	Time at which the processor p_k is available to execute the next task
CP_l	Length of the critical path (CP) in a workflow
CP_{min}	Sum of the minimum processing time of each task on CP
S	State selection operator
B	Branching operator
BO	Bounding function
Pr	Pruning rule
LB(N_i)	Lower Bound (LB) of the state N_i
UB(N_i)	Upper Bound (UB) of the state N_i
UB_{best}	Least UB among all the states explored in the search space

time of n tasks on m processors is represented by a $n \times m$ processing time matrix C, where each element $w_{i,k}$ represents the estimated processing time of a task t_i on processor p_k, $1 \leq i \leq n$, $1 \leq k \leq m$. Table 2 presents the processing time of six tasks of the workflow shown in Fig. 1 on two processors p_1 and p_2 respectively.

Beforehand, each workflow task is labelled with a positive integer which signifies its average processing time. The following equation is used to calculate the average processing time ($\overline{w}_i$) of a task t_i on m processors.

$$\overline{w}_i = \sum_{k=1}^{m} w_{i,k}/m \tag{1}$$

$w_{i,k}$ is the processing time of a task t_i on processor p_k. Similarly, all edges are labelled with non-negative weight indicating dataflow time $d_{i,j}$ between tasks t_i and t_j.

Earliest start time (EST) and earliest finish time (EFT) are the two most important criteria essential to define a workflow scheduling problem. EST (t_i, p_k) and EFT (t_i, p_k) indicate the EST and EFT of task t_i on processor p_k respectively. EST (t_{start}, p_k) is 0 for t_{start}. The EST for the remaining tasks is calculated using ready time of task t_i, represented as $ready\_time(t_i)$, and available time of processor p_k, denoted as $avail\_time(p_k)$, (Kwok and Ahmad 1999). The EST (t_i, p_k) is computed as follows.

$$\text{EST}(t_i, p_k) = max\{ready\_time(t_i),\ avail\_time(p_k)\} \tag{2}$$

where $ready\_time(t_i)$ is the earliest time at which processor p_k has received all the data required to execute the task t_i, and $avail\_time(p_k)$ is the earliest time at which processor p_k is available to execute task t_i. $ready\_time(t_i)$ is computed as follows.

$$ready\_time(t_i) = \max_{t_h \in pred(t_i)} \{AFT(t_h) + d_{h,i}\} \tag{3}$$

where t_h is a set of immediate predecessors of task t_i, designated as $pred(t_i)$, AFT(t_h) is the actual finish time

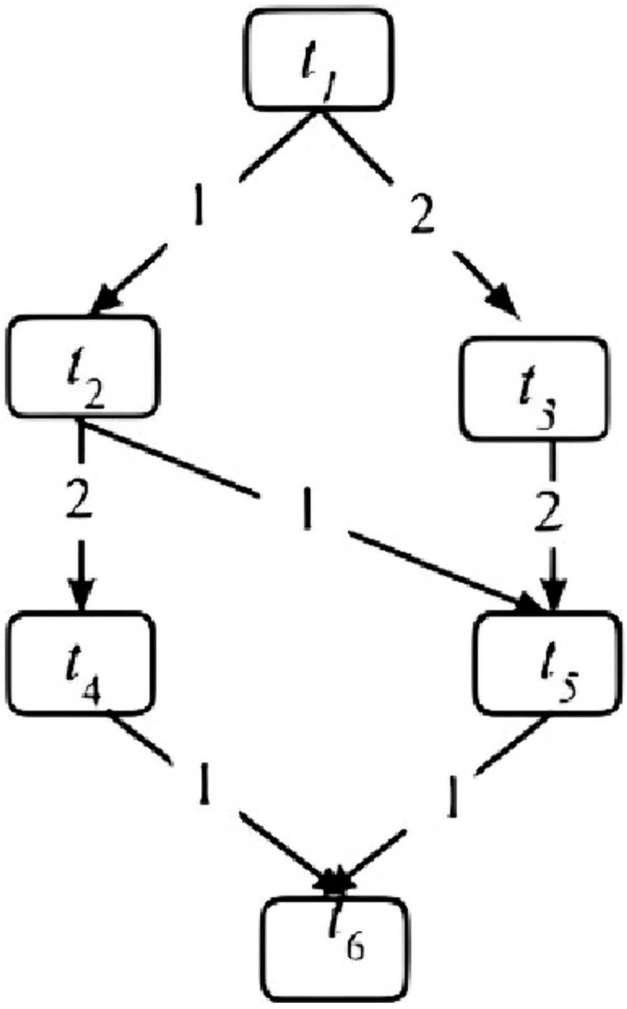

Fig. 1 An example workflow

Table 2 Processing time matrix

Task	p_1	p_2
t_1	2	3
t_2	8	5
t_3	9	13
t_4	4	6
t_5	4	3
t_6	2	3

(AFT) of the task t_h, and $d_{h,i}$ is the dataflow time from the task t_h to task t_i. The EFT(t_i, p_k) is computed as follows.

$$EFT(t_i, p_k) = EST(t_i, p_k) + w_{i,k} \tag{4}$$

where $w_{i,k}$ is the processing time of task t_i on processor p_k and EST(t_i, p_k) is already defined in Eq. 2. The makespan of the workflow, often known as the schedule length, is defined as AFT of t_{sink}.

$$makespan = AFT(t_{sink}) \tag{5}$$

Definition 1 (Critical path) The critical path (CP) of the workflow is the lengthiest path from start task to the sink task. The length of the CP is computed as the sum of the average processing times of the tasks on CP and dataflow times between the tasks on CP and denoted as CP_l.

Definition 2 (Minimum CP) It is computed by the sum of the minimum processing times of the tasks on CP and denoted by CP_{min}.

Definition 3 (Partial schedule) The tasks $T' \subset T$ are assigned to processors constitute a partial schedule. A task that is assigned to a processor is listed in the partial schedule.

3 Branch and bound technique

Branch and Bound (BnB) is an algorithm design approach usually used to address combinatorial optimization problems. Generally, BnB is employed for solving NP-hard problems to obtain optimal solutions (Kasahara and Narita 1985). BnB is an implicit enumeration method that builds a tree-based search space. Scheduling workflows on HDCS using BnB technique is devised as search procedure in the search space which considers all potential combinations of task-to-processor mappings. The representation of the search space is a tree with a collection of nodes, where each node denotes a state and edges between the nodes shows a legitimate move between states.

Each state in the search space denotes the assignment of a task to a processor. The start state of the search space indicates a void schedule and the next level of states emerge by assigning the start task to m processors. The search space consists of l levels, $1 \le l \le n$. At each level, a task is assigned to all processors. And at a level l, l^{th} task is being taken into consideration for mapping while ($l-$ 1) tasks are already scheduled. The intermediary states signify the partial schedules and leaf states indicate a possible solution where a t_i is mapped to a processor p_j.

BnB technique works primarily in two main procedures, viz., branching and bounding. The branching method divides a problem into small sub-problems which are represented as states in the search space. Bounding method determines the lower and upper bounds on the solution for each sub-problem. In a process known as pruning, the entire sub-problem can be eliminated when bounds on a sub-problem reveal that it only includes inferior solutions.

The competence of BnB technique is due to the alternating branching and bounding methods. Moreover, pruning the sub-problem at earlier stages of the search space results in significant progress. As the search space progresses, generated states are placed in either of the lists mentioned below.

- LIVE list—states that are created and are not evaluated; their children are not yet generated.
- CLOSE list—states which are earlier investigated, and all their child nodes are created.

Kohler and Steiglitz (1974) represented the problem using four-tuple $< S, B, BO, Pr >$, where S is state selection operator, B is branching operator, BO is bounding function, and Pr is pruning rule. Initially, operator S selects a state from LIVE list for exploration. Generally, selected state is explored by either of the techniques Breadth First Search (BFS), Depth First Search (DFS), Best First Search (BeFS), Heuristics. B operator divides a problem into smaller sub-problems and the chosen state is branched to produce new states in the search space. For a state $\mathrm{N_i}$, new states are generated for each unique mapping of a task to processor. The newly created states are added to the list of LIVE states. The states are expanded until there are no states in LIVE list. Subsequently, BO function is applied to each new state $\mathrm{N_i}$ to compute Lower Bound (LB) and Upper Bound (UB), denoted as $LB(N_i)$ and $UB(N_i)$ respectively. Lastly, Pr operator is used to estimate every state $\mathrm{N_i}$ using the bounds and the state $\mathrm{N_i}$ is pruned from the LIVE list if found as unpromising. For applying Pr rule, lowest UB value across all created states is identified as the best UB value globally and is signified as UB_{best}. If $LB(N_i) > UB_{best}$ for state $\mathrm{N_i}$, then the state $\mathrm{N_i}$ may be pruned from the LIVE list. Until LIVE list is empty, four operators are sequentially applied to each state. A state is chosen from LIVE list in each iteration of the BnB approach using one of the methods. To probe the sub-problems, B rules are carried out on a chosen state. BO operator computes $LB(N_i)$ and $UB(N_i)$ for state $\mathrm{N_i}$. The new state $\mathrm{N_i}$ is discarded if the P_r operator determines that it has an inferior solution.

BnB technique relies upon the accuracy of bounding functions in assessing the promise of each state. Tighter bounds facilitate efficient exploration of the search space rather than exhaustive enumeration of m^n permutations, m and n are number of processors and tasks respectively, which results in exponential time. By narrowing the search space, better efficacies of the BnB strategy can be examined in order to get around this bottleneck and boost the performance.

4 Related work

As the workflow scheduling is a grand challenging problem, several BnB schemes are available in the literature focusing on improving the complexity of bounding functions. Kasahara and Narita (1985) proposed Depth First with Implicit Heuristic Search (DF/IHS) algorithm. The authors employed DFS in combination with heuristic namely Critical Path/Most Important Successors First (CP/MISF). This strategy explores the search space based on the priority computed by CP/MISF heuristic. The lower bound is calculated by the ratio of the sum of the processing time of unscheduled tasks to the number of processors, and CP/MISF heuristic is used to determine the upper bound which required $O(n^2)$ for each state. The authors claimed that DF/IHS algorithm generated near-optimal schedules. The main shortcoming of the algorithm are that simple workflows with unit/uniform tasks processing times are considered and the dataflow times among the tasks are ignored which account for major bottlenecks thus limit the performance of large-scale workflows execution. While CPTF algorithm considers heterogeneity in the processing times and the dataflow times while scheduling the tasks.

Kumar Jain and Rajaraman 1994) observed in their study that upper bound plays pivotal role in estimating the schedule length of the workflow and hence introduced upper bounds which aimed at refining lower bounds founded by Fernandez and Bussell (1973). However, workflows with unit/uniform tasks were considered and dataflow times were ignored while computing lower and upper bounds. In contrast, CPTF algorithm takes into account the wide range of processing times of the tasks besides dataflow times among the tasks.

The parameterized BnB approach proposed by Jonson and Shin (1997) included another constraint on the interdependent tasks i.e., deadlines. The authors realized the significance of critical path for workflow scheduling and used this for computing lower bounds. For computing upper bounds, authors considered deadlines of tasks and employed Earliest Deadline First algorithm (Jonsson and Shin 1997). The search space was explored in the DFS method. The authors findings were crucial and reported that critical path based lower bounds were tighter and had impact on BnB technique.

Fujita (2011) coupled DFS strategy and Highest Level First Estimated Time (HLFET) (Adam et al. 1974) heuristic to determine sequence of exploring the states. Author has focused on improvising lower bounds devised by Fernandez and Bussell (1973). However, lower bounds resulted in quadratic time complexity. Upper bound was derived employing HLFET heuristic which required $O(n^2)$ time.

The authors claimed that even though the bounds required high time but improved the solutions. The primary flaw in this technique is that *LB* functions take O(n^2) time for each iteration whereas *UB* functions take O(n^2) time. For experimentations, two sets of workflows were considered, one ignored dataflows while the other considered uniform dataflow times among the tasks. However, both cases had impact on the performance. Moreover, DFS technique was used, which is a blind search strategy that doesn't look into the search space based on each state's promise. This approach is not practical because the bounds have a high computational complexity. In contrast, CPTF algorithm takes O($n+e$) time for computing lower and upper bound for each state and explores search space in the Best First Search fashion that chooses the best state in the search space. As a result, converges swiftly on the solution without much expanding the search space.

Sirisha (2023) studied the significance of bounds for gauging the promise of a state in the search tree. The author proposed Critical Path/Earliest Finish Time (CP/EFT) algorithm which is a heuristic based BnB approach. The remaining Critical Path length of the tasks or otherwise Earliest Finish Time of the tasks was considered for computing the bounds. The computation of the bounds required O($n+e$) and O(($n+e$) $m+n$ log n), where n, e are the number of tasks and edges in the workflow and m is the number of processors. The authors concluded that the bounds were effective in identifying the promise of each state. In contrast, CPTF algorithm computes upper and lower bounds for each state in O($n+e$) time.

The study on the literature reveals that the most of the available BnB strategies are devised considering simple workflows which ignore dataflows. Moreover, the complexity of evaluating each state in the search space is computationally expensive. Therefore, the available BnB techniques are not pertinent to the workflows with heavy processing times and dataflows. Hence, a new BnB based CPTF algorithm is proposed in the current work. The proposed algorithm considers dataflows while scheduling. Also, the devised bounds are of less complexity.

5 The proposed CPTF algorithm

The CPTF algorithm aims at minimzing turnaround time by curtail the size of the search space. The bounds for each state are calculated while the search space is being explored. The schedule is examined at all potential lengths, and pruning is used to eliminate the inferior states. Initially, the state having lowest lower bound (LB) is chosen for expansion. Greedy heuristics are devised for computing the bounds which abet in narrow downing the search space to smaller portion (Kwok and Ahmad 2005).

The CPTF algorithm is described using four operators. The S operator is first used to choose a state N_i from LIVE list in order to explore the search space. CPTF algorithm employs Best First Search (BeFS) technique that is certain to explore the globally best states (Zhang and Korf 1991) i.e., state with least *LB*. When more than one state in the LIVE list possesses least *LB* then tie-breaking is resolved using upper bound (*UB*) i.e., a state with least *UB* is selected. And if more than one state possesses least *LB* and *UB*, then the issue is resolved randomly. The best state chosen from LIVE list is labeled as best state (BS). Successively, branching operator *B* is used to branch the BS and produce its child nodes.

The LIVE list is updated with each new state N_s. A goal state is a leaf state where n tasks in the workflow are assigned to the processors. The promise of each state is then determined by computing bounds for every new state N_s. For each state N_s, bounding function computes $LB(N_s)$ and $UB(N_s)$ which signify best-case and worst-case makespan of the workflow from the state N_s respectively. In idealistic situations, LB value for each state is the best feasible solution.

The proposed CPTF algorithm adopts critical path (CP)-based heuristic to compute the bounding functions. The search for a solution can be successfully directed by the heuristic estimate. The $LB(N_s)$ and $UB(N_s)$ for each state N_s are computed as follows.

Theorem 1 *The lower bound on the estimate of the makespan at the state N_s in the state space tree is given by*

$$LB\left(N_s\right) = \sum_{t_i \in T_u and t_i \in CP,\ 1 \le k \le m} \left(\min\left\{w_{i,k}\right\}\right) \tag{6}$$

$w_{i,k}$ is processing time of task t_i on processor k, k is a set of m processors, T_u is set of unscheduled tasks of workflow and t_i belongs to the set of unscheduled tasks on CP, i.e., actual makespan of remaining tasks is at least the sum of the processing times of unscheduled tasks on CP on best processor for tasks i.e., CP_{min}.

Any state N_s in the state space for finding the minimal length makespan of a given workflow $W = <T, E>$ is given by $N_s = <T_p, T_u, F(N_s)>$. T_p is a set of tasks $<t_1, t_2, \dots t_h>$ in the partial schedule which are already scheduled and T_u is a set of unscheduled tasks of the workflow. $G(N_s)$ is the actual length of the schedule of T_p which is given by $max\{AFT(t_h)\}$, t_h belongs to T_p. $LB(H(N_s))$ is the estimate on the makespan *at N_s (minimum length of schedule of remaining tasks, T_u). $LB(F(N_s)) = G(N_s) + LB(H(N_s))$.*

Proof CP is the length of the longest path in the workflow stretching from the start task and ending at the sink task that includes at most one task at each level. The sequential bottlenecks between the CP tasks compel these tasks to be executed in a sequence. The sum of the best processing time of the tasks on the CP is at least the completion time of any other path in the workflow.

In situations when the tasks at level l are chosen for executing several tasks in parallel, but does not include CP task at that level, the remaining schedule length will still be the sum of the minimum processing time of the unscheduled CP tasks. The minimum amount of time still remaining from any task until the completion of the workflow is the best processing time of the CP tasks that are yet to be executed. Hence, for any number of tasks that are explored at this level, this will remain the minimum time still required to complete the workflow. Moreover, the execution of the non-CP tasks in the next levels will further delay the tasks on the CP stretching its length. Thus, the sum of the partial schedule length and best processing time of the unscheduled tasks on CP shall remain the lower bound on the schedule length on any path. The lower bound for a state is updated accordingly.

Theorem 2 *The upper bound on the estimate of the makespan at state N_s in the state space tree is given by*

$$UB\left(N_s\right) = \sum_{t_i, t_j \in T_u and t_i, t_j \in CP} \overline{w}_i + d_{i,j} \tag{7}$$

$\overline{w}_i$ is average processing time of task t_i and $d_{i,j}$ is dataflow time between task t_i and its successor t_j, T_u is a set of unscheduled tasks of the workflow, and t_i and t_j belong to a set of unscheduled tasks on the CP i.e., actual schedule length of remaining tasks is at most the summation of average processing times of unscheduled tasks on CP and dataflow times between the tasks on the CP.

Any state N_s in the state space for finding the maximal length makespan of a given workflow $W = < T, E >$ is given by $N_s = < T_p, T_u, F(N_s) >$. T_p is a partial schedule which includes a set of tasks $<t_1, t_2, \ldots t_h>$ already scheduled and T_u is a set of unscheduled tasks of the workflow. $G(N_s)$ is the actual length of the schedule of T_p which is given by $max\{AFT(t_h)\}$, t_h belongs to T_p. $UB(H(N_s))$ is the estimate on the makespan at N_s (maximum length of schedule of remaining tasks, T_u). $UB(F(N_s)) = G(N_s) + UB(H(N_s))$.

Proof CP is the longest path in the workflow and its length is computed by the sum of the average processing time of the tasks on CP and average dataflow times between the tasks on CP. Being the longest path, the constrained sequential execution time of the tasks on CP is at most the length of any other path of the worflow. Hence, the length of CP reflects the maximal time required to complete the execution of the workflow. Given a state where $\{t_1, \ldots, t_h\} \in T_p$ i.e., the tasks are already scheduled, the choice of the next state shall be based on remaining length of the CP. Among the tasks ready to execute, when all tasks are ready for execution are non-CP tasks then optimal choice would be the task with least remaining CP length. Any optimal schedule of the workflow cannot exceed the schedule length thus generated. Given a state N_s where a partial schedule $\{t_1,\ldots,t_h\}$ tasks are already scheduled. The length of the makespan is given by

$$G\left(N_s\right) + Schedule\ length\ of\ T_u \tag{8}$$

By calculating the maximum schedule length of T_u, an upper bound can be found. It is observed that the length of any schedule involving T_u cannot exceed the remaining CP length. The duration of the makespan is constrained because the CP tasks include one task at each level. In other words, any ready task that is delayed on the CP will at a minimum result in corresponding increase in the makespan. Therefore, a greedy strategy for allocating processors to a list of ready tasks must prioritize CP tasks first, followed by tasks with shortest remaining CP length. This greedy method yields a schedule that has a makespan that can never be surpassed by an optimal schedule for the remaining tasks. Hence the length of the remaining CP imposes an upper bound on the schedule. Thus, the upper bound on state N_s is the sum of the actual schedule length of T_p i.e., $G(N_s)$ and the length of the remaining CP from the tasks T_u i.e., $UB(H(N_s))$. The makespan of workflow from any state N_s always lies between $LB(N_s)$ and $UB(N_s)$.

Lastly, *Pr* operator compares *LB* computed for every new state N_s with UB_{best} and prunes the state if $LB(N_s) > UB_{best}$. Tight bounds aid in pruning considerable search space leading to lessening the search space size.

5.1 Illustration of CPTF algorithm

The search space generated by CPTF algorithm for the workflow given in Fig. 1 is illustrated in Fig. 2. The search space is generated from the initial state N_1 depicting empty schedule { }. Initially, state N_1 is placed in LIVE list. $UB(N_1)$ is computed using Eq. 9 for state N_1 is 25.5. A global variable UB_{best} records least UB amongst all the expanded states. UB_{best} is initialized to $UB(N_1)$ i.e., 25.5. Currently, task t_1

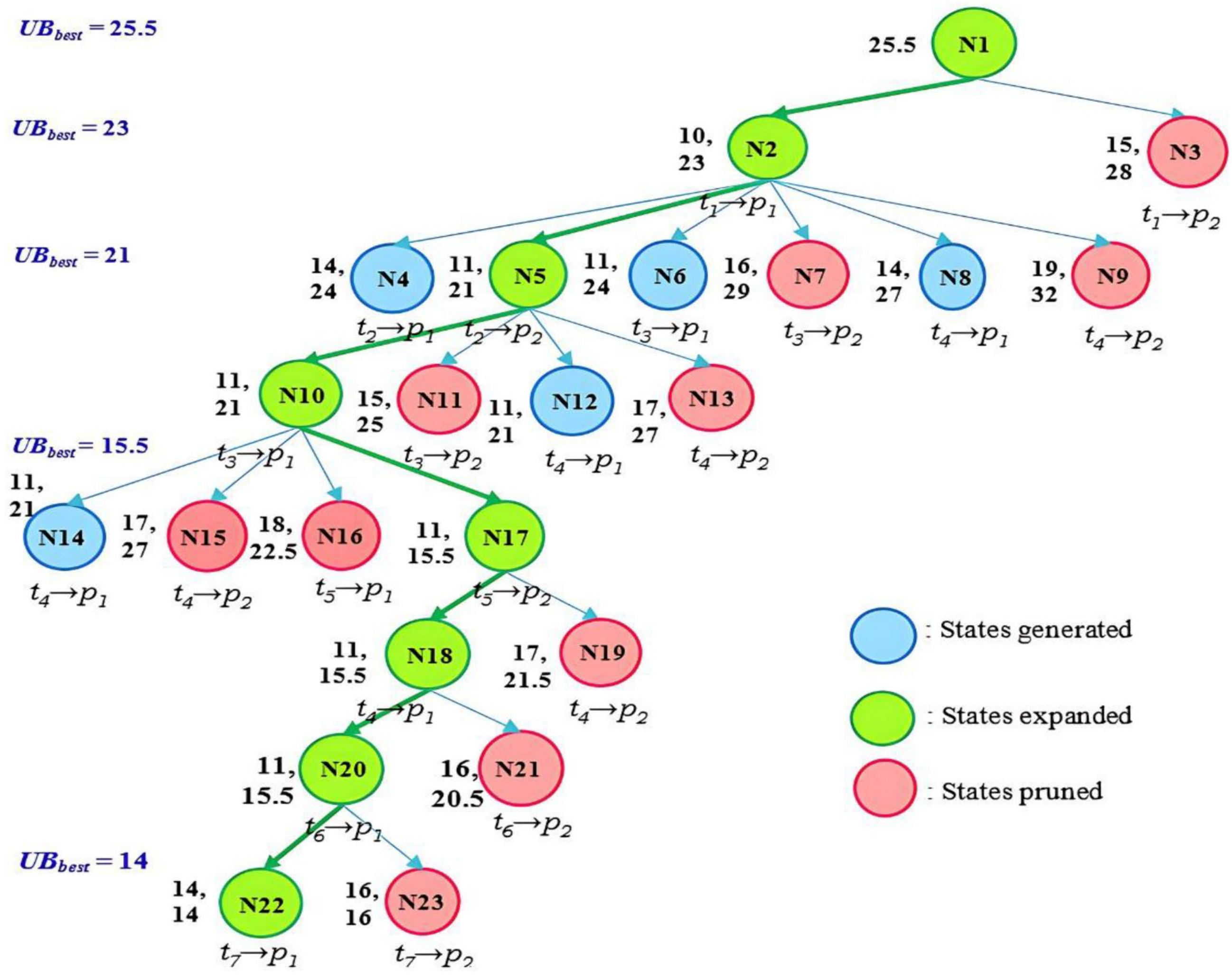

Fig. 2 Enumeration of search space by CPTF algorithm for the workflow given in Fig. 1

in the workflow is ready. In the search space, two new states N_2 and N_3 are generated from the state N_1 by assigning task t_1 to p_1 and task t_1 to p_2 processors respectively. The state N_1 is put in the CLOSE list as it is explored. States N_2 and N_3 are added to LIVE list and their bounds are computed. The $(LB(N_2), UB(N_2))$ are (10, 23) and $(LB(N_3), UB(N_3))$ are (15, 28).The UB_{best} is updated to 23. The CPTF algorithm chooses N_2 with minimum LB from the LIVE list. The state N_2 indicates the mapping of task t_1 to processor p_1. Once task t_1 is scheduled, tasks t_2, t_3, and t_4 become ready and each task is mapped to p_1 and p_2 processors which leads to the generation of six states viz., N_4, N_5, N_6, N_7, N_8, and N_9 from the state N_2. The (LB, UB) computed for N_4, N_5, N_6, N_7, N_8, and N_9 states are (14, 24), (11, 21), (11, 24), (16, 29), (14, 27), and (19, 32) respectively. The UB_{best} is updated to 21. The state N_5 with least LB is selected. From the state N_5, four new states N_{10}, N_{11}, N_{12}, and N_{13} are generated by mapping the currently ready tasks t_3 and t_4 to p_1 and p_2 processors. The CPTF algorithm progresses in this approach to expand the search space. At each stage, algorithm picks the lowest LB state, branches it and adds new states to LIVE list. Subsequently computes the bounds for the new states. The UB of new states is compared with UB_{best} if found that UB of the new state is less than UB_{best} then it updates UB_{best} with UB of the new state and prunes all the states having LB greater than UB_{best}. The CPTF algorithm terminates at the state N_{22}, as all tasks are mapped to processors and *UB* of N_{22} state is the optimal solution. The search space is explored in the sequence N_1, N_2, N_5, N_{10}, N_{17}, N_{18}, N_{20}, and N_{22}. The states N_3, N_7, N_9, N_{11}, N_{13}, N_{15}, N_{16}, N_{19}, N_{21}, and N_{23} are pruned from the search space. In Fig. 2, $t_i \rightarrow p_k$ at each state indicates the assignment of task t_i to p_k processor, $1 \le i \le n$, $1 \le k \le m$. The CPTF algorithm is presented in Algorithm 1.

Algorithm 1 CPTF algorithm (T, E, P, D)

Input: A workflow W=<T,E,P,D>, T and E are a set of tasks and edges. $w_{i,k}$ is the processing time of task $t_i \in$ T on processor $p_k \in$ P, $1 \le i \le n$, $1 \le k \le m$. $LB(N_i)$ and $UB(N_i)$ are lower and upper bounds on a state N_i computed using equations 8 and 9 respectively. UB_{best} is the minimum *UB* among all states generated.

Output: Schedule of $t_i \in$ T tasks assigned to $p_k \in$ P processors.

```
1.  Initialize LIVE list with the initial state $N_1$.
2.  Compute $UB(N_1)$ and assign $UB_{best}$ to $UB(N_1)$ value.
3.  while (LIVE $\neq \emptyset$)
4.      Select a state with least $LB$ from LIVE list, label it
        as Best State (BS).
5.      Branch BS to create new states $N_s$.
6.      for each new state $N_{si} \in N_s$ do
7.      Check if $N_{si}$ is included in either LIVE or CLOSE
        list.
8.      if true, eliminate new state; otherwise, place in
        LIVE list.
9.          Compute $LB(N_{si})$ and $UB(N_{si})$
10.             if ($UB(N_{si}) < UB_{best}$ then
11.                 $UB_{best} := UB(N_{si})$
12.             if ($LB(N_{si}) > UB_{best}$ ) then
13.                 Prune $N_{si}$
14.     end for
15. end while
```

5.2 Time complexity analysis of CPTF algorithm

Time complexity of CPTF algorithm is majorly based on the time required for computing *LB* and *UB* for each state in the search space. *LB* and *UB* for each state requires $O(n + e)$ time for examining *n* tasks and *e* edges on the CP.

6 Results and discussions

The CPTF algorithm is evaluated by simulation experiments conducted using a benchmark of random and scientific workflows. The results of CPTF algorithm is studied against best available heuristic scheduling algorithms, viz., critical path/earliest finish time first (CP/EFT) algorithm (Sirisha 2023), global highest_degree task first (GHTF) algorithm (Sirisha and prasad 2022), improved predict priority task scheduling (IPPTS) algorithm (Djigal et al. 2021), heterogeneous earliest finish time (HEFT) (Topcuoglu et al. 2002), performance effective task scheduling (PETS) (Illavarasan and Thambidurai 2007) and critical path on processor (CPOP) (Topcuoglu et al. 2002) algorithms. Experimentations are implemented on Windows 10 Computer with configuration Intel(R) Core i7-10870H processor, 2.20 GHz, 8 GB RAM in Java programming.

The workflows are generated using the parameters viz.

- Workflow size (n) signifies the number of tasks in a workflow.
- Dataflow to processing time ratio (DPR) is the ratio of average dataflow time to average processing time of the workflow. Computation-intensive workflows can be produced for DPR ≤ 1, where computations dominate dataflows in the workflows. Workflows with heavily data dependent tasks are produced for DPR > 1.
- Shape parameter (α) defines the shape of the workflow, i.e., the number of levels l and tasks k at each level. The formulas $\sqrt{n}/\alpha$ and $\sqrt{n} \times \alpha$ are used to calculate the values of l and k, respectively. When $\alpha < 1$, longer workflows with less parallelism are generated while for $\alpha \ge 1$, shorter and broader workflows with higher parallelism are produced.
- Heterogeneity factor (β) refers to the variation in the processing times of the tasks in a workflow. High β values signify substantial deviation in the processing times of tasks whilst low values show minor variation. The processing time of task t_i on processor p_k, represented as $w_{i,k}$ is randomly selected from the following range:

$$\overline{w} \times (1 - \beta/2) \le w_{i,k} \le \overline{w} \times (1 + \beta/2) \tag{9}$$

where $\overline{w}$ is the average processing time of a workflow.

The following metrics are used to analyze the performance of the scheduling algorithms [1], [3], [5].

i. Normalized makespan. Makespan is a key metric used to assess a scheduling approach. It is crucial to normalize the makespan to its lower bound, which is described as the Normalized Makespan (NM) because a significant number of workflows with various characteristics are generated. NM is computed as follows.

$$\text{NM} = \frac{\text{Makespan}}{\text{CPmin}} \tag{10}$$

The denominator CP_{min} is calculated as given in definition 2. NM of a workflow is always greater than 1 since CP_{min} is considered as the lower bound of the makespan. The scheduling algorithm that produces the least NM is considered to be more efficient.

ii. Speedup is defined as the ratio of the sequential processing time (sequential makespan) of the workflow to the parallel processing time on m processors (makespan) and is calculated as following.

$$\text{Speedup} = \frac{\text{Sequential makespan}}{\text{Makespan}} \tag{11}$$

The numerator, sequential makespan is calculated by assigning all the tasks to a single processor that minimizes the overall processing time. Typically, it is better to use a scheduling strategy that results in a higher speedup.

iii. Efficiency is characterized as the ratio of speedup to the m processors used to execute a workflow. Efficiency typically varies from 0 to 1. A scheduling strategy is considered more efficient if its efficiency is closer to 1. The scheduling algorithm's efficiency is calculated using the formula below.

$$\text{Efficiency} = \frac{\text{Speedup}}{m} \tag{12}$$

iv. Running time of the algorithm is the amount of time taken by the algorithm to generate an output schedule for the workflow.

6.1 Performance analysis on random workflows

The random workflows for experimentations are generated according to the parameters detailed in Topcuoglu et al. (2002), Illavarasan and Thambidurai (2007). The wide-ranging parameter values used for experimentations are given in Table 3. The combinations of these characteristics result in a collection of 42,240 workflows with various topologies. The number of processors (m) considered are 4, 8, 16, and 32.

In the first experiment, the impact of workflow size (n) on average NM and speedup of the scheduling algorithms is analysed, corresponding graphs are depicted in Fig. 3a and b respectively. The workflow sizes range from 40 to 500, increasing by 10 till 100 and then by 100 between each. The graph's points are all plotted using average data from 3840 experiments. The graphs show an upward trend in the data. It is clear from Fig. 3a that, as n increases from 40 to 80, average NM rises gradually, and as n increases from 80 to 100, the increase is stable. For workflow sizes greater than 100, a linear increase in average NM is noticed. Trivial variations in average NM for workflow sizes below 100 is noticed. Moreover, substantial deviation in average NM is observed for workflow sizes greater than 100. The average NM of CPTF algorithm is better than CP/EFT, GHTF, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by 3.95, 12.57, 15.04, 21.27, 27.92, and 33.5 respectively.

The average speedup for varied workflow sizes is shown in Fig. 3b. The graph's data trend seems to be growing as the size of the workflow. The graph shows that CPTF algorithm's average speedup was better than CP/EFT, GHTF, IPPTS, HEFT, PETS, and CPOP algorithms and noted as 2.43, 5.05, 7.57, 8.89, 10.27, and 13.7 respectively.

The following experiment examines the dataflow to processing time ratio (DPR) and shape parameter (α). The average NM is displayed on the graph in Fig. 3c in relation to DPR values ranging from 0.1 to 10. The graph's points are drawn from 5280 experiments. For DPR ≤ 1, performance of all the scheduling algorithms differed marginally. However, when DPR > 1, a considerable variation in the average NM is evident. Higher DPR values i.e., > 1 signify that dataflow overheads dominate task processing times, which increased the NM of scheduling algorithms. When DPR ≤ 1, CPTF algorithm outperformed CP/EFT, GHTF, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by 1.96, 5.19, 11.33, 13.91, 16.26, and 29.46 respectively. For data intensive workflows having DPR > 1, GHTF algorithm is better than CPTF, CP/EFT, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by 6.79, 8.96, 7.54, 11.28, 16.5, and 27.51 respectively. In this regard, CPTF algorithm is observed to be performing next to GHTF algorithm. Overall, GHTF algorithms' performance improved by 3.03, 5.10, 6.99, 10.32, 14.58, and 26.08 respectively against CPTF, CP/EFT, IPPTS, HEFT, PETS, and CPOP algorithms. This

Table 3 Characteristics of random workflows

Parameter	Values									
n	40	50	60	70	80	90	100	200	300	400	500
DPR	0.1	0.5	0.75	1.0	2.0	5.0	7.5	10.0			
α	0.25	0.5	0.75	1.0	1.25	1.5					
β	0.1	0.25	0.5	0.75	1.0						
$\overline{w}$	40	60	80	100							

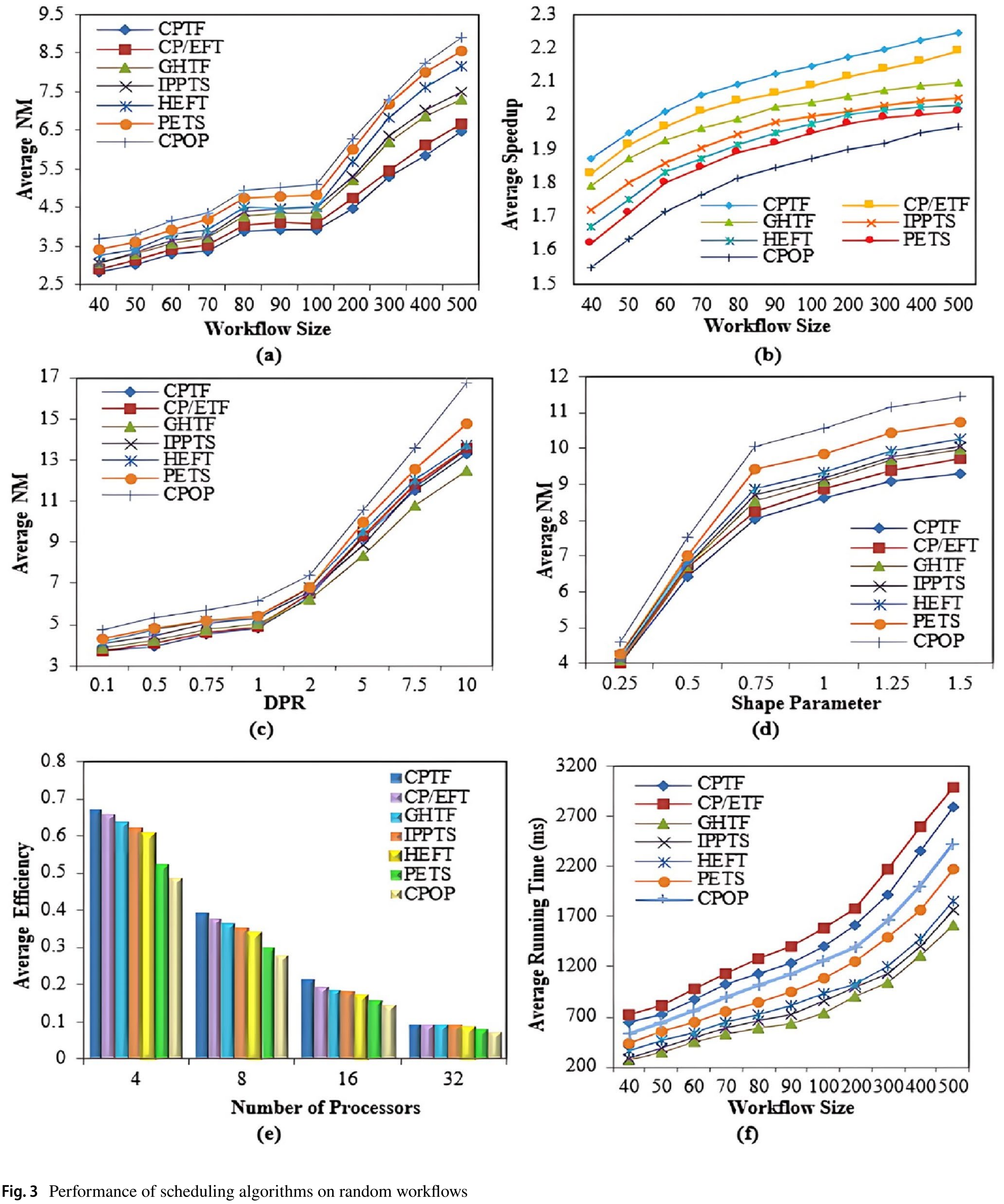

Fig. 3 Performance of scheduling algorithms on random workflows

experimentation reveals that CPTF algorithm has effectively shortened the makespan for data intensive workflows.

The graph's points are drawn from 5280 experiments. For DPR ≤ 1, performance of all the scheduling algorithms differed marginally. However, when DPR > 1, a considerable variation in the average NM is evident. Higher DPR values i.e., > 1 signify that dataflow overheads dominate task processing times, which increased the NM of scheduling algorithms. When DPR ≤ 1, CPTF algorithm outperformed CP/EFT, GHTF, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by 1.96, 5.19, 11.33, 13.91, 16.26, and 29.46 respectively. For data intensive workflows having DPR > 1, GHTF algorithm is better than CPTF, CP/EFT, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by 6.79, 8.96, 7.54, 11.28, 16.5, and 27.51 respectively. In this regard, CPTF algorithm is observed to be performing next to GHTF algorithm. Overall, GHTF algorithms' performance improved by 3.03, 5.10, 6.99, 10.32, 14.58, and 26.08 respectively against CPTF, CP/EFT, IPPTS, HEFT, PETS, and CPOP algorithms. This experimentation reveals that CPTF algorithm has effectively shortened the makespan for data intensive workflows.

The shape parameter (α) is a vital metric that manifests the proficiency of a scheduling algorithm in exploiting the parallelism of a workflow. The graph shown in Fig. 3d shows the influence of shape parameter on average NM. The range of α values taken into account for experiments is 0.25 to 1.5, with 0.25 increments between each value. The 7040 experiments are used to get each data point on the graph. According to the graph, the average NM climbed linearly up to α value of 0.75, after which a constant increase was seen. Additionally, it is clear from the graph that, until α value 0.75, there is only a minor difference in the average NM, which steadily grew as α values climbed. The performance improvement of CPTF algorithm against CP/EFT, GHTF, IPPTS, HEFT, PETS, and CPOP algorithms is noted as (in percent) by 2.54, 5.06, 6.26, 8.2, 13.06, and 20.98 respectively. Exploration of better workflow parallelization is what has led to an improvement in the CPTF algorithm's performance for shape parameters.

The next experiment examines the scheduling algorithms' average efficiency with respect to number of processors (m), depicted in the graph in Fig. 3e. With each successive rise in power of 2, number of processors increased from 4 to 32. Data from 10,560 experiments were used to plot each point on the graph. The graph reveals that average efficiency of scheduling algorithms declined with increase in processors.

From experimental results, average efficiency of CPTF algorithm has improved compared with CP/EFT, GHTF, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by 3.61, 6.56, 9.14, 12.49, 22.725, 28.13 respectively. It is concluded that efficiency of scheduling algorithms declined as processors increased.

The following experiment compares the CPTF algorithm's average running time against various heuristic scheduling approaches for varying workflow sizes, presented in Fig. 3f. It is observed that CPTF algorithm is 23.41 percent faster running time than the CP/EFT algorithm. However, GHTF algorithm has surpassed all other algorithms 86.25, 106.44, 10.18, 18.78, 41.33, 62.20 percent against CPTF, CP/EFT, IPPTS, HEFT, PETS, and CPOP algorithms respectively.

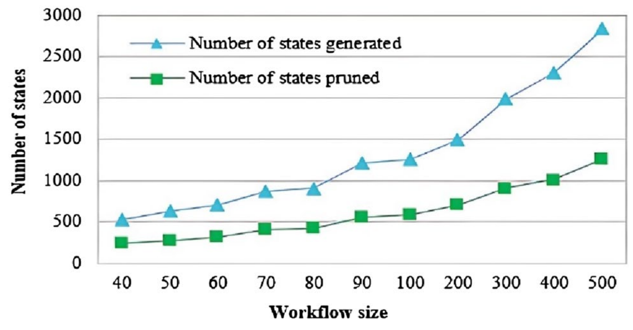

Fig. 4 Number of states generated and pruned by CPTF algorithm with respect to workflow sizes for random workflows

The following experiment demonstrates the effectiveness of bounding functions devised, shown in Fig. 4. It can be observed from Fig. 4 that the number of states generated and pruned have increased as the workflow size increased. The CPTF algorithm could prune 45.38 percent of the generated states.

6.2 Performance analysis on scientific workflows

Scientific applications comprise of a sequence of stages involving enormous complex computations having dependencies. Scientific applications structures are characterized as scientific workflows. For experimentation, four different real-world scientific workflows well illustrated by Juve et al. (2013) are published by Pegasus project are used. Two workflows are compute intensive workflows viz., LIGO (Abramovici et al. 1992) and Epigenomics (USC Epigenome Center 2019), and other two workflows viz., Cybershake (Graves et al. 2011), and Montage (Berriman et al. 2006) are data intensive.

LIGO is an application in the field of physics used for detecting gravitational waves. Epigenomics workflows are used for automation of numerous operations in genome sequence processing. Cybershake application identifies Earthquake Rupture Forecast (ERF) within 200 km of area of interest. Montage is an astronomical application used as image mosaic engine. The structure of the scientific workflows are depicted in Fig. 5.

The parameters required for generating scientific workflows randomly are DPR, heterogeneity factor (β), average processing time ($\overline{w}$). The shape parameter (α) is not required as the structure of the scientific workflows is known. Experiments are conducted using 4, 8, 12, 16, and 32 number of processors (m). LIGO and Epigenomics workflows are computation intensive generated with the parameters given in Table 4. While Cybershake and

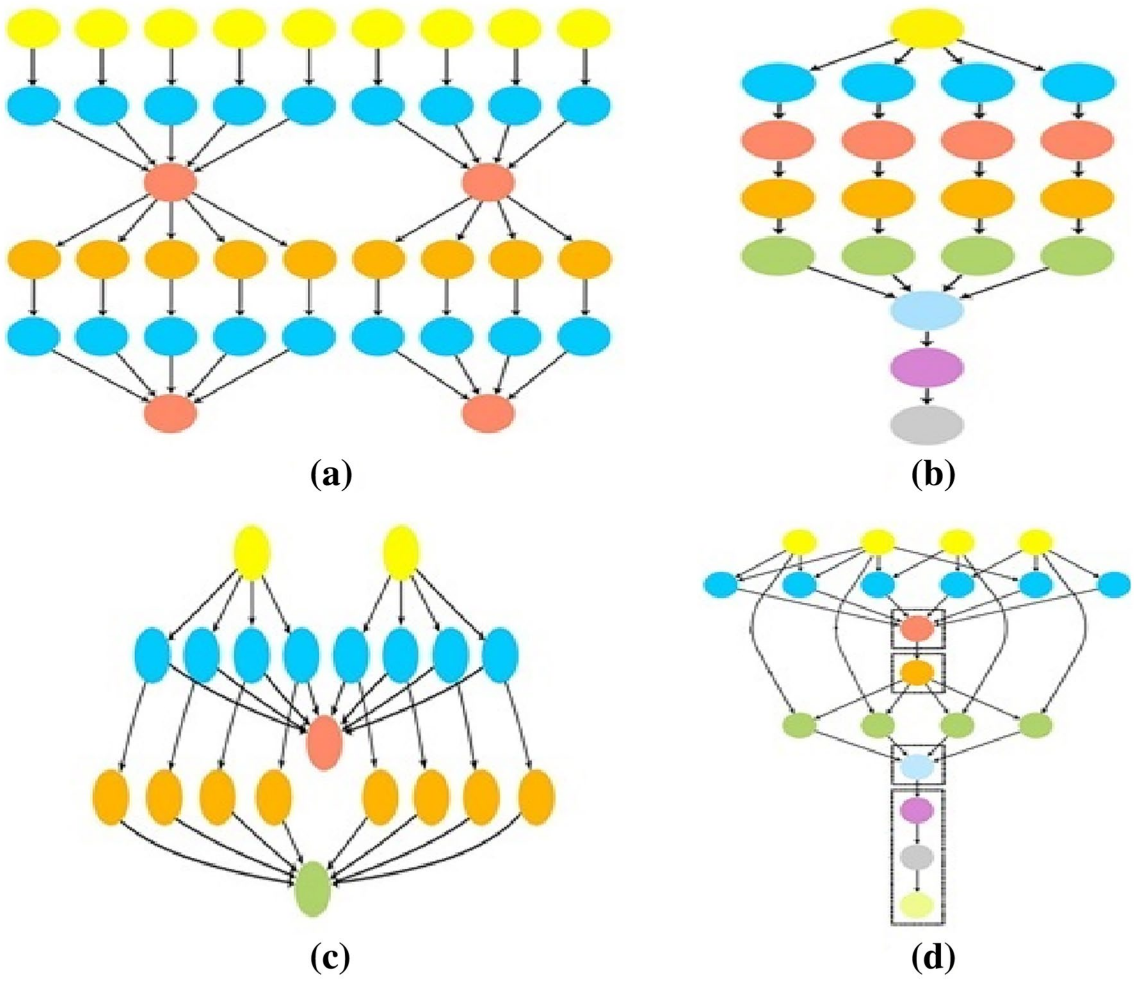

Fig. 5 Scientific workflows, **a** LIGO **b** epigenomics **c** cybershake **d** montage. Courtesy Juve et al. (2013), https://pegasus.isi.edu/workflow_gallery/

Table 4 LIGO and epigenomics workflow characteristic

Parameter	Values				
DPR	0.1	0.25	0.5	0.75	1.0
β	0.1	0.25	0.5	0.75	1
$\overline{w}$	100	150	200	250	

Table 5 Cybershake and montage workflows characteristic

Parameter	Values				
DPR	1	2	5	7	10
β	0.1	0.25	0.5	0.75	1
$\overline{w}$	20	30	40	50	

Montage are data intensive workflows generated with parameters mentioned in Table 5.

Figure 6 displays the effectiveness of scheduling strategies for scientific workflows. Based on the findings of the experiments, for LIGO workflows CPTF algorithm surpassed CP/EFT, GHTF, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by (2.79, 5.76, 10.05, 13.07, 18.41, 22.39), shown in Fig. 6a, for Epigenomics workflows by (4.43, 8.37, 11.79, 15.68, 22.2, 27.56) depicted in Fig. 6b, Cybershake workflows by (3.2, 6.84, 9, 13.4, 19.68, 22.75) presented in Fig. 6c. However for Montage workflows that are highly data intensive, GHTF algorithm outperformed CPTF, CP/EFT, IPPTS, HEFT, PETS, and CPOP algorithms (in percent) by (2.85, 5.64, 8.46, 11.71, 15.88, 28.12) respectively, shown in Fig. 6d.

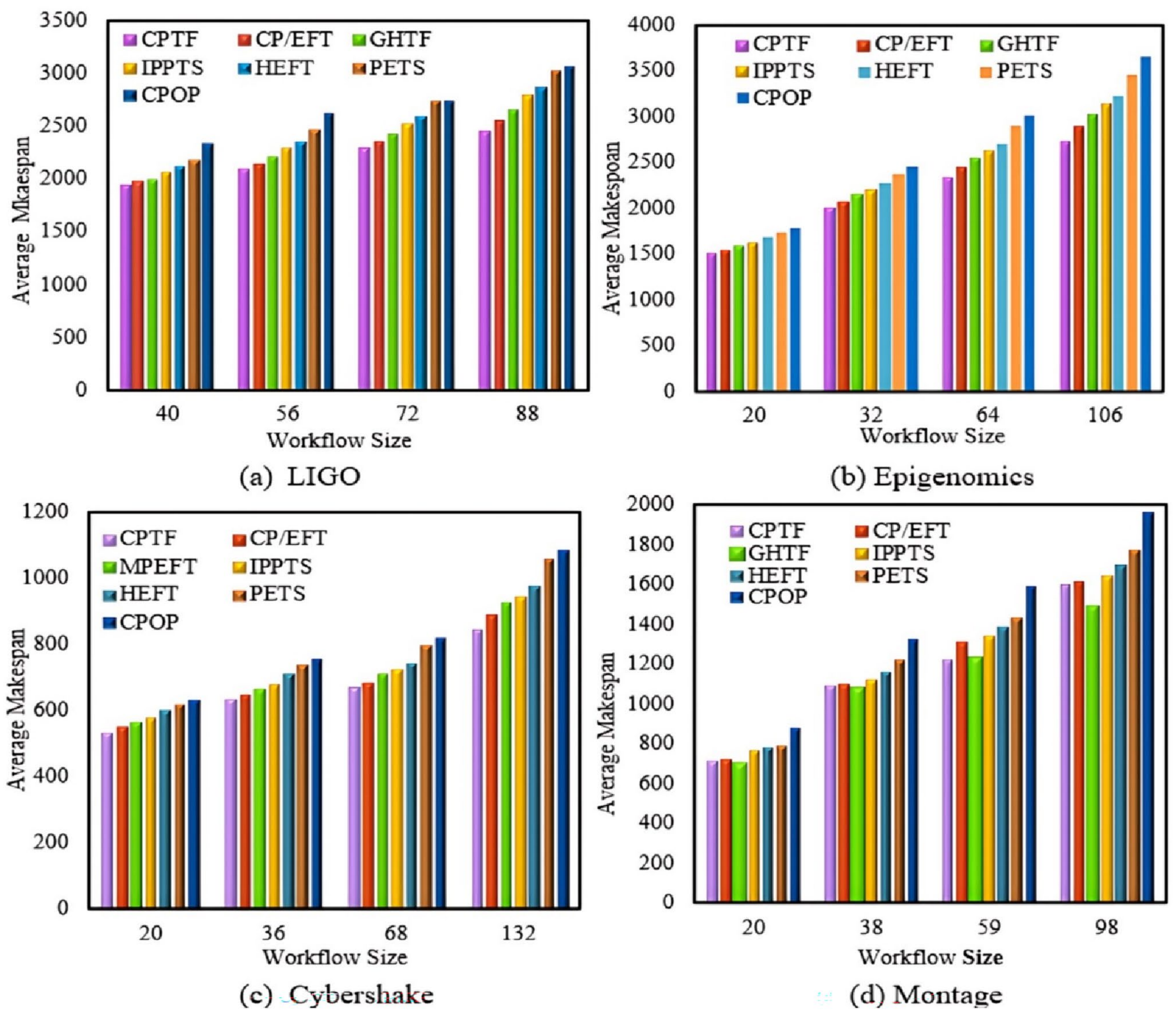

Fig. 6 Performance of scheduling algorithms on scientific workflows

7 Conclusions

In the current work, the CPTF algorithm, a novel heuristic-based BnB technique for scheduling workflows is proposed to produce optimal schedules. CPTF algorithm primarily emphasizes on reducing the turnaround time by sinking search space size. Keeping this aim in view, CPTF algorithm works on devising tight bounds to prune the sub-trees with inferior solutions. The search process is steered by the heuristics and solutions are shrinked to smaller search space size. Thus, limiting the solution searching resulted in generating reduced search space.

Therefore, less states are explored thus quickly converges to the solution. The bounds formulated for estimating the goodness of each state are tight, less complicated, and less complex. The computation of LB and UB for each state required $O(n+e)$ time, where n and e are number of tasks and number of edges in workflow.

The efficiency of the CPTF algorithm has been demonstrated through experiments using random and scientific workflows. The experimental findings lead to the conclusion that the devised bounds are effective and CPTF algorithm pruned 45.38 percent of the generated states. Moreover, CPTF algorithm showed improvement in average NM for workflow sizes larger than 100 tasks by 8.5 to 19 percent and produced best schedules for 89.36 percent of the cases. The results of the scientific workflows reveal that CPTF algorithm is also feasible for computation and data-intensive scientific application workflows viz., LIGO, Epigenomics, Cybershake, and Montage, As a future extension to this work, a proposal to include another constraint among the tasks i.e., deadlines can be considered and also to extend the proposed algorithm to dynamic scenarios.

Data availability The data that support the findings of this study are available from the corresponding author, D. Sirisha, upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest.

References

Abramovici, M., Althouse, W.E., Drever, R.W., Gursel, Y., Kawamura, S., Raab, F.J., Shoemaker, D., Sievers, L., Spero, R.E., Thorne, K.S.: LIGO the laser interferometer gravitational-wave observatory. Science **256**(5055), 325–333 (1992)

Adam, T.L., Chandy, K.M., Dickson, J.: A comparison of list scheduling for parallel processing system. Commun. ACM **17**(12), 685–690 (1974). https://doi.org/10.1145/361604.361619

Ahmad, W., Alam, B.: An efficient list scheduling algorithm with task duplication for scientific big data workflow in heterogeneous computing environments. Concurr. Comput. Pract. Exp. (2020). https://doi.org/10.1002/cpe.5987

Berriman, G., Laity, A., Good, J., Jacob, J., Katz, D., Deelman, E., Singh, G., Su, M., Prince, T.: Montage: the architecture and scientific applications of a national virtual observatory service for computing astronomical image mosaics. In: Proceedings of Earth Sciences Technology Conference (2006)

Djigal, H., Feng, J., Lu, J., Ge, J.: IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. **32**(05), 1057–1071 (2021). https://doi.org/10.1109/TPDS.2020.3041829

Fernandez, E.B., Bussell, B.: Bounds on the number of processors and time for multiprocessor optimal schedules. IEEE Trans. Comput. **22**(8), 745–751 (1973). https://doi.org/10.1109/TC.1973.5009153

Fujita, S.: A branch-and-bound algorithm for solving the multiprocessor scheduling problem with improved lower bounding techniques. IEEE Trans. Comput. **60**(7), 1006–1016 (2011). https://doi.org/10.1109/TC.2010.120

Gary, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co. San Francisco, CA (1979)

Graves, R., Jordan, T.H., Callaghan, S., Deelman, E., Field, E., Juve, G., Kesselman, C., Maechling, P., Mehta, G., Milner, K.: Cybershake: a physics-based seismic hazard model for Southern California. Pure Appl. Geophys. **168**(3–4), 367–381 (2011)

Illavarasan, E., Thambidurai, P.: Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. J. Comput. Sci. **3**(2), 94–103 (2007). https://doi.org/10.1109/71.993206

Jonsson, J., Shin, K.G.: A parameterized branch-and-bound strategy for scheduling precedence-constrained tasks on a multiprocessor system. In: Proceedings of the 1997 International Conference on Parallel Processing, Bloomington, IL, 158–165 (1997). https://doi.org/10.1109/ICPP.1997.622580

Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Gener. Comput. Syst. **29**(3), 682–692 (2013)

Kasahara, H., Narita, S.: Practical multiprocessor scheduling algorithms for efficient parallel processing. IEEE Trans. Comput. **33**(11), 1023–1029 (1985). https://doi.org/10.1109/TC.1984.1676376

Kelefouras, V., Djemame, K.: Workflow simulation and multi-threading aware task scheduling for heterogeneous computing. J. Parallel Distrib. Comput. **168**, 17–32 (2022). https://doi.org/10.1016/j.jpdc.2022.05.011

Khojasteh Toussi, G., Naghibzadeh, M., Abrishami, S., et al.: EDQWS: an enhanced divide and conquer algorithm for workflow scheduling in cloud. J. Cloud Comp. (2022). https://doi.org/10.1186/s13677-022-00284-8

Kohler, W.H., Steiglitz, K.: Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. J. ACM **21**(1), 140–156 (1974). https://doi.org/10.1145/321796.321808

Kumar Jain, K., Rajaraman, V.: Lower and upper bounds on time for multiprocessor optimal schedules. IEEE Trans. Parallel Distrib. Syst. **5**(8), 879–886 (1994). https://doi.org/10.1109/71.298216

Kung, H.-L., Yang, S.-J., Huang, K.-C.: An improved Monte Carlo Tree Search approach to workflow scheduling. Connect. Sci. **34**(1), 1221–1251 (2022). https://doi.org/10.1080/09540091.2022.2052265

Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4), 406–471 (1999). https://doi.org/10.1145/344588.344618

Kwok, Y., Ahmad, I.: On multiprocessor task scheduling using efficient state space search approaches. J. Parallel Distrib. Comput. **65**, 1515–1532 (2005). https://doi.org/10.1016/j.jpdc.2005.05.028

Sirisha, D.: Complexity versus quality: a trade-off for scheduling workflows in heterogeneous computing environments. J. Super Comput. **79**, 924–946 (2023). https://doi.org/10.1007/s11227-022-04687-x

Sirisha, D., Prasad, S.S.: MPEFT: a makespan minimizing heuristic scheduling algorithm for workflows in heterogeneous computing systems. CCF Trans. HPC. (2022). https://doi.org/10.1007/s42514-022-00116-w

Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance effective and low complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002). https://doi.org/10.1109/71.993206

USC Epigenome Center. http://epigenome.usc.edu (2019). Accessed 18 Nov 2020

Vempaty, N.R., Kumar, V., Korf, R.E.: Depth first vs best first search. In: Proceedings of the 9th National Conference on AI, AAAI-92, San Jose, CA University, 545–550 (1992)

Zhang, W., Korf, R.E.: An average case analysis of branch and bound with applications: summary of results. In: Proceedings of the 10th National Conference on AI, AAAI-91, CA University; 434–440 (1991)

D. Sirisha is a Senior member IEEE and Professor of CSE Department. Her research interests include task scheduling approaches in heterogeneous environments, high performance computing algorithms, artificial intelligence, and machine learning.

S. Sambhu Prasad is a Professor of Mechanical Department. His interests included job shop scheduling, supply-chain management.