Σχολή Θετικών Επιστημών

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

«ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ»

“Master of Science in Informatics and Telecommunications”

Κατεύθυνση: Ψηφιακές Δεξιότητες

«Ευφυής τοποθέτηση ροών επεξεργασίας»

Κωνσταντέλλος Δημήτριος του Κωνσταντίνου

Οκτώβριος 2024

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Σχολή Θετικών Επιστημών
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ»

"Master of Science in Informatics and Telecommunications"

Κατεύθυνση: Ψηφιακές Δεξιότητες

«Ευφυής τοποθέτηση ροών επεξεργασίας»

Μεταπτυχιακή Διπλωματική Εργασία

που υποβλήθηκε στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Θεσσαλίας
ως μέρος των απαιτήσεων για την απόκτηση
Διπλώματος Μεταπτυχιακών Σπουδών στην Πληροφορική και Τηλεπικοινωνίες
από τον

Κωνσταντέλλο Δημήτριο του Κωνσταντίνου

Δήλωση Αυθεντικότητας, ζητήματα Copyright

«Ο μεταπτυχιακός Μ.Φ. που εκπόνησε την παρούσα διπλωματική εργασία φέρει ολόκληρη την ευθύνη προσδιορισμού της δίκαιης χρήσης του υλικού, η οποία ορίζεται στη βάση των εξής παραγόντων: του σκοπού και χαρακτήρα της χρήσης (μη-εμπορικός, μη-κερδοσκοπικός, αλλά εκπαιδευτικός-ερευνητικός), της φύσης του υλικού που χρησιμοποιεί (τμήμα του κειμένου, πίνακες, σχήματα, εικόνες κ.λπ.), του ποσοστού και της σημαντικότητας του τμήματος που χρησιμοποιεί σε σχέση με το όλο κείμενο υπό Copyright, και των πιθανών συνεπειών της χρήσης αυτής στην αγορά ή την γενικότερη αξία του υπό Copyright κειμένου».

Ο Μεταπτυχιακός Φοιτητής

Κωνσταντέλλος Δημήτριος

NIKOLAOS TZIRITAS
25/12/2024 11:47

Οκτώβριος 2024

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

ΣΕΛΙΔΑ ΤΡΙΜΕΛΟΥΣ ΕΞΕΤΑΣΤΙΚΗΣ ΕΠΙΤΡΟΠΗΣ

Η παρούσα Μ.Δ.Ε. εγκρίθηκε ομόφωνα από την τριμελή εξεταστική επιτροπή η οποία ορίστηκε από τη Συνέλευση του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Θεσσαλίας, σύμφωνα με το νόμο και τον εγκεκριμένο Οδηγό Σπουδών του Π.Μ.Σ. «Πληροφορική και Τηλεπικοινωνίες». Τα μέλη της Επιτροπής ήταν:

- Τζιρίτας Νικόλαος (Επιβλέπων)
- Κολομβάτσος Κωνσταντίνος (Εξεταστής)
- Κωνσταντίνου Ιωάννης (Εξεταστής)

Η έγκριση της μεταπτυχιακής διπλωματικής εργασίας από το Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Θεσσαλίας δεν υποδηλώνει αποδοχή των απόψεων του συγγραφέα.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

ΣΕΛΙΔΑ ΠΕΡΙΛΗΨΕΩΝ / ABSTRACTS PAGE

Ο γενικός επιστημονικός τομέας αυτής της διπλωματικής εργασίας είναι ο προγραμματισμός εργασιών σε ετερογενή περιβάλλοντα. Οι διάφορες εργασίες σχηματίζουν μία εφαρμογή που μοντελοποιείται από έναν κατευθυνόμενο άκυκλο γράφο (ΚΑΓ). Εστιάζουμε στον αλγόριθμο Critical-Path-on-a-Processor (CPOP) - ένα δημοφιλή αλγόριθμο με καλή ποιότητα αποτελεσμάτων στην περίπτωση των ετερογενών επεξεργαστών - και στο στατικό προγραμματισμό εργασιών. Έπειτα από κάποιους απαιτούμενους ορισμούς, αναλύουμε το πρόβλημα και τον τρόπο με τον οποίο ο αλγόριθμος CPOP λειτουργεί. Όσο πειραματιζόμαστε, χρησιμοποιούμε τον αλγόριθμο Heterogeneous Earliest-Finish-Time (HEFT) για σύγκριση, έναν αλγόριθμο διάσημο για το ότι διαπρέπει σε ετερογενή πεδία. Κατά τη διάρκεια των πειραμάτων, χρησιμοποιούμε και μία προσωπικά υλοποιημένη γεννήτρια ΚΑΓ και μία ήδη υπάρχουσα γεννήτρια ΚΑΓ. Τέλος, σχετικά με τα πειραματικά αποτελέσματα, χρησιμοποιούμε κάποιες επικρατούσες μετρικές για να αξιολογήσουμε την επίδοση του αλγόριθμου CPOP και του αλγόριθμου HEFT.

Λέξεις-κλειδιά: πρόβλημα προγραμματισμού εργασιών, κατευθυνόμενοι άκυκλοι γράφοι, ετερογενή συστήματα, ο αλγόριθμος Critical-Path-on-a-Processor, τεχνητή νοημοσύνη

This dissertation’s general scientific field is the task scheduling in heterogeneous environments. The various tasks form an application which is modeled by a directed acyclic graph (DAG). We focus on the Critical-Path-on-a-Processor (CPOP) algorithm - a popular algorithm with good quality of results in the case of heterogeneous processors - and static task scheduling. After some required definitions, we analyze the problem and the way the CPOP algorithm works. While experimenting, we use the Heterogeneous Earliest-Finish-Time (HEFT) algorithm for comparison, an algorithm famous for shining in heterogeneous domains. During experiments, we use both a personally implemented DAG generator and an already existed DAG generator. Finally, regarding experimental results, we use some mainstream metrics in order to evaluate the performance of the CPOP algorithm and the HEFT algorithm.

Keywords: task scheduling problem, directed acyclic graphs, heterogeneous systems, the Critical-Path-on-a-Processor algorithm, artificial intelligence

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

PROLOGUE AND THANKS

This dissertation is the project that concludes my Master of Science in Informatics and Telecommunications at the Informatics and Telecommunications Department of University of Thessaly, established in Lamia, Greece. The fourth and last semester of the postgraduate studies is totally dedicated to the derivation of a dissertation that best fits the acquired knowledge and the interests of the postgraduate student. So, while personally I have acquired much knowledge at both basic fields (informatics and telecommunications), my high interest in algorithms and programming made it an easy choice to select the generic context of my dissertation. Of course, the exact contextualization of my project and some specific settings on the way were discussed and chosen together with my supervisor professor, Tziritas Nikolaos.

At this point, I would like to express my special thanks to my supervisor professor, not only for the starting ideas regarding the context of my dissertation, but also for the directions he was always giving me, the general communication we were having and the time he was spending in order to give me further directions and ideas, to make corrections on the go and to help me generally. Finally, I would also like to thank the other two members of the selection board of my dissertation. They also were teaching me various courses and knowledge taken from these courses was proven very useful for the derivation of my dissertation and for my personal upskilling regarding programming and algorithm analysis.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

TABLE OF CONTENTS

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

LIST OF ABBREVIATIONS

- **DAG**: **d**irected **a**cyclic **g**raph
- **QoS**: **q**uality **o**f **s**ervice
- **CPOP**: **C**ritical-**P**ath-**o**n-a-**P**rocessor (algorithm)
- **HEFT**: **H**eterogeneous **E**arliest-**F**inish-**T**ime (algorithm)
- **WMS**: **w**orkflow **m**anagement **s**ystem
- **AST**: **a**ctual **s**tart **t**ime (of a task)
- **AFT**: **a**ctual **f**inish **t**ime (of a task)
- **EST**: **e**arliest execution **s**tart **t**ime (of a task on a processor)
- **EFT**: **e**arliest execution **f**inish **t**ime (of a task on a processor)

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

LIST OF IMAGES AND GRAPHS

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

1. INTRODUCTION

1.1. The general task scheduling problem

Undoubtedly, a critical parameter for high-performance computing is how efficient is the scheduling of the computation tasks on processors. This issue has become the main objective of many studies over the past years and still is a top target to be researched extensively, as there are still many points and aspects that can be improved (especially in some specific settings). The task scheduling problem's two main components are the assigning of tasks that form an application to appropriate processors and the ordering of task executions on the given processors. Regarding the so-called application model, the application's representation is done via a directed acyclic graph (DAG). When we have set properties of the whole application (execution times, data sizes, as well as task dependencies) by deduction, we utilize a static model. However, sometimes task dependencies can change dynamically and this leads to non-static application models; these dynamic application models can still use a DAG (that dynamically changes), but a more sophisticated approach is needed (and this is not inside the confines of this dissertation).

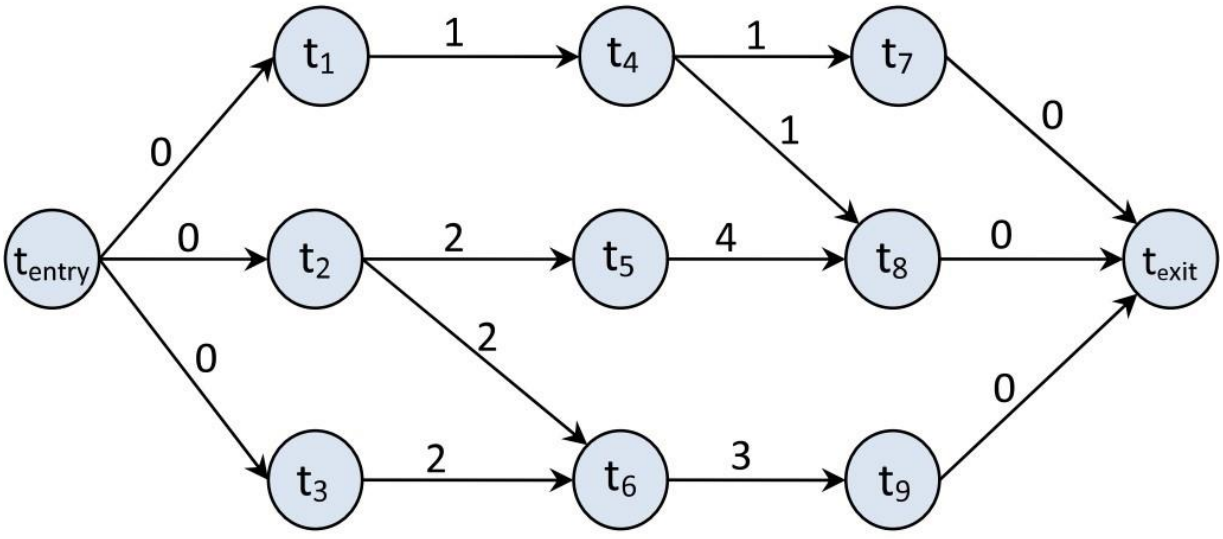


Image 1.1: A sample workflow

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

We can see that the DAG that models an application can vary from a very simple task graph to a very complex one with a great number of tasks and directed edges (that represent the dependency between the tasks) (Image 1.1, Image 1.2). As we can easily confirm observing these two workflows, there are no cycles.

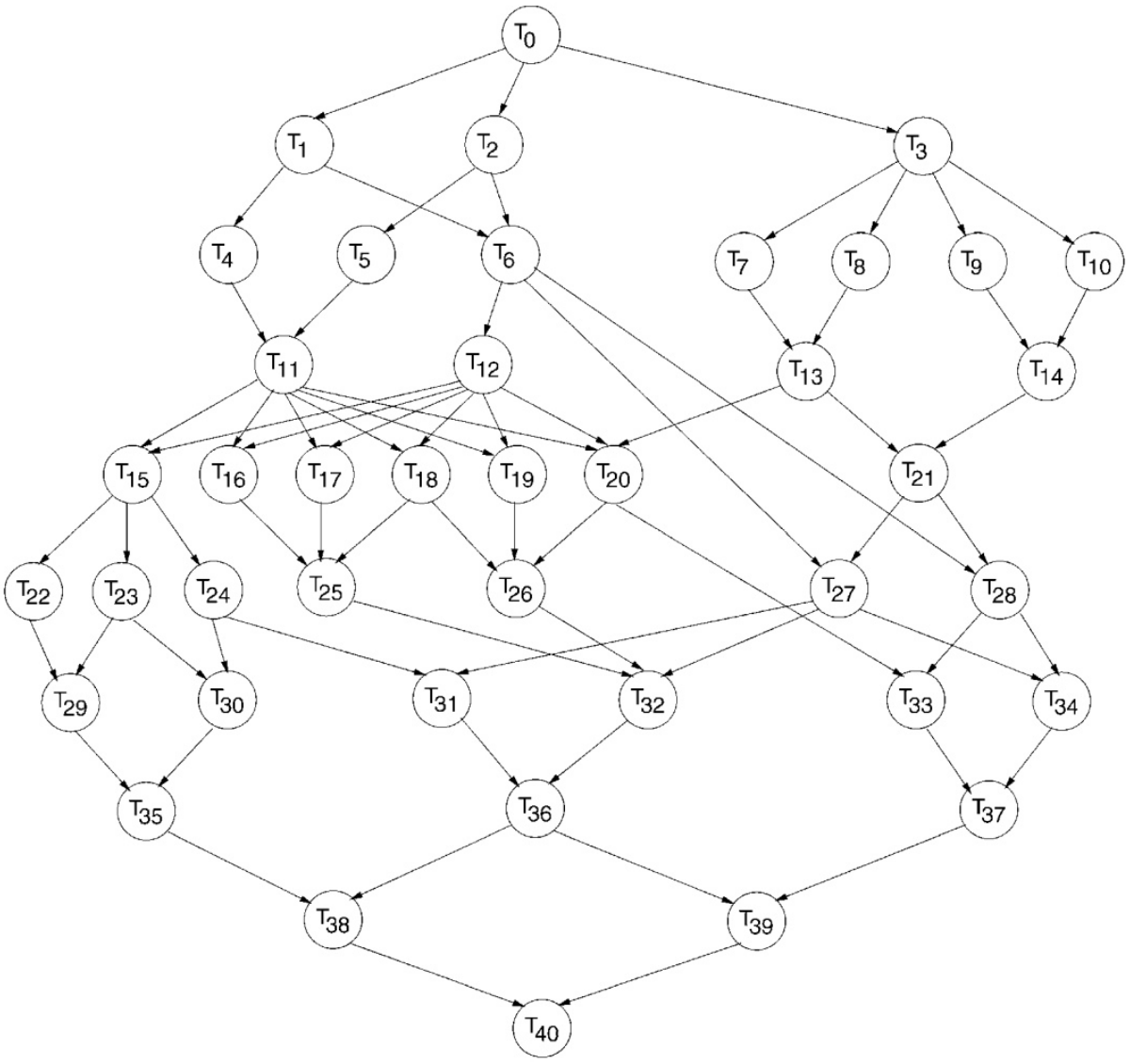


Image 1.2: A complex workflow with many tasks

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

1.2. The cloud computing concept

Having said nothing until now about processors, we should emphasize the advent of cloud computing, a manifest and mainstream trend in distributed computing that provides in the sense of services not only hardware infrastructure but also software applications [13, 14, 15, 19]. Via cloud computing services are offered through internet using the famous pay-as-you-go model [46]. Data centers make use of clever scheduling tactics to allocate resources to various tasks in an optimal way. A new way for developing workflow management systems (WMS) is presented by the cloud computing concept [16, 20]. Many researchers take into account cloud computing and newly developed WMSs and investigate the benefits of using them for their scientific applications. Workflow scheduling is a very important issue for optimization in cloud computing. An example of a such challenging issue is the problem of satisfying the quality of service (QoS) requirements of a user, as well as lowering as much as possible the cost of workflow execution [45]. Using cloud computing, we can speak about a cloud model, while always using a DAG for the application model (Image 1.3). Adopting the cloud model, a service provider offers virtualized resources to its clients and charges them accordingly. Through this provider, any interested person or company can get access to several computation services with different QoS parameters such as processor category and memory capacity. Of course, one can easily imagine that these variations lead to different pricing.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

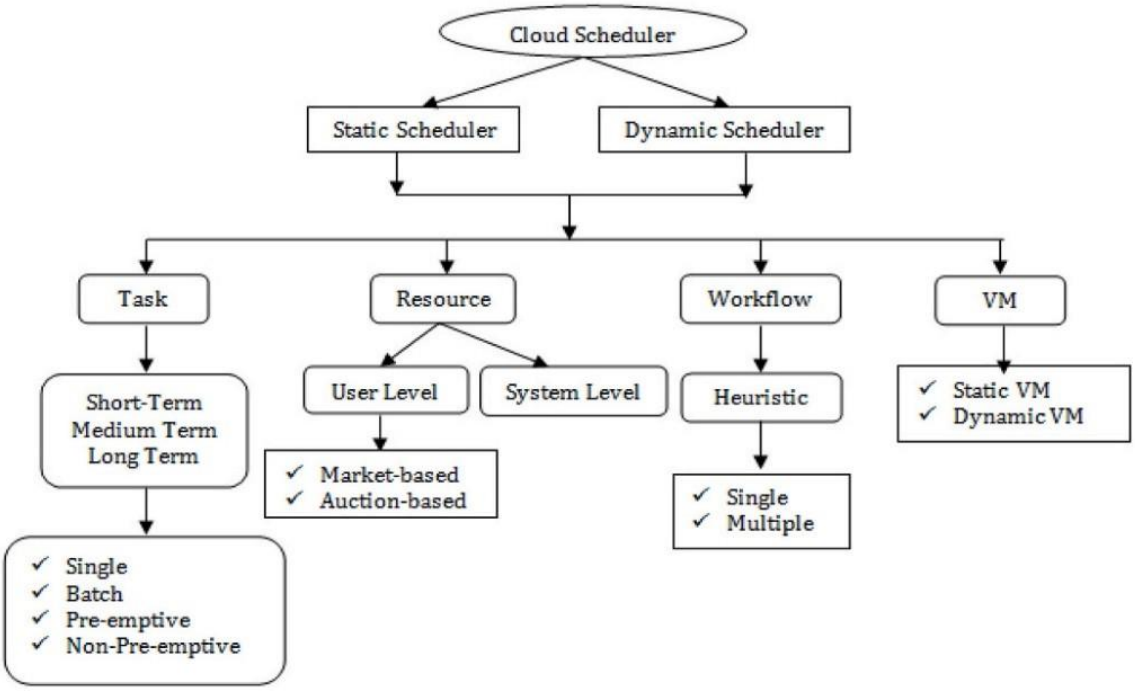


Image 1.3: Cloud scheduling levels

Because of its importance and its utility in realistic scientific workflows (Image 1.4), the task scheduling problem has already been researched to a great extent, as we stated before, but the plethora of algorithms that have been proposed in the related publications are mostly geared towards homogeneous resources. Even if they are not geared towards only homogeneous processors, it is proven in most cases that they are not suitable for heterogeneous resources; they do not usually give good quality of results or their complexity is high. As a consequence, other kinds of algorithms started to be researched in order to attain better results in the case of heterogeneous processors, a case with high needs for efficient algorithms nowadays due to the existence of cloud computing and the variation of its services. Without any doubt, task scheduling affects heavily the efficiency of the computing system and is very demanding due to the heterogeneity of cloud resources, meaning that the available resources can have a variable computation ability, different memory and personalized characteristics regarding communication.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

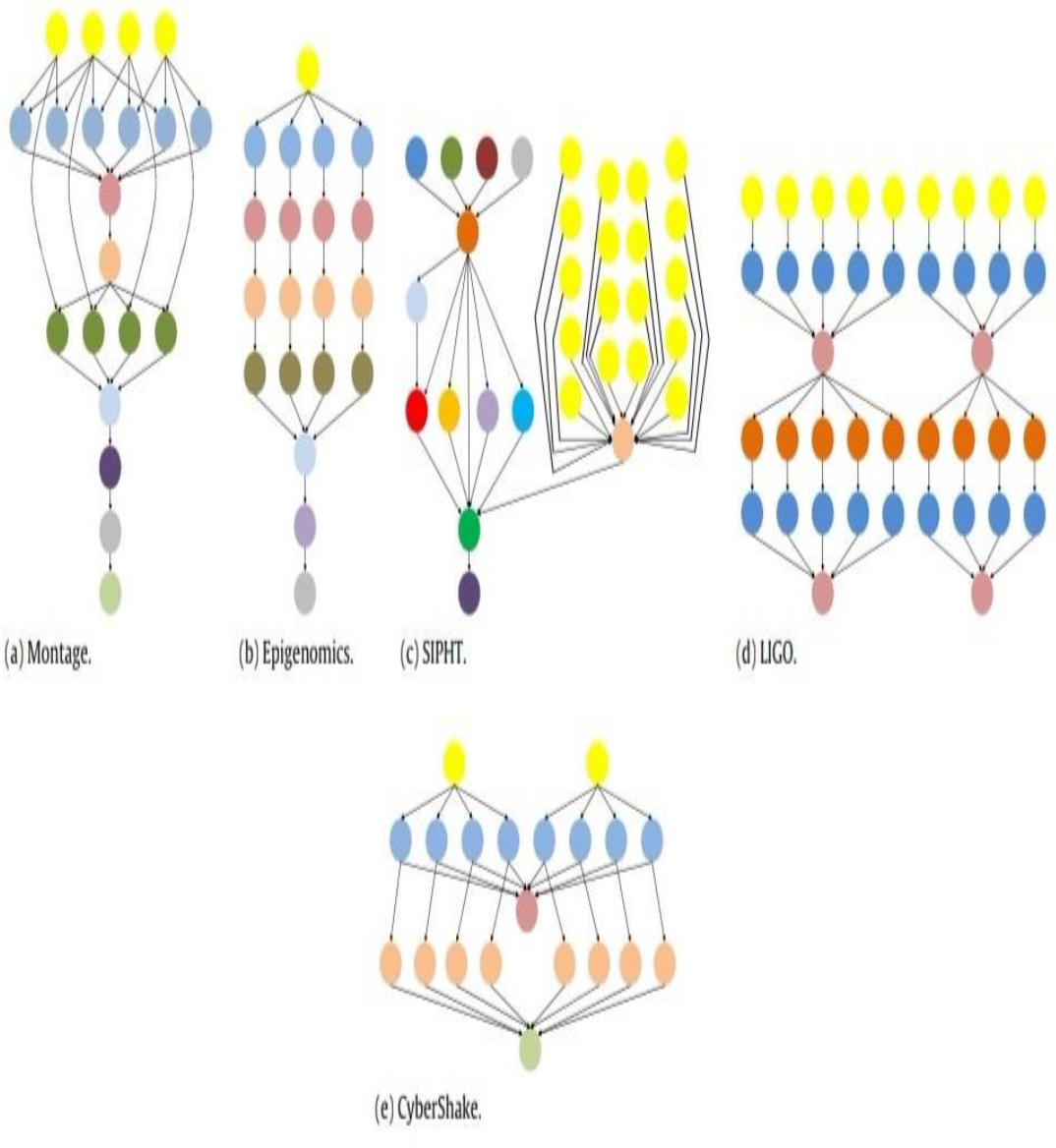


Image 1.4: The structure of five realistic scientific workflows

1.3. Objectives in conjunction with the existing literature

Due to the previous statements, this dissertation has been decided to focus on heterogeneous environments, where there is much more space and need for research. In

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

heterogeneous domains, low-complexity efficient heuristics are more than welcome. At this point, we should state that the general task scheduling problem is NP-complete [33, 50] and many research efforts propose heuristics suitable for heterogeneous processors nowadays. Two very popular algorithms with good quality of results - that dominate in heterogeneous domains - are the Critical-Path-on-a-Processor (CPOP) algorithm and the Heterogeneous Earliest-Finish-Time (HEFT) algorithm [1]. They have been presented in many papers and compared with other algorithms, having as a goal to achieve great performance and competitive scheduling time at the same time under the assumption of a constrained number of heterogeneous resources.

After an extensive literature review, we notice that many researchers have experimented with the HEFT algorithm, proposing some variations of it like the Experiential Heterogeneous Earliest Finish Time algorithm (for dynamic task scheduling) [2]. We decided to focus more on the CPOP algorithm and static task scheduling using a DAG for the representation of an application. For this dissertation, the initial objectives regarding the CPOP algorithm varied; an implementation in the Python programming language was a required first goal and - after studying and familiarizing with the CPOP algorithm's distinctive phases known as task prioritizing phase and processor selection phase - it seemed like a challenging idea to try making a novel variation at a phase after programming the original CPOP algorithm. Furthermore, an obvious objective after implementing the CPOP algorithm in Python was to experiment with various DAG generators (personally implemented and already existed), testing the efficiency of the CPOP algorithm given the way the input (DAG that models an application) of the CPOP algorithm is generated. Finally, regarding the aspect of data placement, which has become very popular in the domain of scientific workflows after the advent of cloud computing [3], we assume a fixed location of the datasets (non-flexible datasets; term is explained in the following chapter).

1.4. Potential work continuation

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

Of course, scientific work and research never stop and a project can always evolve further. As a result, there are many ways to continue with this project outside the confines of this dissertation, but this matter fits better to be stated in the last chapter after the conclusions. Making use of some more advanced tools (that were not so easily accessible due to various reasons) and thus achieving greater quality of experimentation, this dissertation can be transformed into a less extensive and more experiments-included paper in the near future.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

2. RELATED WORK

2.1. Related work regarding the CPOP and HEFT algorithms

Regarding static task scheduling in heterogeneous domains, the related literature is full of work and experimentation with the HEFT algorithm. A particular scientific workshop in 1999 was one of the first instances where the CPOP algorithm and the HEFT algorithm were presented and explained [1]. The provided comparison study showed that these two task scheduling algorithms outperform previously proposed heuristics (both performance-based and cost-based). The authors included not only efficiency study with respect to DAG size but also efficiency study with respect to DAG structure. Some years later (in 2002), the same authors managed to propose three alternate policies for the task prioritizing phase of the HEFT algorithm, but no alternate policy for the task prioritizing phase of the CPOP algorithm was proposed. Finally, a famous paper regarding the task scheduling problem focused in 2013 on a specific category of static task scheduling algorithms, the guided random search based algorithms [4]; the authors compared them with the HEFT and CPOP algorithms and presented various experimental results with no clear “winner” between the guided random search based algorithms and the heuristic based HEFT and CPOP algorithms, but with major performance differences according to the chosen comparison metric.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

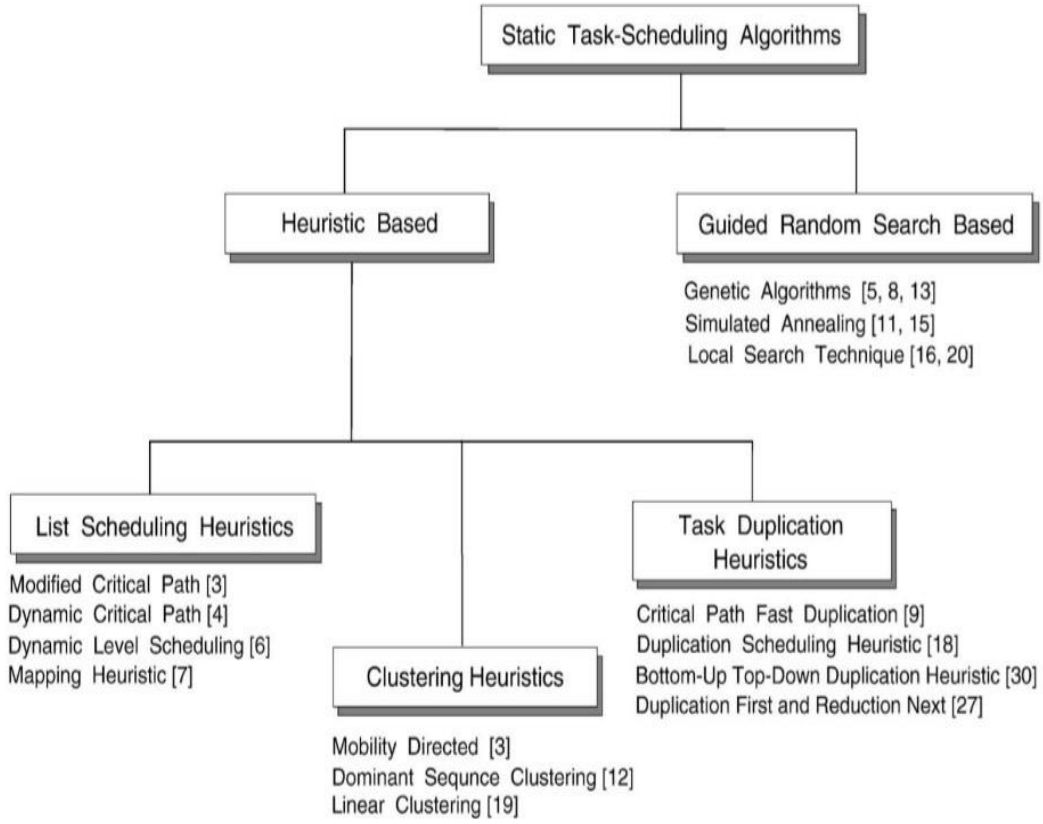


Image 2.1: Groups seen in related literature

The static task scheduling algorithms are categorized into groups with different characteristics and adopted aspects (Image 2.1). In this chapter, we will analyze the various heuristic groups (their advantages and disadvantages proven by extensive experimental evaluation after the initial theoretical concept) and the related work regarding them.

2.2. List scheduling heuristics

List task scheduling heuristics provide usually the best schedules with no worsening of the schedule length, having quadratic complexity in relation to the number of tasks [34, 48]. Generally, this type of task scheduling algorithms has two phases, the task prioritizing phase and the processor selection phase, as they are usually called by the various authors. In the task

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

prioritizing phase, each task is given a priority. If we have a tie regarding this task priority, the tie is administered by selecting a task randomly in most proposed heuristics. The processor selection phase results in choosing an appropriate resource that minimizes the adopted cost function (defined a priori). Furthermore, besides the good scheduling quality, the algorithms in this category dominate almost always in scheduling time [7, 8] and this sets them as a top group of heuristics to consider for static task scheduling.

Fast Load Balancing algorithm was proposed having as a basic goal to outperform regarding time complexity the HEFT algorithm [44]. It results in schedules similar to schedules of the HEFT algorithm, but its scheduling is far from good for irregular DAGs and for highly heterogeneous computing environments. However, in a specific paper, the authors proposed in 2007 a list task scheduling algorithm that was the first recorded list heuristic that managed to outperform the HEFT and CPOP algorithms and have better complexity; the related experimental results were derived both from randomly generated DAGs and DAGs of some real applications with known usage and importance [36]. Later, in 2014, another list heuristic was proposed; its complexity is quadratic in relation to the number of tasks. The authors conclude that this algorithm also outperforms the HEFT algorithm, while having no worse time complexity [32]. Getting more specific, the proposed algorithm is centered on a cost table, which the authors call optimistic because it does not take into account resources availability at a given time. Thus, the authors introduced a “look ahead” feature while complexity is still quadratic. Furthermore, in 2017, a modification of the HEFT algorithm was introduced [5]; the goal was to reduce both underloading cases and overloading cases of the existing resources and the modification achieved uniformly distributed load, a result not provided by the original HEFT algorithm and the CPOP algorithm.

Another author made an interesting overview on various list task scheduling algorithms (included the HEFT and CPOP algorithms) [6]. Overview’s experiments included also the Performance Effective Task Scheduling algorithm, the Lookahead algorithm and some other algorithms. The paper’s conclusion mainly stated the settings in which the Lookahead algorithm outperforms the other list task scheduling algorithms. The Lookahead algorithm relies mostly on the HEFT algorithm, whose “signature” is its novel processor selection policy. The Lookahead algorithm manifests the same formation as the HEFT algorithm; however, it

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

computes earliest execution finish time (EFT) for each descendant task of the current task [32]. Two authors also proposed the Best Imaginary Level algorithm; they define a static level for tasks that incorporates the effect of communication between the processors overhead and heterogeneity of resources [43]. They concluded that the proposed heuristic provides a schedule that can be called optimal if the topology of the input DAG is linear.

The Mapping Heuristic algorithm and the Dynamic Level Scheduling algorithm also belong to the group of list task scheduling heuristics; they suppose a model that addresses how to manage the conflicts of communication links between tasks, but their implementations for a network in which all nodes have a direct communication link to every other node, were used mainly for comparisons by many authors. The scheduling outputs that the Mapping Heuristic algorithm yields are usually worse than the other developed heuristics; the cause is that the algorithm marks a resource as ready at the time it ends the last task given to it [41]. So, the algorithm does not take into account the scenario that a resource that is engaged when a task is being scheduled have the ability to deal with the task more quickly; the aftermath is, of course, worse scheduling. The Dynamic Level Scheduling algorithm was one of the first algorithms that incorporated some kind of evaluation of processors' readiness; it did not prevent a task from being scheduled on a currently engaged resource [42]. As a result, the Dynamic Level Scheduling algorithm provides better results than the Mapping Heuristic algorithm [32]. Additionally, we should mention that the Dynamic Level Scheduling algorithm does not try to fill unused time periods between two tasks already scheduled on the same processor in a processor schedule; this is an obvious differentiation from other (mostly newer) list task scheduling algorithms.

Finally, a paper proposed a novel heuristic that adopts a blended approach for task scheduling using DAGs; this interesting algorithm segments the problem to smaller ones for scheduling tasks with no dependency [47]. The authors stated that this novel algorithm can result in having a new group of algorithms regarding the task scheduling problem. The same authors also focused on the problem of scheduling more than one DAG at the same time using a set of heterogeneous resources [49]. Their aim was to make better the makespan and to grant fairness (defined regarding the issue of the downshift that each DAG would confront because of the fact that the DAG has conflicts for resources with other DAGs). This paper provided an

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

initial study of heuristics for multiple DAG scheduling, a much more challenging version of the general task scheduling problem.

2.3. Task duplication heuristics

A novel way to face the task scheduling problem is via use of task duplication heuristics. The main concept of this group is to schedule a DAG by mapping a subset of its tasks in a superfluous way; this lessens the processor-to-processor communication overhead [23]. A proposed algorithm using task duplication heuristics managed to achieve both optimal makespan and improved resource utilization [21]. The authors' strategy capitalized on the existing time slots on resources in order to advance workflow tasks' start time and completion time (while ensuring the budget of workflows).

The whole general work regarding this group of heuristics also proved that - given that duplication-based task scheduling algorithms vary due to chosen duplication strategy - they are mostly suitable for same-type processors and that the algorithms in the other categories are noticeably better regarding complexity [24]. Generally, the duplication-based heuristics produce the best makespans (in comparison with list task scheduling or the - later analyzed - clustering-based scheduling) [22, 38], but the disadvantages, besides the already mentioned high time complexity (for example, a cubic one, in relation to the number of tasks), expand further into the fact of more processor energy consumption due to the duplication.

2.4. Clustering heuristics

An interesting strategy that has been heavily proposed makes use of clustering heuristics to manage the task scheduling problem [26, 27]. Generally, clustering task scheduling algorithms try to schedule tasks with massive communication on the same processor, ignoring the factor of the availability of other resources in the system. As a consequence, clustering heuristics sacrifice parallelism and "exchange" it with interprocess

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

communication [36]. A clustering heuristic maps the tasks of a DAG to some clusters. At each step, regarding the chosen tasks for clustering, there is no constraint to be a ready task. Each iteration negates the previous clustering and modifies it; some clusters become merged. Also, execution of tasks of the same cluster will take place on the same processor when making use of clustering heuristics [28, 30].

We should note that a clustering algorithm needs extra steps to reach a definitive schedule. Firstly, we have a cluster merging step so that the number of clusters after the merge is equal to the number of processors. Then, a cluster mapping step exists for mapping the clusters on processors. Finally, a task ordering step orders the mapped tasks [25]. Clustering for Heterogeneous Processors algorithm is a well-known heuristic based on the clustering approach [37]. A comparison work showed that the Dominant Sequence Clustering algorithm is superior in practice regarding the group of clustering algorithms; generally, this heuristics group is characterized from having higher parallel time and being slower [29].

2.5. A reference to dynamic task scheduling

As stated in the introduction - despite the fact that we are moving for a while to dynamic task scheduling (in heterogeneous domains) - we should make reference to the Experiential Heterogeneous Earliest Finish Time algorithm, a variation presented during an international conference [2]. After experimenting with the original HEFT algorithm, the researchers proposed this experiential algorithm in 2019.

The most interesting thing to state regarding this novel algorithm is that these researchers managed to implement a modification at the task prioritizing phase of the original HEFT algorithm. More analytically, they modified the way the HEFT algorithm computes the ranks of the tasks; the authors introduced a parameter that designates the minimum average execution time of a task on each pertinent processor. Experimental results indicated that the Experiential Heterogeneous Earliest Finish Time algorithm outperforms both the original HEFT algorithm and the CPOP algorithm, so we can conclude that this variation is ideal for cloud environments with heterogeneous resources.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

2.6. Task scheduling enhanced by artificial intelligence

Even if there are so many groups of algorithms and proposed approaches in the existing literature regarding task scheduling in heterogeneous domains, there is still need for further optimization of the ways the computing system operates. The (mostly commercially driven) demand for further optimization from the era of cloud computing onwards can mainly be achieved through even better scheduling of the tasks that form an application [14, 16]; this needed progress in the quality of scheduling is reachable in many situations with the contribution of artificial intelligence and its various subsets like machine learning and neural networks.

Machine learning - and more specifically reinforcement learning - can provide great schedules via its great abilities in making predictions and decisions; its function is based on patterns and inference. For example, artificial intelligence can quickly learn from patterns related to the behaviour of a scheduler. Furthermore, heterogeneous resource environments, like the ones of cloud computing, can rely on these “intelligent” schedules presented by artificial intelligence methods [55] and, as a consequence, reach new heights of performance, which can be safely considered unattainable without the use of artificial intelligence in most cases. Thus, reinforcement learning is implemented as a powerful tool to deal with the task scheduling problem nowadays and neural networks can be introduced into this type of machine learning, utilizing them for the training of the learning process. We should note that learning is a continuous process; the knowledge base of an agent always adds more information. Undoubtedly, higher learning rates and convergence speed of agents are important goals and achieving them can lead to task scheduling’s further evolution.

(Artificial) neural networks are computational models based on the way biological ones in the human brain process acquired information. Their neurons’ (interconnected nodes) main tasks are analyzing data and making decisions. After a proper training, they have the ability to give solutions to complex problems like the task scheduling problem, especially given a high heterogeneity in resources.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

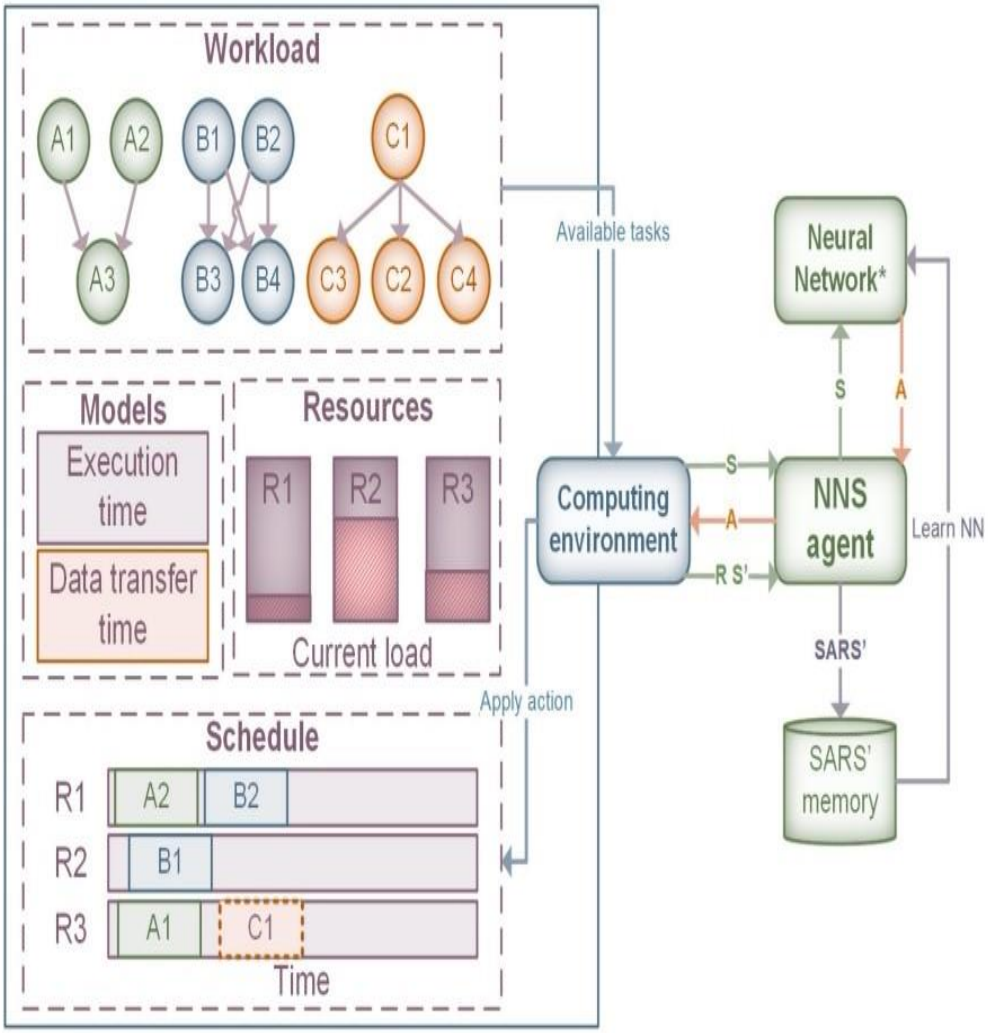


Image 2.2: Neural network scheduling scheme

Regarding task scheduling utilizing artificial intelligence in heterogeneous domains, the related literature expands year by year, as there is much space for novel and effective projects. In 2015, a study regarding task scheduling in cloud computing included a great analysis of the impact of reinforcement learning on virtual machine scheduling [53]. Two examples of the above-mentioned impact are less energy consumption and dealing with the resource shortage issue. The authors also made an overview regarding the learning

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

mechanism, focusing then on its implementation in scheduling, and proposed a novel way to train a scheduler. Some other authors proposed a noteworthy scheduling algorithm (Image 2.2) that mainly incorporates a neural network that achieves great evaluation of the various conditions and manages to make favorable and profitable decisions for the assignment of a task to a resource [51]. A great amount of work is put in order to encode the status and the conditions of the computing environment. Experimental results confirmed great quality in terms of the schedule length, making the proposed algorithm a dominating one regarding that metric.

Later, a research article featured a novel reinforcement learning algorithm that makes use of a combinatorial optimization algorithm [52]. This results in great performance in many different environments, even when the algorithm confronts tough and complicated problem scenarios. Finally, in 2024, a paper presented a system framework that incorporates a scheduler enhanced by reinforcement learning (Image 2.3); this framework is ideal for many environments like the cloud environment [54]. Specifically, the authors used an algorithm based on reinforcement learning and gave it the role of the core scheduler. As a scheduler, it manages to provide shorter execution time of tasks, given the settings of the computing environment.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

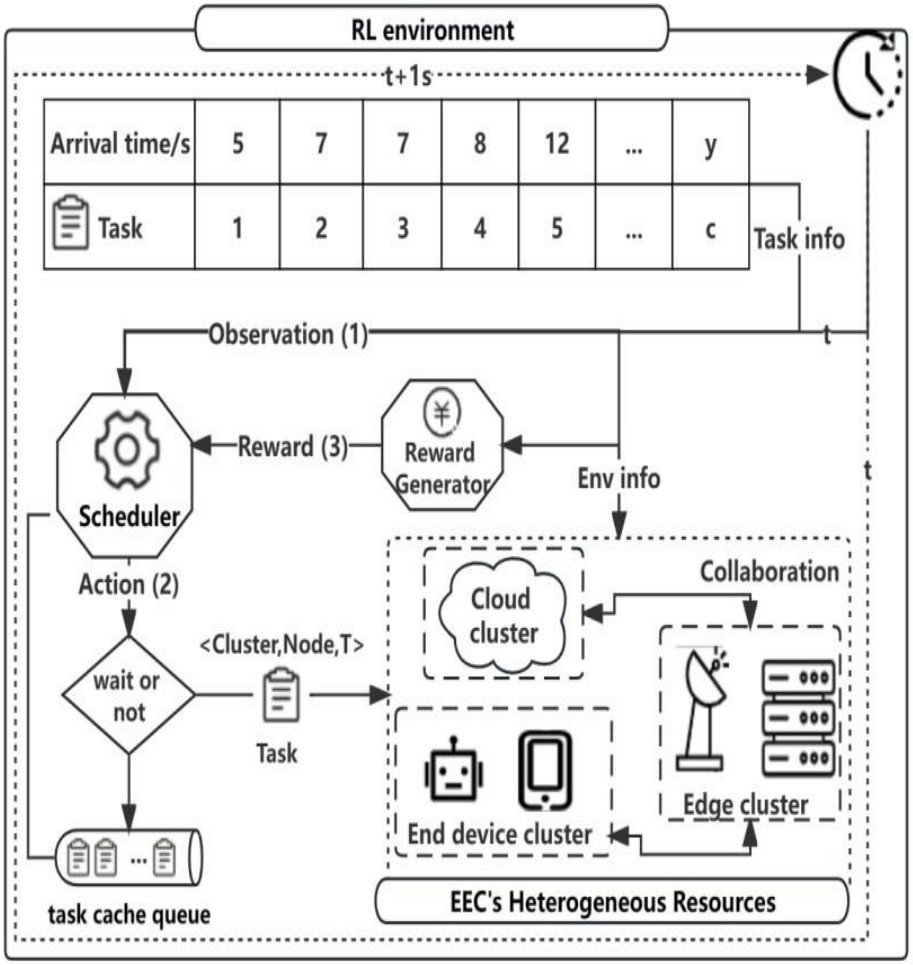


Image 2.3: Reinforcement learning model

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

3. MODEL DEFINITION AND PROBLEM ANALYSIS

3.1. System model

A parallel or distributed application is analyzed into a set of tasks with various sizes and data dependencies among them. As stated in the introduction, DAGs are a great tool for the split of an application into tasks. Thus, the various task scheduling algorithms, like the already mentioned CPOP and HEFT algorithms, take a DAG as an input. Each task of a DAG relates to the arrangement of operations with a defined size and a directed edge denotes the existence of dependency among the tasks.

So, in our model, a weighted DAG stands for an application. To be more precise, a DAG is represented by the graph G = (V,E), where V is the set of v tasks (nodes) and E is the set of e edges between the tasks. We assume a set P of p heterogeneous processors and also a total connectivity configuration regarding them. All communications between resources in the computing system are hypothesized to function with no contention.

Firstly, each edge (i,j) of set E represents a task-dependency constraint; task n_i of set V should finish its execution before task n_j of set V can be started. In a given DAG, a task (node) with no predecessor node is an entry task and a task (node) with no descendant node is an exit task. If a task graph has more than one entry task, the entry tasks are connected to a virtual entry task without cost. Similarly, if a task graph has more than one exit task, the exit tasks are connected to a virtual exit task without cost. In both cases, the edges used for connection purposes have no cost. The reason for this common practice regarding entry and exit tasks (before a task scheduling algorithm takes action) is that the majority of task scheduling algorithms take as input single-entry and single-exit task graphs. We assume that computation can coincide with communication. The application is represented with a static model. It is known that data in the cloud environment can be categorized into two main groups. The first group (the one assumed here) consists of unmovable datasets with fixed locations and the second group consists of flexible datasets without constraints to set them unmovable.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

Based on the above definitions, we have a v x p computation cost matrix W; $w_{i,j}$ represents the estimated execution time to complete task n_i on processor p_j. The average execution time of task n_i (notation: $\overline{w_i}$) is the sum of all $w_{i,j}$ (j = 1,2,…,p) divided by p. We also have a p x p matrix B that stores the data transfer rates between the available resources and a p-dimensional vector L that stores the communication startup times of the above-mentioned resources of the computing system. So, $\bar{B}$ is the notation of the average data transfer rate among the processors and $\bar{L}$ is the notation of the average communication startup time of the processors. Finally, we have a v x v matrix Data; $data_{i,j}$ represents the amount of data needed to be sent from task n_i to task n_j.

Regarding the communication cost of edge (i,k) of set E, which is for transferring data from task n_i scheduled on processor p_m to task n_k scheduled on processor p_n, it is defined as follows:

$$c_{i,k} = L_m + \frac{data_{i,k}}{b_{m,n}}$$

When tasks n_i and n_k are scheduled on the same processor, $c_{i,k}$ becomes zero (under the assumption that the intraprocessor communication cost is insignificant in comparison with the interprocessor communication cost). The average communication cost of edge (i,k) is defined as follows:

$$\bar{c}_{i,k} = \bar{L} + \frac{data_{i,k}}{\bar{B}}$$

3.2. The makespan metric and the critical path

The most popular metric in task scheduling is the makespan (or schedule length); the makespan represents the finish time of the exit task in a scheduled DAG or - simply - the total

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

execution time. Let Pred(n_i) be the set of immediate predecessor tasks of task n_i, Succ(n_i) be the set of immediate successor tasks of task n_i, AST(n_i) be the actual start time (AST) of task n_i and AFT(n_i) be the actual finish time (AFT) of task n_i. The earliest execution start time (EST) of task n_i on processor p_j is defined as follows:

$$EST(n_i, p_j) = \max\{T_{avail}(p_j), \max(AFT(n_m) + c_{m,i})\}$$

where $T_{avail}(p_j)$ is the earliest time at which processor p_j is ready to execute the task and, regarding the inner max block, $n_m \in Pred(n_i)$. We should note that the communication cost $c_{m,i}$ is zero if the predecessor task n_m is assigned to processor p_j. Also:

$$EST(n_{entry}, p_j) = 0$$

Furthermore, the earliest execution finish time (EFT) of task n_i on processor p_j is defined as follows:

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j}$$

Finally, after task n_i is scheduled on processor p_j, EST(n_i,p_j) is equal to AST(n_i) and EFT(n_i,p_j) is equal to AFT(n_i). Based on the above definitions:

$$makespan = AFT(n_{exit})$$

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

An important definition regarding DAGs is the definition of the critical path; the critical path of a DAG is the longest path from the entry task (node) to the exit task (node). The length of the critical path is the sum of the computation costs of the nodes and the communication costs between the nodes along the critical path. We should note that the length of the critical path of a DAG is the lower threshold of the makespan.

The various task scheduling algorithms sometimes incorporate directly the idea of the critical path. For example, the CPOP algorithm is a paradigm of direct utilization of the critical path (the reasoning becomes obvious in the next chapter), as its name implies. Other algorithms do not utilize the critical path concept in a direct and obvious way; an example is the HEFT algorithm, which incorporates implicitly the critical path in its task prioritizing phase in order to compute an attribute called upward rank. Either way, the critical path is a noteworthy concept regarding DAGs and task scheduling and its above-mentioned relation to the makespan is noteworthy too.

3.3. Problem formulation

In this chapter, we defined thoroughly the system model that we adopt regarding the task scheduling problem; a static model with a weighted DAG to play the role of an application and with various heterogeneous processors. Given these settings, the objective function of the task scheduling problem is to determine the assignment of tasks of a given application to processors in a way that this assignment minimizes the makespan. Of course, all the precedence constraints should be taken into consideration. Satisfying them, the assignment of the tasks to the heterogeneous resources should result in the minimum makespan possible [39, 40].

Given the certainty of the fact that the DAG has no more than one exit task (if it has more than one exit task, then - as we stated when discussing about the system model - the exit tasks are connected to a virtual exit task), the above problem formulation could safely be interpreted in saying that the objective function of the task scheduling problem is to determine the assignment of tasks to processors such that the AFT of the (single) exit task is minimized.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

In the next chapter, we examine thoroughly how this objective is met using the CPOP algorithm. An analysis of its two distinctive phases is needed, as well as some more definitions that do not relate to the generic aspects of the task scheduling problem, but to the function of the specific algorithm.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

4. THE CPOP ALGORITHM AND IMPLEMENTATION IN THE PYTHON PROGRAMMING LANGUAGE

4.1. The CPOP algorithm

Focusing now on the CPOP algorithm (Image 4.1), its time complexity is $O(v^2 \times p)$, the same as the time complexity of the HEFT algorithm. The CPOP algorithm is a list scheduling heuristic for task scheduling on a set number of heterogeneous resources. We call critical path processor the processor p_j of set P that minimizes the length of the critical path.

1. Compute $rank_u$ for all nodes by traversing graph upward, starting from the exit node.
2. Compute $rank_d$ for all nodes by traversing graph downward, starting from the start node.
3. $|CP| = rank_u(n_s)$, where n_s is the *start* node.
4. **For** each node n_i do
5. **If** $(rank_d(n_i) + rank_u(n_i) = |CP|)$ then
6. n_i is a critical path node (CPN).
7. Select the critical-path-processor that minimizes $\sum_{n_i \in CPN} w_{i,j}$.
8. Initialize the priority-queue with the entry nodes.
9. **while** there is an unscheduled node in the priority-queue do
10. **begin**
11. Select the highest priority node from priority-queue,
12. which maximizes $rank_d(n_i) + rank_u(n_i)$.
13. **if** (n_i is a CPN) then
14. schedule n_i to critical-path-processor.
15. **else**
16. Assign the task n_i to the processor p_j which minimizes the (EFT) value of n_i.
17. Update the priority-queue with the successor(s) of n_i if they become ready-nodes.
18. **end**

Image 4.1: The CPOP algorithm

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

Before familiarizing with the CPOP algorithm's phases known as task prioritizing phase and processor selection phase, we should talk about two attributes of the tasks (nodes), the upward rank and the downward rank. The upward rank of task n_i is defined as follows:

$$rank_u(n_i) = \overline{w}_i + \max(\bar{c}_{i,j} + rank_u(n_j))$$

where $rank_u(n_{exit}) = \overline{w_{exit}}$ and, regarding the max block, $n_j \in Succ(n_i)$. Because of the fact that this rank is calculated via recursion by exploring the task graph upwards, starting from the exit task, it has the particular name (upward rank). The downward rank of task n_i is defined as follows:

$$rank_d(n_i) = \max(rank_d(n_j) + \overline{w}_j + \bar{c}_{j.i})$$

where $rank_d(n_{entry}) = 0$ and, regarding the max block, $n_j \in Pred(n_i)$. Because of the fact that this rank is calculated via recursion by exploring the task graph downwards, starting from the entry task, it has the particular name (downward rank).

Starting with the task prioritizing phase, upward and downward ranks are computed for all tasks. The CPOP algorithm uses the sum of upward rank and downward rank ($rank_u(n_i)+rank_d(n_i)$) of the tasks in order to set the priority of each task. In the beginning, the entry task is the chosen task and is signified as a critical path task. The algorithm chooses an immediate successor of the selected task based on highest priority value and signifies it as a critical path task. Until the exit task is reached, the above-stated process is repeated.

In the processor selection phase, if the critical path includes the current task, the task is assigned to the critical path processor; in the other case, it is assigned to the processor that minimizes the EFT of the task.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

1. Compute $rank_u$ for all nodes by traversing graph upward, starting from the exit node.
3. Sort the nodes in a list by nonincreasing order of $rank_u$ values.
4. **while** there are unscheduled nodes in the list **do**
5. **begin**
6. Select the first task n_i in the list and remove it.
7. Assign the task n_i to the processor p_j that minimizes the (EFT) value of n_i.
8. **end**

Image 4.2: The HEFT algorithm

Finally, we should clarify that, despite the fact that the HEFT algorithm (Image 4.2) has also two distinctive phases known as task prioritizing phase and processor selection phase, both phases of the CPOP algorithm are different from those of the HEFT algorithm. For example, in the task prioritizing phase of the HEFT algorithm there is no use of the downward rank of tasks; the HEFT algorithm takes only the upward ranks of tasks into consideration. Furthermore, the CPOP algorithm explicitly identifies the critical path and its scheduling decisions mainly focus on optimizing the execution of tasks on the critical path, while the HEFT algorithm does not identify the critical path and has more flexibility in assigning tasks to different processors. However, as we stated briefly in the previous chapter, the idea of the critical path is embedded in its task prioritizing phase and - more specifically - in calculation of the upward ranks of tasks.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

4.2. Various programming aspects

Starting with required Python modules, the most important Python module we made use of during the implementation of the original CPOP algorithm is no other than the queue module. As someone can easily find out by checking this module's documentation, the queue module implements three queue types. We imported "PriorityQueue" from the queue module via a simple "import" statement. Using this queue type (priority queue), the records remain sorted (using the heapq module) and the record with the lowest value is accessed first.

Secondly, the advantages of functional programming (like hierarchical design, readability, abstraction and having less extensive code due to reusability of functions) are blatant, so we constructed functions for computing values like the upward and downward ranks of tasks, the EST and the EFT of tasks on each of the resources and the makespan.

Finally, regarding the entire implementation of the CPOP algorithm in the Python programming language, its objective is task scheduling, so the input it takes is obviously a DAG (serialized in JSON format). We did an extensive analysis in the previous chapter regarding how the CPOP algorithm handles the task scheduling problem using DAGs; thus, the implementation in Python is nothing difficult to imagine for someone experienced generally in programming.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

5. NOVEL AND INTELLIGENT VARIATIONS OF THE CPOP ALGORITHM

5.1. The idea of alternate phase policies

Both the CPOP algorithm and the HEFT algorithm have two distinctive phases known as task prioritizing phase and processor selection phase and we should note that both phases of the CPOP algorithm are different from those of the HEFT algorithm. As stated in a previous chapter, alternate policies for the task prioritizing phase of the HEFT algorithm have been proposed with success, but no alternate policy for this phase of the CPOP algorithm has been proposed. So, an early thought within the context of this dissertation was to try making a novel variation at a phase of the CPOP algorithm. Undoubtedly, it was a challenging idea.

Focusing firstly in the task prioritizing phase of the CPOP algorithm, the only aspect that one can think to compute in a different way is how the upward and downward ranks of tasks are computed. But it seems that computing these two attributes of the tasks recursively, as their definitions (stated in a previous chapter) imply, is the only way to go. On the flip side, changing something in the processor selection phase of the CPOP algorithm never seemed like a good idea; it would denature the algorithm. For example, an alteration in this phase would act against the algorithm's tendency to optimize the execution of tasks on the critical path.

5.2. Important already existed novel and intelligent variations

However, despite the above-mentioned remarks, some variations of the CPOP algorithm have been proposed. The impact of these variations varies; in specific settings some variations result in improved efficiency and in some other settings they may be the only way to deal with a given problem. One interesting variation of the CPOP algorithm is an application of ant colony optimization. This is a famous population-based metaheuristic that evidently has great utility in finding non-exact solutions to difficult optimization problems [9,

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

18, 31]. Going into detail about this metaheuristic that provides approximate solutions, artificial ants, some software agents, search for as good as possible solutions to an optimization problem. In order to utilize ant colony optimization, the initial problem must be modified; it is turned into the problem of determining the optimal path on a weighted graph. The artificial ants progressively construct solutions by traversing the DAG. The process is heavily affected by a pheromone model; the artificial ants modify at execution time the values of various parameters related to nodes or edges. This variation of the CPOP algorithm is very effective when we deal with a large DAG, offering a more global search of task scheduling options.

Another known variation of the CPOP algorithm can be achieved with the aid of genetic algorithms. Genetic algorithms are also population-based heuristic algorithms [10, 11, 35]; they are search algorithms that solve (constrained and unconstrained) optimization problems using practices and strategies inspired by natural evolution. Main axioms of natural evolution are reproduction, natural selection and diversity of the species (that is maintained by the differences between consecutive generations). We should note that application of genetic algorithms is one of the most ideal ways to solve a problem in cases we know little information and we do not have a whole “image” of problem components. The advantage of a variation of the CPOP algorithm that uses genetic algorithms is improved makespan when the DAG is large and complex.

Finally, when execution time of tasks is characterized by uncertainty or imprecision, fuzzy logic can be introduced in order to deal with these uncertainties. Fuzzy logic is an applied method that accepts more than one possible truth value through the same variable. This approach is designed to solve problems using the concept of considering all information at our disposal and making the best possible decision given the input. Going into detail about fuzzy logic, while in standard logic – the one we are accustomed to - all statements have one or zero as a truth value, in fuzzy logic statements can have a value of partial truth, such as 0.5 or 0.7. Theoretically (but also practically, as fuzzy logic implementations have proven), this provides more chances and ways to get close to real-life circumstances because statements of absolute truth or falsehood are not so common in real life. Applications of fuzzy logic are seen in a broad range of fields nowadays and one of these

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

fields is task scheduling algorithms [12, 17], so there is a variation of the CPOP algorithm that uses fuzzy logic.

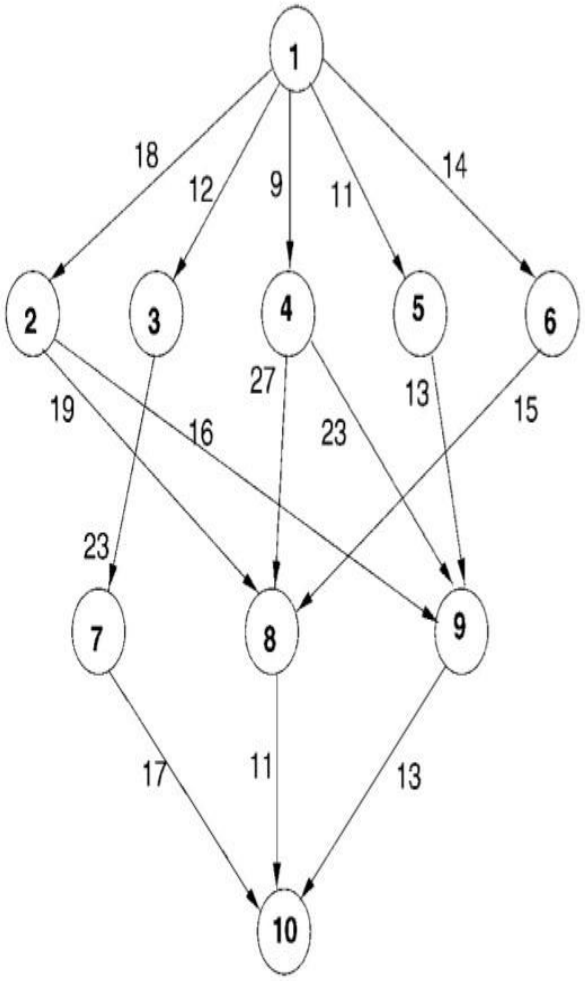
Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

6. EXPERIMENTAL EVALUATION

6.1. Experimental setup

DAG generators are algorithms that generate DAGs in a desired way (for example, randomly given a constraint regarding the number of nodes or the shape of the DAG to be generated). The generated DAGs are very useful when experimenting with a task scheduling algorithm like the CPOP algorithm; we can use the generated DAGs to evaluate the performance of a task scheduling algorithm. Furthermore, we can test the change in the efficiency of a task scheduling algorithm when the way its input (DAG) is generated changes (due to use of a different DAG generator). During experiments, in order to achieve relatively good quality of results, we used many combinations of input parameters to generate diverse DAGs with various characteristics. Thus, we examine, for example, the algorithms with respect to graph size. The number of tasks (nodes) can vary and this affects the results greatly, making one task scheduling algorithm better and more suitable and one other worse and less suitable. Also, we include performance study with respect to graph structure, which also can vary and, as a consequence, the parallelism of a DAG can vary. This fact regarding parallelism has no negligible importance, as it affects clearly the scheduling procedure, regardless the task scheduling algorithm chosen. The above-mentioned experimental use of many combinations of input parameters to generate various DAGs is very important in order to avoid being biased towards a particular task scheduling algorithm. Furthermore, it allows to form subgroups of DAGs, which have a set common parameter; this leads to experiments for each subgroup individually, giving the chance to come to a conclusion about a subgroup (for example, concluding regarding DAGs with low parallelism). Finally, we should state the utilization of an additional metric besides the makespan in some experiments, which happens to incorporate in a direct way the makespan in its formula.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250



Computation Costs

Task	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

Image 6.1: A task graph and the computation costs of its tasks per resource

Before continuing with the experimental evaluation, we can firstly see a scheduling example regarding the CPOP algorithm in comparison with the HEFT algorithm (Image 6.1, Image 6.2).

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

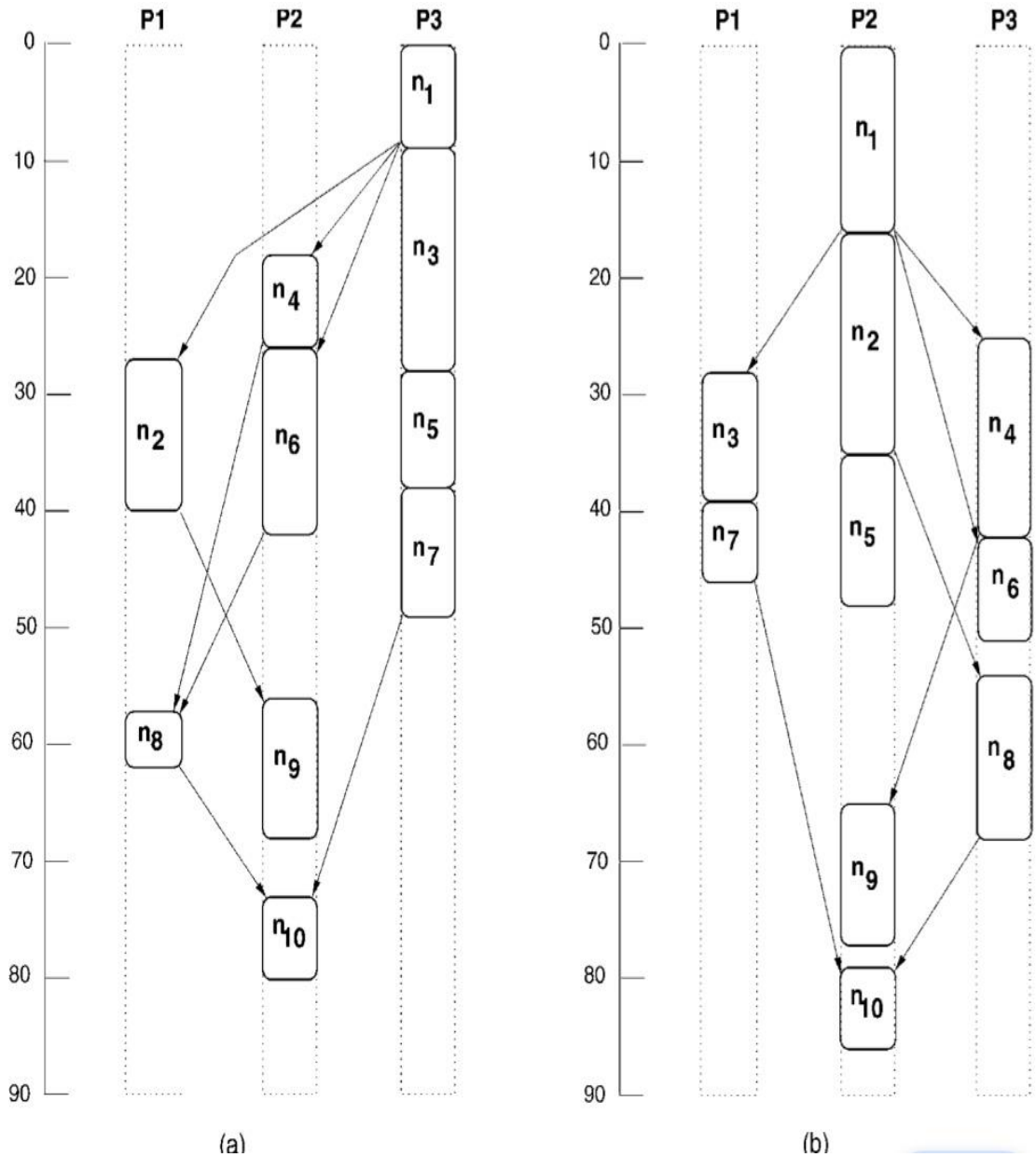


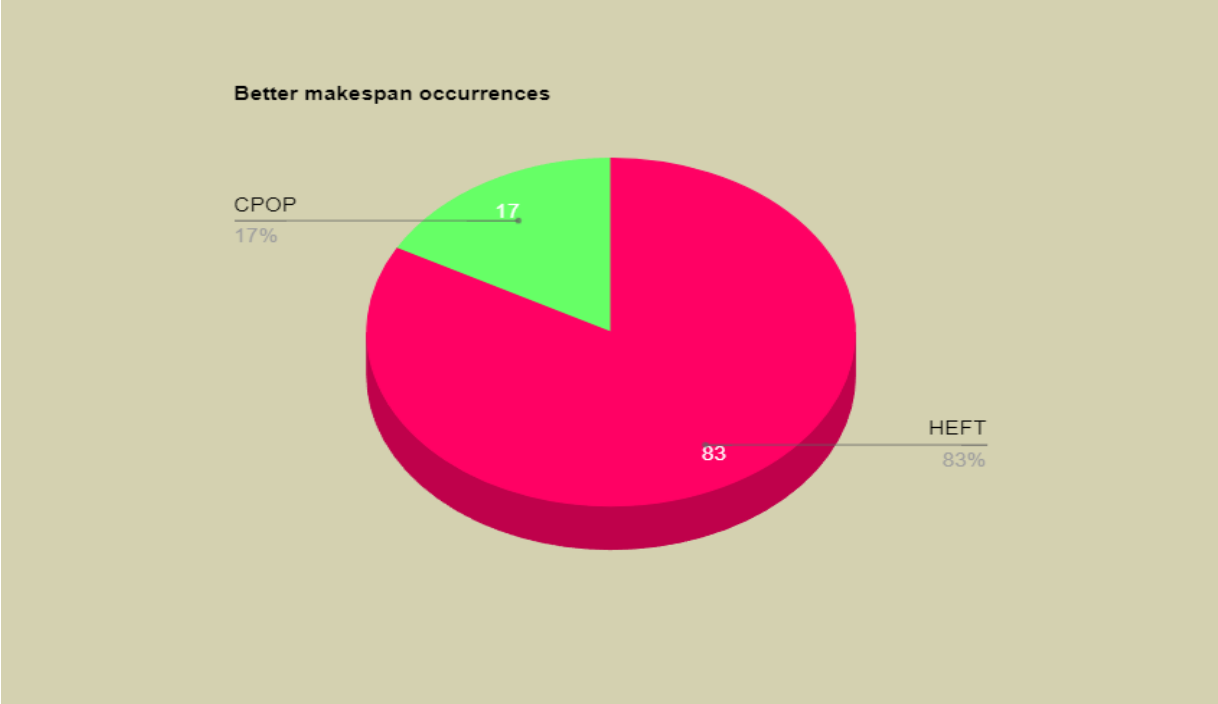
Image 6.2: Scheduling of the previous DAG with the HEFT (a) and CPOP (b) algorithms

6.2. Experimenting with a personally implemented DAG generator

We developed a DAG generator in the Python programming language; this generator constructs DAGs as randomly as possible. Regarding required Python packages, a very important Python package we made use of during the implementation of this DAG

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

generator is no other than the networkx package. The networkx package is ideal for creation, manipulation and study of complex networks. We imported the networkx package and required Python modules (like the random module and the json module) via simple “import” statements. Of course, this DAG generator is developed taking into consideration that its output is not the final objective; the output of the generator is used as an input of the CPOP algorithm. The objective is obviously to experiment with the CPOP algorithm. As stated in a previous chapter, our implementation of the CPOP algorithm in Python takes as input a DAG serialized in JSON format; that is the main reason we imported the json module. Finally, we should note an important difference between the personally implemented DAG generator and most of the already existed DAG generators; our generator requires no input parameters to build a weighted random DAG, while most of the other generators require input parameters. Examples of input parameters are the number of nodes (notation: v) and an out-degree of nodes that the DAG generator tries making it the out-degree of as many nodes as possible.



Graph 6.1: Better makespan occurrences (using a personally implemented DAG generator)

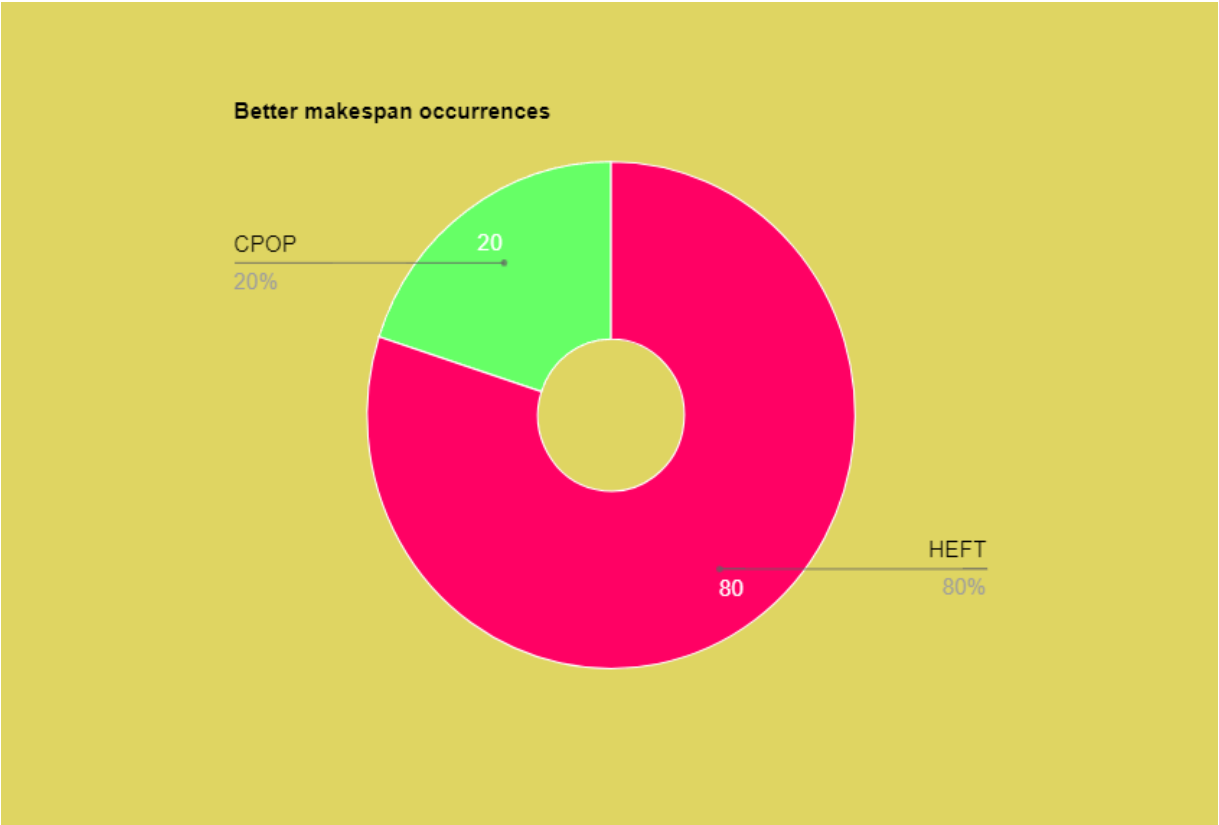
Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

Regarding the performance of the CPOP algorithm, when its input is a weighted random DAG generated by our generator, the CPOP algorithm manages to outperform the HEFT algorithm in terms of the makespan - which is an output of our implementation of the CPOP algorithm - approximately 17 percent of the time (Graph 6.1); this experimental result is honorary for the CPOP algorithm having in mind that the HEFT algorithm is famous for dominating in heterogeneous domains in terms of the makespan.

6.3. Experimenting with a DAG generator that requires input parameters and various metrics

After this initial experiment, we moved towards using an already existed DAG generator that does not construct DAGs as randomly as possible; it requires some input parameters to build a weighted random DAG. While the makespan is a popular metric in task scheduling, we also used a normalized metric known as schedule length ratio (notation: SLR).

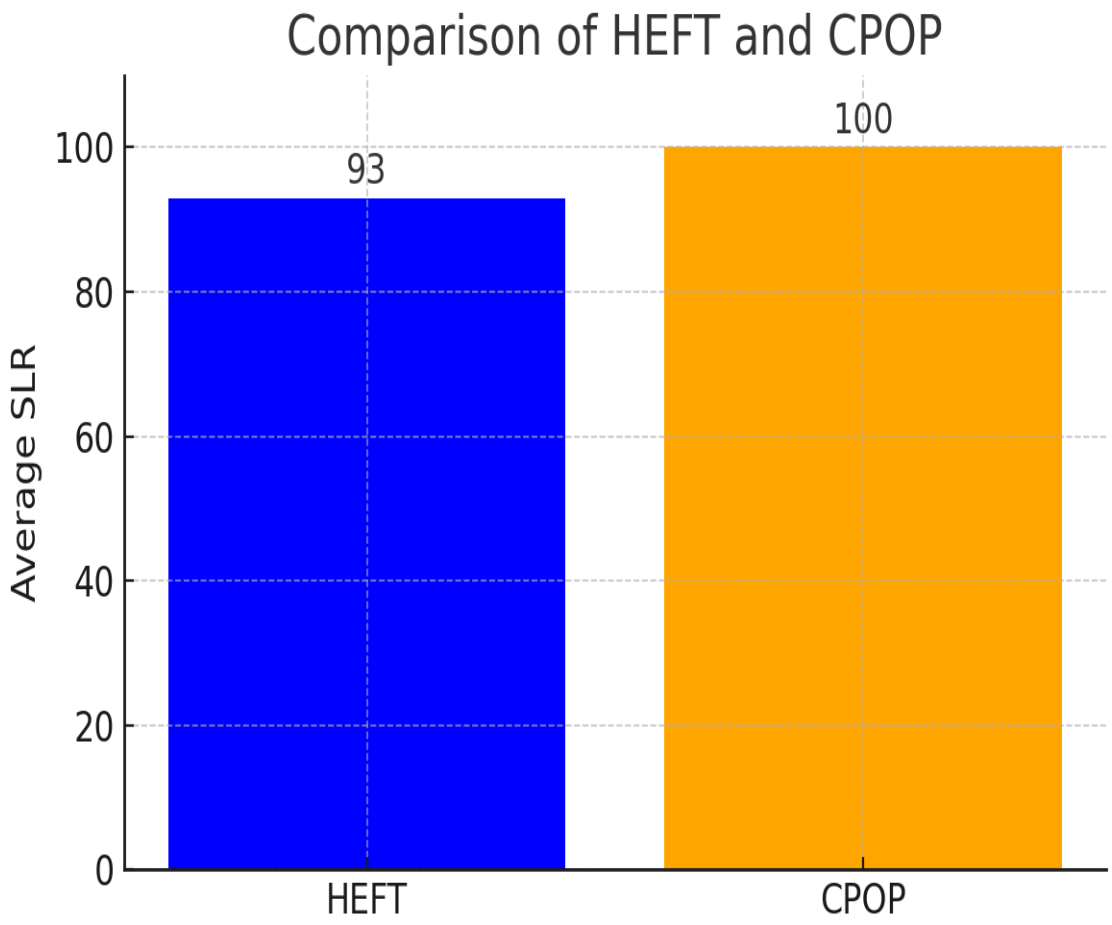
Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250



Graph 6.2: Better makespan occurrences (using a DAG generator that requires input parameters)

In terms of the makespan, the HEFT algorithm is better than the CPOP algorithm approximately 80 percent of the time (Graph 6.2). This experimental result came as no surprise having in mind the result of the initial experiment. In terms of the schedule length ratio, the HEFT algorithm also outperforms the CPOP algorithm; regarding all generated DAGs, the average value of the schedule length ratio of the HEFT algorithm is better than the average value of the schedule length ratio of the CPOP algorithm by 7 percent (Graph 6.3).

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

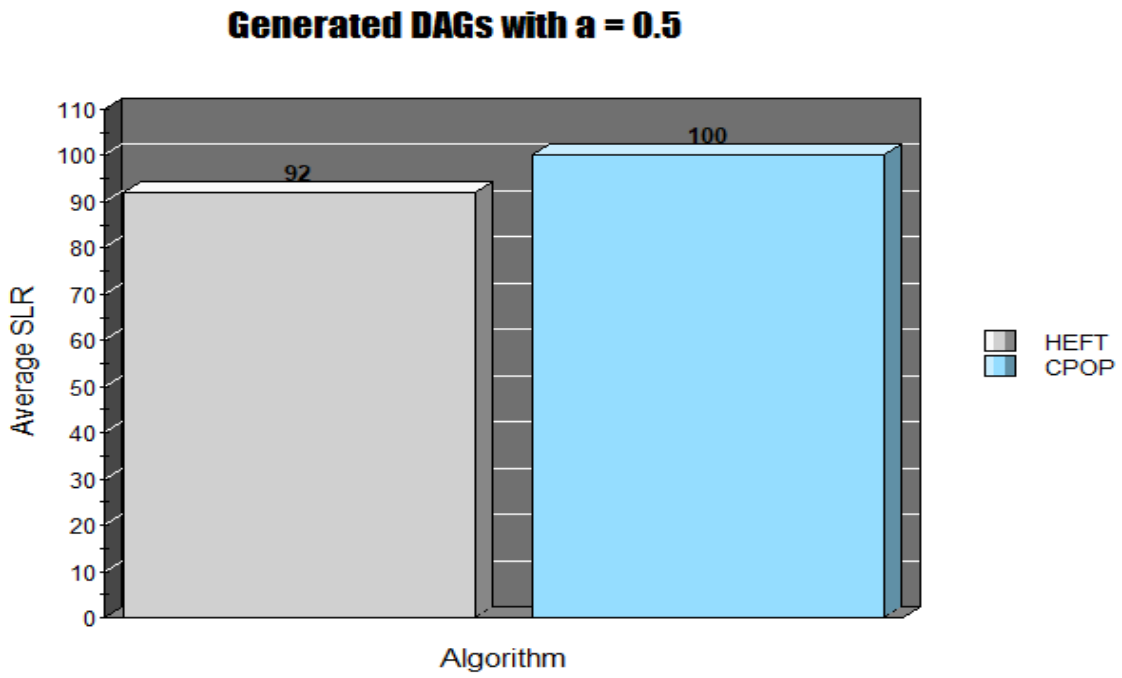


Graph 6.3: Average SLR regarding all generated DAGs

Before continuing with some other experimental results, we should focus on a particular input parameter, the shape parameter (notation: a). Regarding the already existed DAG generator we used, a uniform distribution with mean value equal to $\sqrt{v}/a$ generates randomly the height of the DAG and a uniform distribution with mean value equal to $a\sqrt{v}$ generates randomly the width of each level of the DAG. A short DAG characterized by high parallelism can be generated by selecting a>1, while a long DAG characterized by low parallelism can be generated by selecting a<1. Experimenting with this DAG generator, we assigned the following values to the shape parameter: 0.5, 1 and 2. Firstly, regarding all generated DAGs with a = 0.5, the average value of the schedule length ratio of the HEFT

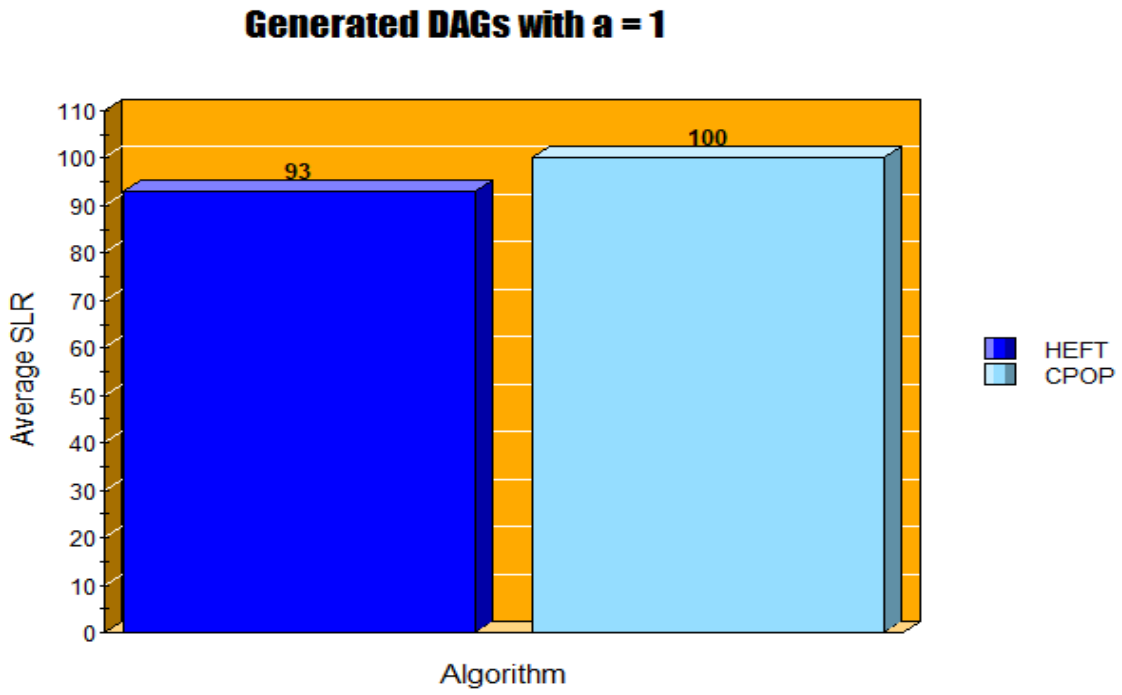
Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

algorithm is better than the average value of the schedule length ratio of the CPOP algorithm by 8 percent (Graph 6.4). Regarding all generated DAGs with a = 1, the average value of the schedule length ratio of the HEFT algorithm is better than the average value of the schedule length ratio of the CPOP algorithm by 7 percent (Graph 6.5). Finally, regarding all generated DAGs with a = 2, the average value of the schedule length ratio of the HEFT algorithm is better than the average value of the schedule length ratio of the CPOP algorithm by 6 percent (Graph 6.6).

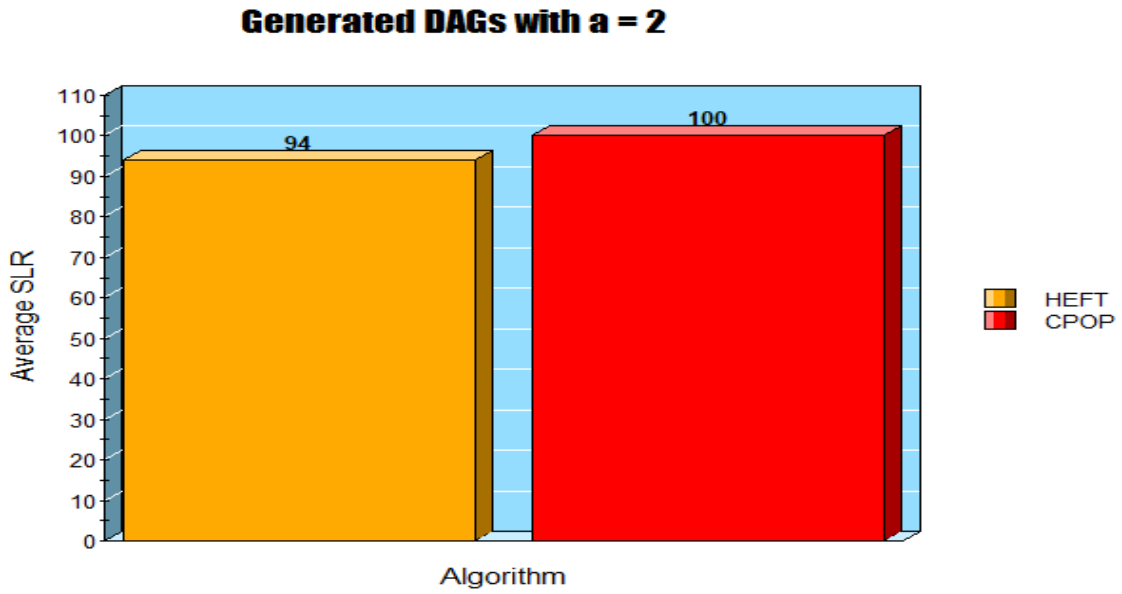


Graph 6.4: Average SLR given a less than 1 shape parameter

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250



Graph 6.5: Average SLR given an equal to 1 shape parameter



Graph 6.6: Average SLR given a greater than 1 shape parameter

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

7. CONCLUSION

7.1. A brief overview

Starting with the implementation of the original CPOP algorithm, it was a very useful first step for this dissertation in order to familiarize with the CPOP algorithm's distinctive phases and the choice of the Python programming language was a good one. Secondly, serialization of DAGs in JSON format was a great tool. Regarding thoughts about making a novel variation at a phase of the CPOP algorithm within the context of this dissertation, they triggered, without doubt, a lot of brain function. However, as stated in a previous chapter, it will denature the CPOP algorithm if we change something in the processor selection phase. The implementation of a DAG generator in Python was such a good idea, as it led to exploration of the networkx package before experimenting with the CPOP algorithm and the HEFT algorithm.

Experimental results (using both the personally implemented DAG generator and an already existed DAG generator) confirmed that the HEFT algorithm performs better than the CPOP algorithm in terms of the makespan approximately 80 percent of the time. We also evaluated the performance of these task scheduling algorithms using the schedule length ratio, an important metric in task scheduling. Finally, we made an analysis of a particular input parameter that most of the already existed DAG generators require, the shape parameter of the DAG; this input parameter not only affects the height of the DAG and the width of each level of the DAG, but also has effect on the difference in performance between the CPOP algorithm and the HEFT algorithm.

7.2. Objective difficulties while experimenting

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

Regarding objective difficulties within the confines of this dissertation, we created a virtual machine that simulates Ubuntu, a famous and useful operating system; Ubuntu is a Linux distribution that would assist us with our experiments. However, due to both technical and temporal reasons, it was not possible to use WfCommons, a very helpful framework when engaging with scientific workflow research and development. Furthermore, the absence of courses related to artificial intelligence and machine learning at any semester of these postgraduate studies hindered further utilization of artificial intelligence regarding the various aspects (and experiments) of this dissertation.

7.3. Future work

As stated in the introduction, there are various ways to continue with this project outside the confines of this dissertation. Obviously, we are supposed to overcome the aforesaid obstacle and use WfCommons; thus, we can take experimentation to the next level utilizing this useful framework. Our future work may also introduce more task scheduling algorithms, more metrics and more experimentation with various input parameters that most of the already existed DAG generators require in order to generate random DAGs. According to these plans, we will achieve greater quality of experimentation and have a larger amount of content in the near future.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

REFERENCES

- [1] H. Topcuoglu, S. Hariri and M. Wu, “Task Scheduling Algorithms for Heterogeneous Processors”, Proceedings of the 1999 8th Heterogeneous Computing Workshop, 1999.
- [2] A. Mazrekaj, A. Sheholli, D. Minarolli and B. Freisleben, “The Experiential Heterogeneous Earliest Finish Time Algorithm for Task Scheduling in Clouds”, Proceedings of the 9th International Conference on Cloud Computing and Services Science, 2019.
- [3] A. Pasdar and T. Ghafarian, “Data Placement Based On Hierarchical Clustering on Scientific Workflows”, 18th Iranian Student Conference on Electrical Engineering, 2015.
- [4] A. Manudhane and A. Wadhe, “Comparative Study of Static Task Scheduling Algorithms for Heterogeneous Systems”, International Journal on Computer Science and Engineering - Volume 5, 2013.
- [5] K. Dubey, M. Kumar and S. Scharma, “Modified HEFT Algorithm for Task Scheduling in Cloud Environment”, 6th International Conference on Smart Computing and Communications, 2017.
- [6] H. Arabnejad, “List Based Task Scheduling Algorithms on Heterogeneous Systems - An overview”, Proceedings of the 7th Doctoral Symposium in Informatics Engineering, 2012.
- [7] Y. Kwok and I. Ahmad, “Benchmarking the Task Graph Scheduling Algorithms”, Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, 1998.
- [8] A. Radulescu, A. Gemund and H. Lin, “LLB: A Fast and Effective Scheduling Algorithm for Distributed-Memory Systems”, Proceedings of the Second Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, 1999.
- [9] J. Elcock and N. Edward, “An efficient ACO-based algorithm for task scheduling in heterogeneous multiprocessing environments”, Array - Volume 17, 2023.
- [10] E. Hou, N. Ansari and H. Ren, “A Genetic Algorithm for Multiprocessor Scheduling”, IEEE Transactions on Parallel and Distributed Systems - Volume 5, 1994.
- [11] H. Singh and A. Youssef, “Mapping and Scheduling Heterogeneous Task Graphs using Genetic Algorithms”, Proceedings of the 1996 5th Heterogeneous Computing Workshop, 1996.
- [12] Z. Abadi and N. Mansouri, “A comprehensive survey on scheduling algorithms using fuzzy systems in distributed environments”, Artificial Intelligence Review - Volume 57, 2024.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

- [13] S. Abrishami, M. Naghibzadeh and D. Epema, “Deadline-constrained workflow scheduling algorithms for Infrastructures as a Service Clouds”, Future Generation Computer Systems - Volume 29, 2013.
- [14] M. Menaka and K. Sendhil Kumar, “Workflow scheduling in cloud environment - Challenges, tools, limitations & methodologies: A review”, Measurement: Sensors - Volume 24, 2022.
- [15] Y. Kotb, I. Ridhawi, M. Aloqaily, T. Baker, Y. Jararweh and H. Tawfik, “Cloud-Based Multi-Agent Cooperation for IoT Devices Using Workflow-Nets”, Journal of Grid Computing, 2019.
- [16] J. Sujana, T. Revathi, T. Priya and K. Muneeswaran, “Smart PSO-Based Secured Scheduling Approaches for Scientific Workflows in Cloud Computing”, Soft Computing - A Fusion of Foundations, Methodologies and Applications - Volume 23, 2019.
- [17] N. Tziritas, K. Kolomvatsos, P. Oikonomou, G. Stamoulis, T. Loukopoulos and G. Theodoropoulos, “A Fuzzy Logic Controller for Tasks Scheduling Using Unreliable Cloud Resources”, IEEE International Symposium on Network Computing and Applications, 2020.
- [18] B. Xiang, B. Zhang and L. Zhang, “Greedy-Ant: Ant Colony System-Inspired Workflow Scheduling for Heterogeneous Computing”, IEEE Access - Volume 5, 2017.
- [19] M. Masdari, S. ValiKardan, Z. Shahi and S. Azar, “Towards workflow scheduling in cloud computing: a comprehensive analysis”, Journal of Network and Computer Applications - Volume 66, 2016.
- [20] Z. Ahmad, A. Jehangiri, M. Ala’anzy, M. Othman, R. Latip, S. Zaman and A. Umar, “Scientific Workflows Management and Scheduling in Cloud Computing: Taxonomy, Prospects, and Challenges”, IEEE Access – Volume 9, 2021.
- [21] F. Yao, C. Pu and Z. Zhang, “Task Duplication-Based Scheduling Algorithm for Budget-Constrained Workflows in Cloud Computing”, IEEE Access – Volume 9, 2021.
- [22] Y. Kwok and I. Ahmad, “A New Approach to Scheduling Parallel Programs Using Task Duplication”, Proceedings of the 1996 5th Heterogeneous Computing Workshop, 1996.
- [23] B. Kruatrachue and T. Lewis, “Grain Size Determination for Parallel Processing”, IEEE Software - Volume 5, 1988.
- [24] G. Park, B. Shirazi and J. Marquis, “DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems”, Proceedings of the 11th International Parallel Processing Symposium, 1997.
- [25] J. Liou and M. Palis, “A Comparison of General Approaches to Multiprocessor Scheduling”, Proceedings of the 11th International Parallel Processing Symposium, 1997.
- [26] M. Wu and D. Gajski, “Hypertool: A Programming Aid for Message Passing Systems”, IEEE Transactions on Parallel and Distributed Systems - Volume 1, 1990.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

- [27] T. Yang and A. Gerasoulis, “DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors”, IEEE Transactions on Parallel and Distributed Systems - Volume 5, 1994.
- [28] S. Kim and J. Browne, “A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures”, Proceedings of the International Conference on Parallel Processing, 1988.
- [29] T. Yang and A. Gerasoulis, “A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors”, Journal of Parallel and Distributed Computing - Volume 16, 1992.
- [30] D. Wei, J. Liou and M. Palis, “Task Clustering and Scheduling for Distributed Memory Parallel Architectures”, IEEE Transactions on Parallel and Distributed Systems - Volume 7, 1996.
- [31] W. Chen and J. Zhang, “An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem with Various QoS Requirements”, IEEE Transactions on Systems - Volume 39, 2009.
- [32] H. Arabnejad and J. Barbosa, “List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table”, IEEE Transactions on Parallel and Distributed Systems - Volume 25, 2014.
- [33] Y. Kwok and I. Ahmad, “Static scheduling algorithms for allocating directed task graphs to multiprocessors”, ACM Computing Surveys - Volume 31, 1999.
- [34] H. Arabnejad and J. Barbosa, “Performance Evaluation of List Based Scheduling on Heterogeneous Systems”, Euro-Par 2011: Parallel Processing Workshops, 2011.
- [35] Y. Cui and Z. Xiaoqing, “Workflow Tasks Scheduling Optimization Based on Genetic Algorithm in Clouds”, 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis, 2018.
- [36] E. Ilavarasan and P. Thambidurai, “Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments”, Journal of Computer Sciences - Volume 3, 2007.
- [37] C. Boeres, J. Filho and V. Rebello, “A cluster-based strategy for scheduling task on heterogeneous processors”, Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing, 2004.
- [38] Y. Kwok and I. Ahmad, “On exploiting task duplication in parallel program scheduling”, IEEE Transactions on Parallel and Distributed Systems - Volume 9, 1998.
- [39] S. Basker and P. SaiRanga, “Scheduling Directed A-cyclic Task Graphs on Heterogeneous Network of Workstations to Minimize Schedule Length”, Proceedings of the 2003 International Conference on Parallel Processing Workshops, 2003.
- [40] R. Bajaj and D. Agrawal, “Improving scheduling of tasks in a heterogeneous environment”, IEEE Transactions on Parallel and Distributed Systems - Volume 15, 2004.
- [41] H. El-Rewini and T. Lewis, “Scheduling Parallel Program Tasks onto Arbitrary Target Machines”, Journal of Parallel and Distributed Computing - Volume 9, 1990.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250

- [42] G. Sih and E. Lee, “A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architecture”, IEEE Transactions on Parallel and Distributed Systems - Volume 4, 1993.
- [43] H. Oh and S. Ha, “A Static Scheduling Heuristic for Heterogeneous Processors”, Euro-Par 1996: Parallel Processing Workshops, 1996.
- [44] A. Radulescu and A. Gemund, “Fast and Effective Task Scheduling in Heterogeneous Systems”, Proceedings of the 2000 9th Heterogeneous Computing Workshop, 2000.
- [45] P. Bailin, W. Yanping, L. Hanxi and Q. Jie, “Task Scheduling and Resource Allocation of Cloud Computing Based on QoS”, Advanced Materials Research - Volumes 915-916, 2014.
- [46] L. Singh and S. Singh, “A Survey of Workflow Scheduling Algorithms and Research Issues”, International Journal of Computer Applications - Volume 74, 2013.
- [47] R. Sakellariou and H. Zhao, “Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems”, Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004.
- [48] H. Topcuoglu, S. Hariri and M. Wu, “Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing”, IEEE Transactions on Parallel and Distributed Systems - Volume 13, 2002.
- [49] R. Sakellariou and H. Zhao, “Scheduling Multiple DAGs onto Heterogeneous Systems”, Proceedings of the 20th International Parallel and Distributed Processing Symposium, 2006.
- [50] J. Ullman, “NP-complete scheduling problems”, Journal of Computer and System Sciences - Volume 10, 1975.
- [51] M. Melnik and D. Nasonov, “Workflow scheduling using Neural Networks and Reinforcement Learning”, 8th International Young Scientist Conference on Computational Science, 2019.
- [52] C. Jin, Y. Han, Z. Deng, Y. Chen, C. Liu and J. Huang, “Reinforcement Learning-Based Intelligent Task Scheduling for Large-Scale IoT Systems”, Explainable Artificial Intelligence for Next Generation Internet of Things, 2023.
- [53] N. George, K. Chandrasekaran and A. Binu, “An Objective Study on Improvement of Task Scheduling Mechanism Using Computational Intelligence in Cloud Computing”, 2015 IEEE International Conference on Computational Intelligence and Computing Research, 2015.
- [54] W. Shen, W. Lin, W. Wu, H. Wu and K. Li, “Reinforcement Learning-based Task Scheduling for Heterogeneous Computing in End-Edge-Cloud Environment”, Research Square, 2024.
- [55] N. Khaledian, M. Voelp, S. Azizi and M. Shirvani, “AI-based & heuristic workflow scheduling in cloud and fog computing: a systematic review”, Cluster Computing - Volume 27, 2024.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 03:03:41 EEST - 216.73.216.250