symmetry

MDPI

Article

Controlling the Difficulty of Combinatorial Optimization Problems for Fair Proof-of-Useful-Work-Based Blockchain Consensus Protocol

Uroš Maleš [1], Dušan Ramljak [2], Tatjana Jakšić Krüger [3], Tatjana Davidović [3]*, Dragutin Ostojić [4] and Abhay Haridas [2]

[1] Faculty of Electrical Engineering, University of Belgrade, 11000 Belgrade, Serbia
[2] School of Professional Graduate Studies at Great Valley, The Pennsylvania State University, Malvern, PA 19355, USA
[3] Mathematical Institute, Serbian Academy of Sciences and Arts, 11001 Belgrade, Serbia
[4] Faculty of Science, Department of Mathematics and Informatics, University of Kragujevac, 34000 Kragujevac, Serbia
* Correspondence: tanjad@mi.sanu.ac.rs; Tel.: +381-11-2630-170

Abstract: The wide range of Blockchain (BC) applications and BC's ubiquity come from the fact that BC, as a collection of records linked to each other, is strongly resistant to alteration, protected using cryptography, and maintained autonomously. All these benefits come with a cost, which in BC is expressed by a very high use of energy needed to execute consensus protocols. Traditionally, consensus protocols based on Proof-of-Work (PoW) ensure fairness, but are not very useful. The paradigm proposed in the recent literature, known as Proof-of-Useful-Work (PoUW), assumes the completion of additional useful work for the same amount of resources (energy) used. However, the majority of the proposed PoUW approaches do not adequately consider fairness in balancing and controlling the difficulty of the work miners need to perform. A minority of the studies that do address fairness in miners' work utilize PoW as a tool to ensure it. Therefore, a general framework to provide a structure for understanding the difficulty of useful work and how it can be used to fine-tune the complexity of miners' effort in PoUW-based consensus protocols is proposed in this paper. The main characteristic of the proposed framework is that controlling the difficulty and fairness of miners' work in PoUW-based consensus protocols is achieved exclusively through the useful work. The modules of the framework are discussed, and many research challenges and opportunities are articulated. The benefits of the proposed approach are illustrated taking as an example two optimization algorithms for a variant of the scheduling problem. In addition, the steps that should be taken to make this general framework applicable to any PoUW-based consensus protocols are identified.

Keywords: balancing workload; energy efficiency; machine learning; heuristic optimization; distributed computing

Citation: Maleš, U.; Ramljak, D.; Jakšić Krüger, T.; Davidović, T.; Ostojić, D.; Abhay, H. Controlling the Difficulty of Combinatorial Optimization Problems for Fair Proof-of-Useful-Work-Based Blockchain Consensus Protocol. *Symmetry* **2023**, *15*, 140. https://doi.org/10.3390/sym15010140

Academic Editors: Chin-Ling Chen and Sergei D. Odintsov

Received: 31 October 2022
Revised: 27 November 2022
Accepted: 6 December 2022
Published: 3 January 2023

1. Introduction

Blockchain (BC) consensus protocols based on the Proof-of-Useful-Work (PoUW) paradigm [1–4] were developed to ensure the efficient exploration of resources. They capitalize on the effort of BC participants, called miners, who solve problems useful to some other participants or the community in general. For example, useful work considered in [1–8], consists of solving the real-life instances of some Combinatorial Optimization (CO) problems. At the same time, PoUW-based consensus protocols should provide security, consistency, reliability, and immutability of the maintained transaction ledger. A common concern for these approaches is the inability to control the difficulty of the said useful work and the inevitable impact of this issue on the system's fairness and security. The fairness in

balancing miners' work is related to providing miners with equal chances to add blocks in a timely manner (also called liveness), rather than guaranteeing that miners would spend an equal amount of time mining the blocks.

There are two main reasons to keep very similar the complexities of all work given to miners (expressed by the expected instance solving times). The first is to ensure fairness in rewarding miners. For example, if some of the submitted CO instances are too easy, miners who solved them would gain an unfair advantage. At the same time, some of the CO instances are never solved. The second reason is to accommodate for similar Block Insertion Times (BITs) depending on the complexity of the performed useful work. Some of the submitted CO problem instances may be quite difficult, which could result in miners not being able to solve them efficiently. As a consequence, latency may appear in the PoUW consensus protocols.

The difficulty controlling is not the problem-specific to PoUW-based consensus protocols. A special parameter called *difficulty target* (*difficulty* for short) has been introduced in BC systems based on the traditional Proof-of-Work (PoW) to control the BITs [9,10]. In PoW-based BCs, the BITs depend on two parameters: the hashrate of the entire network and the *difficulty* of the current block. The hashrate parameter measures the number of calculations that can be performed per second. It depends on the computational power currently available in the BC system, i.e., on the number of active miners. To keep the BIT stable as the number of miners changes, it is important for the *difficulty* to correspond to the total hashrate. Therefore, the difficulty adjustment refers to changing the value of the *difficulty* parameter with respect to the change of the number of miners.

There are two main differences between the difficulty adjustment in PoW and controlling the difficulty in PoUW. The fairness of rewarding miners is explicitly taken into consideration in PoUW-based BC systems, while it is implicitly assumed in PoW. The difficulty adjustment in PoW is closely related to the number of active miners, while in PoUW, controlling the difficulty of the workload of each miner corresponds only to BIT. On the other hand, the workload of miners has to be balanced throughout the whole BC system's lifetime in both types of consensus protocols. Additionally, the robustness to the luck of some miners in solving the cryptographic puzzle should match the robustness of mining in PoUW-based BC systems with respect to the difficulty of the CO problem instances. Therefore, we believe that the treatment of difficulty in the two mentioned BC systems is equally important and should be addressed adequately. Our main goals in this paper are to contribute to controlling the difficulty of miners' work in PoUW-based consensus protocols and to ensure that fairness is achieved exclusively through the useful work.

The early approaches to controlling the difficulty of miners' work proposed in the literature were usually based on the size of the CO problem instances needed to be solved [6–8]. However, it is well known that the size of the instance does not always directly correspond to the runtime of the selected optimization algorithm [11,12]. Therefore, some of the later approaches included a request to provide the proof that a certain amount of work had been invested in the instance solving process [2,3,13–15]. The proof was based on the PoW concept, i.e., the miners were requested to solve both a CO problem instance and a cryptographic puzzle. The motivation for our study is our belief that fairness can be controlled more efficiently without solving cryptographic puzzles, i.e., that there is no reason to apply PoW in any part of PoUW-based consensus protocols.

Our main contributions are the development of a general framework for difficulty estimation, a case study simulation, and a discussion of the applicability of the proposed framework to any PoUW-based consensus protocol. The proposed framework considers the issue of controlling and balancing the work of miners with the aims to ensure the fairness in their reward and to keep the BIT stable. We address the ways to estimate the computational effort invested in solving CO problem instances using the modified CRoss-Industry Standard Process for Data Mining (CRISP-DM) model [16], Artificial Intelligence (AI), Machine Learning (ML), and data science methods. After the estimation, our framework also balances the amount of work for miners by grouping CO problem instances into

equally difficult work elements executed by the miners. The novelty of our approach with respect to the existing methodology is the development of a framework that controls the difficulty of miners' work and ensures fairness in their reward exclusively through the useful work.

The proposed difficulty estimation procedure is considered to be general because it can be applied to any CO problem, any optimization method, and any set of the corresponding problem instances. In the relevant literature, the difficulty estimation has been performed for various CO problems: propositional Satisfiability (SAT), the Traveling Salesmen Problem (TSP), the Maximum Flow Problem (MFP), the Job Shop Scheduling Problem (JSSP), 0-1 Knapsack, the Strip Packing Problem (SPP), etc. [12,17–20]. To the best of our knowledge, the difficulty estimation of the $P||C_{max}$ scheduling problem [21] instances in the general case has not been considered so far. Therefore, we selected this problem as a case study simulation performed for two optimization methods. More precisely, the proposed difficulty estimation model is applied to predict the execution time of the ArcFlow exact solver [22] and the Greedy Iterative Stochastic Transformation (GIST) heuristic proposed in [23] on benchmark instances for $P||C_{max}$. As an illustrative example, the estimated values for 20 randomly selected instances are used in the grouping module to pack instances into equally sized bins representing the workload of the miners.

Finally, the proposed framework could easily be adapted to address controlling the balance of the miners' work in any consensus protocol that requires the miners to perform useful work. Especially, PoUW-based consensus protocols that apply PoW in part could be modified to adjust the difficulty with our framework and, thus, make all components of their framework useful, as well as fair and stable. Additionally, the learning process of the developed difficulty estimation procedure could be executed online and within PoUW consensus protocol, yielding a self-contained BC system.

This paper is organized in the following manner. In Section 2, we explain the BC users' activities and present a brief survey of papers to explain the evolution from the difficulty adjustment in PoW to controlling the difficulty in PoUW and papers considering the difficulty estimation of CO problems. A description of the methodology for the proposed difficulty controlling framework is presented in Section 3. The performed case studies are described in Section 4. In Section 5, the takeaways from the case studies and the discussion of the proposed solution framework are provided. Finally, Section 6 concludes the paper with a summary of the contributions and suggestions for future research.

2. Related Work

In this section we briefly survey the BC consensus protocols literature, focusing on papers that consider controlling the difficulty of miners' workload. In addition, we examine how to achieve difficulty control for CO problems.

2.1. Controlling the Difficulty in PoW and PoUW Protocols from the Literature

To support autonomy of the BC systems and their management based on the democratic principles, participants take different roles, shown in Figure 1. *Basic users* are participants that want to store some data (in the form of transactions) in BC. Occasionally, they submit these data to the transaction pool and wait for it to be included in one of the forthcoming blocks. One of the most important role is the role of *miners*, who compose blocks out of transactions, perform some predefined tasks and, if successful, announce the composed block. The miner whose block is appended to the BC gets reward for the performed mining process. All announced blocks have to pass the verification phase, performed by the participants who choose the role of *verifiers*. Verifiers select one of the announced blocks, check its validity and, if the block is valid (correct), they support its insertion in BC. Otherwise, they announce that the block is invalid and should be discarded from the BC system. *Clients* do not exist in the PoW-based consensus protocols, but it can be assumed that BC system plays a role of a client. Specifically, as the difficulty of the miners' work needs to be adjusted to ensure a fair rewarding scheme, the client/system adjusts the

difficulty of the cryptographic puzzle for miners to solve. In case of PoUW-based consensus protocols, clients submit tasks as the useful work that miners need to solve, while the system should control the difficulty of the miners' work composed of these tasks. To better explain the evolution from the difficulty adjustment to controlling the difficulty, we divided this section into parts that focus on the difficulty adjustment, utilization of AI, ML and Deep Learning (DL) as a useful work, and PoUW-based protocols that utilize combinatorial optimization and could directly benefit from the framework proposed in this work.

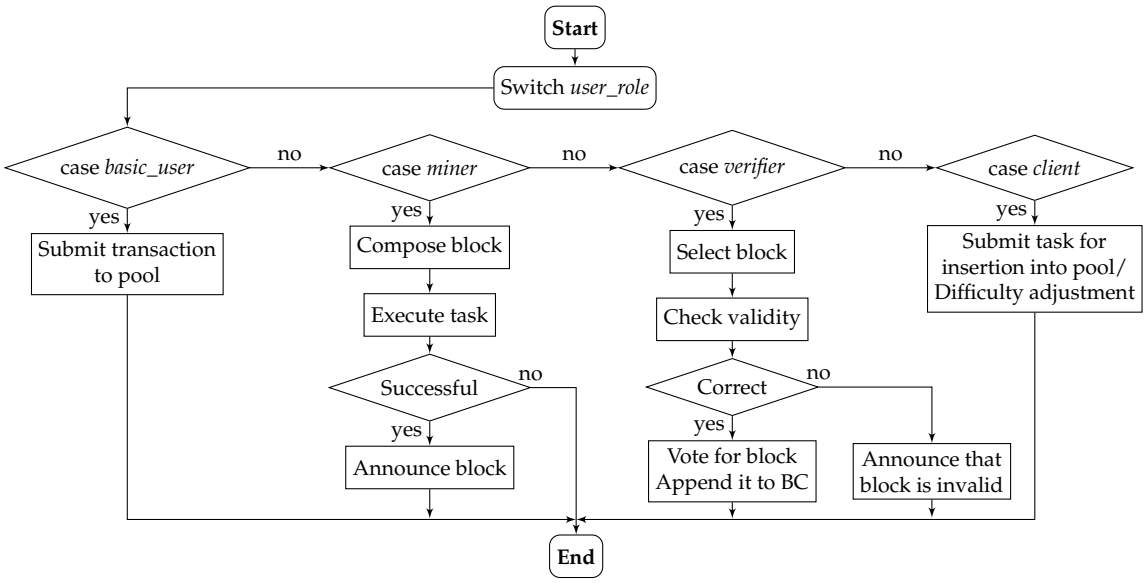

Figure 1. Flow-chart of BC users activities.

2.1.1. PoW Difficulty Adjustment

Classical PoW, which uses computational puzzles defined in [24], is proposed in [25] along with a difficulty update procedure. That procedure was established under the assumption that honest participants possess the biggest portion of computing power. A formal proof that PoW satisfies consistency and liveness is established in [26,27] with the additional assumptions. The first is that honest participants control the computational power. The second assumption states that the difficulty is expressed by a function that depends on both the maximum delay of messages in the network and the network's computational power, and finally that a random oracle is used to model the computational puzzle.

Difficulty adjustment in PoW-based BC systems is necessary to stabilize mean latency between blocks and make it robust to changes in available computational power (utilized for solving the puzzles) caused by participants joining and leaving the system. The authors in [28] proposed the difficulty adjustment procedure to address two problems. The first is the significant variance in computational power. The second is the prevention of coin-hopping strategy, i.e., malicious uses of the changes in difficulty. An economic paradigm for PoW-based BC environment was proposed in [29]. The authors claim that instability of the difficulty adjustment algorithms works in favor of miners by increasing their profits. Additionally, the authors in [30] explore financial incentives of the difficulty adjustment algorithms and further confirm their influence on the miners' profits. The authors of [31] proposed a general difficulty control algorithm and discuss the difficulty adjustment guidelines for PoW-based BCs. Their algorithm can maintain fast-updating and low volatility demands for the difficulty adjustment. In addition, it is capable of identifying an anomaly and handle abnormal cases. A prediction-based difficulty control algorithm was developed by the authors in [32] to track the hashrate of a network and to adjust the difficulty.

The importance of the difficulty adjustment and a good literature overview of the methods in the PoW consensus protocol that address them along with a Non-dominated Sorting Genetic Algorithm II (NSGA-II) to improve those methods are presented in [33]. The authors wanted to make sure that the BC could adapt quickly to an unexpected event, like a significant drop or surge in hashrate, by modifying the difficulty and BIT. Using NSGA-II to optimize both the BIT and the adjustment of the difficulty interval can help BC to achieve a lower standard deviation of those measures compared to the default BC network without NSGA-II.

2.1.2. PoUW-Based Protocols that Utilize Difficulty Adjustment

The problem of controlling the difficulty of miners' work appears also in the PoUW-based consensus protocols [1–5], known in the literature as an energy efficient alternative to PoW. Some of the early attempts to implement PoUW-based consensus protocols involve [6–8]. The authors consider solving NP-hard optimization problems and illustrate their approaches on the well known TSP.

Wei Li [6] was among the first authors who tried to replace the cryptographic puzzle of PoW with hard optimization problems. Considering the problem of controlling the difficulty of miners' work, he suggested the utilization of the decomposition strategy. For example in TSP, the local optimization phase tries to improve only k_{TSP} coordinates (locations, nodes) while the remaining $n_{TSP} - k_{TSP}$ are fixed. The justification of the proposed approach lies in the fact the TSP represents a NP-hard problem whose complexity grows exponentially with the increase in the number of nodes n_{TSP}. The hash value of the block is used to determine which k_{TSP} nodes should be optimized and to establish the connection between the block and the sub-problem miners need to solve. The main idea related to balancing miners' work is that all miners need to solve sub-problems of the same size. That should directly correspond to the PoW case when all miners have to solve the cryptographic puzzle of the same difficulty, i.e., to find the nonce corresponding to the same number of zeros in the resulting hash value. However, it is well known that the size of instances is not the only criterion for determining their difficulty [34]. Therefore, fixing the value of k_{TSP} may not be enough to achieve fairness in balancing miners' work.

In the hybrid scheme developed in [7], the authors combine PoW with solving CO problem instances. The first stage of this hybrid scheme, consists of solving cryptographic puzzle for a given amount of time. In the second stage, an artificial instance of TSP is generated using the reported nonce values to determine the coordinates of locations. With respect to difficulty controlling, the authors limit the number of locations to $n_{TSP} = 5000$ that corresponds to a 4 min solution time of Concorde exact solver [35] estimated on the well-known TSPLIB instances. Besides the questionable usefulness in solving artificial TSP instances, the fairness in balancing miners' work should also be under suspicion due to the fact that the instances of the same size can significantly differ in difficulty.

Another paper involving TSP in the PoUW-based scheme is [8]. To add a new block to BC, a miner has to find the new best solution of the considered TSP instance. The difficulty of the problem is controlled by adding new cities to TSP once the sub-instance of the problem is solved. Specifically, the algorithm starts with a hard TSP instance with N_{TSP} locations that represents the goal of miners' useful work. The solution of this instance is obtained by an iterative process involving the random selection of some small enough subsets of n_{TSP} locations ($n_{TSP} \ll N_{TSP}$), for which the applied solver can find good enough solutions. "Good enough" actually means that the objective function value is smaller than a given threshold T_1. The new block requires an extension of the current sub-problem by a randomly selected new location from the initial TSP instance. The hash value of the previous block and TSP input data influence the selection of a new location. As the size of instances does not uniquely determine their difficulty, it is not clear if the fairness in balancing miners' work is achieved.

The approach proposed in [5] guarantees balanced work between miners as they are always working on the same clustering instance. The authors considered a BC system

applied in maritime transport to store the information about the transportation requests. As the useful work in block mining process, a set of transportation requests that share origin and destination is identified and clustered in such a way that each cluster is to be transported by a single vessel and the transportation cost is minimized. The corresponding cost savings are used to reward the miner who provided the best solution of the clustering problem. All transportation requests included in the problem instances, together with the best obtained solution, are kept in a block's header to enable block verification process and to serve as a connection between the clustering problem instance and the block itself. The authors claim that the instance difficulty, and consequently, the BIT, can be controlled by the number of transportation requests. Here again, we have to comment that the size of CO problem instance is not the only characteristic that influences its difficulty. However, in this case, the solution time impacts only the block insertion frequency, while the fairness in balancing miners' work is similar to the PoW case.

2.1.3. PoUW-Based Protocols Considering AI, ML, and DL

To the best of our knowledge, only several papers describing the development of new PoUW-based consensus protocols that involve training of ML and DL models consider difficulty adjustment or estimation [13–15,36,37].

In the PoUW-based on training distributed DL models [13], a block is mined only when the performance of these models exceeds a threshold. The authors successfully addressed neural architecture search, a technique for automating DL design. However, DL models are complex and follow non-linear stochastic behaviors. Thus, it is possible that controlling the learning threshold leads to unsolvable problems and prevents finishing the mining process. Their solution to unsolvability involves raising the threshold after each block is mined, and then lowering it at a slower pace as time goes on. This approach might ensure the continuation of mining process but does not guarantee the fairness.

In [14], the authors introduced a PoUW consensus protocol that satisfies PoW properties and offers an additional reward by clients for the useful work miners had performed. However, miners need to do both the useful work training ML models and PoW with a small difficulty. Moreover, only one BIT variant (10 min) is used and it is not clear how the protocol could be used with a smaller BIT.

Deep Learning-Based Consensus protocol (DLBC), another variant of PoUW, is considered in [15]. The useful work performed by miners consists of training DL models. The miner whose model is verified to have the best performance is allowed to insert a block. The developed mechanism handles multiple DL tasks, larger model, and training datasets in comparison with other similar approaches in the literature. The authors introduce a ranking mechanism that considers tasks difficulty, but it is not clear how or why they include model complexity, network burden, data size, and queue length. Moreover, the authors did not provide analysis of the mechanism's fairness and it stays unclear whether it is achieved.

Proof-of-Learning (PoLe) introduced in [36,37], similarly to PoUW-based consensus protocols, leverages the DL training as the useful work. The PoLe-based BC systems contain two kinds of participants: *Data* node, which announces a task with a reward, and the *consensus* node, which can accept the task and seek a model that meets the required training accuracy. When a valid block is generated by a user, it gets the reward supplied by the *data* node for training and a fixed reward for adding a block. The *consensus* nodes choose solutions according to their performance and distribute the reward correspondingly after the *data* node has released the test set. The authors introduced a secure mapping layer implemented as a linear neural network to link adjacent blocks together. The Secure Media Library encryption was found to detect tampering behavior without affecting the model performance significantly. Even though PoLe is capable of producing a reliable stream of blocks, it is not clear whether controlling difficulty is possible and whether this protocol could be applied for any BIT values.

2.1.4. Controlling the Difficulty in PoUW

In the PoUW consensus protocol proposed in [1], hard optimization problems are considered, especially those that can be represented by Orthogonal Vectors. To control the difficulty of miner's work, the authors strongly rely on the theoretically proven complexity of the Orthogonal Vectors scheme.

Another PoUW considering optimization problems, named the Proof-of-Search (PoS), is proposed in [2]. The newly introduced BC participants, called clients, submit instances of some optimization problem, the corresponding optimization algorithms, and an evaluation function used to calculate the objective function value. To add a new block to the BC, miners need to provide high quality solution for some of the submitted CO problem instances. The validation of a miner's hard work is performed in the following way. Each visited solution is coded and used as a nonce in the classical PoW sense. The optimization algorithm is executed until a coded solution corresponds to a proper nonce (yielding a given number of zeros at the beginning of the block hash value). In such a way, it can be guaranteed that the miner visited an adequate number of candidate solutions and performed hard enough work when mining the composed block.

Similar approach was applied in [3] for the PoUW-based consensus protocol, referred to as Ofelimos. The authors explore a general purpose stochastic local-search algorithm, known as Doubly Parallel Local Search (DPLS). Controlling mining difficulty in Ofelimos is performed by using the classical nonce value as a seed for a random number generator in DPLS. After obtaining the result of DPLS, a miner calculates the hash value of the block including nonce. If the resulting hash value of the block is below the threshold, defined by the current mining difficulty, the miner can announce the composed block. Otherwise, the mining process continues.

The approaches proposed in [2,3] are easily implementable, straightforward methods for controlling difficulty of miners' work. However, they do not take into account the difficulty of CO problem instances. Specifically, the size of an instance is not the only measure of its difficulty, i.e., the increase of instance dimension does not necessarily yield the increase of the time required to obtain the desired solution [34]. On the other hand, requiring that for each instance a certain number of solutions are to be visited in order to prove the amount of invested work, may decrease the usefulness of the mining process. More precisely, it may happen that, for some instances, high quality solutions are obtained very fast. In that case, a miner is forced to examine more solutions than necessary, maybe even to examine the same solutions multiple times, and that is highly counter productive.

In a most recent work in the literature [4], the authors also proposed a PoUW-based consensus protocol, named CO Consensus Protocol (COCP). The main advantage of COCP is the efficient utilization of computing resources by providing valid solutions for the real-life instances of any CO problem. In addition to basic users, miners, and verifiers, COCP also involves clients, the participants of a new type. Clients can be companies, organizations or even individuals, and they join the BC system for its software and hardware resources needed to solve their real-life CO problem instances. Submitted and yet unsolved CO problem instances obtain a unique identification and are stored in the instance pool. After composing a block, the miner finds corresponding unsolved instance in the instance pool and begins the solution process by executing an available optimization algorithm. If a valid solution is obtained, a miner can publish the created block. Solution validity is determined by a threshold that client provided together with the instance input data. Given that multiple miners can work on the same instance, the best solution is provided to the client. The miners that add a new block are rewarded both for adding a new block, and by the client for solving their CO problem instance. Those who do not provide new blocks but solve CO instances receive rewards from the clients as well.

Regarding the control of mining difficulty, in [4] the authors considered three avenues. They put an emphasis on the COCP usefulness stating that for the clients both hard and easy instances are equally important. On the other hand, in COCP it is enough to obtain a valid solution (the one with the objective function value better than a given threshold)

and it is a reasonable assumption that these solutions are easier to obtain. Finally, the authors proposed to use the stopping criterion for optimization algorithms as a mechanism to prove that a certain amount of work is invested in the mining process. The usual stopping criterion is the maximum CPU time, which should correspond to the BIT. Another well-known stopping criterion is the number of objective function evaluations that could correspond to the difficulty controlling approach from [2,3].

Judging by the presented literature review, it is evident that ensuring fairness in balancing miners' work is a very important issue in BC systems maintenance. However, it is not adequately treated in the relevant literature, especially if one strives to implement a useful, secure, and efficient PoUW consensus protocol. Therefore, we propose a systematic framework that includes difficulty estimation and grouping of tasks, which should help to increase fairness of the mining process.

2.2. Difficulty Estimation for CO Problem Instances

The CO community explores difficulty estimation for the construction of comprehensive and unbiased benchmark test suites for algorithm testing. Recent research has shown that the difficulty value of CO problem instances differs with respect to the chosen problem-specific instances features. Moreover, different algorithms may explore different regions of the solution space. ML can be utilized to estimate the runtime of algorithms applied to hard CO problems [11]. A comprehensive review of different instance features and models for SAT, TSP, and Mixed Integer Programming (MIP) problems can be found in [34]. The authors introduced new modeling techniques and features for the mentioned instance-specific models in order to better predict runtime performance.

In [12], the need for problem understanding and reasonable feature extraction is demonstrated through multiple problems (Knapsack, Bin Packing, Graph Coloring, TSP, Timetabling) and their corresponding solving algorithms, e.g., feature indicating the degree to which the costs satisfy triangle inequality (for TSP), consideration of a gap between optimal solution and given heuristics (for Knapsack problems). For five-mentioned CO problems, important features contributing to higher prediction power were identified.

The most recent research [17–20] has shown improvements in identifying the crucial features and their impact on difficulty estimation. The authors tested the proposed ML methods for predicting the performance of algorithms applied on MFP [17], JSSP [18], 0-1 Knapsack [19], and SPP [20]. The obtained results offer new insights into the performance of algorithms on mentioned problems and further insights into problems themselves. Additionally, as it is stated in [36], ML may help us not only in detecting important features, but also in choosing the best suitable algorithm for a given problem, as well as in performance evaluation, parameter setting, etc.

Inspired by the above-mentioned research, we rely on CRISP-DM model [16] to define a difficulty estimation module of our framework, in which we could take any predefined features or define new features to predict the running time of the CO algorithm for a given problem instance. In the next section, we introduce our framework and explain its modules.

3. Controlling the Difficulty—Methodology

We propose a methodology to control the difficulty of miners' work in the PoUW-based BC systems. Its place in the consensus protocol is illustrated by the gray-shaded parallelogram in Figure 2. Similarly to the difficulty adjustment in PoW, controlling the difficulty is performed by the system. However, in PoUW there are actual clients that submit instances of their real-life problems to be solved by the miners. Their instances, although equally important to the clients, may significantly differ in the difficulties. In addition, the actual difficulty of each particular instance may not be known. Thus, it is necessary to estimate the difficulty of submitted instances and to group them in such a way that these groups represent similarly difficult tasks for miners. These two steps constitute our framework for controlling the difficulty. It is executed each time an instance of the real-life problem is submitted by the client and its output is a task (group of instances) to be

executed by the miners. This is the reason why the gray-shaded parallelogram in Figure 2 is connected to the client role and positioned below the submission of the task candidate for insertion into the pool.

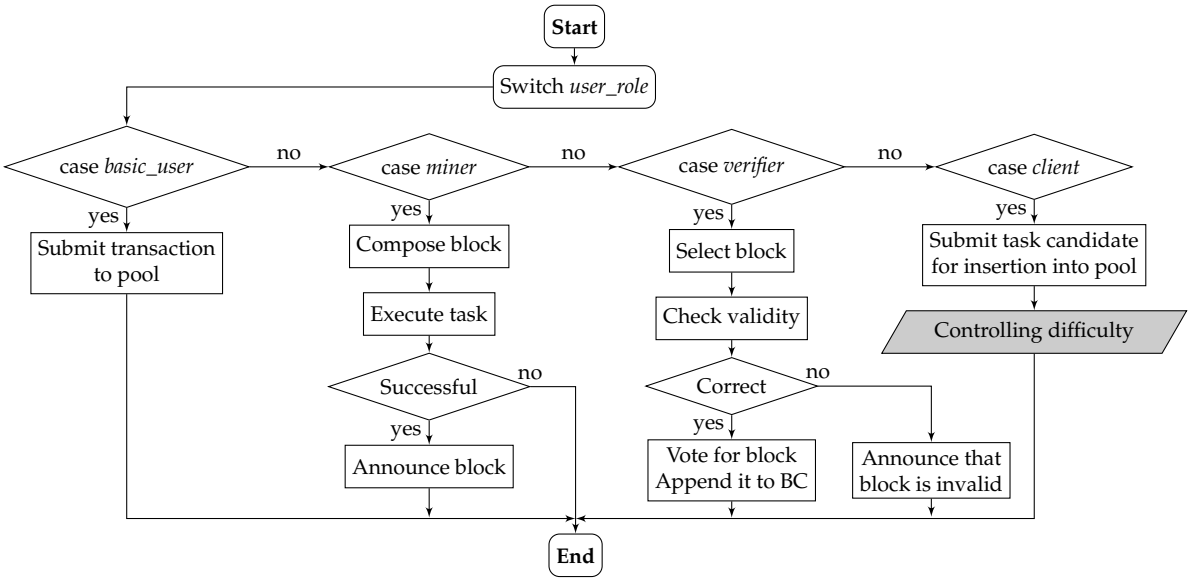

Figure 2. Flow-chart of BC users activities in our proposed framework.

As explained above, our framework for controlling the difficulty consists of two modules. The first module addresses the difficulty estimation of CO problem instances. The output of this module is *EstimateDifficulty* model (function) that would enable grouping module to balance the work of miners. This is realized by packing instances into groups. The two modules are explained in detail in the remainder of this section.

3.1. Difficulty Estimation Module

Our goal is to develop methods that will help us estimate the difficulty of a particular CO problem instance, i.e., the time needed for the considered optimization algorithm to solve it. The existing models for difficulty estimation could be used in the process. If no models are available, we could make a satisfactory difficulty estimation either by using the features already identified in the literature or by defining them in the process.

We modify CRISP-DM model that is presented in [16] and present our modifications in a flow-chart in Figure 3. More details of the model are explained in the rest of this section.

For a given problem in this module, we need to collect the relevant data and understand which features we could use. Better understanding of the data, its size and quality could lead to simplified and faster data transformation, adequate model choosing, and better model performance [37]. It is well known that domain experts could suggest characteristics/features of a problem that are crucial for improving the model performance [37,38].

As data could be collected from various sources, some essential tasks need to be conducted before building a model. These tasks include performing exploratory data analysis, visualization, looking for outliers and missing values, and checking the data quality [39]. They should employ statistical analysis, determine appropriate features, along with their collations, in order to perform data selection and cleaning. We obtain usable data through the execution of these tasks.

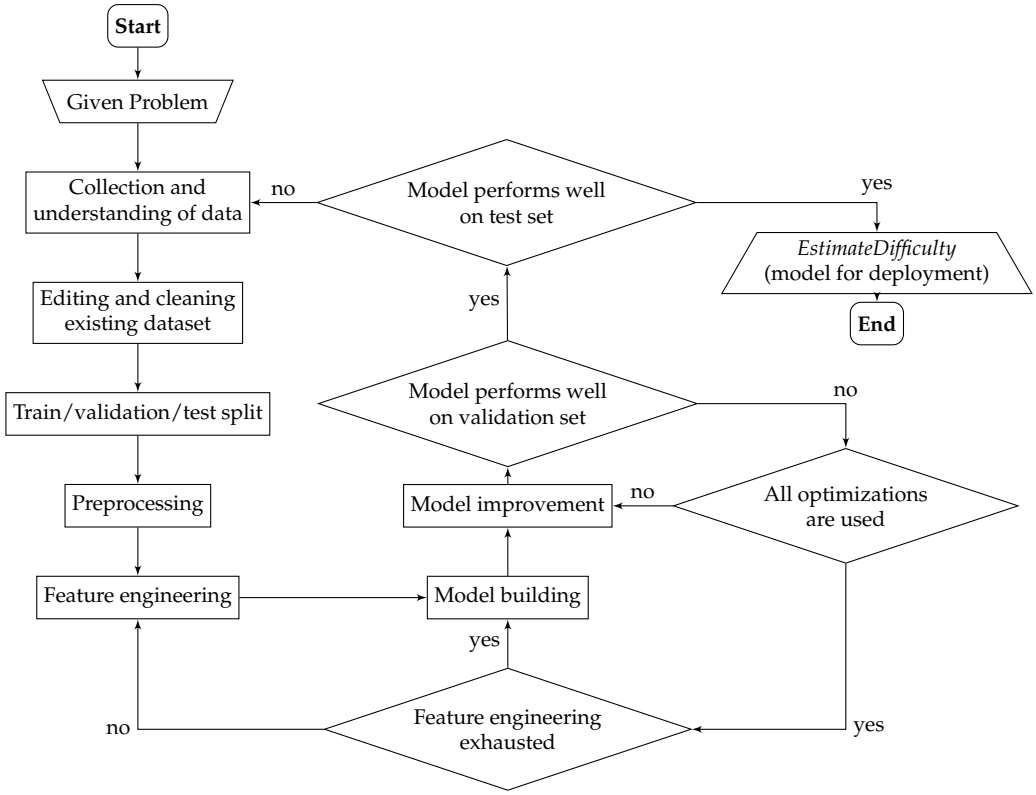

Figure 3. Flow-chart of difficulty estimation module.

A dataset is divided in standard train/validation/test split. In general, model parameters are tuned on a train/validation split and final tests are performed on test data. This gives a resulting model a better ability to perform well on unseen data [40,41].

Preprocessing is a first part of the model building and, depending on the used model, derived attributes have to be constructed. For preprocessing steps, different methods are possible and are model dependent. For example, data are normalized in order to improve data quality and thus, the performance of selected ML algorithms/models [42–44]. If data distribution is normal, standardization could be performed. During preprocessing, outlier removal, imputing missing data, and other actions that require fitting the data, should be performed and tested on a train/validation split.

In order to increase the prediction power, feature engineering is performed so that most critical parameters are kept relevant, while redundant and irrelevant ones are eliminated. First, we could estimate a number of features needed for capturing a predefined variance threshold [45]. A good threshold is considered to be above 70% but, in social science data, it could be as low as 50%. Next, several methods of feature selection [46–48] could be used in order to find the most relevant features, i.e., features that are commonly rated as highly important by all used methods. Finally, a needed number of features is kept (as defined by variance threshold). In the following steps, only features selected in this step are considered.

In the model building block, models with high bias, such as linear regression or logistic regression, are built first as they are simple and widely well understood. The goal is to figure out how to leverage the aforementioned selected features and obtain a better sense of the model's predictive power. When we obtain a better understanding of the problem and are not satisfied with the model's performance, we may apply more complex models that have high variance [37,49,50].

In the model improvement step, we optimize the current model based on the feedback from a validation set, and continue updating it until all optimizations are used and all feature engineering is exhausted. When satisfied with the model performance on the validation set, final testing could be performed. To avoid overfitting, optimizations are

performed gradually until acceptable performance on the validation set is achieved. Several methods [51–53] of model evaluation may be used, as well as some ad-hoc evaluations dependent on nature of a problem that we try to estimate. For example, eliminating extreme misclassifications in case of classification (that is, avoidance of non-zero values in upper-right and lower-left corner of a confusion matrix [54,55]). As for the model optimization/improvement, hyper-parameter optimization [56–58] is primarily used. Furthermore, we can implement DL models such as Convolutional Neural Network (CNN), Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM), etc. Using DL models [49], we can eliminate the need for feature engineering, maximize the utilization of the data, and procure better results through a feedforward network. Implementing DL also enables us to use many high-performing, pre-trained models and implement them using a transfer learning approach [50]. These multi-layered, pre-trained algorithms can be used as a starting point in our DL model, which allows rapid performance optimization with relatively smaller training duration. The ideal depth of the pre-trained models can be tuned automatically in the training phase using the Keras auto-tuner. Moreover, the entire training, validation, and testing phase can be encapsulated using pipelining, making the model clean, scalable, and robust [49,50,59].

Model deployment in our framework consists of providing assessment of the performed estimation and deployment plan. In particular, the deployment plan addresses incorporation of this module into general framework, i.e., providing a grouping module with a function *EstimateDifficulty* that will estimate the execution time for all given instances.

3.2. Instance Grouping Module

The next step of our study involves the usage of the difficulty estimation model, i.e., leveraging the predicted execution time of the considered CO algorithms, for balancing the work of miners in the BC systems. We propose to group CO problem instances together into packages of approximately the same size with respect to the sum of the expected runtimes. The size of packages should correspond to the BIT, and therefore, the instances with the estimated runtime close to the BIT may be alone in the package. The model for the estimation of CO problem instances difficulty is deployed as the basic step in the instances grouping module. We assume that there is an instance pool and the corresponding solution pool. The instance pool contains all unsolved instances that PoUW miners will use during the mining process. All solutions are stored in the solution pool to serve for validation purposes and to be provided to clients.

Grouping is applied within the instance submission part of the considered PoUW-based consensus protocol. Specifically, each submitted instance i undergoes the difficulty estimation process. If it is scored as hard, i.e., if it requires a large amount of CPU time, t_i, for finding a valid solution, it enters the instance pool as a single instance. Otherwise, the instance i becomes a member of a group containing several easier instances. In such a way, it is possible to balance the work of miners needed to publish a new block. This is important to ensure both the fairness in the mining process and almost constant BITs.

The difficulty of instance i is defined by the time, referred to as *solving time* t_i, required by the exact or stochastic heuristic method to find its valid solution. The solving time should correspond to BIT whose value is denoted by f, meaning that extremely hard instances (requiring more than f time units) cannot be accepted for submission to the instance pool. If $t_i \approx f$, the corresponding instance is considered to be hard and it solely represents the work miner needs to complete before publishing a composed block. All instances for which the estimated solving time is significantly smaller than f, are grouped together until the sum of their solving times approaches f. The external loop of instances grouping module, the function referred to as *Grouping*, is described by the flow-chart presented in Figure 4.

The $Grouping(i, \rho, f, \Delta)$ function is executed as an infinite loop that waits for the submission of an instance i and then decides if it should be discarded, executed as a stand-alone instance, or included into group of instances that the miner should solve before

publishing a new block. The input parameters of this function are i, ρ, f, and Δ. Here, ρ and Δ represent the tolerance parameters, whose roles are explained later in the text. In the case when instance i should be inserted in a group, the function $Insert(i, \rho, f, \Delta)$ (see Figure 4) is called to perform the insertion. The output of this function is set $G = \{G_1, \ldots, G_l\}$ of groups composed of the new instance i and/or the easy or moderately hard instances, submitted by clients, that have been waiting for the insertion. If G is an empty set, the *Insert* function was not able to compose any group, and function *Grouping* completes the current iteration without submitting anything to the instance pool. In the case G is not empty, each composed group of instances is submitted separately to the instance pool.

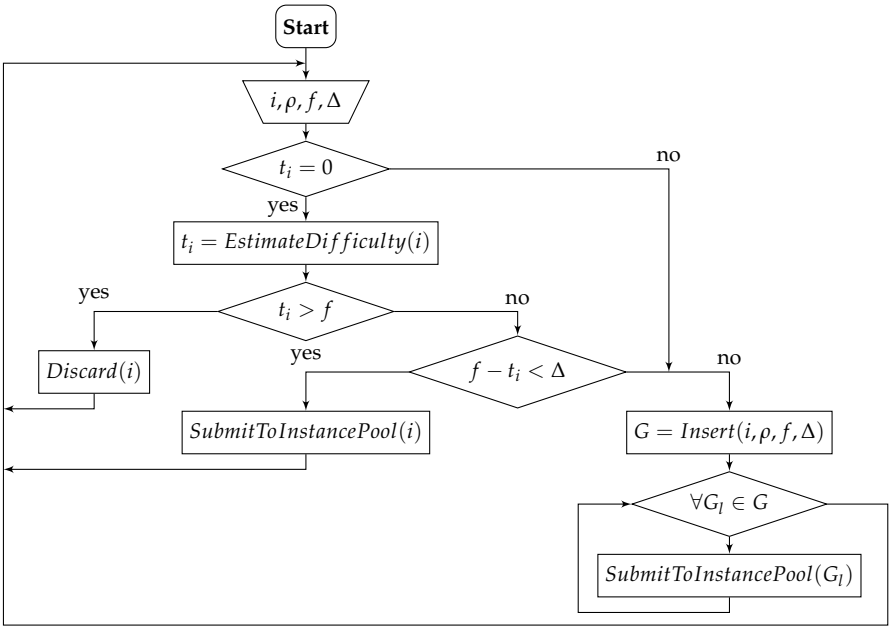

Figure 4. Flow-chart of instances *Grouping* module external loop.

The first **if** statement in the *Grouping* function identifies the instances that have already passed the difficulty estimation module. Specifically, it may happen that the miner did not manage to be the first one to solve all instances from the group and publish a new block. In such a case, all solved instances are transferred to the solution pool and the best found solution for each of them is provided to the client. The unsolved instances are returned to the instances grouping module to be inserted in some new group. For these instances, difficulty estimation is already performed, and it is known that they should be a part of some group. This is because too difficult instances have already been discarded and the ones with $t_i \approx f$ are sent individually to the instance pool on the first entrance to the *Grouping* function. Therefore, if for instance i, $t_i > 0$ holds, it is directly passed to the *Insert* function.

Our approach for grouping instances is based on Bin Packing Problem (BPP) [12,60] methodology under the assumption that the capacity of each bin is f. The only difference is that the bins are filled one by one. Greedy heuristic approach, based on the generalized algorithm for the well-known Subset Sum Problem (SSP) [61], is adopted: instances are packed into current bin until it is filled, or as long as it is possible. When the new instance cannot fit into the current bin, i.e., its inclusion would exceed the bin capacity, re-arrangement of instances is performed in such a way that several easy instances are removed from the bin to make room for the newly arrived harder instance.

An additional request, related to the usefulness of the PoUW, is to ensure that the instance deadline is not missed. Specifically, for each instance i, the condition $t_c + \rho \cdot t_i < d_i$ should be checked within *Insert* function. Here, t_c refers to the current time, while ρ represents the number of block insertion cycles expected for an instance to wait in the instance pool before it is solved, and d_i is the deadline for execution of instance i. In such

a way, we give chance for each instance from the group to be solved before its deadline, if the corresponding group is selected in due time by any miner. However, the selection of groups from the instance pool is dynamic, stochastic, concurrent process within the consensus protocol and it cannot be controlled by the instance grouping module. Challenges that PoUW-based consensus protocols face, presented in [4], indicate that stochastic and distributed execution of any PoUW consensus protocol, can prevent some instances from being completed before the pre-specified deadline.

The *Insert* function can be formalized by flow-chart presented in Figure 5. This function also represents an infinite loop, and within it the following instructions are performed. First, all data structures are cleaned for the new iteration. The set of composed groups G and the set of mandatory instances M are initialized to $\emptyset$. The input parameters of this function are the same as for the *Grouping* function: i, ρ, f, Δ. All instances that are not yet included in any group are the members of the working set called heap H. When a new instance i arrives, it joins others in the heap. To favor instances with closer deadlines, a set of mandatory instances M is formed from all instances j for which $d_j > t_c - \rho \cdot f$. These instances should be immediately sent to the instance pool for execution even if it is not possible to compose large enough group, i.e., the group of instances for which the relation (1) holds.

$$f - \Delta \leq \sum_j t_j \leq f. \tag{1}$$

Out of set M, instances are packed in the best possible way to form a group G_M by the application of a generalized algorithm for the well-known SSP [61]. In addition, if G_M is not large enough, we try to compose the supplement S out of non-mandatory instances from the heap H. Again, the SSP algorithm is employed. If a valid group is obtained, i.e., $G_M \cup S$ is not empty, it is added to a set of composed groups G to be returned as an output from the *Insert* function. In the case when a valid group is not obtained, the set of composed groups G and the set of mandatory instances M are re-initialized and the loop is restarted. All used instances are removed from the heap H and the set of mandatory instances M. The described procedure is repeated until it is not possible to compose valid group of instances. In that case, the set of composed groups G is returned from the *Insert* function.

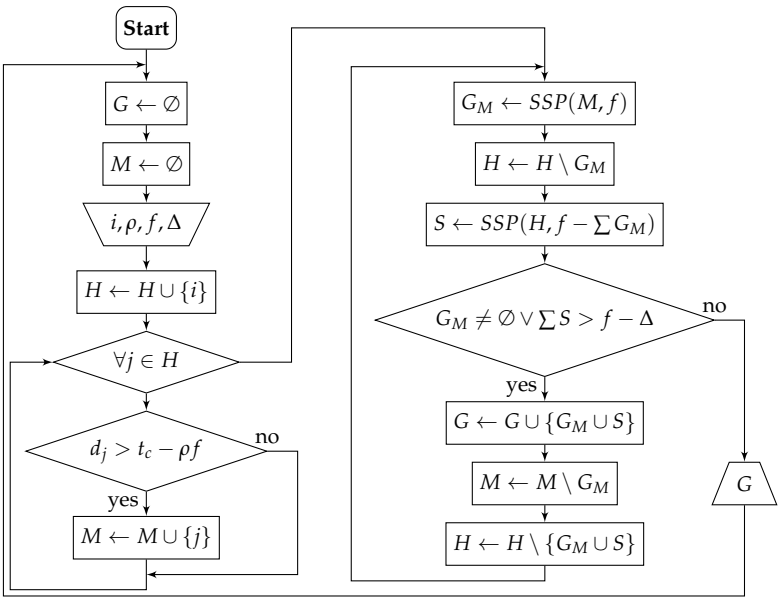

Figure 5. Flow-chart of instances grouping module inner loop (*Insert* function).

Each obtained group, called *package*, is submitted to the instance pool as a single unit of work for miners. A package is identified by the name of file containing web addresses of input data for all instances constituting that package. Miners take instances one by one, execute the corresponding algorithm for finding its valid solution, and write the obtained

solution in the solution pool. When all instances are solved, a new file containing links to the generated output files is created, and the miner is ready to publish the composed block.

The difficulty estimation module cannot predict exactly the solving time of any instance. Therefore, we cannot guarantee that the smaller packages will be executed faster than the packages with larger predicted total time. On the other hand, we can record the exact running time for each instance and use it to improve the performance of our difficulty estimation module either off-line or during its deployment.

4. Case Study

We tested the proposed framework on $P||C_{max}$ problem [21,62,63]. To the best of our knowledge, there are no studies that have tried to establish difficulty estimation for general $P||C_{max}$ problem. The authors in [64] focused on optimization of parallel machine scheduling using estimated processing time for a specific use case. They used the parameters of that specific use case as features. Thus, in this work, our focus is to address general $P||C_{max}$ problem.

The use cases for our framework are CO problem solvers ArcFlow [22] and GIST [23] algorithms developed for $P||C_{max}$ problem [21,62,63]. Model building in the difficulty estimation module works the same way for both problems, but the resulting models might differ from one problem to another. Grouping instances module works in the same way for both case studies. In the remainder of this section, we explain how the framework is applied.

All results are obtained using a free cloud service hosted by Google to encourage ML and AI research (https://colab.research.google.com/notebooks/intro.ipynb, accessed on 1 August 2022). The code is available in our Github repository cited in data availability section.

4.1. Difficulty Estimation Module

We develop a model for predicting difficulty of the given $P||C_{max}$ scheduling problem instances when solved by the considered exact (ArcFlow) and heuristic (GIST) algorithms. The input data of these instances are number of tasks n, number of processors m, and a set of processing times for n independent tasks $P = \{p_1, p_2, \ldots, p_n\}$. By difficulty, we mean the estimated time needed for the algorithm to reach the value of the makespan C_{max} that is better than a predefined threshold value.

Our dataset consists of 8750 $P||C_{max}$ instances, each described by 23 features, two of which are target variables (time needed for instance execution and indicator if algorithm managed to find an optimal solution). Only one target variable is considered: time needed for instance execution, and we refer to it as y. The 21 non-target features are proposed by the domain experts. Two of them represent the names and types of instances, and we disregard them. Three features are indicators of how data are organized, while the remaining 16 are extracted statistical properties of elements of the above-mentioned set P and m. Therefore, 19 features are utilized in the development of our difficulty estimation model, all presented in Table 1.

Table 1. List of features used in difficulty estimation module.

Feature Notation	Explanation of Feature
n	Cardinality of set P, that is, number of tasks
m	Number of processors
av.length	Average length of elements in set P
median	Median value of elements in set P
std.dev	Standard deviation of set P
max	Maximum value in set P

Table 1. *Cont.*

Feature Notation	Explanation of Feature
min	Minimum value in set P
range	Difference between maximum and minimum value in set P
k	Number of different values of elements in set P
rel.bound	Difference between a solution upper and lower bound relative to the upper bound
n/m	Average number of tasks per processor
$(n/m)^2$	Polynomial feature of n/m
$(n/m)^3$	Polynomial feature of n/m
m/n	Polynomial feature of n/m
$(m/n)^2$	Polynomial feature of n/m
$(m/n)^3$	Polynomial feature of n/m
subtype	Instances with same subtype have the same n/m value
class	Probability distribution of random generator for elements of P
index	Index of instance in dataset with same subtype and class [1–10]

4.1.1. Experimental Setup

The domain experts suggested that the instances should be classified into one of four categories: 0-*easy*, 1-*moderate*, 2-*hard*, 3-*very hard*. However, as the running times of algorithms can differ for several orders of magnitude, we opted to define categories for each algorithm separately. When doing this, we used the logarithmic scale to capture the values from the whole space.

We aim for a simple and interpretable model, which could be used by the domain experts to understand the importance of features for each considered algorithm. As our goal is not to explain the importance of features in this manuscript, we opted to show the process and the performance we could achieve in more details.

Following the recommendations in [40,41], we divided our dataset into train/validation/test with the following ratios: (train + validation)/test we split in ratio 83/17 and train/validation, and we also split in the ratio 83/17.

For the performance measure of the classification, we use two variants. The first variant is the precision, which refers to how close to each other are the measurements of the same item. The second measure is the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) [65]. As the limits for categories are too broad, we also perform regression to find more fine-grained estimates of the execution times. To evaluate the performance of linear regression model, we use statistical properties for goodness of fit and error. For a goodness of fit, we use R squared, a measure that evaluates how much variance of a dependent variable is explained by an independent variable. Even though that measure tells us how good regression model is, the final decisions are made using Root Mean Squared Error (RMSE), defined by (2), as it properly indicates the quality of the estimates.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{D}(x_i - y_i)^2}{D}} \tag{2}$$

4.1.2. ArcFlow Case Study

In this case study, the limits for categories are defined by instance runtime values as follows *easy*: 0–10 s, *moderate*: 10–100 s, *hard*: 100–1000 s, *very hard*: >1000 s.

Not solved instances were assigned the maximum observed execution time of 4510 s. Thus, for a list of our 19 features, we provide their correlations with target y in Table 2.

From Table 2, it could be concluded that, individually, statistical properties like standard deviation (std.dev), maximum (max), average length (av.length), and median as well as characteristic of the problem instance (k), are among the highest correlated features with the execution time for ArcFlow algorithm (target y). As it is shown in Table 2, feature *rel.bound* is not correlated with the target y.

Table 2. Correlation of features with execution time of ArcFlow algorithm.

Feature	Correlation with Target y
y	1.000000
std.dev	0.539327
max	0.538683
av.length	0.538287
median	0.536837
k	0.529941
range	0.528752
min	0.495868
n/m	0.423243
$(n/m)^2$	0.420624
$(n/m)^3$	0.412105
subtype	0.386236
m/n	0.377794
class	0.364556
n	0.356450
$(m/n)^2$	0.337697
$(m/n)^3$	0.300864
rel.bound	0.246362
m	0.100150
index	0.005231

Next, we examine how many features are needed to capture a threshold of 90% of the variance [66]. We opted to be more conservative with our threshold as it is just providing a number of features to be considered in an ideal case. In order to do Principal Component Analysis (PCA), we need to perform proper scaling (using StandardScaler()) and oversampling [67] (using RandomOverSampler()). The results of the applied PCA show that 5 features are needed so that their cumulative explained variance could reach the desired threshold, as presented in Figure 6.

Trying to obtain the number of features that PCA indicated, we applied feature permutation [68], PCA [69] and looked into feature importance in logistic regression [70] to decide upon feature selection. The five most frequently highly rated features were n, m, std. dev, max, and av. length. Therefore, we chose them for input features of logistic regression. To build it, we performed grid search on hyperparameter C [71] using 15 values uniformly distributed on logarithmic scale $[-5, 5]$. The best hyperparameter value turned out to be $C = 26.82695$ with max iter $= 1000$. Logistic regression achieved a precision of 77.46% on the validation set, with the following AUC scores for categories: easy: 0.93, moderate: 0.77, hard: 0.78, very hard: 0.91.

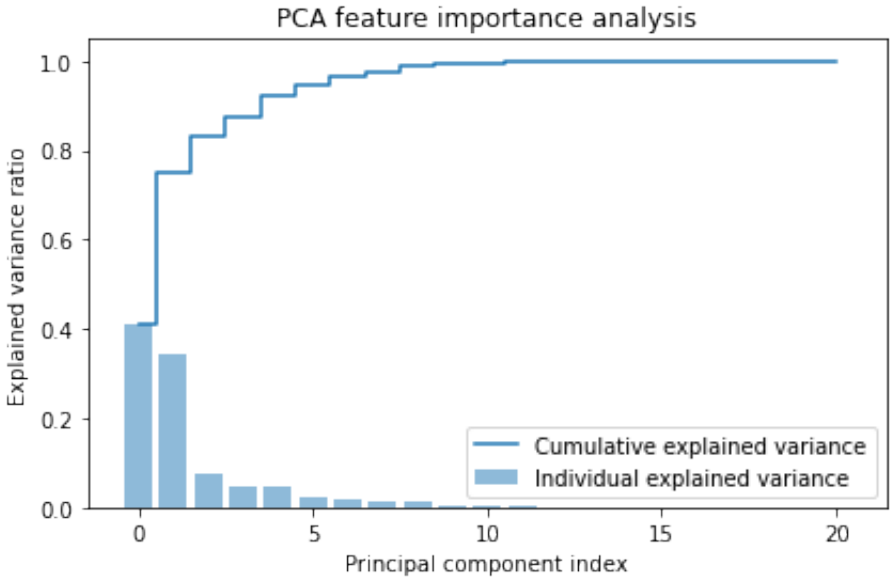

Figure 6. PCA feature importance analysis.

From AUC scores, we can conclude that logistic regression performs better on easy and very hard instances than on moderate and hard ones. Possible reasons for this performance could be very sharp boundaries between categories. Moderate and hard categories are in some sense in the "middle". Therefore, it is expected that some moderate instances can be misclassified as easy or hard, and some easy instances can be misclassified as moderate.

However, we noticed that some very hard instances are misclassified as easy. In Figure 7, we plotted the four misclassifications from very hard to easy, in the space of av. length, max and std. dev as axes. It can be seen that large black points (representing very hard instances misclassified as easy) are very close in feature space to correctly classified very hard instances (yellow).

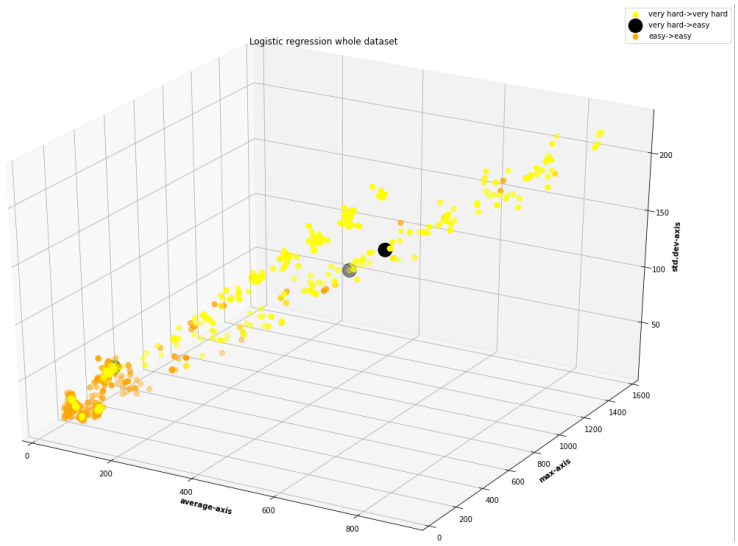

Figure 7. Logistic regression—3D scatter plot for correct and incorrect classifications.

To understand why this happens, in Figure 8, we also generated 2D scatter plot in the space with n and m as axes containing points from the validation set (correctly classified easy instances are orange points, correctly classified very hard are yellow, and four mentioned misclassified instances are large black points). In Figure 8, black points completely coincide with the orange ones. Hence, our deduction would be that we need

more information than we have for features n, m. However, the coefficients of features in logistic regression $[11.20589791, 8.24008468, 2.05775954, 1.89851106, 2.6349542]$ for n, m, max, av.length, and std.dev, respectively, confirmed that these features (n, m) are more important than the remaining three.

In Figure 8, we note that all large black points occupy part of 2D space, where n and m are relatively big, with additional trait that $n = 2m$. There are clearly many more easy instances with $n = 2m$ than the hard ones, as seen in Figure 8. Therefore, it is not unusual for our model to classify black points as easy. Finally, the question should be, how is it possible for them to be very hard?

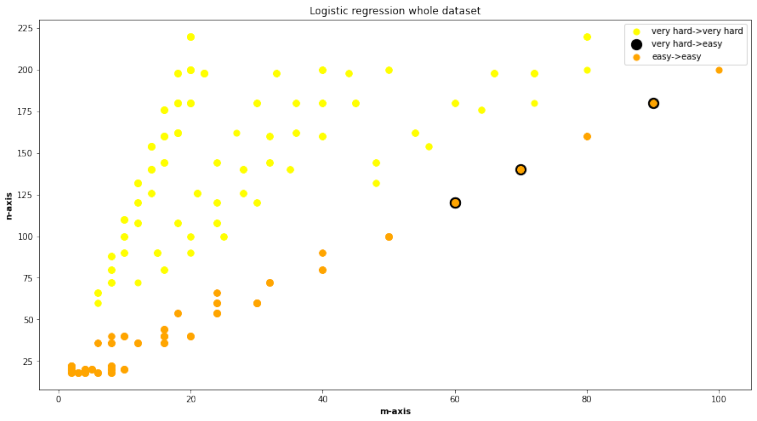

Figure 8. Logistic regression—2D scatter plot for correct and incorrect classifications.

Binomial coefficient $\binom{n}{m}$ could provide an answer to this question. Specifically, with fixed n, it is known that the largest value of $\binom{n}{k}$ is obtained for $k = \lfloor \frac{n}{2} \rfloor$. Hence, for larger n and m, the coefficient $\binom{n}{m}$ is by far the largest for $n = 2m$, when n is fixed. For example, for $n = 180$ $\binom{180}{90} \approx 25000\binom{180}{60}$ and $\binom{180}{90} \approx 1.5 \cdot 10^{10}\binom{180}{45}$ and, given that $\binom{n}{m}$ is correlated to time complexity of algorithm execution (simply, the larger the coefficient, the more possible combinations for distribution of tasks on processors), it is clear why mentioned black points are originally very hard.

In short, we concluded that instances with $\frac{n}{m} = 2$ exhibit a different behavior for large n, m. Based on that observation, in the sequel, we divided our dataset on instances with $\frac{n}{m} = 2$ and with $\frac{n}{m} \neq 2$. What follows is the analysis of models for each of the two mentioned subsets.

After dividing dataset, feature selection is repeated. For $\frac{n}{m} \neq 2$, the following 4 features are sufficient to obtain a well-performing model: $\frac{n}{m}$, max, av.length and (k). ANN is used as an improvement over logistic regression. The characteristics of this model are: one hidden layer consisting of 16 neurons and the output layer of size 4. ReLU is used as an activation function in the hidden layer with dropout rate of 0.2 (which showed the best performance among values $[0.1, 0.125, 0.2, 0.25, 0.5]$, tested on validation set). For the output layer, activation function is softmax. The model performs with precision of 81.01% on the validation set and with following AUC scores for categories: easy: 0.94, moderate: 0.82, hard: 0.79, very hard: 0.94. Misclassifications from very hard to easy, or from easy to very hard are eliminated.

For $\frac{n}{m} = 2$, different ANN and the following 4 features are used: n, max, av.length and std.dev. Characteristics of ANN are two hidden layers (8 and 12 neurons, respectively) and the output of size 3 (because category of 'hard' instances is empty). ReLU is used as an activation function in both hidden layers. The second hidden layer has dropout rate of 0.125 (which showed the best performance among values $[0.1, 0.125, 0.2, 0.25, 0.5]$, tested on the validation set). The output layer activation function is softmax. The model performs with precision of 95.38% on the validation set and with following AUC scores for categories: easy: 0.94, moderate: 0.96, very hard: 1.00. Misclassifications from very hard to easy, or from easy to very hard, are eliminated on this subset. However, it is not impossible for a

few easy instances to be misclassified as very hard-given that the number of instances in the training set is smaller than in case when $\frac{n}{m} \neq 2$.

From the previous analysis, we may conclude that a hybrid ANN model (processing the mentioned two subsets of data separately) outperforms logistic regression. However, as we need to obtain more granular time estimates, our final model has to perform regression. If we apply linear regression on the whole dataset, i.e., without any separation, RMSE for the validation set is as low as 80. That performance could be misleading as extremely inaccurate predictions for easy instances could be noticed (see Figure 9). The model predicts much longer runtime for solving instances than it is really needed. Our small RMSE can be explained by more correct prediction of runtime for harder instances. Therefore, we conclude that we need to build a hierarchical model [72] to treat easier instances independently from the harder ones. We use previously constructed classifier, and what follows is an analysis of our final model.

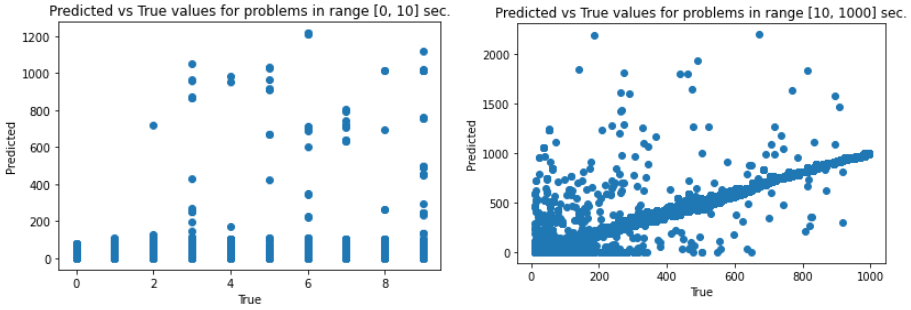

Figure 9. ANN regression predictions (without classifier applied beforehand) for instances with execution time in range [0, 10] s on the left and [10, 1000] s on the right.

The hierarchical model that we built, classifies instances into one out of three categories (easy, medium+hard, very hard), and proceeds with regression built for each of three categories separately. Those regression models are independent from each other, therefore, we need to perform feature selection for each one separately.

For feature selection performed for the instances classified in [0, 10] s range, the five most important features turned out to be n, m, k, std.dev, and max. Using a validation set, ANN is chosen to have two hidden layers, the first with 8, and the second with 12 neurons, after which dropout with a rate of 0.1 was applied. All activation functions are ReLUs, except for the output layer, for which we took $10 * sigmoid(x)$ as an activation function, where $sigmoid(x)$ is defined by (3).

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

To prevent the domination of misclassified values over the correct ones, logarithmic scale was used for target (y), i.e., function $\log(100 + y)$ is applied. Before performance evaluation, inverse function is applied to obtain the real RMSE values. In such a way, the final RMSE value was 1.2 and R-squared score of 0.982. For more insight, a plot of predicted vs true values for instances from the test set, is presented in Figure 10.

For the instances classified in [10, 1000] s range, the five most important features turned out to be n, k, n/m, std.dev and av.length. We chose ANN with 2 hidden layers, the first with 8, and the second with 32 neurons, after which dropout with a rate of 0.1 is applied. All activation functions are ReLUs, except for the output layer activation function, for which we took $4510 * sigmoid(x)$. We note the increase in model complexity (larger number of neurons in the second hidden layer), which is a consequence of a larger range of target variables, i.e., runtimes. On the test set, the model performance was quantified with RMSE equal to 118.26 and R-squared score of 0.86. Figure 11 illustrates the relationship between predicted and true values for instances classified in range [10, 1000] s.

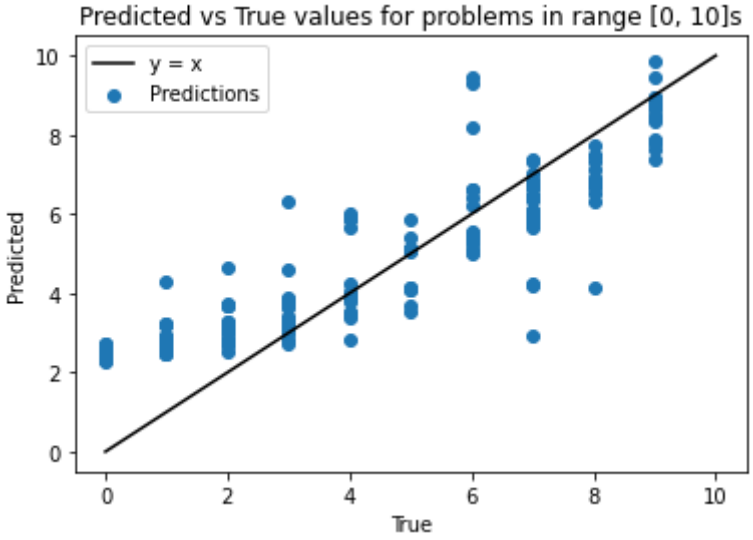

Figure 10. ANN regression predictions (with classifier applied beforehand)—for instances from test set with execution time in [0, 10] s.

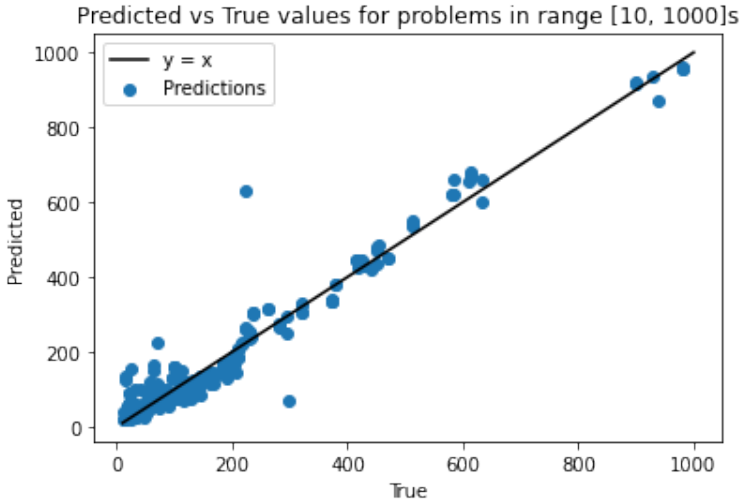

Figure 11. ANN regression predictions (with classifier applied beforehand)—for instances from test set with execution time in [10, 1000] s.

Finally, for the instances classified in [1000, 4510] s range, the five most important features are identified as n, k, std.dev, av.length, and median. However, given the extremely high correlation (0.999507) between av.length and median, we replaced median with m. This subset of instances required even higher ANN model complexity: four hidden layers; the first with 16, and the second with 32 neurons, after which dropout with rate 0.1 was applied, the third with 16, and the fourth with 4 neurons. All activation functions are ReLUs, except for the output layer activation function, for which we took $4510 * sigmoid(x)$. The increase in the model complexity is a consequence of even larger range of target variables (runtimes). The model performed on test set with RMSE of 136.61 and R-squared score of 0.98. A plot of predicted vs true values for instances in range [1000, 4510] s is presented in Figure 12.

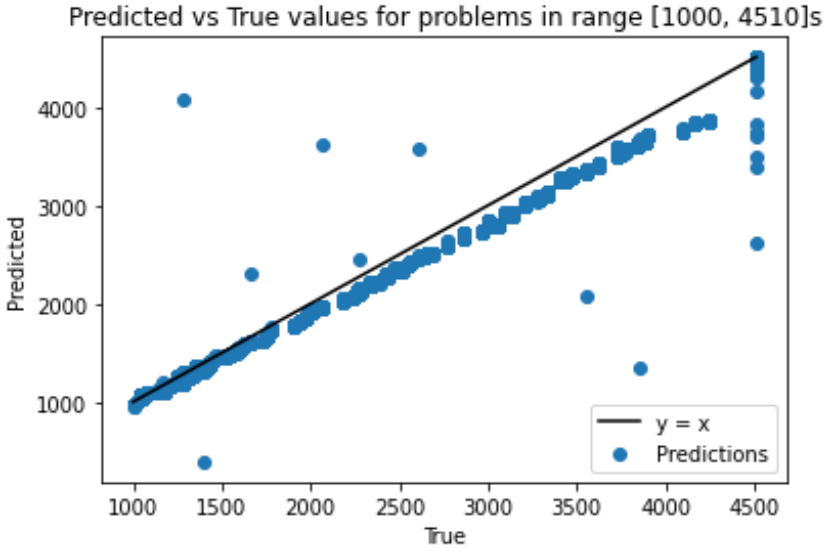

Figure 12. ANN regression predictions (with classifier applied beforehand)—for instances from the test set with execution time in [1000, 4510] s.

Combining the results of the three models, the performance of our hierarchical model on the test set resulted in RMSE of 130.66, and R-squared score of 0.987. The final plot, illustrating the relationship between the predicted and true values for all instances from the test set, is presented in Figure 13.

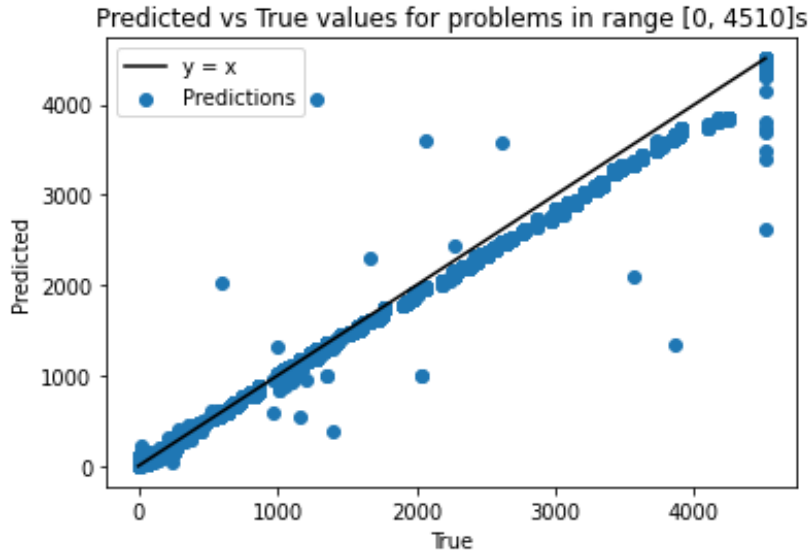

Figure 13. Hierarchical model predictions for the whole test set of ArcFlow runtimes.

Applying this hierarchical model, we obtained the significant improvement in prediction of instances with smaller runtimes, as well as a slight improvement for harder instances. RMSE values are much smaller for easier instances than for the harder, indicating a good quality of the model.

4.1.3. GIST Case Study

In this case study, very different correlations of features with target y are noted, as shown in Table 3.

Table 3. Correlation of features with runtime for GIST algorithm.

Feature	Correlation with Target y
y	1.000000
rel.bound	0.476547
m	0.442684
subtype	0.430256
m/n	0.418171
$(m/n)^2$	0.411345
$(m/n)^3$	0.382118
n/m	0.340878
av.length	0.306203
median	0.305065
min	0.302683
max	0.297263
$(n/m)^2$	0.289090
range	0.283401
std.dev	0.253432
$(n/m)^3$	0.250783
k	0.215968
class	0.211151
n	0.160885
index	0.007393

We see that *rel.bound* is the feature with the highest correlation with target y. At the same time k, the feature having the highest correlation with target y in ArcFlow case study, is now among the least correlated features. Our analysis in this case study led to a very similar hierarchical model as obtained in ArcFlow case study. Combining the results of the three models, the final performance of our hierarchical model resulted in RMSE of 0.99, and R-squared score of 0.97. A plot of predicted vs true values for all instances from the test set is presented in Figure 14.

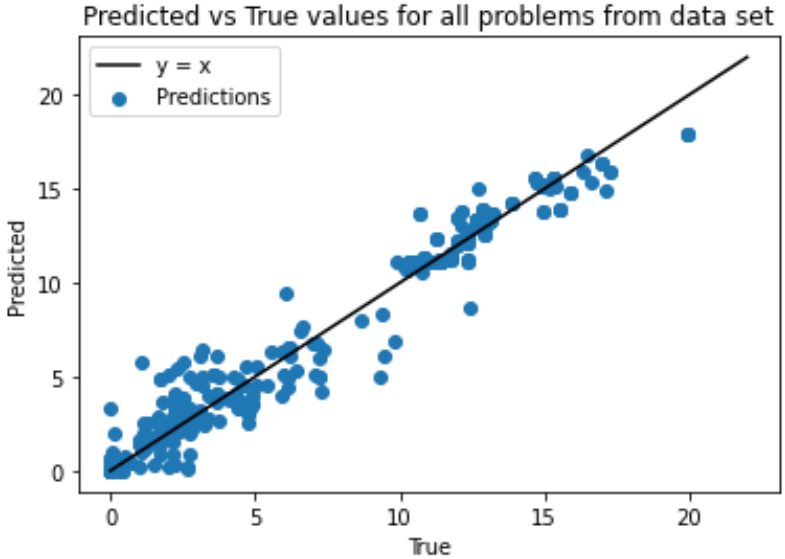

Figure 14. Hierarchical model predictions for the whole test set of GIST runtimes.

The fact that resulting RMSE is much smaller for GIST case study could be just a consequence of a different runtimes' range. R-squared values in both hierarchical models indicate a good fit for data that we have. Additionally, from the calculated R-squared values we can conclude that the obtained runtime estimates for instances of $P||C_{max}$ problem are reliable.

4.2. Instances Grouping Module Case Study

To illustrate the execution of instances grouping module, we performed a simulation on small example containing 20 instances of $P||C_{max}$ problem. The estimated runtimes t_i of these instances are presented in Table 4, together with the order $o(i)$ of their arrival to the start of *Grouping* function. In the text to follow, we use notation I_i when referring to the instance i. Having in mind that the t_i values are ranging from 0.000013 to 19.907111, we set $f = 17$ and $\Delta = 2$. For this simulation, it is not important to specify values for d_i and ρ because it is executed outside the actual BC environment. However, to cover the case with critical instances, we include a description of what has to be done when the deadline is approaching for an instance.

Table 4. Estimated running times of the considered instances.

i	1	2	3	4	5	6	7	8	9	10
t_i	0.000013	0.0001	0.000105	0.000231	0.000471	0.000607	0.001301	0.275697	1.420326	4.651514
$o(i)$	10	20	6	19	17	3	13	7	12	11
i	11	12	13	14	15	16	17	18	19	20
t_i	4.905763	4.928929	5.492491	5.833624	6.058383	6.132013	9.387262	11.366228	15.392948	19.907111
$o(i)$	8	4	9	14	2	18	15	1	5	16

In the first iteration of instances grouping module, the instance I_{18} arrives. It passes both checking points in *Grouping* and enters *Insert* function. As there are no other instances in the heap, and assuming that d_{18} is not critical, it is not possible to generate a large enough group to be submitted to the pool. Therefore, an empty package G is returned by the *Insert* function.

The next instance to arrive is I_{15}. Combining instances I_{18} and I_{15} into a single package is not possible as the sum of their execution times exceeds f. Both of them remain in the heap and an empty package is returned. After instances I_6 and I_{12} are passed to the *Insert* function as the third and the fourth, respectively, it is possible to compose a package with the total execution time of all instances within the interval $[f - \Delta, f]$ and to send it for the submission to the pool. This package is composed of instances I_{18}, I_6, and I_{12}, while instance I_{15} remains in the heap. The described state is illustrated in Figure 15.

When the fifth instance, I_{19}, arrives to *Grouping* it is immediately sent to the instance pool because $f > t_{19} = 15.392 > f - \Delta = 15$.

The second package in the *Insert* function is composed combining instances I_3, I_8, I_{11}, and I_{13} with instance I_{15}, which is already in the heap. When this package is returned for submission to the pool, no instances remain in the heap.

Now, we assume that the new instances arrive in the following order: I_1, I_{10}, I_{12}, I_7, and I_{14}. There are no critical deadlines among them, the sum of their estimated runtimes is smaller then $f - \Delta$, and they are all stored in the heap. Consequently, an empty package is always returned to *Grouping* function.

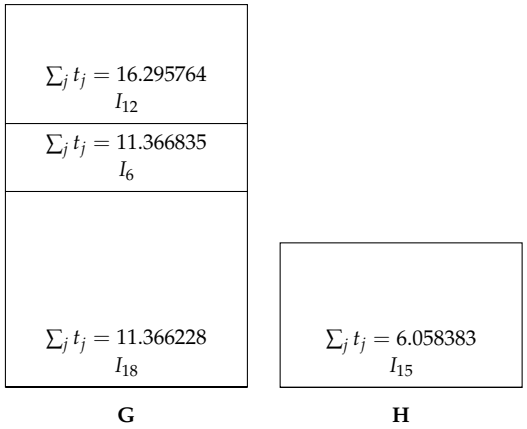

Figure 15. The first generated package and the heap content after analyzing four CO problem instances.

Let us assume that instance I_{17} arrives next. With this instance, the SSP algorithm manages to compose a large enough package in such a way that only instance I_{10} remains in the heap. The generated package is returned for submission to the pool.

The estimated runtime of instance I_{20} is greater than f. This means that it will never manage to complete its execution in the BC system with the above defined time parameters. Therefore, it is discarded from the system.

The new instances that enter the *Grouping* function one by one are I_5, I_{16}, and I_4. Although they (together with I_{10} that is already in the heap) cannot create a large enough package, we assume that the deadline of instance I_{16} approaches and the *Insert* function returns a new package, composed of all the available instances, for the submission to the pool. The structure of the composed package G is presented in Figure 16. The heap remains empty.

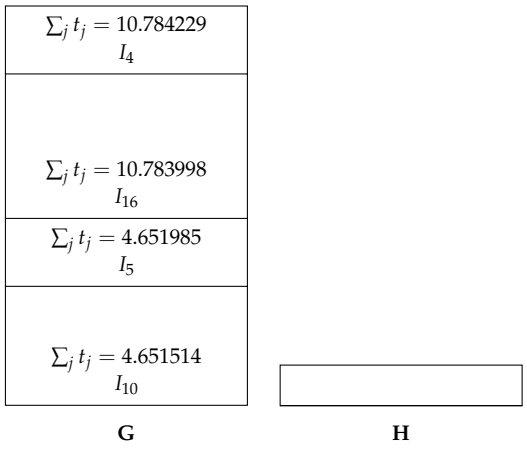

Figure 16. The structure and size of the last composed package.

I_2 is the last instance that we consider in our simulation. It passes two checking points of *Grouping* module and is put in the heap by the *Insert* function (Figure 17) waiting to be included in some new package and sent away for submission to the instance pool.

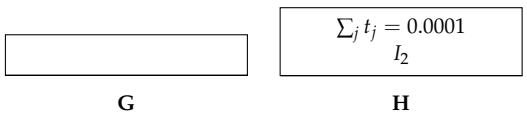

Figure 17. The content of the heap at the end of simulation.

5. Discussion

Within our difficulty estimation module, we tested how well we can predict runtimes of instances for two different optimization algorithms (ArcFlow and GIST), developed for scheduling independent tasks to homogeneous multiprocessor systems. Following flowchart from Figure 3, we concluded that, for both algorithms, the best prediction model is hierarchical. That is, for the sake of grouping together instances with the same order of magnitude of runtimes, the classification is applied first. Then, for finer grained prediction, regression is applied within each of the obtained groups. This way, we better captured the unique characteristics of instances from each of the classes. Moreover, we noticed different correlations of independent variables (features) with target between ArcFlow and GIST algorithms, which shows that our framework is flexible and works successfully no matter what is the nature of the prediction algorithm. Additionally, different time ranges (from 0 to 4510 s for the ArcFlow algorithm and from 0 to 21 s for GIST), do not have significant impact on the prediction results. For both algorithms, and all three categories, we got satisfactory results with regression on each class individually. Average R-squared score of 0.987 for ArcFlow, and average R-squared score of 0.97 for GIST indicate almost perfect goodness of fit. RMSE shows sub-linear monotonic increase tendency for ArcFlow algorithm. RMSE is 1.2 on the class with time range from 0 to 10s, 118.26 for the middle class (from 10 to 1000 s), and 136.61 on the class involving times in the range 1000 to 4510 s with the global RMSE taking value 130.66. Global RMSE for GIST is 0.99, while individual RMSE values are 0.45 for the first class (when runtimes range from 0 to 1 s), 1.42 on the second class (from 1 to 10 s), and 0.85 on the third class (from 10 to 21 s). Here, RMSE does not show monotonic tendency. However, the obtained values are very small with respect to the runtime ranges of each class.

In the grouping instances simulation, we used 20 CO problem instances with estimated runtimes ranging from negligible 10^{-5} s to almost 20 s. As the BIT was set to $f = 17$ s, the largest instance needed to be discarded from this BC system and, hopefully, executed in another. Out of the remaining 19 instances, we were able to compose five packages of similar size. One contains a single instance with the estimated running time equal to 15.392948 s. One package is a bit shorter (10.784229 s), as it contains an instance whose deadline approaches and, favoring the usability of the PoUW consensus protocols, we wanted to give it a chance to be solved. The size of remaining packages fits into the predefined time interval $[f - \Delta, f] = [15, 17]$. Having in mind that the difficulty estimation procedure is not 100% accurate, we believe that each miner is given a chance to solve all instances from the assigned package and publish the composed block. However, the main goal of our framework, to group the instances that may differ in size for several orders of magnitude into the packages of similar size, is certainly achieved.

Our case study clearly illustrates all advantages of the proposed framework. First of all, it enables the fairness in distribution of work among miners, gives them equal chances to complete the required amount of useful work, and to add the composed block to the BC. The second advantage is that the amount of work can be easily adapted to any desired BIT and to keep the intervals between insertion of blocks stable. Finally, our framework ensures that all the performed work is useful, no other overhead is required to keep the resulting PoUW consensus protocol fair and stable.

This actually recommends our framework to be incorporated in PoUW-based consensus protocols and to replace the usage of nonce values in PoW sense for controlling difficulty of miners' work. Specifically, our approach enables elimination of both disadvantages of using nonce in PoUW-based consensus protocols. First, the solution space could be unexplored if the valid nonce value is obtained too early, i.e., before high-quality solution is found. Second, the application of our framework prevents exploration of the solution space more than it is necessary. More precisely, it prevents the continuation of the search until valid nonce is obtained in the case when the solution of satisfactory quality was already found.

It is important to note that the proposed framework cannot be used as a standalone procedure, it needs to be integrated into a functional PoUW-based consensus protocol. Our goal was to ensure fairness and stable BIT for any PoUW-based consensus protocol because it represents energy efficient alternative to classic PoW. The performed simulation has shown that, by packing instances into groups, miners will receive more balanced tasks than in the case when individual instances are directly submitted to the pool.

6. Conclusions

Proof-of-Useful-Work (PoUW) consensus protocols have been developed to increase the efficiency and usability of Blockchain (BC) systems. The main idea of PoUW is to solve some problems useful to a wider community. Common issues in implementing these consensus protocols are controlling the difficulty of the miners' work and ensuring fairness and security. There are two main reasons for resolving these issues: to ensure fairness in rewarding miners and to keep the value of Block Insertion Time (BIT) stable. We developed the general difficulty estimation framework using modified Cross Industry Standard process for Data Mining (CRISP-DM) model, Artificial Intelligence (AI), Machine Learning (ML), and data science methods. Using the estimated execution times required to complete the useful work, the balance of workload for miners is achieved by grouping instances into equally difficult work elements to be executed by miners. Case study simulation is performed to illustrate the benefits of the proposed framework.

The generality of our framework could be interpreted in many ways. First, it can be adapted to any PoUW-based consensus protocol. Next, it enables us to use already available estimates (by some other prediction model) of runtimes for the considered instances. In addition, the size of instance packages is an input parameter of our framework and it allows easy adjustment to any BIT value. Moreover, our difficulty estimation does not depend on the number of miners involved in the mining process. Therefore, we believe that any PoUW-based consensus protocol with our framework applied may serve as an adequate replacement for PoW-based consensus protocols.

Our main contribution is in showing that controlling the difficulty and fairness of miners' work in PoUW-based consensus protocols can be achieved exclusively through the useful work, without the need for solving cryptographic puzzles in the classical PoW sense.

For the future work, we plan to incorporate the proposed framework into prominent PoUW-based consensus protocols that ensure usefulness and security. Incorporating our framework would eliminate the need for applying PoW concept as a part of those protocols. Implementation of the framework might request addressing other Combinatorial Optimization (CO) case studies for which prediction models do not exist. That would both provide further validation of generality of our framework and contribute to algorithm design and analysis research field. Additionally, the difficulty estimation learning process could be submitted as a useful task of the PoUW consensus protocol. Finally, having a breadth of CO and ML/AI problems, hopefully, we will be able to attract more clients and ensure large enough number of instance packages for miners to work on.

Author Contributions: Conceptualization, T.D., T.J.K., U.M. and D.R.; methodology, T.D., U.M., A.H. and D.R.; software, U.M. and D.O.; validation, T.D., T.J.K., D.O., U.M., A.H. and D.R.; formal analysis, T.D., D.O. and D.R.; investigation, U.M., A.H. and D.O.; resources, D.O., U.M. and D.R.; data curation, D.O., U.M. and T.J.K.; writing—original draft preparation, U.M., T.J.K., T.D. and D.R.; writing—review and editing, T.J.K., T.D., D.R. and A.H.; visualization, U.M. and D.O.; supervision, T.D.; project administration, T.D.; funding acquisition, T.D. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been funded by the Serbian Ministry of Education, Science and Technological Development, Agreement No. 451-03-9/2021-14/200029 and by the Science Fund of Republic of Serbia, under the project "Advanced Artificial Intelligence Techniques for Analysis and Design of System Components Based on Trustworthy BC Technology (AI4TrustBC)".

Data Availability Statement: Project repository is available on https://github.com/Uros-Males/pouw (accessed on 10 November 2022). All other data could be recovered from the article.

Acknowledgments: This project has been supported by Penn State Great Valley Big Data Lab. We would like to express our gratitude to Rathore Falguni and Gayatri Bangar for their help with identifying and explaining the related work.

Conflicts of Interest: The authors declare no conflict of interest.

Notation

The following notation is used in this manuscript:

Abbreviations

ANN	Artificial Neural Network
AUC	Area Under the Curve
BC	Blockchain
BPP	Bin Packing Problem
BIT	Block Insertion Time
CNN	Convolutional Neural Network
CO	Combinatorial Optimization
COCP	Combinatorial Optimization Consensus Protocol
CPU	Central processing unit
CRISP-DM	CRoss Industry Standard Process for Data Mining
DL	Deep Learning
DPLS	Doubly Parallel Local Search
GIST	Greedy Iterative Stochastic Transformation
JSSP	Job Shop Scheduling Problem
LSTM	Long Short Term Memory
MFP	Maximum Flow Problem
MIP	Mixed Integer Programming
ML	Machine Learning
NSGA-II	Non-dominated Sorting Genetic Algorithm II
$P\|\|C_{max}$	Problem of scheduling independent tasks on identical parallel machines
PCA	Principal Component Analysis
DLBC	Deep Learning-Based Consensus Protocol
PoS	Proof-of-Search
PoUW	Proof-of-Useful-Work
PoW	Proof-of-Work
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristics
R-squared	Coefficient of Determination
SAT	Propositional Satisfiability
SNN	Shallow Neural Network
SPP	Strip Packing Problem
SSP	Subset Sum Problem
TSP	Traveling Salesman Problem

Symbols

$\emptyset$	Empty set
Δ	Package size tolerance parameter
$softmax$	Type of activation function
$ReLU$	Type of activation function
$sigmoid$	Type of activation function
k_{TSP}	Number of chosen coordinates in TSP instance
n_{TSP}	Number of coordinates in TSP instance
N_{TSP}	Number of coordinates in hard TSP instance
T_1	Threshold for good enough solution for TSP instance
i	Submitted instance

t_i	Instance score (difficulty)
t_c	Current time
f	BIT value
ρ	Number of insertion cycles an instance is expected to wait in the pool before it is solved
G_l	Group of instances
G	Set of groups of instances
L	Number of groups
d_i	Deadline for execution of instance
H	Heap of instances not included in any group yet
j	Instance from heap
M	Set of mandatory instances
G_M	Group formed of mandatory instances
SSP	Subset sum problem solver function
n	Number of tasks in $P\|\|C_{max}$
m	Number of processors in $P\|\|C_{max}$
P	Set of tasks in $P\|\|C_{max}$
p_q	Processing time of task q in $P\|\|C_{max}$
y	Time needed for $P\|\|C_{max}$ problem execution (target variable)
I_w	w-th example of instance for $P\|\|C_{max}$
Blockchain and machine learning jargon	
Miners	BC participants who have the computer hardware and appropriate software needed to mine digital currencies or solve complex mathematical problems
Consensus Protocol	Mechanism to perform BC management without the central authority
Transaction	A unit measure of data in BC
Block	Blocks are the basic containers of information in a blockchain, they contain transaction as stored data
Cryptographic Puzzle	A mathematical puzzle that miners must solve in PoW-based BCs in order to append their blocks
Proof-of-Work (PoW)	A common mechanism used to validate peer-to-peer transactions and maintain highly secured immutability of the blockchain
Proof-of-Useful-Work (PoUW)	Energy efficient consensus protocol that re-purposes the computational effort required to maintain protocol security to solve complex real-world problems
Proof-of-Learning (PoLe)	PoW that exploit the computation power of miners for training ML models as a useful work in consensus protocol
Proof-of-Search (PoS)	Combines BC consensus formation with solving optimization problems
Instance	An example of a problem with all the inputs needed to compute a solution to the problem
Feature	Individual measurable property or characteristic of an instance
Solution Space	The set of all possible solutions for the combinatorial optimization problem
Randomized Algorithm	An algorithm that employs a degree of randomness as a part of its logic or procedure
Parameter	The configuration variable that is internal to the model and whose value can be estimated from the given data
Hyperparameter	The explicitly specified parameter that controls the training process
Training (Train) Dataset	A dataset of examples used during the learning process to fit the parameters
Validation Dataset	A dataset of examples used to tune the hyperparameters
Test Dataset	A dataset that is independent of the training dataset, but follows the same probability distribution

Data Normalization	The organization of data to appear similar across all records and fields
Data Standardization	The process of converting data to a common format suitable for analysis by users
Data Preprocessing	Data mining technique used to transform the raw data in a useful and efficient format
Bias	Describes how well a model matches the training set
Confusion Matrix	A table that is used to define the performance of a classification algorithm
Transfer Learning	Taking the relevant parts of a pre-trained ML model and applying it to a new problem
Depth	Number of layers in ML model
Group	Group of instances that the miner should solve before publishing a new block
Heap	All instances that are not included in any group yet
Package	Successfully created group
Model	A decision process in an abstract manner
Framework	A tool that provides ready-made components or solutions that are customized in order to speed up development
Cloud Service	Refers to a wide range of services delivered on demand to companies and customers over the internet
Relative Bound	Difference between upper and lower bound of a solution relative to the upper bound value
Scaling	A technique to standardize the independent features present in the data in a fixed range
Oversampling	A technique that creates synthetic samples by randomly sampling the characteristics from occurrences in the minority class
Neuron	A connection point in an ANN
Activation Function	Decides whether a neuron should be activated or not
Layer	In a DL model a structure or network topology in the model's architecture
Hidden Layer	A layer in between input layer and output layer
Dropout	A regularization method that approximates training a large number of ANNs with different architectures in parallel
Output Layer	The last layer of neurons that produces given outputs for the ANN
Input Layer	Brings the initial data into the ANN for further processing by subsequent layers of artificial neurons
Regression	A technique for investigating the relationship between independent variables or features and a dependent variable or outcome
Classification	A supervised learning concept which basically categorizes a set of data into classes
Pool	Place to store unprocessed objects
Hierarchical Model	A model in which lower levels are sorted under a hierarchy of successively higher-level units

References

1. Ball, M.; Rosen, A.; Sabin, M.; Vasudevan, P.N. Proofs of Useful Work. IACR Cryptology ePrint Archive. Last update 2021. Available online: https://eprint.iacr.org/2017/203.pdf (accessed on 1 April 2021).
2. Shibata, N. Proof-of-search: Combining blockchain consensus formation with solving optimization problems. *IEEE Access* **2019**, *7*, 172994–173006. [CrossRef]
3. Fitzi, M.; Kiayias, A.; Panagiotakos, G.; Russell, A. Ofelimos: Combinatorial Optimization via Proof-of-Useful-Work\A Provably Secure Blockchain Protocol. IACR Cryptology ePrint Archive. 2021. Available online: https://eprint.iacr.org/2021/1379.pdf (accessed on 28 January 2021).
4. Todorović, M.; Matijević, L.; Ramljak, D.; Davidović, T.; Urošević, D.; Jakšić-Krüger, T.; Jovanović, Đ. Proof-of-Useful-Work: BlockChain Mining by Solving Real-life Optimization Problems. *Symmetry* **2022**, *14*, 1831. [CrossRef]
5. Haouari, M.; Mhiri, M.; El-Masri, M.; Al-Yafi, K. A novel proof of useful work for a blockchain storing transportation transactions. *Inf. Process. Manag.* **2022**, *59*, 102749. [CrossRef]
6. Li, W. Adapting Blockchain Technology for Scientific Computing. *arXiv* **2018**, arXiv:1804.08230.

7. Loe, A.F.; Quaglia, E.A. Conquering generals: An NP-hard proof of useful work. In Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, Munich, Germany, 15 June 2018; pp. 54–59.
8. Syafruddin, W.A.; Dadkhah, S.; Köppen, M. Blockchain Scheme Based on Evolutionary Proof of Work. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 771–776.
9. Chin, Z.H.; Yap, T.T.V.; Tan, I.K. Simulating the adjustment of the difficulty in blockchain with SimBlock. In Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure, Taipei, Taiwan, 6 October 2020; pp. 192–197.
10. Feng, W.; Cao, Z.; Shen, J.; Dong, X. RTPoW: A Proof-of-Work Consensus Scheme with Real-Time Difficulty Adjustment Algorithm. In Proceedings of the 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), Beijing, China, 14–16 December 2021; pp. 233–240.
11. Hutter, F.; Hamadi, Y.; Hoos, H.H.; Leyton-Brown, K. Performance prediction and automated tuning of randomized and parametric algorithms. In Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming—CP 2006, Nantes, France, 25–29 September 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 213–228.
12. Smith-Miles, K.; Lopes, L. Measuring instance difficulty for combinatorial optimization problems. *Comput. Oper. Res.* **2012**, *39*, 875–889. [CrossRef]
13. Baldominos, A.; Saez, Y. Coin. AI: A proof-of-useful-work scheme for blockchain-based distributed deep learning. *Entropy* **2019**, *21*, 723. [CrossRef]
14. Lihu, A.; Du, J.; Barjaktarevic, I.; Gerzanics, P.; Harvilla, M. A Proof of Useful Work for Artificial Intelligence on the Blockchain. *arXiv* **2020**, arXiv:2001.09244.
15. Li, B.; Chenli, C.; Xu, X.; Shi, Y.; Jung, T. DLBC: A Deep Learning-Based Consensus in Blockchains for Deep Learning Services. *arXiv* **2020**, arXiv:1904.07349v2.
16. Schröer, C.; Kruse, F.; Gómez, J.M. A systematic literature review on applying CRISP-DM process model. *Procedia Comput. Sci.* **2021**, *181*, 526–534. [CrossRef]
17. Alipour, H.; Mu noz, M.A.; Smith-Miles, K. Enhanced instance space analysis for the maximum flow problem. *Eur. J. Oper. Res.* **2023**, *304*, 411–428. [CrossRef]
18. Strassl, S.; Musliu, N. Instance space analysis and algorithm selection for the job shop scheduling problem. *Comput. Oper. Res.* **2022**, *141*, 105661. [CrossRef]
19. Jooken, J.; Leyman, P.; De Causmaecker, P. A new class of hard problem instances for the 0–1 knapsack problem. *Eur. J. Oper. Res.* **2022**, *301*, 841–854. [CrossRef]
20. Piechowiak, K.; Drozdowski, M.; Sanlaville, É. Framework of algorithm portfolios for strip packing problem. *Comput. Ind. Eng.* **2022**, *172*, 108538. [CrossRef]
21. Pinedo, M.L. *Scheduling: Theory, Algorithms, and Systems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
22. Mrad, M.; Souayah, N. An Arc-Flow Model for the Makespan Minimization Problem on Identical Parallel Machines. *IEEE Access* **2018**, *6*, 5300–5307. [CrossRef]
23. Ostojić, D.; Davidović, T.; Jakšić Kruger, T.; Ramljak, D. Comparative Analysis of Heuristic Approaches to P||Cmax. In Proceedings of the 11th International Conference on Operations Research and Enterprise Systems, ICORES 2021, Online Streaming, 3–5 February 2022; pp. 352–359.
24. Dwork, C.; Naor, M. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*; Springer: Cham, Switzerland, 1992; pp. 139–147.
25. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://nakamotoinstitute.org/bitcoin/ (accessed on 29 November 2019).
26. Garay, J.; Kiayias, A.; Leonardos, N. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Cham, Switzerland, 2015; pp. 281–310.
27. Pass, R.; Seeman, L.; Shelat, A. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Cham, Switzerland, 2017; pp. 643–673.
28. Meshkov, D.; Chepurnoy, A.; Jansen, M. Short paper: Revisiting difficulty control for blockchain systems. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*; Springer: Cham, Switzerland, 2017; pp. 429–436.
29. Noda, S.; Okumura, K.; Hashimoto, Y. An Economic Analysis of Difficulty Adjustment Algorithms in Proof-of-Work Blockchain Systems. 2019. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3410460 (accessed on 20 September 2022).
30. Aggarwal, V.; Tan, Y. A Structural Analysis of Bitcoin Cash's Emergency Difficulty Adjustment Algorithm. 2019. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3383739 (accessed on 20 September 2022).
31. Zhang, S.; Ma, X. A general difficulty control algorithm for proof-of-work based blockchains. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 3077–3081.
32. Zheng, K.; Zhang, S.; Ma, X. Difficulty prediction for proof-of-work based blockchains. In Proceedings of the 2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Atlanta, GA, USA, 26–29 May 2020; pp. 1–5.
33. Chin, Z.H.; Yap, T.T.V.; Tan, I.K.T. Genetic-Algorithm-Inspired Difficulty Adjustment for Proof-of-Work Blockchains. *Symmetry* **2022**, *14*, 609. [CrossRef]

34. Hutter, F.; Xu, L.; Hoos, H.H.; Leyton-Brown, K. Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.* **2014**, *206*, 79–111.
35. Cook, W. Concorde TSP Solver. 2016. Available online: http://www.math.uwaterloo.ca/tsp/concorde.html (accessed on 1 August 2022).
36. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; Karimi-Mamaghan, A.M.; Talbi, E.G. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *Eur. J. Oper. Res.* **2022**, *296*, 393–422. [CrossRef]
37. Vaughan, D. *Analytical Skills for AI and Data Science: Building Skills for an AI-Driven Enterprise*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2020.
38. Allen, M.; Leung, R.; McGrenere, J.; Purves, B. Involving domain experts in assistive technology research. *Univers. Access Inf. Soc.* **2008**, *7*, 145–154. [CrossRef]
39. Janssens, J. *Data Science at the Command Line*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2021.
40. Oymak, S.; Li, M.; Soltanolkotabi, M. Generalization guarantees for neural architecture search with train-validation split. In Proceedings of the 38th International Conference on Machine Learning, Virtual, 18–24 July 2021; PMLR: London, UK, 2021; pp. 8291–8301.
41. Saunshi, N.; Gupta, A.; Hu, W. A Representation Learning Perspective on the Importance of Train-Validation Splitting in Meta-Learning. In Proceedings of the 38th International Conference on Machine Learning, Virtual, 18–24 July 2021; PMLR: London, UK, 2021; pp. 9333–9343.
42. Patro, S.; Sahu, K.K. Normalization: A preprocessing stage. *arXiv* **2015**, arXiv:1503.06462.
43. Singh, D.; Singh, B. Investigating the impact of data normalization on classification performance. *Appl. Soft Comput.* **2020**, *97*, 105524. [CrossRef]
44. Bhanja, S.; Das, A. Impact of data normalization on deep neural network for time series forecasting. *arXiv* **2018**, arXiv:1812.05519.
45. Hou, Z.; Hu, Q.; Nowinski, W.L. On minimum variance thresholding. *Pattern Recognit. Lett.* **2006**, *27*, 1732–1743. [CrossRef]
46. Khalid, S.; Khalil, T.; Nasreen, S. A survey of feature selection and feature extraction techniques in machine learning. In Proceedings of the 2014 Science and Information Conference, London, UK, 7–29 August 2014; pp. 372–378.
47. Cai, J.; Luo, J.; Wang, S.; Yang, S. Feature selection in machine learning: A new perspective. *Neurocomputing* **2018**, *300*, 70–79. [CrossRef]
48. Hall, M.A. Correlation-Based Feature Selection for Machine Learning. Ph.D. Thesis, The University of Waikato, Hamilton, New Zealand, 1999.
49. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
50. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
51. Novaković, J.D.; Veljović, A.; Ilić, S.S.; Papić, Ž.; Milica, T. Evaluation of classification models in machine learning. *Theory Appl. Math. Comput. Sci.* **2017**, *7*, 39–46.
52. Reich, Y.; Barai, S. Evaluating machine learning models for engineering problems. *Artif. Intell. Eng.* **1999**, *13*, 257–272. [CrossRef]
53. Jiao, Y.; Du, P. Performance measures in evaluating machine learning based bioinformatics predictors for classifications. *Quant. Biol.* **2016**, *4*, 320–330. [CrossRef]
54. Haghighi, S.; Jasemi, M.; Hessabi, S.; Zolanvari, A. PyCM: Multiclass confusion matrix library in Python. *J. Open Source Softw.* **2018**, *3*, 729. [CrossRef]
55. Marom, N.D.; Rokach, L.; Shmilovici, A. Using the confusion matrix for improving ensemble classifiers. In Proceedings of the 2010 IEEE 26th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, 17–20 November 2010; pp. 000555–000559.
56. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for hyper-parameter optimization. *Adv. Neural Inf. Process. Syst.* **2011**, *24*. Available online: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf (accessed on 1 August 2022).
57. Feurer, M.; Hutter, F. Hyperparameter optimization. In *Automated Machine Learning*; Springer: Cham, Switzerland, 2019; pp. 3–33.
58. Yang, L.; Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* **2020**, *415*, 295–316. [CrossRef]
59. Shawki, N.; Nunez, R.R.; Obeid, I.; Picone, J. On Automating Hyperparameter Optimization for Deep Learning Applications. In Proceedings of the 2021 IEEE Signal Processing in Medicine and Biology Symposium (SPMB), Philadelphia, PA, USA, 4 December 2021; pp. 1–7.
60. Coffman, E.G., Jr.; Garey, M.R.; Johnson, D.S. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-Hard Problems*; PWS Publishing Company: Boston, MA, USA, 1996; pp. 46–93.
61. Pisinger, D. *Algorithms for Knapsack Problems*; DIKU rapport 95/1; University of Copenhagen: Copenhagen, Denmark, 1995.
62. Davidović, T.; Šelmić, M.; Teodorović, D.; Ramljak, D. Bee colony optimization for scheduling independent tasks to identical processors. *J. Heuristics* **2012**, *18*, 549–569. [CrossRef]
63. Graham, R.L. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* **1969**, *17*, 416–429. [CrossRef]
64. Yamashiro, H.; Nonaka, H. Estimation of processing time using machine learning and real factory data for optimization of parallel machine scheduling problem. *Oper. Res. Perspect.* **2021**, *8*, 100196. [CrossRef]
65. Flach, P. Performance evaluation in machine learning: The good, the bad, the ugly, and the way forward. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 9808–9814.

66. Lorenzo-Seva, U. *How to Report the Percentage of Explained Common Variance in Exploratory Factor Analysis*; Department of Psychology: Tarragona, Italy, 2013.
67. Shelke, M.S.; Deshmukh, P.R.; Shandilya, V.K. A review on imbalanced data handling using undersampling and oversampling technique. *Int. J. Recent Trends Eng. Res* **2017**, *3*, 444–449.
68. Yang, J.B.; Shen, K.Q.; Ong, C.J.; Li, X.P. Feature selection for MLP neural network: The use of random permutation of probabilistic outputs. *IEEE Trans. Neural Netw.* **2009**, *20*, 1911–1922. [CrossRef]
69. Song, F.; Guo, Z.; Mei, D. Feature selection using principal component analysis. In Proceedings of the 2010 International Conference on System Science, Engineering Design and Manufacturing Informatization, Yichang, China, 12–14 November 2010; Volume 1, pp. 27–30.
70. Cheng, Q.; Varshney, P.K.; Arora, M.K. Logistic regression for feature selection and soft classification of remote sensing data. *IEEE Geosci. Remote Sens. Lett.* **2006**, *3*, 491–494. [CrossRef]
71. Abdel-Gawad, A.; Ratner, S. Adaptive Optimization of Hyperparameters in L2-Regularised Logistic Regression; Technical report; 2007. Available online: http://cs229.stanford.edu/proj2007/AbdelGawadRatner-AdaptiveHyperparameterOptimization.pdf (accessed on 1 August 2022).
72. Khadem, H.; Eissa, M.R.; Nemat, H.; Alrezj, O.; Benaissa, M. Classification before regression for improving the accuracy of glucose quantification using absorption spectroscopy. *Talanta* **2020**, *211*, 120740. [CrossRef]