Computing fundamentals project
A classifier hyper-heuristic approach for the firefighter problem

Team #04

A01169881, Omar Rodrigo Muñoz Gómez
A01222528, Leonardo Francisco Garrafa Pacheco

2022-06-15

Abstract

The firefighter problem is a mathematical representation of the spread of a fire through a network. It also models the action of the firefighters whose job is to protect the network from burning. This model can be extrapolated to other domains, for instance, to model the spread of a virus in a computer network or the propagation of a disease among a population. The number of firefighters in the network affects the ability to contain the fire, the case of having one firefighter only is considered in this work. We propose using a selection hyper-heuristic approach based on a classifier to select the nodes the firefighter needs to protect next. Our hyper-heuristic chooses the best heuristic to use at each time step of the process among two possible heuristics representing the classes of the classifier. The heuristics used are: "LDEG" and "GDEG". Throughout this work, we used a pool of 260 different problem instances where 70% of the data is used to train the classifier and 30% is used to test its performance against the standalone heuristics and other approaches from the literature. The results of this work indicate that the features used to train the three different classifier proposals are insufficient to determine a decision boundary between the usage of one heuristic or the other. This behavior is mainly due to a high correlation in the features and a high class imbalance, so the hyper-heuristic behaves just like the "LDEG" heuristic. The current model implementation saves, on average, 9.84 nodes less than the best of the solutions in the comparison.

1 Introduction

The firefighter problem was introduced, by Hartnell [6], as an attempt to model firefighting (or virus control) on a network. However, due to the deterministic discrete-time nature of the model, it can also serve as a simple model of the spread of some malady on a network. e.g. the propagation of malware on a computer network or the propagation of an epidemic on a population.

The firefighter problem is difficult. It has been proved to be NP-complete for bipartite graphs [11] and cubic graphs [9]. However, it is solvable in polynomial time for graphs of maximum degree three when the fire starts at a vertex of degree two.

Besides its complexity, different objectives can be pursued in a firefighter problem, which can get in conflict [3]. To mention a few examples:

- Save the maximum possible number of vertices.
- Put the fire out as quickly as possible.
- Determine whether a set of vertices can be prevented from burning.

- Find the smallest number of firefighters required to contain the fire.

Furthermore, there can be different variations of the problem. For example, the temporal firefighter problem, in which the original problem is extended by the addition of civilians that help to maintain firebreaks [5]. Making the problem even harder.

Nonetheless, the firefighter problem can reach several practical applications. Including communication network security, containment of floods, disease control, and so on. This relevance in real-world scenarios is what makes the firefighter problem a focus of attention for researchers. Particularly for us, we want to tackle this problem with the use of hyper-heuristics and explore whether our model is capable to solve this problem efficiently.

Mathematically, the firefighter problem represents a fire spreading through the nodes of a graph and the action of the firefighters to control it, and the goal is to save the many nodes as possible. Different authors have proposed and studied a variety of solutions. For example, Hartell [7] studied the performance of a greedy algorithm, which always saved more than half of the nodes that an optimal algorithm could save. More recently, Michalak [10] proposed the SimX crossover operator, which was significantly better than other ten crossovers used for comparison. However, as far as we know, there are no hyper-heuristics applied to this domain.

Another motivation we had to explore the use of hyper-heuristics in the firefighter problem comes from Garcia's and Hu's works. Garcia et al. [4] suggest the use of different heuristics for different objectives and outline the strategy to select them according to these objectives, which, to some extent, resembles the process of designing a hyper-heuristic. Hu et al. [8] suggest that a change in the solution representation can improve the performance of an algorithm. In this sense, the fact that hyper-heuristics works in the heuristic space (instead of the solution space) acts as a spur for us to perform this work too.

The remainder of the document is organized as follows. Section 2 presents an overview of different elements related to our research. Afterward, Section 3 briefly presents the proposed solution approach by describing how it operates. We then move on to summarizing the training and testing we carried out for our model as well as the results, performance comparisons against related work, and some discussion in Section 4. We wrap up our manuscript by laying out our conclusions and future works in Section 5.

2 Background and Related Work

2.1 The Firefighter Problem

The firefighter problem, originally introduced by Hartnell in [6], is a mathematical representation of the spread of a fire and the action of the firefighters in order to control it. In [7] Hartnell and Qiyan describe the problem as follows:

> A fire (or virus) breaks out at a vertex of a graph. The firefighter (or defender) then chooses any vertex not yet on fire (or affected by the virus) to protect. The fire and the firefighter then alternate moves on the graph. Once a vertex has been chosen by the firefighter, it is considered protected or safe from any future moves of the fire. After the firefighter's move, the fire makes its move by spreading to all vertices which are adjacent to the ones on fire, except for those that are protected. The firefighter and the fire alternate taking turns until the fire can no longer spread. The object of the firefighter is to save the maximum number of vertices by the time the firefighting process ends. The general problem is for F fires to start at a set of F vertices of the graph and then D firefighters to each choose a vertex not yet affected by the fires to protect.

To be more precise, the firefighting process is modeled as a discrete-time process where the vertices in the graph can be in either one of three sets: 'burned', 'not burned', or 'saved'. When the fire can no longer spread, vertices in the 'not burned' set are moved to the 'saved' set.

As mentioned before, there can be different variations of the problem found in the literature. For example, the temporal

firefighter problem, in which the original problem is extended by the addition of civilians that help to maintain firebreaks [5]. The former was proved to make the problem even harder. But, for the goal of this work, we are going to work on the simplest version of the firefighter problem, where the fire starts in a single node and only one firefighter is available to defend the nodes.

This simplest version of the firefighter problem can be formally defined on an undirected graph. Let $G = (V, E)$ be an undirected graph with size $|V| = n$, where each node $v \in V$ is flagged as untouched. At step $t = 0$, a fire starts in a predefined node $s \in V$, which is flagged as burnt. Then, for each subsequent step $t > 0$, the next two steps are performed in order:

- *Protection Phase*: A firefighter D protects a node $v \in V$ that is not burnt. Thus, tagged as protected.
- *Spreading Phase*: Every untouched node adjacent to a burnt node starts burning. Thus, tagged as burnt.

This process continues until no new node can be burned in the next step.

Even though we are working with the simple version of the problem, it can serve as a simple model that can reach several practical applications including communication network security, containment of floods, and disease control.

Multiple authors have tried to give solutions to the firefighter problem using different techniques. A lot of the work done has been on the theoretical side [2], that is, proving that a given conjecture is true for specific types of graphs. Many of the solution proposals are approximation algorithms and in some cases simplifications of the problem, for instance, considering the graph is a tree. Hartnell and Qiyan [7] showed that using a greedy algorithm (saving the vertex that maximizes the number of vertices saved) can save more than half of the vertices the optimal algorithm saves for F $= 1$ (fire starting at a single vertex) on trees, and that the optimal algorithm outperforms the greedy algorithm only if there exists at least one greedy move that is the ancestor of some optimal move.

From a practical point of view, Blum et al. [2] proposed the usage of Ant Colony Optimization (ACO) algorithms and compared them to a linear programming approach making use of the CPLEX optimizer. A benchmark was done for different graph densities, graph sizes, and number of firefighters. Since this problem set is varied, it has been used by other authors [8, 4, 13] to benchmark and compare their solution proposals. Using this framework, it was found that CPLEX obtains the best results for small graphs of around 50 to 100 vertices, while a hybrid ACO (HyACO) algorithm outperforms CPLEX and the basic ACO algorithm for larger graphs. Hu et al. [8] proposed a Variable Neighborhood Search (VNS) to decide which vertices should be protected at each timestep. They tested their solution proposal on the same benchmark used by Blum et al. in [2]. Their results show that their approach is able to save a similar amount of vertices as the HyACO algorithm proposed in [2].

García et al. [4], attempted to find a good solution strategy for the general case by developing and comparing nine integer linear programming (ILP) techniques and six heuristic methods. For their comparison, a set of random graphs with vertices ranging from 50 to 1000 was created, and their results show that ILP techniques generally provide the best results. However, if the longest path in the graph is high (greater than 20), the ILP techniques require large computational resources which make the heuristic methods more advantageous in such cases. From the six heuristics studied, the ones with the best performance were "ThDesc,Now" and "ThDegree,Now". For some graph sizes, these heuristics outperform some of the ILP techniques. One interesting finding in this work is that as the number of firefighters increases, there is no significant difference between the ILP and heuristic methods. Suggesting that the real challenge is to be found in the case of a few firefighters.

More recently, Ramos et al. [13] developed a *matheuristic* approach that consists of a constructive phase and a local search phase, the implementation of the former uses a greedy heuristic inspired by the greedy randomized adaptive search procedure (GRASP), and the latter is done solving an ILP model. Their model was evaluated using the frameworks described in [2, 8] (referred to BBGRL) and [4] (referred to GBRL), this allows for a comparison of the matheuristic approach with previous work. The BBGRL set is used to compare to the algorithms in [2, 8] and the GBRL set is used to compare to the algorithms in [4]. The experimental results show that the matheuristic and the VNS [8] have the best performance in the BBGRL problem set (without a significant statistical difference) followed by HyACO, CPLEX and ACO. For the GBRL set, the matheuristic and the ILP(L,T+1) algorithm have the best performance (without a significant statistical difference).

2.2 Hyper-heuristics

Most of the research found in the literature approaches the problem from a theoretical point of view [8]. Furthermore, there are no hyper-heuristics applied to the domain as far as we know. However, we get significant insight from Garcia's and Hu's works that hyper-heuristics could have a good performance in this field.

In the work of Garcia et al. [4], they study the next heuristics:

- **Degree:** save the vertex with the highest degree.
  The idea is to protect the vertex that would spread the fire the most.
- **Desc:** save the vertex with the maximal number of descendants.
  Defending this node, as an entry point for the largest subgraphs, will protect the maximal number of vertices. However, for graphs that are not trees, the nodes could be reached by the fire from other paths.
- **Threat:** focus on saving the nodes that would be reached by the fire in the next step. However, this heuristic seems to not be optimal for general graphs.
- **Now vs. Later:** this considers the idea to prioritize nodes that should be defended first, such as Degree and Desc, but more intelligent. For example, let some nodes burn to protect other nodes that would require more available firefighters in the next steps.

They suggest the use of those heuristics for different objectives and outline the strategy to select them according to these objectives, which, to some extent, resembles the process of designing a hyper-heuristic. However, they did not test this strategy, so it is unclear what the performance of such a strategy would be.

Hu et al. [8] introduced a new more compact solution representation for the problem, which proved to have better results than the existing approaches at that time. The representation consisted of a vector where each of its positions represents a node in the graph, to be combined with a variable neighborhood search method. This work suggests that a change in the solution representation can improve the performance of an algorithm. In this sense, the fact that hyper-heuristics works in the heuristic space, instead of the solution space itself, motivate to study the performance of hyper-heuristics over this domain.

Rule Based Hyper-Heuristic Models
We want to introduce the notion of ruled-based hyper-heuristic models because our approach belongs to this kind of hyper-heuristics.

This is well described by Sánchez et al. [14]: "In this kind of hyper-heuristics, a set of rules guides the decisions of the solver that will be employed. One way to represent such rules is in the form of a selection matrix, where each row represents a different rule. So, the rule most similar to the current problem state is used."

2.3 Machine Learning Classifiers

In the area of supervised machine learning, there is a class of models used to solve classification problems, which are known as classifiers. The goal of a classifier model is to take an input vector x and to assign it to one of K discrete classes C_k where $k = 1, ..., K$ [1].

For this work, due to its popularity and good performance, we are going to be working with the next three classifiers (description obtained from `scikit-learn.org` [12]):

1. *Decision Tree Classifier*: This classifier works by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.
2. *Multilayer Perceptron Classifier*: This classifier implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation. An MLP is a neural network model.

3. *Linear Discriminant Classifier*: This classifier uses a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.

For the application of those classifiers, we are going to be using the Scikit-learn library [12], which is an open-source machine learning python library that supports supervised and unsupervised learning.

3 Solution Approach

This section is going to be divided into two parts. First, we are going to provide the details of our solution model; and second, we are going to detail the approach we used to generate the database to train our model.

Solution Model

Our model is a ruled-based model of hyper-heuristic. First, we characterize a given problem P by getting its features, and then, we use those features as input for a classifier. Then, the classifier will try to predict the best heuristic h^* to apply to get the best results. After applying h^* to P, we get a new problem state P' in which we repeat the same steps.

Figure 1 shows the process.

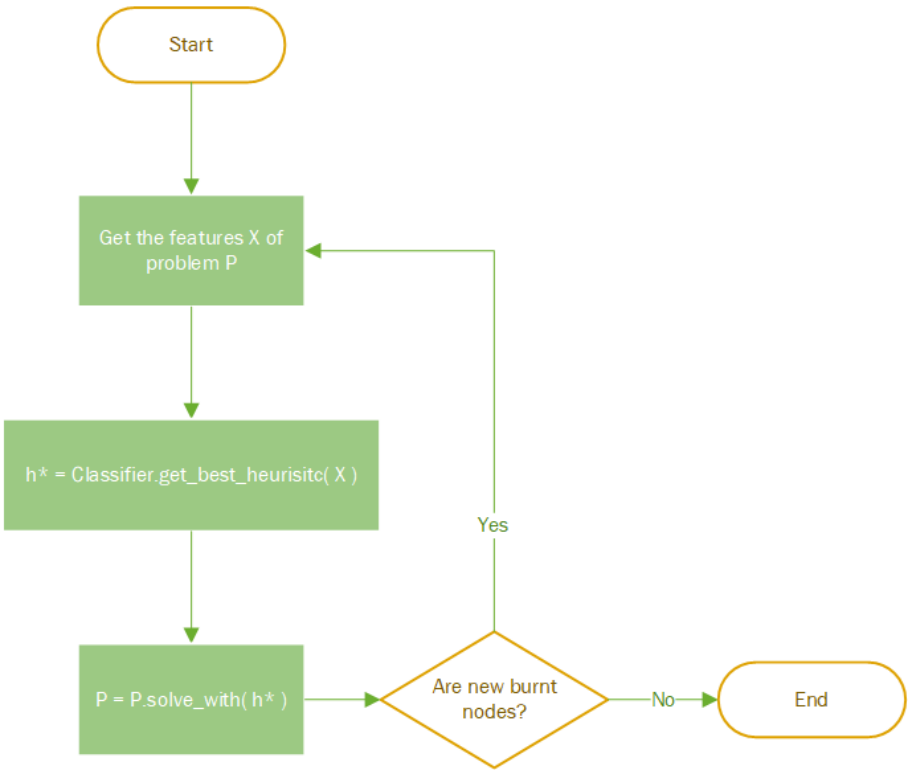

Figure 1: Hyper-heuristic model process.

Database Generation

To construct a database to train the classifier, we need a set of input data associated with a class. In our case, the input data is the features of the problem, and the class is a heuristic, which should be the best heuristic to apply. We determine which is the best heuristic, by running an iterative process applying the next two operations:

1. *Performance index evaluation*. In this operation, we simulate the spread of the fire for k steps into the future after applying a certain heuristic h to the problem state P. We repeat that with each available heuristic. Then, the

heuristic with the lowest number of burning nodes after k steps is the best heuristic to apply for the problem state P. We can name this heuristic as h^*.

2. *Firefighter problem solver*. In this operation, we compute the current features X of the problem state P, and create a new data point for our database, by joining X and h^*. Additionally, the problem P is evolve to P' by performing one step (without applying any heuristic). After that, we repeat *Performance index evaluation* and *Firefighter problem solver* over P'.

Figure 2 shows the process.

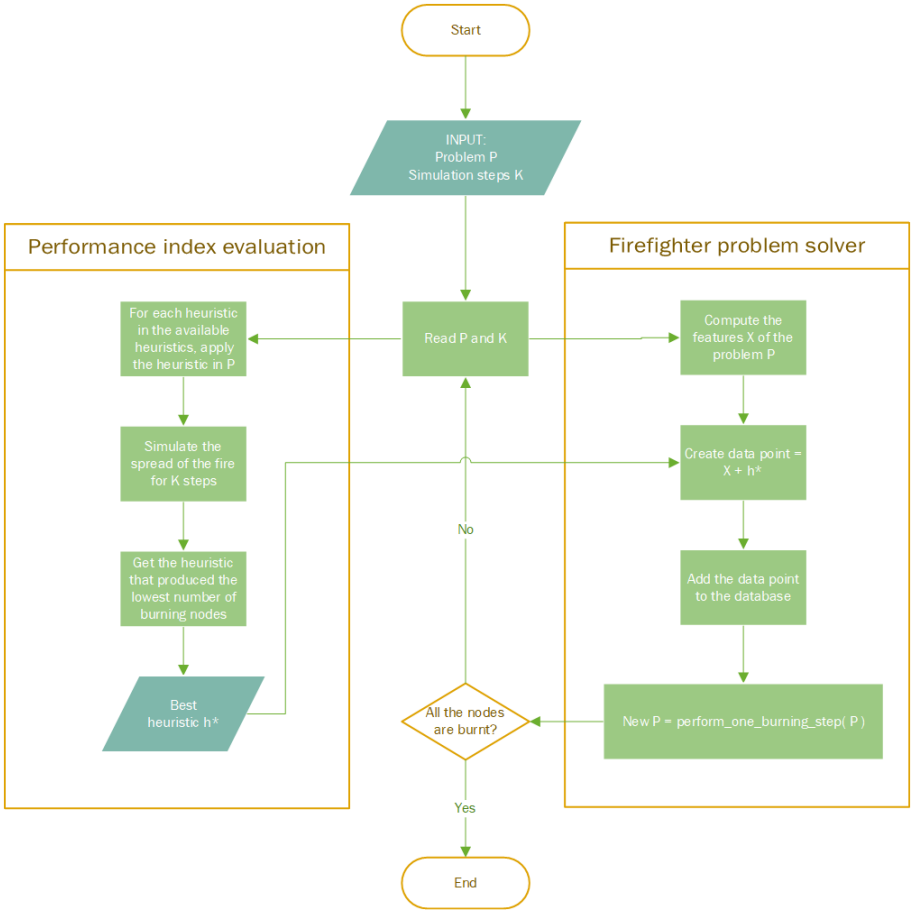

Figure 2: Database generation process.

3.1 The Set of Heuristics

You are not requested to include information about the heuristics since it will be later included by the instructors.

3.2 Problem Characterization

You are not requested to include information about the features to characterize the instances since it will be later included by the instructors.

3.3 Problem Instances

You are not requested to include information about the problem instances since it will be later included by the instructors.

4 Experiments and Results

This section presents the results of the experiments to build the hyper-heuristic model and compare its performance with the standalone heuristics. Section 4.1 presents the results of the database creation for different values of the parameter K in the performance index discussed earlier. Then, Section 4.2 presents the results and performance of the different classifiers used to choose the proper heuristic according to the features of a given problem. Section 4.3 presents a comparison of the heuristics against the proposed hyper-heuristic. Finally, Section 4.4 compares the performance of our approach against the related work in [8].

4.1 Database creation for the classifier

Our work uses a pool of 260 different problem instances generated based on the work by [8, 4]. We reserved 70% of the data for training the classifier and 30% for testing our hyper-heuristic against each heuristic. We ensured that we got a uniform distribution of the different problem instance types in each data split. The possible types of problem instances are discussed in Section 3.3.

Using problems in the training set, we evaluated each heuristics' performance as the fire spreads through the graph. The performance index used to evaluate each heuristic is discussed in detail in Section 3. Since the fire spreads at discrete time steps, the problem and its defining characteristics (or features) evolve with each iteration. Because of this, we evaluate the performance index at each time step. As a result, a single problem instance generates multiple data points, where a data point represents the state (in terms of features) of a given problem at a given times tep and the corresponding minimizing heuristic. Figure 2 illustrates this process; we feed a problem from the training database to the "Performance index evaluation" block, which determines the best heuristic for the problem; this becomes the class of the problem. In parallel, we feed the problem to the "Firefighter problem solver," who is in charge of computing the problem features and evolving the problem by spreading the fire to the connected nodes. Finally, we feed the new problem state back to the process until all nodes are burnt. In each time step, a new data point is generated, which creates the database used for training the classifier.

After performing the steps described in Figure 2, the database used for the classifier consists of 1,564 data points where 97% belong to the class "LDEG" and only 3% belong to the "GDEG" class. It is important to recall that the performance index looks at K steps into the future and averages the number of burning nodes in each simulated step; this means that the parameter K affects the resulting class of the problem instance. By increasing the parameter K, we saw an increment of the data points classified as "GDEG." Since there is a significant class imbalance ("LDEG" appears to be better than "GDEG" for most cases), we used a high number for K to mitigate this issue. The class distribution mentioned above results from using the performance index with $K = 15$.

4.2 Classifier

Once the database is created, we can continue with the creation of a classifier. In this work we considered three different classifier models: decision tree, multilayer perceptron, and linear discriminant. Each of the models was fitted using a 4-fold stratified cross-validation approach (to deal with the class imbalance problem) with a grid search for the best parameters. The classifier database from Figure 2 was divided into training (75%) and validation (25%) and the final performance of the models is reported based on the validation set. All of the models were trained and evaluated using Python, Sci-kit learn and Keras. The results for each of the classifiers is shown in Table 1.

Notice that the performance of the three classifiers is the same. Based on the accuracy, the classifiers perform an

Classifier	Accuracy
Decision tree	97%
Multilayer perceptron	97%
Linear discriminant	97%

Table 1: Performance comparison between three different classifier models

excellent job. However, considering the high class imbalance, we see this is not necessarily true. All three classifiers could not correctly classify any of the "GDEG" data points in the validation set. We expected to see that the more complex models were able to find the decision boundary between both classes better even under a significant class imbalance. However, studying the database shows two problems that make this task very difficult for all of the proposed classifiers. The first issue is the correlation between the features. Figure 3 shows the correlation matrix for the five features proposed in Section 3.2. This high correlation suggests that two of the five features may be removed without significantly affecting the classifier's performance. Additionally, a mistake was made in computing the features BURNING_EDGES and NODES_IN_DANGER. Thus, the feature is wrong for particular problem types, making it even harder for the classifier to find a decision boundary. Since the features seem to be the problem, we proposed to extract a different set of latent features using principal component analysis (PCA). We discovered that the first two principal components explain around 91% of the variance in the data. Figure 4 shows the first two principal components. The second issue we found is that not even with the principal components shown in Figure 4, it is possible to visualize a clear division between problems in both classes. To solve this issue, we could try to either fix the features that are incorrectly computed, add more features that are not correlated with the existing ones, or change the performance index so that the class assignment is done differently. The last alternative seems the least promising since the performance index is something relevant for us.

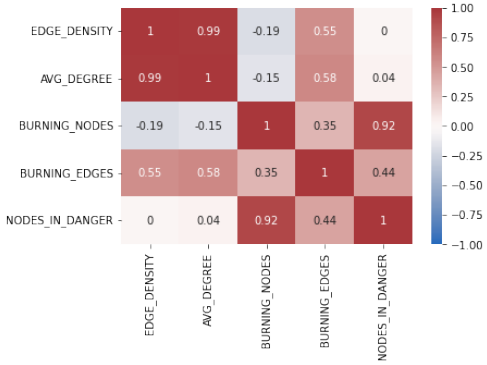

Figure 3: Correlation matrix between the five features in the database. Features AVG_DEGREE and EDGE_DENSITY are highly correlated as well as BURNING_NODES and NODES_IN_DANGER.

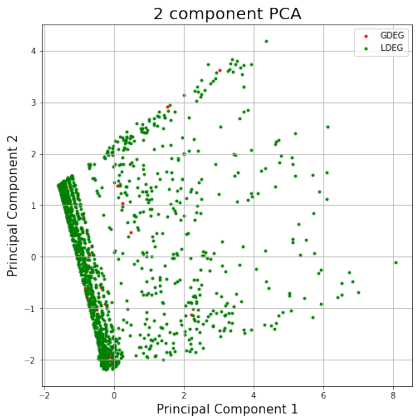

Figure 4: Principal component analysis for the classifier database. The two principal components shown are attributed 91% of the variance in the data. Problems in the GDEG class are shown in red and problems in LDEG are shown in green.

Based on these results, we can say that all the classifiers perform the same as all of them predict "LDEG" every time. Among these, we chose the decision tree due to its simplicity. Since the classifier predicts the same class every time, the reader can expect its performance, as shown in Sections 4.3 and 4.4, to be the same as the performance for the "LDEG" heuristic.

4.3 Performance of hyper-heuristics vs. heuristics

In this section, we benchmark the performance of the proposed classifier hyper-heuristic against the two heuristics described in Section 3.1. The performance is measured using the 30% testing data we saved before creating the classifier database, as described in Section 4.1. We present the results for different types of problems (Table 3) and the complete testing set (Table 2). For each problem type in Table 3, the testing dataset contained three instances, and the value shown is the percentage of burning nodes (average of three) after the fire stops spreading through the graph. The term

"Oracle" refers to the performance of the best heuristic for a given problem type. Since we are trying to minimize the percentage of nodes burnt, the lower the score, the better. For details about the different problem types, see Section 3.3.

GDEG	LDEG	Oracle	Classifier HH
0.95	0.94	0.94	0.94

Table 2: Average burning nodes (percentage) for all problem instances in the testing set for each solution model.

Group	GDEG	LDEG	Oracle	Classifier HH	Group	GDEG	LDEG	Oracle	Classifier HH
50_ep0.1	0.89	0.88	0.88	0.88	100_r0.219	0.94	0.94	0.94	0.94
50_ep0.15	0.93	0.93	0.93	0.93	500_ep0.015	0.99	0.99	0.99	0.99
50_ep0.2	0.93	0.93	0.93	0.93	500_ep0.02	0.99	0.99	0.99	0.99
50_ep0.25	0.95	0.95	0.95	0.95	500_ep0.025	0.99	0.99	0.99	0.99
50_r0.259	0.88	0.87	0.87	0.87	500_r0.071	0.95	0.95	0.95	0.95
50_r0.299	0.91	0.91	0.91	0.91	500_r0.083	0.96	0.97	0.96	0.97
50_r0.334	0.93	0.93	0.93	0.93	500_r0.093	0.97	0.97	0.97	0.97
100_ep0.05	0.95	0.95	0.95	0.95	1000_ep0.0075	0.99	0.99	0.99	0.99
100_ep0.075	0.96	0.96	0.96	0.96	1000_ep0.01	1.00	1.00	1.00	1.00
100_ep0.1	0.97	0.97	0.97	0.97	1000_ep0.0125	1.00	1.00	1.00	1.00
100_ep0.125	0.97	0.97	0.97	0.97	1000_r0.05	0.96	0.96	0.96	0.96
100_r0.169	0.91	0.62	0.62	0.62	1000_r0.058	0.97	0.97	0.97	0.97
100_r0.195	0.93	0.92	0.92	0.92	1000_r0.065	0.98	0.98	0.98	0.98

Table 3: Performance comparison between the heuristics, the oracle and the hyper-heuristic for the firefighter problems in the test set. The table is divided in groups of problem types depending on the characteristics of the problem. The number shown is the average burning nodes (percentage) for each problem type and solution model.

As mentioned in Section 4.2, due to problems in the features, the classifier hyper-heuristic behaves exactly the same as the "LDEG" heuristic. Table 3 shows that "LDEG" is consistently better for all problem types except for problems in the group "500_r0.083" where "GDEG" is marginally better. Problems in the groups "1000_ep0.01" and "1000_ep0.0125" appear to be the hardest of all the tested instances since for all the solution models, 100% of the nodes were burnt at the end of the process. Since the hyper-heuristic behaves just as the "LDEG" heuristic, it is also consistently better than "GDEG", still, it is unable to beat the Oracle because of the better performance of "GDEG" in the "500_r0.083" group. In principle, by combining the action of both heuristics, it is expected that a hyper-heuristic surpasses the performance of the individual heuristics. The issues with the data explained earlier do not allow the classifier hyper-heuristic to choose the "GDEG" heuristic in the small set of problems where it is better.

4.4 Performance against related work

In the previous section, we compared the hyper-heuristic with the heuristics from Section 3.1 in terms of the average number of nodes burnt at the end of the process (percentage-wise). However, in this section we have converted the percentage of burnt nodes to the actual number of nodes saved, with the purpose of making it comparable to the work done in [8]. Table 4 shows the comparison between the results obtained in [8] and the results in this work. The authors in [8] presented a Variable Neighborhood Search approach and compare it to the work in [2] where two different ant colony optimization (ACO) metaheuristic approaches were benchmarked against an integer programming method (CPLEX). The results shown in Table 4 for the algorithms CPLEX, ACO, HyACO, and VNS, are copied over from [8] for the case of one firefigher (D=1). Admittedly, the table shows only an approximate comparison between the two works since this work used only 3 problem instances (per group) to test the performance and [8] uses 10. Additionally, the results obtained in [8] are the result of running the solution models under a time constraint and our results are not. Still, with this

comparison we have a rough estimate of the performance of the heursitics described in Section 3.1 and the classifier hyper-heuristic presented in this work.

From Table 4 we see that the VNS metaheuristic approach proposed in [2] has the best overall performance for all the problem groups. If we consider the Oracle to be the best performing solution model for each group, the hyper-heuristic approach reviewed in this work saves on average 9.84 nodes less than the Oracle. Notice that this average gets significantly affected by the group "1000_ep0.0075" where all the solution models from the related work outperform the heuristics and hyper-heuristics in this work by a large margin.

Group	CPLEX	ACO	HyACO	VNS	GDEG	LDEG	Oracle	Classifier HH
50_ep0.1	7.4	7.4	7.4	7.3	5.3	6.0	7.4	6.0
50_ep0.15	4.5	4.5	4.5	4.5	3.3	3.3	4.5	3.3
50_ep0.2	3.1	3.1	3.1	3.1	3.3	3.3	3.3	3.3
100_ep0.05	9.2	9.1	9.2	9.1	5.3	5.3	9.2	5.3
100_ep0.075	5.4	5.4	5.4	5.4	4.0	4.0	5.4	4.0
100_ep0.1	3.9	3.9	3.9	3.9	3.3	3.3	3.9	3.3
500_ep0.015	7.6	7.5	7.8	7.6	5.0	5.0	7.8	5.0
500_ep0.02	5.3	5.2	5.6	5.7	4.3	4.0	5.7	4.0
500_ep0.025	4.2	4.4	4.3	4.5	4.0	4.0	4.5	4.0
1000_ep0.0075	105.2	107.4	107.8	108	6.0	6.0	108	6.0
1000_ep0.01	4.9	5.7	6	6.3	4.7	4.7	6.3	4.7
1000_ep0.0125	4	4.9	4.7	5.3	4.3	4.3	5.3	4.3

Table 4: Performance comparison with the work done in [8]. The Oracle column represents the best performance for each problem type.

4.5 Discussion

The results for the classifier hyper-heuristic presented in Section 4.3 show that the hyper-heuristic behaves just like the "LDEG" heuristic; the basis of a hyper-heuristic is to choose the most appropriate heuristic given a particular type of problem, and so we expected the results to be better than the individual heuristics. These results indicate that the hyper-heuristic is not behaving as expected. However, in Sections 4.1 and 4.2 we presented some of the issues we faced that could explain this performance. The main issues with our approach are the features selected to characterize the instances of a problem, and the high class imbalance we obtained since one of the chosen heuristics ("LDEG") appears to be consistently better than the other ("GDEG"). These two problems combined, make it difficult for the proposed classifiers to find a decision boundary between the two classes, that is, selecting the correct heuristic to use given a particular type of problem. We think that if we were to solve the issues with the data presented in Section 4.2 and we were to add additional heuristics, the approach could have better results than the individual heuristics. However, the results obtained in this work suggest that to take full advantage of this approach we must ensure that the selected features are not highly correlated. Additionally, we think that having more than two heuristics to choose from may create a better hyper-heuristic. Although it is unclear how many heuristics would be ideal, we think that the selection should be done based on the distribution of the classes resulting from the database creation process described in Figure 2. That is, if a heuristic has a low representation in the final class distribution, such heuristic may not be a good candidate to use in our hyper-heuristic approach, still, it may be considered as long as it is possible to draw a decision boundary around the classes.

The performance comparison shown in Section 4.4 identified our work saves on average 9.84 nodes less than the Oracle. Although the results of this work are admittedly lower than the related work, it is interesting to see that if we put the results as a percentage of the number of nodes in each problem type, the results are, on average, 2% worst than the Oracle. Considering that our approach behaves just like the "LDEG" heuristic, whose implementation is trivial, based on these results, we think that the "LDEG" heuristic may be a good solution model when this small change in performance is not critical. However, it was identified that this approach works poorly for problems in the group "1000_ep0.0075" compared to the related work. In this case, our work behaves around 10% worst than the Oracle. This group belongs to graphs with 1000 nodes and a low edge probability of 0.0075. We suspect that since the "LDEG" heuristic and the "GDEG" heuristics make a decision based on the largest degree of the graph, when dealing with graphs with low edge

probability, the heuristics do not work as good as in other cases. Notice that this group is the exception among the problem types with 1000 nodes, in general, our results conform to those in [8] where it was shown that for the case of only one firefighter, the large graphs are the most challenging.

In this work, we only tested the case for one firefighter. The results from [4] indicate that the problem's real challenge is for cases with few firefighters. However, if more firefighters were to be considered, our solution model would need to be retrained to assess the performance under the different testing conditions. Almost any tiny change in the framework would require us to update the database and thus retrain the classifier. Although this process can be pretty much automated, this is one of the drawbacks of our approach. Another possible drawback is the choice of the classifier. For complex models, for instance, deep neural networks, to find the decision boundary, the model could potentially contain many parameters that may or may not be supported by the running environment. Simpler models such as a decision tree could mitigate this issue, but depending on the feature space, this could imply a drop in the classification performance.

5 Conclusion and Future Work

In this work, we implemented a selection hyper-heuristic based on a classifier to solve the firefighter problem for the case of only one firefighter. That is, the classifier is designed to choose the best heuristic to use among "LDEG" and "GDEG" for each problem instance. The choice depends on the features of a given problem, and the classifier is trained using a labeled database of the best heuristic for different problems. We found that the classifier cannot find a decision boundary between the problems where the "GDEG" heuristic should be chosen above "'LDEG". We attribute this limitation to the implementation and choice of features and a high class imbalance in the generated dataset. The class imbalance indicates that the "LDEG" heuristic is better for a wide range of problems than the "GDEG" heuristic. However, this is dependent on the performance index used for the class assignment during labeling. Because of this limitation, the classifier hyper-heuristic behaves just like the "LDEG" heuristic for all problems.

When comparing the classifier performance against the related work, we found our current implementation saves, on average, 9.84 nodes less than the best of the related solution models (Oracle). However, expressing the performance as a percentage of the number of nodes in each problem type, we see that our approach is only 2% worst than the Oracle. These results suggest that the "LDEG" heuristic may be a good solution model for applications where such a performance drop can be tolerated, and the solution model needs to be simple.

Finally, we consider the classifier hyper-heuristic approach promising, given that the correct set of features is chosen, and the hyper-heuristic can choose among more heuristic approaches. However, this is an open question we could tackle in future work. The critical point for this approach to work is that the classes can be separated based on the features extracted for each problem type which implies that the solution model's most challenging part is getting appropriate data. Unfortunately, it is unknown if such a set of features and heuristics exist for this problem at the moment.

References

[1] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, January 2006.

[2] Christian Blum, Maria J. Blesa, Carlos García-Martínez, Francisco J. Rodríguez, and Manuel Lozano. The firefighter problem: Application of hybrid ant colony optimization algorithms. In Christian Blum and Gabriela Ochoa, editors, *Evolutionary Computation in Combinatorial Optimisation*, pages 218–229, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[3] Stephen Finbow and Gary Macgillivray. The firefighter problem: a survey of results, directions and questions. *The Australasian Journal of Combinatorics*, 2009.

[4] C. García-Martínez, C. Blum, F.J. Rodriguez, and M. Lozano. The firefighter problem: Empirical results on random graphs. *Computers & Operations Research*, 60:55–66, 2015.

[5] Samuel Hand, Jessica Enright, and Kitty Meeks. Making life more confusing for firefighters, 2022.

[6] Bert Hartnell. Firefighter! In *An application of domination. Presentation at the 25th Manitoba Conference on Combinatorial Mathematics and Computing, University of Manitoba, Winnipeg, Canada*, 1995.

[7] Bert Hartnell and Qiyan Li. Firefighting on trees: How bad is the greedy algorithm? *Congressus Numerantium*, 2000.

[8] Bin Hu, Andreas Windbichler, and Günther R. Raidl. A new solution representation for the firefighter problem. In Gabriela Ochoa and Francisco Chicano, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 25–35, Cham, 2015. Springer International Publishing.

[9] Andrew D. King and Gary MacGillivray. The firefighter problem for cubic graphs. *Discret. Math.*, 2010.

[10] Michalak Krzysztof. Simulation-based crossover for the firefighter problem. 2017.

[11] Gary MacGillivray and Ping Wang. On the firefighter problem. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 2003.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Natanael Ramos, Cid Carvalho de Souza, and Pedro Jussieu de Rezende. A matheuristic for the firefighter problem on graphs. *International Transactions in Operational Research*, 27(2):739–766, 2020.

[14] Melissa Sánchez, Jorge M. Cruz-Duarte, José carlos Ortíz-Bayliss, Hector Ceballos, Hugo Terashima-Marin, and Ivan Amaya. A systematic review of hyper-heuristics on combinatorial optimization problems. *IEEE Access*, 8:128068–128095, 2020.