Iterated Beam Search for Wildland Fire Suppression

Gustavo Delazeri(✉) and Marcus Ritt

Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
{gustavo.delazeri,marcus.ritt}@inf.ufrgs.br

Abstract. Wildfires cause significant damage costs globally, and it is likely that they are becoming more damaging due to climate change. Here we study methods for fire suppression, after a breakout of fire. In our model, we have a grid graph $G = (V, A)$ that represents the discretization of a terrain into cells and an ignition node $s \in V$ from which the fire spreads to other nodes. The spread of the fire is defined by the arc weights, which can be used to model important factors such as wind direction and vegetation type. At various points in time, one or more fire suppression resources become available to be applied to nodes in the graph that are not yet burned. Applying a resource to a node $v \in V$ adds a delay to the outgoing edges of v, which causes a local slowdown in fire propagation. The goal is to find an allocation of resources to the nodes of the graph such that the total burned area at a target time is minimized. In this work, we propose a heuristic algorithm based on beam search to tackle this problem. Our computational experiments show that our approach is able to consistently find the optimal solution to almost all instances used in literature, but in considerably less time than previous approaches.

Keywords: wildfire suppression · heuristic search · beam search

1 Introduction

Wildfires are estimated to have caused global damage costs of about USD 69 billion in 2018–2023 [8,12]. Their frequency and damage are likely to increase with climate change, with longer wildfire seasons, larger affected areas, and new locations of occurrence. They are at the same time harder to handle, since they coincide more frequently with dry air [6,14]. Although deaths from wildfires are rare in comparison to other natural disasters, they destroy ecosystems, threaten homes, livelihoods, technical infrastructure such as railways and the electricity grid, and lead to a reversal of carbon capture [6,13]. An increased frequency of wildfires demands a comprehensive and urgent response, and governments around the world already are investing in wildfire research with the goal of understanding its causes and how damages can be mitigated [3].

According to [10], the operations research community has been studying wildfire management since the early 1960s, and [7] is one of the first works dealing

S. Smith et al. (Eds.): EvoApplications 2024, LNCS 14634, pp. 273–286, 2024.
https://doi.org/10.1007/978-3-031-56852-7_18

with the application of operations research techniques to forest fire problems. Since then, we can find in the literature a variety of mathematical models that aim to capture decisions related to the process of preventing and suppressing a wildfire, such as the coordination of fire crews, the deployment of aerial firefighting assets and the routing of vehicles to transport firefighters and other equipment. To give some examples, in 1995 [5] proposed the firefighter problem, which is defined on a graph where fire spreads from an ignition node to adjacent nodes in sequential time steps. At each time instant, a certain number of fire suppression resources is available and can be deployed to unburned nodes. Applying a resource to a node prevents the fire from spreading through its outgoing edges to adjacent nodes, and the goal is to stop the fire in the minimum amount of time steps. [2] proposed a more realistic mixed-integer linear programming model that integrates fire spread behavior and the placement of suppression resources. The landscape is represented by a graph and the model comprises control variables, to decide which nodes will receive fire suppression resources, and response variables, which define fire spread paths, fire arrival times, and fire intensity for all the nodes. The goal is to minimize the total value of the burned area together with operational costs.

In this work, we consider a problem first defined in [1] and [11]. Similarly to [2], we are given a graph representing a landscape, an ignition node and some fire suppression resources spread over time. The goal is to allocate the fire resources to the nodes in order to minimize the burned area at some target time instant. In this context, our main contribution is a heuristic algorithm based on iterated beam search that achieves better results than previous approaches in a fraction of the time.

To close this section, we give an overview of what follows. In Sect. 2, we formally define the problem. Section 3 goes over the algorithmic approaches to this problem that can be found in the literature. Section 4 provides a series of definitions that will be used to explain our algorithm, which is presented in Sect. 5. In Sect. 6, we conduct some computational experiments to study the performance of our algorithm. Section 7 concludes the work and proposes new research directions.

2 Problem Description

Fire propagation is modeled by a directed graph $G = (V, A)$ with travel times t_a on arcs $a \in A$, which model the time required for fire to propagate from a node to a neighboring node. A directed graph permits to model different fire travel times in opposite directions, which can occur due to factors like wind and terrain slope. Given an ignition node $s \in V$, the travel times define a shortest-path tree rooted at s in which each node $v \in V$ has an associated fire arrival time a_v and a predecessor p_v. Now assume we have k fire suppression resources which can be allocated to nodes $v \in V$, and each resource adds a delay Δ to the outgoing arcs of v. Each resource $i \in [k]$ is available at time r_i, and can only be allocated

to a node v if $a_v \geq r_i$, i.e. if v is not burned yet[1]. We also assume that each node can receive at most one resource. Finally, we have a time horizon H and are interested in nodes that do not burn until H.

The allocation of resources to nodes can be represented by an injective function $\Lambda : [k] \rightarrow V$. By definition, such an allocation changes the travel times t, but it can also change the topology and the arrival times of the shortest-path tree. As a result, given an allocation of resources Λ, we denote the resulting fire propagation times by t^{Λ}, the fire arrival times by a^{Λ}, and the predecessor relation by p^{Λ}. The problem, then, is to find a feasible allocation of resources Λ that minimizes the number of burned nodes at time instant H, i.e.

$$b = \sum_{v \in V} [a_v^{\Lambda} \leq H].$$

3 Related Work

The problem we are interested in was first proposed as a mixed-integer linear programming (MIP) model by [1]. In [11], the authors propose a set of representative instances for this model and an iterated local search to solve them. The authors compare the performance of the local search with the performance of a commercial solver on the mathematical formulation of [1]. In computational experiments, they show that the heuristic achieves good results in a reasonable amount of time for all instances, while the solver needs more time to produce results and, for some large instances, fails to produce a feasible solution within the time limit of 2 h.

[4] extend the work of [1] and [11] by proposing a better MIP formulation of the problem, an exact algorithm using logic-based Benders decomposition, and a simple greedy heuristic used to warm-start the exact algorithm. In computational experiments, they show that the exact algorithm and a commercial solver using the new MIP model can solve all the instances proposed by [11] in a few seconds. In light of that, they propose new instances consisting of 20×20 grids and a larger optimization horizon. In another round of computational experiments, they compare the performance of the solver, the iterated local search of [11] and the proposed exact algorithm considering a time limit of 2 h. The solver was not able to prove the optimality of any instance, failed to produce a feasible solution in some cases and had the overall worst performance regarding solution quality. The iterated local search was able to find the optimal solution of some instances, but in most of the time it stayed behind the exact algorithm, which was able to find and prove the optimality of all instances.

4 Preliminaries

Consider a grid graph $G = (V, A)$. The immediate neighborhood of a node $v \in V$, denoted as $\mathbb{N}(v)$, encompasses nodes reachable through outgoing arcs of v. Sim-

[1] We use $[n]$ to denote a set containing the first n natural numbers, i.e. $[n] = \{1, \ldots, n\}$.

ilarly, the extended neighborhood $\mathbb{N}^*(v)$ includes nodes reachable via outgoing arcs as well as diagonal connections from v. Time instants where resources become available are represented by a sequence of times $T = (t_1, t_2, \dots)$, in ascending order. We denote by $\alpha(t) = \min_{i>0|t_i>t} t_i$ the first time instant after t when new resources become available, with $\alpha(t) = H$ if no further resources become available after t. Finally, for each time instant $t \in [0, H]$, $R_t \subseteq [k]$ is a set containing the resources that become available at time t.

When an allocation of resources Λ assigns a resource to a node $v \in V$, we say that v is *protected*. We denote by $P^\Lambda \subseteq V$ the set of nodes protected by Λ. If $|P^\Lambda| = k$ we say that Λ is a *complete allocation*. Conversely, if $|P^\Lambda| < k$ we say that Λ is *partial*. The special allocation that does not protect any node is denoted by Λ_0, i.e. $P^{\Lambda_0} = \emptyset$. Finally, given an allocation Λ and a time instant $t \in [0, H]$, we define $B_t^\Lambda = \{v \in V \mid a_v^\Lambda < t\}$ as the set of nodes that are burned at t. Note that our goal is to find an allocation Λ such that $|B_H^\Lambda|$ is minimized.

5 Proposed Algorithm

Beam search is a graph search algorithm that visits nodes in a breadth-first manner until a target node is reached. Starting from the root node, beam search keeps a list of β nodes and, at each level of the search tree, nodes in the list are expanded η times. In the literature, β is known as the beam width and η as the ramification factor. A heuristic function is then used to rank the $\beta\eta$ expansions, and the best β nodes are selected to continue to the next iteration. Beam search has been extensively used to tackle optimization problems [9]. In the context of our problem, each interior node of the search tree represents a partial allocation of resources, and leaf nodes are complete allocations. The root node is Λ_0, and for each $t \in T$, we expand the current set of allocations by applying the resources in R_t. The best leaf node is returned by the algorithm.

5.1 Beam Search

Algorithm 1 gives a high level view of our approach. In line 1, we create a set $\mathcal{A}$ containing only Λ_0, which will represent the current state of the search tree. For each time instant $t \in T$, we use the function Step to expand each node in $\mathcal{A}$, and we store all the expansions of the current level in the set E. In line 4, we use a heuristic function to select the β best allocations to continue to the next iteration, and in line 6 we return the best leaf node in the search tree. We explain how to expand a given allocation in Sect. 5.2. We will next define the heuristic function used to prune the search tree.

We propose two heuristic functions to evaluate a partial allocation of resources Λ. The first one, which we call h_1, is equal to the number of burned nodes at time instant H.

$$h_1(\Lambda) = \sum_{v \in V} [a_v^\Lambda \leq H]$$

Algorithm 1: BeamSearch

Data: Fire perimeter size z.
Result: An allocation of resources Λ.
1 $\mathcal{A} \leftarrow \{\Lambda_0\}$
2 **for** $t \in T$ **do**
3 $\quad E \leftarrow \bigcup_{\Lambda \in \mathcal{A}} \text{Step}(\Lambda, t, z)$
4 $\quad \mathcal{A} \leftarrow \text{prune}(E, t)$
5 **end**
6 **return** $\arg\min_{\Lambda \in \mathcal{A}} |B_H^\Lambda|$

Heuristic h_1 can be quite uninformative in the first few time instants, especially when the delay Δ is low and the optimization horizon H is large. In such situations, it is likely that the first few resources available cannot save any nodes, hence a comparison between two allocations is uninformative. In light of that, we propose a second heuristic, called h_2, which aims to measure how much delay an allocation Λ introduces in the network.

$$h_2(\Lambda) = \sum_{v \in V} \max\{H - a_v^\Lambda, 0\}$$

As we will see in the experimental section, we can obtain better results by starting with h_2 as the guiding heuristic and then switching to h_1 at some point in time. We call the time instant at which we start using h_1 *transition instant*, and we denote it by $\hat{t}$. It is better to define the transition instant relative to the velocity with which the fire propagates. To this end, we define the *free burning time* of an instance as the time instant at which the last node is burned assuming that no node is protected by a resource, i.e. the free burning time equals $\max_{v \in V} a_v^{\Lambda_0}$. We can now specify the transition instant as a percentage of the free burning time, and we denote this percentage by $\hat{p}$. To give an example, if the free burning time of an instance is 10 and $\hat{p} = 0.5$, we have that the transition instant $\hat{t}$ is equal to 5.

In summary, if $t < \hat{t}$ we prune the search tree by selecting the β best partial allocations in E using h_2. If $t \geq \hat{t}$, we use h_1. In Sect. 6.3, we study how the transition instant affects the performance of our algorithm.

5.2 Expanding an Allocation of Resources

We now consider the problem of generating the expansions of a given allocation in the search tree, as is done in line 3 of Algorithm 1. As a first step, we will develop a procedure to create a single expansion (Algorithm 2) and later we will embed it into Algorithm 3, which implements the function Step, called in line 3 of Algorithm 1.

Algorithm 2: Expand

Data: A partial allocation of resources Λ, a time instant t, the fire perimeter size z.
Result: An expansion of Λ using the resources in R_t.

$F \leftarrow F_t^{\Lambda}(z)$
$N \leftarrow F \cap \bigcup_{v \in P^{\Lambda}} \mathbb{N}^*(v)$
for $i \in R_t$ **do**
  if $N \neq \emptyset$ and $p < \text{rand}(0, 1)$ **then**
    $v \leftarrow$ Randomly pick an element of N
  else
    $v \leftarrow$ Randomly pick an element of F
  end
  $\Lambda_i \leftarrow v$
  $F \leftarrow F \setminus \{v\}$
  $N \leftarrow (N \cup \mathbb{N}^*(v)) \cap F$
end
return Λ

Given an allocation of resources Λ and a time instant $t \in T$, we have a set of candidate nodes $C = V \setminus (B_t^{\Lambda} \cup P^{\Lambda})$ which can receive a resource and $|R_t|$ resources available. Our goal is to select a subset of C of size $|R_t|$ to apply the resources in R_t. Algorithm 2 is based on two key observations about which nodes tend to receive a resource first in high quality solutions:

1. Nodes that are close to burned nodes;
2. Nodes that are neighbors of protected nodes.

Motivated by the first observation, we define the notion of *fire perimeter*, i.e. a set of nodes that are close to the current set of burned nodes. Since the arcs of an instance represent fire velocity instead of physical distance, our notion of closeness must be based on fire arrival time. With that in mind, we define the fire perimeter at a time instant t as

$$F_t^{\Lambda}(z) = \{v \in C \mid t \leq a_v^{\Lambda} \leq f(t, z)\}$$

for some non-negative integer z, where $f : T \times \mathbb{N}_0 \to [0, H]$ is[2]

$$f(t, z) = \left(\alpha^{\lceil (z+1)/2 \rceil}(t) + \alpha^{\lceil (z+2)/2 \rceil}(t)\right)/2.$$

where $\alpha(t)$ is the earliest time after t in which new resources get available, as defined at the end of Sect. 4.

Intuitively, the fire perimeter at time instant t is the set of unprotected nodes whose fire arrival time is between t and some other time instant t', where

[2] We write $\alpha^n(t)$ for the composition of α with itself n times, e.g. $\alpha^2(t) = \alpha(\alpha(t))$.

$t' = f(t, z)$ for some non-negative integer z. Increasing z will increase t', which in turn may increase the size of $F_t^{\Lambda}(z)$. As a result, z controls the size of the fire perimeter. We clarify this notion with an example. Suppose we have resources at time instants 10, 20, and 30, and the optimization horizon H is equal to 60. Assume that the current time instant is 10 and no resources were deployed yet, i.e. the current allocation is Λ_0. In this scenario, $F_{10}^{\Lambda_0}(0)$, $F_{10}^{\Lambda_0}(1)$, $F_{10}^{\Lambda_0}(2)$, $F_{10}^{\Lambda_0}(3)$ contain the nodes that will burn between time instant 10 and $f(10, 0) = 20$, $f(10, 1) = 25$, $f(10, 2) = 30$, and $f(10, 3) = 45$, respectively. Figure 1 illustrates the example.

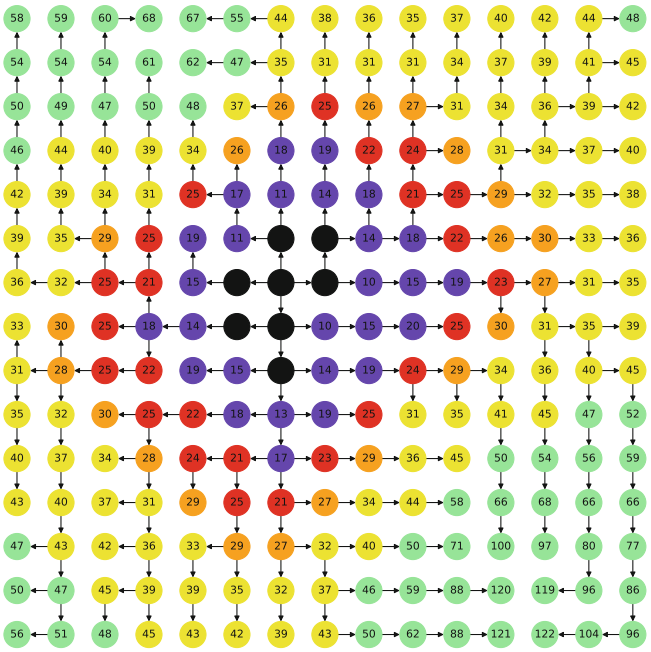

Fig. 1. A simple illustration of our definition of fire perimeter. In the example, we have resources at time instants 10, 20, and 30, and the optimization horizon H is equal to 60. We are at time instant 10 and no resources were deployed yet, i.e. the current allocation is Λ_0. The set $B_{10}^{\Lambda_0}$ is represented by the black colored nodes, $F_{10}^{\Lambda_0}(0)$ by purple nodes, $F_{10}^{\Lambda_0}(1)$ by red nodes, $F_{10}^{\Lambda_0}(2)$ by orange nodes, and $F_{10}^{\Lambda_0}(3)$ by yellow nodes. Note that $F_{10}^{\Lambda_0}(0) \subset F_{10}^{\Lambda_0}(1) \subset F_{10}^{\Lambda_0}(2) \subset F_{10}^{\Lambda_0}(3)$. (Color figure online)

In summary, Algorithm 2 considers only nodes in $F_t^{\Lambda}(z)$ instead of exploring all nodes in C (line 1). Similarly, and motivated by the second observation, we define a set N with all the nodes in $F_t^{\Lambda}(z)$ that have a neighbor in P^{Λ} (line 2). Using the sets $F_t^{\Lambda}(z)$ and N, Algorithm 2 proceeds as follows: for each resource $i \in R_t$, with probability p we select a node from N to be protected, and with probability $1 - p$ we select any node from F (lines 3 to 11).

Algorithm 3: Step

Data: A partial allocation of resources Λ, a time instant t, the fire perimeter size z.
Result: A set with at most η expansions of Λ using the resources in R_t.

```
1 E ← ∅
2 repeat c|F_t^Λ(z)| times
3 |   Λ' ← Expand(Λ, t, z)
4 |   E ← E ∪ {Λ'}
5 end
6 E ← sort(E, t)
7 return First η expansions in E
```

We now embed Algorithm 2 into Algorithm 3, which gives us a procedure to create a set of candidate expansions of a partial allocation Λ. In lines 2 to 5 we create a number of expansions proportional to the size of $F_t^{\Lambda}(z)$, for some constant $c \in \mathbb{Z}_+$. In line 6 we sort all the expansions in E using some heuristic function. Similarly to Algorithm 1, if $t < \hat{t}$, we use h_2, otherwise we use h_1. Finally, in line 7 we return the first η allocations in E. Note that in line 4 we do not check whether Λ' already is in E, hence it could be the case that $|E| < \eta$.

5.3 Dynamical Update of the Fire Perimeter Size

In Sect. 5.2, we defined the notion of fire perimeter, which depends on an integer constant z. Setting z to a value that is too high may increase running times, since the number of iterations performed by Algorithm 3 is directly proportional to the size of the fire perimeter. On the other hand, setting z to a value that is too low may impede the algorithm to find optimal solutions. To account for that, we propose to start with $z = 0$ and iteratively increase its value once a call to Algorithm 1 is not able to improve the current best solution. We observed in preliminary experiments that increasing z indefinitely does not improve performance and slows downs the algorithm in some cases, so we propose to define a maximum value for z and, once this value is reached, we cycle back to $z = 0$. Line 8 of Algorithm 4 illustrates that. Note that, for a given choice of $z_{\max}$, the maximum value of z is $z_{max} - 1$.

6 Experimental Evaluation

In this section we present some computational experiments. All the experiments were done on a platform with a 3.5 GHz AMD Ryzen 9 3900X 12-Core processor, 32 GB of main memory, and Ubuntu Linux 20.04 LTS. Our algorithm was implemented in C++ and compiled with GCC 9.4 with maximum optimization. Our implementation and detailed computational data is available at https://github.com/gutodelazeri/Iterated-Beam-Search.

Algorithm 4: Main Algorithm

Result: An allocation of resources Λ^*.

$\Lambda^* \leftarrow \Lambda_0$
$z \leftarrow 0$
while Termination criteria not met **do**
  $\Lambda \leftarrow \text{BeamSearch}(z)$
  if $|B_H^{\Lambda}| < |B_H^{\Lambda^*}|$ **then**
    $\Lambda^* \leftarrow \Lambda$
  else
    $z \leftarrow (z+1) \mod z_{max}$
  end
end
return Λ^*

Table 1. Instances used in the experiments.

Group	Resources per time instant						H	Δ
	10	20	30	40	50	60		
LA	3	3	3	3	0	0	70	50
LB	3	3	3	3	3	3	70	30

6.1 Test Instances

In this work we consider the set of instances proposed by [4]. This set consists of 16 instances, where each instance is a 20×20 grid graph. In all instances, the ignition node is at a central location in the graph and the optimization horizon is 70. The 16 instances are divided into two groups of 8 instances each, based on the magnitude of the delay caused by a resource and the quantity of resources released at each time instant. The optimal solution of all 16 instances is known, so in the sections below we report algorithm performance in terms of the absolute deviation from the optimal objective value[3]. Table 1 summarizes the instance set.

In this set of instances, edge weights attempt to model the fire propagation influenced by wind direction. In practice, the weight of each edge is sampled from a uniform distribution, and the range of values in this distribution depends on the direction to which the edge points. For further information, readers can refer to Table 5 in [4].

[3] In [4], the optimality of instance LB7 could not be proved. By executing their method with a time limit of 3 h we were able to find the optimal solution.

Table 2. Description of parameter values.

Parameter	Value	Description
β	50	Beam width
η	70	Ramification factor
c	30	See Algorithm 3
p	0.5	See Algorithm 2
z_{max}	3	See Algorithm 4

Table 3. Performance of beam search using different transition instants, as a function of a percentage $\hat{p}$ of the free burning time. We denote by δ the absolute difference between the obtained solution and the optimal solution, and columns $\overline{\delta}$ and σ_δ show the average and the standard deviation of δ across the 320 executions (16 instances and 20 replications). Similarly, we denote by ttb the time in seconds required to find the best (not necessarily optimal) solution. Columns $\overline{ttb}$ and σ_{ttb} give the average and the standard deviation of ttb. Lastly, column "Opt." has the percentage all 320 executions in which the optimal solution was found.

$\hat{p}$	$\hat{t}$	$\overline{\delta}$	σ_δ	$\overline{ttb}$	σ_{ttb}	Opt. (%)
0.1	6.9	1.41	3.59	61.18	105.06	75
0.2	13.8	0.15	0.35	52.23	110.73	85
0.3	20.7	0.12	0.33	26.68	46.40	88
0.4	27.6	0.12	0.33	26.62	46.24	88
0.5	34.5	0.18	0.45	43.38	82.68	85

6.2 Parameter Values

In all the experiments below, our algorithm uses the same set of parameter values, which are specified in Table 2. As stated in Sects. 5.1 and 5.2, the guiding heuristic used depends on the transition instant. In the next section, we conduct an experiment to find the best transition instant for the instances we are considering.

6.3 Transition Instant

In this section, we analyse how the transition instant affects the performance of our algorithm. Recall that in Sect. 5.1 we defined the transition instant as a percentage $\hat{p}$ of the free burning time of an instance. In this experiment, for each value of $\hat{p} \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ we run the beam search algorithm 20 times with different seed values on each of the 16 instances. The termination criterion was a maximum running time of 600 s. For the set of instances we are considering, the free burning time is always equal to 69, so for any value of $\hat{p}$ the transition instant is the same for all 16 instances. For each run, we collected

the best objective value obtained and the time to find the best solution. Table 3 summarizes the results.

For the instances we are considering, a transition instant of 6.9 means that only the heuristic h_1 is used. As the first row shows, this is the worst version of our algorithm. When the transition instant is 13.8, we use h_2 when $t = 10$ and h_1 otherwise. As the second row shows, this version obtains better results when compared to using h_1 only. When the transition instant is between 20 and 30, as is the case of rows three and four, heuristic h_2 is used when $t = 10$ and when $t = 20$. The table shows that this is the best version of our algorithm. This version was able to find the optimal solution in 88% of the 320 executions, and obtained an average absolute gap of $\delta = 0.12$.

6.4 Comparison with the Literature

In this section we compare the best version of our algorithm found in the last experiment (BS) against the logic-based Benders decomposition of [4] (LBBD) and the iterated local search of [11] (ILS). We use the implementation of LBBD and ILS provided by [4] and run them in the same computational environment as BS. Following the protocol of Sect. 6.3, all three algorithms were executed 20 times on each of the 16 instances. The termination criterion for LBBD and BS was a maximum running time of 600 s. The termination criterion of ILS was a maximum number of iterations in stagnation, as specified in [11]. Table 4 shows the results.

As we can see, BS solves to optimality 14 out of the 16 instances in all 20 replications, while LBBD does so for 9 instances and ILS for only one instance. We can also see that BS obtains the smallest average absolute gap δ. In [4], LBBD was compared to ILS given a time limit 7200 s. Here we can see that, even with a fraction of the time limit, LBBD still beats ILS by a significant margin. Regarding the average time to find the best solution, we can see that BS obtains the smallest one in all instances. Considering that BS finds an optimal solution in most of the executions, this shows the efficiency of our algorithm.

To close this section, we analyse the performance profile of the three algorithms over the 320 executions. Figure 2 shows the percentage of the 320 executions that found an optimal solution within a particular interval of time. As we can see, within just 100 s our algorithm finds an optimal solution in about 80% of all executions, while LBBD does so for around 40% and ILS for around 20%. Within 300 s, the curves of BS and ILS stagnate. This is not true for LBBD, since it explores the search space systematically. Finally, within 600 s LBBD finds an optimal solution in about 70% of all executions and ILS in about 20%. As we saw in the last section, BS is able to find an optimal solution in 88% of all executions.

Table 4. Comparison between BS, ILS, and LBBD. We denote by δ the absolute difference between the obtained solution and the optimal solution, and the first six columns show the average and the standard deviation of δ over the 320 executions. Similarly, we denote by ttb the time in seconds required to find the best (not necessarily optimal) solution, and the last three columns show the average ttb of each algorithm. The values in these columns are expressed in terms of the average ttb of BS. For example, by looking at the first row we can see that ILS takes, on average, 50 s more than BS to arrive at the best solution.

	$\overline{\delta}$			σ_δ			$\overline{\text{ttb}}$		
	BS	ILS	LBBD	BS	ILS	LBBD	BS	ILS	LBBD
LA0	0	1.45	0.00	0	2.28	0.00	0	50.59	37.39
LA1	0	5.65	0.00	0	4.85	0.00	0	61.25	5.72
LA2	0	3.05	0.00	0	3.17	0.00	0	0.52	13.57
LA3	0	10.65	0.00	0	6.27	0.00	0	53.38	52.64
LA4	0	0.00	0.00	0	0.00	0.00	0	49.03	66.41
LA5	0	1.85	0.00	0	1.46	0.00	0	83.55	64.96
LA6	0	8.05	0.00	0	6.68	0.00	0	91.25	100.63
LA7	0	6.40	0.00	0	5.08	0.00	0	42.70	154.64
LB0	1	5.75	0.55	0	5.34	1.10	0	126.58	174.00
LB1	0	12.00	0.00	0	4.67	0.00	0	45.62	104.94
LB2	1	5.00	1.10	0	4.77	0.79	0	97.23	164.39
LB3	0	10.30	3.50	0	4.05	2.42	0	115.61	206.35
LB4	0	9.65	7.05	0	6.13	4.97	0	118.18	262.09
LB5	0	14.40	3.85	0	3.39	3.10	0	82.33	266.53
LB6	0	15.00	8.60	0	5.34	3.90	0	106.62	340.72
LB7	0	9.20	6.35	0	5.52	3.25	0	71.82	242.15
Avg	0.12	7.40	1.94	0	4.31	1.22	0	74.77	141.07

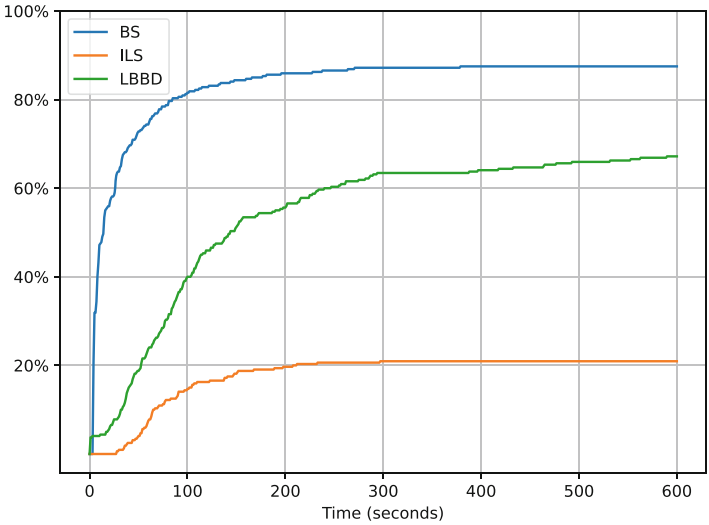

Fig. 2. Performance profile for the three algorithms, considering all 320 executions (16 instances and 20 replications). The x-axis shows time in seconds and the y-axis shows the percentage of the 320 runs in which the algorithm found the optimal solution within that time.

7 Conclusions and Future Work

In this work we proposed a heuristic algorithm for a problem related to wildfire suppression. The goal was to allocate fire suppression resources to regions of a landscape represented by a graph in order to minimize the total burned area. Our algorithm is a beam search guided by two heuristic functions to evaluate partial solutions and some heuristic rules on how to better expand the search tree at each level. In computational experiments, we showed that we can obtain better results by starting with one of the heuristic functions and then switching to the other at some point in time. Using these findings, we compared our approach to previous works in the literature. Our results indicate that the beam search algorithm can consistently find the optimal solution of most instances in considerably less time than alternative algorithms.

As future work, we would like to test our algorithm in more challenging instances, both in terms of grid size and the degree of irregularity of the landscapes. It would also be interesting to extend our approach to take into account different objective functions, like operational costs and the cost of the burned area.

Acknowledgments. M. R. acknowledges support from CNPq (grant 437859/2018-5), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), Finance Code 001, and CYTED (Grant P318RT0165).

References

1. Alvelos, F.: Mixed integer programming models for fire fighting. In: Gervasi, O., et al. (eds.) Computational Science and Its Applications - ICCSA 2018. Lecture Notes in Computer Science(), vol. 10961, pp. 637–652. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95165-2_45
2. Belval, E.J., Wei, Y., Bevers, M.: A mixed integer program to model spatial wildfire behavior and suppression placement decisions. Can. J. For. Res. **45**(4), 384–393 (2015)
3. Dimitropoulos, S.: Fighting fire with science. Nature **576**(7786), 328–328 (2019). https://doi.org/10.1038/d41586-019-03747-2
4. Harris, M.G., Forbes, M.A., Taimre, T.: Logic-based benders decomposition for wildfire suppression (2023)
5. Hartnell, B.L.: Firefighter! an application of domination. In: Proceedings of 25th Manitoba Conference on Combinatorial Mathematics and Computing (1995)
6. IPCC: Climate Change 2022: Impacts, Adaptation and Vulnerability. Contribution of Working Group II to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge University Press, Cambridge, UK and New York, NY, USA (2022). https://doi.org/10.1017/9781009325844
7. Jewell, W.S.: Forest fire problems—a progress report. Oper. Res. **11**(5), 678–692 (1963)
8. Joint Economic Committee, U.S. Senate: Climate-exacerbated wildfires cost the U.S. between $394 to $893 billion each year in economic costs and damages (2023). https://web.archive.org/web/20231114011935/https://www.jec.senate.gov/public/index.cfm/democrats/reports?id=E31AF93E-34C7-4C35-A416-533FF796369B. Accessed 13 Nov 2023
9. Lowerre, B.: The harpy speech recognition system. Ph.D. thesis, CMU (1976)
10. Martell, D.L.: A review of operational research studies in forest fire management. Can. J. For. Res. **12**(2), 119–140 (1982)
11. Mendes, A.B., e Alvelos, F.P.: Iterated local search for the placement of wildland fire suppression resources. Eur. J. Oper. Res. **304**(3), 887–900 (2023)
12. Munich Re: Wildfires and bushfires - Climate change increasing wildfire risk (2023). https://www.munichre.com/en/risks/natural-disasters/wildfires.html. Accessed 19 Jan 2024
13. Reuters: Death toll from Hawaii wildfires drops to 97 (2023). https://www.reuters.com/world/us/death-toll-hawaii-wildfires-drops-97-missing-is-now-31-hawaii-governor-2023-09-15. Accessed 19 Jan 2024
14. United Nations: As wildfires increase, integrated strategies for forests, climate and sustainability are ever more urgent (2023). https://www.un.org/en/un-chronicle/wildfires-increase-integrated-strategies-forests-climate-and-sustainability-are-ever-0. Accessed 19 Jan 2024