A New Solution Representation for the Firefighter Problem

Bin Hu(✉), Andreas Windbichler, and Günther R. Raidl

Institute of Computer Graphics and Algorithms,
Vienna University of Technology,
Favoritenstraße 9-11/1861, 1040 Vienna, Austria
{hu,raidl}@ads.tuwien.ac.at
windbichler.a@gmail.com

Abstract. The firefighter problem (FFP) is used as a model to simulate how a fire breaks out and spreads to its surroundings over a discrete time period. The goal is to deploy a given number of firefighters on strategic points at each time step to contain the fire in a most efficient way, so that as many areas are saved from the fire as possible. In this paper we introduce a new solution representation for the FFP which can be applied in metaheuristic approaches. Compared to the existing approach in the literature, it is more compact in a sense that the solution space is smaller although the complexity for evaluating a solution remains unchanged. We use this representation in conjunction with a variable neighborhood search (VNS) approach to tackle the FFP. To speed up the optimization process, we propose an incremental evaluation technique that omits unnecessary re-calculations. Computational tests were performed on a benchmark instance set containing 120 random graphs of different size and density. Results indicate that our VNS approach is highly competitive with existing state-of-the-art approaches.

Keywords: Firefighter problem · Spreading simulation · Variable neighborhood search

1 Introduction

The firefighter problem (FFP) was introduced by Hartnell [12] as a model to simulate the spread and containment of fire or disease over a discrete time period in a simplified way. In each time step the fire (or other malady) infects surrounding areas whereas a number of firefighters strategically protect certain regions in order to seal off the respective area and to prevent further spreading. In recent years this model has also been used to simulate information or virus spreading in computer networks.

Formally, the problem is defined on an undirected graph $G = \langle V, E \rangle$ where $|V| = n$ and each vertex $v \in V$ is initially flagged as untouched. At time $t = 1$,

This work is supported by the Austrian Science Fund (FWF) grant P24660-N23.

G. Ochoa and F. Chicano (Eds.): EvoCOP 2015, LNCS 9026, pp. 25–35, 2015.
DOI: 10.1007/978-3-319-16468-7_3

a fire breaks out at a pre-defined set of vertices $B_{\text{init}} \subseteq V$, and these vertices are flagged as burnt. For each time step $t = 2, 3, \ldots$, each member of a set of D firefighters protects a vertex $v \in V$ that is not burnt whereas the fire propagates to the neighboring vertices that are untouched and burns them. This process continues until the fire is contained, i.e., it cannot spread any further. The objective is to choose those D vertices in each step to be protected by the firefighters so that when the fire is contained, the number of vertices that are not burnt is maximal. It is assumed that protection is permanent, i.e., a vertex will not catch fire once it is protected, and that firefighters are able to move between arbitrary vertices from one time step to the next one.

An example is shown in Fig. 1. Suppose the number of firefighters is two, i.e., $D = 2$. At time $t = 1$ the fire starts at v_7 (red circle). At $t = 2$ we protect the vertices v_4 and v_9 (blue squares); the fire spreads to vertex v_{10}. At $t = 3$ we protect the vertices v_5 and v_{12}; the fire spreads to vertex v_{11}. At $t = 4$ we protect the vertices v_6 and v_8; the fire is finally contained. The objective value of this solution is 9 saved vertices.

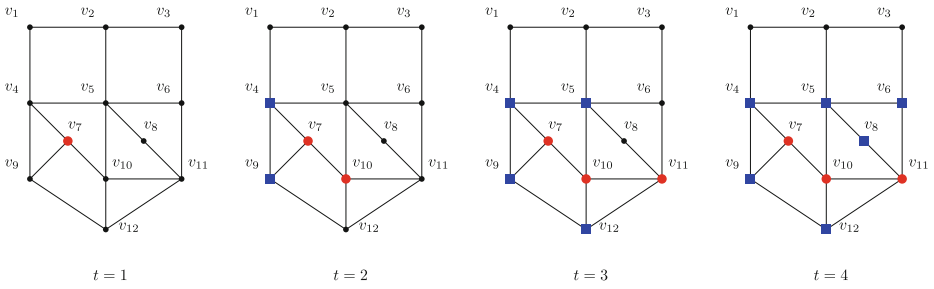

Fig. 1. Example for a graph with 12 vertices and two firefighters (Color figure online).

The contribution of this paper is to introduce a new compact solution representation for metaheuristic approaches for the FFP. We use it in a variable neighborhood search (VNS) approach along with an incremental evaluation technique to boost the performance and compare our results with those existing in the literature.

2 Previous Work

During the last 10 to 20 years, the FFP was studied by several researchers, but mostly from a theoretical point of view. An extensive survey can be found in [8]. The complexity of FFP was studied on various types of graphs: Finbow et al. [9] showed that it is NP-hard on trees with maximum degree three, but solvable in polynomial time if the fire starts at a vertex of degree two. MacGillivray and Wang [17] showed that it is NP-hard on bipartite graphs. King and MacGillivray [16] showed that the FFP is NP-hard if G is a cubic graph. In addition to these

graph types, grid structured graphs are particularly interesting due to their relevance in real world scenarios. Properties of the FFP such as the ability to contain the fire was studied by Fogarty [10] and Moeller et al. [19] on two dimensional grids. Later the results were generalized for higher dimensions by Develin and Hartke [7]. Besides the graph structure, complexity also depends on the number of available firefighters. Cases with more than one firefighter were studied by Bazgan et al. [1] and Costa et al. [6]. Apart from the complexity, approximation algorithms for FFP have been extensively studied in the literature, especially the case where G is a tree caught great interest. While Hartnell et al. [13] considered greedy approaches, Hartke [11] proposed a linear programming relaxation based algorithm and Cai et al. [3] proposed a subexponential algorithm. Later the approximation ratios of these approaches were improved by Iwaikawa et al. [15].

Recently Blum et al. [2] presented the so far only existing metaheuristic approach for the FFP based ant colony optimization (ACO). It uses a permutation based solution encoding representing the order of vertices that are considered for protection. The pheromone model contains values for each vertex appearing at each possible position in the permutation. As an extension, the authors also presented a hybrid ACO variant where half of the computation time is spent by the ACO and the latter half is used by further tuning the best solution found by the ACO via mixed integer programming. For this purpose they apply solution polishing which is a mechanism in CPLEX that emphasizes on improving a given solution instead of proving optimality. Surprisingly there are so far no other metaheuristic approaches in the literature to the best of our knowledge. Therefore the (hybrid) ACO is the main competitor for our VNS approach with which we will experimentally compare.

There are other variants of the FFP such as the fractional FFP where firefighters can split their strength into fractions to protect the vertices [10] or the stochastic variant where spreading is non-deterministic [4]. However, those variants are not the scope of this work.

3 Proposed Algorithm

The main aspect of this paper is to introduce a new solution representation for the FFP as a more compact alternative to the permutation based encoding in [2]. We use this new representation in a general variable neighborhood search (VNS) approach with variable neighborhood descent (VND) as embedded local improvement, see [18] for basic information on this kind of metaheuristics.

3.1 Solution Representation

We encode a solution as bitvector $P = \langle p_1, \ldots, p_n \rangle$ where $p_v, v \in V$, is 1 if the vertex should be protected and 0 otherwise. The solution does not explicitly state which vertices to be protected at a particular time step, but this information is implicitly derived during evaluation by Algorithm 1.

Algorithm 1. evaluate(P)

Input: solution $P = \langle p_1, \ldots, p_n \rangle$
Output: repaired solution P and its objective
$\forall v \in B_{\text{init}} : \ status[v] = -1$; `// the fire starts at time step 1`
$\forall v \in V \setminus B_{\text{init}} : \ status[v] = 0$; `// all other vertices are untouched`
$t = 2$;
$freeff = 0$; `// number of available firefighters in each step`
$burnt = 0$; `// number of burnt vertices`
$burning = true$;
while *burning* **do**
  $burning = false$;
  $freeff = freeff + D$;
  $neededff = 0$; `// number of currently needed firefighters`
  foreach vertex v adjacent to a burnt vertex at step $t - 1$ **do**
    if $p_v == 1$ **then**
      $neededff = neededff + 1$;
  `// now each vertex that is adjacent to a just burnt one has to be updated, i.e., either protected or burnt`
  foreach vertex v adjacent to a burnt vertex at step $t - 1$ **do**
    if $p_v == 1$ **then**
      `// if there are too many vertices that should be protected in the current step, drop a respective number with uniform probability`
      if $neededff > freeff$ and random$(1, neededff) > freeff$ **then**
        $p_v = 0$;
        $neededff = neededff - 1$;
      else
        $status[v] = t$; `// protect vertex at step` t
        $freeff = freeff - 1$;
        $neededff = neededff - 1$;
    if $p_v == 0$ **then**
      $status[v] = -t$; `// burn vertex at step` t
      $burnt = burnt + 1$;
      $burning = true$;
  $t = t + 1$;
return $n - burnt$;

During this procedure, we store for each vertex its *status*: 0 for untouched, a positive number z for protected at time step z and a negative number $-z$ for burnt at time step z. The number of burnt vertices is stored in *burnt* and potentially increases over the time steps. The number of available firefighters in each time step is indicated by *freeff* and corresponds to the number of firefighters D minus the number of vertices that would be set on fire at the current step but are marked as protected. This is the actual time when the vertex gets

protected. Additionally, spare firefighters in *freeff* can be buffered for future steps in the case not all of them are required at the current time step. Note that the concept of buffering does not conflict with the original problem definition since excess firefighters can be regarded as if they were protecting vertices that would become relevant later, at a time where more than D firefighters are required to protect the desired vertices. Then the number of required firefighters *neededff* is calculated, which consists of the number of vertices that are next to a burning vertex but should be protected according to the solution P. If *neededff* is larger than *freeff* at a certain time, the solution is actually infeasible since there are not enough firefighters to protect all the desired vertices from burning. Therefore, the evaluation procedure also contains an implicit repair function for this situation. If too many vertices are required to be protected, only *freeff* out of *neededff* vertices are saved and the others are dropped on a random basis where each vertex has equal probability. Other than that, the fire spreads over to adjacent vertices that are labeled as untouched.

An example for this procedure is shown in Fig. 2. The number of firefighters is again two and the fire starts at v_7 (red circle) at time $t = 1$. The solution vector is $P = \langle 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0 \rangle$ and states that vertices $v_1, v_2, v_4, v_6, v_8, v_9, v_{10}$ should be protected (blue boxes). At $t = 2$ three protected vertices are adjacent to a burning vertex: v_4, v_9 and v_{10}. Since there are only two firefighters, vertex v_4 gets dropped at random and catches fire (red square) while the other two are protected (blue squares). At $t = 3$ only one protected vertex v_1 is adjacent to burning vertices, so one spare firefighter is buffered. The unprotected vertex v_5 catches fire. At $t = 4$ three protected vertices are adjacent to burning vertices and all of them can be saved due to the buffered firefighter. This is equivalent to as if the spare firefighter had saved one of these vertices in the previous time step. Now the fire is fully contained and the objective value of this solution is again 9.

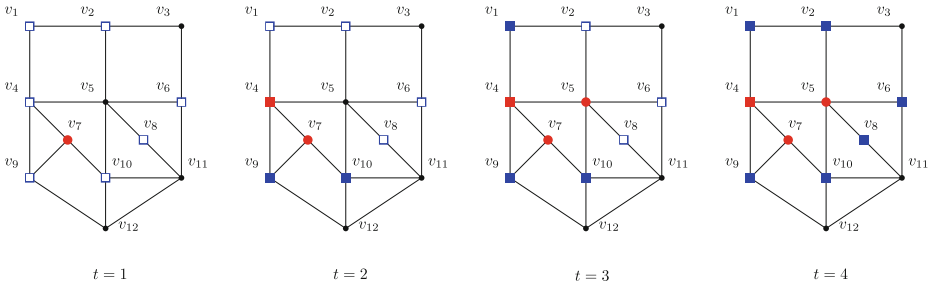

Fig. 2. Solution evaluation example for a graph with 12 vertices and two firefighters (Color figure online).

The whole evaluation procedure that includes the fire-spreading simulation runs in time $O(|V| + |E|)$ since each vertex and each edge is considered only once. This is the same complexity as the evaluation procedure of the ACO in [2].

However, the bitvector representation is more compact as its solution space has a size 2^n instead of $n!$ permutations.

3.2 Initial Solution

We tested two different strategies for creating initial solutions. In variant A, we assign a *closeness centrality* value c_v to each vertex $v \in V \setminus B_{\text{init}}$. This is done by calculating for each vertex the sum of its distances to all other vertices and then taking the inverse value as its closeness. This criterion is often used to determine the influence of a vertex to spread information through a network [20]. A vertex with a small closeness value indicates that once it burns, it has a high potential to further spread the fire. Based on the closeness centrality, we protect each vertex $v \in V \setminus B_{\text{init}}$ with a probability of $\frac{c_{\max} - c_v}{2(c_{\max} - c_{\min})}$ where $c_{\max}$ and $c_{\min}$ are the minimal and maximal closeness values over all vertices $V \setminus B_{\text{init}}$, respectively. This means that the vertex with minimal closeness gets protected with a probability of 50 % whereas the vertex with maximal closeness never gets protected.

In variant B, we just use an empty solution, i.e., where no vertex is set to be protected. This way we entirely rely on the improvement procedures of the VNS to set the protection status.

During preliminary tests, we applied local improvement via VND on these initial solutions and it turned out that no obvious trends between the two variants could be observed. Over a set of 120 instances (see Sect. 4), variant A was better in 32 cases and variant B in 49 cases. However the average solution quality of variant A was better by 0.2 %. On the one hand, it is a good sign that the improvement procedure works so well since it minimizes the differences of initial solutions. On the other hand, we also feel that the construction heuristic certainly has some improvement potentials. Since there was no obvious trend, we used variant B in the later tests since it is the simpler approach.

3.3 Variable Neighborhood Descent

In order to locally improve a solution, we use VND which considers the following neighborhood structure in a best improvement fashion. For neighborhood $N_l(P)$, $l = 1, 2, \ldots$, we consider in the current solution P a set W_l consisting of vertices with l unprotected adjacent vertices, respectively. For each vertex $w \in W_l$, we test if the solution improves by protecting its adjacent vertices, see Algorithm 2. This procedure maximizes the locality of the neighborhood structure since correlated vertices are considered together. If an infeasible solution arises, it is repaired during the evaluation procedure automatically. After that, the *improve* procedure tries to add single vertices to the protection until the solution does not improve.

In order to boost the performance, we implemented an *incremental evaluation scheme* on the evaluation procedure. Whenever we change for a vertex v its protection status p_v, we know the time it was processed in the last evaluation

Algorithm 2. VND

Input: solution $P = \langle p_1, \ldots, p_n \rangle$, neighborhood l
Output: Improved solution P
W_l = set of vertices with l unprotected adjacent vertices;
$P_{\text{best}} = P$;
foreach $w \in W_l$ **do**
  $P' = P$;
  $\forall v$ adjacent to w : $p'_v = 1$;
  evaluate(P');
  improve(P');
  if P' better than P_{best} **then**
    $P_{\text{best}} = P'$;
return P_{best};

by looking at $status[v]$. Either it was protected at time t if $status[v] > 0$, or it got burnt if $status[v] < 0$. Therefore when we evaluate the new solution after changing p_v in the neighborhood search or in the improve-procedure, we only have to re-calculate vertices $w \in V$ with $|status[w]| > t$ and possibly untouched vertices. In other words, the fire spreading simulation until time t with the status values of corresponding vertices will not change. We only have to take the existing status values from the previous evaluation, calculate the number of spare firefighters *freeff* and set B_{init} to those vertices burnt at time $t - 1$ to continue the fire simulation. If the protection status of more than one vertex gets changed, we have to continue the fire simulation at the smallest time where these vertices were processed. During preliminary tests this incremental evaluation scheme was able to speed up the whole approach by a significant amount. One variant we also considered was to use a first improvement strategy where we could cascade the neighborhood solutions in a way that solutions requiring less re-calculations are evaluated first to maximize the potential gain of the incremental evaluation. However, the best improvement strategy still produced better results in comparable time.

3.4 Variable Neighborhood Search

We use the general VNS framework with shaking when VND is not able to improve the solution any further. A common situation where VND gets stuck in local optima is that when evaluating and repairing an infeasible solution, vertices close to B_{init} are less likely to become dropped from protection than vertices that are farther away. This is due to the repair mechanism that always "activates" the protection for vertices close to B_{init} first. If there are too many vertices marked for protection at a certain time step, it repairs the solution by dropping vertices from the current step. However, dropping vertices from previous time steps would actually also work since excess firefighters would be buffered and carried over to the time where the deficit arises. Since this is not considered in the current

implementation, we have a slight bias that prefers vertices close to B_{init}. While this is in fact a reasonable strategy most of the time since protecting vertices close to B_{init} from early on often has a positive impact, there are situations where this behaviour causes the VND to get stuck in local optima.

For this reason, if shaking is called with a size k, it drops the protection status of k vertices selected randomly in the solution so that VND is able to explore new combinations of protected vertices.

4 Computational Results

We tested our VNS on the benchmark set from [2]. It consists of instances with 50, 100, 500, and 1000 vertices with three different edge densities, respectively. The edge density describes the probability of having an edge between two vertices in a random graph and are stated in the Tables 1, 2, 3 and 4. Each combination of vertex count and edge density contains 10 random instances, which results in a benchmark set containing 120 instances in total. In addition, each instance is considered with different number of firefighters $D \in \{1, \ldots, 10\}$. All tests were carried out on a single core of an Intel Xeon E5540 with 2.53 GHz and 3 GB RAM. Compared to the hardware used in [2] for the (hybrid) ACO, it is around twice as fast according to the Standard Performance Evaluation Corporation (SPEC) benchmark[1]. Therefore we use half of the ACO runtime for our VNS which is $n/4$ seconds for each instance to obtain roughly comparable results.

In Tables 1, 2, 3 and 4 we show the results grouped by the number of vertices and the edge density. We compare four different approaches: beside our VNS, the other three are taken from [2] and consist of an exact approach using mixed integer programming solved via CPLEX, the ACO and the hybrid ACO. Each line corresponds to a different number of firefighters D whereas each cell shows the average final solution values over 10 random graphs. Best results are marked bold. In the last two lines of the tables, we give a summary by showing for each

Table 1. Results for graphs with 50 vertices, time limit of 12.5 s for VNS.

	Edge propability $p_e = 0.1$				Edge propability $p_e = 0.15$				Edge propability $p_e = 0.2$			
D	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS
1	**7.4**	**7.4**	**7.4**	7.3	**4.5**	**4.5**	**4.5**	**4.5**	**3.1**	**3.1**	**3.1**	**3.1**
2	**26.6**	26.4	26.5	**26.6**	**9.7**	**9.7**	**9.7**	**9.7**	**7.2**	**7.2**	**7.2**	**7.2**
3	**41.8**	40.9	41.6	41.7	**18.8**	16.5	18.5	18.7	**11.2**	11.1	**11.2**	**11.2**
4	**47.9**	47.8	**47.9**	**47.9**	**31.2**	30.5	30.9	**31.2**	**17.5**	16.0	17.2	17.2
5	**48.5**	**48.5**	**48.5**	**48.5**	**39.1**	36.1	**39.1**	**39.1**	**27.7**	26.1	27.6	27.6
6	**48.8**	**48.8**	**48.8**	**48.8**	**43.7**	42.7	**43.7**	**43.7**	**33.0**	31.4	**33.0**	32.9
7	**49.0**	**49.0**	**49.0**	**49.0**	**46.3**	45.4	**46.3**	**46.3**	**37.5**	35.7	**37.5**	**37.5**
8	**49.0**	**49.0**	**49.0**	**49.0**	**48.1**	46.8	**48.1**	**48.1**	**42.7**	40.3	42.6	42.6
9	**49.0**	**49.0**	**49.0**	**49.0**	**48.6**	48.2	**48.6**	**48.6**	**46.1**	44.4	**46.1**	**46.1**
10	**49.0**	**49.0**	**49.0**	**49.0**	**48.8**	**48.8**	**48.8**	**48.8**	**47.5**	47.1	**47.5**	**47.5**
$\sum$	**417.0**	415.8	416.7	416.8	**338.8**	329.2	338.2	338.7	**273.5**	262.4	273.0	272.9
%	**100.00%**	99.71%	99.93%	99.95%	**100.00%**	97.17%	99.82%	99.97%	**100.00%**	95.94%	99.82%	99.78%

[1] www.spec.org/cpu2006.

Table 2. Results for graphs with 100 vertices, time limit of 25 s for VNS.

	Edge propability $p_e = 0.05$				Edge propability $p_e = 0.075$				Edge propability $p_e = 0.1$			
D	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS
1	**9.2**	9.1	**9.2**	9.1	**5.4**	**5.4**	**5.4**	**5.4**	**3.9**	**3.9**	**3.9**	**3.9**
2	26.9	25.7	27.6	**27.7**	**11.3**	11.2	**11.3**	11.2	8.5	8.3	**8.7**	**8.7**
3	62.8	54.6	62.7	**63.6**	41.5	41.0	**41.6**	41.5	**21.4**	21.0	21.3	**21.4**
4	85.3	66.3	85.5	**86.0**	53.7	52.4	53.3	**53.8**	25.5	24.5	25.5	**25.7**
5	**97.3**	92.3	**97.3**	**97.3**	65.7	63.5	65.9	**66.5**	**30.2**	29.1	29.5	30.1
6	**98.5**	98.3	**98.5**	**98.5**	87.5	75.1	87.3	**87.6**	41.8	33.9	41.0	**42.3**
7	**98.8**	**98.8**	**98.8**	**98.8**	**98.1**	87.9	**98.1**	**98.1**	**58.7**	46.4	56.3	56.9
8	**98.9**	**98.9**	**98.9**	**98.9**	**98.6**	93.5	**98.6**	**98.6**	**74.8**	62.0	74.0	**74.8**
9	**99.0**	**99.0**	**99.0**	**99.0**	**98.8**	**98.8**	**98.8**	**98.8**	**89.2**	77.3	88.0	**89.2**
10	**99.0**	**99.0**	**99.0**	**99.0**	**99.0**	**99.0**	**99.0**	**99.0**	**94.7**	85.9	94.4	94.6
$\sum$	775.7	742.0	776.5	**777.9**	659.6	627.8	659.3	**660.5**	**448.7**	392.3	442.6	447.6
%	99.70%	95.37%	99.81%	**99.99%**	99.83%	95.02%	99.79%	**99.97%**	**99.80%**	87.26%	98.44%	99.56%

Table 3. Results for graphs with 500 vertices, time limit of 125 s for VNS.

	Edge propability $p_e = 0.015$				Edge propability $p_e = 0.02$				Edge propability $p_e = 0.025$			
D	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS
1	7.6	7.5	**7.8**	7.6	5.3	5.2	5.6	**5.7**	4.2	4.4	4.3	**4.5**
2	5.6	13.0	13.6	**14.6**	10.6	10.4	11.2	**11.3**	9.1	8.5	9.0	**9.3**
3	3.1	18.8	21.3	**22.3**	60.3	63.1	63.6	**65.0**	12.8	12.7	**13.8**	13.7
4	150.2	119.9	168.3	**170.3**	69.5	67.6	**70.4**	70.0	17.7	16.9	**18.6**	18.1
5	250.5	218.9	265.9	**267.5**	45.6	72.7	74.6	**75.1**	6.5	21.6	22.4	**23.2**
6	349.1	268.8	**363.0**	362.9	102.4	123.8	**126.5**	126.1	25.6	26.3	**33.8**	28.7
7	448.9	407.6	453.5	**453.7**	102.7	128.1	130.1	**133.1**	78.1	77.6	**93.0**	80.1
8	449.1	453.9	**455.0**	454.7	299.0	135.8	315.6	**316.1**	154.2	127.6	173.5	**175.5**
9	449.1	454.7	456.6	**456.9**	349.0	317.5	363.8	**364.1**	223.2	221.8	**225.4**	223.8
10	**498.8**	455.5	**498.8**	**498.8**	409.2	321.1	**410.2**	409.3	221.5	225.1	**229.6**	227.3
$\sum$	2612.0	2418.6	2703.8	**2709.3**	1462.6	1245.3	1571.6	**1575.8**	753.0	742.4	**823.5**	804.2
%	96.39%	89.25%	99.77%	**99.98%**	92.72%	78.94%	99.63%	**99.89%**	91.09%	89.80%	**99.61%**	97.28%

Table 4. Results for graphs with 1000 vertices, time limit of 250 s for VNS.

	Edge propability $p_e = 0.0075$				Edge propability $p_e = 0.01$				Edge propability $p_e = 0.125$			
D	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS	CPLEX	ACO	HyACO	VNS
1	105.2	107.4	107.8	**108.0**	4.9	5.7	6.0	**6.3**	4.0	4.9	4.7	**5.3**
2	107.9	112.7	**115.0**	114.6	4.4	10.9	10.6	**12.1**	8.0	9.4	10.1	**10.2**
3	101.7	118.0	118.0	**122.1**	13.7	15.9	17.0	**17.9**	4.6	14.3	13.9	**15.4**
4	399.4	318.3	415.7	**417.9**	14.8	21.4	**24.1**	**24.1**	99.9	116.6	116.5	**117.8**
5	399.6	419.0	421.2	**422.8**	104.3	27.1	123.6	**126.6**	103.3	120.8	120.9	**122.8**
6	598.8	423.6	614.2	**617.1**	201.5	129.1	226.0	**228.2**	99.9	125.5	126.2	**128.7**
7	898.1	523.5	902.5	**903.3**	299.7	525.5	325.2	**328.2**	199.8	226.7	226.6	**228.8**
8	**998.2**	528.6	**998.2**	**998.2**	399.5	329.7	424.4	**426.5**	218.7	231.1	**234.8**	233.9
9	**998.9**	905.6	**998.9**	**998.9**	399.6	427.9	427.7	**431.8**	301.0	236.2	332.1	**332.6**
10	**999.0**	**999.0**	**999.0**	**999.0**	499.5	432.6	528.4	**530.7**	602.2	335.0	**620.5**	619.6
$\sum$	5606.8	4455.7	5690.5	**5701.9**	1941.8	1725.8	2113.0	**2132.4**	1641.4	1420.8	1806.3	**1815.1**
%	98.33%	78.14%	99.79%	**99.99%**	91.06%	80.93%	99.09%	**100.00%**	90.34%	78.20%	99.42%	**99.90%**

approach the summed up average solution values ($\sum$) and the percentage of the best values reached (%) when considering for each line the best performing approach.

As reported in [2], CPLEX was able to consistently solve all instances with 50 vertices to optimality. On larger instances, it was usually terminated after

reaching the time limit. Especially instances with dense graph and a low number of firefighters proved to be difficult. In these cases CPLEX is outperformed by the metaheuristic approaches. We observe that our VNS performs slightly better than the hybrid ACO. On the majority of the instance sets it is better and in some cases it is worse by a small margin. Compared to the pure ACO approach the VNS is consistently better. Overall, it seems that VNS performs better on sparse graphs and on larger instances. The latter is due to the reduced search complexity of the bitvector representation. On sparse graphs we suspect that it is more convenient for the VND neighborhood structure to iterate through the vertices since their degrees are lower and more diverse. Looking at the closeness of the different approaches, we also think that being able to solve a considerable part of the instances in this benchmark set optimally by CPLEX shows that they are not very hard, thus the margin for improvement is rather slim. Therefore more sophisticated approaches in the future should be tested on more complex instances so that differences become more obvious.

5 Conclusions and Future Work

We proposed a variable neighborhood search approach for the firefighter problem based on a bitvector solution representation. By storing for each vertex only its protection status, it is more compact than a permutation based representation. We also proposed an incremental evaluation technique to speed up the computation significantly. Although the VNS is not able to outperform the hybrid ACO approach in a substantial way, it is typically at least as good when it comes to solution quality and performs significantly better than the standard ACO.

In future work we want to investigate approaches that make use of both representations and associate neighborhood structures since they can be considered as complementary to each other: The bitvector representation stores the protection status but the actual order in which the vertices are protected is obtained during evaluation. The permutation representation stores the order in which the vertices are considered for protection but whether a vertex is protected or not is determined during evaluation. It has been shown that solution methods on combinatorial optimization problems can substantially benefit from using complementary representations, e.g., in the case of generalized minimum spanning tree problem as shown in [5,14]. Considering the FFP, using both representations either in a VNS fashion or infusing the ACO from [2] with a local search method based on the new representation appears to be particularly promising.

References

1. Bazgan, C., Chopin, M., Ries, B.: The firefighter problem with more than one firefighter on trees. Discrete Appl. Math. **161**(7–8), 899–908 (2013)
2. Blum, C., Blesa, M.J., García-Martínez, C., Rodríguez, F.J., Lozano, M.: The firefighter problem: application of hybrid ant colony optimization algorithms. In: Blum, C., Ochoa, G. (eds.) EvoCOP 2014. LNCS, vol. 8600, pp. 218–229. Springer, Heidelberg (2014)

3. Cai, L., Verbin, E., Yang, L.: Firefighting on trees: $(1 - 1/e)$–approximation, fixed parameter tractability and a subexponential algorithm. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 258–269. Springer, Heidelberg (2008)
4. Comellas, F., Mitjana, M., Peters, J.G.: Broadcasting in small-world communication networks. In: Proceedings of the 9th International Colloquium on Structural Information and Communication Complexity, pp. 73–85 (2002)
5. Corus, D., Lehre, P.K., Neumann, F.: The generalized minimum spanning tree problem: A parameterized complexity analysis of bi-level optimisation. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 519–526. ACM (2013)
6. Costa, V., Dantas, S., Dourado, M., Penso, L., Rautenbach, D.: More fires and more fighters. Discrete Appl. Math. **161**(16–17), 2410–2419 (2013)
7. Develin, M., Hartke, S.: Fire containment in grids of dimension three and higher. Discrete Appl. Math. **155**(17), 2257–2268 (2007)
8. Finbow, S., Science, C., Scotia, N., MacGillivray, G.: The firefighter problem: a survey of results directions and questions. Aust. J. Comb. **43**, 57–77 (2009)
9. Finbow, S., King, A., MacGillivray, G., Rizzi, R.: The firefighter problem for graphs of maximum degree three. Discrete Math. **307**(16), 2094–2105 (2007)
10. Fogarty, P.: Catching the fire on grids. Master's thesis, University of Vermont, USA (2003)
11. Hartke, S.: Attempting to narrow the integrality gap for the firefighter problem on trees. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 225–231 (2006)
12. Hartnell, B.: Firefighter! An application of domination. In: 20th Conference on Numerical Mathematics and Computing, pp. 218–229 (1995)
13. Hartnell, B., Li, Q.: Firefighting on trees: How bad is the greedy algorithm? In: Proceedings of the Thirty-first Southeastern International Conference on Combinatorics, Graph Theory and Computing, pp. 187–192 (2000)
14. Hu, B., Leitner, M., Raidl, G.R.: Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. J. Heuristics **14**(5), 473–499 (2008)
15. Iwaikawa, Y., Kamiyama, N., Matsui, T.: Improved approximation algorithms for firefighter problem on trees. IEICE Trans. **E94.D**(2), 196–199 (2011)
16. King, A., MacGillivray, G.: The firefighter problem for cubic graphs. Discrete Math. **310**(3), 614–621 (2010)
17. MacGillivray, G., Wang, P.: On the firefighter problem. J. Comb. Math. Comb. Comput. **47**, 83–96 (2003)
18. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
19. Moeller, S., Wang, P.: Fire control on graphs. J. Comb. Math. Comb. Comput. **41**, 19–34 (2002)
20. Newman, M.J.: A measure of betweenness centrality based on random walks. Soc. Netw. **27**(1), 39–54 (2005)