Simheuristics for the Multiobjective Nondeterministic Firefighter Problem in a Time-Constrained Setting

Michalak, Krzysztof ; Knowles, Joshua D.

DOI:
10.1007/978-3-319-31153-1_17

Document Version
Peer reviewed version

Citation for published version (Harvard):
Michalak, K & Knowles, JD 2016, Simheuristics for the Multiobjective Nondeterministic Firefighter Problem in a Time-Constrained Setting. in G Squillero & P Burelli (eds), *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Part II.* vol. LNCS 9598, Lecture Notes in Computer Science, vol. 9598, Springer, pp. 248-265, Evolutionary Algorithms and Meta-heuristics in Stochastic and Dynamic Environments (EVOSTOC 2016), Porto, Portugal, 30/03/16. https://doi.org/10.1007/978-3-319-31153-1_17

Simheuristics for the Multiobjective Nondeterministic Firefighter Problem in a Time-Constrained Setting

Krzysztof Michalak[1], Joshua D. Knowles[2]

[1] Department of Information Technologies,
Institute of Business Informatics,
Wroclaw University of Economics, Wroclaw, Poland
krzysztof.michalak@ue.wroc.pl
[2] School of Computer Science University of Birmingham
Edgbaston, Birmingham
j.knowles.1@cs.bham.ac.uk

Abstract. The firefighter problem (FFP) is a combinatorial problem requiring the allocation of 'firefighters' to nodes in a graph in order to protect the nodes from fire (or other threat) spreading along the edges. In the original formulation the problem is deterministic: fire spreads from burning nodes to adjacent, unprotected nodes with certainty.
In this paper a nondeterministic version of the FFP is introduced where fire spreads to unprotected nodes with a probability P_{sp} (lower than 1) per time step. To account for the stochastic nature of the problem the simheuristic approach is used in which a metaheuristic algorithm uses simulation to evaluate candidate solutions. Also, it is assumed that the optimization has to be performed in a limited amount of time available for computations in each time step.
In this paper online and offline optimization using a multipopulation evolutionary algorithm is performed and the results are compared to various heuristics that determine how to place firefighters. Given the time-constrained nature of the problem we also investigate for how long to simulate the spread of fire when evaluating solutions produced by an evolutionary algorithm. Results generally indicate that the evolutionary algorithm proposed is effective for $P_{sp} \geq 0.7$, whereas for lower probabilities the heuristics are competitive suggesting that more work on hybrids is warranted.

Keywords: graph-based optimization, nondeterministic firefighter problem, simheuristics

1 Introduction

The firefighter problem (FFP) is a discrete-time optimization problem in which spreading of fire is modelled on a graph and the goal is to select nodes that should be protected in order to prevent fire from spreading. The same formalism can

be used for analyzing threats in computer networks and a spread of diseases in humans as well as in livestock. There are other similar problems complementary to the FFP For example, in the area of research on broadcasting and computer networks a question arises how many edges in the graph (rather than nodes) should be prevented from transmitting to stop the spread of an infection [2]. The paper [9] studies evolving (i.e. growing) graphs and analyzes their properties. While the aforementioned paper does not discuss any means of preventing the growth, some similarities can be seen between the evolving graphs and the subgraph composed of the nodes on fire in the FFP.

Originally, the firefighter problem was proposed by Hartnell in 1995 [6] as a single-objective, deterministic problem. Since then the research on the FFP has followed three main directions. The first area is the analysis of various theoretical aspects of the problem itself. For example, some researchers analyzed boundary conditions for which it is possible to save the graph [4]. The second line of work is the application of classical optimization methods, such as the linear integer programming [3]. Only recently a third area emerged in which metaheuristic algorithms are applied to the FFP. The paper [1], presented at the EvoCOP 2014 conference was, according to its authors, the first attempt to use a metaheuristic approach (ACO) to solving the FFP. In a paper [11] published later the same year the multiobjective version of the FFP was proposed and solved using and evolutionary algorithm (EA). Two more papers on the FFP were published at EvoCOP in 2015: [7] where variable neighborhood search (VNS) approach was applied to the single-objective FFP and a solution representation suitable for this algorithm was proposed, and [13] where the multiobjective version of the FFP was tackled using a multipopulation algorithm Sim-EA with migration based on similarity between search directions assigned to subpopulations.

All papers mentioned above concerned the deterministic version of the FFP. In a paper [5] the FFP is studied on randomly constructed graphs with randomly chosen starting points, but the spreading of fire is deterministic. In this paper we introduce and study the nondeterministic version of the FFP which involves uncertainty in the spreading of fire in the graph.

The rest of this paper is structured as follows. Section 2 provides a definition of both the deterministic and the nondeterministic version of the FFP. In section 3 the optimization method, an EA combined with simulation-based evaluation, is described. Section 4 presents experiments including comparisons with some simple but effective heuristics, and discusses the results. Section 5 concludes the paper.

2 Problem Definition

The **deterministic** version of the firefighter problem can be formalized in the following way. Let $G = \langle V, E \rangle$ be an undirected graph with N_v vertices. Each of the nodes of G can be in one of the states from a set $L = \{'\mathrm{B}', '\mathrm{D}', '\mathrm{U}'\}$ with the interpretation $'\mathrm{B}'$ = burning, $'\mathrm{D}'$ = defended and $'\mathrm{U}'$ = untouched. The state of the graph at time t is denoted S_t and for any vertex v the state of

this vertex at time t is $S_t[v]$. Apart from the graph (stored, for example as an adjacency matrix $A_{N_v \times N_v}$) the initial state S_0 of the graph and the number N_f of firefighters assigned per a time step are provided in each FFP instance. Most often the initial state S_0 is constructed by setting the state of vertices from a given non-empty set $\emptyset \neq S \subset V$ to $'\mathrm{B}'$ and the remaining ones to $'\mathrm{U}'$. Therefore, we can consider an FFP instance as an ordered triple $\langle G, S_0, N_f \rangle$.

The spread of fire is simulated in discrete time steps $t = 0, 1, \ldots$, with the graph set to the initial state S_0 at $t = 0$. In each time step $t > 0$ two events occur. First, a predefined number N_f of still untouched nodes become defended by firefighters (i.e. change their state from $'\mathrm{U}'$ to $'\mathrm{D}'$). Then the fire spreads along the edges of the graph G from the nodes labelled $'\mathrm{B}'$ to the neighbouring nodes labelled $'\mathrm{U}'$. Nodes defended by firefighters before fire gets to them (i.e. marked $'\mathrm{D}'$) remain protected until the end of the simulation. Conversely, nodes that catch on fire are considered lost and firefighters are not assigned to them. The simulation ends when fire cannot spread anymore because all nodes adjacent to the burning ones are defended (we say that fire is contained) or when all undefended nodes are burning.

A very often used representation of **solutions** of the FFP is a permutation-based representation, even though other representations are also considered [7]. In the permutation-based representation an individual solution is encoded as a permutation π of numbers $1, \ldots, N_v$. When firefighters are assigned, the first N_f numbers for which the corresponding vertices are untouched ($'\mathrm{U}'$) are taken from π and the state of these selected vertices is changed to $'\mathrm{D}'$. The goal in the **single-objective FFP** is to maximize the number of non-burning nodes ('D' or 'U') at the simulation end. Note, that we are only interested in the state of the nodes. Edges in the graph just define the topology and are not themselves subject to burning nor protecting by firefighters.

In the paper [11] the **multiobjective version of the FFP** was proposed in which there are m values $v_j(v)$, $j = 1, \ldots, m$ assigned to each node v in the graph. The objectives f_j, $j = 1, \ldots, m$ attained by a given solution π are calculated by simulating the spreading of fire from the initial state S_0 until the fire stops spreading. When the final state S_t is reached the objectives are calculated as: $f_j = \sum_{v \in V : S_t[v] \neq 'B'} v_j(v)$, where $v_j(v)$ is the value of the node v according to the j-th criterion.

In this paper we tackle the multiobjective version of the FFP described above, but contrary to the paper [11] we introduce the nondeterminism to the problem.

2.1 Nondeterministic FFP as a dynamic optimization problem

The **nondeterministic** version of the FFP used in this paper can be described similarly as the deterministic one with one key difference that there is an uncertainty in the way the fire spreads (we assume, however, that the initial set S of burning nodes is fixed). For describing the nondeterministic spreading of fire a simple model is used in which, in each time step t, for every edge $e = \langle v_1, v_2 \rangle \in E$ such that $S_t[v_1] = 'B'$ and $S_t[v_2] = 'U'$ the state of the vertex v_2 is set to $'\mathrm{B}'$ with a constant probability $P_{sp} \leq 1$. Formally, assume that the state of the graph is

represented as a vector of length N_v of states from the set L and the edges are represented as an adjacency matrix $A = [a_{ij}]_{N_v \times N_v}$ containing $\{0, 1\}$ elements with $a_{ij} = 1$ denoting that there exists an edge between vertices v_i and v_j. Then a procedure that transforms the current state S_t at a time step t to the state S_{t+1} at the next time step can be implemented as shown in Algorithm 1. In this procedure rand() is a function that returns a value randomly drawn from the uniform probability distribution on the $[0, 1)$ range.

Algorithm 1 Nondeterministic spreading of fire in the graph (one time step).

IN: S_t - the state of the graph at a time step t
OUT: S_{t+1} - the state at a time step $t + 1$ (to be calculated by this procedure)

$S_{t+1} := S_t$ // Copy the state to the next time step
for $i := 1, \ldots, N_v$ **do** // Allow fire to spread
  if $S_t[i] == 'B'$ **then**
    for $j := 1, \ldots, N_v$ **do**
      if $(A[i, j] == 1)$ and $(S_t[j] == 'U')$ and $(\text{rand}() < P_{sp})$ **then**
        $S_{t+1}[j] := 'B'$
      end if
    end for
  end if
end for
return S_{t+1}

It is worth noticing that in the deterministic version of the FFP the spread of fire in the graph is dynamic, but the optimization problem is, in fact, static. This follows from the fact that once a solution (i.e. a permutation) is selected the spreading of fire can be exactly simulated from the initial state to the very end. Therefore, the optimizer can always work with the initial state and can obtain an exact evaluation regardless if it is a single-objective problem or a multiobjective one. To the contrary, the nondeterminism makes the problem truly dynamic. Because it is not possible to predict the spread of fire with certainty, the solution chosen as the best one may change as time progresses depending on which nodes actually caught on fire and which did not.

As with any dynamic optimization problem two typical approaches are the **offline** and **online** optimization. In the offline approach we try to find the best permutation π of firefighters and then, as fire spreads, we assign firefighters using the same π at each time step. In the online approach the optimizer can take into consideration how the fire did actually spread and which nodes are burning at each time step $t > 0$. Therefore, a different permutation π_t can be produced at a time t step and used for assigning firefighters in this particular time step. In this paper we assume, that the already assigned firefighters cannot be reallocated, so the nodes that were defended remain defended and the new permutation π_t only affects the assignment of firefighters done after this permutation was generated.

3 Simheuristics

The prohibitive computational cost of running exact solvers on larger instances of some combinatorial problems can suggest the use, instead, of heuristics, or metaheuristics such as EAs. We adopt this approach here and use simulation of the spreading fire to enable the candidate solutions proposed by the EA to be evaluated. In previous work on the deterministic version of the FFP the evaluation of a permutation π was performed by simulating the spread of fire in the graph while simultaneously assigning firefighters according to the ordering determined by the permutation π. This approach can also be used for the nondeterministic version of the FFP. In fact, the approach called "simheuristics" – combining simulation with metaheuristic optimization was proposed in a recent survey [8] as a proper approach to nondeterministic optimization. Of course, in the case of nondeterministic problems each run of the simulation can yield different values of the optimization criteria. Therefore, in this paper we adopt an approach in which we perform a number N_{sim} of simulations for a given solution (permutation) π and then average the results. In order to speed up computations, in the experiments described in this paper the simulation routine was implemented for a GPU massively-parallel architecture using the CUDA technology. From the N_{sim} different values of the number of the saved nodes obtained in these runs we calculate a mean value for each of the objectives $\overline{f_j}$, $j = 1, \ldots, m$. Therefore, the implementation of the optimization algorithm is split into two parts: an EA that runs on the CPU and a simulation routine that runs on the GPU in N_{sim} parallel threads (see Figure 1).

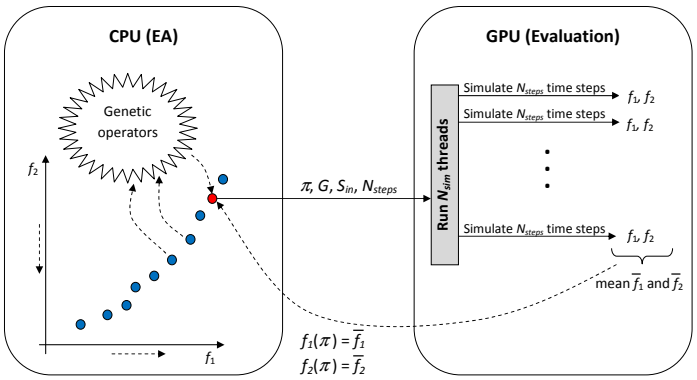

Fig. 1. Implementation of the simheuristic on a CPU/GPU machine.

Each thread simulates the spreading of fire starting from a given initial state S_{in} until a predefined number of time steps N_{steps} is completed or until fire can no longer spread. The working of a single simulation thread is presented in Algorithm 2.

The **SpreadingFinished**(S_t) function used in Algorithm 2 checks if the fire spreading has finished for a given graph state S_t. The spreading of fire is

Algorithm 2 The working of a single simulation thread.

IN: S_{in} - an initial state of the graph for the simulation
π - a solution to test

$t := 0$ // Simulation of the spreading of fire
$S_0 = S_{in}$
while not SpreadingFinished(S_t) and $t < N_{steps}$ **do**
  S_{t+1} := AssignFirefighters(S_t, π)
  S_{t+1} := SpreadFire(S_{t+1});
  $t := t + 1$
end while

Set $f_j := 0$ for $j = 1, \ldots, m$ // Evaluation of the final state
for $v \in V$ **do**
  if $S_t[v] \neq$ 'B' **then**
    for $j := 1, \ldots, m$ **do**
      $f_j := f_j + v_j(v)$
    end for
  end if
end for

considered finished if there are no untouched nodes adjacent to the burning ones, that is $\neg\exists i, j : (S_t[v_i] = \text{'B'} \wedge S_t[v_j] = \text{'U'} \wedge A[i, j] = 1)$, where A is the adjacency matrix of the graph G. The **AssignFirefighters** procedure assigns firefighters to those N_f untouched nodes that are placed first in the permutation π. The **SpreadFire** procedure performs one time step of the nondeterministic fire spreading according to Algorithm 1.

In this paper the Sim-EA algorithm [12] is used that was applied to the multiobjective FFP in a previous paper [13] and was found to outperform a well-known multiobjective optimization algorithm MOEA/D [10]. The Sim-EA is a decomposition-based approach in which several populations $P_1, \ldots, P_{N_{sub}}$ perform optimization with respect to scalar objectives obtained by aggregating the original ones using different weight vectors $\lambda^{(1)}, \ldots, \lambda^{(N_{sub})}$. In Sim-EA specimens can migrate between populations according to various migration strategies.

In the paper [13] several migration strategies were tested and the "rank" strategy that worked best is used this paper. For each destination subpopulation P_d, $d = 1, \ldots, N_{sub}$ all subpopulations P_s, where $s \neq d$ are ranked according to the dot product of weight vectors $\lambda^{(d)} \cdot \lambda^{(s)}$. The source population is selected using the roulette wheel selection with probabilities proportional to the ranks of the subpopulations. Then, N_{imig} best specimens from the source population are taken and merged into the population P_d. In the merge phase each migrated specimen is matched against the currently weakest specimen in the destination population P_d. The new specimen replaces the currently weakest specimen in the destination population P_d if it has a higher value of the objectives aggregated using the weight vector $\lambda^{(d)}$. The main loop of the Sim-EA algorithm is presented in Algorithm 3. It is very similar to the one used in papers [12] and [13], but a different stopping condition is used in this paper. In the previous works the

optimization was run for a preset number of generations N_{gen}. In this paper we use a time limit T_{max} in which the main loop of the algorithm has to finish, because we assume, that decisions have to be made in a given amount of time. Also, we are interested in how to use the available time effectively, by either making longer simulation runs (large N_{steps}) or making shorter simulation runs (small N_{steps}) allowing more generations for the EA. Because the EA runs on a regular CPU and simulations are performed in parallel on a GPU it is hard to find a common measure of the amount of computations done on both devices other than the running time limit.

Algorithm 3 The main loop of the Sim-EA algorithm (see also [12] and [13])

IN: $P_1, P_2, \ldots, P_{N_{sub}}$ - populations, one for each search direction $\lambda^{(d)}$
S_{in} - state of the graph for which to optimize
OUT: $P_1, P_2, \ldots, P_{N_{sub}}$ - populations after evolution

for $d := 1, \ldots, N_{sub}$ **do** // Initial evaluation
  Evaluate(S_{in}, $P_1, \lambda^{(d)}$)
end for
while not StoppingConditionMet() **do**
  for $d := 1, \ldots, N_{sub}$ **do** // Genetic operators
    P' := GeneticOperators(P_d)
    Evaluate(S_{in}, $P', \lambda^{(d)}$)
    $P_d := P_d \cup P'$
  end for
  for $d := 1, \ldots, N_{sub}$ **do** // Source populations
    s := SelectSourcePopulation(d)
    P'_d := the N_{imig} best specimens from P_s
  end for
  for $d := 1, \ldots, N_{sub}$ **do** // Migration
    for $x \in P'_d$ **do**
      Evaluate(S_{in}, $\{x\}$, $\lambda^{(d)}$)
      w := the weakest specimen in P_d
      $P_d := P_d - \{w\}$
      b := BinaryTournament($w, x, \lambda^{(d)}$)
      $P_d := P_d \cup \{b\}$
    end for
  end for
  for $d := 1, \ldots, N_{sub}$ **do** // Elitist selection
    e := the best specimen in P_d
    P_d := Select($P_d \backslash \{e\}$, $N_{pop} - 1$)
    $P_d := P_d \cup \{e\}$
  end for
end while

The main loop of the Sim-EA uses the following procedures:

StoppingConditionMet() – A function that determines if the stopping condition is satisfied, such as the numbed of generations completed, maximum running time, etc.

GeneticOperators(P) – Applies the crossover and mutation to specimens in P and returns the new specimens. In papers [11] and [13] a mechanism for adjusting probabilities of the application of several different crossover and mutation operators was introduced. The same mechanism is used in this paper. The same 10 crossover and 5 mutation operators were used as in the previous papers. The crossover operators were: Cycle Crossover (CX), Linear Order Crossover (LOX), Merging Crossover (MOX), Non-Wrapping Order Crossover (NWOX), Order Based Crossover (OBX), Order Crossover (OX), Position Based Crossover (PBX), Partially Mapped Crossover (PMX), Precedence Preservative Crossover (PPX) and Uniform Partially Mapped Crossover (UPMX). The mutation operators were: displacement mutation, insertion mutation, inversion mutation, scramble mutation and transpose mutation.

Evaluate(S_{in}, P, λ) – Evaluates specimens from a given set using N_{sim} parallel simulations performed on a GPU as shown in Figure 1 starting at the state S_{in}. In the simulation estimates of the average values $\overline{f_1}, \ldots, \overline{f_m}$ of m objectives are obtained. The fitness of the specimens is then calculated by aggregating these objectives using a given weight vector λ.

SelectSourcePopulation(d) – Selects a population P_s to migrate specimens from into P_d. Various strategies can be used for selecting the source population P_s. In the paper [13] several strategies were tested and the one that worked best with the FFP turned out to be to select the source population using the roulette wheel selection procedure based on ranking calculated using the dot product of weight vectors $\lambda^{(d)} \cdot \lambda^{(s)}$.

BinaryTournament(s_1, s_2, λ) – compares two specimens s_1 and s_2 according to the fitness calculated by aggregating the m objectives of the specimens using a given weight vector λ. Returns the winning specimen.

Select(P, n) – selects n specimens from a given population P using the binary tournament selection method.

Because this paper concerns a dynamic optimization problem, the optimization can be performed either in an offline or in an online mode. In the offline mode one long optimization run is performed based on the initial state of the graph. The best permutation π is selected and then, as fire spreads, firefighters are assigned according to this permutation in increments of N_f per a time step. Because in this paper a multiobjective problem is concerned, we select a different permutation $\pi^{(d)}$, $d = 1, \ldots, N_{sub}$ for each optimization direction $\lambda^{(d)}$. In the offline mode we simulate the spreading of fire for each optimization direction $\lambda^{(d)}$ separately starting from the initial state of the graph and assigning firefighters using $\pi^{(d)}$. The final evaluation E_d along each direction $\lambda^{(d)}$ is equal to the sum of the values of the non-burning nodes weighted using $\lambda^{(d)}$ when the simulation ends. The offline optimization is presented in Algorithm 4.

In the online mode the optimization is performed at each time step for a short period of time. The best currently known permutation is selected and

Algorithm 4 The optimization in the offline mode.

```
for d := 1, ..., N_sub do
    P_d := InitPopulation(N_pop)
end for
Evolve({P_1, P_2, ..., P_{N_sub}}, S_0, T_max)

for d := 1, ..., N_sub do
    π^(d) := SelectBestSolution(P_d)

    t := 0                  // Simulation of the spreading of fire
    S_t^(d) := S_0
    while not SpreadingFinished(S_t^(d)) do
        S_{t+1}^(d) := AssignFirefighters(S_t^(d), π^(d))
        S_{t+1}^(d) := SpreadFire(S_{t+1}^(d));
        t := t + 1
    end while

    E_d := 0                // Evaluation of the final state for the d-th subproblem
    for v ∈ V do
        if S_t^(d)[v] ≠ 'B' then
            for j := 1, ..., m do
                E_d := E_d + λ_j^(d) v_j(v)
            end for
        end if
    end for
end for
```

firefighters are assigned using this permutation when fire spreads in the current time step. For the multiobjective problem a different permutation $\pi_t^{(d)}$ is selected for each optimization direction $\lambda^{(d)}$ at each time step t. Because of that we have to keep the current state $S_t^{(d)}$ of the simulation at the time step t for each search direction $\lambda^{(d)}$. The final evaluation E_d along each direction $\lambda^{(d)}$ is equal to the sum of the values of the non-burning nodes in the state $S_t^{(d)}$ weighted using $\lambda^{(d)}$ when the simulation ends. The optimization in the online mode is presented in Algorithm 5.

The algorithms 4 and 5 use the following procedures:

InitPopulation(n) - Creates a given number n of new specimens initialized as random permutations.

Evolve($\{P_1, P_2, \ldots, P_{N_{sub}}\}, S_t, T_{max}$) - Runs the main loop of the Sim-EA algorithm described in Algorithm 3 with the maximum running time T_{max} as the stopping criterion. This run of the algorithm optimizes solutions based on a given graph state S_t.

SelectBestSolution(P_d) - Selects the best solution from a given population P_d with respect to fitness calculated using the weight vector $\lambda^{(d)}$.

SpreadingFinished(S_t) - Checks if the fire spreading has finished for a given graph state S_t. The spreading of fire is considered finished if there are no

Algorithm 5 The optimization in the online mode.

```
t := 0
for d := 1, ..., N_sub do
    P_d := InitPopulation(N_pop)
    S_t^(d) := S_0
end for

while ∃d not SpreadingFinished(S_t^(d)) do
    Evolve({P_1, P_2, ..., P_{N_sub}}, S_t^(d), T_max)
    for d := 1, ..., N_sub do
        if not SpreadingFinished(S_t^(d) then
            π_t^(d) := SelectBestSolution(P_d)
            S_{t+1}^(d) := AssignFirefighters(S_t^(d), π_t^(d))
            S_{t+1}^(d) := SpreadFire(S_{t+1}^(d))
        end if
    end for
    t := t + 1
end while

for d := 1, ..., N_sub do
    E_d := 0              // Evaluation of the final state for the d-th subproblem
    for v ∈ V do
        if S_t^(d)[v] ≠ 'B' then
            for j := 1, ..., m do
                E_d := E_d + λ_j^(d) v_j(v)
            end for
        end if
    end for
end for
```

untouched nodes adjacent to the burning ones, that is $\neg\exists i, j : S_t[v_i] = \text{'B'} \wedge S_t[v_j] = \text{'U'} \wedge A[i, j] = 1$, where A is the adjacency matrix of the graph G.

AssignFirefighters(S_t, π) - Modifies the state S_t by assigning firefighters to those N_f untouched nodes that are placed first in the permutation π.

SpreadFire(S_t) - Performs one time step of the nondeterministic fire spreading according to Algorithm 1.

4 Experiments and Results

The experiments were aimed at investigating three issues: comparing the online and offline optimization, comparing the two with some simple heuristics that determine how to place firefighters and determining the influence of the N_{steps} parameter on the quality of the obtained results. The larger the N_{steps} parameter the longer the simulation used for evaluating specimens in the EA. Therefore, we assumed that the entire main loop of the EA can run for at most T_{max} seconds and when N_{steps} is smaller more generations of the EA can fit within the same time limit.

The time limit for the online optimization was set to $T_{max} = 60$ seconds per a time step and to $T_{max} = 300$ seconds for the offline optimization. The equipment used for experiments was the Intel Q6600 CPU running at 2.4 GHz with 4GB of RAM with a 470 GTX GPU with 1.25 GB of RAM. However, the memory sizes did not play an important role in the experiments, because the actual memory usage was far lower than the available maximum. The number of simulations run in parallel was set to $N_{sim} = 200$. It is worth mentioning, that even on moderately advanced 470 GTX GPUs such number of threads can easily run in parallel, so the running time of the simulations cannot be significantly decreased by just lowering the N_{sim} number.

Data sets used in the experiments were created in the same way as used in a previous paper on the FFP [13]. In these data sets the graph G was generated by randomly determining, for each pair of vertices v_i, v_j, if there exists an edge $\langle v_i, v_j \rangle$. The probability of generating an edge was set to $P_{edge} = 2.5/N_v$, where N_v was the number of vertices. This value was used in order to ensure that the mean number of edges adjacent to a vertex was similar for all the instances.

Costs $v_j(v)$, $j = 1, \ldots, m$ assigned to vertices of the graph G were generated by drawing pairs of random values with the uniform probability on a triangle formed in $\mathbb{R}^2$ by points [0, 0], [0, 100], [100, 0]. With this method of cost assignment individual costs fall in the range $[0, 100]$ but also the sum of costs associated with a single vertex cannot exceed 100. Therefore it is not possible to maximize both objectives at the same time, so the ability of the algorithm to find good trade-offs can be tested.

The number of initially burning nodes in each graph was set to $N_s = 1$ and the number of firefighters to assign in each time step was set to $N_f = 2$. Because of the higher computational cost of the experiments involving several values of the P_{sp} and N_{steps} parameters the instances used in this paper were smaller than in [13] with $N_v = 30$, 40, 50, 75, 100 and 125. For each graph size N_v, 30 test instances $I^{(1)}_{N_v}, \ldots I^{(30)}_{N_v}$ were prepared following the procedure described above to allow comparing the algorithms on different, but fixed, test cases.

The nondeterminism is involved in how the fire spreads in the graph, therefore the same graph structure can be used for different values of P_{sp}. In this paper five values $P_{sp} = 0.3, 0.5, 0.7, 0.9$ and 1.0 were used. For the length of the simulation $N_{steps} = 2, 4, 6, 8, 10$ and ∞ were used. The last value of $N_{steps} = \infty$ causes the simulation to run until the fire can no longer spread regardless of how many time steps it takes.

The number of subpopulations was $N_{sub} = 20$ with each subpopulation size $N_{pop} = 100$. The number of specimens migrating between the populations was set to $N_{imig} = 0.1 \cdot N_{pop} = 10$. Similarly as in the paper [13] a set of 10 crossover operators and 5 mutation operators was used in the experiments with an autoadaptation mechanism used for adjusting probabilities of the usage of the operators. The number of new specimens generated by the crossover operator N_{cross} was equal to the population size N_{pop} and the probability of mutation was set to $P_{mut} = 0.05$. The minimum probability of selecting a particular op-

erator in the auto-adaptation mechanism was set to $P_{min} = 0.02$ for crossover auto-adaptation and to $P_{min} = 0.05$ for mutation auto-adaptation.

For each mode of optimization (online and offline) and for each P_{sp}, N_{steps} pair 30 repetitions of the optimization were performed. In each repetition the E_d values, calculated as shown in Algorithms 4 and 5 for $d = 1, \ldots, N_{sub}$ were stored.

The performance of the optimization algorithms was compared to heuristics that suggest where to place firefighters. Three different heuristics were tested:

- **Max degree** - Assign firefighters to those nodes first that have higher degrees among untouched ('U') nodes.
- **Max degree (adjacent)** - Assign firefighters to those nodes first that have higher degrees among those untouched ('U') nodes that are adjacent to burning ('B') nodes.
- **BFS** - Perform a breadth-first search (BFS) in the graph starting at the burning nodes that finds possible paths from the burning ('B') nodes to the untouched ('U') ones. For each untouched node v determine the smallest number of steps s it takes to get to this node from a burning one. Assign a value of $P_{BFS} = (P_{sp})^s$ to node v. Assign firefighters to those nodes first that have higher values of P_{BFS}.

These heuristics were used in each time step to select the best N_f locations for firefighters. If in one time step more than N_f nodes had the maximum score according to the selected heuristic, N_f nodes were selected randomly from them. An evaluation of the performance of the heuristics was done in a similar way to the evaluation of the offline optimizer (see Algorithm 4). For each optimization direction $\lambda^{(d)}$, $d = 1, \ldots, N_{sub}$, a simulation of fire was performed with a selected heuristic used for allocating firefighters. When simulation finished, the score E_d was calculated by adding cost values assigned to non-burning nodes weighted using $\lambda^{(d)}$. Similarly as with the optimization algorithms, the tests with the heuristics were repeated 30 times. In the k-th of the 30 repetitions of the tests the same test instance $I_{N_v}^{(k)}$ was used with both modes of optimization, all heuristics and all P_{sp} and N_{steps} values. Therefore, a range of methods was tested on a diversified set of 30 test instances, but each method worked in the same starting conditions.

Median values of the E_d score (see Algorithms 4 and 5) are presented in Tables 1-6. The best result obtained by a given type of methods (online, offline, heuristic) for a given P_{sp} is marked by an arrow '$\rightarrow$'. Table 7 shows the results of a statistical comparison of the optimizers performed using the Wilcoxon test at $\alpha = 0.05$. This comparison was performed between the 30 measurements obtained for the best value of N_{steps} for each of the optimizers. For smaller instances ($N \leq 50$) both optimization approaches are often comparable, even though the online method was significantly better four times (the offline one just once). For larger instances ($N \geq 75$) the online optimizer outperformed the offline one 12 times (with 3 cases undecided, none in favour of the offline optimizer).

Table 1. Median values of the E_d score obtained for 30 nodes.

Method		$\mathbf{P_{sp} = 0.3}$	$\mathbf{P_{sp} = 0.5}$	$\mathbf{P_{sp} = 0.7}$	$\mathbf{P_{sp} = 0.9}$	$\mathbf{P_{sp} = 1.0}$
Online optimization	$N_{steps} = 2$	1364.81	1228.16	959.29	703.56	676.03
	$N_{steps} = 4$	1386.63	1253.57	→ 1096.32	→ 914.42	→ 855.57
	$N_{steps} = 6$	→ 1388.47	1254.53	1064.32	889.36	826.10
	$N_{steps} = 8$	1382.90	→ 1276.79	1081.58	887.28	801.36
	$N_{steps} = 10$	1363.90	1241.03	1061.86	871.30	823.18
	$N_{steps} = \infty$	1382.90	1269.30	1088.25	882.06	828.57
Offline optimization	$N_{steps} = 2$	1189.17	817.99	653.77	566.83	534.29
	$N_{steps} = 4$	1329.74	1156.53	892.96	831.48	791.11
	$N_{steps} = 6$	→ 1341.22	1176.64	984.11	→ 888.73	866.42
	$N_{steps} = 8$	1323.47	1160.76	968.56	880.54	→ 869.50
	$N_{steps} = 10$	1322.26	→ 1187.83	→ 1010.22	886.86	859.31
	$N_{steps} = \infty$	1330.42	1155.04	965.01	885.53	819.07
Heuristics	Max. degree	850.32	676.07	549.14	416.60	385.08
	Max. degree (adjacent)	→ 1352.98	→ 1188.69	→ 907.59	→ 722.75	→ 703.93
	BFS	1294.78	768.77	752.02	562.32	488.12

Table 2. Median values of the E_d score obtained for 40 nodes.

Method		$\mathbf{P_{sp} = 0.3}$	$\mathbf{P_{sp} = 0.5}$	$\mathbf{P_{sp} = 0.7}$	$\mathbf{P_{sp} = 0.9}$	$\mathbf{P_{sp} = 1.0}$
Online optimization	$N_{steps} = 2$	1795.27	1545.76	903.35	720.89	661.38
	$N_{steps} = 4$	1803.58	→ 1677.30	→ 1294.07	→ 1002.20	→ 923.94
	$N_{steps} = 6$	→ 1806.78	1675.89	1114.45	913.54	872.59
	$N_{steps} = 8$	1800.09	1541.41	1064.09	895.61	872.05
	$N_{steps} = 10$	1795.85	1378.15	1049.11	901.01	877.04
	$N_{steps} = \infty$	1759.37	1420.91	1047.97	896.04	886.23
Offline optimization	$N_{steps} = 2$	1470.07	925.41	723.90	617.66	574.65
	$N_{steps} = 4$	1762.00	1310.05	975.70	846.58	843.00
	$N_{steps} = 6$	→ 1765.74	1512.08	→ 1101.16	→ 968.91	921.43
	$N_{steps} = 8$	1764.88	→ 1561.17	1051.66	938.20	920.22
	$N_{steps} = 10$	1729.15	1373.84	1055.34	948.79	→ 939.76
	$N_{steps} = \infty$	1720.21	1418.45	1032.26	951.91	904.45
Heuristics	Max. degree	1120.14	724.68	583.10	481.13	442.67
	Max. degree (adjacent)	1766.37	→ 1771.33	→ 1019.21	→ 813.96	→ 809.71
	BFS	→ 1848.82	894.25	825.46	552.55	539.19

Table 3. Median values of the E_d score obtained for 50 nodes.

Method		$\mathbf{P_{sp} = 0.3}$	$\mathbf{P_{sp} = 0.5}$	$\mathbf{P_{sp} = 0.7}$	$\mathbf{P_{sp} = 0.9}$	$\mathbf{P_{sp} = 1.0}$
Online optimization	$N_{steps} = 2$	1300.12	626.98	503.00	431.68	408.95
	$N_{steps} = 4$	→ 1386.02	679.02	615.17	→ 579.97	562.21
	$N_{steps} = 6$	1385.33	→ 875.73	→ 670.68	567.47	→ 563.33
	$N_{steps} = 8$	1269.63	795.62	634.16	575.82	556.82
	$N_{steps} = 10$	1313.69	784.41	634.71	577.18	553.11
	$N_{steps} = \infty$	1071.02	771.25	640.98	570.84	560.65
Offline optimization	$N_{steps} = 2$	850.63	601.22	471.63	412.41	390.39
	$N_{steps} = 4$	1081.48	718.70	610.37	572.84	552.81
	$N_{steps} = 6$	→ 1128.72	→ 799.05	→ 654.90	601.09	604.99
	$N_{steps} = 8$	1093.90	750.79	625.35	598.93	613.88
	$N_{steps} = 10$	1029.41	714.61	626.82	→ 604.06	611.61
	$N_{steps} = \infty$	921.29	698.84	629.26	601.70	→ 614.32
Heuristics	Max. degree	716.63	477.24	399.16	336.53	296.22
	Max. degree (adjacent)	→ 1419.40	→ 803.09	→ 577.10	→ 437.66	→ 440.70
	BFS	1275.95	581.44	473.22	337.26	342.84

Another aspect of the optimization is the number of the steps N_{steps} for which the simulation is carried out. The shorter the simulation the more generations can the EA perform in the same amount of time. Figure 2 shows how many times the best result was obtained for $N_{steps} = 2, 4, 6, 8, 10, \infty$. Clearly, letting the simulation run without limit can deteriorate the results of the optimization. For larger instances ($N_v = 75, 100$ and 125) the unlimited simulation length worked best for $P_{sp} = 0.3$ and 0.5, but for larger values of P_{sp} limiting the duration of simulations worked better. Although not conclusive, this observation

Table 4. Median values of the E_d score obtained for 75 nodes.

Method		$\mathbf{P_{sp} = 0.3}$	$\mathbf{P_{sp} = 0.5}$	$\mathbf{P_{sp} = 0.7}$	$\mathbf{P_{sp} = 0.9}$	$\mathbf{P_{sp} = 1.0}$
Online optimization	$N_{steps} = 2$	956.88	614.59	487.15	415.99	389.99
	$N_{steps} = 4$	981.13	620.49	531.32	571.25	550.45
	$N_{steps} = 6$	992.92	774.51	→ 685.25	→ 629.67	→ 596.58
	$N_{steps} = 8$	959.63	806.06	674.45	612.49	595.02
	$N_{steps} = 10$	1090.14	795.37	677.82	608.71	596.54
	$N_{steps} = \infty$	→ 1092.12	→ 807.45	671.97	610.85	593.34
Offline optimization	$N_{steps} = 2$	922.30	640.93	522.01	454.48	424.78
	$N_{steps} = 4$	→ 1042.61	700.90	608.54	567.38	587.20
	$N_{steps} = 6$	961.70	→ 715.41	→ 633.43	→ 585.37	600.81
	$N_{steps} = 8$	931.17	691.37	602.31	573.06	604.34
	$N_{steps} = 10$	895.80	676.03	598.90	579.47	→ 606.50
	$N_{steps} = \infty$	868.76	680.01	598.87	569.24	604.17
Heuristics	Max. degree	849.68	570.76	440.48	343.04	341.51
	Max. degree (adjacent)	→ 2272.92	→ 1114.03	→ 617.77	→ 526.89	→ 485.55
	BFS	2269.66	664.75	491.11	400.65	381.74

Table 5. Median values of the E_d score obtained for 75 nodes.

Method		$\mathbf{P_{sp} = 0.3}$	$\mathbf{P_{sp} = 0.5}$	$\mathbf{P_{sp} = 0.7}$	$\mathbf{P_{sp} = 0.9}$	$\mathbf{P_{sp} = 1.0}$
Online optimization	$N_{steps} = 2$	956.88	614.59	487.15	415.99	389.99
	$N_{steps} = 4$	981.13	620.49	531.32	571.25	550.45
	$N_{steps} = 6$	992.92	774.51	→ 685.25	→ 629.67	→ 596.58
	$N_{steps} = 8$	959.63	806.06	674.45	612.49	595.02
	$N_{steps} = 10$	1090.14	795.37	677.82	608.71	596.54
	$N_{steps} = \infty$	→ 1092.12	→ 807.45	671.97	610.85	593.34
Offline optimization	$N_{steps} = 2$	922.30	640.93	522.01	454.48	424.78
	$N_{steps} = 4$	→ 1042.61	700.90	608.54	567.38	587.20
	$N_{steps} = 6$	961.70	→ 715.41	→ 633.43	→ 585.37	600.81
	$N_{steps} = 8$	931.17	691.37	602.31	573.06	604.34
	$N_{steps} = 10$	895.80	676.03	598.90	579.47	→ 606.50
	$N_{steps} = \infty$	868.76	680.01	598.87	569.24	604.17
Heuristics	Max. degree	849.68	570.76	440.48	343.04	341.51
	Max. degree (adjacent)	→ 2272.92	→ 1114.03	→ 617.77	→ 526.89	→ 485.55
	BFS	2269.66	664.75	491.11	400.65	381.74

Table 6. Median values of the E_d score obtained for 125 nodes.

Method		$\mathbf{P_{sp} = 0.3}$	$\mathbf{P_{sp} = 0.5}$	$\mathbf{P_{sp} = 0.7}$	$\mathbf{P_{sp} = 0.9}$	$\mathbf{P_{sp} = 1.0}$
Online optimization	$N_{steps} = 2$	963.07	637.80	519.84	440.04	397.65
	$N_{steps} = 4$	868.58	600.06	501.59	461.53	482.77
	$N_{steps} = 6$	906.73	692.04	723.36	→ 660.97	→ 630.15
	$N_{steps} = 8$	917.48	893.64	756.26	658.30	625.60
	$N_{steps} = 10$	1108.00	907.41	→ 766.36	653.44	623.34
	$N_{steps} = \infty$	→ 1230.37	→ 912.10	751.03	656.34	621.58
Offline optimization	$N_{steps} = 2$	→ 961.28	688.38	571.53	491.56	462.26
	$N_{steps} = 4$	953.17	662.25	532.82	483.99	491.69
	$N_{steps} = 6$	891.00	653.68	→ 601.89	→ 566.59	559.24
	$N_{steps} = 8$	905.52	701.63	597.22	538.33	559.82
	$N_{steps} = 10$	943.15	→ 711.44	594.74	536.91	→ 563.81
	$N_{steps} = \infty$	936.90	701.85	597.07	539.40	559.68
Heuristics	Max. degree	917.63	587.32	500.42	414.06	377.07
	Max. degree (adjacent)	3957.17	→ 844.74	→ 672.27	→ 537.26	→ 503.02
	BFS	→ 3975.47	703.07	551.18	444.86	415.94

may indicate that for lower values of P_{sp} it is harder to predict the short-time movement of fire and thus averaged behaviour observed in the longer run becomes more important. The optimization algorithms were also compared with heuristic methods described in Section 4. Clearly, the best of the tested heuristics is the one that considers placing the firefighters adjacently to the burning nodes (see Figure 3). Statistical comparison of optimization algorithms with heuristics shown that the "Max degree (adjacent)" heuristic is a very effective strategy when the probability of the spreading of fire is low ($P_{sp} \leq 0.5$). It was signifi-

cantly better than the online optimizer in 7 out of 12 cases, significantly worse in 3 cases, with 2 cases undecided (p-value > 0.05). Compared to the offline optimizer it was significantly better in 11 cases, with only one case undecided. On the other hand, in the tests for $P_{sp} \geq 0.7$ the heuristic was significantly worse than the online optimizer 15 times out of 18 (3 cases undecided). Compared to the offline optimizer it was significantly worse 13 times and significantly better once (4 cases undecided).

Table 7. The results of a statistical comparison of the online and offline optimization modes: 'N' - online better, 'F' - offline better, '=' - no statistical difference.

N_v	$P_{sp} = 0.3$	$P_{sp} = 0.5$	$P_{sp} = 0.7$	$P_{sp} = 0.9$	$P_{sp} = 1.0$
30	=	N	N	=	=
40	N	=	=	=	=
50	=	N	=	=	F
75	=	N	N	N	=
100	N	N	N	N	=
125	N	N	N	N	N

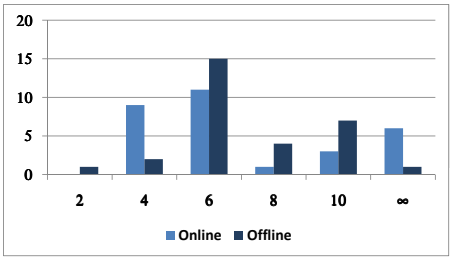

Fig. 2. The number of times each value of N_{steps} produced the best result.

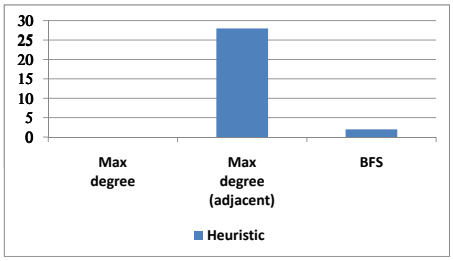

Fig. 3. The number of times each heuristic produced the best result.

These results might also indicate the fact that for lower values of P_{sp} it is harder to predict the short-time movement of fire. In such situation heuristics may be more effective than trying to assess what location of firefighters will be best based on simulations with a high level of uncertainty.

5 Conclusion

In this paper the multiobjective nondeterministic firefighter problem is studied. For solving this problem the Sim-EA multipopulation EA is applied that was shown in [13] to outperform the MOEA/D algorithm on the FFP. Solving the nondeterministic FFP requires dynamic optimization and therefore the effectiveness of the algorithm in the online and offline optimization modes is compared. For small problem instances both approaches are comparable, but for larger ones the online approach seems better. Also, it was tested how long to run the simulation that evaluates candidate solutions. Limiting the length of simulation runs often improves the results, but for low P_{sp} longer simulations may be better. Evolutionary optimization was compared with several heuristic approaches that

use simple rules for allocating firefighters. The heuristic that selects nodes with a high degree placed adjacently to the already burning ones worked by far the best. It significantly outperformed the online optimizer in several cases for $P_{sp} \leq 0.5$ and the offline optimizer in all cases except one. For $P_{sp} \geq 0.7$, however, the heuristic approach performed poorly. The observations concerning the length of simulation runs and comparison of optimization and heuristics seem to indicate that for low P_{sp} it is hard to obtain good evaluations of solutions using simulations. Because different types of methods work best for different probabilities of the spreading of fire, further work on the nondeterministic FFP may concern hybrid methods combining heuristics with online optimization.

References

1. Blum, C., et al.: The firefighter problem: Application of hybrid ant colony optimization algorithms. In: Blum, C., Ochoa, G. (eds.) Evolutionary Computation in Combinatorial Optimisation, LNCS, vol. 8600, pp. 218–229. Springer Berlin Heidelberg (2014)
2. Comellas, F., Mitjana, M.: Broadcasting in small-world communication networks. In: Kaklamanis, C., Kirousis, L. (eds.) 9th Int. Coll. on Structural Information and Communication Complexity. pp. 73–85 (2002)
3. Develin, M., Hartke, S.G.: Fire containment in grids of dimension three and higher. Discrete Appl. Math. 155(17), 2257–2268 (2007)
4. Feldheim, O.N., Hod, R.: 3/2 firefighters are not enough. Discrete Applied Mathematics 161(1-2), 301–306 (2013)
5. Garca-Martnez, C., et al.: The firefighter problem: Empirical results on random graphs. Computers & Operations Research 60, 55–66 (2015)
6. Hartnell, B.: Firefighter! an application of domination. In: 20th Conference on Numerical Mathematics and Computing (1995)
7. Hu, B., Windbichler, A., Raidl, G.R.: A new solution representation for the firefighter problem. In: Ochoa, G., Chicano, F. (eds.) Evolutionary Computation in Combinatorial Optimization, LNCS, vol. 9026, pp. 25–35. Springer (2015)
8. Juan, A.A., other: A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. Operations Research Perspectives 2, 62–72 (2015)
9. Kumar, R., et al.: Stochastic models for the web graph. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science. pp. 57–65. FOCS '00, IEEE Computer Society (2000)
10. Li, H., Zhang, Q.: Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II. IEEE Trans. on Evolut. Comput. 13(2), 284–302 (2009)
11. Michalak, K.: Auto-adaptation of genetic operators for multi-objective optimization in the firefighter problem. In: Corchado, E., et al. (eds.) IDEAL 2014, LNCS, vol. 8669, pp. 484–491. Springer (2014)
12. Michalak, K.: Sim-EA: An evolutionary algorithm based on problem similarity. In: Corchado, E., et al. (eds.) IDEAL 2014, LNCS, vol. 8669, pp. 191–198. Springer (2014)
13. Michalak, K.: The Sim-EA algorithm with operator autoadaptation for the multi-objective firefighter problem. In: Ochoa, G., Chicano, F. (eds.) Evolutionary Computation in Combinatorial Optimization, LNCS, vol. 9026, pp. 184–196. Springer (2015)