Applied Soft Computing 136 (2023) 110094

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

How to stop undesired propagations by using bi-level genetic algorithms☆

Javier Galiana [a], Ismael Rodríguez [b,a], Fernando Rubio [b,a,*]

[a] Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain
[b] Instituto de Tecnología del Conocimiento, Dept. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain

ARTICLE INFO

Article history:
Received 25 May 2022
Received in revised form 3 February 2023
Accepted 3 February 2023
Available online 10 February 2023

Keywords:
Computational complexity
Σ_2^P-completeness
Genetic algorithms
Propagation

ABSTRACT

In this work we introduce a general model to analyse any type of propagation system, whether it represents the spread of a fire, fake news, a virus, etc. We study the computational complexity of the problem of minimising the impact of a propagation by cutting some of the connections it can spread through. Even limiting the scope of our cutting strategy to be short-term, we show that the problem is Σ_2^P-complete, that is, it is one level above NP-complete in the complexity hierarchy. Intuitively, in Σ_2^P-complete problems a hard search for the value fulfilling some property is tackled, but just *evaluating* that property for each candidate value requires performing another hard, independent search. This complexity suggests that a good method to deal with the problem under consideration is a *two-level genetic algorithm*, that is, a genetic algorithm that uses another genetic algorithm as fitness function: the former algorithm searches for good candidates for solving the target optimisation, and for each candidate within its population, its fitness is calculated by running the latter algorithm. We apply this implementation to two case studies, compare its results with those of greedy and minimax algorithms, and report experimental results. Our results indicate that bi-level genetic algorithms are good candidates to deal with Σ_2^P-complete problems.

1. Introduction

Propagations, understood as phenomenons where a property iteratively expands to nearby locations, constitute a fundamental part of human existence. Let us consider, in particular, propagations where some harm expands through a geographical area, such as fires, fake news, plagues, and infectious diseases. A usual method to contain a propagation during the first moments after its outbreak, sometimes being actually the only one available, consists in isolating the propagation by disabling some of the connections it can use to expand to neighbour areas. Firewalls, system disconnections, travel restrictions, and lockdowns are examples of this selective elimination of connections.

These examples reflect the importance of studying propagations to understand their nature and, more importantly, actively combat them. Despite their differences, their common basic mechanics enables the definition of abstract models accurately capturing their behaviour. Indeed, propagations have been typically studied and modelled from a *statistical mechanics* point of view (see e.g. [1–4]). Alternatively, in this paper we propose seeing these systems as interactive *games* where some authority makes decisions (in particular, it cuts some connections), and the propagation reacts to these decisions by spreading through the connections being still present. Thus, rather than predicting the behaviour of these systems on average, or designing *a priori* general strategies against the propagation regardless of the specific scenario (e.g. *"always disconnect the computers with higher connectivity"*, *"always lock down big cities"*, etc.), we will study how to iteratively interact with the propagation so that the harm it can produce in the worst case is minimised.

We will define our common propagation model and will investigate the problem of finding an *immediate* strategy guaranteeing that the damage unleashed by the propagation's reaction will be under a given threshold. By immediate we mean a strategy designed for the short term, that is, one that does not aspire to consider the possible effect of advances in the spread over multiple actions in the future. As we will see, the computational complexity of the resulting problem will be so high that even dealing with this *immediate* case will be challenging by itself. In particular, due to the big number of choices under consideration

☆ Work partially supported by projects PID2019-108528RB-C22, and by Comunidad de Madrid as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union.

* Corresponding author at: Instituto de Tecnología del Conocimiento, Dept. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain.
E-mail addresses: javgal02@ucm.es (J. Galiana), isrodrig@ucm.es (I. Rodríguez), fernando@sip.ucm.es (F. Rubio).

https://doi.org/10.1016/j.asoc.2023.110094

even in simple settings, exploring the explosive number of possible choices and reactions up to some moderately far future (e.g. in the style of a minimax algorithm), will be unfeasible in general, so only some form of immediate decision making is practical. We will formalise this propagation mitigating problem, identify its computation complexity, and offer an implementation that heuristically solves this problem.

Before trying to solve any problem expected to reach some high hardness, it is convenient to determine its computational complexity as narrowly as possible, so that the techniques used in its resolution are consistent with the difficulty of the problem. For instance, Genetic [5–7] and Swarm Intelligence Algorithms [8–11] are known to be good options to deal with NP-complete problems, as it is shown for instance in [12–18]. However, if the problem under consideration is PSPACE-complete, then we have to use other techniques such as minimax with alpha-beta pruning [19,20]. Consequently, right after defining our problem, we identify its complexity, and in particular we prove that it is Σ_2^P-complete. Thus, even without considering the long-term effects of our strategies to combat spreads, the problem is complete in the second level of the Polynomial Hierarchy (PH), which puts it one level over NP-complete problems (see e.g. [21,22]).[1] Informally, solving an Σ_2^P-complete problem requires performing a hard search for the value fulfilling some property -and evaluating that property for each candidate value requires performing another hard search which is independent from the other.

This computational complexity implies that classical heuristic optimisation techniques are not feasible to solve the problem. For instance, using a *single-level*, standard genetic algorithm is not feasible either; let us see what we mean by single-level here. Note that facing NP-complete problems requires performing some difficult optimisation search, whereas facing Σ_2^P-complete problems requires performing a kind of *double-level* search: we perform a difficult search for a good value in a setting where just *evaluating* how good a value is requires performing another difficult search. In our implementation, we will handle this two-level search by using a *two-level genetic algorithm*: we run a genetic algorithm where the fitness function will be evaluated, for each candidate solution, by running another genetic algorithm. Hence, according to this scheme, the higher-level genetic algorithm will use (i.e. invoke) the lower-level one.[2] We present a solution to our problem based on this two-level genetic algorithm, and we prove its practical utility by applying it to different case studies and comparing its performance with that reached by greedy and minimax algorithms. The results obtained are satisfactory, which suggests the applicability of two-level genetic algorithms like the one presented here to other Σ_2^P-complete problems in general.

The rest of the paper is structured as follows. In the next section we review related work. Then, in Section 3 we introduce the formal description of the propagation problem, and we sketch the main ideas needed to prove that the problem is Σ_2^P-complete (the complete proof can be seen in Appendix A). Next, in Section 4 we present our two-level genetic algorithm and its application to two concrete case studies. Finally, in Section 5 we present our conclusions and lines for future work.

It is worth noting that the paper has two parts that are partially independent. On the one hand, Appendix A is devoted to the theoretical complexity analysis. On the other hand, Section 4 deals with the practical solution of the problem. Although the first part is relevant to justify the practical resolution method used in the second part, practitioners can skip the details of the proof without missing important information to understand the rest of the paper.

2. Related work

As it was commented in the introduction, propagations have been typically studied and modelled from a *statistical mechanics* point of view. For instance, in [1] hierarchical cluster analysis is used in order to study the main characteristics of the largest forest fires taking place in Greece. After using the Ward method [23] to define the clusters, a discriminant analysis is used on these clusters to create canonical components. Analogously, in [3] the authors use Prioritised Coloured Petri Nets to analyse the evolution of unhealthy conditions in the context of the sick building syndrome. Stochastic initial markings are used, and then a stochastic quantitative analysis of the results is performed.

Another typical case study of propagations is the spread of diseases. In this context, simulations based on stochastic models are frequently encountered. For example, in [2] a two-dimensional square lattice on which individuals are housed is considered, as well as a set of probabilistic rules for an individual to become infected or die, as well as to move to other locations. From this, a large number of simulation runs are used to calculate the most feasible scenarios. A similar approach is used in [4], where the authors also use a similar two-dimensional square lattice. The difference is that now the authors analyse the influence of mobility. That is, they conduct different simulations analysing the expansion of diseases depending on the mobility radius allowed to each agent. Moreover, they also simulate different vaccination strategies when the number of available vaccines is limited.

Let us remark that, in our problem, we search for good choices in the immediate term, based on a two-step approach: the appeaser breaks connections, and next the propagation tries to create the highest damage through existing connections. If we ignore this two-step approach and focus only on the second step (i.e. we assume that the appeaser decision was already made, and only the propagation reaction must be calculated), then probably the closest problems are those defined as transportation problems (see e.g. [24,25]). In these problems, a set of providers must serve some amount of supply to a set of consumers. Providers serve some known amounts of supply each, and consumers must receive some known amounts each as well. Each provider is connected to one or more consumers, and the problem is deciding how providers should split their supply among its consumers in such a way that all consumers are fully served from their suppliers. Usually, it is assumed that the addition of supply of all suppliers matches exactly the demand of the consumers, and under this restriction the problem is polynomial (although it becomes NP-complete if some constraints are added). Note that, in our setting, the difficulty of the propagation lies in choosing which nodes should be infected to maximise the damage. Hence, that perfect match between total demand and total supply cannot be assumed in our case, and in particular this makes the

[1] The decision problems in NP can be stated as, given input x, finding out whether there exists y with polynomial size w.r.t. x such that some polynomial-time property P on x and y holds. On the other hand, problems in Σ_2^P are those stated as, given input x, finding out if there exists y of polynomial size such that, for all z of polynomial size, some polynomial-time property P on x, y, and z holds.

[2] Recall the roles of terms y and z in the previous footnote. When facing an NP-complete problem, a genetic algorithm searches for solutions y for x. When facing a Σ_2^P-complete problem, our two-level genetic algorithm will search for solutions y for x, where the fitness of each solution y will be computed by searching for second-level solutions z for x and y. We illustrate this idea with an informal example: suppose we wish to find the most robust way y to defend a castle x. A genetic algorithm explores possible castle defences y. Each of them is assigned a fitness value in terms of the worst-case attack z this defence y would *not* be able to properly contain (e.g. a wooden fence could be set on fire, and the fire could propagate to nearby houses). In order to approximately guess those worst-case attacks, another genetic algorithm is run *for each* candidate defence y under consideration, which seeks for the worst attack z this defence could suffer. Hence, this (lower-level) genetic algorithm is run every time the other genetic algorithm (the higher-level one) needs to calculate the fitness of any of its individuals.

"secondary optimisation" of the Σ_2^P-complete problem hard by itself.

If we focus on the algorithmic complexity of our problem rather than on the problem itself, we can find other Σ_2^P-complete problems that have been treated in the literature. However, due to their complexity, there are very few cases in which solutions have been presented for them. We may highlight the work presented in [26], where a competitive facility location problem is considered: a company wants to maximise its market share after (i) it places some facilities in some area *and* (ii) immediately afterwards, a follower company enters the same market by placing its own facilities (it is assumed that customers choose their nearest facility). Thus, the first company has to solve an optimisation problem where the evaluation of each alternative requires solving another optimisation problem for the second company. A genetic algorithm with tabu search is used for the upper-level optimisation. For the lower-level, a combination of upper and lower bounds of the (unknown) optimal value is used, where these bounds are obtained by executing a greedy algorithm and a linear programming relaxation with different accuracy targets. In contrast, we stick to metaheuristics in both optimisation levels in our propagation problem, and to the best of our knowledge, this is the first time a genetic algorithm using another genetic algorithm for calculating its fitness function is used to solve a problem proved to be Σ_2^P-complete. Of course, other configurations of methods for both levels of optimisations could be applied to our problem, and we consider exploring them in our future work.

3. Definition

In this section we will define the problem in question, of which we will later study and prove its difficulty, directly affected by the representation we will choose now. Thus, in this section, we will see the complexity established for the definition of our problem and some generalisations that preserve it.

We will consider a very simple representation, based on the fact that the propagation will start at some nodes of the graph and will continue to propagate depending on some propagation parameters (positive integers), while an *appeaser* (the authority) will try to stop it at all costs, obstructing its progress through the graph by cutting a certain number of edges of it. The simplicity of our definition of the problem and the impossibility of representing all the variables that influence the progress of the propagations will prevent us from representing what happens in real life in certain situations, such as the fact that the propagations do not always have the same intensity. The importance of simplifying the representation of the problem lies in the fact that demonstrating the difficulty (in particular, complexity hardness) of simpler problems is much more interesting since any generalisation of a problem always has at least the same difficulty as the particular problem. We begin by defining the concepts that will lay the foundations of our problem.

Definition 1. A propagation graph is an undirected graph $G = (V, E)$, where each node $v_i \in V$ is a tern $v_i = (i, a_i, b_i)$ where i represents the node identifier, a_i represents the pressure required for the node to be reached, and b_i is the propagation pressure capacity that v_i possesses to be able to reach other nodes. Note that, in this way, $a_i, b_i \in \mathbb{Z}^+ \cup \{0\}$ and two nodes may have the same values for a and b, being univocally differentiated by their identifier. To facilitate the notation from now on, we will omit the identifier from the tuple because, in our case, we are particularly interested in the charges (i.e. pressures) that will generate the propagation. We will call the propagation set to the set $P \subseteq V$ of nodes of the graph that have been reached by the propagation (*infected*) and it is clear that $\overline{P}$ are the nodes that have not yet been reached by it (*uninfected*).

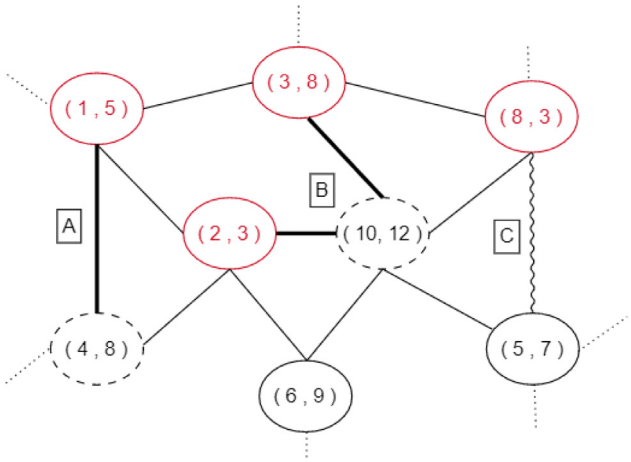

Fig. 1. Example of different ways of propagation.

Note that, in our propagation graph definition, we allow the existence of *multi-edges*, which will be useful later on and will help us to represent specific cases such as, for example, different roads connecting two cities. Now, let us introduce the formal definition of propagation within a graph. Next, we assume that $Adj(p)$ denotes the set of nodes adjacent to node p in the graph.

Definition 2. We say that the set of nodes $R \subseteq V$ can be infected from the set of nodes $P \subseteq V$ in a propagation step when each node $p_i \in P$ can divide its propagation capacity b_i into r_i parts $b_i^j \in \mathbb{Q}^+ \cup \{0\}$ that represent the part of its capacity p_i that goes into $q_j \in R$ if p_i and q_j are adjacent, satisfying:

$$\forall q_j \in R, \quad \sum_{p_i \in P \cap Adj(q_j)} b_i^j \geq a_j,$$

and also

$$\forall p_i \in P, \quad \sum_{j=1}^{r_i} b_i^j \leq b_i.$$

We will assume that b_i^j will be 0 as long as p_i and q_j are not adjacent or the load of p_i is not needed to reach q_j (since there are already other nodes belonging to P that are able to infect it by themselves).

Those free (i.e. uninfected) nodes that are directly connected to the nodes belonging to the propagation are said to belong to the *war zone*, where the action will take place. We present below an example in Fig. 1, where the propagation tries to advance to other nodes that have not yet been reached. In red we present the nodes that already belong to P; the nodes with the dashed outline are the ones that are going to be reached by the propagation in this turn, and the rest are the ones that are still safe.

Situation A in the graph in Fig. 1 is a standard scenario, where propagation proceeds from node $(1, 5)$ to node $(4, 8)$ because the former has a propagation capacity greater than the resistance of the latter. In scenario B, however, nodes $(3, 8)$ and $(2, 3)$ are not able to infect node $(10, 12)$ separately, and must join forces to achieve their goal, adding their infective loads to overcome the pressure needed to reach the target. Finally, in situation C we observe that the propagation is not able to reach node $(5, 7)$ because it only has node $(8, 3)$ to attack and its charge is not enough to neutralise the resistance of node $(8, 3)$.

Having thoroughly understood the role of propagation within the network and the problems it can generate, we now present the role of the *appeaser*. Its objective will be to attempt to reduce

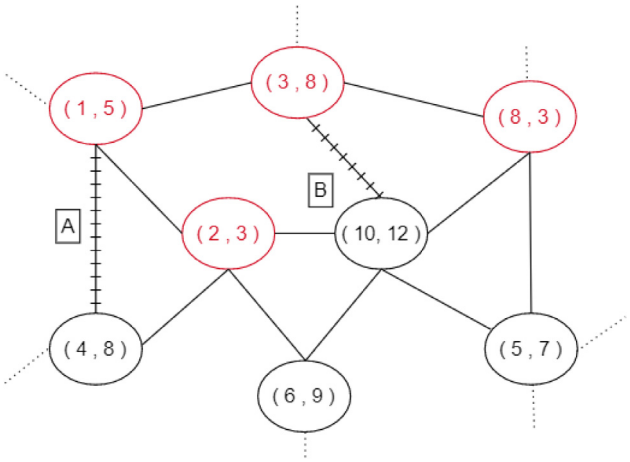

Fig. 2. Example of how the appeaser works.

the propagation's spread and, consequently, its damage by eliminating edges of A from the propagation graph hoping to be able to isolate it and save as many nodes as possible from its reach. Let us now look at an example of how the appeaser works with the help of Fig. 2.

Thus, we present in Fig. 2 the best case for the action of the appeaser, which cuts the edges A and B so that the vertices of the propagation are not able to propagate to any of their neighbouring nodes. We can intuit that, by performing this optimal action, we will be able to slow down the progress of the propagation in this area, minimising the damage caused by the propagation. This will not always be true, because on other occasions, even if the appeaser plays its best cards, it will be doomed to lose.

We are now ready to define our propagation game, where the appeaser will try to ensure that the propagation does as minor damage as possible.

The propagation game, which we denote as PG, is defined as follows. Let $G = (V, E)$ be a propagation graph and suppose that l and k are input parameters of the game. Consider that the propagation and the appeaser play the next game in turns.

The initial configuration of the game c_0 satisfies the condition $|P| = p_0$, where p_0 represents the number of nodes where propagation initially starts. Such nodes are all those where their information is $(0, b_j)$, that is, they do not need any external pressure to be infected.

Players will alternately take turns to execute a change in the propagation graph. The appeaser will start and in each of its turns will remove a maximum of k edges from the set A to prevent the propagation from continuing through the graph. Thus, the set of edges A will be modified over the turns so that the number of edges decreases as the turns pass, hindering the progress of the propagation.

On the other hand, in each of the turns of the propagation, it will advance from the infected nodes to other nodes that, at the beginning of the turn, had not yet been reached, that is, those that do not belong to P. The propagation has no limit to the number of nodes it can advance to, as long as these propagations are valid according to Definition 2. Note that the b_i value of each of the nodes can only be used to propagate once per turn, although it can be split over several nodes not yet infected. After the propagation turn, the new nodes reached by the propagation would be added to the set P of nodes reached.

Definition 3. Let G be a propagation graph and P be the set of nodes reached by the propagation. We will denote an instance or configuration of the game at a turn $i \in \mathbb{N}$ by the tern (G_i, P_i, i), where G_i and P_i denote the state of the graph and the propagation set at turn i.

For this purpose, let c_i and c_{i+1} be two consecutive game configurations. We say that a move z brings the game from c_i to c_{i+1} if, in case the turn is possessed by the propagation (i.e., i is odd), the propagation advances according to Definition 2, and in case the turn is possessed by the appeaser (i even), certain edges are cut, modifying in each case the game configuration as we have explained above.

When we deal with the propagation game turn by turn, we cannot foresee in advance a sequence of configurations that the game will go through. In each turn, the appeaser is free to choose some moves that will condition how the propagation will progress in the next turn, and this will again condition the appeaser's next moves, etc. This exchange of turns and movements between players will eventually lead to a situation in which one of them will finish the game, either by infecting all nodes (propagation) or by blocking further propagation (appeaser). Our goal in the decision problem might then be to decide whether or not it is possible to guarantee the appeaser's victory, in the sense that the appeaser is able to slow down the spread so that it is not able to reach an infection of l units. As it can be expected, when we consider the optimisation problem, we will try to minimise the total infection that the propagation can obtain at the end of the game. In order to do that, we will try to make the best possible decision in each appeaser's turn.

We can define the tree of possible game events from some configuration on, or just *game tree* (see Fig. 3), as a tree where each node represents a configuration of the game (the root denotes the current configuration), each edge connects each configuration to each possible next configuration according to the game rules, and the verdicts about the total infection the propagation has achieved in each case are taken at the leaves of the tree. Moreover, we can also define a possible combination of *decisions* of one player for each possible foreseeable future by taking only a *portion* of that tree from its root. An example of such a portion is also depicted in Fig. 3, where we can see it by considering only those nodes and edges with darker stroke. Assuming that now it is the turn of that player, we can see that only *one* edge leaving the root is taken, representing the current move chosen by the player. This edge leads to another node where the opponent player moves. In this node, the opponent can make any move allowed by the game, so *all* edges of the original game tree leaving this node are taken. For each of the nodes reached by these edges, our player moves next again, so a single edge going down is taken from each, denoting the move actually chosen by our player in each of these possible future situations. We can go on in the same way: for each taken node representing a configuration where it is that player's turn (i.e. the depth is odd), there is a *single* edge representing the choice the player has decided to make in that possible future situation; and for the node this edge reaches, *all* edges leaving this node in the original game tree are taken, as all possible moves open up for the opponent. The resulting subset of the original tree defines a possible *strategy* for the player, as it denotes a possible way in which the player could play the future game turns depending on the corresponding choices made by the opponent player. Each sequence of the edges from the root to the leaves in a game tree (or in a strategy) represents a possible sequence of future decisions taken by both players, where each new level of possible moves depends directly on the previous level produced by the other player's decision.

We will consider that a *short-term strategy for the appeaser* is a strategy that only goes through a certain fixed depth of a strategy tree. In our case, this depth will always be two, allowing the appeaser to consider the possible moves of its opponent depending on the decision it makes in its own previous turn (see

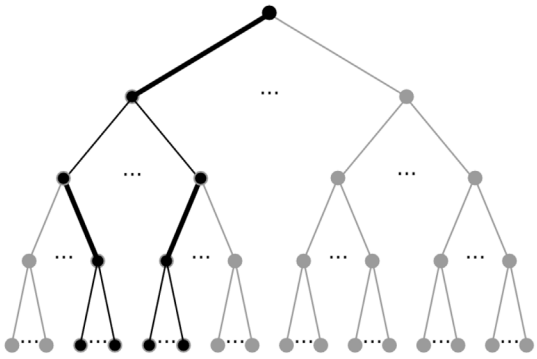

Fig. 3. Game tree and strategy examples.

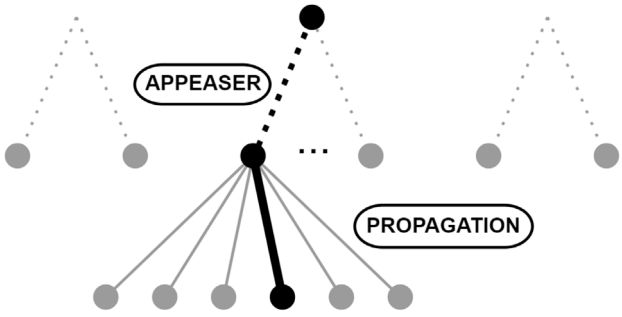

Fig. 4. Short-term strategy example.

Fig. 4). But how can we be sure that a certain move could be successful or not for the appeaser?

The moment the appeaser makes its decision and cuts the corresponding k edges, it will be the turn of the propagation, which will advance towards those nodes whose coverage is possible in terms of Definition 2. Consequently, the *optimal* short-term strategy for the appeaser will be the one that manages to minimise the damage produced by the propagation next, which we will measure as a function of the sum of the propagation capacities b of the nodes reached by the propagation. Thus, we will be interested in those short-term strategies guaranteeing that the total propagation damage will be below a limit l given as a game instance parameter.

Definition 4. The search for a short-term strategy, denoted by SSTS, is the following problem: given a configuration $c_i = (G_i, P_i, i)$ (not necessarily initial) of the propagation game, where turn i corresponds to the appeaser, with game parameters $l, k \in \mathbb{N}$, is there a short-term strategy for the appeaser such that, for any move, it makes after the propagation, the sum of the load capacities of the new nodes it has infected in the configuration $c_{i+2} = (G_{i+2}, P_{i+2}, i+2)$ is less than l, i.e., that $\sum_{v_i \in P_{i+2} \setminus P_i} b_i < l$?

Once the decision problem is known, it is trivial to define the propagation optimisation problem, where we are interested in finding the short-term optimal strategy, which we denote as SOSTS. Given the same input data as before except l, the SOSTS consists of finding a short-term strategy that minimises the sum of the load capacities of the nodes reached by the propagation in its next turn.

Next, we will introduce the complexity of SSTS:

Theorem 1. SSTS $\in \Sigma_2^P$-complete.

The proof of this result (see Appendix A for the complete proof), includes in its first lines, a straightforward method to prove that the problem belongs to Σ_2^P. Then, when it comes to proving Σ_2^P-hardness, we will construct a polynomial reduction from the well-known $QSAT_2$ problem [27]. This problem consists of, given an expression $\exists\, x_1, \ldots, x_n\ \forall\, y_1, \ldots, y_m\ \varphi$ where φ is a propositional formula denoted in DNF (Disjunctive Normal Form) depending only on the propositional variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, check whether the expression holds. Given this instance of $QSAT_2$, we will create an instance of SSTS from it, such that there exists an x' that makes it impossible for the formula φ' to be satisfied for all y' if and only if in the instance of SSTS the propagation fails to reach the proposed total load limit l after the appeaser cuts k edges in a certain way. The propagation will succeed in reaching that bound if and only if the appeaser fails to find a set of edges to cut that prevent the propagation from advancing to that value.

To show how easily we can generalise our problem to represent more complex scenarios, we will briefly introduce some examples of these extensions. On the one hand, we could set the graph to be a weighted graph, in which each edge has a weight associated with it and, therefore, the appeaser has a maximum weight to eradicate at each turn. Actually, it is common when we talk about fire spread in a forest fire that there is greater difficulty in building firebreaks in some areas than in others. We can be sure that the hardness of the problems resulting from adding such modifications to our problem would remain at least the same (Σ_2^P-hard). This hardness is trivially inherited from the original problem since they generalise it.

4. Implementation

In this section we will present our algorithm used to deal with the SOSTS problem: a double-level genetic algorithm, with which we intend to find out, given a configuration of the propagation game, what is the first step that the appeaser should take to try to minimise the damage that the propagation can do in its next move.

Without further ado, let us examine the direct application of genetic algorithms to our problem where, on the one hand, we will use a genetic algorithm to study which is the best movement for the appeaser, and for each of them, we will use another genetic algorithm that will find the best one for propagation, the result of the first one will depend entirely on the second one. Although we could consider using diploid genetic algorithms (see e.g. [28,29]) to increase the diversity of the population, we have decided not to use them: due to the high computational cost we face, the extra cost due to the use of diploid genetic algorithms would worsen our exploration capability.

Let us first look at those common functions that in both will define the Selection and Crossover processes, as they do not depend directly on the chosen problem instance. For the Selection process, we will use Tournament Selection. The core idea is to select based on direct comparisons between individuals. Subgroups of size 2 (pairs) of individuals are chosen from the population. The members of each pair compete with each other and the best one is chosen, using the fitness function, to be crossed. This method has certain elitism as the best individual of each pair is always chosen, however, not necessarily the best in the population will be selected to compete. The total number of pairs will be chosen based on other parameters (in particular, the number of genes and the crossing probability) for each instance. For the Crossover process, we will use the One-Point Crossover technique.

We now move on to develop the functionality of the methods that compose the genetic algorithm for the appeaser. In this case, a chromosome will reflect those edges chosen to be removed, the size of the chromosome thus being equal to the k parameter of our propagation set. Regarding the size of this population, it

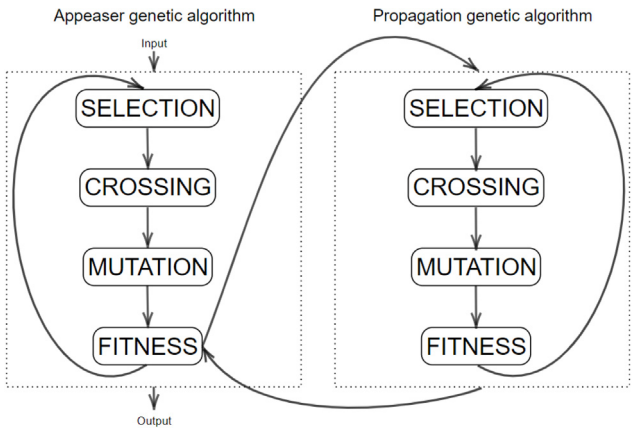

Fig. 5. Implementation idea.

will depend on the specific instance and its difficulty, increasing it in those cases where it is necessary to guarantee more variety within the same population. The generation of our *initial population for the appeaser* will be as follows: we first calculate those edges whose cut can influence the next movement of the propagation and, for each chromosome of the population to be generated, among these edges we randomly choose the k edges that will constitute the chromosome, avoiding repetitions within it.

Once the selection and the crossover have been carried out as we have explained above, it is the turn to mutate the individuals of the population resulting from the crossover. The *mutation process for the chromosomes related to the appeaser* is responsible for randomly choosing a gene from each individual in the population to mutate and, with a certain probability, to mutate it or not. This mutation consists of its substitution, in this case, by another edge of those that can be cut, checking again that the chosen edge does not already belong to the chromosome.

The last relevant process within the implementation of the genetic algorithm for the appeaser is the fitness function which will be the key to linking the two genetic algorithms. This fitness function is responsible for giving, to each chromosome, a value that represents its fitness according to the goal of the appeaser. To this end, we will use another genetic algorithm that will focus on finding the optimal movement for the propagation and whose implementation we will see later. Thus, given a chromosome from the population of the appeaser algorithm, we perform the cuts in the game graph, eliminating the selected edges in the chromosome. Then, we generate the instance of the propagation set with that graph and run the genetic algorithm for propagation, which will return the optimal movement for propagation from that instance and its heuristic value. That value will be the one we associate with the chromosome belonging to the population of the appeaser. The general idea of how both algorithms connect with each other is shown in Fig. 5.

Now let us take a closer look at how each step of the genetic propagation algorithm works.

This algorithm is designed to find the best movement for the propagation so each chromosome will reflect all the existing nodes in the propagation graph. Each gene in the chromosome will be defined by 1 or 0, representing respectively whether propagation will try to move towards it or not. The size then of an individual will be $|V|$ and, again, the number of individuals within the population will be a parameter to be set on a per instance basis. The *initial population for the propagation* will be generated in a similar way as it was done for the case of the appeaser. In this sense, once we have calculated the nodes that the propagation can reach, we will randomly choose how many we want it to try to reach and, subsequently, we will randomly choose that number of nodes among the available ones, setting its gene to 1.

After generating the initial population for the propagation, we carry out the selection and crossover processes and, with the resulting population, we carry out the mutation. When it comes to the *mutation of each of its chromosomes*, it will be made by modifying one of its genes. First, a gene of the chromosome is randomly chosen to mutate and, with a certain probability, it is mutated or not by choosing one of the nodes that can reach propagation and adding one to its gene (applying *mod*2 to ensure that the value is between 0 and 1). It should be noted that if the node chosen is already present in the chromosome (its gene has the value 1), it will be removed from the chromosome and no longer tries to reach it from the propagation.

Finally, the fitness function for the propagation will be in charge of, first, checking that the propagation represented by the chromosome is a feasible propagation and, in addition, assigning to the chromosome a heuristic value representing its fitness. To check the feasibility, we will construct a linear programming problem. The objective of such a problem is to find out if the propagation has a way to distribute the infective loads of its nodes in such a way that, in its turn, it manages to colonise exactly the considered combination of target nodes. Note that we are only interested in knowing if there is a way to reach such a combination. The linear propagation problem considers the objective function and two types of constraints. Let then be a certain subset of nodes (fr) that belong to the war zone and are reachable from any of the nodes belonging to the propagation, that is, there exists some valid propagation in terms of Definition 2 for that node; for each of these, let ady be its list of adjacent nodes belonging to the propagation, and let $pr \subseteq P$ be the total set of nodes belonging to the propagation involved in this particular propagation (which will be all nodes in P that are adjacent to some node in fr).

We build the linear programming problem as follows

$$\begin{aligned} \min \quad & \sum_{i,j} x_{i,j} \\ \text{sa} \quad & \forall j \in fr \quad \sum_{i \in ady(j)} x_{i,j} \geq a_j, \\ & \forall i \in pr \quad \sum_{j \in pr(i)} x_{i,j} \leq b_i, \\ & \forall i \in P, j \in fr \quad x_{i,j} \geq 0, \end{aligned}$$

where the variables $x_{i,j}$ will represent the loading dose sent by the node belonging to the propagation $i \in ady(j)$ to the node belonging to the war zone $j \in fr$. It should be noted that the first type of constraint represents the sum of loads necessary to be able to reach a node $j \in fr$ and, therefore, it is necessary to sum all possible loads of those nodes adjacent to $j \in fr$ that belong to the propagation, and this sum must be greater than or equal to the resistance a_j of node j. Of this type, there are l constraints, with l being the total number of nodes that are in danger, i.e., the cardinal of fr. On the other hand, the second type of constraint focuses on the limits that each of the nodes belonging to the propagation ($i \in pr(j)$) that are related to the nodes of fr have to emit load and advance to these. For this reason, these restrictions consist of the sum of the doses of charge emitted by each node, which will always have to be less than or equal to the maximum available to b_i.

Actually, we are not very interested in the objective function and its result after solving the problem, but simply in whether the problem has a feasible solution or not, that is, whether the propagation between these nodes is possible or not. Even so, we opted for a minimisation problem for the simple intuitive reason

of using the smallest possible number of loads, which is what would naturally occur in a real direct application and, for the same reason, the objective function is the sum of all the doses.

Once the feasibility of the propagation represented by a chromosome has been verified, we must give it the corresponding fitness value. If the propagation is not feasible, that value will be 0 because we are not at all interested in this chromosome and the main objective of the propagation is to *maximise* its fitness function. On the contrary, if the propagation is feasible, its fitness value will be the sum of the propagation loads of those nodes that it has managed to reach, as we have seen throughout this paper.

To summarise, we have a genetic algorithm that looks for the optimal short-term strategy for the appeaser, basing its decisions on another genetic algorithm that looks for the best movement for the propagation. Therefore, it is of vital importance to find the ideal configuration for each of them, especially those decisive parameters that can make the performance of both algorithms optimal or not, trying to avoid at all costs premature convergences that could return local minima instead of global minima. These parameters are the number of individuals per generation, the number of generations, the crossover and mutation probabilities and the number of individuals selected in the selection by tournament, which we will have to set according to the peculiarities of each instance. It is vital to find the appropriate parameters for each of the genetic algorithms (they can be different from each other) because, since both genetic algorithms are linked, a solution will be optimal if and only if the optimal solution is found in both of them. As we will see later, it may happen that, if one of the two parts (appeaser or propagation) does not find the optimal solution, data may appear distorted by suboptimal solutions, which will sometimes make the results confusing.

The probability of mutation, crossover, and the number of individuals are decisive factors that directly influence the biodiversity of our population over generations. Therefore, small changes can cause the convergence of our algorithm to be anticipated or delayed. For this study, we have fixed the number of individuals in the population in order to examine how the variation of the other parameters affects a population of invariable quantity. For the crossover probability, the range of possible values studied varies from 0.9 to 0.5, while for the mutation probability, the values will vary between 0.9 and 0.05. After testing our algorithm with the possible combinations between these values we reach certain conclusions.

The variation of mutation probability values has two key aspects. Our experiments indicate that there is not a statistically relevant difference when we work with probabilities varying from 0.1 to 0.15. The results obtained are basically the same, and the convergence time varies less than 2%. In fact, mutation probabilities up to 0.2 obtain analogous results, but with convergence time increasing around 10%. However, when we consider larger mutation probabilities the results worsen. Especially, if we take this parameter to the limit, choosing values like 0.9, the results are much worse and the convergence time increases by an order of magnitude, the reason being that the algorithm is nearly a random search. Values over 0.3 have this clear effect. However, when we consider low values, i.e. 0.05, biodiversity and therefore convergence to the optimal solution are impaired.

The variation in the crossover probability is to some extent not as decisive as the mutation probability because the results, once the latter is fixed, are quite stable both in time and convergence for values within the range 0.9–0.6, being slightly (about 1%) better with 0.8 when mutation probability is 0.15. Let us remark that this study of the configuration of parameters is consistent with previous studies. In particular, the conclusions are essentially the same as those described in [30].

In conclusion, mutation and crossover probability are very complementary parameters that help us to stabilise biodiversity within the population over generations. For this reason, once we have found a tandem of values that balances good time performance and convergence to the optimal solution, it is time to increase the number of individuals in the population to increase the biodiversity of our sample.

4.1. Running examples

Having understood how our algorithm works, we analyse its performance with some examples. To do this, we will look at the key aspects that will tell us how good its performance has been when we try to find the best short-term strategy for the appeaser, setting some of the parameters to focus on those that will really make the difference. On the one hand, we will set, as we have indicated above, the number of selected candidates for the Selection process up to two. On the other hand, we are going to set now the crossover and mutation probabilities. By the law of large numbers, if in each run we are dealing with thousands of individuals (and therefore thousands of possible probabilistic events, in principle independent), then we can expect a normal distribution in the probability of mutation and crossover. Therefore, some predictability can be expected in the values that the probability of both events can take. Thus, we could define our model using the mean of that normal or, for example, the mean plus the standard deviation of both values. We came to the conclusion that values that work in most cases and still represent this biological concept could be $P_{crossing} = 0.8$ and $P_{muting} = 0.15$. Finally, we will set the number of individuals in each generation to 20 and in each instance, we will vary the number of generations for both the appeaser genetic algorithm and propagation, which will allow us to study the performance of the algorithms as a function of the number of total individuals generated. Since genetic algorithms have, as we have seen throughout this section, a large probabilistic component, it is usual that they do not always obtain the same results. For this reason, we will run each instance a certain number of times (in the following examples we run 50 times) and we will study the mean and standard deviation of the solutions, the percentage of successes of the optimal solution and the minimum and maximum fitness value of the solutions obtained by the algorithm.

Recreating real cases will help us determine the difficulties and challenges we actually face, for example, related to the large number of possible moves available to each player or how varying the parameters of the algorithm can worsen or improve the performance of our algorithm. It should be noted that although the concept of propagation can be directly related to forces of nature such as plagues or fires, it can have a wide variety of applications in other environments, such as preventing the spread of *fake news* or the leakage of sensitive information on the Internet and social networks, or an enemy army reaching the city's castle and managing to dethrone the king or *malware* completely deleting important information in a system. However, we will apply our algorithm, firstly, to a fire and, later, to a case study of notable importance at present, the *COVID-19* pandemic in our country, Spain.

For our first example, we set an easy scene in a forest where a fire will start at a single point, as it is an accidental fire. In this way, the spread will start in a tree located at one point and may spread to all the trees around it, and the fire extinguisher will represent the efforts made by the firemen to stop the advance of the fire. Each tree has a different (A, B), so we may be dealing with different species within the same forest, which would make their humidity or combustibility vary. As the fire can spread to any tree around the burnt tree, the genetic algorithm of the fire suppressor will be forced to find the optimal movement within the different possible combinations, otherwise, the whole forest would be in

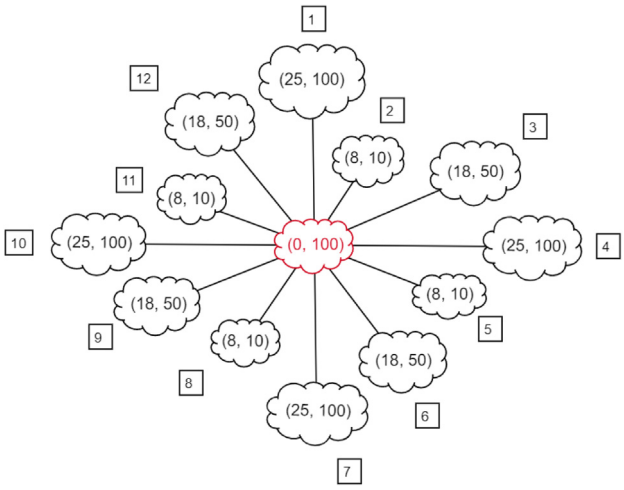

Fig. 6. Fire Example.

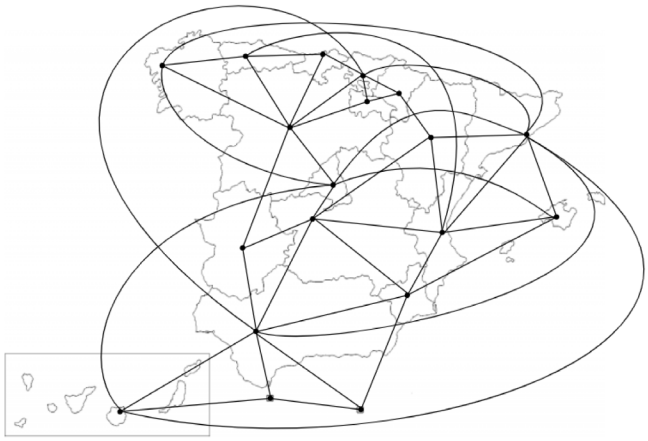

Fig. 7. Map of Spanish regions.

Table 1
Performance of the fire example.

Appeaser generations	Propagation generations	% Success	Average	Minimum	Maximum
50	50	70%	244.03	230	310
	100	74%	241.45	230	290
100	50	74%	240.25	230	310
	100	80%	239.91	230	280
200	50	86%	236.82	190	280
	100	88%	236.21	230	270
	200	90%	235.76	230	280

danger, as the combustibility of the other trees could be very high. In Fig. 6, we can see a representation of this problem.

In Table 1, we see the summary of the performance of our algorithm for this instance, where we vary the generations of the appeaser and propagation between 20, 50 and 100. In this case, the optimal move is the one that obtains a score of 230 and, as we can see in the table, when the number of generations of the appeaser is sufficient, increasing little by little the number of generations of the propagation makes the percentage of hits soar. Meanwhile, when the number of generations of the appeaser is insufficient, even if we increase the number of generations of the propagation, the hit rate does not improve. Therein lies the essence of our problem: the need to correctly adjust the parameters according to each problem.

Note that, despite the fact that the optimal solution has a heuristic value of 230, in the instance where the number of generations is 200 for the appeaser and 50 for the propagation, it has not been able to find the optimal propagation in at least one situation and has been left with a sub-optimal one. In this way, for a given cut of the appeaser, the propagation does not make its optimal movement, obtaining fewer points (190) than it can really obtain (230) and, for this reason, we obtain this data that is distorted with respect to the rest. This is the importance of sample biodiversity in population generation through propagation and appeaser generations. If we do not achieve high biodiversity through choosing the right game parameters, we will fail to explore as many moves as possible, missing out on moves that could be key to slowing down the spread.

Let us now look at a completely different and more complex case study, which will allow us to analyse other aspects of the implementation.

We begin the process of generating our new instance of the propagation game by establishing that the propagation graph will be composed of nodes which, in this case, will represent the capitals of the regions of Spain, while the adjacency matrix will connect two capitals for different reasons. On the one hand, if the territories of their regions are adjacent or, in the case of islands and autonomous cities, belong to those closest to them. On the other hand, we will add to our graph the edges that represent the most frequent air routes between regions. Thus, the map of the capitals of the Spanish regions adapted to the propagation game will be as shown in Fig. 7.

The propagation will clearly represent the advance of the pandemic through Spain and the appeaser could be the entire health team that is in charge of trying to slow down the advance of the pandemic, using certain methods that resemble the cutting of edges, such as the perimeter confinement of some areas or the mandatory quarantines of positive patients and people who have been in contact with them.

For each of the regions, we will choose the values of (a, b) according to the situation. It is important to note that these values must be carefully chosen, depending on certain variables, so that they are balanced, i.e. the values of a and b for each of the autonomous communities must be within the same range of values and there must not be large inequalities. Otherwise, it would not be possible to reflect the operation of propagation in the graph. For the case of a, we need a value that represents the capacity of a region to defend itself from the propagation attempt of the rest. The simplest thing we can think of that measures this criterion is the Health Budget that each one of them had assigned in 2020 [31], which measures the quantity and quality of resources that each one of them had to face the pandemic. On the other hand, to choose the value of b in each case, we will use one of the main causes of contagion that occurred in March 2020: displacements. To do this, we take the Percentages of Mobility to other regions, data extracted from the Study of mobility from mobile telephony carried out in March 2020 by the National Institute of Statistics [32] of each of them in that month, and we apply it to the total number of inhabitants to estimate a specific number of people who moved outside their region. Because the data related to b is targeted to a shorter time period, we will split the Annual Budgets data into 12 months to represent only the monthly budget. In addition, the final choice of a and b will be conditioned because, in order to favour the calculations of our algorithm, we decide not to use such large numbers, dividing the final data by 1000 and approximating them according to convenience. Below, we can see in Table 2 all these data reflected as well as the final choice of a and b for each region and the map of Spain in Fig. 8 in the form of a graph with the associated data.

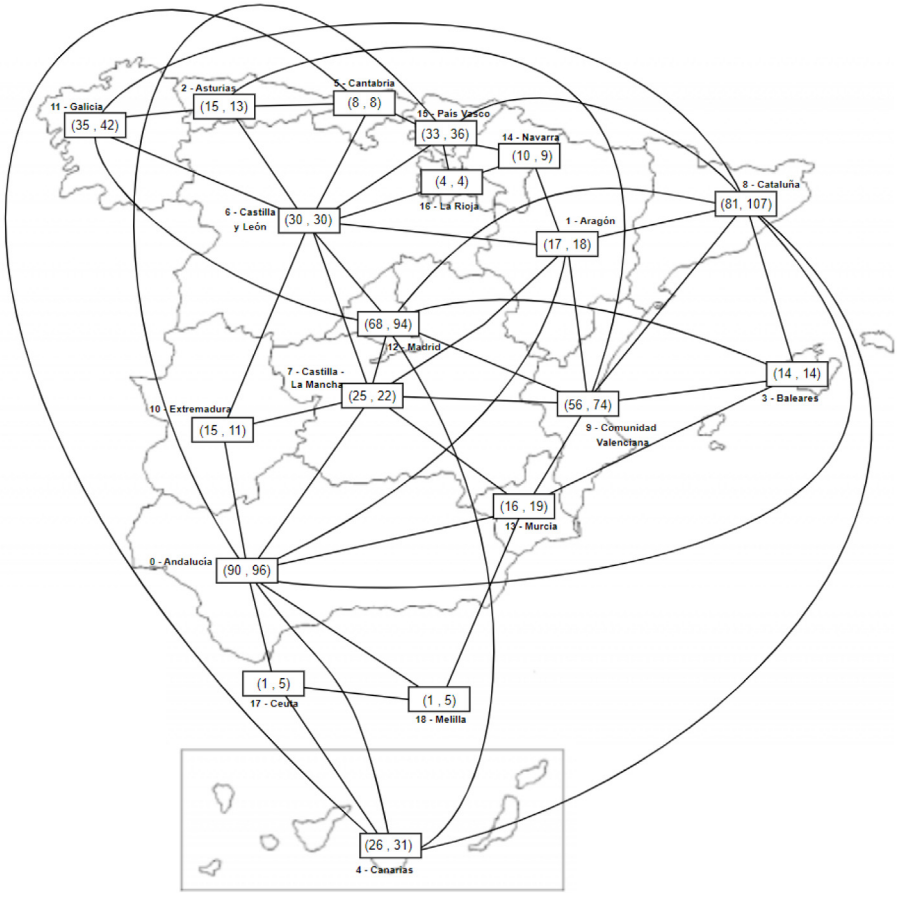

Fig. 8. Map of the regions of Spain.

Table 2
Data related to the instance.

Region	Number of inhabitants	Monthly health budget	% monthly mobility	Monthly mobility * inhabitants	*a*	*b*
Andalucía	8,464,411	902,043	11.32	958171.3252	90	96
Aragón	1,329,391	171,817	13.21	175612.5511	17	18
Asturias	1,018,784	151,300	13.07	133155.0688	15	13
Baleares	1,171,543	143,745	12.29	143982.6347	14	14
Canarias	2,175,952	261,082	14.18	308549.9936	26	31
Cantabria	582,905	76,838	14.53	84696.0965	8	8
Castilla y León	2,394,918	294,504	12.5	299364.75	30	30
Castilla-LaMancha	2,045,221	249,220	10.56	215975.3376	25	22
Cataluña	7,780,479	811,132	13.77	1071371.958	81	107
C.Valenciana	5,057,353	561,017	14.58	737362.0674	56	74
Extremadura	1,063,987	145,210	10.08	107249.8896	15	11
Galicia	2,701,819	342,277	15.78	426347.0382	35	42
Madrid	6,779,888	680,499	13.87	940370.4656	68	94
Murcia	1,511,251	160,213	12.36	186790.6236	16	19
Navarra	661,197	96,547	13.73	90782.3481	10	9
País Vasco	2,220,504	331,537	16.37	363496.5048	33	36
La Rioja	319,914	38,304	12.11	38741.5854	4	4
Ceuta	84,202	631	12.32	10373.6864	1	5
Melilla	87,076	98	12.49	10875.7924	1	5

Note that we choose for Ceuta and Melilla the value of 5, when it should be 1, in order to alleviate the difference between regions and autonomous cities and give them the opportunity to contribute to the graph. Moreover, it is necessary to stress that although it is reasonable that the budget in Health increases the value of a and the increase in mobility increases b, we do not know exactly the relationship between them, i.e. how many euros are needed to counterbalance one more person moving between regions. We have therefore chosen the values that appear in the table with these proportions to give rise to an instance that would not be trivial for either player and would allow us to reflect, to a certain extent, reality.

As for the rest of the game parameters, some of them will be determined according to data extracted from the National Institute of Statistics (i.e. k) and others will depend on the previous study of the algorithm and its performance (i.e. the probability of mutation). In the case of the number of cuts that can be made by the k appeaser, as we have explained above, it represents the

Table 3
Performance in our case study.

Where the prop starts	Appeaser generations	Propagation generations	% Success	Optimal	Minim
9 - C. Valenciana, 12 - Madrid, 15 - País Vasco	500	1000	45%	118	82
		2000	65%		88
		3000	73%		102
4- Canarias, 9 - C. Valenciana, 12 - Madrid	200	1000	84%	80	75
		2000	87%		80
	500	1000	88%	80	67
		2000	100%		80
0 - Andalucía, 1 - Aragón, 11 - Galicia	200	1000	90%	62	48
		2000	95%		62
	500	1000	94%	62	57
		2000	100%		62

cuts that can be made by the health workers. A single cut could seem useless in a graph with so many edges, but on the contrary, setting k too high would mean isolating many communications between regions and could lead to the anger of the population due to the high restrictions. Therefore, and taking into account how difficult it is to coordinate health issues between regions, a fair and mutually beneficial k could be 10% of the total number of edges, i.e. $k = 5$. When it came to choosing the region in which the propagation began, we opted, in the first place, for the places where the first cases really appeared in Spain (Canarias, Comunidad Valenciana and Madrid) and, in other cases, for the places where COVID has had the greatest incidence or those strategic places that interested us to study the performance of the algorithm (for example, Andalucía or País Vasco).

Regarding the parameters of the genetic algorithms, we will continue using the same values chosen in our previous example for the population size (10) and the probability of crossover ($P_{crossover} = 0.8$) and mutation ($P_{mute} = 0.15$) while, as we did before, we will vary the number of generations depending on the needs of each instance and, as an innovation, the places where the propagation starts. Again, we will run each instance 50 times to be able to draw certain conclusions such as the percentage of successes or the range in which the solutions move.

From the summary of the performance shown in Table 3 we can draw similar conclusions to those we discussed in the previous example, the necessity of adjusting the parameters to ensure enough biodiversity in the population. On the one hand, we can clearly see how, in some occasions, the number of generations of the genetic algorithm of the appeaser is not enough, causing a low percentage of success, as, for example, when the propagation begins in Canarias, C. Valenciana and Madrid, with 200 generations only an 87% success rate is achieved, while with 500, a 100% success rate is achieved. On the other hand, when the number of generations for the propagation is not enough, the phenomenon we talked about in the previous example occurs. This leads to a situation in which the propagation is not able to find the optimal solution and returns a heuristic value lower than the required one. Thus, it manages to fool the appeaser into believing that it can find a better movement than the one that is actually optimal, when in fact it is not possible. Despite this, this event does not occur frequently, having appeared only a couple of times out of the 50 iterations of each instance. This situation can be observed in several instances of the table, for instance, when propagation starts in C. Valenciana, Madrid and País Vasco, it occurs in all cases.

We re-emphasise the importance of finding the right parameter settings that allow us to ensure biodiversity within the sample. For this reason, we will try to increase the number of individuals within a population for those instances in which the results of our algorithm have not been as optimal as we would like. By increasing this parameter to 25, we obtain the performance shown in Table 4.

Table 4
Performance for the case study with higher biodiversity.

Where the prop starts	Appeaser generations	Propagation generations	% Success	Optimal	Minim
9 - C. Valenciana, 12 - Madrid, 15 - País Vasco	200	1000	96%	107	107
4- Canarias, 9 - C. Valenciana, 12 - Madrid	200	1000	91%	80	80
0 - Andalucía, 1 - Aragón, 11 - Galicia	200	1000	100%	62	62

As we can see in Table 4, instances that previously did not achieve a high percentage of successes manage to improve considerably, supporting the conclusions we discussed earlier.

The data obtained and the conclusions we can draw from them show, even without fully reflecting all the variables that influence a real propagation, that by seeking the optimal strategies and carrying them out with agility at the appropriate time, it is possible to minimise the damage that can be caused by a propagation. If we were to transfer this case study from the large scale of the regions to smaller scales such as family members, friends, or the population of a town, we would have a tool capable of determining the best strategy depending on certain variables to stop the advance of COVID-19 in our lives. As we have been saying throughout this paper, in reality, countless variables from biological to social are involved in making precise and optimal decisions, so any model used in a real situation should take into account many more factors in order to determine the optimal immediate strategy to minimise the damage caused by the spread.

4.2. Comparison with other methods

Note that the perfect solution of a Σ_2^P-complete problem can be achieved by exhaustively searching for all possible solutions of the problem. However, each possible candidate solution generates a secondary optimisation problem that has to be solved to evaluate the quality of that candidate solution. That is, the primary optimisation requires solving a new optimisation for each candidate solution of that primary level. This bi-level exhaustive optimisation could be seen as a minimax algorithm where, in particular, only a two-level tree must be explored.

The *minimax* is a recursive algorithm used mainly to try to find out the optimal move for our player assuming that the other player will also be playing optimally. The minimax algorithm is called so it helps you choose the optimal move to minimise your loss, while the other player wants to maximise it. Each of the game configurations, and therefore each of the iterations performed by the minimax algorithm, has a heuristic value associated with it, which determines how good the game state is, in our case, the sum of infection achieved by reaching those new nodes (as we did in our genetic algorithms). As we said previously, we will only need to explore the first two levels of that recursion. That is, the algorithm will explore all the possible movements of the appeaser and, for each one of them, all the possible movements that the propagation could make.

However, it is worth mentioning that each node in this particular exploration tree has an exponential number of child nodes. Minimax algorithms are suitable when the big size of the exploration tree is due to its depth (in particular, due to the exponential growth of the number of nodes in relation to the depth) and this depth can be cut to some extent to provide approximate

Table 5
Algorithms behaviour in the COVID Example.

COVID Instance	Algorithm	Execution time	Solution
9 - C. Valenciana,	Greedy	10 s	137
12 - Madrid,	Minimax	>12 h	–
15 - País Vasco	Genetic	5 min 30 s	107
4- Canarias,	Greedy	10 s	124
9 - C. Valenciana,	Minimax	>12 h	–
12 - Madrid	Genetic	3 min 30 s	80
0 - Andalucía,	Greedy	10 s	160
1 - Aragón,	Minimax	>12 h	–
11 - Galicia	Genetic	4 min 28 s	62

rather than perfect solutions. Nevertheless, when the problem comes from the exponential number of child nodes, we need algorithms which are good at exploring in width rather than in-depth, such as genetic algorithms, which are designed precisely to traverse only a tiny proportion of these exponential-size spaces.

Of course, the branching in a minimax algorithm can be reduced by analysing all available moves, next assigning each of them their heuristic value according to how promising it looks like for the current player, and finally developing the minimax tree only through those moves which look more promising according to these heuristic values. However, this approach is not feasible when the number of available (immediate) moves to be considered and heuristically assessed is exponential. Alternatively, if we give up this exhaustive analysis of choices and just pick some moves according to some a priori suitability, then we are rather considering a greedy algorithm.

A *greedy* algorithm is an approach for solving a problem by selecting the best option available at the moment. In many cases, a greedy approach does not result in an optimal solution, but a greedy heuristic can quickly create locally optimal solutions that come close to an ideal global solution. One way in which a greedy algorithm and our problem could fit together could be the following: given an instance of our propagation set, we could sort those nodes that are in direct danger of being infected according to some value, for example, the division between their infectious load and the amount of propagation necessary to be infected (b/a). This ratio makes sense, since it is higher the more gain the propagator obtains with less effort. Once sorted, it is time to choose which edges should be cut. To do this, we will start choosing those that lead to the most beneficial nodes for the propagation, eliminating the possibility of the propagation reaching them in its turn and minimising the damage it will do in the future. In fact, we have implemented both the aforementioned greedy algorithm and the minimax method.

Let us take a closer look at a brief comparison of the performance of these three algorithms: a greedy algorithm using some heuristic to produce promising moves, an exhaustive minimax algorithm fully exploring all possibilities in its first two levels and our bi-level genetic algorithm. In Table 5, these algorithms will face the three instances we have considered in the COVID propagation (see Table 3), a scenario that will help us see each other's strengths and weaknesses.

Taking a look at the table, we can draw quick conclusions about the performance of each of the algorithms in these instances. It is worth noting that the three instances show the same behaviours in all algorithms. As we described before, the exhaustive minimax approach takes too long times to be useful even for this moderate-size instance (up to the extent of having to be manually stopped in all cases due to reaching reasonable timeout limits, more than 12 h of execution). On the other hand, the greedy algorithm is able to find solutions very quickly. However, the solutions' quality is significantly worse than that provided by our bi-level genetic algorithm (160 vs. 62 in the more complex case). All in all, the bi-level genetic algorithm is clearly superior to the other two methods, since it combines reasonably short execution times (although worse than those of the greedy algorithm) with a high quality in the solutions found.

4.3. Long-term game

Pushing our experiment to the limit, we can even carry out a complete game in which the propagation and the appeaser face each other, turn by turn until no more movements can make any effect. As for the propagation, its objective is to maximise the amount of infection it holds by invading the nodes, in this case, Regions that are most beneficial for it. As for the appeaser, its objective is quite the opposite, that is, to minimise the damage that can be caused by the propagation, trying to save as many nodes as possible among those considered "more beneficial". The end of the game may be due to the fact that the propagation has reached all nodes, or that the appeaser has managed to stop the propagation, which can no longer infect any more nodes.

Keeping this in mind, we will use our bi-level genetic algorithm on the appeaser's turn to find its best possible move. Then, the move of the propagation will be decided by using its own genetic algorithm maximising its own fitness function. After that, it will be again the turn of the appeaser, and we will use again the bi-level genetic algorithm. We will iterate the process until the end of the game.

Following the same scheme as in the previous subsection, we will compare the results obtained by our bi-level algorithm with the results obtained by the greedy algorithm that we have already presented. In this sense, we will play again the complete game turn by turn, but using the greedy algorithm (instead of the bi-level genetic algorithm) in the steps corresponding to the appeaser.

The concrete case study we have considered is the last instance we have studied in the previous sections. In this case, it is worth noting that after playing the game to its last moves, the greedy algorithm was only able to save 4 regions (minimising the infection up to 427). On the other hand, when we used our bi-level genetic algorithm, the final result was that we were able to save 8 regions minimising the propagation damage to 222, a great improvement over the greedy result. To put these results in perspective, a complete game in which the appeaser failed to stop the propagation and the latter succeeded in conquering the entire map of Spain, the total sum of the propagation elements would be 638. Thus, the bi-level genetic algorithm obtains clearly better results than the greedy algorithm, not only in a single step view, but also when we consider a complete game with several iterative steps.

5. Conclusions and future work

We have formally specified a general framework to deal with any kind of propagation. Our formal definitions allowed us to identify the computational complexity of the propagation problem, showing that it is Σ_2^P-`complete`. This computational complexity suggests that a good method to deal with the problem in practical terms is to implement a two-level genetic algorithm. The observed experimental results support that idea, and suggest that a similar scheme could also be used to deal with other Σ_2^P-`complete` problems.

In order to analyse the quality of the solutions found by the bi-level genetic algorithm, we have implemented two other solutions: a minimax and a greedy algorithm. We observed that the minimax requires excessive time to find solutions, whereas the results of the greedy algorithm are clearly worse than those of our method. Moreover, we have analysed not only what happens

when we consider a single step of our problem, but also when we tackle a complete game where the appeaser and the propagation take turns. All in all, our experiments suggest that our method obtains good solutions both in the short and in the long term.

Based on our implementation, we can see how easily we could generalise both our problem and its implementation to deal with other interesting cases. A generalisation discussed earlier in a previous section considered using *weighting edges* so that the appeaser would have a certain number of points to spend on cutting edges, rather than always cutting k edges. Translating this generalisation to our algorithm involves a change in our representation of the edges of the graph, accompanying the adjacency matrix with a dictionary relating each edge to its value, e. In addition, k would come to denote the amount of weight that the appeaser is able to cut, instead of the number of edges to be cut. Accordingly, the fitness function of the appeaser would check if these cuts are feasible — which just consists in adding the weight associated to each edge belonging to the chromosome. From the theoretical point of view, note that the complexity hardness always propagates via problem generalisation, so the resulting new problem would also be Σ_2^P-hard (note that the particular case where all weights are 1 is exactly the problem studied in this paper). On the other hand, the inclusion of the new problem in Σ_2^P could be proved similarly as the inclusion of the problem studied in this paper, thus yielding also Σ_2^P-completeness.

Also, we would like to study other combinations of metaheuristics applied to the upper-level optimisation and the lower-level one, in such a way that they are not necessarily the same. This way, each optimisation level could be more easily customised to its different needs.

Regarding ongoing work, we are currently dealing with the analysis and implementation of propagations in the long term. We are studying the problem of finding an *adaptive* strategy defining the successive reactions of the appeaser to all possible actions of the propagation in any case during their whole interaction, in such a way that some number of nodes is kept away from the propagation in *any* case. Our preliminary studies suggest that the problem is PSPACE-complete, and that minimax-like strategies could be used to deal with it.

CRediT authorship contribution statement

Javier Galiana: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing, Visualization. **Ismael Rodríguez:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing, Supervision. **Fernando Rubio:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing, Supervision, Project administration, Funding adquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

The authors would like to thank the anonymous reviewers for valuable suggestions on the previous version of this paper.

Appendix A. Complexity analysis

Throughout this section we will analyse the complexity of the Propagation Problem introduced in the previous section, proving that SSTS is a Σ_2^P-complete problem through a polynomial reduction to the $QSAT_2$ problem.

Theorem 2. SSTS $\in \Sigma_2^P$-complete.

Proof. First, we will prove that the problem belongs to Σ_2^P. Note that our problem is equivalent to proving that there exists a set of edges that the appeaser can cut guaranteeing that, for all possible propagation moves, it is satisfied that the sum of the load capacities of the nodes it reaches is less than l. Therefore, we can define our problem as finding something of polynomial size such that, for something of polynomial size, a property that can be checked using polynomial size is satisfied. Consequently, SSTS belongs to Σ_2^P.

When it comes to proving Σ_2^P-hardness, we will construct a polynomial reduction from the well-known $QSAT_2$ problem [27]. This problem consists in, given an expression $\exists\ x_1, \ldots, x_n\ \forall\ y_1, \ldots, y_m\ \varphi$ where φ is a propositional formula denoted in DNF (Disjunctive Normal Form) depending only on the propositional variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, check whether the expression holds.

Let us denote as x' and y' the abbreviations of $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_m)$, respectively. Then we have that $\exists\ x'\ \forall\ y'\ \varphi \equiv \exists\ x'\ \neg\neg\forall\ y'\ \varphi \equiv \exists\ x'\ \neg\exists\ y'\ \neg\ \varphi \equiv \exists\, x'\ \neg\exists\, y'\ \varphi'$ where $\varphi' \equiv \neg\,\varphi$ is given in CNF (Conjunctive Normal Form). In the following we will only consider the last expression $\exists\, x'\ \neg\exists\, y'\ \varphi'$, where $\varphi' = \alpha_1 \wedge \cdots \wedge \alpha_p$ and, for all $1 \leq i \leq p$, we have $\alpha_i = \beta_1^i \vee \cdots \vee \beta_{l_i}^i$ where, for all $1 \leq j \leq l_i$, β_j^i are literals over propositional variables.

Given this instance of $QSAT_2$, we will create an instance of SSTS from it, such that there exists an x' that makes it impossible for the formula φ' to be satisfied for all y' if and only if in the instance of SSTS the propagation cannot reach the proposed total load limit l after the appeaser cuts k edges in a certain way. The propagation will succeed in reaching that bound if and only if the appeaser fails to find a set of edges to cut that prevent the propagation from advancing to that value. In our reduction, each possible value, $\top$ or $\bot$, of each of the variables will represent a node in the propagation graph. Thus, if a node is reached by the propagation, the variable associated to it will be assigned the value corresponding to that node, being the appeaser's responsibility to avoid assigning two different values to the same variable, in order to avoid contradictions that will imply the defeat of the appeaser. The propagation will be in charge of setting the values of the variables associated to the universal quantifiers, while the appeaser will control the existential variables. We will set in our instance $k = n$ being n the number of existential variables of our $QSAT_2$ problem and, therefore, the appeaser will be able to cut n edges in its turn. By cutting the edges leading to the nodes representing the values of x_i that the appeaser wants to save, it will force the propagation to advance, if possible, towards the remaining x_i values. In addition, the propagation will be free to choose among all possible values ($\top$ or $\bot$) of each y_j in its turn. For the appeaser to achieve its goal and succeed, it will have to make the propagation load addition of the nodes reached by the propagation on its next turn be less than a bound l, namely, $l = T + n*(6 \times B^{B-1} + 2 \times B^B)$, as we will see later. For the problem instance SSTS that we will construct from $QSAT_2$, we will see that there exists a successful short-term strategy for the appeaser if and only if there is some way to set the x_i variables guaranteeing that the formula φ is satisfied (equivalently, that the formula φ' is not).

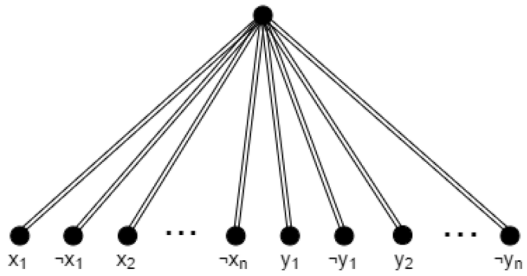

Fig. A.9. Initial state.

The process of assigning the variables of the problem is implicitly represented by the graph itself that we construct for the instance of SSTS (Fig. A.9), where a node appearing as "root" initiates the propagation and the possible values ($\top, \bot$) of the variables x_i and y_j constitute the war zone between the appeaser and the propagation, represented as x_i ($x_i = \top$), $\overline{x_i}$ ($x_i = \bot$), y_j ($y_j = \top$) and $\overline{y_j}$ ($y_j = \bot$), joined by *multiple edges* to the root.

Let us assume we have the instance $\exists\, x_1x_2\ \neg\exists\, y_1y_2\ \varphi'$ where $\varphi' = (x_1 \vee \overline{x_2} \vee y_1 \vee \overline{y_2}) \wedge (\overline{x_1} \vee x_2)$. Satisfying φ' involves assigning values (0 or 1) to each of the variables involved. Therefore, solving the following system of equations, by assigning the variables involved in φ' the values 0 or 1, is equivalent to finding the correct combination of variables that will make the formula φ' hold. While the first two equations represent the two clauses of the φ' formula (we will see the roles of variables d_{ij} later), the rest denote the restrictions that the same variable cannot be set to true and false at the same time.

$$\begin{cases} x_1 + \overline{x_2} + y_1 + \overline{y_2} + d_{11} + d_{12} + d_{13} = 4 \\ \overline{x_1} + x_2 + d_{21} = 2 \\ x_1 + \overline{x_1} = 1 \\ x_2 + \overline{x_2} = 1 \\ y_1 + \overline{y_1} = 1 \\ y_2 + \overline{y_2} = 1 \end{cases} \tag{A.1}$$

As we know, the formula φ' is a conjunction of clauses which in turn are disjunctions of literals. Therefore, for the formula to be satisfied, all of its clauses must be satisfied, and for that, some literal of each clause must be satisfied. If we look at the two equations that represent the two clauses of our formula φ', we must make sure that at least some of the four literals are satisfied in the first equation and some of the two in the second. We use the auxiliary variables d_{ij} to help us achieve that goal, whose values we will choose at convenience in order to allow us to choose all possible options among the values of the rest of the variables appearing in the equation. For example, in the first equation we introduce three auxiliary variables d_{11}, d_{12}, d_{13} that will take at the same time the value 1 as long as there is only one variable of the rest of the equation that takes the value 1, allowing the equation and, consequently, the clause to be fulfilled. If on the other hand $x_1, \overline{x_2}, y_1, \overline{y_2}$ were all to take the value 1, all the auxiliary variables would be forced to take the value 0 for the equation to be satisfied. Intermediate cases would lead to giving the value 1 to only some of these variables.

Next, we will be interested in constructing a single equation from the problem, which will allow us to relate it directly to our SSTS problem. The procedure we will follow in the following lines is an abstraction of another complexity proof, the proof that the famous *Sum of Subsets* problem [33] is NP-complete [21], based on reducing 3 − SAT to that problem. To perform the transformation of the system of equations to a single equation that represents all the information of the system and, at the same time, the importance of each of the variables, we will assign to each of the equations of the system a power of some base (in our next running example, the decimal system) in increasing order from the bottom. The last equation would represent the units, the penultimate the tens and so on until reaching, in this case, the hundreds of thousands. (See Eq. (A.2) in Box I.)

In this way, we associate a weight to each of the variables according to the equations in which they appear. This weight will generally be established by $\sum_{i\in I} 10^i$, where I represents the set of equation numbers where the variable appears. The same applies to the independent terms of each of the equations in the system. For example, in the case of $\overline{x_2}$, as it appears in the first and fourth equations representing the hundreds of thousands and hundreds, respectively, its associated weight will be 100100. The equation for our example is the following:

$$\begin{aligned} &101000x_1 + 11000\overline{x_1} + 10100x_2 + 100100\overline{x_2} + 100010y_1 \\ &+ 10\overline{y_1} + y_2 + 100001\overline{y_2} + 100000d_{11} + 100000d_{12} \\ &+ 100000d_{13} + 10000d_{21} = 421111 \end{aligned} \tag{A.3}$$

Note that, depending on the problem instance being codified, in principle it could happen that there is carry-over when adding the occurrences in the equations of a particular variable, and this is undesirable because then each equation in (A.2) would interfere with the others. If there can only be three literals per clause (as in 3 − SAT), then this would not occur if we use the decimal base as before, since no equation will have more than five terms (with value 0 or 1) to the left of the equal. However, if the number of literals per clause is not limited (as in $QSAT_2$), this is possible. This is not really a problem, because depending on each instance of our problem we can choose the concrete base (instead of the decimal) that will avoid this situation, avoiding the overlapping of rails that would lead us to a bad representation of our problem. In this way, the base should be chosen according to the minimum number we need to cover the possible carry, that will depend, as we can see in Eq. (A.1), on the double of the maximum number of literals that appear in a clause, being in our example 4 (variables) + 3 (auxiliary variables) + 1 (of margin) = 2*4.[3] Thus, we could choose octal as the base in that case, avoiding any kind of carry, although decimal has been chosen for the reader's convenience.

Thus, we have constructed an equation in which there will exist a way to give value to the variables and it will have a solution if and only if the formula φ' holds. We modify the graph in Fig. A.9 to accommodate the auxiliary variables d_{ij}. In addition, we must differentiate between which variables the appeaser controls (existential) and which the propagation controls (universal). To do this, we will use the structure shown in Fig. A.10, which will relate the possible values of each of the existential variables and will allow us to force the appeaser not to simultaneously set the same variable to true and false.

Note that the (a, b) values of each of the nodes representing the variables appearing in Eq. (A.3) (the nodes on the left of Fig. A.10) are those appearing in Table A.6. These values a and b are the same for each node, since the value the propagation will need to reach in that area, by adding up the b values of each new infected node in the area, will be 421111 — the same amount the propagation will be able to spread among those nodes. The vast majority of that amount will come from the top left node (421111−2). Therefore, the left zone of Fig. A.10 encodes Eq. (A.3) in such a way that it will be satisfied if and only if a beneficial infection points allocation can be chosen by the propagation.

Let us now see how the auxiliary construction on the right of Fig. A.10, created to handle the evaluation of the existential

[3] As we will see later, the base might need to be raised a little beyond that threshold in order to keep the consistency with other gadgets needed in the construction.

$$\begin{cases} x_1 + \overline{x_2} + y_1 + \overline{y_2} + d_{11} + d_{12} + d_{13} = 4 \rightarrow Eq5 \text{ (}\textbf{Hundredofthousands } 10^5\text{)} \\ \overline{x_1} + x_2 + d_{21} = 2 \rightarrow Eq\ 4 \text{ (}\textbf{Tensofthousands } 10^4\text{)} \\ x_1 + \overline{x_1} = 1 \rightarrow Eq\ 3 \text{ (}\textbf{Thousands } 10^3\text{)} \\ x_2 + \overline{x_2} = 1 \rightarrow Eq\ 2 \text{ (}\textbf{Hundreds } 10^2\text{)} \\ y_1 + \overline{y_1} = 1 \rightarrow Eq\ 1 \text{ (}\textbf{Tens } 10^1\text{)} \\ y_2 + \overline{y_2} = 1 \rightarrow Eq\ 0 \text{ (}\textbf{Units } 10^0\text{)} \end{cases} \tag{A.2}$$

Box I.

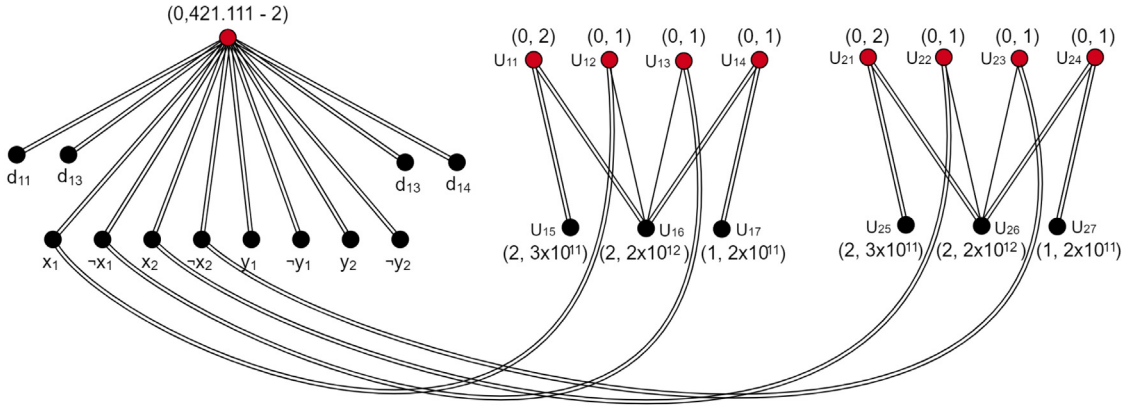

Fig. A.10. Initial state for our instance.

Table A.6
Values of the nodes representing each variable in our equation.

	a	b
x_1	101000	101000
$\overline{x_1}$	11000	11000
x_2	10100	10100
$\overline{x_2}$	100100	100100
y_1	100010	100010
$\overline{y_1}$	10	10
y_2	1	1
$\overline{y_2}$	100001	100001
d_{11}	100000	100000
d_{12}	100000	100000
d_{13}	100000	100000
d_{21}	10000	10000

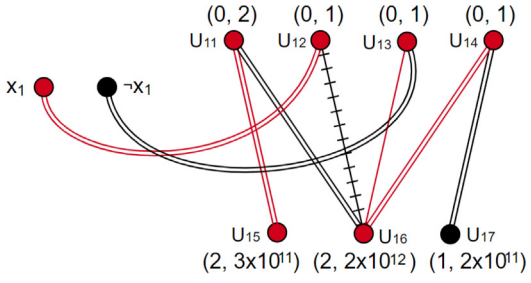

Fig. A.11. Propagation if the appeaser cuts x_1.

Table A.7
Truth table for φ' when $x_1 = \top$ y $x_2 = \bot$.

x_1	x_2	y_1	y_2	φ'
$\top$	$\bot$	$\bot$	$\bot$	$\bot$
		$\bot$	$\top$	$\bot$
		$\top$	$\bot$	$\bot$
		$\top$	$\top$	$\bot$

variable x_1, works. The number of cuts that the appeaser will be able to perform is given by the number of existential variables that exist in our instance, $k = 2$. We will focus on the construction for the variable x_1 since the case of x_2 is analogous. Looking at the charges of the nodes that already belong to the propagation ($U_{11}, U_{12}, U_{13}, U_{14}$), if the appeaser did not cut any of the two single edges, then the propagation would manage to reach the three nodes that are now free from the propagation (U_{15}, U_{16}, U_{17}), far exceeding the target it will want to reach. As we will see later, in that case it will not even need to infect any of the nodes on the left to achieve its goal. If, on the contrary, the appeaser decided to cut one of the two simple edges, for example the one from node U_{12} to node U_{16} (again, the case where the edge from node U_{13} to node U_{16} is cut is analogous), Fig. A.11 shows the sequence of moves by the propagation that would occur.

On the one hand, the propagation would advance from node U_{12} to x_1 (since it has no other available path) managing to bring to the left part of Fig. A.10 one of the points it needs to achieve its goal in that area (421111). Then, it is worth asking which other nodes are of more interest to the propagation to achieve the maximum possible goal. Clearly, the priority order for propagation is $U_{16} > U_{15} > U_{17}$, but it is not feasible to progress to all three because the loads of U_{11}, U_{13} and U_{14} are not sufficient. Then, to get the maximum possible (U_{16} and U_{15}), one must distribute the load of these nodes as we see in Fig. A.11. Alternatively, what would happen if the appeaser tried to cut *both* edges related to the same variable? Since it can cut as many edges as existential variables, this would imply leaving both edges of *another* variable uncut. If this situation were to happen, as shown in Fig. A.12, the problem would be completely biased towards the propagation, since it would be very easy to reach a much larger load than necessary with only the nodes $U_{16}, U_{17}, U_{25}, U_{26}$ and U_{27}. Actually, the additional points reached by the propagation from these nodes in this case would make

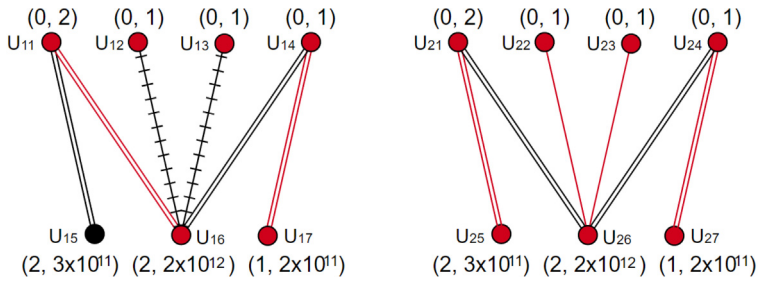

Fig. A.12. The appeaser tries to create a contradiction.

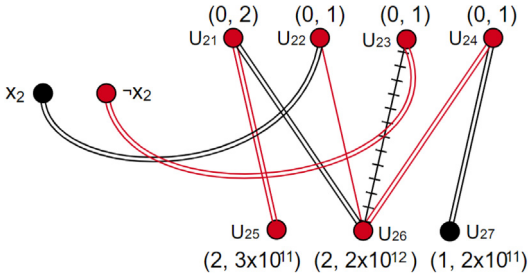

Fig. A.13. How the propagation act if the appeaser cuts $\overline{x_2}$.

unnecessary reaching also 421111 points from the left gadget, the one codifying Eq. (A.3).

By deciding which edge of the pair of edges involving each variable is cut, the appeaser will decide towards which literal of the existential variables (x_1 or $\overline{x_1}$) it will let go the free point coming from one of the nodes (U_{12} or U_{13}, respectively) so that an *unfavourable* variable assignment for the propagation is formed — that is, a situation that makes it impossible to meet φ' and thus achieve the target score 421111 on the left side of our construction. In the particular problem instance we are considering, it turns out that, by setting $x_1 = \top$ and $x_2 = \bot$, the equation cannot be fulfilled and, therefore, the appeaser would be victorious by performing the corresponding moves. First, the appeaser proceeds in the same way as we have done previously in Fig. A.11, thereby getting the propagation to reach x_1 and thus $x_1 = \top$. To get it to $x_2 = \bot$, it must follow a procedure analogous to the previous one, which we present in Fig. A.13.

From both structures (the one concerning x_1 in Fig. A.11 and the one involving x_2 in Fig. A.13), the propagation gets $2 * (3 \times 10^{11}) + 2 * (2 \times 10^{12})$ propagation charge points, which brings it closer to its goal.[4] It still needs to place the remaining 421111 points on the left side of Fig. A.10 since, as we will see, for this instance the propagation goal will be to reach $2 * (3 \times 10^{11}) + 2 * (2 \times 10^{12})$ plus 421111. To do so, it needs the two points that the appeaser has only left to shift to the variables (nodes) x_1 and $\overline{x_2}$ by construction. Consequently, the propagation is forced to spread towards these nodes (among others) in order to try to succeed. Recall that the top-left node can only distribute $421111 - 2$ load points among the bottom-left nodes, so those two additional points coming from the right-hand side will be necessary in any case. However, for this instance of the $QSAT_2$ problem, regardless of what values we assign to the variables y_1, y_2 the formula φ' will be unsatisfied, since its truth table when $x_1 = \top$ and $x_2 = \bot$ is reduced to what it is shown in Table A.7.

Note that the propagation cannot (for its own sake) sabotage the propagation by setting the y_j nodes to true and false at the same time. Since a and b are equal in all nodes representing the values of the variables, and the appeaser must let two additional points reach this area (as it is the only way to not allow the propagation reach its goal solely from the right-side hand gadgets), the propagation aims to get exactly 421111 points on the left-hand side. It cannot get less because, if so, then it would not meet its goal of reaching the l limit, and it cannot get more because it will not be able to deliver more than 421111 points in total from those nodes. Therefore, he has no choice but to try to fulfil exactly Eq. (A.3) (and, therefore, the system of Eqs. (A.2)) for the particular case where all nodes x_i and $\neg x_i$ are taken as allowed by the appeaser, as it is the only way to take advantage of the two additional points coming from the right-side gadgets. Fulfilling system of Eqs. (A.2) makes it impossible to reach (i.e. take) any two nodes y_i and $\neg y_i$ at the same time — in the same way as in the standard polynomial reduction from $3 - \texttt{SAT}$ to the Subset Sum problem.

The formula φ will be satisfied for a given assignment of values to the variables x_i if and only if the formula φ' will be unsatisfied for it. At the same time, this will occur if and only if, by cutting at most two edges, we manage to force the equation formed as a result of the development of the system of equations based on φ' to be unsatisfied, i.e., the sum of propagation loads (b) of the nodes reached by the propagation in the next turn does not exceed the limit which, in this case, would be $l = 421111 + 2 * (3 \times 10^{11}) + 2 * (2 \times 10^{12})$.

Next, we formally present the polynomial reduction from $QSAT_2$ to SSTS. Without loss of generality, let $\exists\, x_1, \ldots, x_n\ \forall\, y_1, \ldots, y_m\ \varphi$ be an instance of $QSAT_2$ where $\varphi = \alpha_1 \vee \cdots \vee \alpha_p$ and, for all $1 \leq i \leq p$, we have that $\alpha_i = \beta_1^i \wedge \cdots \wedge \beta_{l_i}^i$ where, for all $1 \leq j \leq l_i$, the β_j^i are literals over propositional variables.

- The set of vertices of the propagation graph, V, is composed of those nodes that are associated with the $\top, \bot$ values of the variables $x_1 \ldots x_n, y_1, \ldots, y_m$, the auxiliary nodes $d_{i,j}$, the node where the propagation starts and connects with the previous nodes, and the nodes required for the auxiliary structures of each of the existential variables.
- The set of edges of the propagation graph, A, is made up of the edges joining the vertices of V as we have seen in the previous development. It should be emphasised that, in its totality, the graph is composed of multi edges, with the exception of those single edges necessary for the choice of existential variables.
- The initial configuration of the game is c_0, consisting of the initial graph G with the form we have established in the previous explanations, the set P of nodes initially belonging to the propagation as explained, and the turn $i = 0$.
- When representing the formula φ' as equations we need, as we have seen, auxiliary variables $d_{i,j}$. To specify exactly how many are necessary we must understand that their existence serves to guarantee the possibility of choice among all the possible values of the literals in a clause (equation). For this reason exactly $l_i - 1$ are necessary in each equation, and in total, $D = \sum_{i=1}^{p} l_i - 1$.

[4] Note that, if alternatively the appeaser cut edges like in Fig. A.12, then the propagation would get *more* points, $2 * (2 \times 10^{11}) + 3 \times 10^{11} + 2 * (2 \times 10^{12})$.

- We will now define the foundation on which we will build those game parameters or auxiliary constructions that might seem arbitrary during the previous example. We will consider as base B a number that can avoid carry-overs in any case. On the one hand, we need to avoid any carry-over in the additions made in Eq. (A.3), as each power of the chosen base will be used to codify one of the equations in system (A.2), and these equations must not interfere with each other. For this purpose, we need B to be (at least) twice the maximum number of literals appearing in any of the clauses of φ (or, equivalently, φ'). On the other hand, we also need to avoid any carry-over in the additions made in the right-side gadgets, involving terms such as 2×10^{11}, 3×10^{11}, and 2×10^{12} for each existential variable x_i in our previous example (where the base was 10 for the sake of presentation simplicity). In this case, carry-overs can be avoided by using a base being, at least, $5n+1$. All in all, we conclude that the base B can be safely defined as follows:

  $$B = \max(2 * \max_i(l_i), 5n+1)$$

  Let z_i be any variable (existential or universal). The coefficient of that variable in Eq. (A.3), which in turn denotes the a and b values of the corresponding node in the left-hand side of the graph, is defined as follows:

  $$Coefficient(z_i) = \sum_{j\in EC} B^j,$$

  where EC is the set of indexes of equations where z_i appears, ordered according to the criteria seen in system (A.2). Similarly, in order to find the independent term T of the equation (421111 in our previous example) we must follow a similar process, in this case with the corresponding independent terms of the equations. In the initial node at the left-side, $a=0$ and $b=T-n$.
- In order to develop the auxiliary structures, which ensure unique values for the propositional variables controlled by the appeaser (the existential ones), we will need 7 nodes per existential variable. This means a total of $7n$ nodes U, whose values (a,b) are determined as follows. In those belonging to the propagation $(U_{i1}, U_{i2}, U_{i3}, U_{i4}, 1\le i\le n)$ the value a is zero and the value b will be determined in the same way as we saw in the example (2, 1, 1 and 1, respectively). For the nodes that are threatened by the propagation, $(U_{i5}, U_{i6}, U_{i7}, 1\le i\le n)$ their a values will be the ones seen during the examples, while the b values will depend directly on the base we are dealing with throughout the problem, since their weight with respect to the rest of the nodes is of great importance. Consequently, the b values will be

  $$U_{i5} = 3\times B^{|EC|+1}, \quad U_{i6} = 2\times B^{|EC|+2},$$
  $$U_{i7} = 2\times B^{|EC|+1}, \quad 1\le i\le n$$

  where we choose, as exponents of B, the number of equations plus one in the case of U_{i5} and U_{i7} (to guarantee their priority over the addition of all b values in the nodes in the left-side of the graph), and plus two in the case of U_{i6} (to guarantee its priority with respect to the previous ones).
- The total number of vertices, $|V|$, is the sum of
  - The initial node where the propagation starts at the left-side,
  - $2n$ nodes representing the $\top, \bot$ values of each of the existential variables x_i,
  - $2m$ nodes representing the $\top, \bot$ values of each of the existential variables y_j,
  - $D = \sum_{i=1}^{p} l_i - 1$ auxiliary nodes to represent φ' in the form of equations,
  - $7n$ nodes constituting the auxiliary constructions for each existential variable x_i.
- The remaining parameters of the problem instance are, as we have seen before, set as follows:
  - $l = T + n*(3\times B^{B-1} + 2\times B^B)$,
  - $k=n$, i.e., the appeaser can perform as many cuts as the number of existential variables.

It is easy to prove that the size of the SSTS instance is polynomial with respect to the size of the original $QSAT_2$ instance. Next, we will prove that the solution of the original problem $QSAT_2$ is *yes* if and only if the answer to the SSTS instance we have constructed is *yes*.

$\Longrightarrow$ Let us assume that $\exists\, x_1,\ldots,x_n\ \forall\, y_1,\ldots,y_m\ \varphi$ holds. That is, there exists $x_1,\ldots,x_n$ such that there exists no $y_1,\ldots,y_m$ such that φ' is satisfied, where $\varphi' = \neg\varphi$ is given in CNF. That means, if we view the problem $QSAT_2$ as a game where player 1 chooses the value of the existential variables and player 2 chooses the value of the universal ones, where the goal for player 1 is to complete the φ-clause, then player 1 has a winning strategy. Let us see that the direct translation of this strategy to the propagation game is a short-term strategy for the appeaser that guarantees that the propagation fails to reach the limit l. In the $QSAT_2$ game the value of the n existential variables are set by player 1 and the m universal variables by player 2, so evidently this is also the case when player 1 follows such a winning strategy. Note that if the appeaser were to make a move contrary to what is allowed in $QSAT_2$, it would result in the immediate victory of the propagation even without the need to set the values of all the propositional variables. This is because, in all such cases, the propagation would succeed in reaching at least one more of the U_{ij} nodes with high propagation load, trivially achieving the imposed l-limit. Thus, if for example the appeaser were to attempt to set to *true and false* one of the variables under its control, the propagation victory would be declared immediately afterwards, as we have seen in the previous construction. Suppose then that the appeaser does not make an unexpected move that causes it to lose trivially, and the propagation follows the rules of the game as established. We can then settle that, given the previously developed construction, if player 1 successfully establishes the value of the existential variables x_i, then the appeaser will cut the corresponding edges and the propagation will succeed in adding $3\times B^{B-1} + 2\times B^B$ points from the right zone, lacking only the T points to achieve its goal. Having made these cuts, the excess points must be diverted to the existential variables the appeaser is letting the propagation move into. Consequently, and since there is no $y_1,\ldots,y_m$ such that φ' is satisfied, we can be sure that the equation formed as a result of the development of the system of equations based on φ' will not be satisfied by any of the values of $y_1,\ldots,y_m$. Thus, there is no chance for the propagation to get over the point limit l and the victory will go to the appeaser.

$\Longleftarrow$ Let us now assume that the appeaser has a short-term strategy with which it gets the propagation damage to be less than l. Let us see that a strategy for $QSAT_2$ resulting from such short-term strategy is necessarily a winning strategy for player 1. A winning short-term strategy for the appeaser implies that, after cutting k edges in the graph and the propagation progresses, the propagation will necessarily fail to reach the target propagation load of l. Let us see that this

implies that there is a winning strategy of the game $QSAT_2$ for player 1.

If the appeaser has succeeded in defeating the propagation, it is because the latter has failed to achieve the objective l. Let us see how this is possible. If the appeaser has cut certain edges that will lead the propagation towards the desired values of the existential variables, then the propagation has achieved the points from the auxiliary structures, in total $n*(3\times B^{B-1}+2\times B^{B})$. Consequently, whether the propagation will succeed in reaching its goal or not will depend entirely on whether it manages to find a feasible solution to the equation formed from the development of the system of equations based on φ', which we know does not happen, because the appeaser has a successful immediate strategy. Thus, we can ensure that, if some combination of edges is cut, then it is impossible to propagate in such a way that the equation is satisfied. Therefore, the strategy for $QSAT_2$ resulting from the successful short-term strategy of SSTS (where only valid moves are carried out within the problem $QSAT_2$) is a winning strategy for player 1 in $QSAT_2$. □

References

[1] A. Dimitrakopoulos, C. Gogi, G. Stamatelos, I. Mitsopoulos, Statistical analysis of the fire environment of large forest fires (> 1000 ha) in Greece, Polish J. Environ. Stud. 20 (2) (2011) 327–332.

[2] A.A. Alemi, M. Bierbaum, C.R. Myers, J.P. Sethna, You can run, you can hide: The epidemiology and statistical mechanics of zombies, Phys. Rev. E 92 (5) (2015) 052801.

[3] G. Díaz, H. Macia, V. Valero, J. Boubeta-Puig, G. Ortiz, Facilitating the quantitative analysis of complex events through a computational intelligence model-driven tool, Sci. Program. (2019).

[4] Ó. Toledano, B. Mula, S.N. Santalla, J. Rodríguez-Laguna, Ó. Gálvez, Effects of confinement and vaccination on an epidemic outburst: A statistical mechanics approach, 2021, arXiv:2102.11160.

[5] J.H. Holland, Genetic algorithms, Sci. Am. 267 (1) (1992) 66–73.

[6] O. Kramer, Genetic algorithms, in: Genetic Algorithm Essentials, Springer, 2017, pp. 11–19.

[7] S. Katoch, S.S. Chauhan, V. Kumar, A review on genetic algorithm: Past, present, and future, Multimedia Tools Appl. 80 (5) (2021) 8091–8126.

[8] M. Dorigo, C. Blum, Ant colony optimization theory: A survey, Theoret. Comput. Sci. 344 (2–3) (2005) 243–278.

[9] J. Kennedy, Swarm intelligence, in: Handbook of Nature-Inspired and Innovative Computing, Springer, 2006, pp. 187–219.

[10] P. Rabanal, I. Rodríguez, F. Rubio, Applications of river formation dynamics, J. Comput. Sci. 22 (2017) 26–35.

[11] J.C. Bansal, P.K. Singh, N.R. Pal, Evolutionary and Swarm Intelligence Algorithms, vol. 779, Springer, 2019.

[12] K.A. De Jong, W.M. Spears, et al., Using genetic algorithms to solve NP-complete problems., in: ICGA, 1989, pp. 124–132.

[13] A. Salman, I. Ahmad, S. Al-Madani, Particle swarm optimization for task assignment problem, Microprocess. Microsyst. 26 (8) (2002) 363–371.

[14] H. Drias, S. Sadeg, S. Yahi, Cooperative bees swarm for solving the maximum weighted satisfiability problem, in: International Work-Conference on Artificial Neural Networks, IWANN, Springer, 2005, pp. 318–325.

[15] I. Rodríguez, F. Rubio, P. Rabanal, Automatic media planning: Optimal advertisement placement problems, in: 2016 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2016, pp. 5170–5177.

[16] Y.H. Kim, Y. Yoon, Z.W. Geem, A comparison study of harmony search and genetic algorithm for the max-cut problem, Swarm Evol. Comput. 44 (2019) 130–135.

[17] A. Muñoz, F. Rubio, Evaluating genetic algorithms through the approximability hierarchy, J. Comput. Sci. 53 (2021) 101388.

[18] M. Kumar, A. Sahu, P. Mitra, A comparison of different metaheuristics for the quadratic assignment problem in accelerated systems, Appl. Soft Comput. 100 (2021) 106927.

[19] D.E. Knuth, R.W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence 6 (4) (1975) 293–326.

[20] I. Rodríguez, P. Rabanal, F. Rubio, How to make a best-seller: Optimal product design problems, Appl. Soft Comput. 55 (2017) 178–196.

[21] M. Sipser, Introduction to the theory of computation, ACM Sigact News 27 (1) (1996) 27–29.

[22] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2009.

[23] T. Ward, Design archetypes from group processes, Des. Stud. 8 (3) (1987) 157–169.

[24] K. Kowalski, B. Lev, On step fixed-charge transportation problem, Omega 36 (5) (2008) 913–917.

[25] S. Schrenk, G. Finke, V.D. Cung, Two classical transportation problems revisited: Pure constant fixed charges and the paradox, Math. Comput. Modelling 54 (9–10) (2011) 2306–2315.

[26] B. Biesinger, B. Hu, G. Raidl, A hybrid genetic algorithm with solution archive for the discrete ($r|p$)-centroid problem, J. Heuristics 21 (3) (2015) 391–431.

[27] M. Schaefer, C. Umans, Completeness in the polynomial-time hierarchy: A compendium, SIGACT News 33 (3) (2002) 32–49.

[28] Y. Yukiko, A. Nobue, A diploid genetic algorithm for preserving population diversity—Pseudo-meiosis GA, in: International Conference on Parallel Problem Solving from Nature, Springer, 1994, pp. 36–45.

[29] H. Bhasin, G. Behal, N. Aggarwal, R.K. Saini, S. Choudhary, On the applicability of diploid genetic algorithms in dynamic environments, Soft Comput. 20 (9) (2016) 3403–3410.

[30] O. Boyabatli, I. Sabuncuoglu, Parameter selection in genetic algorithms, J. Syst. Cybern. Inform. 4 (2) (2004) 78.

[31] Expansión / Datosmacro.com., Presupuestos de las comunidades autónomas: Sanidad, 2020, https://datosmacro.expansion.com/estado/presupuestos/espana-comunidades-autonomas?sc=PR-G-F-31.

[32] Instituto Nacional de Estadística, Estudios de movilidad a partir de la telefonía móvil, 2020, https://www.ine.es/jaxiT3/Datos.htm?t=35167.

[33] Wikipedia contributors, Subset sum problem — Wikipedia, the free encyclopedia, 2021, https://en.wikipedia.org/w/index.php?title=Subset_sum_problem&oldid=1029155199. (Accessed 18 April 2022).

Javier Galiana obtained a bachelor degree in Mathematics and another bachelor degree in Computer Engineering from Complutense University of Madrid (Spain) in the year 2021. He is currently working as engineer in the industry. His research interests cover computational complexity and evolutionary algorithms.

Ismael Rodríguez is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2001 and his Ph.D. in the same subject in 2004. Dr. Rodríguez received the Best Thesis Award of his faculty in 2004. Dr. Rodríguez has published more than 100 papers in international refereed conferences and journals. His research interests cover formal testing techniques, swarm and evolutionary optimisation algorithms, computational complexity, formal methods, and functional programming.

Fernando Rubio is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 1997, and he was awarded by the Spanish Ministry of Education with "Primer Premio Nacional Fin de Carrera". He finished his Ph.D. in the same subject four years later. Dr. Rubio received the Best Thesis Award of his faculty in 2001. Dr. Rubio has published more than 100 papers in international refereed conferences and journals. His research interests cover formal methods, swarm and evolutionary optimisation methods, parallel computing, and functional programming.