# The Problems of Selecting Problems [⋆]

Alberto de la Encina, Natalia López, Ismael Rodríguez, Fernando Rubio,

Dpto. Sistemas Informáticos y Computación.
Facultad Informática. Universidad Complutense de Madrid.
28040 Madrid, Spain
{albertoe,natalia,isrodrig,fernando}@sip.ucm.es

**Abstract.** We face several teaching problems where a set of exercises has to be selected based on their capability to make students discover typical misconceptions or their capability to evaluate the knowledge of the students. We consider four different optimization problems, developed from two basic decision problems. The first two optimization problems consist in selecting a set of exercises reaching some required levels of coverage for each topic. In the first problem we minimize the total time required to present the selected exercises, whereas the surplus coverage of topics is maximized in the second problem. The other two optimization problems consist in composing an exam in such a way that each student misconception reduces the overall mark of the exam to some specific required extent. In particular, we consider the problem of minimizing the size of the exam fulfilling these mark reduction constraints, and the problem of minimizing the differences between the required marks losses due to each misconception and the actual ones in the composed exam. For each optimization problem, we formally identify its approximation hardness and we heuristically solve it by using a genetic algorithm. We report experimental results for a case study based on a set of real exercises of Discrete Mathematics, a Computer Science degree subject.

**Keywords:** Computational Complexity, Optimization, Education, Genetic Algorithms.

## 1 Introduction

Every time a new topic has to be taught to the students of any subject from any academic level, several examples are usually presented to help understanding the general description. However, selecting an appropriate set of examples is not a trivial task (see e.g. [9,10]). Notice that the teacher has to take into account the most common misconceptions the students may have. For instance, 6-year old students may think the number that is carried over in additions is always 1, and novice programming students may think the variable controlling a while loop is always incremented by one unit every iteration. Thus, in order to cover all possible misconceptions, a good set of examples

[⋆] This work has been partially supported by Spanish project TIN2015-67522-C3-3-R, the UCM project PIMCD 2016_176, and by Comunidad de Madrid as part of the programs S2013/ICE-2731 (NGREENS-CM), and S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union.

has to be selected: For each possible misconception, there should be at least one example that cannot be solved successfully in case the misconception is present. By doing so, students having any misconceptions will notice them, as examples will make any contradiction between their view and the correct view arise.

Given a set of examples covering misconceptions on a given topic, we tackle the teaching problem of finding a subset of examples that covers all the possible misconceptions a given number of times. For instance, let us assume we have 5 possible misconceptions (A, B, C, D, and E) and 5 possible examples where Example 1 covers errors A, B, C, and E; Example 2 covers C and D; Example 3 covers B and C; Example 4 covers A, B, and D; and Example 5 covers C, D, and E. Besides, suppose we want to cover each misconception at least *twice*. Then, it is not necessary to present all the examples (although they would obviously cover each misconception at least twice), it is enough to present examples 1, 4, and 5.

Going one step forward, we will also apply those notions to a related teaching activity. When we want to evaluate the knowledge of the students with an exam, we should be able to detect and assess the misconceptions of the students. Thus, the exam should include exercises whose elaboration is sensitive to the main misconceptions (i.e. exercises which will be failed if some misconceptions are present), and each misconception should minor the score of the exam appropriately, in some desired extent. Let us suppose each misconception is assigned a given desired grade penalty. Then, designing a suitable exam does not consist just in covering all misconceptions a minimum number of times (as we can do when teaching the topic in the classroom), as this would wrongly increase the relative penalty of all misconceptions which are covered by a proportion of exercises higher than desired. For instance, in the example presented before, in case an exam includes the five examples 1-5 (in this case, exam exercises), a student with a misconception of type C will fail 4 out of 5 exercises, while a student with a misconception of type A will only fail 2 out of 5 exercises. Given how many points should ideally be penalized by each misconception, the second teaching problem tackled in this paper will be selecting the appropriate set of exercises to cover each misconception the specified number of times, or to minimize the *distance* to such optimal distribution in case it is not possible to obtain the exact solution required.

These two teaching tasks motivate the definition of the four optimization problems faced in this paper. On one hand, for the task of selecting a set of exercises reaching some target coverage levels for each misconception, we will consider the problem of minimizing the time required to teach them and the problem of maximizing the surplus coverage of misconceptions beyond their minimum required levels. On the other hand, for the task of composing exams where each misconception yields its corresponding required mark reduction, we will consider the problem of minimizing the time required to complete the exam and the problem of minimizing the differences between the required mark losses due to the misconceptions and the actual losses in the composed exam. As we will see, all four optimization problems are `NP-hard`, meaning that no algorithm can find optimal solutions for them in reasonable time when the size of the problem is not trivial. Moreover, the *approximability* (see e.g. [4,18,1]) of these four problems is not good either, as all of them are `Log-APX-hard` (some even worse). Being `Log-APX-hard` means that it is not possible to find good approximations to their optimal solutions: If $\mathtt{P} \neq \mathtt{NP}$, then it is not possible to find polynomial-time algorithms guaranteeing that their solutions will be a given fix percentage worse than the optimal ones in the worst case.

In order to heuristically solve these problems, we will use genetic algorithms [15,6,23], as they have proven to be successful solving many different types of optimization problems (see e.g. [14,3,5,16,22,21,17]). Given a set of misconceptions, a set of examples (exercises), the time required to present (and solve) each example in the classroom, the list of misconceptions covered by each example, the maximum total time available to present all examples, and the number of times each misconception has to be presented by means of examples (or the expected mark loss each misconception should produce in an exam), the algorithms will face the aforementioned optimization problems.

In order to test our algorithms, we consider a concrete subject: Discrete Mathematics. This subject is taught to first-year students of the Computing Engineering Degree, and it covers many different concepts, including probability, graph theory, recursion, induction, number theory, etc. Although most of the concepts are independent, a single exercise usually combines several of them. The teachers of the subject have created a list of 261 exercises covering 55 possible misconceptions. For each exercise, we know the time required to present it, the time required to solve it in an exam, and the misconceptions covered by it. Given this information, our algorithms are used to both design exams and select a list of exercises to be presented in the classroom to introduce a given topic (or to summarize the main topics of the subject during the last lessons). Experimental results are reported.

The rest of the paper is structured as follows. In the next section we formally define the problems considered in our work. Then, in Section 3 we present algorithms solving the problems and we present the results obtained when using them to deal with the set of exercises of a concrete subject. Finally, in Section 4 we present our conclusions and lines for future work.

## 2 Formal Description of the Problems

In this section we formally present our problems. The first problem under consideration, that is, the problem of selecting the exercises to be taught in the classroom so that they cover all required misconceptions some numbers of times, is introduced next. The most basic version of the problem will be its *decision* version: given the coverage level of each misconception provided by each exercise (i.e. the *capability* of the solution of the exercise to make students with the corresponding misconception realize their view is wrong), the required levels of coverage of all misconceptions, and the maximum time to be spent presenting exercises in the classroom, it asks whether there exists a set of exercises meeting all these requirements. We also consider two associated optimization problems. In the first one, we select the set of exercises meeting the required minimum misconception coverage conditions which also *minimizes* the total time required to introduce the chosen set of exercises. This is the goal of a teacher interested in minimizing the time needed to present a set of problems covering all target errors, in order to maximize the productivity of the teaching time. Alternatively, it could be the case that we have a fix amount of time available to present exercises. In such situation, we do not need to minimize the total required time, but to select a set of problems fitting into the available time. In this case, we may be interested in taking profit of all the available time to cover the misconceptions as much as possible (as long as all misconceptions reach their corresponding minimum required coverage levels). The corresponding problem is the following: for all solutions fitting into the maximum time and reaching all minimum coverage levels for each error, we want to maximize the total coverage *surplus* (i.e. the excess of coverage over the required minimums) of all misconceptions. In this case, the teacher can define the relative contribution of the surplus coverage of each misconception, in such a way that exceeding the required coverage of some misconceptions is more valuable than exceeding the required coverage of others.

Hereafter we consider that, given a vector $v$, $v^T$ denotes the transpose of $v$.

**Definition 1.** Let us consider that we have $n$ questions or problems available $(P_1,\dots,P_n)$ and $m$ misconceptions to be covered $(M_1,\dots,M_m)$. Let $c \in \mathbf{N}^n$ be a vector of naturals where each value $c_i \in \mathbf{N}$ denotes the time needed to present and solve problem $P_i$, $A \in \mathbf{R}^{m \cdot n}$ be a matrix where each value $a_{ij} \in \mathbf{R}$ denotes the *coverage level* of the misconception $M_i$ provided by problem $P_j$, and $b \in \mathbf{R}^m$ be a vector where each value $b_j \in \mathbf{R}$ denotes the minimum coverage we want to reach for each misconception $M_j$. Finally, let $w \in \mathbf{R}^m$ be a vector where each $w_j \in \mathbf{R}$ denotes the contribution of the surplus coverage of misconception $M_j$. Then,

- Given $k \in \mathbf{N}$, the *Misconception Coverage* problem, denoted by `MC`, is the decision problem of finding out whether there exists $x \in \{0,1\}^n$ such that $c^T \cdot x \leq k$ and $A \cdot x \geq b$.
- The *Misconception Coverage Time Optimization* problem, denoted by `MCTO`, is the problem of minimizing $c^T \cdot x$ subject to $x \in \{0,1\}^n$ and $A \cdot x \geq b$.
- Given $k \in \mathbf{N}$, the *Misconception Coverage Surplus Optimization* problem, denoted by `MCSO`, is the problem of maximizing $\sum_i w_i \cdot ((\sum_j a_{ij} \cdot x_j) - b_i)$ subject to $c^T \cdot x \leq k$, $x \in \{0,1\}^n$, and $A \cdot x \geq b$.

□

We have the following properties.

**Theorem 1.** We have

(a) $\texttt{MC} \in \texttt{NP-complete}$.
(b) $\texttt{MCTO} \in \texttt{Log-APX-complete}$.
(c) $\texttt{MCSO} \in \texttt{NPO-complete}$.

*Proof.* Let us start proving (b). In order to do it, we have to provide a Log-APX-hardness preserving polynomial reduction from a `Log-APX-hard` problem into `MCTO`. We consider an S-reduction (see e.g. [4]) from *Minimum Set Cover*, `MSC` [8]. Given a collection $\mathscr{C}$ of sets $\mathscr{C} = \{S_1,\dots,S_k\}$ with $\bigcup_{S\in\mathscr{C}} S = \{e_1,\dots,e_p\}$, `MSC` consists in picking a subset $\mathscr{C}'$ of $\mathscr{C}$ such that $\bigcup_{S\in\mathscr{C}'} S = \{e_1,\dots,e_p\}$ in such a way that $|\mathscr{C}'|$ (i.e., the number of sets of $\mathscr{C}'$) is minimized. The construction of an S-reduction from `MSC` into `MCTO` can be done as follows. From a `MSC` instance $\mathscr{C}$, we define an `MCTO` instance in the same terms as in Definition 1 where

- $n = k$, $m = p$,
- $c = 1^n$, $b = 1^m$,
- For all $1 \leq i \leq m$ and $1 \leq j \leq n$,

$$a_{ij} = \begin{cases} 1 \texttt{ if } e_i \in S_j \\ 0 \texttt{ otherwise} \end{cases}$$

We can see that there exists a solution to this MCTO instance with cost $q \in \mathbf{N}$ iff there exists a solution to the original MSC instance with the same cost $q$. Note that each linear inequality $a_{i1} \cdot x_1 + \ldots + a_{in} \cdot x_m \geq 1$ imposed by the matrix constraint $A \cdot x \geq 1^m$ requires that at least one of the sets containing element $e_i$ is picked (in particular, picking $x_j = 1$ in MCTO represents picking the set $S_j$ in MSC). Since all costs in $c$ are set to 1, the cost of any solution for the MCTO instance is the number of variables $x_j$ which are set to 1, which equals the cost of picking the corresponding sets as a solution for the corresponding MSC instance. Therefore, the optimal solution for the MSC solution has cost $o \in \mathbf{N}$ iff the optimal solution for the MCTO solution has cost $o$. We conclude that there is an S-reduction from MSC into MCTO, which proves MCTO $\in$ Log-APX-hard.

Besides, it is easy to see that problem *CIP* [11] generalizes MCTO. Since *CIP* admits a logarithmic approximation, MCTO does too, so MCTO $\in$ Log-APX. We conclude that MCTO $\in$ Log-APX-complete.

In order to prove (a), we can trivially adapt the previous S-reduction to polynomially reduce *Set Cover* (the decision version of MSC) into MC, which implies the NP-hardness of MC. Since MC $\in$ NP (as we can check $A \cdot x \geq b$ and $c^T \cdot x \leq k$ in polynomial time), we conclude MC $\in$ NP-complete.

In order to prove (c), we S-reduce the NPO-complete problem *Max Ones* [12] (that is, the problem of satisfying all the clauses of a 3-CNF formula while maximizing the number of propositional symbols that are set to *True*) to MCSO as follows. For each disjunctive clause $C_i = y^i_1 \vee y^i_2 \vee y^i_3$ of the original Max Ones instance (where each $y^i_j$ is a literal, i.e. it is $p$ or $\neg p$ for some propositional symbol $p$), we create a misconception $M_i$ which represents the clause $C_i$ to be covered. Moreover, for each propositional symbol $p$, we also create two misconceptions $M_p$ and $M'_p$, whose coverage will represent that $p$ has been assigned some value (*True* or *False*) and that $p$ has been assigned a *True* value in particular, respectively. This completes the set of misconceptions, and problems (questions) are defined as follows. For each propositional symbol $p$, we create two problems $PT_p$ and $PF_p$ representing that symbol $p$ is set to *True* or *False*, respectively. Each problem $PT_p$ covers misconceptions $M_p$, $M'_p$, and all misconceptions $M_i$ such that some literal $y^i_1$, $y^i_2$ or $y^i_3$ is of the form $p$, whereas problem $PF_p$ covers the misconception $M_p$ and all misconceptions $M_i$ such that some literal $y^i_1$, $y^i_2$ or $y^i_3$ is of the form $\neg p$.

The idea of the transformation is that if we choose problem $PT_p$ then the propositional symbol $p$ will be *True*, whereas if we choose problem $PF_p$ then the propositional symbol $p$ will be *False*. Notice that if a problem of the form $PT_p$ or $PF_p$ is chosen and that problem covers the misconception $M_j$, then clause $C_j$ will be satisfied.

Regarding the required coverage of each misconception in the constructed instance of MCSO (vector $b$), we require to cover each misconception $M_i$ and $M_p$ at least once (as clauses must be satisfied and proposition symbols must be given some value, respectively), and the extra contribution for being covered more than needed (vector $w$)

will be 0 for both kinds of misconceptions. As we want to maximize the number of propositional symbols set to $True$, for all propositional symbols $p$ we need to reward the problems $PT_p$. We do it by giving some contribution to covering each misconception $M'_p$, but we do not force to cover them because they are not needed to satisfy the original 3-CNF instance. So, the required coverage for misconceptions $M'_p$ in vector $b$ will be 0, whereas the weight of their contribution for covering them more than needed, defined in vector $w$, will be 1. Notice that, as $M'_p$ is only covered by problem $PT_p$ (not by $PF_p$), valuations where propositional symbols $p$ are set to $True$ in the original *Max Ones* instance are preferred.

The cost of all problems will be 1, and the maximum total cost allowed for the whole MCSO instance (i.e. $k$) will be the number of propositional symbols. Recall that, for each propositional symbol $p$, we need to cover $M_p$ at least once. Thus, we need to choose at least one of the problems $PT_p$ or $PF_p$. However, as the maximum total cost allowed is the number of propositional symbols, it is not possible to take at the same time $PT_q$ and $PF_q$ for some propositional symbol $q$ (which would mean that the symbol $q$ is $True$ and $False$ at the same time in the *Max Ones* solution), because in that case the total cost would be greater than the number of propositional symbols. Since every valid solution of MCSO requires that the total cost due to the cost of problems is lower than or equal to the number of propositional symbols, all MCSO solutions denote a correct valuation of the propositional symbols.

Any valid solution of MCSO covers all misconceptions $M_i$, which means it satisfies all clauses $C_i$ of the original *Max Ones* problem. Moreover, it also provides a correct valuation of all the propositional symbols (that is, a symbol and the negation of the same symbol cannot hold at the same time). Regarding the total value to be maximized in the MCSO problem, it corresponds to the contribution due to the coverage of the $M'_p$ misconceptions. That is, it equals the number of problems $PT_p$ that have been selected, which equals the number of propositional symbols $p$ that are set to $True$ in *Max Ones*. Thus, in case we have a solution to MCSO with value $x$, the corresponding solution to *Max Ones* has exactly the same value $x$. This means that we have not only a PTAS-reduction from *Max Ones* to MCSO, but also an S, strict, and AP-reduction. So, we conclude that MCSO $\in$ NPO-hard. Moreover, as we can trivially prove MCSO $\in$ NPO (because its constraints can be checked in polynomial time), we conclude that MCSO $\in$ NPO-complete.

The Log-APX and NPO completeness of problems MCTO and MCSO implies, in particular, that both are Log-APX-hard. This means that if P $\neq$ NP then these problems cannot belong to APX, which in turn implies that no polynomial-time algorithm can guarantee that the ratio between the optimal solutions and the solutions found by the algorithm will be bound by some constant in the worst case. Thus, rather than seeking for specific-purpose approximation algorithms for these problems, using generic-purpose algorithms like Genetic Algorithms (which do not guarantee any performance ratio in the worst case but are known to provide good results on average) is a suitable choice in this case.

Next we introduce our second kind of problems, where we want to design an exam (set of exercises) where the total weight of each misconception is exactly that required by the teacher. Again, a decision problem and two optimization problems are defined. In the decision problem, we check whether there exists an exam with the desired coverage of misconceptions which also takes, to be completed, less time than some maximum allowed time. Two related optimization problems are considered too. In the first one, we minimize the time needed to complete the exam. In the second problem, the exact coverage is relaxed, though we wish to minimize the cumulated differences between the expected coverage of each misconception and the coverage reached by the composed exam.

**Definition 2.** Let us consider that we have $n$ questions or problems available $(P_1, \ldots, P_n)$ and $m$ misconceptions to be covered $(M_1, \ldots, M_m)$. Let $c \in \mathbf{N}^n$ be a vector of naturals where each value $c_i \in \mathbf{N}$ denotes the time needed to deal with problem $P_i$, $A \in \mathbf{R}^{m \cdot n}$ be a matrix where each value $a_{ij} \in \mathbf{R}$ denotes the coverage of the misconception $M_i$ provided by problem $P_j$, and $b \in \mathbf{R}^m$ be a vector where each value $b_j \in \mathbf{R}$ denotes the exact coverage we want to reach for each misconception $M_j$. Then,

- Given $k \in \mathbf{N}$, the *Exact Exam Coverage* problem, denoted by `EEC`, is the problem of finding out whether there exists $x \in \{0,1\}^n$ such that $c^T \cdot x \leq k$ and $A \cdot x = b$.
- The *Exact Exam Coverage Optimization* problem, denoted by `EECO`, is the problem of minimizing $c^T \cdot x$ subject to $x \in \{0,1\}^n$ and $A \cdot x = b$.
- Given $k \in \mathbf{N}$, the *Approximate Exam Coverage Optimization* problem, denoted by `AECO`, is the problem of minimizing $\sum_i |(\sum_j a_{ij} \cdot x_j) - b_i|$ subject to $c^T \cdot x \leq k$ and $x \in \{0,1\}^n$.

$\square$

We have the following properties.

**Theorem 2.** We have

(a) `EEC` $\in$ `NP-complete`.
(b) `EECO` $\in$ `NPO-complete`.
(c) `AECO` $\in$ `Log-APX-hard`.

*Proof.* Let us start proving (c). We can AP-reduce *Minimum Set Cover* (defined in the proof of Theorem 1) to `AECO` as follows.

Given a collection $\mathscr{C}$ of sets $\mathscr{C} = \{S_1, \ldots, S_k\}$ with $\bigcup_{S \in \mathscr{C}} S = \{e_1, \ldots, e_p\}$, we construct an instance of `AECO` as follows. We consider the elements $\{e_1, \ldots, e_p\}$ as misconceptions and we add an extra misconception $e_0$ that we do not want to cover (i.e. we want to cover it 0 times). We create a problem for each set $S_i$ that we call $Problem_{S_i}$, and we consider that this problem covers $k^2+1$ times each misconception $e_j \in S_i$, whereas it covers only once the misconception $e_0$. Besides, for each $i \in \{1, \ldots, p\}$ and $j \in \{1, \ldots, k-1\}$ we define an additional problem $Problem_{ij}$ that covers the misconception $e_i$ exactly $k^2+1$ times (and it does not cover $e_0$). Our coverage goals will be covering each misconception $e_i$ with $i \in \{1, \ldots, p\}$ exactly $k \cdot (k^2+1)$ times, and covering each misconception $e_0$ exactly 0 times. The cost of each problem will be 0, so we will not need to define any total cost limit $k$ in the constructed `AECO` instance (it could be assigned any arbitrary value). There exist two different solutions to the `AECO` instance:

1. Solutions where the accumulated distance to the desired misconception coverage is at least $k^2+1$. In this case, the approximation ratio of this solution in AECO (i.e. the ratio between the value of the solution and the optimal solution) is $(k^2+1)/k$ or worse (i.e. bigger). In order to prove it, let us reason about the optimal solution. Note that, if we construct an exam using all problems of type $Problem_{S_i}$, these problems alone will make each misconception be covered at least $k^2+1$ times and at most $k(k^2+1)$ times (depending on the number of problems covering the misconception). The remaining coverage until reaching the desired $k(k^2+1)$ coverage for each misconception can be obtained by selecting as many problems of type $Problem_{ij}$ as needed (notice that, for each misconception, there are $k-1$ of them, so if problems of the form $Problem_{S_i}$ cover it at least once, then the $k(k^2+1)$ desired coverage can be reached by taking the appropriate number of $Problem_{ij}$ problems). Hence, the distance from the optimal exam to the exact desired coverage can only be due to the number of times misconception $e_0$ is covered, which is a value in $1,\ldots,k$. That is, the cost of the optimal exam is lower than or equal to $k$. Hence, for a solution whose cost is greater than or equal to $k^2+1$, its approximation ratio is greater than or equal to $(k^2+1)/k$.
2. Solutions whose accumulated distance to the desired coverage is in the $1,\ldots,k$ range. This situation can only happen if, for each misconception, we have chosen $k$ problems covering it. Since there are $k-1$ problems of the form $Problem_{ij}$ covering it, at least one of them is of the form $Problem_{S_i}$, so the MSC element represented by this misconception is actually covered. We conclude that all elements of the original MSC instances are covered by the corresponding AECO solution. Note that the cost of the AECO solution is the number of times $e_0$ is covered, which equals the number of taken problems of the form $Problem_{S_i}$. In turn, this is the number of sets taken in the MSC instance. Thus, a solution to AECO has value $x$ iff the corresponding solution to MSC has $x$ value. This applies to optimal solutions too. Thus, in this case the approximation ratios of solutions is exactly the same in both problems, MSC and AECO.

The AP-reduction between both problems will map approximation ratios as follows. AP-reductions require that, if the approximation ratio for the second problem is $\leq r$, then the approximation ratio for the first problem is $\leq 1+\alpha\cdot(r-1)$ for some constant $\alpha$. Let us use $\alpha=1$. Then, if the approximation ratio of AECO is $\leq r$, the approximation ratio of MSC must also be $\leq r$. Let us describe the function mapping AECO solutions back into MSC solutions. If we have an AECO solution of type (1) (that is, the deviation from the desired coverage is at least $k^2+1$, so the approximation ratio for AECO is greater than or equal to $(k^2+1)/k$), then we return the MSC solution which selects *all* subsets $S_i$ (whose approximation ratio in MSC is $\leq k$, as any set cover needs to select at least one subset). Thus, in this case the approximation ratio of the returned MSC solutions is at least as good as the ratio of the original AECO solution, as required by the AP-reduction. On the contrary, when we have a solution of type (2), the approximation ratios are equal in both problems. Thus, we have an AP-reduction in both cases. Thus, AECO $\in$ Log-APX-hard.

In order to prove (a), we can trivially adapt the previous AP-reduction to polynomially reduce *Set Cover* (the decision version of MSC) into EEC, which implies the NP-hardness of EEC. Since EEC $\in$ NP (as we can check $A \cdot x = b$ and $c^T \cdot x \leq k$ in polynomial time), we conclude EEC $\in$ `NP-complete`.

In order to prove (b), we S-reduce (see e.g. [4]) the `NPO-complete` problem *Min Ones* [13] (that is, the problem of satisfying all clauses of a 3-CNF formula while *minimizing* the number of propositional symbols which are set to *True*) to EECO as follows. For each clause $C_i$ of the original *Min Ones* instance, we define a misconception $M_i$. Besides, for each propositional symbol $p$, we define two problems $PT_p$ and $PF_p$, representing setting the propositional symbol $p$ to *True* or *False*, respectively. Each problem $PT_p$ covers all misconceptions $M_i$ such that setting $p = True$ satisfies clause $C_i$, whereas problem $PF_p$ covers all misconceptions $M_i$ such that $p = False$ satisfies $C_i$.

In addition to that, for each propositional symbol $p$ we create a new misconception $MP_p$ such that both $PT_p$ and $PF_p$ cover misconception $MP_p$. Finally, for each clause $C_i$ we create two additional problems $PD1_i$ and $PD2_i$ such that they cover misconception $M_i$. Problems of the form $PT_p$ have cost 1, whereas the rest of the problems have cost 0.

We require to cover each misconception $M_i$ exactly three times, and each misconception $MP_p$ exactly once (so that $p$ and $\neg p$ cannot be true at the same time). If the selected problems of the form $PT_p$ and $PF_p$ satisfy some clause $C_i$, then the corresponding misconception $M_i$ will be covered once, two times, or three times by these problems (depending on the number of literals of $C_i$ satisfied by the valuation represented by the selected problems of the form $PT_p$ and $PF_p$). In this case, covering misconception $M_i$ exactly three times (as required in the EECO instance) just requires taking zero, one, or two problems of the form $PD1_i$ and $PD2_i$, whose cost is 0. We conclude that any solution for the EECO instance represents a valuation of the propositional symbols such that all clauses $C_i$ are satisfied, and the goal of EECO is minimizing the number of symbols set to 1 (because only $PT_p$ problems have a cost greater than zero, in particular 1). Thus, the cost of a solution for the EECO instance is exactly the same as the cost of the corresponding solution of the *Min Ones* instance, and this applies to optimal solutions too. Hence, we have an S-reduction between both problems, so EECO $\in$ `NPO-hard`.

Moreover, as we can trivially prove that EECO $\in$ NPO (because its constraints can be checked in polynomial time), we conclude that EECO $\in$ `NPO-complete`.

Again, since both optimization problems are `Log-APX-hard` (note that the hardness in NPO implies it), using Genetic Algorithms rather than tailored specific-purpose algorithms for these problems is a suitable choice.

## 3 Algorithms

For each of the four optimization problems described in the previous section (MCTO, MCSO, EECO, and AECO) we have implemented a genetic algorithm. The basic representation in all cases is the same. Each individual in the population (i.e., each chromosome) is the bit vector $x = (x_1, \ldots, x_n)$ to be set, where $n$ is the number of questions available in the system. Each bit denotes whether the corresponding question is to be included or not in the solution. That is, a bit vector $(1, 0, 0, 1)$ represents a solution where only the first and the last question are included in the solution. In addition to the representation of chromosomes, we need to deal with the representation of all the information of the problem. In particular, we need to handle a vector representing the time needed to present and to solve each problem (i.e. its cost), a matrix indicating the coverage level of each question for each misconception, and a vector denoting the minimum coverage required for each misconception (in the case of MCTO and MCSO) or the exact desired coverage for each misconception (in the case of EECO and AECO).

The difference among the four problems is the fitness function to be optimized. In the case of MCTO, we minimize the cost of the time vector (i.e., the sum of costs due to all values $c_k$ such that $x_k$ is set to 1) if all constraints hold (i.e., $A \cdot x \geq b$). If the constraints do not hold, then the fitness function returns a value bigger than the sum of costs of all questions. Moreover, in this case the returned value is also proportional to the *distance* to fulfilling all constraints.

In the case of MCSO, we maximize the extra coverage of the questions, that is, the sum of additional coverage for each of the misconceptions

$$\sum_i w_i \cdot ((\sum_j a_{ij} \cdot x_j) - b_i)$$

if all constraints hold (i.e, the minimum coverage is satisfied for each misconception, that is, $A \cdot x \geq b$, and the total time required to deal with all the questions is small enough, i.e. $c^T \cdot x \leq k$). When the constraints do not hold, the fitness function returns a value smaller than the minimum possible value, and proportional to the distance to fulfilling all constraints.

In the case of EECO we minimize the same time cost as in MCTO, but subject to a stronger constraint ($A \cdot x = b$). If the constraints do not hold we proceed in the same way as in MCTO. In the case of AECO the constraints are the same as in MCSO, but we have to *minimize* a function, not to maximize. In particular, the fitness function computes the *distance* to the optimal coverage distribution, that is, we want to minimize $|A \cdot x$ - $b|$. If the constraints do not hold, the fitness function returns a value bigger than the maximum possible distance, and this value is proportional to the distance to fulfilling the constraints.

Note that, given any three variables $x_i$, $x_j$, $x_k$ represented at different positions of the chromosome, there is no (a priori) reason to believe that the suitability of the value given to $x_i$ will depend more on the value given to $x_j$ than on the value given to $x_k$. Consequently, the crossover function just randomly picks which genes (i.e., bits $x_i$) are inherited from each parent, that is, no specific crossover point is considered.

Regarding other aspects of the algorithm, we use elitist selection, so that the quality of the best solution of each generation increases monotonically. In order to fix the parameters configuration of the algorithm, we have performed a wide range of experiments over a set of randomly generated instances of the problems. After analyzing the results, we have set the population size to 100, the mutation probability to 0.01, and the number of iterations to 1000. In order to avoid over-tunning [2,19,20], we clearly separate this training phase from the evaluation phase. Thus, we have provided such fix Genetic Algorithm configuration to the teachers of the subject Discrete Mathematics, and they have used it to generate both exams and sets of exercises to be presented during the course. Their satisfaction with the tool was very high. First, they acknowledged that the selection of exercises to be presented in the last sessions of the course improved a little bit, as they could maximize the coverage of misconceptions. However, the main improvement they acknowledged appears in the selection of exercises to create exams. Adjusting an exam so that each misconception minors the marks of the students the desired amount of points is a really difficult task. In fact, in many situations it is impossible to do that, and we can only approximate the desired weights for each error. By using our tool, the task of creating exams is simplified. This is particularly useful when several *similar* exams have to be created to evaluate different students of the same subject in different days.

## 4 Conclusions and Future Work

In this work we have considered the problem of selecting a set of exercises fulfilling different conditions. By doing so, we can use it to select exercises covering a list of misconceptions a given amount of times and minimizing the time needed to present them; or to cover these misconceptions with a given amount of time while maximizing the extra coverage; or to select a set of exercises to create an exam where each misconception minors the marks of the student an exact amount of points (while minimizing the time needed to solve the exam); or to design an exam which can be solved in a given amount of time, while minimizing the distance between the desired mark reductions due to misconceptions and the actual ones.

We have proved the approximability complexity of these problems. Moreover, we have provided genetic algorithms solving them. The usefulness of such algorithms has been proved by applying them in the context of a specific subject of our university. The teachers of this subject (Discrete Mathematics) have used these algorithms for generating both exams and lists of exercises to be presented to students, obtaining satisfactory results.

As future work we plan to perform two independent extensions. First, our aim is to apply our algorithms with other repertories of exercises from other subjects. Second, in order to speedup the computations, we plan to provide a parallel version of our implementations by using the parallel library described in [7].

## References

1. Giorgio Ausiello and Vangelis Th Paschos. Reductions that preserve approximability. *Handbook of Approximation Algorithms and Metaheuristics: Methologies and Traditional Applications*, 1, 2018.
2. Mauro Birattari. *Tuning metaheuristics: a machine learning perspective*, volume 197. Springer, 2009.
3. Shu-Heng Chen. *Genetic algorithms and genetic programming in computational finance*. Springer Science & Business Media, 2012.
4. Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proc. 12th IEEE Conf. on Computational Complexity*, pages 262–273. IEEE, 1997.
5. Dipankar Dasgupta and Zbigniew Michalewicz. *Evolutionary algorithms in engineering applications*. Springer Science & Business Media, 2013.
6. K. de Jong. *Evolutionary computation: a unified approach*. MIT, 2006.

7. A. de la Encina, M. Hidalgo-Herrero, P. Rabanal, and F. Rubio. A parallel skeleton for genetic algorithms. In *Int. Work-Conference on Artificial Neural Networks, IWANN 2011*, volume 6692 of *LNCS*, pages 388–395. Springer, 2011.
8. Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.
9. Mercedes Hidalgo-Herrero, Ismael Rodríguez, and Fernando Rubio. Testing learning strategies. In *Fourth IEEE Conference on Cognitive Informatics, 2005.(ICCI 2005).*, pages 212–221. IEEE, 2005.
10. Mercedes Hidalgo-Herrero, Ismael Rodríguez, and Fernando Rubio. Comparing learning methods. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, 3(3):12–26, 2009.
11. S.G. Kolliopoulos and N.E. Young. Approximation algorithms for covering/packing integer programs. *Journal of Computer and System Sciences*, 71:495–505, 2005.
12. Stefan Kratsch, Dániel Marx, and Magnus Wahlström. Parameterized complexity and kernelizability of max ones and exact ones problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 489–500. Springer, 2010.
13. Stefan Kratsch and Magnus Wahlström. Preprocessing of min ones problems: A dichotomy. In *International Colloquium on Automata, Languages, and Programming, ICALP 2010*, pages 653–665. Springer, 2010.
14. Kim F Man, Kit Sang Tang, and Sam Kwong. *Genetic algorithms for control and signal processing*. Springer Science & Business Media, 2012.
15. M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
16. Stjepan Oreski and Goran Oreski. Genetic algorithm-based heuristic for feature selection in credit risk assessment. *Expert systems with applications*, 41(4):2052–2064, 2014.
17. Sankar K Pal and Paul P Wang. *Genetic algorithms for pattern recognition*. CRC press, 2017.
18. V.Th. Paschos. An overview on polynomial approximation of np-hard problems. *Yugoslav Journal of Operations Research*, 19(1):3–40, 2009.
19. Pablo Rabanal, Ismael Rodríguez, and Fernando Rubio. On the uselessness of finite benchmarks to assess evolutionary and swarm methods. In *Proc. Comp. Genetic and Evolutionary Computation Conference, GECCO 2015*, pages 1461–1462. ACM, 2015.
20. Pablo Rabanal, Ismael Rodríguez, and Fernando Rubio. Assessing metaheuristics by means of random benchmarks. *Procedia Computer Science*, 80:289–300, 2016.
21. Ismael Rodríguez, Pablo Rabanal, and Fernando Rubio. How to make a best-seller: Optimal product design problems. *Applied Soft Computing*, 55(C):178–196, 2017.
22. Ismael Rodríguez, Fernando Rubio, and Pablo Rabanal. Automatic media planning: Optimal advertisement placement problems. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 5170–5177. IEEE, 2016.
23. Kumara Sastry, David E Goldberg, and Graham Kendall. Genetic algorithms. In *Search methodologies*, pages 93–117. Springer, 2014.