The Stochastic Test Collection Problem: Models, Exact and Heuristic Solution Approaches

Yifat Douek-Pinkovich, Irad Ben-Gal, Tal Raviv*
Department of Industrial Engineering, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel
E-mail: yifatdouek@gmail.com, bengal@tauex.tau.ac.il, talraviv@tauex.tau.ac.il

December 2021

Abstract

The classic test collection problem (TCP) selects a minimal set of binary tests needed to classify the state of a system correctly. The TCP has applications in various domains, such as the design of monitoring systems in engineering, communication, and healthcare. In this paper, we define the *stochastic test collection problem* (STCP) that generalizes the TCP. While the TCP assumes that the tests' results can be deterministically mapped into classes, in the STCP, the results are mapped to probability distributions over the classes. Moreover, each test and each type of classification error is associated with some cost. A solution of the STCP is a subset of tests and a mapping of their results to classes. The objective is to minimize the weighted sum of the tests' costs and the expected cost of the classification errors. We present an integer linear programming formulation of the problem and solve it using a commercial solver. To solve larger instances, we apply three metaheuristics for the STCP, namely, Tabu Search (TS), Cross-Entropy (CE), and Binary Gravitational Search Algorithm (BGSA). These methods are tested on publicly available datasets and shown to deliver nearly optimal solutions in a fraction of the time required for the exact solution.

Keywords: Combinatorial Optimization, The test collection problem, Integer linear programming, Metaheuristics.

* Corresponding author

1 Introduction

The well-studied minimum test collection problem (TCP) is known in the literature as *the minimum test set problem* or *the minimum test cover problem*. Halldórsson et al. (2001) described the minimum TCP as follows: Given a set of entities (e.g., individuals) and a set of binary attributes (tests) that may or may not occur in each entity, the incidence vector of each entity represents a *reading*. The goal is to find the minimal subset of attributes (a test collection) such that each entity can be uniquely identified from the information on which of the attributes in the test collection it contains. In this way, the reading's coordinates that represent tests included in test collection form a unique binary vector referred to as *signature* for distinguishing it from all the other entities. For example, in the domain of botanic taxonomy, given a set of mushroom varieties (entities) and a collection of binary mushroom attributes (such as bad odor, spore-print in red color, whether it is found in a large group, etc.) such that each attribute can characterize some mushroom varieties. To identify the mushroom type efficiently, one looks for a minimum subset of attributes that is enough to distinguish between all the varieties.

This paper presents the stochastic test collection problem (STCP), which complements and generalizes the minimum TCP. The STCP is defined as follows: we are given a set of tests with categorical outputs and all possible combinations of the results of these tests for a population of tested entities (e.g., patients). Each such combination of the tests' results is called a reading and is associated with a probability distribution over a given finite set of *classes* (e.g., diagnoses) and with the probability of obtaining this reading when sampling an entity from the population (i.e., the relative frequency of the reading). A subset of some selected tests is called *configuration*. The outcomes of the tests in a configuration of a given reading are jointly called a *signature*. Note that several different readings may have the same signature for a particular configuration and hence can be indistinguishable. In a situation where one needs to determine the state of a system based on a particular signature, classification errors may occur. A classification error of type A-B is said to happen when a signature is classified as B, while it is actually originated from a subject of class A. The input of the STCP includes an error cost matrix that specifies a cost associate with each type of error. A solution of the STCP consists of two components: (1) a configuration; (2) a mapping of each possible signature into a class. The objective is to minimize a weighted sum of the cost of the tests that comprise the configuration as well as the expected classification error cost implied by the mapping. The tests' costs are weighted since they may represent a one-time initial investment, while the errors may occur each time the tests are applied. For example, the cost associated with the installation of the sensors in a water network should be amortized in terms of a single usage, which is equivalent to performing one test.

The trade-off between the two components of the objective function follows as executing more tests (or more accurate and expensive ones, e.g., performing additional blood tests to improve the patient's diagnosis) is likely to reduce the chances of classification errors and thus their expected cost but will increase the total testing cost.

The challenge is that the class can rarely be inferred from the signature of a single test. Instead, it is inferred from a combination of the results obtained from several ones.

A formal definition of the problem with mathematical notation is presented in Section 2.

The TCP can be seen as a special case of the STCP in which (a) each reading is associated deterministically with a unique class; (b) the costs of all the tests are equal and set to one; and (c) the classification error costs are set to prohibitively large numbers (e.g., larger than the number of tests).

In the STCP, instead of a one-to-one relation between readings and classes, it allows a probabilistic many-to-many relation. It thus enables various readings-to-class mappings, as often happens in reality. In such a case, a deterministic diagnosis is often impossible, hence the need to introduce the classification error cost into the objective function.

The TCP decision version was shown to be NP-hard by (Garey and Johnson, 1979). Halldórsson et al. (2001) and De Bontridder et al. (2003) established the APX-Hardness of the minimum TCP. They provided constant ratio approximation algorithms for some special cases with a small number of positive attributes in each reading. Therefore, the intractability of the STCP follows.

Many authors have studied various applications of the TCP and its extensions. Such applications include sensor placement in structures and networks, e.g., in Kammer (1991), Sela et al. (2016), Douek-Pinkovich et al. (2020); robotics, e.g., in Hovland and McCarragher (1997); energy consumption strategies, e.g., in Slijepcevic and Potkonjak (2001); medical diagnosis, e.g., in Wendt and Potkonjak (2011); protein identification Halldórsson et al. (2001) and De Bontridder et al. (2003); process monitoring, e.g., in Bacher and Ben-Gal (2017), among others.

The STCP can naturally enrich most of the applications described above since the stochastic relation between the tests' readings and the state of the system is an inherent characteristic of almost any realistic testing system. The stochasticity stems both from the noise in the tests and from the fact that it is typically too costly to perform enough tests to eliminate all possible uncertainties. With modern sensing systems, a large quantity of data is collected, and it can be used to estimate the probability distributions required as input for the STCP model. In other situations, these distributions can be calculated based on the physical characteristics of the tested system.

The STCP also extends another model, known as the *generalized test collection problem* (GTCP) that was introduced by Bertolazzi et al. (2016) and studied by Douek-Pinkovich et al. (2020). In the GTCP, the tests' outputs are categorical rather than binary, different readings may be associated with a single class, and each test is associated with a cost. A solution of the GTCP is a collection of tests and a correct mapping of the signatures to the classes. The objective is to find a collection with minimal costs. As in the TCP, classification errors are not allowed. Douek-Pinkovich et al. (2020) apply their model and solution method to sensors' placement problem in urban water networks, introduced by Sela et al. (2016). In water networks, sensors that detect pressure waves caused by pipe bursts can be placed on each node in the network. A burst in each edge is associated with signals in several sensors located close to it. The goal is to select a minimal subset of nodes where sensors should be placed to detect each fault in the system.

The main difference between the GTCP and the STCP is that in the former, each reading is deterministically associated with a single class, while in the latter, a reading is associated with a probability distribution over the classes. Consequently, in the GTCP, the goal is to select a set of tests that enable classifying the subject deterministically. In the STCP setting, the classification is inherently subject to errors that should be minimized.

The GTCP can be seen as a generalization of the TCP. Using the above-mentioned mushrooms example for the illustration purpose, a mushroom variety (characterized by its reading) can be classified as toxic or nontoxic. Instead of identifying the specific mushroom type, one may wish to select a minimum-cost set of attributes that can determine the mushroom's toxicity. The STCP enriches the GTCP in the sense that it can handle uncertainty. Naturally, mushroom attributes are subject to noise due to measurement errors. Thus, a mushroom variety can be classified with some probability as toxic and with another probability as edible. Hence, the objective function must now include the error cost of identifying a mushroom as toxic while it is not such and vice versa. In such a case, it is likely that the error of classifying a toxic mushroom as edible will be set to have a very high cost, while the opposite error may be considered less costly.

In many realistic applications, the STCP can be considered as a more realistic model since: *i*) the readings observed in real-world scenarios may not necessarily identify the class of the subjects with certainty; *ii*) the readings are often affected by some (random) noise; *iii*) sometimes it is preferred to save testing costs by including a cheap set of tests that is not sufficient by itself for a deterministic classification. For example, in the water network sensor placement problem, bursts in the same location may result in different readings. In fact, the intensity of the burst, and sediments in pipes, affect the signals and contribute to the uncertainty. Another example for the STCP relevance is related to the known wine classification problem (e.g., Cortez et al., 2009), where the wine class is defined by its quality score and characterized by a numerical scale in the range of 3-9. This example represents well the trade-off between the testing costs and the classification error. Different scores can be reflected by the same readings, therefore mapping the results to probability distributions over the classes is much more feasible than a deterministic classification (even with all the considered variables).

Table 1 summarizes the differences between the STCP, the GTCP, and the TCP. The first column of the table summarizes the input types and the objective function. In the second, third, and fourth columns, the properties of the three problems are detailed.

This paper presents an effective, exact algorithm for the STCP based on an integer linear program (ILP) formulation. It also applies three metaheuristic methods to solve larger instances of the STCP. Namely, Tabu Search (TS), Cross-Entropy (CE), and Binary Gravitational Search Algorithm (BGSA). Finally, it shows the merits of our generalization by comparing it to the solution of equivalent TCP instances.

Table 1: The TCP, GTCP, and STCP

	TCP	GTCP	STCP
Input types:			
Test's results	binary	categorical	
Reading to class relation	one to one	many to one	many to many
Test's cost	identical	any	
Classification errors are allowed	No		Yes, with a cost
Objective function:	Minimize the cost (or number) of the selected tests		Minimize the cost of the selected tests and the expected error cost

Finally, let us note that the known Feature Selection (FS) problem is also related to the STCP in the sense that in both problems, the goal is to select a subset of characteristics of an entity that enables its proper classification. However, the two problems differ in their objectives and settings. In the STCP, one looks for a minimal set of characteristics that enable to distinguish among entities that belong to different classes with a sufficient probability measure. It is further assumed that all the relevant entities, with their characteristics, are available as input to the problem. Since one considers the trade-off between the cost of the information obtained for the classification and the expected cost of misclassification, a minimal set of characteristics without a need for redundancy should be selected. On the other hand, in feature selection problems, the goal is to select a subset of characteristics that minimizes the probability of misclassification over a test set that is not available during the training phase. Thus, over-fitting and under-fitting effects should be considered. Accordingly, in feature selection, one may look for solutions with redundancy since future test datasets are unknown at the training stage.

The contributions of this paper are in formulating the stochastic variant of the TCP and in presenting effective heuristic methods to solve it.

The rest of the paper is organized as follows. In Section 2, we present some formal notation and a mathematical formulation of the problem. In Section 3, we demonstrate the properties of the problem using a small illustrative example. Sensitivity analysis is carried out to demonstrate some counterintuitive properties of the problem and its optimal solutions. In Section 4, we present an ILP formulation of the STCP. In section 5, we present heuristic methods to solve the STCP based on the TS, CE, and BGSA metaheuristics. In Section 6, the proposed heuristics are tested against the exact solution obtained from our ILP formulation based on publicly known data from the UCI Machine Learning Repository (Dua and Graff, 2019). Also, a dataset with probabilistic labels is applied to show the advantages of the STCP. A comparison between the STCP and the TCP is also presented. Some concluding remarks are offered in Section 7.

2 Notation and problem definition

To present our mathematical formulation of the STCP, we use the following notation, where bold letters denote vectors and matrices:

N	The set of candidate tests available in a given system; the number of tests is denoted by $n = \|N\|$. $S \in N$ is a subset of selected tests called a *configuration*.
c_i	The cost of test i for any $i \in N$.
V_i	The set of outputs or results that can be obtained from test i.
R	The set of valid readings, $R \subseteq V_1 \times V_2 \times \cdots \times V_n$; for each reading $\tilde{\mathbf{r}} \in R$, we denote the result of the i^{th} test as $\tilde{r}_i$.
K	The set of possible classes; $K = \{1, \ldots, k\}$.
λ_{kl}	The misclassification error of type (k, l), $k, l \in K$, i.e., the cost of classifying an object as class l when its true class is k.
$p(\tilde{\mathbf{r}})$	The a-priory probability of obtaining the reading $\tilde{\mathbf{r}} \in R$.
$p(k\|\tilde{\mathbf{r}})$	The conditional probability of class $k \in K$ given the reading $\tilde{\mathbf{r}} \in R$.
β	The relative weight of the testing cost in the objective function.

Let us denote the prior probability of each class by $p(k)$. This probability is related to the above parameter as follows.

$$p(k) = \sum_{\tilde{\mathbf{r}} \in R} p(k|\tilde{\mathbf{r}}) p(\tilde{\mathbf{r}}) \tag{1}$$

Let the conditional probability of each reading given a class by $p(\tilde{\mathbf{r}}|k)$. This probability is determined by the above parameters by Bayes' rule,

$$p(\tilde{\mathbf{r}}|k) = \frac{p(k|\tilde{\mathbf{r}}) p(\tilde{\mathbf{r}})}{p(k)} \tag{2}$$

For each configuration $S \subset N$, define $R(S)$ as the set of all its signatures. I.e., partial readings can be obtained from the results of the selected tests. When the configuration is known, we denote the signature of $\tilde{\mathbf{r}}$ by **r.** The signature $\mathbf{r} \in R(S)$ is a vector of dimension $|S|$. For convenience, the elements r_i of the signature vectors are indexed by the original indices of the tests in N. For example, if $N = \{1,\ldots,5\}$ and $\mathbf{r} \in R(\{1,2,5\})$, then $\mathbf{r} = (r_1, r_2, r_5)$. In this example, r_5 is the third element of the signature **r**. We denote the set of all the possible readings from which signature **r** can be obtained when the configuration is S as $Q(S, \mathbf{r})$. That is, $Q(S, \mathbf{r}) = \{\tilde{\mathbf{r}} \in R: \tilde{r}_i = r_i \quad \forall i \in S\}$.

Next, for each configuration $S \subset N$ and $\mathbf{r} \in R(S)$, it is possible to calculate the following three probability components: the probability of a signature given class k, $p_S(\mathbf{r}|k)$; the prior probability $p_S(\mathbf{r})$ of signature $\mathbf{r} \in R(S)$; and the a-posteriori probability $p_S(k|\mathbf{r})$ that the class is $k \in K$ given that the signature is $\mathbf{r} \in R(S)$. These probabilities can be calculated using Equations (3)-(5). For configuration S, the conditional probability of class k when observing a signature **r**, $p_S(k|\mathbf{r})$, is calculated with Equation (5) using Bayes' rule.

$$p_S(\mathbf{r}|k) = \sum_{\tilde{\mathbf{r}} \in Q(S,\mathbf{r})} p(\tilde{\mathbf{r}}|k) \tag{3}$$

$$p_S(\mathbf{r}) = \sum_{\tilde{r} \in Q(S,\mathbf{r})} p(\tilde{\mathbf{r}}) = \sum_{k \in K} p(k) \cdot p_S(\mathbf{r}|k) \tag{4}$$

$$p_S(k|\mathbf{r}) = \frac{p_S(\mathbf{r}|k) \cdot p(k)}{p_S(\mathbf{r})} \tag{5}$$

Consider a given configuration $S \subset N$. The expected classification error cost for signature $\mathbf{r}$ of S if it is mapped to class l is:

$$E_S(\mathbf{r}|l) = \sum_{k \in K} \lambda_{kl} \cdot p_S(k|\mathbf{r}) \tag{6}$$

let $l_S^*: R(S) \longrightarrow K$ be a function that maps each possible signature of S to a class that minimizes the expected classification error cost.

$$l_S^*(\mathbf{r}) = \operatorname{argmin}_{l \in K}\{E_S(\mathbf{r}|l)\}. \tag{7}$$

The minimum expected classification error cost for signature $\mathbf{r}$ is:

$$E_S^*(\mathbf{r}) = \min_{l \in K}\{E_S(\mathbf{r}|l)\} \tag{8}$$

Note that (7) coincides with the "minimum Bayes risk decision rule" as given, for example, in Duda et al. (2012).

The STCP can now be formulated mathematically. Namely, given an instance of the problem $[N, R, K, \mathrm{p}(\tilde{\mathbf{r}}), \mathrm{p}(k|\tilde{\mathbf{r}}), \boldsymbol{\lambda}, \mathbf{c}]$, select a configuration S, such that the weighted sum of the expected classification error and test costs are minimized,

$$\min_{S \subseteq N} \left\{ \sum_{\mathbf{r} \in R(S)} p_S(\mathbf{r}) E_S^*(\mathbf{r}) + \beta \sum_{i \in S} c_i \right\} \tag{9}$$

The weight coefficient is used to adjust the scale of the error cost and the testing cost. Higher values of β lead to testing configurations that are more prone to classification errors. The designer of the testing system can use β to explore the efficiency frontier of the costs of classification errors and costs of tests. Note that multiplying β by a constant is equivalent to dividing the error cost matrix, $\boldsymbol{\lambda}$, by that constant or to multiply the testing costs, c_i by it. Note that given the set of tests S, the set of signatures is uniquely defined by $R(S)$, while the optimal mapping of each signature to a class is given by (7).

3 Motivating example

Let us demonstrate the problem using the following small example. Consider a medical testing system comprising three potential tests aimed at detecting a viral disease. Each test produces a binary result, i.e., the result of medical test i may be either $V_i = 0, \text{or } 1$. The input, in this case, in terms of the notation presented in Section 2, is:

N	$\{1,2,3\}$
c_i	[2,0.5,0.5]
β	1
V_i	$\{0,1\}$ for $i = 1,2,3$; i.e., the result of each binary test can be either 0 or 1.
R	$V_1 \times V_2 \times V_3$, all possible combinations of the tests' results. See also the first group of columns in Table 2.
K	$\{N, P\}$; $\mathcal{N}$ for Negative and $\mathcal{P}$ for Positive

λ_{kl} $\begin{bmatrix} 0 & 50 \\ 50 & 0 \end{bmatrix}$; i.e., both false-positive and false-negative costs are equal to 50.

$p(\tilde{\mathbf{r}})$ See the second group of columns in Table 2.

$p(k|\tilde{\mathbf{r}})$ See the third group of columns in Table 2.

Table 2: Valid readings $\tilde{\boldsymbol{r}} \in R$ and the probabilities $p(\tilde{\boldsymbol{r}})$, $p(\tilde{\boldsymbol{r}}|k)$, and $p(k|\tilde{\boldsymbol{r}})$.

Test Readings, R			$p(\tilde{\mathbf{r}})$	$p(\tilde{\mathbf{r}}\|k)$		$p(k\|\tilde{\mathbf{r}})$	
Test 1	Test 2	Test 3		$\mathcal{N}$	$\mathcal{P}$	$\mathcal{N}$	$\mathcal{P}$
0	0	0	0.067	0.035	0.105	0.289	0.711
0	0	1	0.119	0.015	0.245	0.070	0.930
0	1	0	0.105	0.105	0.105	0.550	0.450
0	1	1	0.135	0.045	0.245	0.183	0.817
1	0	0	0.097	0.140	0.045	0.792	0.208
1	0	1	0.080	0.060	0.105	0.411	0.589
1	1	0	0.251	0.420	0.045	0.919	0.081
1	1	1	0.146	0.180	0.105	0.677	0.323

In this small example, the solution can be readily calculated by an exhaustive search overall $2^3 = 8$ possible test configurations. The value of each subset S is calculated by enumerating all the signatures in $R(S)$. An example of such calculations for the configuration $S = \{1,2\}$ is described in Table 3. First, all signatures obtained from the subset S are shown in the first group of columns. Next, for each signature $\mathbf{r} \in R(S)$ and each diagnosis $k \in K$, the probabilities $p_S(\mathbf{r}|k)$, $p_S(\mathbf{r})$, and $p_S(k|\mathbf{r})$ are calculated using (3)-(5) and are shown in the second, third, and fourth group of columns, respectively. Now, the expected error cost of diagnosing l when the true diagnosis is k can be seen in the fifth group of columns and is calculated when l is decided; i.e., it is $E_S(\mathbf{r}|l)$, as given in (6). Equation (7) shows the diagnosis that minimizes the expected classification error cost for the signature $\mathbf{r}$ (sixth group of columns). Its expected classification error cost is given by (6) and can be seen in the seventh group of columns. The results of multiplying the minimum expected classification error cost by the probability of obtaining each signature $\mathbf{r} \in R(S)$ can be seen in the eighth group of columns. The sum of this column is 12.25, which denotes the expected classification error cost for the subset as given by the first addend of (9). The second addend of (9) indicates the cost of the tests, which is 2.5. Thus, the expected total cost of the subset $S = \{1,2\}$ is 14.75.

Table 3: Calculating the expected total cost for $S = \{1,2\}$

$R(S)$		$p_S(\mathbf{r}\|k)$		$p_S(\mathbf{r})$	$p_S(k\|\mathbf{r})$		$E_S(\mathbf{r}\|l)$		$l_S^*(\mathbf{r})$	$E_S^*(\mathbf{r})$	$p_S(\mathbf{r})E_S^*(\mathbf{r})$
Test 1	Test 2	$\mathcal{N}$	$\mathcal{P}$		$\mathcal{N}$	$\mathcal{P}$	$l=\mathcal{N}$	$l=\mathcal{P}$			
0	0	0.05	0.35	0.19	0.15	0.85	42.57	7.43	$\mathcal{P}$	7.43	1.38
0	1	0.15	0.35	0.24	0.34	0.66	32.81	17.19	$\mathcal{P}$	17.19	4.13
1	0	0.2	0.15	0.18	0.62	0.38	19.01	30.99	$\mathcal{N}$	19.01	3.38
1	1	0.6	0.15	0.40	0.83	0.17	8.49	41.51	$\mathcal{N}$	8.49	3.38

$\sum_{\mathbf{r}\in R(S)} p_S(\mathbf{r})E_S^*(\mathbf{r})$	12.25
$\sum_{i\in S} c_i$	2.5
Expected total cost:	14.75

In Table 4, for each possible configuration (given in the first column), we present the expected classification error cost (second column), the testing cost (third column), and the expected total cost, which is the error cost plus the testing cost (in the fourth column). One can observe that the configuration {1,2,3}, i.e., when using all the tests, is the one that minimizes the cost function (9), resulting in a value of 14.01.

Table 4: The expected classification error cost, testing cost, and expected total cost of each configuration

Configuration	**The expected classification error cost**	**Testing cost**	**Total cost**
{1,2,3}	11.01	3	14.01
{1,2}	12.25	2.5	14.75
{1,3}	12.25	2.5	14.75
{2,3}	15.00	1	16.00
{1}	12.25	2	14.25
{2}	18.13	0.5	18.63
{3}	15.00	0.5	15.50
{ }	22.50	0	22.50

Interestingly, the second-best configuration is {1}. Adding tests 2 or 3, i.e., using the configurations {1,2} or {1,3}, results in the same classification error cost as {1} but incurs a higher testing cost; this demonstrates the complex structure of the problem and that a simple greedy or local search heuristic is unlikely to solve it.

Numerical analysis of the optimal decision, as a function of the classification error costs (false positive, $\lambda_{Negative,Positive}$, and false negative, $\lambda_{Positive,Negative}$) is presented in Figure 1. In this figure, the colors denote the optimal configuration. The vertical black line illustrates the optimal configuration changes when the false-negative error cost ranges from 0 to 100, and the value of the false-positive error cost is fixed at 35. As seen, the optimal decision can be very sensitive to changes in this parameter. The expected classification error when performing all the tests is always smaller than

performing other combinations. However, the optimal decision considers the trade-off between the expected classification error and the cost of the tests.

Another analysis is performed to test how changes in the prior probabilities $\mathbf{p}(k)$ affect the optimal decision. The results are shown in Figure 2. The parameter $p(negative)$ changes along the horizontal axis; note that $p(positive) = 1 - p(negative)$. The rest of the parameters are fixed to their values, as in the original example. It is clear that when $p(negative) = 1$ or $p(negative) = 0$, i.e., when the diagnosis is always negative or always positive, the solution is trivial: tests are not required. The number of tests increases as the entropy of the classes increases.

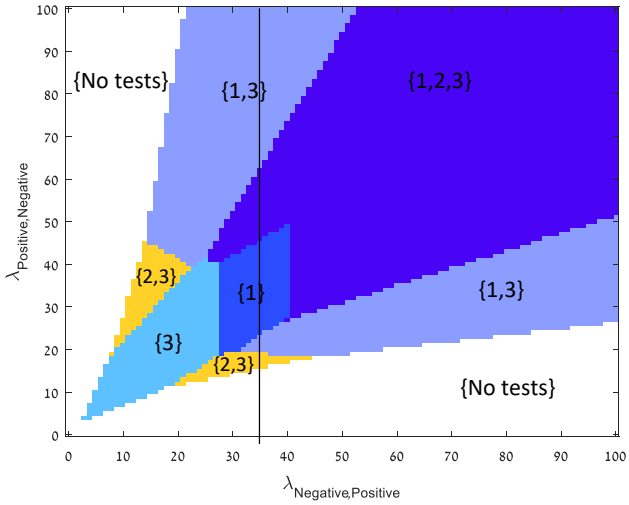

Figure 1: An optimal configuration as a function of the false positive and false negative costs.

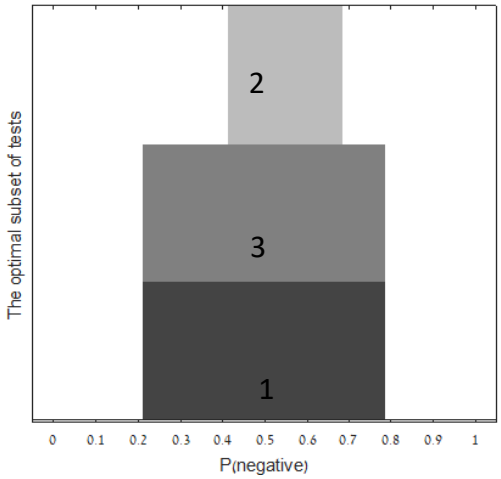

Figure 2: The optimal subset of tests (configuration) as a function of the probability of diagnoses. For example, for P(Negative)=0.3 the optimal configuration is {1,3}.

Note that this problem could be solved by optimality using an exhaustive search, which is valid when the number of tests is small. However, since the number of configurations grows exponentially with the number of tests, the problem quickly becomes computationally intractable in the number of tests. In the next two sections, we present more effective methods to solve large instances of the problem.

4 Integer linear programming formulation and solution method

This section presents the proposed ILP formulation to address the STCP and the lazy constraints generation mechanism that solves it. Following the notation above, we define two decision variables. For each test $i \in N$, let x_i be equal to "1" when the test is included in the solution and equal to "0" otherwise. For each reading $\tilde{\mathbf{r}} \in R$, we define a binary decision variable $y_{\tilde{\mathbf{r}}k}$ that indicates whether the reading is classified as $k \in K$.

Following Equation (6), the expected error cost of each reading $\tilde{\mathbf{r}}$ if it is classified as $l \in K$ is $E(\tilde{\mathbf{r}}|l) = \sum_{k \in K} \lambda_{kl} \cdot p(k|\tilde{\mathbf{r}})$. Now, the STCP can be formulated as an ILP:

$$\min \sum_{i \in N} c_i x_i + \sum_{\tilde{\mathbf{r}} \in R, l \in K} p(\tilde{\mathbf{r}}) \cdot E(\tilde{\mathbf{r}}|l) \cdot y_{\tilde{\mathbf{r}}l} \tag{10}$$

subject to

$$\sum_{i:\tilde{r}_i \neq \tilde{q}_i} x_i \geq y_{\tilde{\mathbf{r}}k} - y_{\tilde{\mathbf{q}}k} \qquad \forall (\tilde{\mathbf{r}}, \tilde{\mathbf{q}}) \in R \times R, k \in K \tag{11}$$

$$\sum_{k \in K} y_{\tilde{\mathbf{r}}k} = 1 \qquad \forall \tilde{\mathbf{r}} \in R \tag{12}$$

$$x_i \in \{0,1\} \quad \forall i \in N$$

$$y_{\tilde{\mathbf{r}}k} \in \{0,1\} \quad \forall \tilde{\mathbf{r}} \in R, k \in K \tag{13}$$

The objective function (10) minimizes the expected classification error and testing costs. The set of constraints (11) ensures that every pair of identical readings will have the same class. The set of constraints (12) ensures that each reading will be classified with a specific class. While the model requires binary values for both types of decision variables, once the value of x_is is fixed, the remaining coefficient matrix is unimodular, and thus (13) can be replaced by the nonnegativity constraint of $y_{\tilde{\mathbf{r}}k}$.

We first note that the dimension of the ILP (10)-(12), i.e., the number of decision variables $y_{\tilde{\mathbf{r}}k}$ and x_i, is equal to the number of readings multiplied by the number of classes plus the number of candidate tests. The number of constraints is quadratic in the number of readings and linear in the number of classes, $O(|R|^2|K|)$. In a typical application, we expect thousands of readings, which implies millions of constraints, whereas the number of candidate tests is typically much smaller. Accordingly, we use the *lazy constraints* scheme to solve the STCP. That is, we solve a relaxed version of the problem with only a small subset of the constraints (11), and repeatedly add violated instances of the constraint whenever an integer (super optimal) solution is obtained. The set of initial instances of constraint (11) consists of those that are related to the $(\tilde{\mathbf{r}}, \tilde{\mathbf{q}})$ reading pairs that have the smallest number of tests with different outputs, i.e., the pairs with a minimum cardinality of $\{i: \tilde{r}_i \neq \tilde{q}_i\}$. In our numerical experiment, we included in the master problem only instances of (11) where the cardinality was no more than 10% of the total number of tests (rounded to the nearest integer). Note that these constraint instances are likely to be the tightest since the sum on the left-hand side is likely to be the smallest. The rest of the constraints are added to the model only after a tentative integer solution that violates them is found.

5 Metaheuristics solution methods to the STCP

This section presents three different methods to address the STCP based on the known *Tabu Search* (TS), *Cross-Entropy* (CE), and *Binary Gravitational Search Algorithm* (BGSA) metaheuristics, all of them are considered to be of great potential to solve the STCP. The TS is a deterministic method that exploits the structure of the STCP, where potential solutions are generated based on the current solution (Glover, 1989). In contrast, the CE is a random population-based method, where the solution at each iteration is tuned according to the best solutions at the last iteration (Rubinstein, 1997). Finally, the BGSA is a relatively new stochastic search algorithm where the interaction among the solutions is modeled by physical gravitation law (Rashedi et al., 2009). We study and analyze all three methods as none of them was found to predominate the others over all the scenarios, yet the TS was shown to be more effective in most cases. Note that the STCP is often applied within an offline long-term design problem, and thus the decision-maker may wish to use all available heuristics and select the best solution obtained so far.

Finally, in terms of notation, recall that a solution to a problem is defined by the selected configuration, whereas the mapping of each signature to a class is defined by (7). We denote a solution by the characteristic vector $\mathbf{x}$ of this set, in which $x_i = 1$ if test i is included in the configuration and 0 otherwise. The value of a solution, calculated as in (9), is denoted by $g(\mathbf{x})$.

5.1 The TS method

The TS method extends the basic local search techniques to facilitate the exploration of the solution space beyond local optima. Once a local optimum is reached, the method allows one to move to a new solution even if it is inferior. The TS method uses a *tabu list* (TL) to disallow moves that cancel previous moves during several subsequent iterations in order to escape a neighborhood of locally optimal solutions.

The TS algorithm involves three main steps: (a) generate an initial solution and initialize the TL to be empty; (b) explore the current solution's neighborhood defined by a set of candidate moves, yet excluding moves listed in the TL; and (c) move to the best-explored solution and add a new entry to TL to avoid any move that can direct the search back to the previous solution. If the TL is longer than a predefined length, the algorithm removes its oldest entries. Steps (b) and (c) are repeated up to a predefined number of iterations or until some other stopping criterion is satisfied.

In our implementation, the initial solution is the empty set ($\mathbf{x} = \mathbf{0}$). Given a current solution $\mathbf{x}$, its value is evaluated for all possible readings as explained and demonstrated in Sections 2 and 3. The neighborhood of $\mathbf{x}$, $N(\mathbf{x})$ is defined by three types of moves: ADD - add one test that is not included in the current solution; REMOVE - remove one test from the current solution; and SWAP - swap a test from the current solution with a test that was not included in the solution. The set of neighboring solutions induced by each type of the moves mentioned above are denoted by $A(\mathbf{x})$, $R(\mathbf{x})$ and $S(\mathbf{x})$, respectively, thus, $N(\mathbf{x}) = A(\mathbf{x}) \cup R(\mathbf{x}) \cup S(\mathbf{x})$. Note that $|A(\mathbf{x}) + R(\mathbf{x})| = n$ and $|S(\mathbf{x})| \leq \frac{1}{4}n^2$. Each entry in the tabu list consists of one or two tests that should not be added or removed from the solution as long as the entry remains in the list. The ADD and REMOVE operations add entries with a single test, while the SWAP operation adds

an entry with a pair of tests. One is forbidden for removal and the other for appending. If a candidate move involves a test in the tabu list, then its respective solution is excluded from the neighborhood. We denote this reduced neighborhood by $N'(\mathbf{x})$. In our implementation, we use a stopping criterion based on the total number of iterations to allow a fair comparison with the other methods. However, other criteria used in the literature may apply. The main algorithm is outlined as pseudocode in Figure 3.

Initialized
Set **x,** $\mathbf{x}^*$ and TL as empty
Set $v^* = g(\mathbf{x})$
Repeat
Set $v = \infty$
For each $\mathbf{x}'$ in $N'(\mathbf{x})$
Set $v' = g(\mathbf{x}')$
If $v' < v$ then
$v = v'$
$\mathbf{y} = \mathbf{x}'$
Let m be the test(s) by which **x** and **y** differ
Set $\mathbf{x} = \mathbf{y}$
If $v < v^*$ then $\mathbf{x}^* = \mathbf{x}$ and $v^* = v$
Append *m* as an entry to the TL
If $|TL|$ is greater than the maximal tabu length, remove its first entry
Until a pre-determined number of iterations is performed

Figure 3: Pseudocode of the TS algorithm

5.2 The CE Method

The CE algorithm execute iterative steps, whereby each iteration can be broken down into three main phases: (a) generate a random population of solutions using a specified *probabilistic selection rule*; (b) evaluate the value of each of the generated solutions, and (c) update the probabilistic selection rule for the next iteration based on the best solutions (termed as the *elite set*) and iterate until some stopping criterion is satisfied.

At each iteration, we generate w solutions using a multi-Bernoulli distribution with 'success probabilities' $\mathbf{p} = (p_1, \dots, p_n)$, i.e., $\mathbf{x} = (x_1, \dots, x_n)$ such that $x_i \sim Ber(p_i)$. We initialize the probabilities with $p_i = 0.5$, for all i, and update all these probabilities at step (c) of each iteration. In a given iteration of the CE, we use $\mathbf{x}_i^{(j)}$ to denote the i^{th} test in solution j, while $\mathbf{x}^{(j)} \in \{0,1\}^n$ is a binary vector that represents solution j. The probabilities p_i are updated at the end of each iteration based on the best ρw solutions (the elite set) and subject to exponential smoothing with a weight parameter $\alpha \in [0,1]$. The parameter $\rho \in (0,1)$ defines the relative size of the elite set, while w is the number of solutions that are generated at each iteration. w and ρ are selected such that $w\rho$ is an integer. Previous studies used $\rho = 0.1$, i.e., the top ten percent of the solutions are taken as the elite set. The indices of the solutions in the elite set of iteration t are denoted by $\mathcal{E}_t$. The parameters of the multi-Bernoulli distribution are updated as follows:

$$q_{t,i} = \frac{\sum_{j \in \mathcal{E}_t} \mathbb{I}_{\{\mathbf{x}_i^{(j)}=1\}}}{\rho w}, \quad i = 1, \dots n \tag{14}$$

where $\mathbb{I}_{\{.\}}$ denotes an *indicator function* defined as follows:

$$\mathbb{I}_{\{condition\}} = \begin{cases} 1, & condition\ holds \\ 0, & otherwise \end{cases}$$

That is, $q_{t,i}$ is the proportion of the solutions that include test i in the elite set of iteration t. The following exponential smoothing formula is then used to update $\mathbf{p}_t$:

$$p_{t,i} = \alpha q_{t,i} + (1-\alpha) p_{t-1,i}, \quad i = 1, \dots n \tag{15}$$

We use exponential smoothing to prevent the premature convergence of $p_{t,i}$ to 0 or 1. It has been empirically shown, e.g., by Alon et al. 2005, that a value of α between $0.7 \leq \alpha \leq 0.9$ often obtains the best results. In this study, we use $\alpha = 0.8$.

Several types of stopping criteria have been used in the literature, such as i) stop when the worst solution in the elite set does not change for a predefined number of consecutive iterations; ii) stop when all the elements of $\mathbf{p}_t$ are close enough to 0 or 1, and thus no new solutions are likely to be generated; iii) stop after a predefined number of iterations or computation time. Combinations of the above may also apply. In our implementation, we use the third stopping criterion to enable a fair comparison with other studied heuristics given a similar computational effort.
The pseudocode that describes our CE algorithm is presented in Figure 4.

Initialized **p** such that all test probabilities are equal to **0.5**.
Set $v^* = \infty$
Repeat
  For j = 1 to w
    Generate solution $\mathbf{x}^{(j)}$ such that $x_i^{(j)} \sim Ber(p_i)$
  Sort the solutions in **x** in non-decreasing order of $g(\mathbf{x}^{(j)})$
  Let $\mathrm{x}^{[1]}, \mathrm{x}^{[2]}, \dots$ be the solutions in their sorted order
  If $g(\mathbf{x}^{([1])}) < v^*$
    $v^* = g(\mathbf{x}^{([1])})$ and $x^* = \mathbf{x}^{([1])}$
  Update **p** using (15)
Until a pre-determined number of iterations is performed
Return $\mathbf{x}^*, v^*$

Figure 4**:** Pseudocode of the CE algorithm

5.3 The Binary Gravitational Search Algorithm (BGSA) Method

The Binary Gravitational Search Algorithm (BGSA) is a relatively recent metaheuristic inspired by the Newtonian law of gravitation and motion. Solutions are represented by vectors and considered as objects (also called agents), and their position in a multidimensional space is determined by the coordinates of these vectors. The mass of each solution is determined by its objective function value. At each iteration, the

positions and velocities of the objects are updated based on their current positions, masses and velocities. The mass of each object is then updated based on the population's values of solutions. The process is repeated until all the objects are merged into one or more heavy objects or when another stopping criterion is met.

For the implementation of the BGSA for the STCP, consider an initial set of w solutions, each represented by a vector $\mathbf{x}^{(j)} \in \{0,1\}^n$ for $j = 1,2,\dots,w$; we refer to each coordinate of these vectors $x_i^{(j)}$ as the position of j^{th} solution in the i^{th} dimension. These values are updated from iteration to iteration, and we use $\mathbf{x}^{(j)}(t)$ to denote the position of the solution at iteration t of the algorithm. Let us further define

$$best(t) = \min_{j\in\{1,\dots,w\}} g\left(\mathbf{x}^{(j)}(t)\right).$$
$$worst(t) = \max_{j\in\{1,\dots,w\}} g\left(\mathbf{x}^{(j)}(t)\right).$$

Based on these values, one can calculate a normalized measure of each solution j:

$$q_j(t) = \frac{g\left(\mathbf{x}^{(j)}(t)\right) - worst(t)}{best(t) - worst(t)}.$$

Next, the mass of each solution j is updated as follows:

$$M_j(t) = \frac{q_j(t)}{\sum_{j'=1}^{w} q_{j'}(t)}. \tag{16}$$

At a specific time t, the force acting on agent j_1 from agent j_2 is defined as follows:

$$F_i^{(j_1,j_2)}(t) = G_0\left(1 - \frac{t}{T}\right)\frac{M_{j_1}(t)\cdot M_{j_2}(t)}{\sum_{i'=1}^{n}\left|x_{i'}^{(j_1)}(t) - x_{i'}^{(j_2)}\right| + \varepsilon}\left(x_i^{(j_2)}(t) - x_i^{(j_1)}\right),$$

where G_0 is a gravitational constant, T is the total number of planned iterations for the algorithm, and ε is a small positive constant. Using some preliminary experiments, we set $G_0 = 0.01T$ and $\varepsilon = 2.2 \times 10^{-16}$.

Next, we find an elite set E_t comprising the best solutions at iteration t and generate random numbers $p_j(t) \sim U[0,1]$ for each $j \in E_t$. The cardinality of the E_t is set to $\lceil \rho_t w \rceil$ where ρ_t linearly decreases from iteration to iteration according to the following formula:

$$\rho_t = 1 - \frac{t}{T}(1 - \rho_T),$$

where ρ_T is a parameter of the algorithm, while in our experiment, we used $\rho_T = 0.02$. Next, we define the force that acts on solution j in dimension i at iteration t by:

$$F_i^{(j)}(t) = \sum_{j'\in E_t\setminus\{j\}} p_{j'} F_i^{(j,j')}(t).$$

In such a way, at the initial stage, all solutions apply forces on each other, and as the iterations progress, only the few best solutions affect all the others. Now, according to the law of motion, the acceleration of a solution j at iteration t in dimension i is given by:

$$a_i^{(j)}(t) = \frac{F_i^{(j)}(t)}{M_j(t)}. \tag{17}$$

The velocity of an agent is considered as a random fraction of its current velocity added to its acceleration:

$$v_i^{(j)}(t) = \pi_j(t) \cdot v_i^{(j)}(t-1) + a_i^{(j)}(t-1), \tag{18}$$

where $v_i^{(j)}(0)$ is initialized to zero and $\pi_j(t)$ is drawn from $U[0,1]$. Moreover, to increase the chance of convergence, the velocity is limited by some parameter v_{max}. That is, $\left|v_i^{(j)}\right| < v_{max}$. We followed (Rashedi et al. 2010) and set $v_{max} = 6$. Based on the velocity in each dimension, i, we flip the position of each agent, j, between 0 and 1 with probability $\left|tanh\left(v_i^{(j)}(t)\right)\right|$ and leave it as $x_i^{(j)}(t)$ otherwise.

$$x_i^{(j)}(t+1) = \begin{cases} 1 - x_i^{(j)}(t), & with\ probability\ \left|tanh\left(v_i^{(j)}(t)\right)\right| \\ x_i^{(j)}(t), & otherwise \end{cases} \tag{19}$$

The BGSA algorithm is outlined as pseudocode in Figure 5.

Initialized
  $v^* = \infty$
  Draw initial population of w solutions $\mathbf{x}^{(j)}$, $j = 1,2,\dots,w$
Repeat
  $Set = \infty$, $worst = -\infty$
  For j = 1 to w
    Evaluate $g\left(\mathbf{x}^{(j)}\right)$
      If $g\left(\mathbf{x}^{(j)}\right) < best$
        $best = g\left(\mathbf{x}^{(j)}\right)$
      If $g\left(\mathbf{x}^{(j)}\right) > worst$
        $worst = g\left(\mathbf{x}^{(j)}\right)$
    Calculate M_j // *See (16)*
    Calculate $a^{(j)}$ // *See (17)*
    Calculate $\mathbf{v}^{(j)}$ // *See (18)*
    Update $\mathbf{x}^{(j)}$ // *See (19)*
    If $g\left(\mathbf{x}^{(j)}\right) < v^*$
      $v^* = g\left(\mathbf{x}^{(j)}\right)$ and $x^* = \mathbf{x}^{(j)}$
Until it reaches the pre-determined number of iterations, T
Return the best solution $\mathbf{x}^{(j)}$

Figure 5: Pseudocode of the BGSA algorithm

5.4 Memoization

Recall that calculating the value, $g(\mathbf{x})$, for each solution with a given configuration requires evaluating the signatures of all the related readings, which is a computationally demanding task. Indeed, almost all the running time of the three heuristics described above is spent on computing these value evaluations. In some situations, the same

solutions may be required multiple times in the same run of the algorithm. To avoid repeated calculations, we store each calculated solution's value in a hash table that is indexed by the binary representation of the solution. When the algorithms require the value of **x**, they first check if it already exists in the table. If it exists, the value is retrieved; otherwise, the solution is evaluated, and its value is stored in the table. This mechanism significantly reduces the running time of all three heuristics and is especially beneficial in the last iterations of the two randomized heuristics, namely, the CE and BGSA. In our experiments, we created a different memoization hash table for each algorithm to benchmark them fairly. However, in practice, the same hash table can be used by different algorithms in parallel or sequentially to obtain the best solution.

6 Experimental results

In the first part of this section (Subsection 6.1), we tested the proposed integer programming formulation of the problem together with the lazy constraints generation mechanism and compared it to the three heuristic methods for the STCP. In the second part (Subsection 6.2), we added a dataset with probabilistic labels to analyze the advantages of the STCP. In the third part (Subsection 6.3), we show the merits of our generalization of the STCP by comparing it to the TCP.

6.1 A comparison between the exact and heuristics methods

In this section, our integer programming formulation with lazy constraints are tested and compared with the three heuristic results. We coded the lazy constraint generation mechanism in Python 3. The linear programming relaxations were solved using an IBM CPLEX 12.10 commercial solver. All three heuristic methods were implemented in MATLAB 2018b. The testing environment was an i9-9900K Linux machine with 64 GB RAM.

For the evaluation of the algorithms presented above, we used representative datasets from the UCI Machine Learning Repository (Dua and Graff, 2019) with up to 8,124 readings and 68 tests. In some of the datasets, we removed tests and readings to eliminate missing values. Moreover, tests with numerical values were discretized by dividing their values into quintiles or quartiles (depending on the number of readings).

Using this data, we estimated the prior probabilities of the classes and the readings as well as the conditional probability of each class given a reading. Our algorithms were executed based on these estimated probabilities.

We used three tests cost vectors for each dataset: one with a fixed (unit) cost per test and two with randomly generated values as described below. Lastly, three classification error cost matrices were generated for each combination of dataset and cost vector. All cleaned and processed input data of our experiment are available online at Douek-Pinkovich's drive (2020). In total, the use case study contained 45 problem instances based on five different UCI datasets.

In Table 5, we show the number of tests, type of test values (continuous or discrete), discretization level (in the case of continuous test values), number of readings, and number of classes in the datasets. The information in the table refers to the cleaned data after removing some tests and readings to eliminate the missing values.

Table 5: Characteristics of the five datasets used in the experiments.

Dataset	*Tests*	*Test value type*	*Discretization*	*Readings*	*Classes*
Wine	12	Continuous	Quintiles	6,463	7
Thyroid	21	Continuous	Quintiles	3,103	5
Mushrooms	21	Discrete	-	8,124	2
Cortex nuclear	68	Continuous	Quartiles	1,077	8
Molecular biology	60	Discrete	-	3,190	3

The Wine Dataset from UCI contains two tables related to red and white wine samples, as described in Cortez et al. (2009). We followed Kaggle (Parmar, 2018) and used a merged version of this dataset where the type of wine was added as a new feature. The class in this dataset is the wine quality score represented by a numerical value in the range 3-9. The classification error matrix is based on the distance between the classified quality and the true quality (a Toeplitz matrix with values range 0-6). The (i, j) element of the Toeplitz matrix represents the absolute difference between the two classes. An error cost matrix proportional to the Toeplitz matrix reflects that missing the ordinal class by a greater gap is costlier than minor misses. Specifically, we created one matrix that is 20 times the Toeplitz matrix and one that is 30 times that matrix.

In the Thyroid dataset, there are five classes: four related to pathological conditions and one (negative) related to a healthy one. We created two matrices that assign a high cost to a false negative diagnosis, a low one to a false-positive diagnosis, and medium values to the misdiagnosis of a pathological condition.

In the Mushrooms Dataset, each reading should be classified as toxic or nontoxic. The classification matrices were constructed to reflect the fact that a false negative error (classifying a poisonous mushroom as an edible one) is much more expensive or dangerous than a false positive error.

In the Cortex Nuclear Dataset, the classes are described by three binary features that define the eight classes. We constructed error cost matrices based on the Hamming distance of this binary description of the class, i.e., the distance can be zero, one, two, or three. The matrices were created by multiplying these distances by 100 and by 200.

The Molecular Biology Dataset contains DNA sequences of 60 nucleotides (each nucleotide is a test, in our terminology). Each sequence belongs in one of three classes (exon-intron, intron-exon, or neither). For this dataset, we used three fixed classification error matrices with three different values (low, medium, high).

Our experiment is full factorial. That is, we tested all combinations of the three test cost vectors and three classification error cost matrices – nine runs for each of the five datasets. Fractional factorial designs for larger experiments are left for future research.

For each dataset, we created one *fixed* test cost vector and two random cost vectors that were drawn from a Normal distribution $N(1,0.1)$ and a Uniform one $U(0,2)$. The Thyroid Dataset from UCI included one test cost vector that we used in our experiment after normalizing it to make its mean equal one. In this instance, we used it instead of the cost vector with normally distributed values.

For each dataset, we created three error cost matrices based on particular dataset characteristics, while the testing cost scale parameter, β, was fixed to be one. All these error cost matrices have zeros in their diagonal and values off the diagonal, as described

in Table 6. The values of the error cost matrix in the experiments were chosen by a trial-and-error process, in order to find parameter values that do not lead to trivial solutions, such as those including all or none of the tests. Note that the values of the error matrix in Error cost 2 were obtained by multiplying Error cost matrix 1 by a constant, which is equivalent to dividing the value of β by the same constant. This setup allows examining the trade-off between the total testing and error costs, as discussed below. An exception to this is the error cost matrices of the Mushroom dataset, in which our goal was to compare cases with various magnitude of difference between the costs of false positive and false negative errors.

In a realistic setting, the true ratio between the testing and the error costs is frequently unknown to the designer. Therefore, we recommend solving the problem for multiple values of β and construct an efficacy frontier between the two components of the objectives function. The designers can then pick a testing configuration from the efficacy frontier that fits their needs.

Table 6: Description of the classification error cost matrices.

Dataset	*Error cost matrix 1*	*Error cost matrix 2*	*Error cost matrix 3*
Wine	20 × Toeplitz matrix {0,..,6}	30 × Toplitz matrix {0,..,6}	50 at each off-diagonal element
Thyroid	1500 to false negative 600 to false positive 1200 other errors	3000 to false negative 1200 to false positive 2400 other errors	1500 at each off-diagonal element
Mushrooms*	300 to false positive 500 to false negative	100 to false positive 400 to false negative	50 to false positive 700 to false negative
Cortex Nuclear	100 × $hamming\ distance$	200 × $hamming\ distance$	300 at each off-diagonal element
Molecular Biology**	30 at each off-diagonal element	60 at each off-diagonal element	120 at each off-diagonal element

* In the Mushrooms Dataset, *Error cost matrix 3* is not fixed.
** In the Molecular Biology Dataset, all error cost matrices are fixed.

We conducted some preliminary experiments to decide upon some of the parameters of the algorithm. We found that the CE works well with iterations of $20n$ solutions and typically converges before the 50^{th} iteration. We found that the best number of solutions per iteration in BGSA is not affected by the number of tests and that the algorithm works well with 100 solutions per iteration. To make a fair comparison between the CE and BGSA, we set the number of iterations in BGSA to $10n$ so that we kept the total number of evaluated solutions to approximately $1000n$ in both methods. In both cases, many solutions were sampled more than once and were retrieved from the hash table without being reevaluated. Since the TS is a much faster heuristic, we run it with a limit of 90 iterations but repeat each run three times with tabu list lengths of 0, 2, and 4, while keeping the hash table from iteration to iteration. Note that setting the tabu length to 0 is equivalent to a naïve local search. The reported solution values for the TS are the best out of the three. We note that no single alternative list length predominates the others. The solution times reported for the TS are the sums of the three runs with the different

tabu lengths. All the other tuning parameters of the heuristic methods are specified in Section 5.

We applied the three heuristic methods for each of the 5×3×3 combinations of datasets, error cost matrices, and test cost vectors. For all instances, we also computed the exact optimal solution by the lazy constraints' generation mechanism. For this method, we allocated up to 24 hours to each instance, and we were able to solve 39 out of the 45 benchmark problems. The six instances that we could not solve were all based on the Molecular Biology dataset. Table 7 presents all results for the smaller datasets (Wine, Thyroid, and Mushrooms).

For each run, the solution value is presented first, and optimal ones (obtained from the lazy constraints generation mechanism) are in boldface. Next, the two components of the solution values are listed – the expected error cost and the tests cost. In addition, the iteration number when the best solution is first found (only from the heuristic methods) and the solution times in seconds are displayed. It can be seen that we succeed in achieving the exact solution using our lazy constraints mechanism in all of these instances.

In Table 8, the same results are reported for the larger datasets, namely, Molecular Biology and Cortex Nuclear. Here, we could not achieve the exact solution using the lazy constraints mechanism for all these instances. Thus, we added a column for the lazy constraints (denoted as LC) solution that was achieved within 24 hours. The solution values in bold are the best found using the three heuristic methods or the lazy constraints generation method.

It is apparent from Tables 7 and 8 that none of the three solution methods consistently provide a better solution than the others. All instances of the three smaller datasets were solved to optimality using both CE and BGSA and the TS while missing the optimal solutions occurred in only one case out of the 27. Thus, to save space in the table, a separated column for the lazy constraints' solution values was not added. In the six instances that could not be solved to optimality within the 24-hour time limit, the best solution found was similar to the one obtained by the heuristics, but the lower bound provided by the solver was very weak (with optimality gaps of 44-59%). These results support the strength of these heuristic solution methods.

In the larger datasets, the TS provided the best solutions (or the optimal ones, when it obtained by our exact method) in 14 out of 18 instances and missed the best solution within a small margin of up to 1.5%. In these datasets, the CE and BGSA found the best solutions in 8 and 7 cases, respectively.

In terms of solution times, the heuristic methods are up to approximately 100 times faster than the implementation of the lazy constraint's solution algorithm.

We further observed that the best solutions in almost all the runs of the three heuristic methods were found in an early iteration (relative to the number of allowed iterations), which implies that with the other tuning parameters used, our stopping criteria were correct. However, it may be the case that other criteria could save computation time without sacrificing quality. Lastly, it seems that in most of the cases, the TS outperforms the two other heuristics in terms of computation time. However, since the STCP is a long-run design problem, a good practice would be to apply both all three heuristics and lazy constraints generation mechanism.

Table 7: Result summary for Wine, Thyroid, and Mushrooms datasets.

Test cost vector	Error cost matrix	Parameters	Wine			Thyroid			Mushrooms		
			TS	BGSA	CE	TS	BGSA	CE	TS	BGSA	CE
fixed	1	Solution value	**11.602**	**11.602**	**11.602**	**17.058**	**17.058**	**17.058**	**4**	**4**	**4**
		Expected error cost	4.602	4.602	4.602	7.058	7.058	7.058	0	0	0
		Tests cost	7	7	7	10	10	10	4	4	4
		# of iterations until best solution	10/90	2/120	3/50	12/90	106/210	10/50	6/90	66/210	11/50
		Solution time (sec.)	19.2	108.2	64.4	32.3	191.9	103.0	22.7	188.1	56.1
fixed	2	Solution value	**13.765**	**13.765**	**13.765**	**23.762**	**23.762**	**23.762**	**3.591**	**3.591**	**3.591**
		Expected error cost	5.765	5.765	5.765	12.762	12.762	12.762	0.591	0.591	0.591
		Tests cost	8	8	8	11	11	11	3	3	3
		# of iterations until best solution	10/90	4/120	4/50	13/90	124/210	10/50	4/90	132/210	10/50
		Solution time (sec.)	36.7	107.4	62.3	38.5	183.8	96.9	17.3	180.2	54.3
fixed	3	Solution value	**15.388**	**15.388**	**15.388**	**23.568**	**23.568**	**23.568**	**3.295**	**3.295**	**3.295**
		Expected error cost	7.388	7.388	7.388	12.568	12.568	12.568	0.295	0.295	0.295
		Tests cost	8	8	8	11	11	11	3	3	3
		# of iterations until best solution	10/90	11/120	4/50	13/90	115/210	10/50	4/90	98/210	9/50
		Solution time (sec.)	32.7	108.7	61.4	26.4	192.6	89.7	15.2	175.1	58.0
Normal*	1	Solution value	**11.691**	**11.691**	**11.691**	40.122	**39.609**	**39.609**	**4.014**	**4.014**	**4.014**
		Expected error cost	4.601	4.601	4.601	40.122	37.609	37.609	0.217	0.217	0.217
		Tests cost	7.090	7.090	7.090	0	2	2	3.797	3.797	3.797
		# of iterations until best solution	8/90	4/120	4/50	1/90	102/210	9/50	5/90	110/210	13/50
		Solution time (sec.)	25.8	105.4	60.3	2.4	60.9	22.4	23.0	178.3	78.0
Normal*	2	Solution value	**13.881**	**13.881**	**13.881**	**60.991**	**60.991**	**60.991**	**3.895**	**3.895**	**3.895**
		Expected error cost	5.765	5.765	5.765	27.071	27.071	27.071	0.098	0.098	0.098
		Tests cost	8.116	8.116	8.116	33.920	33.920	33.920	3.797	3.797	3.797
		# of iterations until best solution	9/90	1/120	4/50	13/90	129/210	13/50	5/90	119/210	13/50
		Solution time (sec.)	28.9	106.7	59.4	20.6	129.4	70.0	11.0	168.6	68.0
Normal*	3	Solution value	**15.504**	**15.504**	**15.504**	**40.122**	**40.122**	**40.122**	**3.846**	**3.846**	**3.846**
		Expected error cost	7.388	7.388	7.388	40.122	40.122	40.122	0.049	0.049	0.049
		Tests cost	8.116	8.116	8.116	0	0	0	3.797	3.797	3.797
		# of iterations until best solution	10/90	8/120	5/50	1/90	72/210	12/50	6/90	113/210	11/50
		Solution time (sec.)	17.9	110.4	62.5	3.0	55.0	21.7	10.9	174.3	68.3
Uniform	1	Solution value	**12.369**	**12.369**	**12.369**	**15.790**	**15.790**	**15.790**	**2.235**	**2.235**	**2.235**
		Expected error cost	5.471	5.471	5.471	8.121	8.121	8.121	0.492	0.492	0.492
		Tests cost	6.898	6.898	6.898	7.669	7.669	7.669	1.743	1.743	1.743
		# of iterations until best solution	8/90	1/120	4/50	12/90	131/210	11/50	6/90	128/210	10/50
		Solution time (sec.)	33.2	94.4	55.9	36.4	177.5	95.0	11.2	198.6	65.1
Uniform	2	Solution value	**15.004**	**15.004**	**15.004**	**22.159**	**22.159**	**22.159**	**2.137**	**2.137**	**2.137**
		Expected error cost	7.288	7.288	7.288	21.743	21.743	21.743	0.394	0.394	0.394
		Tests cost	7.716	7.716	7.716	0.416	0.416	0.416	1.743	1.743	1.743
		# of iterations until best solution	9/90	10/120	5/50	13/90	119/210	12/50	6/90	105/210	9/50
		Solution time (sec.)	22.7	103.7	60.0	33.5	184.5	93.0	7.2	174.6	64.5
Uniform	3	Solution value	**16.974**	**16.974**	**16.974**	**22.183**	**22.183**	**22.183**	**2.038**	**2.038**	**2.038**
		Expected error cost	7.426	7.426	7.426	21.767	21.767	21.767	0.295	0.295	0.295
		Tests cost	9.548	9.548	9.548	0.416	0.416	0.416	1.743	1.743	1.743
		# of iterations until best solution	11/90	8/120	4/50	13/90	128/210	7/50	9/90	143/210	12/50
		Solution time (sec.)	28.5	100.7	60.2	20.8	187.3	95.8	13.4	170.0	60.3

*Except in the Thyroid dataset, where the cost vector was taken from UCI

Table 8: Result summary for Cortex Nuclear and Molecular Biology datasets.

Test cost vector	Error cost matrix	Parameters	Cortex nuclear				Molecular biology			
			TS	BGSA	CE	LC	TS	BGSA	CE	LC
fixed	1	Solution value	**7.371**	7.650	8.093	**7.371**	**7.103**	7.969	7.828	
		Expected error cost	0.371	0.650	0.093	0.371	3.103	0.969	0.828	
		Tests cost	7	7	8	7	4	7	7	
		# of iterations until best solution	9/90	642/680	45/50	---	5/90	569/600	49/50	
		Solution time (sec.)	261.6	357.8	315.3	57,109	405.6	1561.0	1342.0	>day
fixed	2	Solution value	**7.743**	8.371	8.371	**7.743**	**8.282**	8.395	8.357	
		Expected error cost	0.743	0.371	0.371	0.743	0.282	0.395	0.357	
		Tests cost	7	8	8	7	8	8	8	
		# of iterations until best solution	9/90	674/680	33/50	---	23/90	521/600	46/50	
		Solution time (sec.)	264.7	357.7	320.7	56,509	1230.0	1571.0	1417.5	>day
fixed	3	Solution value	8	8.836	8.279	**7.836**	**8.564**	8.978	8.865	
		Expected error cost	0	0.836	0.279	0.836	0.564	0.978	0.865	
		Tests cost	8	8	8	7	8	8	8	
		# of iterations until best solution	10/90	650/680	42/50	---	23/90	565/600	20/50	
		Solution time (sec.)	269.3	357.4	322.8	59,740	1230.7	1562.6	1456.3	>day
Normal	1	Solution value	6.852	7.353	**6.747**	**6.747**	**6.760**	**6.760**	**6.760**	
		Expected error cost	0.464	0.372	0.464	0.464	0.903	0.903	0.903	
		Tests cost	6.388	6.981	6.283	6.283	5.857	5.857	5.857	
		# of iterations until best solution	10/90	645/680	42/50	---	29/90	558/600	41/50	
		Solution time (sec.)	169.1	354.2	312.5	10,866	528.8	1539.6	1192.8	>day
Normal	2	Solution value	**7.056**	7.293	7.458	**7.056**	**7.189**	**7.189**	**7.189**	
		Expected error cost	0.185	0.186	0.371	0.185	0.470	0.470	0.470	
		Tests cost	6.871	7.107	7.087	6.871	6.719	6.719	6.719	
		# of iterations until best solution	40/90	659/680	50/50	---	32/90	581/600	36/50	
		Solution time (sec.)	194.4	354.4	322.2	12,288	702.7	1564.3	1374.6	>day
Normal	3	Solution value	**7.087**	7.771	7.274	**7.087**	7.675	7.660	**7.652**	
		Expected error cost	0	0	0	0	0.790	0.941	0.753	
		Tests cost	7.087	7.771	7.274	7.087	6.885	6.719	6.899	
		# of iterations until best solution	86/90	645/680	49/50	---	43/90	570/600	40/50	
		Solution time (sec.)	292.3	350.4	320.5	7,151	980.4	1541.2	1396.3	>day
Uniform	1	Solution value	**2.224**	**2.224**	**2.224**	**2.224**	**1.758**	**1.758**	**1.758**	**1.758**
		Expected error cost	0.186	0.186	0.186	0.186	0.301	0.301	0.301	0.301
		Tests cost	2.038	2.038	2.038	2.038	1.457	1.457	1.457	1.457
		# of iterations until best solution	30/90	615/680	37/50	---	13/90	526/600	31/50	---
		Solution time (sec.)	169.3	341.6	277.2	178	214.1	1467.9	1082.7	921
Uniform	2	Solution value	2.307	**2.295**	2.307	**2.295**	**1.969**	**1.969**	**1.969**	**1.969**
		Expected error cost	2.094	0	2.094	0	0.151	0.151	0.151	0.151
		Tests cost	0.213	2.295	0.213	2.295	1.818	1.818	1.818	1.818
		# of iterations until best solution	21/90	609/680	39/50	---	15/90	534/600	29/50	---
		Solution time (sec.)	258.8	342.0	292.0	161	623.0	1465.9	1099.6	804
Uniform	3	Solution value	**2.295**	2.307	2.307	**2.295**	**2.119**	**2.119**	**2.119**	**2.119**
		Expected error cost	0	2.094	2.094	0	0.301	0.301	0.301	0.301
		Tests cost	2.295	0.213	0.213	2.295	1.818	1.818	1.818	1.818
		# of iterations until best solution	50/90	620/680	30/50	---	16/90	533/600	33/50	---
		Solution time (sec.)	303.4	340.0	290.6	76	690.0	1468.0	1126.5	1252

The trade-off between the total testing and error costs is demonstrated in Tables 7 and Table 8, for the obtained optimal solutions. Note that Error cost matrix 2 is a constant multiplication of Error cost matrix 1 (except for the Mushrooms dataset). Such a multiplication is equivalent to the division of β by the same factor. That is, Error cost matrix 1 represents higher β values than Error cost matrix 2. As expected, increasing the error cost (or equivalently decreasing the testing costs) would result in an optimal solution that consist of additional tests.

In Table 9, we present some aggregated statistics that measure our memorized mechanism's success for the three heuristic methods. In the first row, we present the average number of solutions that are evaluated for each of the nine instances. In the second row, we present the average number of times when the required solution could be obtained from the hash table (memoized hits), and thus, there was no need to reevaluate it. The ratio between the number of memoized hits and the total number of scanned solutions (actually evaluated and retrieved from the memoize) is presented in the third row, entitled "Frac. hits rate." In the fourth row, we give the ratio between the number of evaluated solutions and the number of all the possible ones.

It can be seen that for all the three algorithms, the hash table is beneficial, especially in instances with a small number of tests. When the number of tests grows, the hash table is effective mostly for the TS algorithm that searches in previous good solutions but not so much for the BGSA and CE. We note that for all three methods, the fraction of evaluated solutions approaches zero as the number of tests grows. Given that all three algorithms spend very most of their computation time in the evaluation of solutions, this implies that their solution time is much shorter than the time needed for complete enumeration.

Table 9: Statistics of the memorization mechanism.

Solution method	Measure	Wine	Thyroid	Mushrooms	Cortex nuclear	Molecular biology
	# Tests	12	21	21	68	60
TS	# of evaluations	558	1,685	2,329	50,047	29,062
	# of memoized hits	4,055	16,664	15,208	68,128	62,248
	Frac. hits rate	0.88	0.91	0.87	0.58	0.68
	Frac evaluated	0.14	0.0008	0.0011	1.7×10^{-16}	2.5×10^{-14}
BGSA	# of evaluations	2,376	13,101	13,989	64,136	56,346
	# of memoized hits	9,624	7,899	7,011	3,864	3,654
	Frac. hits rate	0.80	0.38	0.33	0.06	0.06
	Frac evaluated	0.58	0.0062	0.0067	2.2×10^{-16}	4.9×10^{-14}
CE	# of evaluations	1,342	5,858	6,007	63,769	52,298
	# of memoized hits	10,658	15,142	14,993	4,231	7,702
	Frac. hits rate	0.89	0.72	0.71	0.06	0.13
	Frac. evaluated	0.33	0.0028	0.0029	2.2×10^{-16}	4.5×10^{-14}

6.2 Probabilistic Dataset

In Section 6.1, the proposed solution was applied to instances of the STCP problem that are degenerated in the sense that each reading is mapped deterministically to a single class. As indicated earlier, the STCP model is relevant for such cases and differs from the GTCP model since it is capable of considering the trade-off between the classification accuracy and the testing cost. Another virtue of the STCP model is that it is capable of dealing with noisy input, i.e., when each reading indicates a distribution vector over the classes. In this section, we apply the TS on the noisy MNIST dataset of handwritten digits that obtains probabilistic input labels of the digits for each reading. This dataset was first presented in Gruber et al. (2021). It is an adaptation of the well-known deterministic version of MNIST that was introduced at LeCun et al. (2010). Moreover, the MNIST dataset has been used to analyze and reduce the images classification error in a noisy environment (e.g., Huang et al., 2015, Cheng et al., 2020) and we further discuss this point later in this section.

The MNIST dataset contains 10,000 black and white images (readings). Each image consists of 28×28 pixels, i.e., 784 pixels (that represent tests in this context). The value of each pixel is a number from 0 to 255 that represents its greyscale. Each reading is represented by ten probabilistic labels (the number of digits). For example, an image may be classified as the digit '7' with a probability of 0.8, the digit '1' with a probability of 0.15, and the digit '4' with a probability of 0.05.

Since the range of the possible pixel outputs is very large with 256 different possible levels, we consider these outputs and discretized them to quantiles representing two, four, and eight color levels. Figure 6 shows for illustration purpose the four-color levels of one of the images from the dataset.

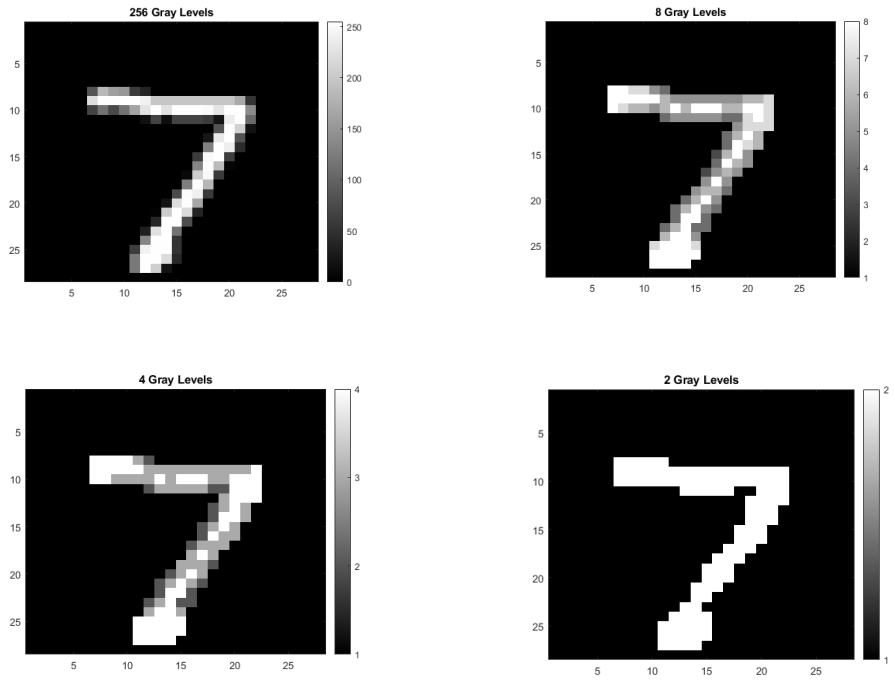

Figure 6: Four-color levels of a noisy image from the MNIST dataset.

Since the MNIST dataset is larger than the datasets that were examined in Section 6.1, we applied only the TS heuristic that was shown to be the fastest (and efficient) heuristic. We used a single test cost vector with a fixed (unit) cost per pixel and four classification error cost matrices with 900, 1800, 3600, and infinity in each of the off-diagonal elements. Since the solution is determined by the ratio between the error cost and the testing cost, using infinity error cost is equivalent to setting the testing cost as zero, which implies using all the pixels to identify the digits. The processed input data for our experiment is available online at Douek-Pinkovich's drive (2020).

Table 10 presents all results for the three-color discretization levels of the noisy MNIST dataset. For each solution, the error probability is presented first. Next, the number of tests (pixels) that minimizes the expected cost. In addition, the table shows the iteration number in which the best solution is first found and the solution times in seconds. It can be seen that although the noisy MNIST dataset is significantly large, the proposed method can still find solutions in a reasonable time for a design problem that needs to be solved once in a lifetime of the system.

Table 10 shows that the TS significantly reduces the number of pixels (tests) needed. For each color level, the error probabilities decrease slightly when the input cost of the errors increases. The smallest error probability is obtained with infinity error cost and naturally requires using all the 784 tests. This is clearly the lower bound on the obtainable error probability. However, one can observe that it is only slightly smaller than the error probability that can be obtained when using a significantly smaller number of tests. Expectedly, as the grayscale levels of the images decrease, more pixels are required to identify the digits since the images become more distorted.

As Gruber et al. (2021) kindly made their data available to us, in an additional experiment, we reduced the noise over their MNIST dataset, by using distribution over the classes that are closer to the undistorted ones. In this case, the lower bound on the obtainable error probability is less than 0.1, and the TS error probability (with an error cost matrix of 900 on the two-color levels data) is 0.118. Thus, this experiment shows that relying on 20 pixels is enough for a reduced error probability settings, that is close to the lower bound.

As indicated above, note that reducing the images' classification error in a noisy environment by properly analyzing a limited number of pixels can be related to the growing field of adversarial learning. In particular, it was found that despite the fact that Deep Neural Networks (DNN) are significantly good classifiers, these models are not typically robust (Szegedy et al., 2013). That is, by introducing a small perturbation to the model input, the model classification could change significantly. It has been shown that an accurate DNN model can be fooled into misclassifying typical data points by introducing a human-indistinguishable perturbation of the original inputs. Therefore, securing a correct classification by protecting a limited number of pixels (tests in our case) can be a relevant application to the proposed STCP in future research. Such a direction could also include the analysis of fewer inputs on Generative Adversarial Network (GAN) in the field of computer vision for image synthesis. Related research in this direction includes, for example, Cheng et al. (2020) that analyzed different variants of GAN for image generation on the MNIST dataset and evaluated results based on classification accuracy, as well as Huang et al. (2015) that proposed a learning method that attempts to minimize misclassification errors against the adversary.

Table 10: Result Summary for the noisy MNIST dataset

Error cost matrix	Parameters	TS		
		2 color-levels	4 color-levels	8 color-levels
900	Error probability	0.1751	0.1742	0.1736
	Number of tests	19	11	10
	# of iterations until the best solution	90/90	40/90	17/90
	Solution time (sec.)	80,136	157,300	80,171
1800	Error probability	0.1738	0.173**5**	0.1736
	Number of tests	21	12	10
	# of iterations until the best solution	59/90	17/90	17/90
	Solution time (sec.)	105,260	168,490	83,068
66/90	Error probability	0.1735	0.1735	0.1737
	Number of tests	22	12	10
	# of iterations until the best solution	79/90	17/90	66/90
	Solution time (sec.)	110,860	189,400	149,840
∞	Error probability (Lower Bound)	0.1734	0.1734	0.1734

6.3 A comparison between the STCP and the TCP

This subsection addresses one of the merits of the STCP model by comparing its configurations to the ones obtained from the GTCP model. Recall that the GTCP does not allow classification errors and thus, focuses on problem instances that can (with a sufficient number of tests) be classified deterministically. Therefore, each reading is mapped to a unique class with probability 1. Yet, even for such instances, it may be desired to allow a small probability of errors because often it is cheaper to absorb the error cost and save on the testing costs.

For a comparison between the two models, we adapted the datasets used in Douek-Pinkovich et al. (2020) that are available in the UCI repository. We set the cost of each sensor (test) to 1, so the value of the solution of the GTCP is the minimal number of sensors that are sufficient to map the readings deterministically to their classes. For the input of the STCP, we used the same sensor costs. In addition, we created three error cost matrices denoted by $\Lambda^{low}, \Lambda^{med}, \Lambda^{high}$. The elements of these matrices have fixed positive values except the diagonal, where the elements are zeros. The fixed values of the positive elements are selected based on the other parameters of the instances.

For each dataset, we estimated the prior probabilities of the classes and the readings based on their frequency in the input data. We solved the STCP using the exact method presented in this paper and the GTCP instances using an exact method presented in Douek-Pinkovich et al. (2020). The experiment consists of 21 problem instances based on seven different UCI datasets with three different error cost matrices each. We limited the running time of our algorithm to 24 hours for each instance. If the algorithm did not converge within this period, we report the best-found solution and mark the instance with an asterisk (*).

Let us note that in the experimental setting presented above, the solutions of the GTCP are always feasible for the STCP and have the same objective function value. Therefore, the optimal solution values of the GTCP can be considered as upper bounds to the optimal solution of the STCP.

In Table 11, we present the result of this experiment. A summary of the datasets is presented in the four first columns. In the next column, we present the solution value of the GTCP, which is, in this case, the number of selected tests. Columns 4-6 of the table show the fixed positive values in $\Lambda^{low}, \Lambda^{med}, \Lambda^{high}$. In the last three columns, we present the testing cost and error probability under each of the three error cost matrices of the STCP model. Note that the error probability in our setting can be obtained as the ratio between the expected classification error cost of the solution and the value of the positive elements in the error cost matrices.

As expected, when the values of the fixed positive elements of the error cost matrix are high enough, the solutions of the STCP and GTCP coincide because the optimal solution is to avoid classification errors completely. This outcome can be achieved by selecting the same set of tests as in the GTCP. In fact, we selected the positive fixed values of Λ^{high} to be high enough to assure such an outcome. For lower error costs, it is more profitable to absorb some errors with small probabilities and to save on the tests. When the positive fixed values of Λ are low enough, it is optimal to use no tests at all. In this case, there is only one (empty) signature that is mapped to the class with the highest prior probability, and the error costs for all other classes are incurred. One can view the solutions for different error cost matrices as points on the efficiency frontier of a bi-objective problem where the goal is to minimize the testing cost and the expected error cost. Note that in the letter recognition instances with Λ^{low} and Λ^{med}, optimal solutions could not be achieved within the 24-hour time limit, and thus we report there on the best-found solutions.

Table 11: A comparison between the STCP with three levels of error costs and the TCP

Dataset summary				GTCP testing cost (number of tests)	Error costs matrices (positive elements)			STCP solution testing cost (error probability)		
Name	Tests	Readings	Classes		Λ^{low}	Λ^{med}	Λ^{high}	Λ^{low}	Λ^{med}	Λ^{high}
Monk 1	6	432	2	3	3	6	9	0 (.5000)	1 (.2500)	3 (0)
Monk 2	6	432	2	6	9	18	27	0 (.3288)	0 (.3288)	6 (0)
Monk 3	6	432	2	3	5	25	45	1 (.1940)	2 (.0276)	3 (0)
Zoo	16	101	7	5	20	80	120	3 (.0495)	4 (.0099)	5 (0)
Tic-tac-toe	9	958	2	8	20	120	240	0 (.3465)	7 (.0042)	8 (0)
Chess	36	3,196	2	29	3000	6000	9000	24 (.0009)	25 (.0006)	29 (0)
Mushrooms	21	8,124	2	4	150	300	450	2 (.0059)	2 (.0059)	4 (0)
Letter recognition	16	20,000	26	11	300	900	18000	8 (.0017)*	9 (.0004)*	11 (0)

*The reported solutions are the best found within the 24-hour time limit.

The results presented in Table 11 demonstrate the advantage of using the STCP model even when the reading can be mapped deterministically to classes. The model allows the planner to consider and address the trade-off between testing cost and the risk of errors rather than always choosing the expensive (and not necessarily feasible) alternative of avoiding classification errors at all costs.

7 Conclusions

This paper introduces a practical problem whereby decisions about the test configuration must be made at the design phase of various systems and processes. The goal is to minimize the sum of the expected error cost associated with classification errors and the testing cost. The STCP is a generalization of the deterministic variant of the minimum TCP as well as the GTCP; both are known to be intractable.

We present an exact solution method for the STCP based on an ILP with a large number of constraints that can be added as lazy constraints. The applicability of the solution is demonstrated using some instances adapted from the UCI repository. While this paper presents the first successful exact solution method for STCP, it is still not capable of consistently solving large instances with reasonable computational resources. Therefore, we also present three heuristic methods. Our numerical experiments show that the three proposed heuristic solution methods are all effective. Specifically, for a given budget of computational effort, the TS method appears to be superior to the CE and BGSA. However, since none of these methods consistently predominates the others, when computational resources are available, all the three methods as well as the lazy constraints should be applied if possible. We believe that this situation is common since the STCP model can be typically implemented within an offline long-term design problem.

The STCP is related to the well-studied feature selection problem, and it may be the case that similar solution methods can be applied to it. However, solutions to the feature selection problem should be evaluated jointly using a classification method to examine their predictive power and control their sensitivity to overfitting and underfitting effects, which is a different objective than that of the STCP and left for future research.

Acknowledgment: The first author of this paper was partially supported by a scholarship from the Shlomo-Shmeltzer Institute. The research was partially supported by the Koret's Digital Living 2030 Grant.

References

Alon, G., Kroese, D. P., Raviv, T., & Rubinstein, R. Y. (2005). Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment. Annals of *Operations Research*, *134*(1), 137-151.

Bacher, M., & Ben-Gal, I. (2017). Ensemble-Bayesian SPC: Multi-mode process monitoring for novelty detection. *IISE Transactions*, 49(11), 1014-1030.

Bertolazzi, P., Felici, G., Festa, P., Fiscon, G., & Weitschek, E. (2016). Integer programming models for feature selection: New extensions and a randomized solution algorithm. *European Journal of Operational Research*, *250*(2), 389-399.

Cangalovic, M. M., Kovacevic-Vujcic, V. V., Ivanovic, L., Drazic, M., & Asic, M. D. (1996). Tabu search: A brief survey and some real-life applications. *Yugoslav journal of operations research*, *6*(1), 5-18.

Cheng, K., Tahir, R., Eric, L. K., & Li, M. (2020). An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset. *Multimedia Tools and Applications*, *79*(19), 13725-13752.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. In *Decision Support Systems*, Elsevier, 47(4):547-553.

De Boer, P. T., Kroese, D. P., Mannor, S., & Rubinstein, R. Y. (2005). A tutorial on the cross-entropy method. *Annals of operations research*, *134*(1), 19-67.

De Bontridder, K. M., Halldórsson, B. V., Halldórsson, M. M., Hurkens, C. A., Lenstra, J. K., Ravi, R., & Stougie, L. (2003). Approximation algorithms for the test cover problem. *Mathematical Programming*, *98*(1-3), 477-491.

Douek-Pinkovich, Y., Ben-Gal, I., and Raviv, T. (2020). The Generalized Test Collection Problem, TOP, Springer. https://doi.org/10.1007/s11750-020-00554-1

Douek-Pinkovich's drive (2020). STCP: Cleaned and Processed Input Data. STCP: Input data from Yifat's drive

Drezner, Z., Marcoulides, G. A., & Hoven Stohs, M. (2001). Financial applications of a tabu search variable selection model. *Journal of Applied Mathematics and Decision Sciences*, *5*(4), 215-234.

Dua, D., and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

Duman, S., Güvenç, U., Sönmez, Y., & Yörükeren, N. (2012). Optimal power flow using gravitational search algorithm. *Energy Conversion and Management*, *59*, 86-95.

Garey, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness.

Glover, F. (1989). Tabu search—part I. *ORSA Journal on computing*, *1*(3), 190-206.

Gruber, N., Ben-Gal, I., Steinberg, D. M. (2021). Supervised active learning algorithm for sequential informative sampling. Unpublished manuscript.

Halldórsson, B. V., Halldórsson, M. M., & Ravi, R. (2001, August). On the approximability of the minimum test collection problem. In *European Symposium on Algorithms* (pp. 158-169). Springer, Berlin, Heidelberg.

Hovland, G. E., & McCarragher, B. J. (1997, April). Dynamic sensor selection for robotic systems. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on* (Vol. 1, pp. 272-277). IEEE.

Huang, R., Xu, B., Schuurmans, D., & Szepesvári, C. (2015). Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database, ATTLabs [Online]. Available: http://yann.lecun.com/exdb/mnist, vol. 2.

Kammer, D. C. (1991). Sensor placement for on-orbit modal identification and correlation of large space structures. *Journal of Guidance, Control, and Dynamics*, *14*(2), 251-259.

Pacheco, J., Casado, S., & Núñez, L. (2009). A variable selection method based on Tabu search for logistic regression models. *European Journal of Operational Research*, *199*(2), 506-511.

Papa, J. P., Pagnin, A., Schellini, S. A., Spadotto, A., Guido, R. C., Ponti, M., ... & Falcão, A. X. (2011, May). Feature selection through gravitational search algorithm. In *Acoustics, Speech, and Signal Processing (ICASSP), 2011 IEEE International Conference on* (pp. 2052-2055). IEEE.

Parmar, R. (2018). Wine quality. https://www.kaggle.com/rajyellow46/wine-quality#winequalityN.csv.

Rashedi, E., Nezamabadi-Pour, H., & Saryazdi, S. (2009). GSA: a gravitational search algorithm. *Information sciences*, *179*(13), 2232-2248.

Rashedi, E., Nezamabadi-Pour, H., & Saryazdi, S. (2010). BGSA: binary gravitational search algorithm. *Natural Computing*, *9*(3), 727-745.

Rubinstein, R. Y. (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research*, *99*(1), 89-112.

Rubinstein, R.Y. and D.P. Kroese. (2004). The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning. Springer-Verlag, New York

Sela Perelman, L. S., Abbas, W., Koutsoukos, X., & Amin, S. (2016). Sensor placement for fault location identification in water networks: A minimum test cover approach. *Automatica*, *72*, 166-176.

Slijepcevic, S., & Potkonjak, M. (2001). Power efficient organization of wireless sensor networks. In *Communications, 2001. ICC 2001. IEEE International Conference on* (Vol. 2, pp. 472-476). IEEE.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Wendt, J. B., & Potkonjak, M. (2011, October). Medical diagnostic-based sensor selection. In *Sensors, 2011 IEEE* (pp. 1507-1510). IEEE.