ORIGINAL ARTICLE

Networks

Finding Minimum Label Spanning Trees using Cross-Entropy Method

Radislav Vaisman[1*]

[1]School of Mathematics and Physics, The University of Queensland, Brisbane 4072, Australia

Correspondence
School of Mathematics and Physics, The University of Queensland, Brisbane 4072, Australia
Email: r.vaisman@uq.edu.au

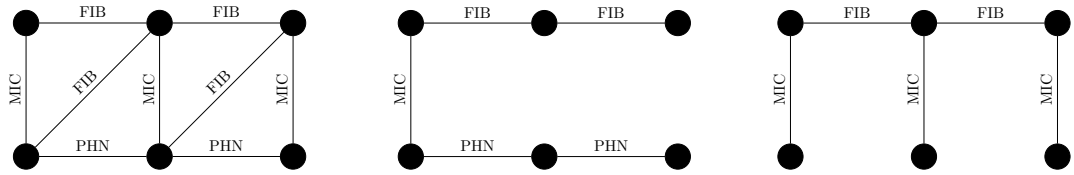
Funding information
This research was supported by the Australian Research Council Centre of Excellence for Mathematical & Statistical Frontiers, under CE140100049 grant number.

Obtaining high-quality solutions to the minimum label spanning tree problem is of crucial importance to efficient communication network design, since such solutions can reduce both the construction cost and the complexity of the ultimate architecture. However, the corresponding optimization task was shown to be hard even for complete graphs. As a consequence, no computationally efficient method for solving this problem exactly in a reasonable time is known to exist and one has to rely on approximation techniques such as heuristic and evolutionary algorithms. In this study, we investigate the performance of a different method called the Cross-Entropy algorithm which relies on rigorous developments in the fields of information theory and stochastic simulation. Our findings indicate that the mathematical soundness of the Cross-Entropy method, makes it very reliable and robust as compared to its counterparts. In particular, the obtained results suggest that the Cross-Entropy method is not sensitive to different graph models and that the proposed algorithm can obtain optimal or near-optimal solutions while using a reasonable computational effort.

KEYWORDS
Communication network design, data compression, minimum label spanning tree, Cross-Entropy, evolutionary optimization.

1 | INTRODUCTION

Given a finite undirected graph $G = (V, E, l)$ with a vertex set V, an edge set E, and a labeling function $l : E \to \mathcal{L}$, where $\mathcal{L} = \{1, \ldots, k\}$ is a finite set of labels, the *minimum label(ing) spanning tree* (MLST) problem seeks to find a spanning tree of G which can be constructed using a minimal number of *different* labels. The MLST problem appears in several important practical applications such as data compression, and communication network design [5, 10, 19, 30]. For example, Figure 1(a) depicts a communication network with optical fiber channels (FIB), telephone lines (PHN), and microwave links (MIC). In this case, as shown in Figure 1(c), a minimal label spanning tree can be constructed using only FIB and MIC links.



(a) Full communication network. (b) Regular spanning tree. (c) Minimum label spanning tree.

FIGURE 1 Panel (a) shows a communication network with optical fiber channels, telephone lines, and microwave communication links. A regular spanning tree and a minimum label spanning tree of the network in panel (a) are depicted in panels (b) and (c), respectively. The spanning tree in panel (b) uses three different channel types, namely, optical fiber channels, telephone lines, and microwave links. On the other hand, the minimum label spanning tree in panel (c), uses only two different channel types, specifically, optical fiber channels and microwave links.

Chang and Leu proved that the MLST problem belongs to the NP-hard complexity class [7]. The latter has necessitated an introduction of several heuristic and evolutionary methods. While some of these methods were shown to have rigorous theoretical performance guarantees [3, 7, 17, 19], our study indicates that they might fail in practice. That is, the convergence time of these algorithms can be prohibitively large. Specifically, we show that the corresponding performance depends on the graph structure under consideration. To resolve this problem, we introduce a different approach called the Cross-Entropy (CE) method which is both theoretically sound [26], and appears to be less sensitive to different types of graphs. In particular, our experimental study indicates that CE always manages to obtain optimal or near-optimal solutions regardless of the network's characteristics.

This work focuses on the CE method, which to the best of our knowledge, was not applied to the MLST problem yet. The CE method is motivated by the idea of Kullback-Leibler divergence (relative entropy) minimization [21]. Originally, the method was proposed by Rubinstein [24] in the context of rare-event probability estimation. However, it was shown that many optimization tasks can be converted into the rare-event estimation problem setting [25, 26]. As a result, it is possible to apply the CE method as an optimization algorithm.

To ensure a fair comparison, we examine four different approaches towards the solution of the MLST problem. In particular, we consider the following methods: the *maximum vertex covering algorithm* (MVCA), the (1+1) *Evolutionary Algorithm* ((1+1) EA), the *Global Simple Evolutionary Multiobjective Optimizer* (GSEMO), and the Genetic algorithm (GA) [3, 7, 17–19, 29]. We choose these methods for the comparison with the CE algorithm, since they were shown to both have rigorous performance guarantee, and exhibit a good practical performance [19, 30]. Specifically, MVCA has an $H_k = \sum_{i=1}^{k} i^{-1}$ approximation factor guarantee [30]. In addition, for the special case of the MLST problem in which each label appears at most b times, both the (1+1) EA and the GSEMO algorithms have a $2^{-1}(b+1)$ approximation

ratio which is achieved in expected polynomial runtime with respect to $|V|$ and k. Moreover, GSEMO was shown to have a $2 \ln |V| + 1$ approximation factor guarantee for the general MLST problem [19]. Finally, the GA algorithm was chosen because it is one of the most popular global evolutionary optimization methods.

It is important to note that the MLST problem attracted a significant amount of research. To name a few, the following methodologies were applied to the MLST problem. The *Ant Colony Optimization* method [8], the *integer programming* [9, 16], and the mixed integer linear formulation approach of Captivo et. al. [4]. In addition, a range of powerful metaheuristic algorithms that are mostly based on sophisticated local search procedures, were introduced by Consoli et. al. [11–14], and Silva et. al. [27]. Among these methods, a very interesting idea of using the so-called *complementary space* and the corresponding *variable neighborhood search* is due to Consoli et. al. [13]. For a comparison of different algorithms, we refer to Gentili et. al. [6].

The rest of the paper is organized as follows. In Section 2 we give a brief overview of the MVCA, the (1+1) EA, the GSEMO, and the specifically designed GA method. In Section 3 we explain how the CE method can be used for obtaining solutions to the MLST problem and provide the necessary mathematical reasoning. In Section 4 we compare the performance of these algorithms and report our experimental findings. In order to ensure the reproducibility of the obtained results and to allow researchers and network design practitioners to benefit from this research, we also provide a freely available research software package. Our benchmark study indicates that the CE method compares favorably with its counterparts and introduces a less sensitive behavior when applied to different types of input graphs. Finally, in Section 5, we summarize our findings and discuss possible directions for future research.

2 | THE MVCA, THE (1+1) EA, THE GSEMO, AND THE GA ALGORITHMS

In this section we provide a brief overview of the MVCA, the (1+1) EA, the GSEMO, and the GA algorithms [3, 7, 17, 19, 29]. The MVCA method for MLST problems is summarized in Algorithm 1.

Algorithm 1: The MVCA algorithm

Input: A graph $G = (V, E, l)$, where V is a set of vertices, E is a set of edges, and $l : E \to \mathcal{L} = \{1, \ldots, k\}$ is a labeling function.

Output: Minimum label spanning tree of G.

$\mathcal{A} \leftarrow \emptyset$ /* a set of labels to be used in the spanning tree construction */

Set $E' \leftarrow \{e \in E \mid l(e) \in \mathcal{A}\}$ and $G' \leftarrow (V, E')$ /* note that $E' = \emptyset$ */

while *G' is not connected* **do**

  $\mathcal{A}' \leftarrow \mathcal{L} \setminus \mathcal{A}$ /* determine the set of unused labels */

  foreach label $\in \mathcal{A}'$ **do**

    /* consider a sub-graph of G with edge labels from the $\mathcal{A} \cup \{\text{label}\}$ set */

    $E'_{\text{label}} \leftarrow E' \cup \{e \in E \mid l(e) = \text{label}\}$

    Set $G'_{\text{label}} \leftarrow (V, E'_{\text{label}})$ and $c_{\text{label}} \leftarrow$ the number of connected components in G'_{label}

  end

  /* find a label for which the number of connected components is minimized, and add it to the set $\mathcal{A}$ */

  Set $\text{label}^* \leftarrow \arg\min_{\text{label}} \{c_{\text{label}} : \text{label} \in \mathcal{A}'\}$ and $\mathcal{A} \leftarrow \mathcal{A} \cup \{\text{label}^*\}$

end

return A spanning tree of $G' = (V, E')$, where $E' = \{e \in E \mid l(e) \in \mathcal{A}\}$.

While MVCA is a relatively simple greedy algorithm which starts with $G' = (V, \varnothing)$ and sequentially adds labels that minimize the number of connected components in G', it was shown to be a very successful method that has a proven approximation factor $\left(H_k = \sum_{i=1}^{k} i^{-1}\right)$ guarantee [30]. Note that as soon as the graph $G' = (V, E')$ where $E' = \{e \in E \mid l(e) \in \mathcal{A}\}$ is available (line 11 in Algorithm 1), one can use any minimum spanning tree algorithm (such as Kruskal's algorithm [15]), to deliver the desired MLST.

Next, we examine the (1+1) EA algorithm [19]. This method is included in the numerical comparison study, since it was shown that it compares favorably with other evolutionary algorithms in terms of fitness function distribution at a given iteration, and with respect to an average optimization time [3]. The (1+1) EA method for MLST problems is summarized in Algorithm 2.

Algorithm 2: The (1+1) EA algorithm

Input: A graph $G = (V, E, l)$, where V is a set of vertices, E is a set of edges, and $l : E \to \mathcal{L} = \{1, \ldots, k\}$ is a labeling function.

Output: Minimum label spanning tree of G.

1 Initialize a binary vector $\mathbf{X} = (X_1, \ldots, X_k)$ uniformly at random.

`/* for` $1 \le i \le k$, $X_i = 1$ `stands for the fact that label` i `participates in the spanning tree construction, and` $X_i = 0$ `otherwise */`

2 **while** *termination criterion is not fulfilled* **do**

3 Obtain $\mathbf{Y}$ from $\mathbf{X}$ by flipping each X_i in $\mathbf{X}$ with probability $1/k$.

4 **if** $S(\mathbf{Y}) < S(\mathbf{X})$ **then**

`/* ` $S : \{0,1\}^k \to \mathbb{R}$ `is a fitness function defined in (1) */`

5 $\mathbf{X} \leftarrow \mathbf{Y}$.

6 **end**

7 **end**

8 **return** A spanning tree of $G' = (V, E')$, where $E' = \{e \in E \mid X_{l(e)} = 1\}$.

In order to complete the description of the (1+1) EA algorithm, we need to specify the termination criterion and the fitness function in lines 2 and 4 of Algorithm 2, respectively. The termination criterion in line 2 depends on the user preferences. For example, one can decide on a certain number of iterations of the *while* loop (lines 2 – 7) or specify a predefined runtime threshold for the algorithm execution. We define the fitness function $S : \{0,1\}^k \to \mathbb{R}$ (see line 4 of Algorithm 2), via

$$S(\mathbf{X}) = (\mathfrak{c}(\mathbf{X}) - 1) \times k \ln(k) + |\mathbf{X}| \quad \text{for } k \ge 3, \tag{1}$$

where $\mathfrak{c}(\mathbf{X})$ is the number of connected components in G', ($G' = (V, E')$, $E' = \{e \in E \mid X_{l(e)} = 1\}$), $\ln(k)$ stands for the natural logarithm of k, and $|\mathbf{X}| = \sum_{i=1}^{k} X_i$ is the total number of labels used in the spanning tree construction. Note that for any optimal solution $\mathbf{X}^*$, it holds that $\mathfrak{c}(\mathbf{X}^*) = 1$. The definition of $S(\mathbf{X})$ in (1) is inspired by the study of Lai et. al. [19]. In order to obtain an optimal solution to the MLST problem, (1) should be minimized. To see that an optimal solution is attained, suppose that an optimal MLST solution X^* uses $m \le k$ labels, that is, $|X^*| = m \le k$. Since $\mathfrak{c}(\mathbf{X}^*) = 1$, we have that $S(\mathbf{X}^*) = (1-1) \times k \ln(k) + |\mathbf{X}^*| = m$. For any non-optimal $\mathbf{X}$, the are two possibilities. If $\mathfrak{c}(\mathbf{X}) = 1$, then $|X| > m$, and thus $S(\mathbf{X}) = 0 + |X| > m = S(\mathbf{X}^*)$. Otherwise, provided that $\mathfrak{c}(\mathbf{X}) > 1$, we also have that $S(\mathbf{X}) \ge (2-1)k \ln(k) > k \ge m = S(\mathbf{X}^*)$ when $k \ge 3$. The above discussion implies that the $(\mathfrak{c}(\mathbf{X}) - 1)$ term should be

multiplied by a number which is greater than k. For example, Lai et. al. [19] use the k^2 multiplier. In this manuscript, we utilize the $k\ \ln(k)$ term, which satisfies $k\ \ln(k) > k$ for $k \geq 3$. Finally, note that for $k < 3$, the corresponding MLST problem is easy in the sense that it can be solved using an exhaustive search.

We turn our attention to the GSEMO algorithm for multiobjective optimization [19]. GSEMO is a powerful method whose performance was investigated for various problems such as covering [17], spanning trees [22], and *pseudo-Boolean* functions [20]. As mentioned in the introduction section, GSEMO achieves the $2\ln|V|+1$ approximation ratio for the MLST problem in expected polynomial runtime with respect to $|V|$ and k. In addition, it was shown that the GSEMO algorithm outperforms other local search procedures on several classes of graphs [19]. The GSEMO method for MLST problems is summarized in Algorithm 3.

Algorithm 3: The GSEMO algorithm

Input: A graph $G = (V, E, I)$, where V is a set of vertices, E is a set of edges, and $I : E \rightarrow \mathcal{L} = \{1, \ldots, k\}$ is a labeling function.

Output: Minimum label spanning tree of G.

1 Initialize a binary vector $\mathbf{X} = (X_1, \ldots, X_k)$ uniformly at random. `/* similar to line 1 in Algorithm 2 */`

2 $P \leftarrow \{\mathbf{X}\}$

3 **while** *termination criterion is not fulfilled* **do**

4 Choose $\mathbf{X}$ from P uniformly at random; that is, every $\mathbf{X} \in P$ is selected with probability $\frac{1}{|P|}$, where $|P|$ stands for the cardinality of the set P.

5 Obtain $\mathbf{Y}$ from $\mathbf{X}$ by flipping each X_i in $\mathbf{X}$ with probability $1/k$.

6 **if** $\mathbf{Y}$ *is not dominated by all* $\mathbf{X} \in P$ **then**

`/* the domination concept is defined bellow */`

7 $Q \leftarrow \{\mathbf{X} \in P \mid \mathbf{Y}$ dominates $\mathbf{X}\}$

8 $P \leftarrow P \cup \{\mathbf{Y}\} \setminus Q$

9 **end**

10 **end**

11 Choose $\mathbf{X} = (X_1, \ldots, X_k)$ from P uniformly at random.

12 **return** A spanning tree of $G' = (V, E')$, where $E' = \{e \in E \mid X_{I(e)} = 1\}$.

Similar to Algorithm 2, the termination criterion in line 3 of Algorithm 3 depends on the user preferences such as the desired number of iteration or the runtime threshold. The *domination* of one solution over the other is determined as follows. Let us suppose that the multiobjective fitness of a solution $\mathbf{X}$ is defined via a tuple $(\mathfrak{c}(\mathbf{X}), |\mathbf{X}|)$, where $\mathfrak{c}(\mathbf{X})$ is the number of connected components in $G' = (V, E')$, $E' = \{e \in E \mid X_{I(e)} = 1\}$, and $|\mathbf{X}| = \sum_{i=1}^{k} X_i$. Then, we say that $\mathbf{X}$ dominates $\mathbf{Y}$ if it holds that: $\mathfrak{c}(\mathbf{X}) < \mathfrak{c}(\mathbf{Y})$ and $|\mathbf{X}| \leq |\mathbf{Y}|$, or $\mathfrak{c}(\mathbf{X}) \leq \mathfrak{c}(\mathbf{Y})$ and $|\mathbf{X}| < |\mathbf{Y}|$.

Finally, we give a brief overview of the GA algorithm that was specifically designed for the MLST problem [29]. The GA method for MLST problems is summarized in Algorithm 4. The crossover operation in line 5 of Algorithm 4 is performed as follows. Given two feasible solutions $\mathbf{X}_1$ and $\mathbf{X}_2$, we create a set of all labels that appear in both $\mathbf{X}_1$ and $\mathbf{X}_2$. Then, the labels are sorted in the decreasing order with respect to the frequency of their appearance in G, to create an ordered set of labels. Finally, a new feasible solution $\mathbf{X}'$ is constructed by sequentially choosing labels from this ordered set (labels with higher frequency are chosen first). Now, we proceed with the mutation operation of a feasible solution $\mathbf{X}'$ in line 6 of Algorithm 4. First, we choose a label that is not in $\mathbf{X}'$ and add it to the $\mathbf{X}'$. Then,

Algorithm 4: The GA algorithm

Input: A graph $G = (V, E, l)$, where V is a set of vertices, E is a set of edges, and $l : E \to \mathcal{L} = \{1, \ldots, k\}$ is a labeling function, and the GA population size N.

Output: Minimum label spanning tree of G.

Initialize a population $\mathcal{P} = (\mathbf{X}_0, \ldots, \mathbf{X}_{N-1})$, where each $\mathbf{X} \in \mathcal{P}$ is a binary vector $\mathbf{X} = (X_1, \ldots, X_k)$.
`/* we assume that for all` $\mathbf{X} \in \mathcal{P}$, `X is a feasible solution to the MLST problem */`
$t \leftarrow 1$
while *termination criterion is not fulfilled* **do**
  for $j = 0, \ldots, N-1$ **do**
    $\mathbf{X}'_j \leftarrow \text{crossover}(\mathbf{X}_j, \mathbf{X}_{(j+t) \bmod N})$
    $\mathbf{X}''_j \leftarrow \text{mutation}(\mathbf{X}'_j)$
    if $S(\mathbf{X}''_j) \leq S(\mathbf{X}_j)$ **then**
      `/*` $S : \{0,1\}^k \to \mathbb{R}$ `is a fitness function defined in (1) */`
      $\mathbf{X}_j \leftarrow \mathbf{X}''_j$
    end
  end
  $t \leftarrow t + 1$
end
Choose $\mathbf{X} = (X_1, \ldots, X_k)$ from $\mathcal{P}$ such that $S(\mathbf{X})$ is minimal.
return A spanning tree of $G' = (V, E')$, where $E' = \{e \in E \mid X_{l(e)} = 1\}$.

the labels of $\mathbf{X}'$ are ordered in decreasing order with respect to the frequency of their appearance in G, to create an ordered set of labels. The mutation procedure then traverses this ordered set of labels from $\mathbf{X}'$ (from the most frequent to the less frequent), and tries to remove labels while keeping the solution $\mathbf{X}'$ feasible. For a detailed implementation of the crossover and the mutation operations, we refer to Xiong et. al. [29].

3 | THE CROSS-ENTROPY METHOD

The CE method is a sequential procedure which similarly to other evolutionary algorithms, can be used to gradually change the sampling distribution of a random search such that the optimal solution is more likely to occur during the corresponding algorithm execution. However, the CE method is distinctive in the sense that it is not directly motivated by a pure evolutionary reasoning. Instead, it relies on information theory and stochastic simulation [2, 21, 23]. The CE method is very versatile; it can be used for rare-event estimation, discrete, continuous, and even noisy optimization [26]. In this manuscript we restrict our attention to the discrete optimization context only. In particular, consider the optimization problem: $\min_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x})$, where $S : \mathcal{X} \to \mathbb{R}$ is a fitness function, and $\mathcal{X}^* \subseteq \mathcal{X}$ is the set of optimal solutions. A very general framework for discrete optimization is depicted in Figure 2.

The general discrete optimization framework in Figure 2 begins with the *initialization* step in which a *probability mass function* (pmf) $g_1(\mathbf{x})$ for $\mathbf{X} \in \mathcal{X}$ is defined. Designing such pmf is generally easy, since one can select almost any distribution. A natural choice can be, for example, a uniform distribution on the $\mathcal{X}$ set. Next, we set the parameter γ_0 to infinity and initialize the iteration counter t. The following task is to *calculate the ρ-th quantile of the fitness* $Y = S(\mathbf{X})$, where $\mathbf{X} \sim g_t(\mathbf{x})$. We assume that the *cumulative distribution function* (cdf) of Y is $F_Y^{(t)}(y)$, where t is

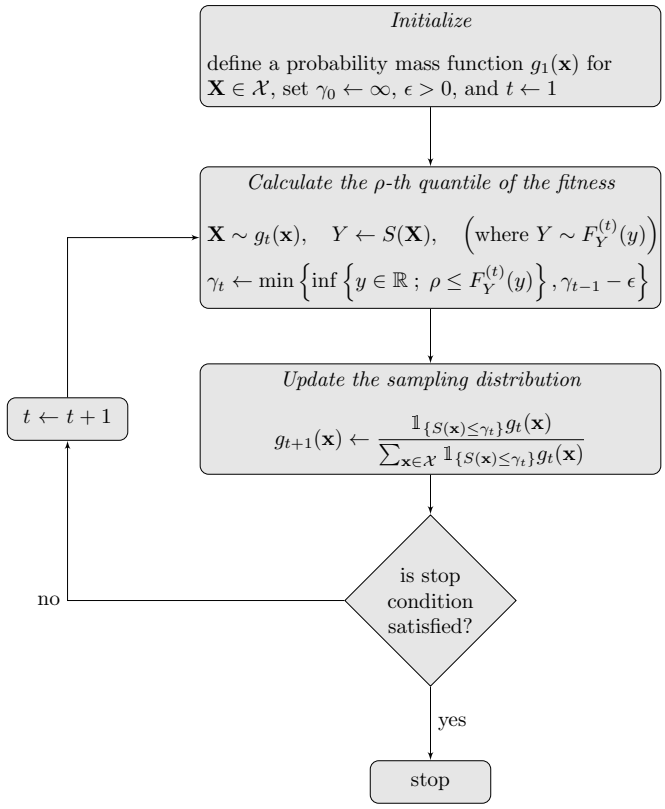


FIGURE 2 A general discrete optimization framework.

the current iteration counter. As soon as the ρ-th quantile is available, one can *update the sampling distribution*. This updated distribution will be used in the consecutive iteration, provided that the stop condition is not satisfied. Finally, the procedure terminates when some predefined stopping criterion is met. For example, one might stop if for all $\mathbf{x} \in \mathcal{X}$, it holds that $\mathbb{1}_{\{S(\mathbf{x}) \le \gamma_t\}} = 0$.

Note that for each iteration t, the fitness of $\mathbf{X} \sim g_t(\mathbf{x})$ satisfies $S(\mathbf{X}) \le \gamma_{t-1}$. In addition, since $\epsilon > 0$, this is not very hard to see that the sequence $\gamma_0, \gamma_1, \ldots$, is strictly decreasing, since $\gamma_t \le \gamma_{t-1} - \epsilon$ for all t. Moreover, upon the termination, the procedure will find a solution $\mathbf{x} \in \mathcal{X}$ that is at most ϵ far away from an optimal solution $\mathbf{x}^* \in \mathcal{X}^*$, that is, for such $\mathbf{x}$, it holds that $S(\mathbf{x}) \le S(\mathbf{x}^*) + \epsilon$. Finally, if the fitness function has a discrete range, namely, if $S : \mathcal{X} \to \mathbb{N}$, than for any $\epsilon \in (0, 1)$, the procedure will find an optimal solution that satisfies $\mathbf{x}^* \in \mathcal{X}^*$.

In practice, however, there are two major problems with the above methodology. First, for almost all practical situations $\mathcal{X}$ is large and therefore, finding an exact ρ-th quantile is computationally infeasible. The second problem is the hardness of sampling from $g_t(\mathbf{x})$ for all $t > 1$; note that the normalization constant of $g_t(\mathbf{x})$ is generally not available analytically. While the first issue can be resolved via simulation, that is, by finding a *sample fitness quantile* $\hat{\gamma}_t$ from $S(\mathbf{X}_1), \ldots, S(\mathbf{X}_N)$, where $X_i \sim g_t(\mathbf{x})$ for $1 \le j \le N$, the sampling from $g_t(\mathbf{x})$ remains hard. We next show how one can use the CE method to provide a solution to these problems in the context of the MLST task.

3.1 | Cross-Entropy method for the minimum label spanning tree problem

Under the setting of the MLST problem, we propose to define $\mathcal{X} = \{0,1\}^k$, $\mathbf{x} = (x_1, \ldots, x_k) \in \mathcal{X}$, and to adopt the (1+1) EA fitness function: $S(\mathbf{x}) = (\mathfrak{c}(\mathbf{x}) - 1) \times k \ln(k) + |\mathbf{x}|$. As usual, $\mathfrak{c}(\mathbf{x})$ is the number of connected components in $G' = (V, E')$, $E' = \{e \in E \mid x_{I(e)} = 1\}$, and $|\mathbf{x}| = \sum_{i=1}^{k} x_i$ is the number of different labels used in the spanning tree construction.

We proceed with the definition of the probability distribution of the random variable $\mathbf{X} = (X_1, \ldots, X_k)$, where $X_i \in \{0,1\}$ for $i = 1, \ldots, k$. Since the sampling from the pmf defined in the *Update the sampling distribution* phase in Figure 2, namely, from

$$g_{t+1}(\mathbf{x}) = \frac{\mathbb{1}_{\{S(\mathbf{x}) \leq \gamma_t\}} g_t(\mathbf{x})}{\sum_{\mathbf{x} \in \mathcal{X}} \mathbb{1}_{\{S(\mathbf{x}) \leq \gamma_t\}} g_t(\mathbf{x})}, \tag{2}$$

is generally hard, we propose to approximate (2) using a parametric family

$$f(\mathbf{x}; \boldsymbol{p}_{t+1}) = \prod_{i=1}^{k} p_{t+1,i}^{x_i} (1 - p_{t+1,i})^{1-x_i}, \tag{3}$$

where $\boldsymbol{p}_{t+1} = (p_{t+1,1}, \ldots, p_{t+1,k})$, and $\boldsymbol{p}_{t+1} \in [0,1]^k$. The parametric family that was defined in (3) is a joint pmf of k independent Bernoulli random variables. That is, one can sample $\mathbf{X} = (X_1, \ldots, X_k)$ component-wise, and independently for each X_i for $1 \leq i \leq k$; namely, $X_i \sim \mathrm{Ber}(p_{t+1,i})$. It is important to note that the sampling from (3) is easy as compared to the corresponding sampling from (2). To summarize, in order to approximate the sequence of pmfs $\{g_t\}$ for all $t \in \mathbb{N} \setminus \{0\}$ in (2), we wish to obtain the corresponding parameter vectors $\{\boldsymbol{p}_t\}$, and use (3) instead.

A note regarding the parametric family defined in (3) It is important to realize that given $G = (V, E, I)$, there exists $\boldsymbol{p}^* = (p_1^*, \ldots, p_k^*)$, such that for any $\mathbf{X} \sim f(\mathbf{x}; \boldsymbol{p}^*)$, $\mathbf{X}$ corresponds to the optimal solution to the MLST problem. The latter is straight-forward because of the following reasoning. Suppose that $\mathbf{x}^* = (x_1^*, \ldots, x_k^*) \in \mathcal{X}^*$ is an optimal solution. Then, noting that $x_i^* \in \{0,1\}$ for $1 \leq i \leq k$, and setting $\boldsymbol{p}^* = (x_1^*, \ldots, x_k^*)$ is sufficient, since in this case, $f(\mathbf{x}; \boldsymbol{p}^*)$ is a *degenerate* joint pmf in the sense that $\mathbb{P}_{f(\mathbf{x}; \boldsymbol{p}^*)}(\mathbf{X} = \mathbf{x}^*) = 1$. In other words, a *single* sample from $f(\mathbf{x}; \boldsymbol{p}^*)$ will result in an optimal solution.

We defined the sequence of pmfs in (3), and highlighted the fact that they are easy to sample from. However, we still need to discuss the problem of calculating the sample fitness quantile and the procedure of obtaining the desired sequence of parameters $\{\boldsymbol{p}_t\}$ for all $t \in \mathbb{N} \setminus \{0\}$.

| Calculating the sample quantile

The calculation of the sample fitness quantile $\hat{\gamma}_t$ is trivial, provided that $\boldsymbol{p}_t$ is readily available. Specifically, it is sufficient to sample $\mathbf{X}_j \sim f(\mathbf{x}; \boldsymbol{p}_t)$ for $1 \leq j \leq N$, and sort the $\{S(\mathbf{X}_j)\}_{j=1}^{N}$ set in the ascending order. Denote such an ordering by $S_{(1)} \leq \cdots \leq S_{(N)}$. Then, $\hat{\gamma}_t \leftarrow S_{(\lceil N \times \rho \rceil)}$ is the desired sample fitness (ρ-th) quantile.

| Approximating the sampling pmf

Our final objective is to approximate the (optimal) sampling pmf (2) via $f(\mathbf{x}; \boldsymbol{p}_{t+1})$, where $f(\mathbf{x}; \boldsymbol{p}_{t+1})$ belongs to the parametric family defined in (3), that is, to find $\boldsymbol{p}_{t+1}$. The latter will be accomplished via a minimization of the rel-

ative entropy (also denoted by Kullback-Leibler divergence) of $f(\mathbf{x};\boldsymbol{p}_{t+1})$ with respect to $g_{t+1}(\mathbf{x})$ [21]. The formal characterization of the relative entropy concept is provided in Definition 3.1.

Relative entropy The relative entropy of a pmf $f(\cdot)$ with respect to a pmf $g(\cdot)$ is given by:

$$\mathcal{D}(g,f) = \mathbb{E}_g \ln\left(\frac{g(\mathbf{X})}{f(\mathbf{X})}\right) = \sum_{\mathbf{x}} \ln\left(\frac{g(\mathbf{x})}{f(\mathbf{x})}\right) g(\mathbf{x}) = \sum_{\mathbf{x}} g(\mathbf{x}) \ln g(\mathbf{x}) - \sum_{\mathbf{x}} g(\mathbf{x}) \ln f(\mathbf{x}).$$

It is convenient to think about the relative entropy as a *distance measure* between these two pmfs. While the relative entropy is not a regular distance measure in the sense that $\mathcal{D}(g,f)$ is generally not equal to $\mathcal{D}(f,g)$, it is possible to show that $\mathcal{D}(g,f) \geq 0$ and that the equality occurs if $g = f$.

Under our setting, we wish to find $\boldsymbol{p}_{t+1}$ such that $\mathcal{D}(g_{t+1}(\mathbf{x}), f(\mathbf{x},\boldsymbol{p}_{t+1}))$ is minimized. From Definition 3.1, it holds that

$$\min_{\boldsymbol{p}_{t+1}} \mathcal{D}(g_{t+1}(\mathbf{x}), f(\mathbf{x},\boldsymbol{p}_{t+1}) = \min_{\boldsymbol{p}_{t+1}} \left(\sum_{\mathbf{x}} g_{t+1}(\mathbf{x}) \ln g_{t+1}(\mathbf{x}) - \underbrace{\sum_{\mathbf{x}} g_{t+1}(\mathbf{x}) \ln f(\mathbf{x};\boldsymbol{p}_{t+1})}_{(*)} \right) \tag{4}$$

Note that the optimization problem (4) is with respect to the $\boldsymbol{p}_{t+1}$ parameter. Thus, the *minimization* problem (4) is equivalent to a *maximization* problem of the second term $(*)$ with respect to $\boldsymbol{p}_{t+1}$. Suppose that $\boldsymbol{p}_{t+1} = (p_{t+1,1},\ldots,p_{t+1,k})$ and that $\mathbf{x} = (x_1,\ldots,x_k)$. Then, using the definitions of $g_{t+1}(\mathbf{x})$ and $f(\mathbf{x};\boldsymbol{p}_{t+1})$ (namely, (2) and (3)), the corresponding maximization problem $(*)$, can be written in the form:

$$\begin{aligned}
\max_{\boldsymbol{p}_{t+1}} \sum_{\mathbf{x}} g_{t+1}(\mathbf{x}) \ln f(\mathbf{x};\boldsymbol{p}_{t+1}) &= \max_{\boldsymbol{p}_{t+1}} \sum_{\mathbf{x}} \frac{\mathbb{1}_{\{S(\mathbf{x})\leq\gamma_t\}} f(\mathbf{x};\boldsymbol{p}_t)}{\sum_{\mathbf{x}\in\mathcal{X}} \mathbb{1}_{\{S(\mathbf{x})\leq\gamma_t\}} f(\mathbf{x};\boldsymbol{p}_t)} \times \ln\left(\prod_{i=1}^{k} p_{t+1,i}^{x_i}(1-p_{t+1,i})^{1-x_i}\right) \\
&= \max_{\boldsymbol{p}_{t+1}} \sum_{\mathbf{x}} \mathbb{1}_{\{S(\mathbf{x})\leq\gamma_t\}} f(\mathbf{x};\boldsymbol{p}_t) \times \ln\left(\prod_{i=1}^{k} p_{t+1,i}^{x_i}(1-p_{t+1,i})^{1-x_i}\right) \\
&= \max_{\boldsymbol{p}_{t+1}} \mathbb{E}_{f(\mathbf{x};\boldsymbol{p}_t)} \mathbb{1}_{\{S(\mathbf{X})\leq\gamma_t\}} \times \ln\left(\prod_{i=1}^{k} p_{t+1,i}^{X_i}(1-p_{t+1,i})^{1-X_i}\right),
\end{aligned} \tag{5}$$

where the second equality follows from the fact that the denominator $\sum_{\mathbf{x}\in\mathcal{X}} \mathbb{1}_{\{S(\mathbf{x})\leq\gamma_t\}} f(\mathbf{x};\boldsymbol{p}_t)$ is both constant and does not depend on the optimization parameter $\boldsymbol{p}_{t+1}$. Thus, this denominator does not affect the optimization problem. The exact evaluation of the expected value (5) is generally not feasible, however, it can be approximated via sampling from the $f(\mathbf{x};\boldsymbol{p}_t)$ pmf. In particular, we can work with the so-called *stochastic counterpart* [26]. Namely, the solution of (5) can be approximated by:

$$\max_{\boldsymbol{p}_{t+1}} \frac{1}{N} \sum_{j=1}^{N} \mathbb{1}_{\left\{S(\mathbf{X}_j)\leq\gamma_t\right\}} \ln\left(\prod_{i=1}^{k} p_{t+1,i}^{X_{j,i}}(1-p_{t+1,i})^{1-X_{j,i}}\right), \tag{6}$$

where $\mathbf{X}_j = (X_{j,1},\ldots,X_{j,k}) \sim f(\mathbf{x};\boldsymbol{p}_t)$ for $j = 1,\ldots,N$.

The function in (6) is concave and differentiable with respect to $\boldsymbol{p}_{t+1}$ (see Lemma 1 in the Appendix, therefore, the optimal parameter $\boldsymbol{p}^*_{t+1} = (p^*_{t+1,1},\ldots,p^*_{t+1,k})$ which maximizes (6), can be obtained by solving:

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{1}_{\left\{S(\mathbf{X}_j)\leq\gamma_t\right\}} \bigtriangledown \ln\left(\prod_{i=1}^{k} p_{t+1,i}^{X_{j,i}}(1-p_{t+1,i})^{1-X_{j,i}}\right) = 0,$$

where the gradient is with respect to $\boldsymbol{p}_{t+1}$. Specifically, from Lemma 2 in the Appendix, it holds that

$$p^*_{t+1,i} = \frac{\sum_{j=1}^{N} \mathbb{1}_{\left\{S(\mathbf{X}_j) \leq \gamma_t\right\}} X_{j,i}}{\sum_{j=1}^{N} \mathbb{1}_{\left\{S(\mathbf{X}_j) \leq \gamma_t\right\}}} \quad \forall\, 1 \leq i \leq k.$$

The above discussion combined with the general discrete optimization framework from Figure 2, results in the CE method for the MLST problem which is summarized in Algorithm 5.

Algorithm 5: The CE algorithm

Input: A graph $G = (V, E, l)$, where V is a set of vertices, E is a set of edges, and $l : E \to \mathcal{L} = \{1, \ldots, k\}$ is a labeling function, a sample size $N \in \mathbb{N}$, a smoothing parameter $\alpha \in (0, 1)$, and a rarity parameter $\rho \in (0, 1)$.

Output: Minimum label spanning tree of G.

1 Set $t \leftarrow 1$ and $\boldsymbol{p}_t \leftarrow (p_1, \ldots, p_k)$, such that $p_i = 0.5$ for $1 \leq i \leq k$.

2 **while** *termination criterion is not fulfilled* **do**

3 Sample $\mathbf{X}_j \sim f(\mathbf{x}; \boldsymbol{p}_t)$, and calculate $S(\mathbf{X}_j)$ for $1 \leq j \leq N$.

4 Let $S_{(1)} \leq \cdots \leq S_{(N)}$ be the elements of the $\{S(\mathbf{X}_j)\}_{j=1}^{N}$ set sorted in an ascending order.

5 $\hat{\gamma}_t \leftarrow S_{(\lceil N \times \rho \rceil)}$ `/* Find the ρ-quantile of the sample fitness */`

`/* Find` $\boldsymbol{p}_{t+1}$ `*/`

6 **for** $i = 1, \ldots, k$ **do**

7 $$\tilde{p}_{t+1,i} \leftarrow \frac{\sum_{j=1}^{N} \mathbb{1}_{\{S(\mathbf{X}_j) \leq \hat{\gamma}_t\}} X_{j,i}}{\sum_{j=1}^{N} \mathbb{1}_{\{S(\mathbf{X}_j) \leq \hat{\gamma}_t\}}}$$

`/* note that` $X_{j,i}$ `is the` i`-th component of` $\mathbf{X}_j$ `*/`

8 **end**

9 $\tilde{\boldsymbol{p}}_{t+1} \leftarrow (\tilde{p}_{t+1,1}, \ldots, \tilde{p}_{t+1,k})$

`/* smoothing */`

10 $\boldsymbol{p}_{t+1} \leftarrow (1 - \alpha)\, \boldsymbol{p}_t + \alpha\, \tilde{\boldsymbol{p}}_{t+1}$

11 $t \leftarrow t + 1$

12 **end**

13 $\mathbf{X} = (X_1, \ldots, X_k) \sim f(\mathbf{x}; \boldsymbol{p}_t)$

14 **return** A spanning tree of $G' = (V, E')$, where $E' = \left\{e \in E \mid X_{l(e)} = 1\right\}$.

Similar to other evolutionary methods discussed in this manuscript, the termination criteria in line 2 of Algorithm 5 depends on the user preferences such as the desired number of iteration or the runtime threshold. In addition, it is also possible to store the obtained sample fitness quantiles $\{\hat{\gamma}_t\}$ for $t > 0$ during the algorithm execution, and declare the algorithm's termination if the sample quantile sequence stops decreasing for several consecutive iterations.

Finally, we discuss the *smoothing* step in line 10 of Algorithm 5. Recall that the optimization problem (5) cannot be generally solved analytically. Consequently, we instead work with the stochastic counterpart (6). The stochastic counterpart approach, however, may result in a possibility of obtaining some that *unfavorable* sample set at a specific

iteration t. Such sample set can affect the update of $\boldsymbol{p}_{t+1}$ and cause the random search procedure to focus on a sub-optimal region of the search space. The latter is especially true if the sample size N is sufficiently small. This, in turn, may result in the algorithm termination in this sub-optimal region of the search space. In order to avoid the above problem, we use the so called smoothing idea [26]. The latter allows to control the speed of convergence of the CE algorithm in the sense that smoothing causes a *slower* updating of the probability distribution parameters in (3).

4 | EXPERIMENTAL RESULTS

In this section, we focus on the performance comparison of the MVCA, the (1+1) EA, the GSEMO, the GA, and the CE algorithms, when applied to different graph models. Specifically, we consider the following four case studies.

1. In the first test-case, we examine a binary tree graph of height 10. For a tree, the optimal solution should contain all graph labels, since every edge must be present in the corresponding spanning tree. Consequently, the availability of the optimal solution allows to benchmark the performance of all algorithms.
2. The second test-case consists of a very specific graph model for which we know the optimal solution, and hence the model is also used to benchmark the accuracy of all algorithms under consideration. It is worth noting that this network type was specifically designed to illustrate the worst case behavior of the MVCA algorithm [30].
3. The third test-case incorporates two-dimensional grid graphs of various sizes. The largest example is a 32×32 2D-grid model with 1024 vertices. The 32×32 grid is considered with a view to resemble a real-world average sized communication network.
4. Finally, for the forth test-case, we consider a well-known Watts-Strogatz (WS) small-world random graph model. We choose the WS model, since it was shown to provide explanation to the "small-world" phenomena in a variety of connected structures such as power grids, protein networks, and social networks [1].

Our study shows the following. For the first and the second test-cases, the (1+1) EA, the GSEMO, the GA, and the CE algorithms always manage to find an optimal solution. As expected, for the second test-case, MVCA introduces an inferior performance since the model was specifically designed to demonstrate its worst-case behavior. For both the first and the second case studies, the GSEMO algorithm experienced difficulties to converge. Specifically, for the binary tree, the GSEMO failed to converge when the number of labels increased. For the second case study, the GSEMO algorithm managed to find solutions to smaller instances (with 19 and 97 vertices), but failed to converge in a reasonable time for larger graphs (with 601 and 4321 vertices). For the third test-case, the performance of the MVCA, the (1+1) EA, and the GA algorithms, was inferior as compared to the GSEMO and the CE methods. In addition, the CE method outperformed the GSEMO algorithm when a larger number of labels was introduced. Similarly, in the forth test-case, both the GSEMO and the CE methods outperformed the MVCA, the (1+1) EA, and the GA algorithms. However, for one graph instance, GSEMO managed to find a better solution than the (sub-optimal) solution obtained by the CE algorithm. Overall, our experimental study supports the conjecture that the CE algorithm is less sensitive to different graph types. The details are provided bellow.

| Experimental setup

We implemented all algorithms in `C++` packages called *Mvca*, *EA*, *GSEMO*, *GA*, and *MlstCE*. These packages are freely available on the author's website. The software was compiled using `GNU g++` with full optimization for speed (using

the -O3 flag). All timing measures were instrumented directly into the code and all algorithms except of the MVCA method, are forced to reach a predefined time limit before termination. All reported times are in seconds. The tests were executed on Intel Core(TM) i7-6920HQ CPU 2.90GHz processor with 32GB of RAM running 64 bit Ubuntu 18.04 LTS. We did not implement parallelization for any algorithm, so all the software is single-threaded. However, due to the nature of the CE algorithm, parallelization would be relatively easy to implement.

With a view to ensure reproducibility of the reported results, unless stated otherwise, we use a fixed seed of 12345 when executing all algorithms. The MVCA, the (1+1) EA, and the GSEMO methods does not need any parameter tuning and the only criterion under consideration is the runtime threshold. The GA algorithm requires the population size to be chosen in advance; we follow the recommendation of Xiong et. al. [29], and set the population size N to be equal to 30. The CE algorithm requires three parameters: ρ, α, and N. For the rest of this section the rarity parameter ρ is fixed to be 0.1 and the smoothing parameter α is set to be 0.7. Unless stated otherwise, the default sample size N is $N = 10 \times k$, where k is the total number of labels.

4.1 | Test-case 1: the binary tree

In order to benchmark the performance of all algorithms, we consider a binary tree of height 10, namely, the corresponding graph has 1024 vertices and 1023 edges. Next, we created 10 MLST problem instances using this tree as follows. For $j \in \{10, 20, \ldots, 100\}$, define $\mathcal{T}_j$ to be a binary tree of height 10, where the corresponding edge labels were assigned uniformly at random from the $\{1, \ldots, j\}$ set. For convenience, we force every label from the $\{1, \ldots, j\}$ set to be selected at least once. Since all edges are required to be present in the optimal solution, we conclude that the number of necessary labels in $\mathcal{T}_j$ is j. All algorithms (except of MVCA), were forced to comply with runtime thresholds of 5 seconds. Each algorithm was executed ten times and for each run, we used the 12345+i, $i = 0, \ldots, 9$ seed. The results are summarized in Table 1. One can observe from the table that the GSEMO algorithm experienced convergence problems, when the number of used labels increased.

TABLE 1 Average performance among ten runs of the MVCA, the (1+1) EA, the GSEMO, the GA, and the CE algorithms when applied to binary trees with various number of labels. The time-limit for all algorithms (except of the MVCA) is set to be 5 seconds. For the $\mathcal{T}_{60}$ case (*), GSEMO found the correct solution in 4 out of 10 runs. For $\mathcal{T}_{70}$, $\mathcal{T}_{80}$, $\mathcal{T}_{90}$, and $\mathcal{T}_{100}$, GSEMO failed to converge.

instance	MVCA	time (sec)	(1+1) EA	GSEMO	GA	CE
$\mathcal{T}_{10}$	**10**	5.15×10^{-3}	**10**	**10**	**10**	**10**
$\mathcal{T}_{20}$	**20**	1.56×10^{-2}	**20**	**20**	**20**	**20**
$\mathcal{T}_{30}$	**30**	2.75×10^{-2}	**30**	**30**	**30**	**30**
$\mathcal{T}_{40}$	**40**	4.98×10^{-2}	**40**	**40**	**40**	**40**
$\mathcal{T}_{50}$	**50**	7.95×10^{-2}	**50**	**50**	**50**	**50**
$\mathcal{T}_{60}$	**60**	8.78×10^{-2}	**60**	**60***	**60**	**60**
$\mathcal{T}_{70}$	**70**	1.20×10^{-1}	**70**	-	**70**	**70**
$\mathcal{T}_{80}$	**80**	1.65×10^{-1}	**80**	–	**80**	**80**
$\mathcal{T}_{90}$	**90**	2.23×10^{-1}	**90**	–	**90**	**90**
$\mathcal{T}_{100}$	**100**	2.71×10^{-1}	**100**	–	**100**	**100**

In order to examine the GSEMO convergence behavior when the allowed execution time grows, we increased the time limit threshold from 5 to 50 seconds. As a result, for the $\mathcal{T}_{60}$ instance, GSEMO always managed to converge to the optimal solution. However, for the $\mathcal{T}_{70}$ and the $\mathcal{T}_{80}$ graphs, GSEMO converged in one out of 10 runs and in two out of 10 runs, repetitively. Finally, GSEMO did not reach an optimal solution for the $\mathcal{T}_{90}$ tree at all.

4.2 | Test-case 2: the specifically designed network

For the additional benchmarking purpose, we consider a very structured graph family from [30]. While this network type was specifically designed to illustrate the worst case behavior of the MVCA algorithm, it was shown that the MVCA method achieves an approximation ratio of $\sum_{i=1}^{b} \frac{1}{i}$. The corresponding graph construction is governed by a parameter $b \in \mathbb{N}$ and is defined as follows. Let $\mathcal{H}_b = (V, E)$ and let $|V| = b \cdot b! + 1$, where V is divided into $b!$ groups such that

$$
\begin{aligned}
V_1 &= \{1, 2, \ldots, b+1\},\\
V_2 &= \{b+1, b+2, \ldots, 2b+1\},\\
&\vdots\\
V_i &= \{(i-1)b+1, (i-1)b+2, \ldots, ib+1\},\\
&\vdots\\
V_{b!} &= \{(b!-1)b+1, (b!-1)b+2, \ldots, b! \cdot b+1\}.
\end{aligned}
$$

The $\mathcal{H}_b$ graph labeling is constructed as follows. First, each edge between consecutive nodes in V_i, namely, the edges $((i-1)b+1,\ (i-1)b+2)),\ ((i-1)b+2,\ (i-1)b+3)), \ldots, (ib,\ ib+1)$, are labeled with label i for $i = 1, \ldots, b!$. Such assignment yields $b!$ different labels and corresponds to the *optimal solution* to the MLST problem of the $\mathcal{H}_b$ graph. Finally, for $k = 2, \ldots, b$, we consider the

$$
\left\{ \Big(\underbrace{(i-1)b+1}_{\in V},\ \underbrace{(i-1)b+1+k}_{\in V} \Big) \right\}_{i=1}^{b!}
$$

edge set. These edges are labeled sequentially using unique labels such that each k edges receive a different label. For example, Figure 3 depicts the $\mathcal{H}_2$ graph.

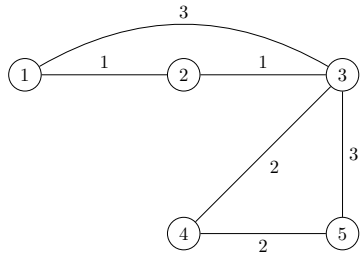


FIGURE 3 The $\mathcal{H}_2$ graph with five vertices $\{1, 2, 3, 4, 5\}$ and three labels $\{1, 2, 3\}$.

The V_i sets for $i \in \{1, 2\}$ are: $V_1 = \{1, 2, 3\}$ and $V_2 = \{3, 4, 5\}$. The edges $(1, 2)$, $(2, 3)$ and $(3, 4)$, $(4, 5)$ are labeled with labels 1 and 2, respectively. Finally, edges $(1, 3)$ and $(4, 5)$ are labeled with label 3. In this test case, we consider four instances with $b = 3, 4, 5$ and $b = 6$. All algorithms (except of MVCA), were forced to comply with runtime thresholds of 0.1, 0.5, 60, and 2000 seconds for $b = 3, 4, 5$ and $b = 6$, respectively. Each algorithm was executed ten times. For each run, we used the $12345+i$, $i = 0, \ldots, 9$ seed. The results are summarized in Table 2.

TABLE 2 Average performance among ten runs of the MVCA, the (1+1) EA, the GSEMO, the GA, and the CE algorithms when applied to $\mathcal{H}_b$ networks for $b \in \{3, 4, 5, 6\}$. The time-limit for all algorithms (except of the MVCA) is set to be 0.1, 0.5, 60, and 2000 seconds for $b = 3, 4, 5$ and $b = 6$, respectively. The *optimal* column stands for the optimal solution; for an $\mathcal{H}_b$ network, the optimal solution can be constructed using $b!$ labels.

b	optimal	MVCA	time (sec)	(1+1) EA	GSEMO	GA	CE
3	**6**	6.6	1.62×10^{-4}	**6**	**6**	**6**	**6**
4	**24**	25.5	3.99×10^{-3}	**24**	28.5	**24**	**24**
5	**120**	124.8	6.26×10^{-1}	**120**	–	**120**	**120**
6	**720**	738.3	2.25×10^{2}	**720**	–	**720**	**720**

As expected, MVCA introduces an inferior performance since these graphs were specifically constructed for this purpose. The (1+1) EA, the GA, and the CE algorithms always managed to find the optimal solution. However, we experienced convergence problems with the GSEMO method for $b = 5$ and $b = 6$. Specifically, in all runs, the GSEMO algorithm could not find a solution (a set of labels), that corresponds to a single connected component for both the $\mathcal{H}_5$ and the $\mathcal{H}_6$ instances. We tried to increase the runtime threshold for $\mathcal{H}_5$ from 60 seconds to 600 seconds, but the GSEMO algorithm was still unsuccessful in all ten runs. For the $\mathcal{H}_6$ instance, we failed to see the convergence of the GSEMO algorithm in a reasonable time, too.

4.3 | Test-case 3: the 2D-grid

For the third test-case, we consider a 32×32 2D-grid with $|V| = 1024$ vertices and $|E| = 1984$ edges. In particular, we examine five 32×32 2D-grid instances where each instance has a different number of labels. These grids are being constructed as follows. First, we define a label density d. Then, for each grid instance, we set the number of possible labels k to be $\lfloor |V| \times d \rfloor$, where $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and $\lfloor \cdot \rfloor$ is the ceiling function. In order to finalize the construction, labels are assigned to grid's edges uniformly at random. Each algorithm (except of the MVCA method), was given a 120 seconds runtime threshold. Table 3 summarizes the obtained results.

TABLE 3 Performance of the MVCA, the (1+1) EA, the GSEMO, the GA, and the CE algorithms when applied to 32×32 2D-grids with different label densities. The time-limit for all algorithms (except of the MVCA) is set to be 120 seconds.

label density d	MVCA	time (sec)	(1+1) EA	GSEMO	GA	CE
0.1	63	0.136	64	**59**	61	**59**
0.2	109	0.501	116	**103**	117	106
0.3	146	0.951	166	146	161	**144**
0.4	186	1.497	210	184	237	**177**
0.5	219	2.234	260	229	257	**216**

Table 3 is instructive in the sense that for this particular network architecture, (1+1) EA and the GA algorithms introduce the worse performance (note that these algorithms managed to find optimal solutions for all test cases in Section 4.1 and Section 4.2). As always, MVCA achieves good (sub-optimal) results. The GSEMO algorithm is almost always better than MVCA (with the exception of the $d = 0.5$ case). However, when we consider the $0.1, 0.3, 0.4$, and the 0.5 densities, CE delivers superior results. For the 0.2 density, however, GSEMO outperforms the CE algorithm.

The inferior performance of CE as compared to GSEMO for the 0.2 density case (note that CE and GSEMO achieve the fitness of 106 and 103, respectively), requires a careful discussion. It turns out that CE performance can be improved by increasing the sample size N. The reason for this suggestion is straight-forward. Recall that the CE algorithm estimates the optimal sampling distribution via the stochastic counterpart. Clearly, increasing the sample size N will improve the estimation, since we obtain a better approximation of the expected value (5) (in (6)), as N grows. Next, the CE algorithm is executed on the 0.2 density grid instance (with $k = \lfloor 1024 \times 0.2 \rfloor = 204$), using the sample size $N = 20 \times k = 20 \times 204 = 4080$ (instead of the default $N = 10 \times 204 = 2040$ sample size). We still impose the 120 seconds runtime threshold and use the 12345 seed. The CE algorithm manages to discover the best known solution that uses 102 labels; note that this is an improvement over the GSEMO algorithm solution that uses 103 labels. Figure 4 depicts the dynamic of the CE algorithm for the 0.2 density grid.

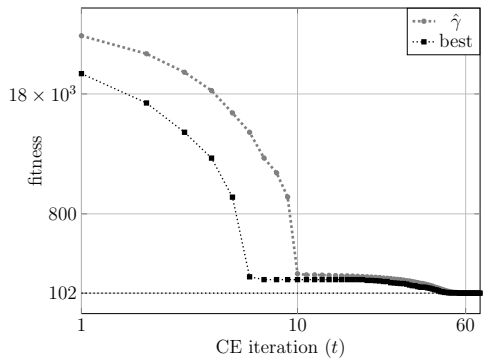


FIGURE 4 CE dynamics of the sample quantile $\hat{\gamma}$ and the best observed fitness as a function of the CE algorithm iteration for the 0.2 density grid with $N = 4080$ sample size.

In order to check how the quality of solutions changes as the size of the problem varies, we generated additional 2D grid instances, namely, 5×5, 10×10, 15×15, 20×20, 25×25, and 30×30 grids, each having the number of possible labels k to be equal to $\lfloor |V| \times d \rfloor$, where $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. We set the time limit to all algorithms (except of the MVCA) to be 0.5, 2, 10, 20, 45, and 90 seconds, for 5×5, 10×10, 15×15, 20×20, 25×25, and 30×30 grids, respectively. As for the Test-cases 1 and 2, for each run of every algorithm, we used the $12345 + i$, $i = 0, \ldots, 9$ seed. The results are summarized in the Table 4. Table 4 is instructive in the sense that the GSEMO has the best performance. Specifically, GSEMO finds the best solution in 22 cases out of 30 as compared to 17 cases out of 30 found by CE. Nevertheless, the results obtained by CE are very close to the one achieved by GSEMO, and, CE outperforms GSEMO when the number of labels in the problem increases. We also note that the GSEMO failed to converge for some instances in test case 1 and 2; that is, CE appears to be more robust than the GSEMO algorithm. The remaining algorithms, namely, the MVCA, the (1+1) EA, and the GA, found best solutions in 4, 5, and 10 out of 30 instances, respectively.

TABLE 4 Performance of the MVCA, the (1+1) EA, the GSEMO, the GA, and the CE algorithms when applied to 2D-grids with different label densities. The times for all algorithms (except of the MVCA) to be 0.5, 2, 10, 20, 45, and 90 seconds for 5×5, 10×10, 15×15, 20×20, 25×25, and 30×30 grids, respectively.

Grid size	d	MVCA	time (sec)	(1+1) EA	GSEMO	GA	CE
5×5	0.1	2	1.51×10^{-4}	**2**	**2**	**2**	**2**
	0.2	4	1.62×10^{-4}	**4**	**4**	**4**	**4**
	0.3	5	1.84×10^{-4}	**5**	**5**	**5**	**5**
	0.4	5.4	2.04×10^{-4}	**5**	**5**	**5**	5.1
	0.5	6	2.20×10^{-4}	**6**	**6**	**6**	**6**
10×10	0.1	7	5.67×10^{-4}	6.1	**6**	**6**	**6**
	0.2	11	9.96×10^{-4}	11.6	**11**	**11**	**11**
	0.3	15.4	1.67×10^{-3}	15.5	**15**	15.6	**15**
	0.4	19	2.44×10^{-3}	19.8	**18**	**18**	18.1
	0.5	22.4	3.20×10^{-3}	25.1	**21**	23.4	21.7
15×15	0.1	13.6	2.62×10^{-3}	13.2	**13**	**13**	**13**
	0.2	23.2	6.45×10^{-3}	24.7	**23**	**23**	23.3
	0.3	33.1	1.25×10^{-2}	35.5	**31.5**	34.9	32.5
	0.4	41.7	1.94×10^{-2}	47.2	**39.8**	49.2	40
	0.5	50.9	2.83×10^{-2}	56.9	**49.8**	59	**49.8**
20×20	0.1	24.8	1.04×10^{-2}	25.7	**24**	25	25
	0.2	41.1	3.01×10^{-2}	45.1	**39.4**	43.2	40.6
	0.3	61.7	6.46×10^{-2}	65.5	59.7	62.4	**58.7**
	0.4	71.6	1.01×10^{-1}	81.9	69.5	69.5	**69.4**
	0.5	83.2	1.46×10^{-1}	98.2	82.4	101.4	**81.7**
25×25	0.1	38.2	3.45×10^{-2}	39.2	**36.2**	36.5	37.5
	0.2	65.1	1.12×10^{-1}	70.6	**62.8**	67.5	63.7
	0.3	89.6	2.30×10^{-1}	100.9	**87.9**	94.7	88
	0.4	118.5	3.94×10^{-1}	133.7	116.4	130.1	**114.4**
	0.5	130.1	5.54×10^{-1}	157.2	133.6	159.4	**129.8**
30×30	0.1	54.9	9.96×10^{-2}	56.6	**51**	53.8	52.9
	0.2	94.6	3.30×10^{-1}	102.5	**90.6**	103.5	91.9
	0.3	129.6	6.98×10^{-1}	147.7	128.4	155.4	**127.9**
	0.4	164.5	1.18	189.7	164.9	183.9	**161.3**
	0.5	192.4	1.73	226.6	198.1	235.1	**186.2**

4.4 | Test-case 4: the Watts-Strogatz small-world random graph

Motivated by the fact that many real-world networks can be modeled via random graphs, we test the algorithmic performance on the well-known WS small-world random graph model [28]. The WS model is characterized by the number of vertices $|V|$, the number of nearest neighbors connected to each node $0 \leq \eta \leq |V|$, and the edge rewiring probability $\beta \in [0,1]$. We generated three graphs using $|V| = 100$, $\eta = 10$, and $\beta = 0.5$. For each edge, a label was assigned uniformly at random from the $\{1,\ldots,100\}$ set. Each algorithm (except of the MVCA method), was given a 20 seconds runtime threshold. Table 5 summarizes the obtained results.

TABLE 5 Performance of the MVCA, the (1+1) EA, the GSEMO, and the CE algorithms when applied to three realizations of the WS graph model. The time-limit for all algorithms (except of the MVCA) is set to be 20 seconds.

instance	MVCA	time (sec)	(1+1) EA	GSEMO	GA	CE
Graph 1	**13**	5.00×10^{-3}	16	**13**	**13**	**13**
Graph 2	16	6.00×10^{-3}	20	**15**	17	**15**
Graph 3	17	6.00×10^{-3}	18	**15**	16	16

Table 5 shows that the GSEMO outperforms its counterparts. The CE algorithm performance is similar, but for Graph 3, CE achieves the fitness of 16 as compared to the fitness of 15 obtained by the GSEMO algorithm. Applying the idea from Section 4.3, we next try to improve the CE results for the Graph 3 instance by increasing the sample size N. For $N = 10^7$, the CE algorithm manages to obtain the best known fitness of 15 (unfortunately, for $N < 10^7$, CE failed to discover this solution). However, the required computation time is about 13700 seconds. Figure 5 depicts the corresponding dynamics. The CE algorithm runtime with $N = 10^7$ sample size is prohibitively large, especially, if the algorithm is used for network design. The latter is due to the fact that for network design problems, we expect the algorithm to run fast for any given network configuration.

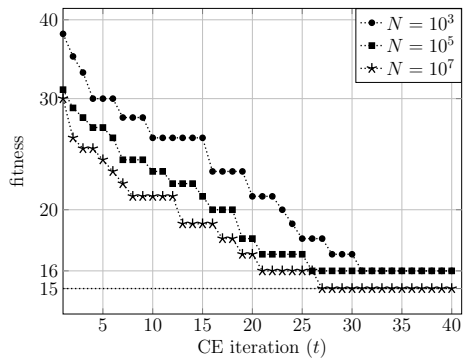


FIGURE 5 CE dynamics of the best observed fitness as a function of the CE algorithm iteration for Graph 3 WS instance with different sample sizes. The execution times for $N = 10^3, 10^5$ and 10^7 are 1.15, 113, and 13616 seconds, respectively.

To ensure a fair comparison, we ran the (1+1) EA, the GA, and the GSEMO methods for 16000 seconds on all WS instances. The (1+1) EA algorithm managed to improve the fitness obtained for Graph 1 from 16 to 15. however, no other improvements to the results from Table 5 were observed.

5 | CONCLUSION

In this manuscript we showed that the Cross-Entropy method can by successfully applied to the problem of finding a minimum label spanning tree. In particular, we showed that the Cross Entropy framework is suitable for both finding an approximation to quantiles of the corresponding fitness function via simulation, and for sampling from a parametric family of distributions with a view to approximating the true probability mass function of interest. A resolution of these problems opens a way for the application of the Cross Entropy method to the problem of finding minimum label spanning trees. We outlined the corresponding mathematical foundation and our experimental study indicates that the practical performance is comparable with or better than its counterparts. In addition, we developed freely available software packages that implement all methods discussed in this paper. As for the future work, the following directions are of crucial importance. First, following our findings in Section 4.4, it is of great interest to provide a rigorous characterization of graphs that impose a challenge to the Cross-Entropy method in the sense of the required sample size. Moreover, for such graphs, one needs to establish bounds on the number of iterations needed for the algorithm to complete. As noted in Section 4.4, the sample size of 10^7 is prohibitively large. Therefore, having in mind that the Cross-Entropy method might necessitate a large sample size to achieve an optimal solution, it will be of interest to develop a version of the algorithm that uses a more efficient memory management scheme. In other words, both researchers and practitioners will definitely benefit from an existence of an on-line version of the method. It is also important to compare the performance of other methods such as the ant colony optimization, the integer programming, local search heuristic, and the intelligent optimization method of Consoli et. al., on different types of networks. Finally, from a practical point of view, a development of a software package that runs on multiple CPUs or a GPU will be useful, since the latter will open a way for an efficient treatment of large networks.

ACKNOWLEDGMENTS

We are thoroughly grateful to the anonymous reviewers for their valuable and constructive remarks and suggestions. This work was supported by the Australian Research Council Centre of Excellence for Mathematical & Statistical Frontiers, under CE140100049 grant number.

A | CROSS-ENTROPY UPDATE

Lemma 1 *Let $\boldsymbol{p} = (p_1, \ldots, p_k)$, such that $\boldsymbol{p} \in [0,1]^k$ and let $\mathbf{x}_j = (x_{j,1}, \ldots, x_{j,k}) \in \{0,1\}^k$ for $1 \leq j \leq N$. Suppose that $S : \{0,1\}^k \to \mathbb{R}$, $\gamma \in \mathbb{R}$, and $N \in \mathbb{N}$. Then, the function*

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{1}_{\left\{S(\mathbf{x}_j) \leq \gamma\right\}} \ln \left(\prod_{i=1}^{k} p_i^{x_{j,i}} (1-p_i)^{1-x_{j,i}} \right) \tag{7}$$

is concave and differentiable with respect to $\boldsymbol{p}$.

Proof The functions $\ln(p_i)$ and $\ln(1-p_i)$ are concave for $1 \leq i \leq k$. To see this, note that:

$$\frac{d^2}{dp_i^2} \ln(p_i) = -\frac{1}{p_i^2}, \quad \frac{d^2}{dp_i^2} \ln(1-p_i) = -\frac{1}{(1-p_i)^2},$$

for $1 \leq i \leq k$. In addition, $\mathbb{1}_{\left\{S(\mathbf{x}_j) \leq \gamma\right\}}$ is non-negative and does not depend on $\boldsymbol{p}$ for $1 \leq j \leq N$. Finally, $(1 - x_{j,i})$ and

$x_{j,i}$ are non-negative, that is,

$$\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}\ln\left(\prod_{i=1}^{k}p_i^{x_{j,i}}(1-p_i)^{1-x_{j,i}}\right)$$
$$=\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}\left(\sum_{i=1}^{k}x_{j,i}\ln p_i+(1-x_{j,i})\ln(1-p_i)\right),$$

is a non-negative weighted sum of concave functions ($\ln(p_i)$ and $\ln(1-p_i)$). Therefore, (7) is concave and differentiable with respect to $\boldsymbol{p}$.

Lemma 2 *Let $\boldsymbol{p}=(p_1,\ldots,p_k)$, such that $\boldsymbol{p}\in[0,1]^k$ and let $\mathbf{x}_j=(x_{j,1},\ldots,x_{j,k})\in\{0,1\}^k$ for $1\leq j\leq N$. Suppose that $S:\{0,1\}^k\to\mathbb{R}$, $\gamma\in\mathbb{R}$, and $N\in\mathbb{N}$. Then, the function*

$$w(p_1,\ldots,p_k)=\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}\ln\left(\prod_{i=1}^{k}p_i^{x_{j,i}}(1-p_i)^{1-x_{j,i}}\right)$$

is maximized for

$$p_i^*=\frac{\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}x_{j,i}}{\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}}\quad\forall\,1\leq i\leq k.$$

Proof From Lemma 1, w is concave and differentiable with respect to $p_1,\ldots,p_k$. Thus, it is sufficient to solve

$$\nabla w=\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}\nabla\ln\left(\prod_{i=1}^{k}p_i^{x_i}(1-p_i)^{1-x_i}\right)=0.$$

It is not very hard to see that for all $1\leq i\leq k$, it holds that:

$$\frac{\partial}{\partial p_i}w=\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}\left(\frac{x_{j,i}}{p_i}-\frac{(1-x_{j,i})}{1-p_i}\right)=0\quad\Rightarrow\quad\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}\left(\frac{x_{j,i}(1-p_i)-p_i(1-x_{j,i})}{p_i(1-p_i)}\right)=0$$
$$\Rightarrow\quad\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}\left(\frac{x_{j,i}-p_i}{p_i(1-p_i)}\right)=0\quad\Rightarrow\quad\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}x_{j,i}-\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}p_i=0$$
$$\Rightarrow\quad p_i^*=\frac{\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}x_{j,i}}{\sum_{j=1}^{N}\mathbb{1}_{\left\{S(\mathbf{x}_j)\leq\gamma\right\}}}.$$

References

[1] B. Al-Anzi, P. Arpp, S. Gerges, C. Ormerod, N. Olsman, and K. Zinn, *Experimental and Computational Analysis of a Large Protein Network That Controls Fat Storage Reveals the Design Principles of a Signaling Network*, PLOS Comput. Biol. **11** (05 2015), 1–28.

[2] S. Asmussen and P. Glynn, *Stochastic Simulation: Algorithms and Analysis*, Stochastic Modelling and Applied Probability, Springer New York, 2007.

[3] P. Borisovsky and A. Eremeev, *Comparing evolutionary algorithms to the (1+1)-EA*, Theor. Comput. Sci. **403** (2008), 33–41.

[4] M. Captivo, J.C. Clímaco, and M.M. Pascoal, *A mixed integer linear formulation for the minimum label spanning tree problem*, Comput. operations research **36** (2009), 3082–3085.

[5] C. Cerrone, C. D'Ambrosio, and A. Raiconi, *Heuristics for the strong generalized minimum label spanning tree problem*, Networks **74** (2019), 148–160.

[6] R. Cerulli, A. Fink, M. Gentili, and S. Voß, "*Metaheuristics comparison for the minimum labelling spanning tree problem*," *The Next Wave in Computing, Optimization, and Decision Technologies*, B. Golden, S. Raghavan, and E. Wasil (eds.), Springer US, Boston, MA, 2005, pp. 93–106.

[7] R.S. Chang and L. Shing-Jiuan, *The minimum labeling spanning trees*, Informat. Process. Lett. **63** (1997), 277–282.

[8] A.M. Chwatal and G.R. Raidl, *Solving the Minimum Label Spanning Tree Problem by Ant Colony Optimization*, Proceedings of the 2010 International Conference on Genetic and Evolutionary Methods, July 12-15, Las Vegas Nevada, USA, 2010, pp. 91–97.

[9] A.M. Chwatal and G.R. Raidl, *Solving the Minimum Label Spanning Tree Problem by Mathematical Programming Techniques*, Adv. Oper. Res. **2011** (2011), 1–38.

[10] A.M. Chwatal, G.R. Raidl, and O. Dietzel, *Compressing Fingerprint Templates by Solving an Extended Minimum Label Spanning Tree Problem*, Proceedings of the Seventh Metaheuristics International Conference, June 25-29, Montreal, Canada, 2007.

[11] S. Consoli, K. Darby-Dowman, N. Mladenović, and J. Moreno Pérez, *Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem*, Eur. journal operational research **196** (2009), 440–449.

[12] S. Consoli, K. Darby-Dowman, N. Mladenović, and J.A. Moreno-Pérez, *Variable neighbourhood search for the minimum labelling steiner tree problem*, Ann. operations research **172** (2009), 71–96.

[13] S. Consoli, N. Mladenović, and J. Moreno Pérez, *Solving the minimum labelling spanning tree problem by intelligent optimization*, Appl. soft computing **28** (2015), 440–452.

[14] S. Consoli and J. Moreno Pérez, *Solving the minimum labelling spanning tree problem using hybrid local search*, Electron. notes discrete mathematics **39** (2012), 75–82.

[15] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 2nd edition 2001.

[16] T.G. da Silva, S. Gueye, P. Michelon, L.S. Ochi, and L. dos Anjos Formiga Cabral, *A polyhedral approach to the generalized minimum labeling spanning tree problem*, EURO J. Comput. Optim. **7** (March 2019), 47–77.

[17] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt, *Approximating Covering Problems by Randomized Search Heuristics Using Multi-Objective Models*, Evolutionary Computation **18** (2010), 617–633.

[18] T. Friedrich and F. Neumann, *Maximizing submodular functions under matroid constraints by evolutionary algorithms*, Evolutionary Computation **23** (2015), 543–558.

[19] X. Lai, Y. Zhou, J. He, and J. Zhang, *Performance Analysis of Evolutionary Algorithms for the Minimum Label Spanning Tree Problem*, IEEE Trans. Evolutionary Computation **18** (Dec 2014), 860–872.

[20] M. Laumanns, L. Thiele, and E. Zitzler, *Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions*, IEEE Trans. Evolutionary Computation **8** (April 2004), 170–182.

[21] D.J.C. MacKay, *Information Theory, Inference & Learning Algorithms*, Cambridge University Press, New York, NY, USA, 2002.

[22] F. Neumann and I. Wegener, *Minimum spanning trees made easier via multi-objective optimization*, Natural Comput. **5** (Sep 2006), 305–319.

[23] B.D. Ripley, *Stochastic Simulation*, John Wiley & Sons, Inc., New York, NY, USA, 1987.

[24] R.Y. Rubinstein, *Optimization of computer simulation models with rare events*, Eur. J. Oper. Res. **99** (1997), 89–112.

[25] R.Y. Rubinstein, *Cross-entropy and rare events for maximal cut and partition problems*, ACM Trans. Model. Comput. Simu **12** (2002), 27–53.

[26] R.Y. Rubinstein, A. Ridder, and R. Vaisman, *Fast Sequential Monte Carlo Methods for Counting and Optimization*, Wiley Publishing, Hoboken, New Jersey, 1st edition 2013.

[27] T.G.d. Silva, E. Queiroga, L.S. Ochi, L.d.A.F. Cabral, S. Gueye, and P. Michelon, *A hybrid metaheuristic for the minimum labeling spanning tree problem*, Eur. journal operational research **274** (2019), 22–34.

[28] D.J. Watts and S.H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature **393** (June 1998), 440–442.

[29] Y. Xiong, B. Golden, and E. Wasil, *A one-parameter genetic algorithm for the minimum labeling spanning tree problem*, IEEE transactions evolutionary computation **9** (2005), 55–60.

[30] Y. Xiong, B. Golden, and E. Wasil, *Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem*, Oper. Res. Lett. **33** (2005), 77–80.