

3005

REPRINT

Daniele Catanzaro and Raffaele Pesenti

Enumerating vertices of the balanced minimum evolution polytope



Computers and Operations Research, 109, 209-217, 2019

CORE
CENTER FOR OPERATIONS RESEARCH AND ECONOMETRICS

CORE

Voie du Roman Pays 34, L1.03.01

B-1348 Louvain-la-Neuve

Tel (32 10) 47 43 04

Email: immaq-library@uclouvain.be

https://uclouvain.be/en/research-institutes/lidam/core/reprints..html

Computers and Operations Research 109 (2019) 209–217

Contents lists available at ScienceDirect

Computers and Operations Research

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

Enumerating vertices of the balanced minimum evolution polytope

Daniele Catanzaro [a,b,c,*], Raffaele Pesenti [c]

[a] Center for Operations Research and Econometrics (CORE), Université Catholique de Louvain, Voie du Roman Pays 34, Louvain-la-Neuve B-1348, Belgium
[b] Luxembourg Institute of Socio-Economic Research (LISER), 11 Porte des Sciences, Esch-sur-Alzette L-4366, Luxembourg
[c] Department of Management, Università Ca' Foscari, San Giobbe, Cannaregio 837, Venezia I-30121, Italy

ARTICLE INFO

Article history:
Received 12 July 2018
Revised 14 March 2019
Accepted 1 May 2019
Available online 6 May 2019

Keywords:
Unrooted binary tree enumeration
Parallel enumeration
Phylogenetics
Balanced minimum evolution polytope
Tree codes
Prüfer coding

ABSTRACT

Recent advances on the polyhedral combinatorics of the *Balanced Minimum Evolution Problem* (BMEP) enabled the identification of fundamental characteristics of the convex hull of the BMEP (or *BMEP polytope*) as well as the description of some of its facets. These results have been possible thanks to the assistance of general purpose softwares for polyhedral analyses, such as Polymake, whose running times however become quickly unpractical. In this article, we address the problem of designing tailored algorithms for the study of the BMEP polytope, with special focus on the problem of enumerating its vertices. To this end, by exploiting some results from the polyhedral combinatorics of the BMEP and a particular encoding scheme based on Prüfer coding, we present an iterative enumeration algorithm that is markedly faster than current general purpose softwares and that can be natively massively parallelized.

© 2019 Elsevier Ltd. All rights reserved.

1. Introduction

Consider a set $\Gamma = \{1, 2, \ldots, n\}$ of $n \geq 4$ vertices, hereafter called *taxa*. A *phylogeny* T of Γ is an unrooted binary tree having Γ as leaf-set. By definition, a phylogeny of Γ has $(n-2)$ *internal vertices* having degree 3 and an overall number of $(2n-3)$ edges, n of which are *external*, i.e., incident to a taxon, and $(n-3)$ are *internal*, i.e., incident to two internal vertices. As an example, Fig. 1 shows a possible phylogeny of a set Γ of five taxa.

Let $\mathcal{T}$ be the set of the possible distinct phylogenies of Γ. The cardinality of $\mathcal{T}$ is a function of the cardinality of Γ and is proved to be equal to $(2n-5)!! = 1 \times 3 \times 5 \ldots \times (2n-5)$ (Felsenstein, 2004). Given a phylogeny T of Γ and a taxon $i \in \Gamma$, let Γ_i be the set $\Gamma \backslash \{i\}$ and let τ_{ij} be the *topological distance* between taxa $i, j \in \Gamma$, $i \neq j$, i.e., the number of edges belonging to the (unique) path from taxon i to taxon j in T. For example, by referring to the phylogeny shown in Fig. 1, $\tau_{12} = 2$, $\tau_{14} = 4$ and $\tau_{54} = 3$. Consider a $n \times n$ symmetric distance matrix $\mathbf{D} = \{d_{ij}\}$, whose generic entry d_{ij} represents a measure of dissimilarity between the corresponding pair of taxa $i, j \in \Gamma$. Then, the *Balanced Minimum Evolution Problem* (BMEP) consists in solving the following combinatorial optimization problem (Catanzaro et al., 2012):

$$\min_{T \in \mathcal{T}} \quad \sum_{i \in \Gamma} \sum_{j \in \Gamma_i} \frac{d_{ij}}{2^{\tau_{ij}}}. \tag{1}$$

The BMEP was introduced in the literature on phylogenetics in (Pauplin, 2000) and systematically investigated from a biological perspective in (Desper and Gascuel, 2004) and (Gascuel and Steel, 2006). The problem is $\mathcal{NP}$-hard and inapproximable within c^n, for some constant $c > 1$, unless $\mathcal{P} = \mathcal{NP}$ (Fiorini and Joret, 2012). This fact has justified the development of a number of implicit enumeration algorithms to exactly solve the problem (Aringhieri et al., 2011; Catanzaro et al., 2012; Pardi, 2009) as well as heuristics to approximate its optimal solution (Catanzaro, 2009; Gascuel, 2005; Pardi, 2009). The current state-of-the-art exact solution algorithm for the BMEP, described in (Catanzaro et al., 2012), is based on an Integer Linear Programming (ILP) model that exploits some combinatorial connections between the BMEP and the *Huffman Coding Problem* (Huffman, 1952; Parker and Ram, 1996; Sayood, 2017). Such connections proved fundamental to identify a number of properties that characterize phylogenies, such as *Kraft equalities* or the *Third Equality* (Catanzaro et al., 2012), and had a central role in the development of tight lower bounds on the optimal solution to the BMEP. The algorithm is however unable to solve instances of the BMEP containing more than two dozen taxa within one hour computing time. Hence, in the attempt to generate improved exact solution algorithms for the problem, particular

* Corresponding author.
E-mail addresses: daniele.catanzaro@uclouvain.be (D. Catanzaro), pesenti@unive.it (R. Pesenti).

https://doi.org/10.1016/j.cor.2019.05.001
0305-0548/© 2019 Elsevier Ltd. All rights reserved.

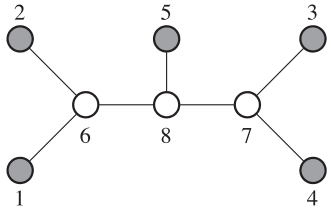


Fig. 1. An example of a phylogeny of the set $\Gamma = \{1, 2, 3, 4, 5\}$.

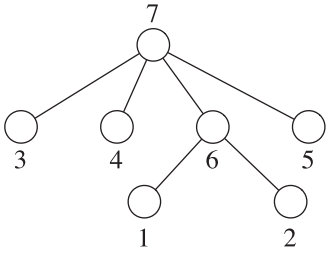


Fig. 2. An example of a generic labeled tree with 7 vertices. Leaves are labeled with integers from 1 to 5 and and internal vertices with integers from 6 to 7. The Prüfer code associated to this tree is [6,6,7,7,7,7].

attention has been recently devoted to the study of its polyhedral combinatorics. The earliest attempt started with Haws et al. (2011), who first characterized some faces of the convex hull of the BMEP (hereafter called the *BMEP polytope*) in the space of the topological distances. Due to the high nonlinearity of Kraft equalities, such a space quickly appeared inappropriate for the purpose, hence Forcey et al. (2016) suggested to study the BMEP polytope in a space in which a linearization of Kraft equalities was possible. The authors proposed the use of Polymake (Gawrilow and Joswig, 2000) to assist their study. This empirical approach proved successful to characterize all of the facets of the BMEP polytope for five taxa as well as some of them for $n \geq 6$ taxa (Forcey et al., 2016). Moreover, it helped to unveil new connections between the BMEP polytope and the *permutoassociahedron* (Billera et al., 2001; Forcey et al., 2017; Kapranov, 1993; Reiner and Ziegler, 1993). The hardness in carrying out this analysis, however, increases with the number of input taxa. For example, already for six taxa, the computing time taken by Polymake both to enumerate all of the feasible solutions to the problem and to generate 90262 facets of the corresponding polytope overcomes one day on a MacBook Pro running MacOS 10.12.6 and equipped with an Intel Core i5, 3.1 GHz, and 8 GB Ram. This fact suggests the search for tailored algorithms to assist the analysis of the polytope. In this article, we focus on the problem of enumerating all of the feasible solutions to the BMEP. In this context, it is worth noting that recent extensions of the results described in Forcey et al. (2016) showed, among others, that the vertices of the BMEP polytope correspond to all (and only) the solutions to the problem (Catanzaro et al., submitted, 2019). Because of this bijection, a tailored enumeration of all of the possible phylogenies of Γ may prove more effective than a general purpose enumeration. Inspired by this observation, in this article we develop a possible tailored iterative algorithm for this purpose. The algorithm exploits a particular Prüfer coding scheme to encode phylogenies and is designed to be massively parallelized. Our hope is that the combined use of a tailored vertex enumeration and a tailored facet generation may better assist the analysis of the polyhedral combinatorics of the BMEP and ultimately pay off in improved exact solution algorithms.

2. Codes and phylogenies

As shown in Catanzaro et al. (submitted, 2019), enumerating the vertices of the BMEP polytope is equivalent to enumerate all of the possible phylogenies of a given set of n taxa. This task becomes quickly intractable even for small values of n (e.g., for $n = 12$, $\mathcal{T}$ already contains 316234143225 phylogenies). Hence, to perform this task it is highly desirable to use both efficient data structure to encode phylogenies as well as fast enumerative routines that may be possibly easy to parallelize. Interestingly, *Prüfer codes* provide an efficient way to encode and decode labeled trees (Caminiti et al., 2007). Specifically, given an acyclic graph with m vertices, the corresponding Prüfer code is defined to be a sequence of vertices $s = [s_1, s_2, \ldots, s_{m-1}]$ built by progressively deleting the leaf with the smallest label and appending to the code the label of its parent. For example, the Prüfer code of the tree shown in Fig. 2 is [6,6,7,7,7,7]: at the first step vertex '1' is deleted from the tree and label '6' is added to the code; similarly, vertex '2' is deleted from the tree and label '6' is added to the code; then, vertex '3' is deleted from the tree and label '7' is added to the code; subsequently, vertex '4' is deleted from the tree and label '7' is added to the code; then, vertex '5' is deleted from the tree and label '7' is added to the code; finally, vertex '6' is deleted from the tree and label '7' is added to the code. Caminiti et al. (2007) provided an algorithm to iteratively perform this task as well as an algorithm to decode the Prüfer code so obtained, in order to uniquely reconstruct the original tree. For the sake of completeness, we report them in Algorithms 1 and 2, respectively, and we refer the interested reader to their article for supplementary information.

Algorithm 1: Encode - This algorithm, described in Caminiti et al. (2007), computes the Prüfer code corresponding to a labeled input tree with m vertices; the tree is encoded by means of an adjacency list L; $L[v]$ denotes the adjacency list of a vertex v of T; the degree of each vertex is assumed to be known.

```
Input  : a labeled tree T with m vertices
Output : the Prüfer code induced by T
Data   : code is a list of m − 1 integers representing the Prüfer code
for each vertex v = 1 to m do
    if degree [v] = 1 then
        let u be the unique vertex in L[v];
        code +={u};
        degree [u]−−;
    while degree [u] = 1 and u < v do
        let z be the unique node in L[u];
        code +={z};
        degree [z]−−;
        u=z;
return code;
```

We observe that the value of the last entry of a Prüfer code is always equal to the vertex with maximum label, hence the code length can be reduced to $(m - 2)$. For example, the Prüfer code of the tree shown in Fig. 2 can be reduced to [6,6,7,7,7]. Moreover, we also observe that Prüfer codes can be appropriately adapted to deal with phylogenies. Specifically, as phylogenies are acyclic graphs having each internal vertex of degree 3, Prüfer codes of phylogenies can be thought as sequences of internal vertices in which each vertex appears exactly twice, except one that appears three times. More formally, let V_i be the set of internal vertices of a phylogeny of a set Γ of n taxa. Assume that taxa are labeled from 1 to n and that the internal vertices are labeled with positive integers from $n + 1$ to $n + k$, where $k = |V_i| = (n - 2)$. Since n is fixed and known, with a little abuse of notation we can relabel the internal vertices with positive integers from 1 to k. Then, we can define the Prüfer code of a phylogeny as follows.

Algorithm 2: Decode - This algorithm, described in Caminiti et al. (2007), reconstructs the tree encoded by a given Prüfer code. If the input code is a PCP, then n must be summed to each entry of the code just before line 3, in order to properly reconstruct the corresponding phylogeny.

```
Input  : a Prüfer code
Output : the tree T encoded by code
Data   : T is a list of edges that encodes the tree to reconstruct; leaves is
         the set of terminal vertices of T;
1  T = {};
2  if code has m − 2 integers then add the integer m in position m − 1 of code;
3  set leaves = {1, ..., r − 1}, where r is the smallest integer in code;
4  for i = 1 to m − 1 do
5      u = code[i];
6      let v be the smallest integer in leaves;
7      add edge {u, v} to T;
8      delete v from leaves;
9      if code[i] is the rightmost occurrence of u in code then
10         leaves+ = {u};
11 return T;
```

Definition 1. The *Prüfer Code of a Phylogeny* (PCP) T of n taxa is a sequence s of $(2k+1)$ integers in the ordered set $\Sigma = \{1, 2, \ldots, k\}$ obtained from T by means of Algorithm 1.

For example, the Prüfer code of the phylogeny shown in Fig. 1 is [1,1,2,2,3,3,3]. The first $(k+2)$ integers in a PCP refer to external edges of T and the remaining ones refer to internal edges. Moreover, even when dealing with phylogenies the last integer of the code is always equal to k, hence the code length can be reduced of one unit, leading to [1,1,2,2,3,3]. We observe that, due to the relabeling of the internal vertices before mentioned, a PCP can be uniquely reconstructed by using Algorithm 2, provided that the integer $n = |\Gamma|$ is summed to each entry of the code just before executing line 3 of the algorithm.

3. Vertex enumeration as Prüfer code enumeration

In this section we present an algorithm to enumerate all of the possible Prüfer codes corresponding to all of the possible phylogenies of a given set Γ of n taxa. Because each phylogeny of Γ is a vertex of the BMEP polytope, this algorithm acts as a vertex enumerator. Before proceeding, we introduce some definitions that will prove useful throughout the article.

Definition 2. Given a positive integer k, let $s = [s_1, s_2, \ldots, s_{2k}]$ and $\hat{s} = [\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_{2k}]$ be two PCPs. Then, s is said to be equal to $\hat{s}$, in symbols $s = \hat{s}$, if $s_i = \hat{s}_i$, for all $i \in \{1, 2, \ldots, 2k\}$, and different, in symbols $s \neq \hat{s}$, otherwise.

Due to the determinism of Algorithm 1, once labeled the internal vertices of a phylogeny T, there exists an unique Prüfer code for T and vice versa (Caminiti et al., 2007). However, since the labeling process is arbitrary, there exist $(2k)!/2^k$ different PCPs corresponding to phylogenies of n taxa, some of which are *equivalent* in the following sense:

Definition 3. Given a positive integer k, let $s = [s_1, s_2, \ldots, s_{2k}]$ and $\hat{s} = [\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_{2k}]$ be two PCPs. Then, s is said to be equivalent to $\hat{s}$, in symbols $s \sim \hat{s}$, if there exists a bijective function ϕ: $\Sigma \to \Sigma$ such that $\phi(s_i) = \hat{s}_i$, $i \in \{1, 2, \ldots, 2k\}$.

For example, the Prüfer codes relative to the phylogenies shown in Fig. 3(a) and (b) are $s = [1, 1, 2, 2, 3, 3]$ and $\hat{s} = [2, 2, 1, 1, 3, 3]$, respectively. It is easy to see that the two codes are equivalent. In fact, there exists a bijection function ϕ defined as follows: $\phi(1) \to 2$, $\phi(2) \to 1$, $\phi(3) \to 3$. The presence of equivalent PCPs involves a non negligible computational overhead for a vertex enumeration algorithm. Hence, it may be desirable to enumerate only the $(2k)!/k!2^k$ non-equivalent PCPs obtainable with $2k$ symbols. To this end, we introduce the following definitions that will prove useful to design an algorithm that enumerates only non-equivalent PCPs.

Definition 4. Given a positive integer k, a mapping function is a function $f : \Sigma \to \{1, 2, \ldots, 2k-1\}$ such that, for any $\sigma \in \Sigma$, $1 \leq f(\sigma) \leq 2k - (2\sigma - 1)$.

Definition 5. Given a mapping function f, a PCP mapper m_f is an algorithm that generates a PCP according to Algorithm 3.

Algorithm 3: PCP Mapper - This algorithm computes a PCP according to a given input mapping function.

```
Input  : k: the number of internal vertices; f, a mapping function
Output : a Prüfer code
Data   : first is a positive integer between 1 and 2k − 1;
         s is an array of 2k integers
1  set s equal to [−1, −1, ..., −1];
2  for σ ∈ Σ do
3      first =index of s with entry equal to -1;
4      s[first] = σ;
5      first =the f(σ)-th index of s with entry equal to -1;
6      s[first] = σ;
7  return s;
```

For example, assume $k = 4$ and consider the following mapping function $f(1)$=5, $f(2)$=5, $f(3)$=2, $f(4)$=1. Then, Algorithm 3 will generate the following code s: at the first iteration s = [1,-1,-1,-1,-1,1,-1,-1]; subsequently, at the second iteration s = [1,2,-1,-1,-1,1,-1,2]; at the third iteration s = [1,2,3,-1,3,1,-1,2]; finally, at the fourth iteration $s = [1, 2, 3, 4, 3, 1, 4, 2]$.

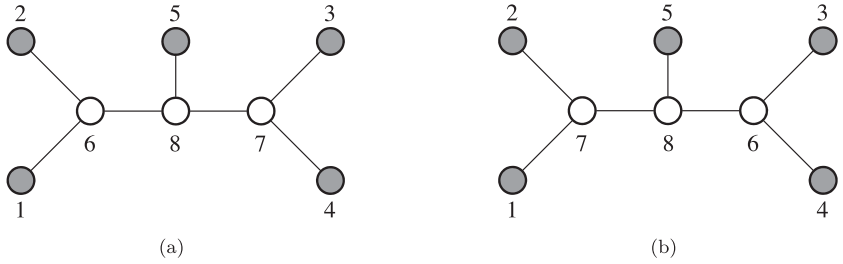


Fig. 3. Two possible examples of a phylogeny of the set $\Gamma = \{1, 2, 3, 4, 5\}$ of five taxa. The Prüfer code for the phylogeny on the left is [6,6,7,7,8,8] and the one for the phylogeny on the right is [7,7,6,6,8,8]. Observe that the internal vertices are labeled from $n+1$ to $2n-2$ and that n is fixed and known. Thus, provided an appropriate relabeling of the internal vertices as integers in 1 to $n-2$, both codes can be rewritten as [1,1,2,2,3,3] and [2,2,1,1,3,3], respectively. It is worth noting that, independently of the relabeling, the two codes are equivalent.

Denote s_f as the PCP generated by a PCP mapper m_f and s_i^f as the i-th entry of s_f. Then, the following properties hold for PCP mappers.

Proposition 1. *Let f and g be two different mapping functions. Then, m_f and m_g return $s_f \neq s_g$.*

Proof. Let $\sigma_{\hat{j}}$ the first symbol in Σ such that $f(\sigma_{\hat{j}}) \neq g(\sigma_{\hat{j}})$. This symbol exists as f differs from g. Then, the assignments generated by the PCP mappers m_f and m_g at lines 3 to 6 of Algorithm 3 are equal in the first $(\hat{j}-1)$ iterations. However, at iteration $\hat{j}$ the assignments at lines 5 and 6, are different. Thus, the resulting PCPs are different. □

Proposition 2. *Let f and g be two different mapping functions. Then, m_f and m_g return $s_f \not\sim s_g$.*

Proof. By contradiction, assume that $s_f \sim s_g$, i.e., that there exists a bijective function ϕ such that $\phi(s_i^f) = s_i^g$, for all $i = 1, 2, \ldots, 2k$. Since PCP mappers always assign the first instance of 1 in the first position of the code, we have that $s_1^f = s_1^g = 1$ and hence $\phi(1) = 1$. Moreover, denote x (respectively y) as the position at which the second instance of 1 appears in s_f (respectively s_g). Since s_f and s_g are equivalent, the following conditions must hold $\phi(s_x^f) = \phi(1) = 1 = s_y^g$ and $x = y$. Now, remove the two occurrences of 1 from both s^f and s^g. Consider the two occurrences of 2. Again, the first instance of 2 appears in the first position of the two codes, then $\phi(2) = 2$. Redefine x (respectively y) as the position at which the second instance of 2 appears in s_f (respectively s_g). Since s_f and s_g are equivalent, the following conditions must hold $\phi(s_x^f) = \phi(2) = s_y^g = 2$ and $x = y$. Remove the two occurrence of 2 from both codes and iterate this argument for the remaining $\sigma_j \in \Sigma$, $j > 2$. We obtain that $\phi(\sigma_j) = \sigma_j$, for all $\sigma_j \in \Sigma$. However, this implies that the mapping functions f and g are equal which in turn contradicts the hypothesis. □

Proposition 3. *For a fixed positive integer k, there are at most $(2k)!/k!2^k$ different PCP mappers.*

Proof. Let $s = [s_1, s_2, \ldots, s_{2k}]$ be a PCP and denote σ_j as the j-th integer of Σ. Since a PCP mapper assigns the first instance of 1 independently of the mapping function, then there are only $(2k-1)$ possible ways to assign the second instance of 1 in s. Similarly, once assigned both the instances of 1 in s, there are only $(2k-3)$ possible ways to assign the second instance of 2 in s. In general, once assigned both the instances of σ_{j-1}, $j < (k-1)$, there are only $2(k-j)+1$ possible ways to assign the second instance of σ_j in s. Thus, there are at most

$$\prod_{j=1,\ odd\ j}^{2k-3} (2k-j) = (2k-1)!! = \frac{(2k)!}{k!2^k}$$

different PCP mappers for a fixed positive integer k. □

It is worth noting that in a PCP mapper the first instance of any $\sigma_j \in \Sigma$ is assigned independently of the mapping function. Hence, a PCP is completely defined by the position of the second instance of each integer $\sigma_j \in \Sigma$. For example, the PCP [1,2,1,3,2,3] is completely defined by the second instance of 1 in the third position of the code, the second instance of 2 in the fifth position, and second instance of 3 is in the sixth position. This insight suggests as possible approach to enumerating all possible phylogenies of a given set Γ of n taxa the generation of all possible $(2k)!/k!2^k$ distinct PCP mappers obtainable for $k = n-2$. Such approach can be performed by means of Algorithm 4.

The algorithm uses two data structures: s, defined as vector of $2k$ integers, representing a PCP and pos, defined as a vector of k integers, periodically storing at the i-th entry the position of the second symbol i in s. We assume that arrays are indexed from 1. Initially, s is set to $[1, 1, 2, 2, \ldots, k, k]$ and pos is set to $[2, 4, 6, \ldots, 2k]$. The dominant operation of the algorithm is represented by the main loop whose core consists of swapping two integers of s (the Swap procedure). For example, starting from the code [1,1,2,2,3,3], after the swap, the code [1,2,1,2,3,3] is obtained. Note that, the new code differs from the previous one for the fact that the leaves labeled with '2' and '3', previously linked to the internal vertices $n+1$ and $n+2$, are now linked to the internal vertices $n+2$ and $n+1$, respectively. After executing the swap, the entries of the vector pos are appropriately updated. For example, after the first swap the vector pos becomes [3,4,6]. The swapping procedure is executed until integer 1 is found in position $2k$. Subsequently, the if-condition of line 4 is activated and s is appropriately reset. Specifically, ShiftCode (whose pseudo-code is shown in Algorithm 5) takes care of shifting right the entries of s from position $2s_{2k}+1$ to $2k$ and setting the $(2s_{2k})$-th entry of s to s_{2k}. The shifting operation is repeated until the integer s_{2k} is not equal to one plus s'_{2k}, where s'_{2k} is the integer that occupied the entry $2k$ of s before the last shift operation. Note that, inside the if-condition, also the entries of pos are consequently updated. An example of all the PCPs generated by Algorithm 4 when $n = 6$ is shown in Table 1.

Algorithm 4: Enumerator - This algorithm enumerates all of the possible Prüfer codes corresponding to all of the possible phylogenies of a set Γ of n taxa.

Input : k : number of internal vertices of a phylogeny of Γ of n taxa (i.e., $n-2$)
Output : Print out the PCPs for n taxa are print out
Data : s is an array of $2k$ integers representing a PCP
pos is an array of k integers periodically storing at the i-th entry the position of the second symbol i in s
r is an integer

```
1  set s = [1, 1, 2, 2, ..., k, k] and pos = [2, 4, 6, ..., 2k];
2  while true do
3      if pos[1] == 2k then
4          r =1;
5          repeat
6              pos[s[2k]] = 2s[2k];
7              ShiftCode;
8              r++;
9          until s[2k] ≠ r;
10         if r ≥ k then
11             return;
12         Swap(s[pos[r]], s[pos[r] + 1]);
13         pos[r]++;
14     PrintCode();
15     for i = 2 to 2k − 1 do
16         Swap(s[i], s[i + 1]);
17         pos[1]++;
18         PrintCode();
```

Algorithm 5: ShiftCode - This algorithm is a routine used in Algorithm 4; it shifts right the entries of a given Prüfer code.

Data: acc is a positive integer;

```
1  acc = s[2k];
2  if acc < k then
3      for c = 2k to 2acc + 1 do
4          s[c] = s[c − 1];
5          c − −;
6      s[2acc] = acc;
```

Proposition 4. *Given a positive integer $k > 1$, Algorithm 4 enumerates $(2k)!/k!2^k$ PCPs.*

Table 1
The PCPs generated by Algorithm 4 for an input of six taxa.

Iteration	PCP	Iteration	PCP
01	[1,1,2,2,3,3]	09	[1,2,3,2,1,3]
02	[1,2,1,2,3,3]	10	[1,2,3,2,3,1]
03	[1,2,2,1,3,3]	11	[1,1,2,3,3,2]
04	[1,2,2,3,1,3]	12	[1,2,1,3,3,2]
05	[1,2,2,3,3,1]	13	[1,2,3,1,3,2]
06	[1,1,2,3,2,3]	14	[1,2,3,3,1,2]
07	[1,2,1,3,2,3]	15	[1,2,3,3,2,1]
08	[1,2,3,1,2,3]		

Proof. Starting from the initial code s, the second instance of the integer 1 is swapped $(2k-1)$ times before arriving in position $2k$. Note that each swap i is equivalent to the mapping function $f(1) = i$, $f(\sigma) = 1$, for all $\sigma \in \Sigma$, $\sigma \neq 1$. Subsequently, the if-condition at lines 5–16 is executed, and inside loop 7–11, 1 is reset in position 2 of s. Unless $\Sigma = \{1, 2\}$ (in which case the algorithm ends), loop 7–11 is repeated once after which the second instance of 2 is swapped with the adjacent integer in s. Note that this peculiar swap is equivalent to the mapping function $f(2) = 5$, $f(\sigma) = 1$, for all $\sigma \in \Sigma$, $\sigma \neq 2$. This process is iterated $(2k-1)(2k-3)$ times until the second instances of 1 and 2 are located in s_{2k} and s_{2k-1}, respectively. In that case, unless $\Sigma = \{1, 2, 3\}$ (in which case the algorithm ends), loop 7–11 is repeated twice after which the second instance of 3 is swapped with the adjacent integer in s. Note that this swap is equivalent to the mapping function $f(3) = 7$, $f(\sigma) = 1$, for all $\sigma \in \Sigma$, $\sigma \neq 3$. In general, iterating this process,

$$\prod_{i=1,\ odd\ i}^{2k-3} (2k-i) = (2k-1)!! = \frac{(2k)!}{k!2^k}$$

swaps are performed before that the second instances of $1, 2, \ldots, k-1$ are located in $s_{2k}, s_{2k-1}, \ldots, s_{k+1}$, respectively. It is easily seen that the swaps correspond to the $(2k)!/k!2^k$ possible PCP mappers obtainable with n taxa. Once the PCP $s = [1, 2, 3, \ldots, k-1, k, k, k-1, \ldots, 2, 1]$ is obtained, loop 7–11 is repeated k times and the algorithm stops. □

4. Computational results

As mentioned in the introduction, the enumeration of all possible vertices of the BMEP polytope via general purpose softwares, such as Polymake (Gawrilow and Joswig, 2000) or PORTA (Christof and Loebel, 2015), is particularly slow. This fact is mainly due to a twofold reason. First, the representation of the feasible vectors in the τ space or in (Forcey et al., 2016) requires to store, for each pair of taxa $i, j \in \Gamma$, $i < j$, the value relative to the length of the path from taxon i to taxon j. In a phylogeny, this length can assume any value in the discrete interval $\{2, \ldots, n-1\}$. Hence, a feasible vector in both spaces must be encoded as a $(n-1)(n-2)(n-2)/2$ boolean vector, by introducing a redundancy with respect e.g., to Prüfer coding. Second, the dimension of the τ space or Forcey *et al.*' space grows quickly in function of n by overcoming, already for $n = 6$, the threshold (roughly 20) beyond which the general purpose enumeration procedures of the feasible vectors become unpractical. In this section, we show that the replacement of these procedures with specific implementations of Algorithm 4 may appreciably improve this situation.

A first implementation that we have considered in our computational experiments consists of a hard-coded version of Algorithm 4 written in C, compiled via the Intel Compiler Profor Linux version 13.0.1.117, and run on an workstation equipped with a 32 cores Intel Xeon E5-2667v3, 3.20 GHz, 256 GB RAM, 2 TB nfs, and CentOS release 6.9 (Final). We refer to this implementation as the "standard enumeration" procedure. In order to evaluate the overhead introduced by the PrintCode routine at lines 14 and 18 of Algorithm 4, we have also considered a modified version of the standard enumeration procedure, in which such calls have been simply omitted. We refer to this implementation as the "pure enumeration" procedure. Table 2 shows the computing times taken by both implementations to enumerate all the PCPs for increasing numbers of taxa.

The table shows that the pure enumeration procedure of all of the PCPs can be performed within 15 minutes computing time up to 14 taxa. From 15 taxa on, the pure enumeration procedure takes more than 5 hours. When, the routine PrintCode is introduced in Algorithm 4, the computing times markedly increases: already with 12 taxa the overall computing time of the standard enumeration procedure approaches 23 minutes and overcomes 3 days from 14 taxa on. The computational overhead introduced by the output handling can be dramatically reduced by means of parallelism (Petersen and Arbenz, 2004). Specifically, the particular iterative nature of Algorithm 4 enables an intrinsic massive parallelism that can be implemented by (i) appropriately setting the starting code and pos array for each CPU and (ii) by interrupting the main loop of Algorithm 4 running on each CPU once a specific stop condition is met. For example, if Algorithm 4 is parallelized on two CPUs, then one CPU can set its starting code and pos array equal to $[1, 1, 2, 2, \ldots, 2k, 2k]$ and $[2, 4, 6, \ldots, 2k]$, respectively; similarly, the second CPU can set its starting code and pos array equal to $[1, 2, 3, \ldots, k, 1, 2, 3, \ldots, k]$ and $[k+1, k+2, k+3, \ldots, 2k]$, respectively. The stop conditions for both CPUs then become the completion of the enumeration of $(2n-5)!!/2$ PCPs for the first CPU and $(2n-5)!!/2+1$ PCPs for the second CPU, respectively. It is quite straightforward to generalize these rules on Q CPUs. For the sake of completeness, we show in Algorithm 6 a simple routine that enables the computation of the z-th PCP to be assigned as a starting code to each available CPU.

Algorithm 6: ComputeCode - This algorithm computes the z-th PCP.

Input : an integer z corresponding to the z-th Prüfer code to generate
Output : the z-th Prüfer code
Data : s is an array of $2k$ integers representing a PCP
pos is an array of k integers storing at the i-th entry the position of the second symbol i in s
temp and newpos are a support integer and a support array of k integers, respectively

```
1  set s = [1, 1, 2, 2, ..., k, k] and pos = [2, 4, 6, ..., 2k];
2  newpos [1]= (z -1) mod (2k -1);
3  for i = 2 to k − 1 do
4      c = 1;
5      for j = 1 to 2i − 3 by 2 do
6          c = c(2k − j);
7      newpos[i] = ((z − 1)/c) mod (2k − (2i − 1));
8  for i = k − 1 to 1 do
9      temp = s[pos[i]];
10     for c = pos[i] to pos[i] + newpos[i] do
11         s[c] = s[c + 1];
12     s[pos[i] + newpos[i]] = temp;
13 return s;
```

The algorithm follows the same rationale at the core of Algorithm 4. Specifically, line 1 sets the initial PCP to $[1, 1, 2, 2, \ldots, k, k]$ and the pos array to $[2, 4, 6, \ldots, 2k]$. Then, starting from the initial PCP, lines 2–7 compute how many swaps must be carried out by Algorithm 4 on each second instance of each integer from 1 to $k-1$ before producing the z-th PCP. Specifically, at line 2, the position of the second instance of the integer 1 is computed by taking into account the fact that Algorithm 4 resets the second instance of integer 1 to its initial position every

Table 2
Computing times (in seconds) taken by a number of implementations of Algorithm 4 to enumerate all of the PCPs for increasing numbers of taxa. The implementations can be grouped into two families called "Single-Core Enumeration" and "Multi-Core Enumeration", respectively, depending on whether they make use of parallelism or not. The implementations within a family can be "standard", "pure" or "brute-force". The standard implementation refers to the translation in C of Algorithm 4. The pure implementation refers to a modified version of the standard implementation in which lines 14 and 18 of Algorithm 4 have been omitted. The brute-force implementation refers to a modified version of the standard implementation in which lines 14 and 18 of Algorithm 4 have been replaced by two calls to Algorithm 7 (see text for details). The value ">259200" refers to runs that took more than 3 days to complete the respective enumerations.

	Single-Core Enumeration			Multi-Core Enumeration			
Number of Taxa	Number of Phylogenies	Pure	Standard	Brute-Force	Pure	Standard	Brute-Force
5	15	$1.200{\cdot}10^{-5}$	$1.010{\cdot}10^{-4}$	$9.700{\cdot}10^{-5}$	$3.250{\cdot}10^{-2}$	$5.240{\cdot}10^{-2}$	$2.258{\cdot}10^{-2}$
6	105	$1.900{\cdot}10^{-5}$	$3.750{\cdot}10^{-4}$	$1.210{\cdot}10^{-4}$	$3.510{\cdot}10^{-2}$	$5.500{\cdot}10^{-2}$	$4.163{\cdot}10^{-2}$
7	945	$2.900{\cdot}10^{-5}$	$5.624{\cdot}10^{-3}$	$8.510{\cdot}10^{-4}$	$4.112{\cdot}10^{-2}$	$6.234{\cdot}10^{-2}$	$5.059{\cdot}10^{-2}$
8	10395	$5.100{\cdot}10^{-5}$	$5.095{\cdot}10^{-2}$	$9.214{\cdot}10^{-3}$	$5.620{\cdot}10^{-2}$	$6.703{\cdot}10^{-2}$	$6.864{\cdot}10^{-2}$
9	135135	$4.070{\cdot}10^{-4}$	$8.078{\cdot}10^{-1}$	$1.152{\cdot}10^{-1}$	$5.722{\cdot}10^{-2}$	$1.063{\cdot}10^{-1}$	$8.041{\cdot}10^{-2}$
10	2027025	$4.999{\cdot}10^{-3}$	3.558	2.718	0.084	0.982	0.397
11	34459425	$8.699{\cdot}10^{-2}$	65.692	62.397	0.167	16.52	6.656
12	654729075	1.644	1368.48	1445.79	0.654	347.1	163.9
13	13749310575	34.237	36336	35857	14	8684	3716
14	316234143225	783.364	>259200	>259200	269	199700	100000
15	7905853580625	19565.8	>259200	>259200	7677	>259200	>259200

$(2k-1)$ swaps. A similar argument is used in the loop at lines 3–7 to determine the position of the remaining $k-1$ integers. The positions of the second instances of the $k-1$ integers are then stored in the newpos array. Finally, the loop at lines 8–12 simply performs the swaps stored in the newpos array on the initial PCP, by providing as output the searched z-th PCP. It is worth noting that the particular data structures used in Algorithm 4 make its parallelization suitable in both a shared memory and a distributed memory environment. Because the workstation used in our experiments constitutes a shared memory environment, we opted for a parallel implementation of the standard and the pure enumeration procedures based on OpenMP (Petersen and Arbenz, 2004). The last three columns of Table 2 (namely, the "Multi-Core Enumeration" columns) show the computing times taken by both implementations to enumerate all the PCPs for increasing numbers of taxa. The table shows that for small numbers of taxa ($n \leq 10$) parallelism is not always worth due to the computational overhead introduced by multi-threading. From $n \geq 11$ on, however, the table shows the dramatic benefits of parallelism: the computing time reduction with respect to the single-core implementations may reach up to 65%. This trend is graphically visible in Figs. 4 and 5.

A question that naturally arises concerns the possibility to compute the topological distances between the taxa belonging to the phylogeny encoded by a given PCP. This possibility could allow the computation of the corresponding objective function value (1) in a given instance of the BMEP and enable, more in general, the exact resolution of the instance by brute force enumeration. An approach to perform this task consists in replacing lines 14 and 18 of Algorithm 4 with two respective calls to Algorithm 7 , which computes the desired topological distances by exploring n times the phylogeny encoded by s. Specifically, the algorithm uses two integer matrices: a $n \times n$ matrix $\mathbf{\Theta}$, whose generic element Θ_{ij} represents the topological distance τ_{ij} between taxa i and j in s; and an integer matrix M having k rows corresponding to the k internal vertices of the phylogeny encoded by s, and 6 columns codifying the following information: for each row r of $\mathbf{M}$ the first three entries encode the three vertices adjacent to r; the fourth entry encodes the topological distance of r from a reference leaf i; the fifth entry encodes the last vertex visited before reaching r; and finally the sixth entry encodes the number of times that vertex r is visited.

Proposition 5. *Algorithm 7 computes the topological distances between taxa in the phylogeny encoded by a PCP in $O(n^2)$ time.*

Algorithm 7: Topological Distance Calculator - This algorithm computes the topological distances between taxa in the phylogeny encoded by a given Prüfer code.

Input : s : a PCP
Output : a $n \times n$ matrix $\mathbf{\Theta}$ whose generic element $\Theta[i, j]$ represents the topological distance τ_{ij} between taxa i and j in the phylogeny coded by s
Data : $\mathbf{M}$ is a $k \times 6$ matrix; $\mathbf{\Theta}$ is a $n \times n$ matrix; CurrentNode , VisitedNode are integers;Leaf is a function returning the label of the i-th taxon

```
1  Decode s with Algorithm 2;
2  for i = 1 to k do
3      assign the label of the three vertices adjacent to vertex i to entries
       M[i, 1], M[i, 2], M[i, 3];
4  for i = 1 to k + 2 do
5      Θ[Leaf(i), Leaf(i)] = 0; M[s_i, 4] = 1; M[s_i, 5] = Leaf(i); M[s_i, 6] = 0;
       CurrentNode = s_i;
6      while true do
7          if M[CurrentNode, 6] < 3 then
8              M[CurrentNode, 6] + +;
9              VisitedNode = M[CurrentNode, M[CurrentNode, 6]];
10             if VisitedNode ≠ M[CurrentNode, 5] then
11                 if VisitedNode ∈ Σ then
12                     M[VisitedNode, 4] = M[CurrentNode, 4] + 1;
13                     M[VisitedNode, 5] = CurrentNode;
14                     M[VisitedNode, 6] = 0;
15                     CurrentNode =VisitedNode;
16                 else
17                     Θ[Leaf(i), VisitedNode] = M[CurrentNode, 4] + 1;
18         else
19             if CurrentNode = s_i then
20                 break;
21             else
22                 CurrentNode = M[CurrentNode, 5];
23 return Θ;
```

Proof. Note that decoding a PCP can be optimally performed in $O(n \log n)$ (Caminiti et al., 2007), and lines 2–3 are $O(n)$, hence the complexity of Algorithm 7 is determined by the main loop 4–22. The for loop at line 4 is repeated n times. In order to evaluate the complexity of the while loop consider s_1. Since $\mathbf{M}[s_1, 6] = 0$ then the if-condition at lines 7–17 is executed. Specifically, line 8 ensures that lines 7–17 are executed exactly 3 times for s_1, i.e., that all vertices adjacent to s_1 are visited. Since a PCP encodes a tree

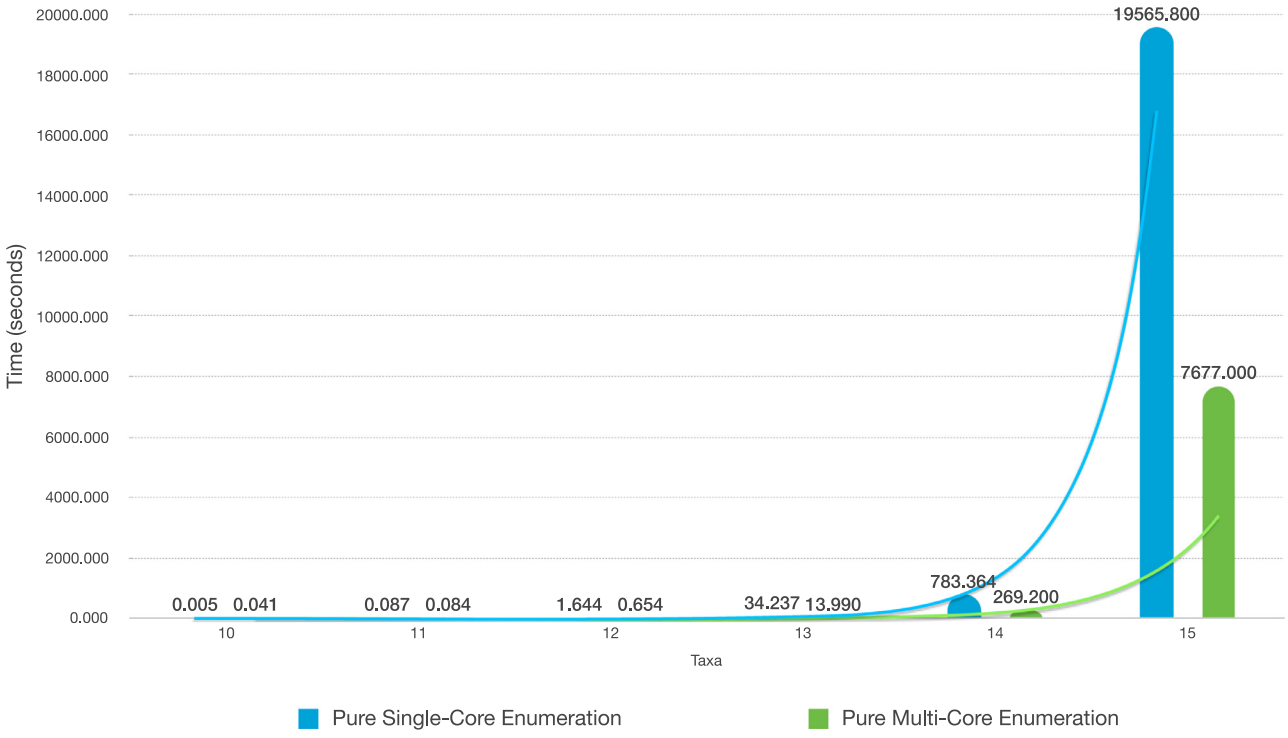


Fig. 4. Trend of the computing times relative to the single-core and multi-core pure enumeration procedures for $n \geq 10$.

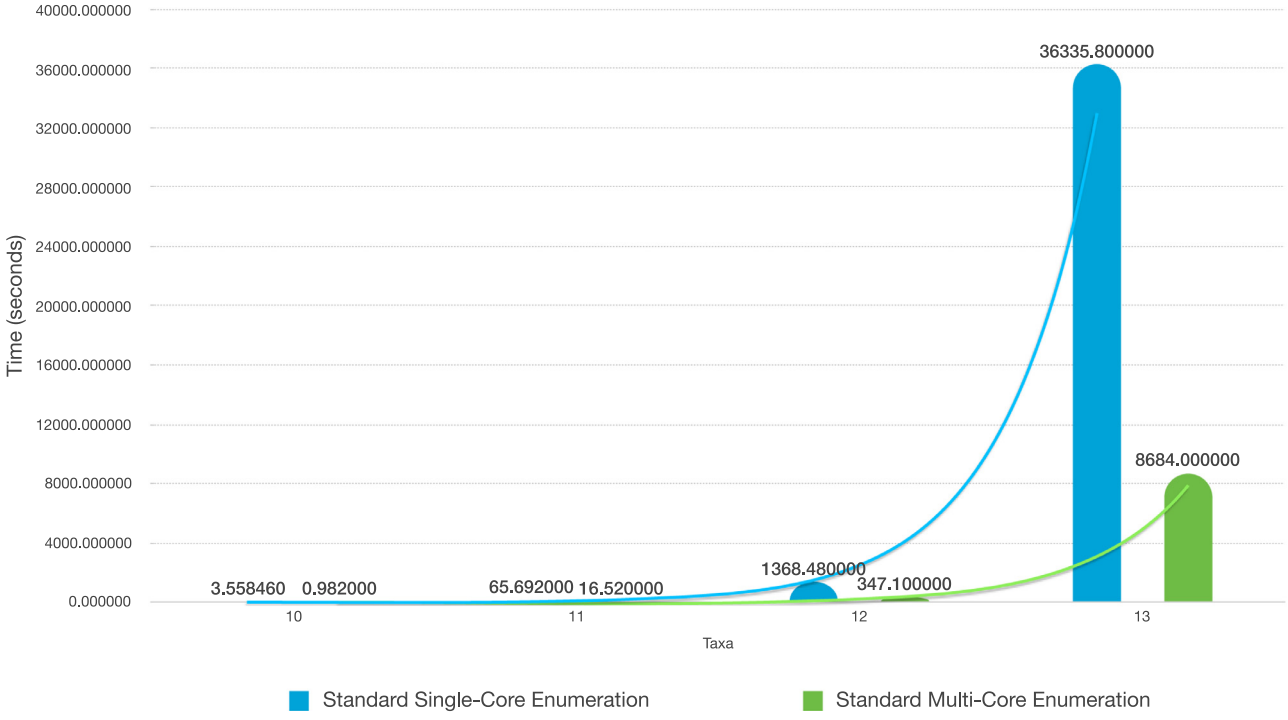


Fig. 5. Trend of the computing times relative to the single-core and multi-core standard enumeration procedures for $n \geq 10$.

whose internal vertices have degree three, assuming $k > 1$, there exists at least one vertex adjacent to s_1 that belongs to Σ. Let s_j this vertex. Then, the if-condition at lines 11–15 is executed, the value of the current vertex is updated to s_j and the if-condition at lines 7–17 re-executed. Since line 8 ensures to visit all vertices adjacent to the vertex stored in the current node, iterating this process, all the internal vertices will be visited. The backtracking at line 22 and the if-condition at lines 19–20 ensure the termination of the while loop. Thus, the while loop is $O(n)$ and Algorithm 7 is $O(n^2)$, hence optimal. □

A simple modification of Algorithm 7 allows to compute at the same time both the topological distances and the objective function value of the input PCP. Specifically, it is sufficient to add in

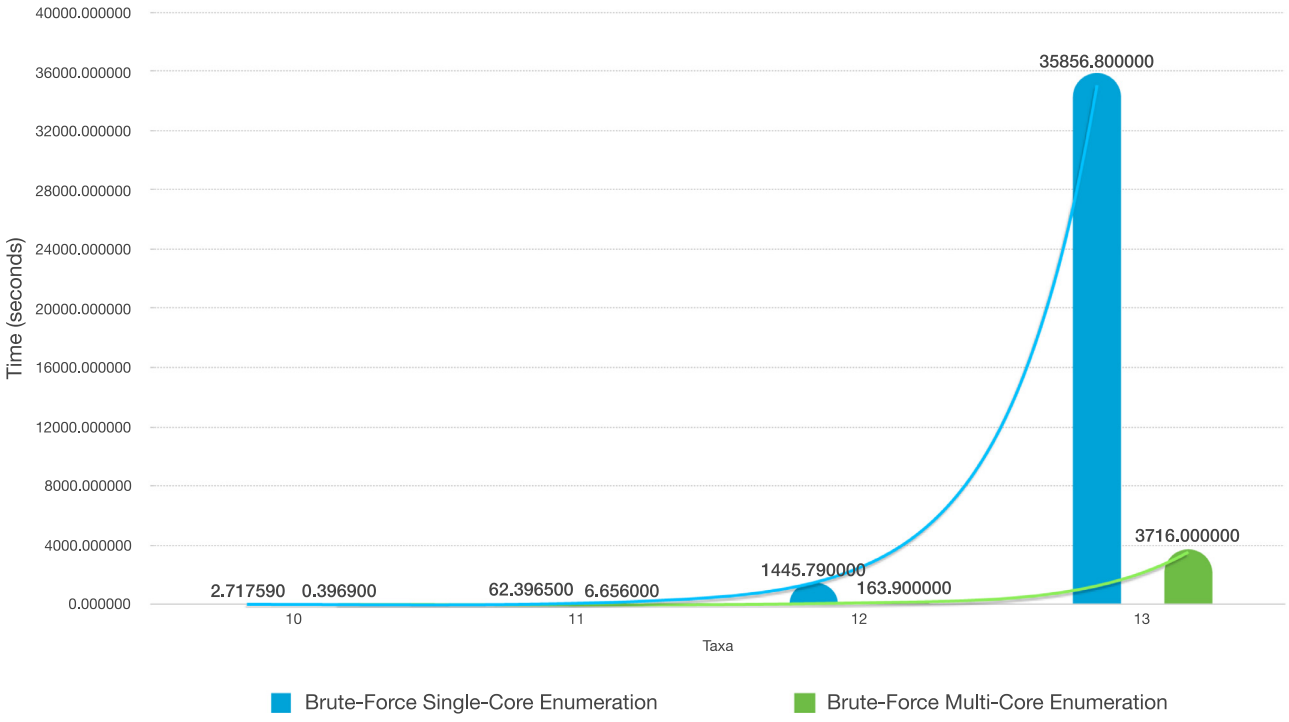


Fig. 6. Trend of the computing times relative to the single-core and multi-core standard enumeration procedures for $n \geq 10$.

input also the distance matrix **D** between taxa (or to make it accessible e.g., by declaring it as a global variable) and to add just after line 17, the line

$$\texttt{TreeLength} = \texttt{TreeLength} + \texttt{d[Leaf(i), VisitedNode]} 2^{-\boldsymbol{\Theta}[\texttt{Leaf(i)},\texttt{VisitedNode}]}$$

where `TreeLength` is a real variable initialized to zero, `d[Leaf(i), VisitedNode]` is the entry of **D** at row `Leaf(i)` and column `VisitedNode` (i.e., $d_{\texttt{Leaf(i)},\texttt{VisitedNode}}$), and $\boldsymbol{\Theta}[\texttt{Leaf(i)},\texttt{VisitedNode}]$ is the topological distance between taxa `Leaf(i)` and `VisitedNode` in the phylogeny encoded by the PCP. In this context Algorithm 7 would not return $\boldsymbol{\Theta}$ but the objective function value (1) of the input PCP, i.e., the final value of `TreeLength`. In order to evaluate the computational overhead introduced by Algorithm 7, we considered a third implementation of the enumeration procedure, denoted in Table 2 as the "brute-force enumeration", both in presence and in absence of parallelism. As the input distance matrix has a negligible impact on the estimation of the computational overhead, we assumed $d_{ij} = 1$, for all $i, j \in \Gamma$, $i \neq j$. As general trend, Table 2 shows that when Algorithm 7 is used, the overall computing time markedly increases with respect to the pure enumeration, but remains lower than the standard enumeration, both in presence and in absence of parallelism. This is mainly due to high latency of the output operations (on file or on screen). We observe that also in this case the use of parallelism is particularly beneficial (see Fig. 6), by allowing not only a decrement of the computing times up to 90% but also enabling the standard and the brute-force enumeration within 56 hours computing time.

As a closing remark, we observe that computing the topological distance can be made more efficient by exploiting the properties of PCPs. Specifically, in a PCP the first $(k+2)$ integers refer to external edges of a phylogeny T. Hence, if an integer s_i is located at position $1 < i \leq k+2$, and the first instances of s_i is met at position $j < i$ then it holds that $\boldsymbol{\Theta}_{ij} = \boldsymbol{\Theta}_{ji}$ (i.e., $\tau_{ij} = \tau_{ji}$), and $\boldsymbol{\Theta}_{iq} = \boldsymbol{\Theta}_{jq}$ (i.e., $\tau_{iq} = \tau_{jq}$), for all $q \in \{1, 2, \ldots, n\}$, $q \neq i, j$. In fact, since taxa i and j share the same internal vertex s_i then the length of the paths between i and the other leaves is equal to the length of the paths between j and the other leaves. Moreover, it is worth noting that, given a PCP s, swapping the second instance of the integer 1 at position i, $1 \leq i \leq k+1$, with an integer s_j, $1 \leq j \leq k+2$ and $j \neq i$, corresponds to swapping the taxa i and j on a same phylogeny T encoded by s. Hence, instead of recomputing Algorithm 7 it is enough to swap appropriately the rows and the columns relative to taxa i and j in matrix $\boldsymbol{\Theta}$, which is an operation that can be performed in $O(n)$.

Acknowledgments

The authors thanks the anonymous referees for their valuable comments. The first author acknowledge support from the Belgian National Fund for Scientific Research (F.R.S.-FNRS) via the research grant "Crédit de Recherche" ref. S/25-MCF/OL J.0026.17; the Université Catholique de Louvain via the "Fonds Spéciaux de Recherche" (FSR) 2017–2019; and the Fondation Louvain via the research grant "Le numérique au service de l'humain". Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Equipements de Calcul Intensif en Fédération Wallonie Bruxelles (CECI), funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11. Part of this work has been developed when D. Catanzaro was Invited Professor at the Department of Management of University Ca Foscari of Venice, Italy, during the academic year 2018.

References

Aringhieri, R., Catanzaro, D., Di Summa, M., 2011. Optimal solutions for the balanced minimum evolution problem. Comput. Opera. Res. 38, 1845–1854.
Billera, L.J., Holmes, S.P., Vogtmann, K., 2001. Geometry of the space of phylogenetic trees. Adv Appl Math 27 (4), 733–767.
Caminiti, S., Finocchi, I., Petreschi, R., 2007. On coding labeled trees. Theor Comput Sci 382, 97–108.
Catanzaro, D., 2009. The minimum evolution problem: overview and classification. Networks 53 (2), 112–125.
Catanzaro, D., Labbé, M., Pesenti, R., Salazar-Gonzáles, J.J., 2012. The balanced minimum evolution problem. INFORMS J Comput 24 (2), 276–294.

Catanzaro, D., Pesenti, R., Wolsey, L.A., submitted, 2019. On the balanced minimum evolution polytope. Technical Report. CORE, Université Catholique de Louvain, Belgium.

Christof, T., Loebel, A., 2015. Polyhedron representation transformation algorithm. http://comopt.ifi.uni-heidelberg.de/software/porta/

Desper, R., Gascuel, O., 2004. Theoretical foundations of the balanced minimum evolution method of phylogenetic inference and its relationship to the weighted least-squares tree fitting. Mol. Biol. Evol. 21 (3), 587–598.

Felsenstein, J., 2004. Inferring Phylogenies. Sinauer Associates, Sunderland, MA.

Fiorini, S., Joret, G., 2012. Approximating the balanced minimum evolution problem. Opera. Res. Lett. 40 (1), 31–35.

Forcey, S., Keefe, L., Sands, W., 2016. Facets of the balanced minimal evolution polytope. Math. Biol. 73, 447–468.

Forcey, S., Keefe, L., Sands, W., 2017. Split-facets for balanced minimal evolution polytopes and the permutoassociahedron. Bull. Math. Biol. 79 (5), 975–994.

Gascuel, O., 2005. Mathematics of Evolution and Phylogeny. Oxford University Press, New York.

Gascuel, O., Steel, M., 2006. Neighbor-joining revealed. Mol. Biol. Evol. 23 (11), 1997–2000.

Gawrilow, E., Joswig, M., 2000. Polymake: a framework for analyzing convex polytopes. In: Kalai, G., Ziegler, G.M. (Eds.), Polytopes - Combinatorics and Computation. Birkhäuser, pp. 43–74.

Haws, D.C., Hodge, T.L., Yoshida, R., 2011. Optimality of the neighbor joining algorithm and faces of the balanced minimum evolution polytope. Bull. Math. Biol. 73 (11), 2627–2648.

Huffman, D.A., 1952. A method for the construction of minimum redundancy codes. In: Proceedings of the IRE.

Kapranov, M.M., 1993. The permutoassociahedron, Mac Lane's coherence theorem and asymptotic zones for the KZ equation. Journal of Pure and Applied Algebra 85 (2), 119–142.

Pardi, F., 2009. Algorithms on Phylogenetic Trees. University of Cambridge, UK Ph.D. thesis.

Parker, D.S., Ram, P., 1996. The construction of huffman codes is a submodular ("convex") optimization problem over a lattice of binary trees. SIAM J. Comput. 28 (5), 1875–1905.

Pauplin, Y., 2000. Direct calculation of a tree length using a distance matrix. J. Mol. Evol. 51, 41–47.

Petersen, W.P., Arbenz, P., 2004. Introduction to Parallel Computing. Oxford University Press, Oxford, UK.

Reiner, V., Ziegler, G.M., 1993. Coxeter-associahedra. Technical Report SC-93-11. Zuse Institute Berlin (ZIB), Berlin, Takustrasse 7, 14195, Germany.

Sayood, K., 2017. Introduction to Data Compression, 5th Morgan Kaufmann, San Francisco, CA.