The divisible load balance problem with shared cost and its application to phylogenetic inference

Constantin Scholl*, Kassian Kobert‡, Tomáš Flouri*‡ and Alexandros Stamatakis*‡
*Karlsruhe Institute of Technology, Institute for Theoretical Informatics, Germany
Email: constantin@scholl.im, alexandros.stamatakis@kit.edu
‡Heidelberg Institute for Theoretical Studies, Germany
Karlsruhe Institute of Technology, Institute for Theoretical Informatics, Germany
Email: {alexandros.stamatakis, tomas.flouri, kassian.kobert}@h-its.org

***Abstract*—Motivated by load balance issues in parallel calculations of the phylogenetic likelihood function, we recently introduced an approximation algorithm for efficiently distributing partitioned alignment data to a given number of CPUs. The goal is to balance the accumulated number of sites per CPU, and, at the same time, to minimize the maximum number of unique partitions per CPU. The approximation algorithm assumes that likelihood calculations on individual alignment sites have identical runtimes and that likelihood calculation times on distinct sites are entirely independent from each other. However, a recently introduced optimization of the phylogenetic likelihood function, the so-called site repeats technique, violates both aforementioned assumptions. To this end, we modify our data distribution algorithm and explore 72 distinct heuristic strategies that take into account the additional restrictions induced by site repeats, to yield a 'good' parallel load balance.**

Our best heuristic strategy yields a reduction in required arithmetic operations that ranges between 2% and 92% with an average of 62% for all test datasets using $2, 4, 8, 16, 32$, and 64 CPUs compared to the original site-repeat-agnostic data distribution algorithm.

***Keywords*-bin packing; site repeats; load balancing; data distribution; phylogenetic likelihood; phylogenomic analysis**

I. Introduction

Maximizing the efficiency of parallel codes, by distributing the data in such a way as to optimize load balance, represents one of the major challenges in high performance computing.

Here, we address a specific data distribution challenge, which, to the best of our knowledge, has not been addressed before. Our work is motivated by parallel phylogenetic likelihood computations, that is, reconstruction of evolutionary histories based on molecular sequence data (see [1] for an overview). In phylogenetics, we are given a *multiple sequence alignment* (MSA) whose columns are nowadays typically subdivided into distinct partitions (e.g., genes or other disjoint subsets of the data). The columns of a MSA represent the sites and the rows represent the taxa (species) of the MSA for which we intend to infer evolutionary histories. All characters in a given column of the MSA are assumed to share a common evolutionary history. Given the MSA and a partitioning scheme, we can calculate the likelihood on a given candidate tree. In partitioned analyses, individual partitions of a MSA, have a separate set of likelihood model parameters, that is, they are assumed to evolve at different rates. This reflects the distinct evolutionary pressures that, for instance, different genes or parts thereof experience in the course of evolution. Partitioned analyses are the standard use-case for current likelihood-based analyses of empirical data with tools such as RAxaML [2] or MrBayes [3]. In current tools for likelihood-based phylogenetic inference, likelihood calculations on candidate trees typically account for over 85% of total runtime [4]. Thus, the *phylogenetic likelihood function* (PLF) is *the* target function for parallelization.

Initially, we formally state the original data distribution problem. Then, we explain why so-called *site repeats* (SR), that is, sub-columns of the MSA that are identical within a given subtree, complicate the matter.

In the standard case (i.e., without site repeats), we are given a list of k partitions and c CPUs. Each partition has a computation cost that is linear to the number of sites/columns it comprises. Analogously to the classic bin packing problem, we try to optimally assign the partitions (items) to the CPUs (bins). The first key difference is that the number of CPUs c is given, and that we intend to balance the data among all CPUs. In other words, we want to minimize the maximal per-CPU load. The second key difference is that partitions *are* divisible, since a partition consists of (originally independent) sites/subelements. Thus, for improving load balance, we can split partitions into disjoint sets of sites that are allocated to distinct CPUs.

The computational cost per partition has two components: All partitions have an identical constant base cost α. If we split a partition among two or more CPUs, each subset of that partition incurs this base cost α for the CPU it is assigned to. In phylogenomics, α is the cost for calculating the transition probability matrix $P(t)$ for a given time t. This probability matrix is an integral part of the statistical model of evolution. Since partitions evolve under distinct models, we need to compute $P(t)$ for each partition separately. However, all MSA sites that belong to the same

partition have identical model parameters. Thus, α incurs only once per partition and per CPU. Alternatively, we could also parallelize the $P(t)$ calculations and then broadcast the $P(t)$ values to all CPUs. However, based on previous computational experiments, the $P(t)$ calculations are excessively fine-grained and frequent, to allow for efficient parallelization. Therefore, we compute $P(t)$ (α) redundantly at several CPUs if the sites of a partition have been split up and allocated to more than one CPU.

The second component of the per-partition PLF calculation is the variable cost ϕ. For standard likelihood implementations ϕ is linear in the number of sites per partition. This is because the same amount of arithmetic operations is required to compute individual per-site likelihoods. Since alignment sites are assumed to evolve independently in the likelihood model, calculations on a single site of a partition can be performed independently of and concurrently to all other sites. Therefore, we can easily distribute the sites of a single partition to several CPUs.

Thus, based on the prolegomena, to maximize parallel efficiency, we need to minimize redundant calculations of $P(t)$ (α) by only splitting partitions when necessary, while distributing sites evenly among CPUs.

Our initial work [5] focused on addressing this data distribution problem for representative PLF implementations in tools such as PLL [6] and RAxML [2]. As mentioned before, the *original data distribution algorithm* (ODDA [5]) assumes that (i) all sites have the same per-site computation cost and (ii) these per-site computation costs do *not* depend on *how* sites of a single partition are split up among CPUs. For example, if a partition with 100 sites is split up among two CPUs and each CPU shall be allocated 50 sites, any split of those 100 sites into 50 sites per CPU will exhibit the same computational cost.

Techniques for accelerating PLF calculations such as *Subtree Equality Vectors* (SEV) [7], [8] or *Site Repeats* (SR) [9], [10] complicate this matter. If we use these techniques in the PLF, the data distribution problem becomes more complex, since both previous assumptions (same cost per site and site independence) regarding per-site computation costs are violated. These techniques take repeating MSA site patterns in subtrees of the phylogeny into account to reduce the amount of per-site PLF computations. A comparison between the PLF functions in the *Phylogenetic Likelihood Library* [6] (derived from RAxML [2]) and a rudimentary implementation that uses SRs shows speedups that range between a factor of 2 up to a factor of 10 [11]. Note that, SR-based techniques can also yield memory savings by more than 50% depending on the input MSA (see e.g., [8]). We describe the SR technique more thoroughly in Section III. Essentially, SR-based optimizations reduce the computation cost ϕ by detecting and re-using identical intermediate PLF results *among* sites. Thus, distinct sites now have varying computation costs. Furthermore, if we assign two sites that can share a large fraction of intermediate results to different CPUs, the accumulated computation cost ϕ will increase. Note that, communicating intermediate shared site computation results between CPUs is not a viable solution, since the computation to communication ratio is unfavorable. Analogously to the argument for recomputating α ($P(t)$) we can not communicate such intermediate results because of the extremely fine-grained nature of the PLF operations.

This has implications for distributing data of SR-based PLF implementations. While we can still split partitions arbitrarily among CPUs, we will have to sacrifice some savings that stem from the shared computations. Thus, as for α, we will need to conduct some redundant computations for sites that belong to the same partition and that share some results if these sites are allocated to distinct CPUs. To leverage the substantial computational savings of SR-based PLF implementations, we present an appropriately adapted data distribution algorithm here. Our goal is to split as few partitions among as few CPUs as possible. At the same time we intend to maximize the amount of shared computations between sites for each partition that had to be split. This minimizes the variable per-partition cost ϕ and the accumulated number of base costs α for calculating $P(t)$. Finally, by reducing the overall accumulated computation cost of the data distribution, we can minimize parallel runtimes.

Given that the problem ODDA aims to solve was shown to be NP-hard, we suspect that the current problem, that has one additional optimality condition, is NP-hard as well. However, a more thorough study of the theoretical properties of our problem is outside the scope of this paper. For this reason, we first assess whether taking into account among-site dependencies for the data distribution algorithm affects parallel efficiency, or not. As we show in Section VI-E, disregarding among-site dependencies *can* decrease parallel efficiency by up to one order of magnitude. In Section V we therefore present and assess several ad hoc heuristics for improving load balance in parallel SR-based PLF computations.

Note that, parallel PLF implementations now form part of several widely-used tools (e.g., ExaML, RAxML, MrBayes, ExaBayes, PhyML) and the results presented here are generally applicable to all of these tools. At present, only ExaML and RAxML offer basic SR-based PLF implementations that merely exploit a fraction of the SRs that are present for accelerating calculations (see [8]). However, we expect that most PLF-based tools will adopt the SR technique [11], because of the substantial runtime *and* memory savings that can be achieved.

The remainder of this paper is organized as follows. In Section II, we survey related work on similar scheduling problems from the area of bin packing with items that can share space. In the subsequent Section III we describe the aspects of the SR technique that are relevant for the work we

present here at an abstract level. In Section IV we discuss the properties of the cost factors α and ϕ. We also define a theoretical *lower bound* that we use as baseline to design and assess our heuristics. Our main contribution is presented in Sections V and VI. Section V introduces polynomial-time heuristics which yield data distributions that are only 5.75% worse than the lower bound. Finally, in Section VI, we present our experimental setup and the performance gains achieved by our heuristics compared to the original data distribution algorithm (ODDA) on both, simulated, and empirical MSAs.

II. Related work

Since we are not aware of any related work in phylogenomics, we briefly review related work on similar scheduling problems.

The so-called *many-constrained bi-objective bin packing* task is one such related problem [12]. Here, items need to be distributed into bins *and* the items are subject to various constraints. The constraints reduce to a cost function between items: The cost of an item can depend on another in a positive, neutral, or negative way if they are in the same bin. In the beginning all items are considered to be located in separate bins. The objective is to minimize the accumulated cost *and* number of bins, by grouping items together into bins.

The Virtual machine (VM) packing problem is also related [13], [14]. The goal is to pack a given amount of VMs (items) into a minimal amount of servers (bins). Each VM has a certain set of memory pages. These memory pages can overlap with those of other VMs, therefore, increasing the number of VMs that fit onto a server.

In contrast to our problem, the items in both scheduling problems are not divisible and the number of bins is not given as input. In fact, the objective function for these related problems is to minimize the number of bins.

III. Site repeats

Repeated PLF evaluations can account for over 85% of total runtime in maximum likelihood (ML) based tree searches and Bayesian phylogenetic inferences (BI) [4]. One simple observation is that two identical sites that evolve under the same evolutionary model (forming part of the same MSA partition) always yield the same likelihood score. Thus, a common method to avoid these globally redundant calculations is to compress identical MSA sites in a pre-processing step. However, such identical calculations also appear locally at the subtree level. That is, MSA columns might be partially identical for a subset of taxa (species/rows) defined by a subtree (see Figure 1 for an example). While there have been previous attempts to exploit this property [7], Kobert *et al.* recently introduced an efficient algorithm [11] for detecting *site repeats* (SR). The PLF optimization relies on recognizing repeating DNA patterns at different column indices, i and j, of the MSA (on the same partition), defined by a subtree of the phylogeny whose PLF is being computed. As a consequence, the amount of SR-based savings depends on the actual tree topology.

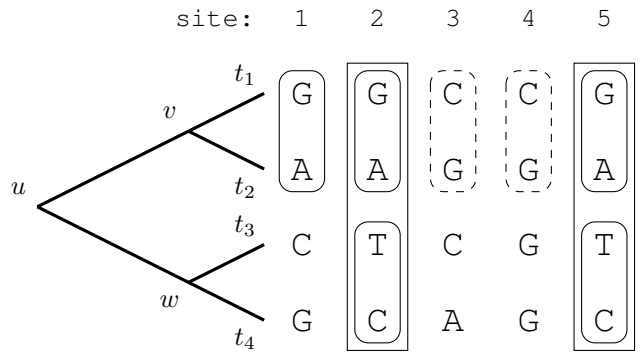


Figure 1. Sites 1, 2, and 5 form a SR at node v as they share the same subtree site pattern GA. Another repeating pattern is present at sites 3 and 4 (CG) for the same node. Note that, node w also induces a subtree with pattern CG at the tips at site 1. However, since branch lengths can be and typically are different from those of the subtree induced by node v, the conditional likelihoods may differ as well. But, there is another SR at sites 2 and 5 for node w (TC), and hence the conditional likelihood is the same for those two sites. Finally, sites 2 and 5 are a SR for node u (GATC).

At an abstract level, PLF calculations conduct post-order tree traversals to update the so-called *conditional likelihood vectors* (CLVs) that have as many entries as the input MSA has columns. Given a fast method to identify repeating subtree site patterns, we can omit all PLF calculations of column indices $p > q$ for identical subtree site patterns, if q is the first occurrence of the specific pattern.

In Figure 1 we only calculate the CLVs at node v for column index $p = 1$ and omit the PLF calculations for column indices $q = 2$ and $q = 5$. Without further optimization we can omit a total of 5 out of 15 CLV calculations in this example. An important metric for distributing the data is the number of SRs a specific column contributes to. Henceforth, we will call this metric the *site repeat count* (SRC). In our example site 1 has a SRC of 1, while site 2 has a SRC of 3.

Kobert *et al.* show that a —at a technical level— not fully optimized SR-based PLF implementation consistently outperforms one of the most efficient available PLF implementations (including AVX intrinsics) for distinct real-world application scenarios and tree topologies. Respective speedups range between a factor of 2 up to 10 [11]. Because the work on SRs is very recent, we are not aware of any phylogenomic data distribution algorithm that takes into account the additional constraints induced by SR-based PLF implementations.

IV. Preliminaries

Assume a MSA of m sequences (taxa) and n sites $S = \{s_1, s_2, \ldots, s_n\}$, and a partition scheme of k disjoint

partitions $p_1, p_2, \ldots, p_k$ such that $p_i \in 2^S$. Our task is to distribute these partitions among c CPUs, such that the maximum computational cost (*load*) among CPUs is minimized. Note that, we are allowed to assign distinct sites of a partition to different CPUs. Let $\lambda : \{1, 2, \ldots, c\} \times \{1, 2, \ldots, k\} \mapsto 2^S$ be the mapping of sites to CPUs and partitions. Further, let

$$P_j = \{\lambda(j, i) \mid \lambda(j, i) \neq \emptyset, 1 \leq i \leq k\}$$

be the set of partitions assigned to CPU j. The total computational cost at a CPU j is then

$$|P_j|\alpha + \sum_{p \in P_j} \phi(p),$$

where the mapping $\phi : 2^S \mapsto \mathbb{N}$ is the SR-based PLF-C (computational cost) for a specific subset of sites from a partition. Note that, for simplicity, in the rest of the text we will treat (site) partitions as lists instead of sets. Hence, we introduce two operations $\setminus$ and $\cup$, for adding and removing sites from partition lists. Operation $p \setminus \{i_1, i_2, \ldots, i_m\}$ removes sites with indices $i_1, i_2, \ldots, i_m$ from a partition list p, and $p \cup \{s_1, s_2, \ldots s_m\}$ appends sites $s_1, s_2, \ldots, s_m$ at the end of list p. A site can be removed in $\mathcal{O}(1)$ time, and adding m sites requires $\mathcal{O}(m)$ time.

The mapping λ can be implemented as a lookup table of c CPUs. Each element of this lookup table is a again a lookup table for each of the k partitions. Note that, all algorithms we present assume that the number of sites n is larger than the number of CPUs c.

A. Computational costs α and ϕ

The first simple observation is, that the computational cost for each CPU depends on the cost components α and ϕ. However, both quantities can only be measured in abstract terms. The exact runtimes depend on the hardware architecture, data type, and tree topology. Therefore, the objective is to minimize both factors, without being able to directly compare them. We will show that the count of additional α values, which we call $extra_\alpha$, created by splitting partitions, does not vary substantially for the heuristics we tested in Section VI-E. Thus, we focus on minimizing the cost ϕ, which corresponds to the number of arithmetic operations required to evaluate the PLF on a given set of sites for a given tree topology. We call this cost the *PLF count of operations* (PLF-C).

B. Lower bound $\mathcal{L}$

The second observation is, that we can calculate a simple theoretical lower bound for the optimal accumulated per-CPU PLF-C. We use this bound, to steer our heuristics *and* to assess the performance of our heuristics.

Assume that all partitions are assigned to a single CPU. We can then calculate the PLF-C for this CPU. Simply dividing this single CPU PLF-C by the number of CPUs c yields a natural lower bound $\mathcal{L}$. Since no partitions have been split up on the single CPU assignment, the respective SR-induced savings are maximal. Thus, for any assignment of the partitions to two or more CPUs, which may not be able to retain all site repeats, the per-CPU PLF-C will be $\geq \mathcal{L}$. Consider the example MSA in Figure 1 with only two sites and assume that we want to assign the data to two CPUs. The lower bound is $\frac{5}{2} = 2.5$ PLF-C. However, the optimum data distribution will assign one site to each CPU. Thus, the optimized PLF can not skip the SR `GA` anymore, resulting in a per-CPU PLF-C of $3 > 2.5$ for both CPUs.

V. Description of heuristic components

The heuristics we present here consist of three phases: Data pre-processing (phase I), initial distribution of sites to CPUs (phase II), and site reshuffling strategies (phase III). The data pre-processing phase prepares the MSA data for the subsequent initial distribution phase. We present two algorithms for computing such an initial partition data to CPU distribution. Finally, we employ three different site reshuffling strategies to further decrease the PLF-C.

The individual components designed for these three phases can be flexibly combined in a plethora of ways and orders to rapidly assemble and test heuristic data distribution algorithms. We assessed a total of 72 heuristic strategies using the 7 core components (2 for pre-processing, 2 for the initial assignment, and 3 for the reshuffling phase) for implementing the three distinct distribution phases. For a complete list of all 72 strategies, see the on-line supplement[1] on `github`. Here, we only present a performance assessment of 32 —including the two best-performing ones— out of 72 heuristics. Presenting all 72 heuristics would merely require describing additional variants of our components.

In addition, we describe the two heuristics that performed best, in more detail. All 32 heuristics we tested are enumerated on the x-axis of Figure 11.

A. Phase I heuristics: Data pre-processing

A commonly used method to avoid redundant calculations in the PLF is to remove all duplicate sites from the MSA and only keep the unique sites. Since identical sites yield the same per-site likelihood if they evolve under the same model, the overall likelihood can be computed by assigning respective weights to unique sites.

As a second pre-processing step we sort the remaining, unique sites lexicographically. Through an empirical analysis we have observed that lexicographic sorting has the effect that sites sharing repeats are more likely to reside next to each other than for a random site order. Figure 9 depicts the sorted versus unsorted PLF-C between all pairs of consecutive sites, accumulated over all sites, all 1200 inferred trees, and all 8 MSAs we used (see Section VI-B). Sorting leads

[1] https://github.com/conscho/phylo_scheduling

to better results for the overall data distribution algorithm (see Section VI-E). The lexicographic sort requires $\mathcal{O}(nm)$ time and $\mathcal{O}(n+m)$ space.

B. Phase II heuristics: Initial distribution

The initial distribution of partitions to CPUs is a two-step process. First, we *pre-fill* the CPUs with entire partitions, that is, without splitting any partition until a filling limit is reached for at least one CPU. Then, we distribute and potentially split up the remaining partitions via the *greedy* or the *cut* approach (see below). Both approaches try to balance the PLF-C among CPUs.

In the pre-filling step, we intend to assign as many entire (without splitting them up) partitions as possible to the CPUs, for minimizing the per-CPU α cost. Initially, the *lower bound* $\mathcal{L}$ (see Section IV-B) is used as a computing capacity limit for each CPU. The reshuffling strategies (phase III, see Section V-C) might adjust this limit later-on. The procedure for assigning entire partitions to CPUs is analogous to phases 1 and 2 of the ODDA [5]: We first sort partitions by increasing order of computational cost (PLF-C). Starting with the partition that has the smallest PLF-C, we assign partitions to CPUs in a cyclic manner until adding a partition exceeds the capacity limit ($\mathcal{L}$) of a CPU. When the procedure stops, the still unassigned (remaining) partitions have indices $r, r+1, \ldots, k$. In a final step we sort the CPUs by their accumulated PLF-C in ascending order. Figures 8a and 8b illustrate the pre-filling process; Figure 2 contains the PREFILL algorithm. The asymptotic runtime complexity for pre-filling is $\mathcal{O}(k \log k + nm + c \log c)$, as sorting the k partitions requires time $\mathcal{O}(k \log k)$, and calculating the PLF-C for all partitions requires $\mathcal{O}(nm)$ time.

1) Greedy approach: The greedy approach is one of two alternatives for distributing the remaining partitions among the pre-filled CPUs. It tries to find the best fit CPU for each individual site. The site fit to a CPU is defined as the PLF-C increase that is induced by assigning the site to the CPU. The lower this PLF-C increase is for a specific CPU, the better the site fits. The underlying idea is, that by increasing the fit of sites from remaining partitions that need to be split up we can reduce the overall PLF-C for the data distribution.

However, we need to first assign at least some sites of the remaining partitions to CPUs to be able to compute a meaningful site fit. We achieve this via a virtual assignment of remaining partitions to CPUs (please see Figures 8c-8f before reading on, since the subsequent description will be easier to follow). Initially, we calculate the average per-site PLF-C cost μ for each remaining partition. Then, we use μ as a proxy for the PLF-C to generate a virtual assignment of the remaining partitions to the CPUs. This virtual assignment ignores SR effects and simply distributes all sites from all remaining partitions to the CPUs based on μ. Thereby, we can approximately balance the per-CPU PLF-C. Then,

```
PREFILL(P, k, c, L)
 ▷  Initialization
 1.  P = (p_1, p_2, ..., p_k) sorted ascending by φ(p_i)
 2.  for j ← 1 to c do
 3.     cost[j] ← 0
 4.     for i ← 1 to k do λ(j, i) ← ∅
 5.  j ← 1
 ▷  Distribution of partitions to CPUs
 6.  for i ← 1 to k do
 7.     if φ(p_i) + cost[j] ≤ L then
 8.        cost[j] ← cost[j] + φ(p_i)
 9.        λ(j, i) ← λ(j, i) ∪ p_i
10.     else
11.        break
12.     j ← (j mod c) + 1
13.  sort cost in ascending order and remap λ
14.  return r ← i
```

Figure 2. The PREFILL function requires the following input parameters: a list P of k partitions sorted in ascending order by their PLF-C, the number of CPUs c, and the lower bound $\mathcal{L}$.

for each CPU that has at least one site from one of the remaining partitions, we select the site in the middle of the subset of sites of the remaining partition (the median site) that has been assigned to this CPU as representative site. This representative site from the virtual assignment is then assigned to the respective CPU and becomes a real site assignment. As a result, at most two sites from at most two of the remaining partitions are assigned to each CPU. Sites of at most two distinct remaining partitions are assigned to each CPU, since we virtually assign the remaining partitions one-by-one in as large as possible monolithic blocks to CPUs. Keep in mind that each of the remaining partitions exceeds the capacity limit of every CPU because of the way we designed the prefilling component.

Figures 8c-8e illustrate and Figure 3 presents the pseudocode for the virtual assignment step of GREEDY. In the pseudocode z is the site index pointing to the first site of the current partition, while z' is the length of the current partition. The variable s in line 9 is the median site. Since we remove the median site s from p_i we also need to decrement z for the next partition (line 13). If the algorithm virtually assigns two partitions to a CPU, *vcost* represents the computational cost that is required for the first partition that was assigned to this CPU and defines how much computing capacity is left for the second partition on this CPU.

Once we have computed this initial assignment of median sites from the remaining partitions to CPUs, the second step of GREEDY is straight-forward. We simply iterate over the yet unassigned sites from the remaining partitions and assign each site to the CPU with the best PLF-C fit. If adding a remaining site to this CPU leads to exceeding the capacity limit $\mathcal{L}$, we assign the site to the CPU with the second-

```
GREEDY(P, r, k, λ, cost, c, ℒ)
 1.  P = (p_1, p_2, ..., p_r, ..., p_k)
 ▷   Initialization
 2.  j ← 1
 3.  vcost ← 0
 ▷   Step 1
 4.  for i ← r to k do
 5.     μ ← ⌊φ(p_i)/|p_i|⌋
 6.     z ← 1
 7.     while z ≤ |p_i| and j ≤ c
 8.        z' ← min(⌈(ℒ − cost[j] − vcost)/μ⌉,
                    |p_i| − z + 1)
 9.        s ← p_i[z + ⌊z'/2⌋]
10.        λ(j, i) ← λ(j, i) ∪ {s}
11.        cost[j] ← cost[j] + φ({s})
12.        p_i ← p_i \ z + ⌊z'/2⌋
13.        z ← z + z' − 1
14.        if z > |p_i|
15.           vcost ← z'μ − φ({s})
16.        else
17.           vcost ← 0
18.           j ← j + 1
```

Figure 3. The GREEDY function requires the following input parameters: The list P of remaining partitions sorted in ascending order by their PLF-C, where index r denotes the first and index k the last unassigned partition; the distribution function λ and the *cost* for the c CPUs; the lower bound $\mathcal{L}$. Here we show step 1 of the algorithm.

```
 ▷   Step 2: Assign remaining sites
19.  for i ← r to k do
20.     for each site s in p_i do
21.        B ← ∅
22.        C ← ∅
23.        for j ← 1 to c do
24.           if λ(j, i) ≠ ∅
25.              scost ← φ(λ(j, i) ∪ {s}) − φ(λ(j, i))
26.           else
27.              scost ← φ({s}) + 1
28.           if cost[j] + scost ≤ ℒ
29.              B ← B ∪ {(j, scost)}
30.           else
31.              C ← C ∪ {(j, scost)}
32.        if B = ∅ then B ← C
33.        (ĵ, ŝcost) ← argmin_(j,scost)∈B(scost)
34.        λ(ĵ, i) ← λ(ĵ, i) ∪ {s}
35.        cost[ĵ] ← cost[ĵ] + ŝcost
```

Figure 4. Step 2 of the GREEDY function.

best PLF-C fit etc., provided that adding the site will not violate its capacity limit. As a consequence we generally assign the site to a CPU that already has some sites of the same partition in order to minimize the α cost. Only, if no CPU below the capacity limit has sites of the same partition, but there are other CPUs below the capacity limit, we will assign the site arbitrarily to one of the latter. Finally, if *all* CPUs exceed $\mathcal{L}$, we assign the site to the CPU with the best fit. Figure 4 presents the pseudocode for the second step of GREEDY. Note that, the second step of Greedy will only terminate once all sites of all remaining partitions have been assigned to a CPU.

The total asymptotic runtime of step 1 is $\mathcal{O}(nm)$. The runtime is dominated by calculating μ in line 5. For all sites in the $r-k$ remaining partitions we have to compute ϕ, which requires $\mathcal{O}(m)$ time per site.

The runtime of step 2 is $\mathcal{O}(ncm)$. For each site of the $k-r$ remaining partitions we need to calculate the *scost* for each CPU. The *scost* in line 25 can be computed in $\mathcal{O}(m)$ by keeping the site repeats data structure in memory and using the algorithms from [10], [15]. We use this observation in all subsequent runtime analyses.

2) Cut approach: As an alternative to GREEDY, we implemented the CUT approach for distributing the remaining partitions to pre-filled CPUs. Here, we cut contiguous regions from the remaining partitions and assign them to CPUs. This has the benefit of incurring *exactly* the same accumulated α cost as the ODDA, which has been proven to exceed the optimal solution by at most one α per CPU [5]. We illustrate the cut approach in Figures 8g and 8h.

After the pre-filling phase, the remaining partitions and CPUs are still sorted by increasing PLF-C. Starting with these sorted lists, we iterate over the CPUs from the least filled to the most filled one (i.e., from largest to smallest available capacity). For each CPU we select a region from at most two of the remaining partitions. The size of the region is computed as follows for every CPU to be filled: we divide the free capacity of the particular CPU by the accumulated free capacity of all CPUs that have not reached their capacity limit yet and multiply this by the total PLF-C of all remaining and yet unassigned partitions. The underlying idea is to distribute the additional arithmetic operations we will need to conduct due to partitions that have already been split up as evenly as possible over all CPUs. To then actually assign such a cut region to the CPU, the region must exhaust or exceed the capacity limit of the current CPU. If this is not the case, we increase the size of the region until it exhausts the capacity limit of the current CPU. This restriction that CPU capacity limits must at least be exhausted is required for the reshuffling strategies (phase III, see V-C).

The pseudocode for CUT is provided in Figure 5. While δ returns the free capacity for a given CPU j, the array Δ contains the accumulated free capacity of all CPUs with an index $\geq j$ for a given j. We use z as site index again and *rcost* is the total PLF-C of all remaining partitions. The value ℓ denotes the target PLF-C capacity of the current CPU j which defines the size of the region that shall be assigned

```
CUT(P, r, k, λ, cost, c, L)
 1.  P = (p_1, p_2, ..., p_r, ..., p_k)
 ▷   Initialization
 2.  Δ[c + 1] ← 0
 3.  for j ← c downto 1
 4.     Δ[j] ← Δ[j + 1] + (L − cost[j])
 5.  i ← r
 6.  z ← 1
 ▷   Filling
 7.  for j ← 1 to c do
 8.     δ ← L − cost[j]
 9.     rcost ← φ(p_i[z ... |p_i|]) + Σ_{i'←i+1}^{k} φ(p_{i'})
10.     ℓ ← cost[j] + max(⌈δ/Δ[j] · rcost⌉, δ)
11.     while ℓ − cost[j] ≥ 0
12.        cost[j] ← cost[j]+
                 φ(λ(j, i) ∪ {p_i[z]}) − φ(λ(j, i))
13.        λ(j, i) ← λ(j, i) ∪ {p_i[z]}
14.        if z < |p_i|
15.           z ← z + 1
16.        else
17.           z ← 1
18.           i ← i + 1
19.           if i > k then end
```

Figure 5. The CUT function requires the following input parameters: The list P of remaining partitions sorted in ascending order by their PLF-C, where index r denotes the first and index k the last unassigned partition; the distribution function λ and the *cost* for the c number of CPUs; the lower bound $\mathcal{L}$.

to CPU j. Note that, CUT terminates when all sites from all remaining partitions have been assigned to CPUs.

The total asymptotic runtime of CUT is $\mathcal{O}(nm + c)$. For line 9 we can initialize two arrays: One, computing ϕ in the order of sites in the remaining partitions and the other computing ϕ in the reverse order of sites. Thus, line 9 requires constant time and the initialization can be computed in $\mathcal{O}(nm)$. By definition we will split up at most $2(k - r)$ partitions. These partitions are disjoint. Therefore, line 12 will be executed once for each site of the partitions that will be split and thus requires $\mathcal{O}(nm)$ time.

C. Phase III heuristics: Reshuffling strategies

After the initial distribution (e.g., computed with GREEDY or CUT), we can employ up to three site reshuffling strategies to further improve the PLF-C: Adjust limit, reduce max, and low SRC.

1) Adjust limit: The initial distribution provides a good estimate of how many partitions need to be split up and how the overall PLF-C for the distribution increases due to these splits. We use this knowledge to adjust the capacity limit of the CPUs: Instead of the lower bound $\mathcal{L}$ we now use the average CPU load from the initial distribution as capacity limit. We then simply re-run the pre-filling and the GREEDY or CUT approach with this updated capacity limit.

2) Reduce max: The bottleneck in parallel PLF computations is always the CPU with the highest PLF-C since it determines the overall runtime. Therefore, we attempt to reduce the load of the CPU with the highest PLF-C.

This reshuffling strategy uses the worst-case PLF-C for one site. This is a MSA-specific value that depends on the number of taxa m. The *worst-case site PLF-C* simply has a SRC of zero.

First, we select all partitions that have been split up and have been assigned to the most loaded CPU. Then, we determine all candidate CPUs that also have a part of those partitions. Out of those candidate CPUs we then select only those with a PLF-C that is smaller than the mean per-CPU PLF-C. Using the worst-case site PLF-C, we estimate how many sites can be moved from the most loaded CPU to the least loaded CPUs, such that they do not exceed the current capacity limit. We then move the sites with the lowest SRC away from the most loaded CPU. Thereby, we only lose a small number of SRs for the sites that will remain on the most loaded CPU. This *reduce max* operation can be repeated for an arbitrary number of times. We choose to repeat this process as many times as there are CPUs to achieve an acceptable trade-off between overall PLF-C improvement and the run-time for this heuristic component. Evidently, one could also design an explicit convergence criterion to determine when additional applications of reduce max are not worthwhile any more.

We present the pseudocode for reduce max in Figure 6. Line 1 assumes that P is the original sorted list of all partitions from the pre-filling phase (Section V-B). The constant *wcost* is the *worst case PLF-C for a site*. P' is the list of split up partitions assigned to one of the most loaded CPU $\hat{j}$.

The total asymptotic runtime for *reduce max* is $\mathcal{O}(c^2k + cnm)$. The first part is defined by the calculation of $\mathcal{L}$. The second part is defined by sorting the sites in p' by their SRC.

3) Low SRC: One goal of our distribution strategies is to maximize the site repeats in order to decrease the overall PLF-C. The *low SRC* strategy intends to reshuffle sites that exhibit a low SRC on their current CPU. For reshuffling, we select 20% (see below for a rationale for this setting) of the sites from each split up partition that have the lowest SRC under the current data distribution. We then provide this selection of sites as input to step 2 of GREEDY (see Section V-B1) to reassign them to the CPUs.

The actual percentage of sites selected for reshuffling is a tuning parameter. It should not be too high, since we want to determine how well the sites fit into a comparatively large, fixed site distribution. It should not be too low either, since this will decrease the overall PLF-C improvement of the reshuffling. We empirically determined that a threshold setting of 20% yields good performance.

Figure 7 presents the pseudocode. Line 1 assumes that P is the original sorted list of all partitions as obtained in the

```
REDMAX(P, k, λ, cost, c)
Let P = (p_1, p_2, ..., p_k)
wcost ← φ(p_1[1])
for l ← 1 to c do
    ĵ = argmax^c_{j←1}(cost[j])
    P' = {(i, λ(ĵ, i)) | λ(ĵ, i) ≠ p_i, 1 ≤ i ≤ k}
    for each (i, p') in P' do
        L ← Σ_{j=1}^{c} cost[j]/c
        if cost[ĵ] ≤ L then next
        sort p' = (s_1, s_2, ..., s_{|p'|}) ascending by SRC
        for j ← 1 to c do
            if λ(j, i) ≠ ∅ and cost[j] < L then
                μ ← ⌈(L − cost[j])/wcost⌉
                cost[j] ← cost[j] − φ(λ(j, i))+
                          φ(λ(j, i) ∪ {s_1, ..., s_μ})
                λ(j, i) ← λ(j, i) ∪ {s_1, ..., s_μ}
                cost[ĵ] ← cost[ĵ] − φ(λ(ĵ, i))+
                          φ(λ(ĵ, i) \ {1, ..., μ})
                λ(ĵ, i) ← λ(ĵ, i) \ {1, ..., μ}
```

Figure 6. The REDMAX function requires the following input parameters: a list P of k partitions, the distribution function λ, and the *cost* for c CPUs.

```
LOWSRC(P, k, λ, cost, c)
1.  Let P = (p_1, p_2, ..., p_k)
▷   Initialization
2.  L ← Σ_{j=1}^{c} cost[j]/c
3.  for i ← 1 to k do
4.      p*_i ← ∅
▷   Collect sites with low SRC
5.  for j ← 1 to c do
6.      P' = {(i, λ(j, i)) | λ(j, i) ≠ p_i, 1 ≤ i ≤ k}
7.      for each (i, p') in P' do
8.          sort p' = (s_1, s_2, ..., s_{|p'|}) ascending by SRC
9.          cost[j] ← cost[j] − φ(λ(j, i))+
                      φ(λ(j, i) \ {s_1, ..., s_{⌊|p'|·0.2⌋}})
10.         λ(j, i) ← λ(j, i) \ {s_1, ..., s_{⌊|p'|·0.2⌋}}
11.         p*_i ← p*_i ∪ {s_1, ..., s_{⌊|p'|·0.2⌋}}
12. call Phase 2 (line 19) of GREEDY with
    L and P = (p*_1, p*_2, ..., p*_k)
```

Figure 7. LOWSRC takes the following input parameters: a list P of k partitions, the distribution function λ, the *cost*, and the number of CPUs c.

pre-filling step (Section V-B). P' is the list of split partitions of CPU j.

The total asymptotic runtime of *low SRC* is $\mathcal{O}(cnm)$. For each CPU we iterate over all split partitions, which are at most k and sort them in $\mathcal{O}(cnm)$. Line 9 will reassign at most n sites, where each site requires $\mathcal{O}(m)$. As before Greedy phase 2 requires $\mathcal{O}(cnm)$ time.

ID	Type	Taxa	Unique Sites	Partitions
1	Empirical	59	3238	8
2	Empirical	128	19,866	34
3	Empirical	404	7444	11
4	Simulated	10	76,754	100
5	Simulated	10	169,939	200
6	Simulated	10	199,974	232
7	Simulated	10	246,897	310
8	Simulated	10	269,697	332

Table I
MSA PROPERTIES.

VI. EXPERIMENTAL SETUP AND RESULTS

A. Software

Given a MSA and a partition scheme, we employed RAxML [2] to infer and sample phylogenetic trees (for details see Section VI-C). Using Ruby and the Ruby Gem Newick-Ruby[2] for parsing trees, we initially characterized the problem and then implemented the heuristics. To visualize and post-process the results we used appropriate `R` packages.

B. Test datasets

We used five simulated and three empirical nucleotide MSAs for benchmarking the heuristics. To simplify the experiments we removed all identical sites from the MSAs prior to running the heuristics such that all MSAs only contained unique sites. The MSA are available for download at https://github.com/conscho/phylo_scheduling/tree/master/datasets and their properties are summarized in Table I.

C. Characterizing the problem

Initially, we assessed to which degree the actual tree shape influences the PLF-C differences between SR-based and SR-agnostic PLF implementations. We started 100 maximum likelihood (ML) tree searches from randomized stepwise addition order parsimony starting trees and another 100 ML searches on random starting trees for every test MSA using RAxML. We then calculated the PLF-C ratios between SR-based and SR-agnostic PLF implementations for each tree and MSA. The average SR-induced PLF-C savings for ML trees inferred on random and parsimony starting trees are almost identical. However, ML trees inferred from random starting trees show a slightly higher variance in savings than ML trees inferred on parsimony starting trees. This variance is more pronounced for our empirical MSAs. Figure 10 shows a representative boxplot for MSA 3.

As shown by Kobert *et al.* [11], the savings are substantial. The average PLF-C saving over all MSA and all trees we generated is 61.68%, with a minimum of 48.92% and a maximum of 91.14%.

[2]https://github.com/jhbadger/Newick-ruby

Next, we evaluated possible factors influencing the SR-induced savings based on several MSA characteristics. We did not find a correlation of savings with the likelihood score of a tree for a given MSA. That is, a tree with a better likelihood score does not necessarily have a lower PLF-C. Also, the PLF-C does not appear to correlate with the number of taxa. However, the number of sites per partition *is* positively correlated with PLF-C savings. Also, the placement of the virtual root, that is used to conduct the post-order tree traversal for calculating the CLVs (see Section III), has an impact on the PLF-C. Our initial experiments revealed that placing the virtual root using the so-called midpoint rooting technique, that reduces the height of the thereby *rooted* binary tree, increases PLF-C savings.

Because of this observation, all of our heuristics use a midpoint rooting on the given input tree for calculating the PLF-C of respective data distributions.

D. Ground-truth comparison

To analyze the performance of our heuristics we also compared them with the exact, optimal solution. We therefore generated and evaluated all possible data distributions exhaustively. Since there are n^c possibilities for assigning sites to CPUs, this was only feasible for small problem instances. The optimal solution is the data distribution that minimizes the PLF-C of the most loaded CPU. From each of the 8 MSAs in Table I we assembled 4 small MSAs by sampling MSA sites from the beginning of each partition. These groundtruth MSAs contained 4 partitions with $3-5$ sites that were assigned to 2 CPUs and 2 partitions with $4-6$ sites that were assigned to 3 CPUs. As one may expect, the exact solution is generally better than our heuristic solution. However, for 11 out of the 32 MSAs one of the heuristics was *on par* with the exact solution. In the worst case, the PLF-C difference between the best heuristic and the optimal solution for the most loaded CPU was 9.34%.

The optimal solution was also the *perfect* solution in 1 out of 32 experiments. We have a perfect solution when the PLF-C of each CPU equals the lower bound $\mathcal{L}$ (see Section IV-B). For instance, this is the case if the data distribution does not need to split up any partition such that sites which share SRs are not distributed over different CPUs (see Section III).

Keep in mind that, due to the small input sizes, the above results are mainly useful for verifying that our heuristics are not 'far off'. Also note that, the groundtruth calculates the optimal solution with respect to ϕ (i.e., the PLF-C) but does not strive to minimize the α cost.

E. Performance of heuristics

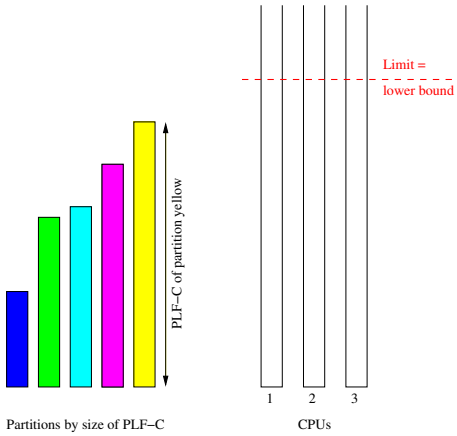
To assess the performance of our heuristics, we calculated data distributions for all MSAs. For each MSA we arbitrarily selected one of the ML trees inferred on a parsimony starting tree with RAxML and computed distributions for 2, 4, 8, 16, 32, and 64 CPUs. We then averaged the results for each heuristic over all MSAs and CPU counts. We assess the resulting heuristic data distributions by comparing the respective PLF-C of the most loaded CPU with the lower bound PLF-C $\mathcal{L}$ (see Section IV-B) and the resulting number $extra_\alpha$ of additional $P(t)$ calculations due to splitting up partitions (where $extra_\alpha := \sum_{j=1}^{c} |P_j| - k$, see Section IV for notation c, P_j, and k) with the lowest $extra_\alpha$ among all heuristic distributions. The heuristic that yielded the lowest PLF-C over all MSAs comprises the following components: a lexicographic MSA sort, then CUT, followed by the ADJUSTLIMIT, LOWSRC, and REDMAX reshuffling strategies. On average, the PLF-C of this heuristic is only 5.75% higher than the rather conservative theoretical lower bound $\mathcal{L}$. Out of the 32 combinations of our heuristic components that we list in this paper (see Section V) it ranks $5th$ with respect to $extra_\alpha$. The $extra_\alpha$ is only 4.69% higher than that of the best result which we deem acceptable based on our experience with developing ExaML and ExaBayes.

The best combination using the GREEDY component, also uses lexicographically sorted MSAs and *all* available reshuffling strategies in the same order as above. Its PLF-C is on average 6.27% worse than $\mathcal{L}$ but the $extra_\alpha$ is 42.83% higher than that of the best result which might be rather prohibitive for real-world parallel PLF implementations.

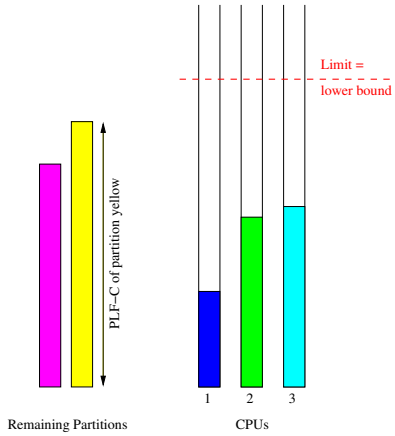
We also need to answer the question if a dedicated data distribution strategy for SR-based parallel PLF implementations is indeed necessary. To this end, we repeated the above experiments with the original SR-agnostic data distribution algorithm (ODDA). The data distribution proposed by ODDA requires on average 176.21% more PLF-C than $\mathcal{L}$. In comparison to ODDA, the best performing heuristic requires on average 61.71% less PLF-C with a minimum of 1.99% and a maximum of 92.31%. The $extra_\alpha$ of the best heuristic is surprisingly 0.81% lower than that of the ODDA. It might appear counter-intuitive, that some heuristics yield a lower $extra_\alpha$ than the ODDA. However, our reshuffling strategies can strive to reduce $extra_\alpha$ by placing entire small partitions with a large SRC onto the same CPU. Also, running ODDA on lexicographically sorted MSAs does not significantly change the results. It merely improves the PLF-C to 175.9% with respect to the lower bound $\mathcal{L}$. Thus, a dedicated SR-aware data distribution heuristic *is* required since it can reduce PLF-C values and hence parallel runtimes by 60% on average.

The results of these performance analyses are summarized in Figure 11. The Figure displays the key heuristic performance metrics averaged over all MSAs. It also demonstrates that $extra_\alpha$ is generally low and does not vary substantially among heuristics.

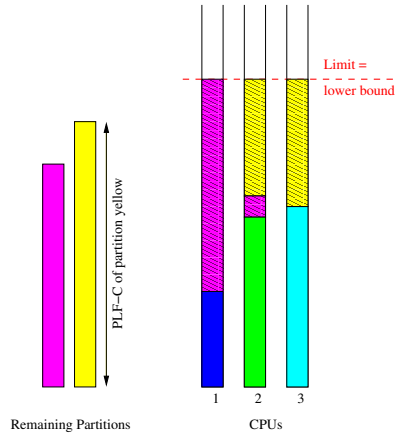
In Figure 12 we provide an example data distribution for MSA 1 on 4 CPUs for our two best heuristics as well as for ODDA.



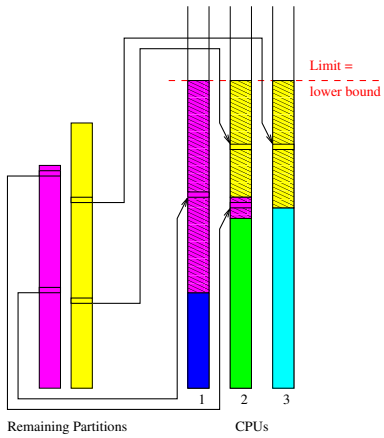
(a) Initial: Partitions sorted by their PLF-C and empty CPUs.



(b) Pre-Filling: Fill partitions sorted by their PLF-C into the CPUs without splitting as far as possible.

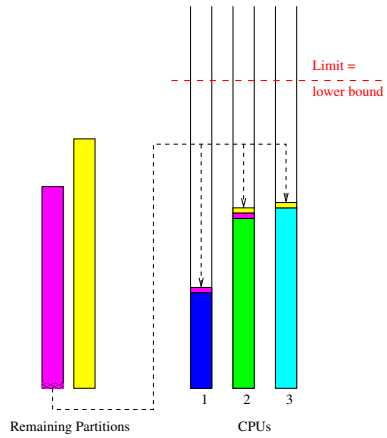


(c) Greedy phase 1a: Virtual assignment of remaining partitions neglecting SR effects.

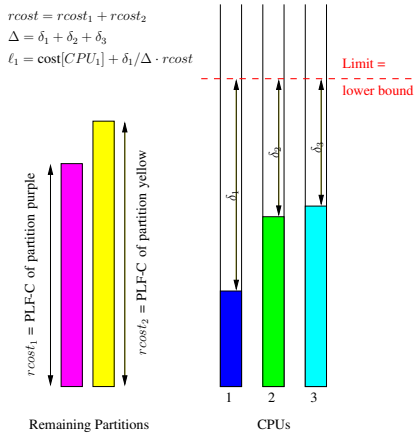


(d) Greedy phase 1b: Assign the sites in the middle of each virtual partition to the CPUs.

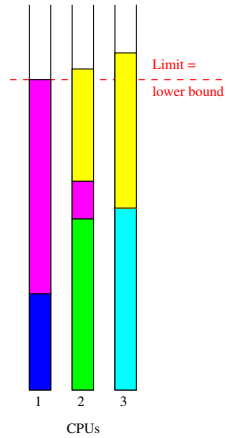
(e) Greedy after phase 1.



(f) Greedy phase 2: Test each site from the remaining partitions in each bin to find the CPU where the site fits best.



(g) Cut 1: Calculate the target capacity ℓ for CPU j.



(h) Cut 2: Typical distribution after applying the cut approach.

Figure 8. Illustrations.

VII. Conclusion and future work

In this paper we present the, to our knowledge, first data distribution problem statement for parallel PLF implementations that rely on the site repeats technique for reducing the number of PLF calculations on partitioned phylogenomic MSAs.

Whether the data distribution problem is NP-hard remains an open question. Here, we assume no polynomial algorithm exists, and we provide a lower bound $\mathcal{L}$ for the optimal data distribution scheme and explore a large number of heuristic data distribution algorithms using a component-based framework. All components and heuristics as well as the MSAs are freely available via `github`.

We initially characterize and quantify some aspects of SR-based calculations. Then, we present the two best performing heuristics that generate data distribution schemes with a PLF-C that is on average only 6% worse than the PLF-C of our conservative lower bound $\mathcal{L}$. More importantly, we show that designing SR-aware data distribution algorithms *does matter*, since the standard SR-agnostic ODDA approach yields an average PLF-C for the most loaded CPU that is more than twice as high as the PLF-C attained by our heuristics. Thus, SR-aware data distribution heuristics can reduce runtimes for parallel phylogenetic analyses by 60%.

We are planning on integrating the heuristics into real-world phylogenetic inference tools. We are currently developing a revised version of the PLL [6] that will include a full SR implementation and a Message Passing Interface (MPI) parallelization. Hence, we will also integrate a variant of the heuristics presented here. Note that, these heuristics were assessed on fixed trees with fixed (midpoint) rootings. While this covers application scenarios where trees remain fixed and only statistical model parameters on the tree are being optimized (e.g., divergence time estimates or tests for positive selection) this is not practical for tree searches where the tree topologies and virtual root positions constantly change. To this end, we will need to test appropriate heuristics that yield "good" data distributions for a larger set of trees and virtual rootings. Evidently, there is a trade-off between slightly increased PLF-C costs and frequent data re-distributions. While the asymptotic complexities of our heuristics might appear to be large, we expect them to be amortized by the substantially higher empirical cost of likelihood-based tree evaluations.

Acknowledgment

Part of this work was financially supported by the Klaus Tschira Foundation.

References

[1] Z. Yang, *Computational Molecular Evolution*. Oxford Series in Ecology and Evolution, 2006.

[2] A. Stamatakis, "RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies," *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, 2014.

[3] J. P. Huelsenbeck, F. Ronquist, and others, "MrBayes: Bayesian inference of phylogenetic trees," *Bioinformatics*, vol. 17, no. 8, pp. 754–755, 2001.

[4] N. Alachiotis and A. Stamatakis, "A generic and versatile architecture for inference of evolutionary trees under maximum likelihood," in *Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on*. IEEE, 2010, pp. 829–835.

[5] K. Kobert, T. Flouri, A. Aberer, and A. Stamatakis, "The Divisible Load Balance Problem and Its Application to Phylogenetic Inference," in *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2014, pp. 204–216.

[6] T. Flouri, F. Izquierdo-Carrasco, D. Darriba, A. J. Aberer, L.-T. Nguyen, B. Q. Minh, A. V. Haeseler, and A. Stamatakis, "The Phylogenetic Likelihood Library," *Systematic Biology*, vol. 64, no. 2, pp. 356–362, 2015.

[7] A. Stamatakis, T. Ludwig, H. Meier, and M. Wolf, "Accelerating Parallel Maximum Likelihood-Based Phylogenetic Tree Calculations Using Subtree Equality Vectors," in *Supercomputing, ACM/IEEE 2002 Conference*, 2002, pp. 40–40.

[8] F. Izquierdo-Carrasco, S. A. Smith, and A. Stamatakis, "Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees," *BMC Bioinformatics*, vol. 12, no. 1, pp. 1–14, 2011.

[9] M. Christou, M. Crochemore, T. Flouri, C. S. Iliopoulos, J. Janouek, B. Melichar, and S. P. Pissis, "Computing All Subtree Repeats in Ordered Ranked Trees," in *Proceedings of the 18th International Conference on String Processing and Information Retrieval*, ser. SPIRE'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 338–343.

[10] T. Flouri, K. Kobert, S. P. Pissis, and A. Stamatakis, "An Optimal Algorithm for Computing All Subtree Repeats in Trees," in *Combinatorial Algorithms*. Springer Berlin Heidelberg, 2013, no. 8288, pp. 269–282.

[11] K. Kobert, T. Flouri, and A. Stamatakis, "Efficient detection of repeating sites to accelerate phylogenetic likelihood calculations," *bioRxiv*, 2015.

[12] M. Sathe, O. Schenk, and H. Burkhart, "Solving Bi-objective Many-Constraint Bin Packing Problems in Automobile Sheet Metal Forming Processes," in *Evolutionary Multi-Criterion Optimization*, M. Ehrgott, C. M. Fonseca, X. Gandibleux, J.-K. Hao, and M. Sevaux, Eds. Springer Berlin Heidelberg, 2009, no. 5467, pp. 246–260.

[13] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '09. New York, NY, USA: ACM, 2009, pp. 31–40.

[14] M. Sindelar, R. K. Sitaraman, and P. Shenoy, "Sharing-aware Algorithms for Virtual Machine Colocation," in *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '11. New York, NY, USA: ACM, 2011, pp. 367–378.

[15] T. Flouri, K. Kobert, S. P. Pissis, and A. Stamatakis, "An optimal algorithm for computing all subtree repeats in trees," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 372, no. 2016, p. 20130140, 2014.

APPENDIX

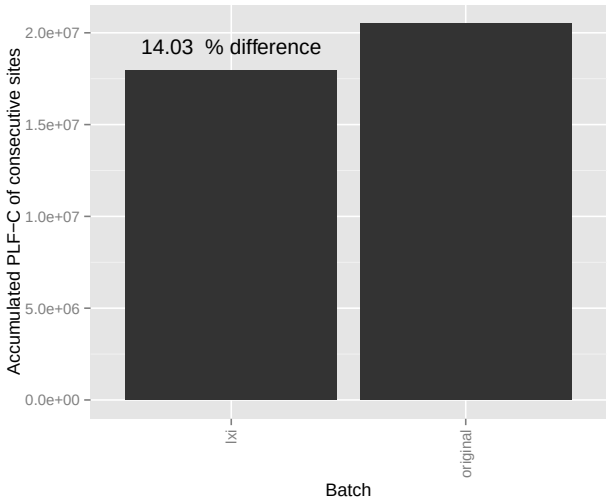


Figure 9. Lexicographically sorted versus unsorted PLF-C between all pairs of consecutive sites, accumulated over all sites, all 1200 trees, and all 8 MSAs.

Figure 10. PLF-C savings when employing the SR technique for MSA 3

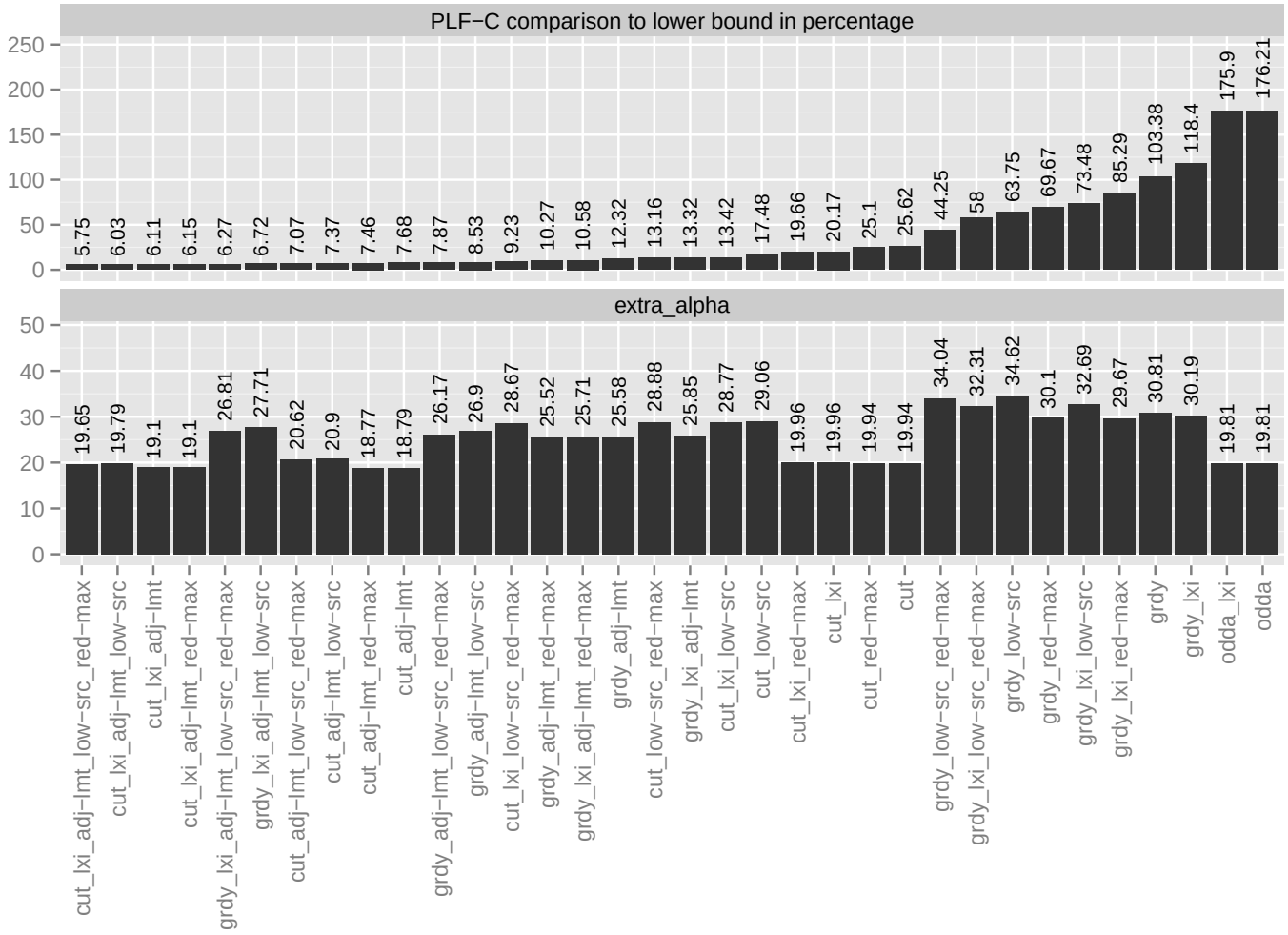


Figure 11. Comparison of all 32 heuristic component combinations, where, cut = cut approach; grdy = greedy approach; lxi = lexicographic sorting of MSA; adj-lmt = adjust limit reshuffling strategy; low-src = low SRC reshuffling strategy; red-max = reduce max reshuffling strategy.
The upper part presents the average PLF-C to $\mathcal{L}$ percentage of the heuristics over all MSAs and numbers of CPUs. The lower part presents the average $extra_\alpha$ for the heuristics over all MSAs and numbers of CPUs.

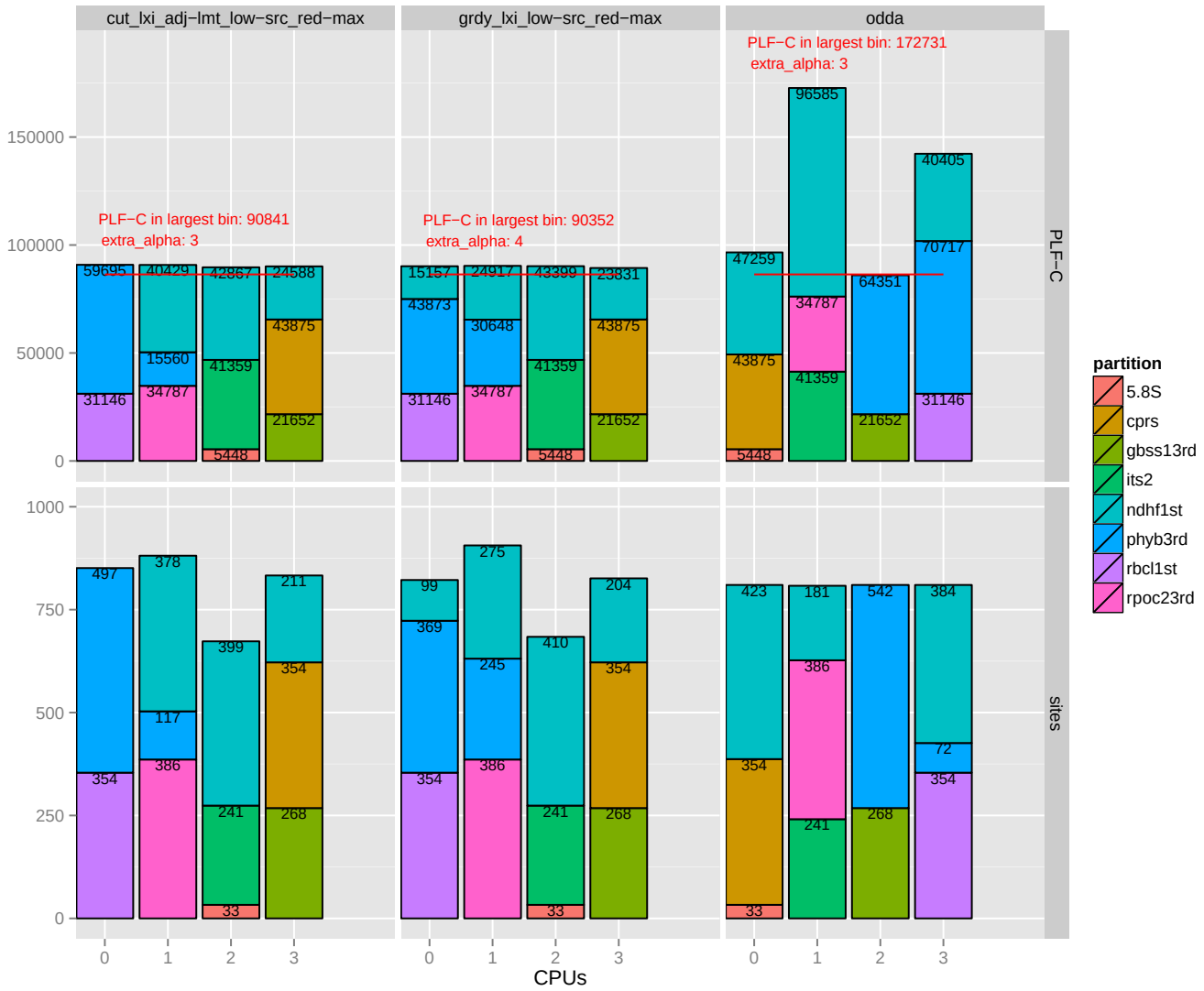


Figure 12. MSA 1 distributed across 4 CPUs. The upper part of the graph presents the data distribution in terms of PLF-C, the lower part in terms of sites. The red horizontal bar indicates the lower bound $\mathcal{L}$. In this example the *greedy approach* in conjunction with a lexicographic MSA sort, the *low SRC* and *reduce max* reshuffling strategies performs best. The $extra_\alpha$ value shows that one more redundant $P(t)$ calculation is required than for the data distribution of the *cut approach*. The ODDA balances the partitions in terms of sites, but this deteriorates PLF-C values for SR-based PLF implementations.