



Efficient discrete firefly algorithm for Ctrie based caching of multiple sequence alignment on optimally scheduled parallel machines

Soniya Lalwani¹ ✉, Harish Sharma¹, Abhay Verma¹, Rajesh Kumar²

¹Department of Computer Science & Engineering, Rajasthan Technical University, Kota, India

²Department of Electrical Engineering, Malaviya National Institute of Technology, Jaipur, India

✉ E-mail: slalwani.math@gmail.com

ISSN 2468-2322

Received on 27th October 2018

Revised on 2nd April 2019

Accepted on 10th April 2019

doi: 10.1049/trit.2018.1040

www.ietdl.org

Abstract: This study introduces a two-level strategy for efficient execution of multiple sequence alignment (MSA) of complex heterogeneous sequences. The two levels of the proposed technique are comprised of: designing the discrete firefly algorithm (DFFA) for the formation and implementation of makespan minimisation on parallel machines, followed by performing Ctrie-based caching for pairwise alignment to reduce the load on the data servers for handling multiple queries. The proposed strategy addresses a multi-client problem that aims to acquire the full advantage of the computational power of parallel connected machines. Further, it is shown that the inclusion of Ctrie as caching mechanism successively improves the performance of the system with accretion in several sequences. Performance of proposed DFFA is also compared with discrete versions of four swarm intelligence based algorithms at the criteria of makespan minimisation and the rate of convergence on two kinds of complex and diverse datasets. The work is unique in this sense: it is the first swarm-intelligence-based implementation for the addressed problem; it is so far the first approach for Ctrie based caching of the MSA on the scheduled parallel machines; hybridisation of DFFA with Ctrie for caching the MSA results is also a novel implementation.

1 Introduction and problem description

Multiple sequence alignment (MSA) is one of the most expensive NP-complete problems from the field of Bioinformatics. It is a process of positioning DNA, RNA or protein sequences into a rectangular array to have the column residues to be either homologous or super-imposable or perform the same functional role. Being homologous sequences means that the sequences have come from the same ancestral sequence, whereas being super-imposable means the sequences have a rigid local structural alignment. Unfortunately, the time required to perform MSA increases exponentially with the number of sequences and lengths of the sequences. Standard alignment algorithms are heuristic and most of them are progressive or hierarchical [1]. The higher computational complexity of most of these algorithms requires high computational speed and memory usage, to harmonise with the speedy growth in available sequence data [2]. Moreover, the multiple queries generated on the data servers performing MSA, also escalate the speed and time complexity of the problem. If the alignment of the most common sequences may be cached, it will enhance the performance with reduced load on the data server.

Continuous efforts are being made to sort out the complexity issues of MSA tools [3]. For this purpose, data caching could serve itself as a cost-effective tool. Cache-aware lock-free concurrent hash tries (Ctrie) is a non-blocking implementation of hash array mapped trie that uses single-word compare-and-swap (CAS) operations, keeping a minimum amount of information in the internal nodes. Since all the parallel processes remain linearisable and lock-free in Ctrie, the complexity of the problem gets reduced [4]. As soon as the concept of parallelisation is introduced for reducing the complexity, it necessitates the scheduling of parallel machines. This scheduling is considered as the makespan minimisation problem, which is an NP-hard combinatorial optimisation problem.

In this paper, addressed makespan problem is solved by developing a discrete version of the firefly algorithm (DFFA) for parallel machines connected through a message passing

interface (MPI). The proposed DFFA performs significantly better than compared discrete versions of swarm intelligence based algorithms (DSIAs), i.e. shuffled frog-leaping algorithm (SFLA) [5], spider monkey optimisation (SMO) [6], ant colony optimisation (ACO) [7] and particle swarm optimisation (PSO) [8].

To improve the processing performance of parallel machines in a multi-client environment, two caching mechanisms B-tree and Ctrie are evaluated for caching the computed alignment results of ClustalW. It is concluded through statistical results analysis that Ctrie caching mechanism significantly improves the processing performance as compared to B-tree. For evaluating the impact of makespan and caching in ClustalW performance, heterogeneous MSA datasets BALiBASE 4 [9] and MUSCLE [10] are used, respectively. Dataset 1 from BALiBASE 4 contains 218 sequence sets, in which sequence range is 4–807 sequences with length range 19–4895 nucleotides. Dataset 2 from MUSCLE contains 1681 sequence sets, in which sequence range is 2–50 sequences with length range 24–1132 nucleotides.

Further subsections provide a short description of an MSA with ClustalW, caching (using B-tree and Ctrie), FFA, and makespan minimisation. In Section 2, the details regarding proposed workflow are provided that includes MPI based parallel implementation of Ctrie; employing DFFA for makespan optimisation and; addressing the MSA problem in a multi-client environment. Further, Sections 3–5 provide details about experimental setup and results followed by the conclusion.

1.1 Multiple sequence alignment

Sequence alignment is an approach of positioning the sequences of DNA, RNA, or protein to determine regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. MSA is an alignment of more than two sequences acquired by inserting gaps (-) into sequences such that all the derived sequences have length l and can be represented as a matrix of n rows and l columns. Each aligned column indicates a homologous position. MSA process is

exemplified in Fig. 1. Fig. 1a presents the unaligned sequences, whereas Fig. 1b presents the aligned sequences obtained by inserting gaps at suitable places to optimise alignment score. Initially, gaps are inserted at random positions, and then the process gets iterated for inserting the gaps at suitable positions, to obtain a maximum alignment score. The coloured columns represent the columns with maximum matches. Here, $n = 7$; $l = 60$ for the problem presented in Fig. 1. The accretion in the values of n and l increases the complexity of the MSA problem [11, 12].

The progressive-alignment strategy is one of the most commonly used heuristic methods for MSA. Progressive-alignment approach develops final MSA by integrating pairwise alignments commencing with the most similar sequence pair and progressing to the most distantly related pairs. The widely used profile-based progressive multiple alignment technique ClustalW was introduced by Thompson *et al.* of EMBL, EBI [13]. The process presented in Fig. 2 is elaborated in the same order as numbered here (i) Given unaligned sequences. (ii) Perform " C_2 " pairwise alignments considering each sequence pair. Here, " C_2 " stands for the number of possible combinations from n sequences taking two sequences at a time formulated as " $C_2 = n!/(n-2)!$ ".

Then, evaluate the distance between each pair of sequences by pairwise alignments and build a distance matrix. (iii) Build up a neighbour-joining guide tree using the distance matrix. (iv) Decide the order of proceeding for the progressive alignment from the guide tree. (v) Obtain the aligned sequences [14].

1.2 Firefly algorithm (FFA)

Computing techniques that are inspired by nature are known as nature-inspired computing techniques. Most popular nature-inspired computation techniques include evolutionary computation, swarm intelligence (SI), neural computation, artificial immune systems, cellular automata, membrane computing, quantum computation, and molecular computing. SI is originally the emergent collective behaviour of groups of simple agents. SI-based algorithms are

required to be self-organised and decentralised; flexible towards internal and external changes and robust upon failure of individuals. Past two decades have witnessed the flourishing of several SI based algorithms [15, 16]. SIAs have a few common characteristics, i.e. all are population-based; computations are carried out by self-governing agents, and the information exchange between agents is performed using social behaviour [17]. The most salient SIAs include PSO, ACO, SFLA, SMO and FFA. This paper proposes the minimisation of makespan by FFA, along with the performance comparison from other SIAs.

Yang developed FFA in 2007 and 2008 [18], inspired by the bioluminescence flashes of fireflies (FFs). The flashes have the basic functions of attracting the potential prey and the mating partners. Also, this flashing reminds the FFs as a protective warning mechanism about the potential predators. The important factors of the signal system of this flashing are the rate of flashing, rhythm in flashes and the amount of time between flashes. They introduced a term namely attractiveness in the algorithm, that is determined by the brightness of FF's flashes and the accuracy in the timing of their flashing patterns. With the increase of distance from the flashing source, the light intensity decreases. The attractiveness is a highly non-linear term due to: the exponential decay in light absorption; and inverse-square law of light variation concerning distance. The position update equation in FFA is

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}} (x_j^t - x_i^t) + \alpha \epsilon_i^t \quad (1)$$

here α is a scaling factor that is responsible for controlling the step sizes of random walks; γ is responsible for controlling the visibility of the FFs; r_{ij} is the Cartesian distance between two FFs i and j ; n is the population size; β_0 is the attractiveness constant (when $r_{ij} = 0$). The second term in (1) represents the non-linear attractiveness, whereas the third term is randomisation associated to a random vector ϵ_i drawn from a Gaussian distribution or uniform distribution. In general, a maximisation problem contains the brightness directly proportional to the value of the objective function. The steps followed by a standard FFA are summarised below for iteration counter t :

- The objective function $f(x)$ is formed for $x = (x_1, x_2, \dots, x_d)^T$.
- Parameters α , β , γ and ϵ are initialised.
- An initial population of fireflies, i.e. x_i ($i = 1, 2, \dots, n$) is generated.
- The fitness of the initial population, i.e. $f(x_i)$ for $i = 1, 2, \dots, n$ is evaluated.
- Then, the light intensity I_i at x_i is determined proportional to $f(x_i)$.
- **while** $t < \text{MaxGeneration}$
for $i=1: n$
for $j=1: n$
if $(I_i < I_j)$
The FF i moves towards j (by (1)).
end if
Now, the attractiveness is updated with distance r by $e^{-\gamma r^2}$.
The new solutions are evaluated and I_i is updated.
End for
End for
The solutions are ranked. The best solution is updated.
The iteration counter is updated by $t = t + 1$.
Numerical value of α is reduced by: $\alpha = \alpha * \mu$
Here μ stands for the damping ratio factor for α .
end while
• The results are post-processed and visualised.

The parameters α , β , γ and ϵ can be adjusted concerning the problem and its scale.

Although, standard FFA is a powerful algorithm, there is always a scope of improvement in every algorithm. Hence, several variants have been proposed in the literature for improving the performance and convergence of the algorithm. Proposed work develops a discrete variant of FFA for optimising the makespan.

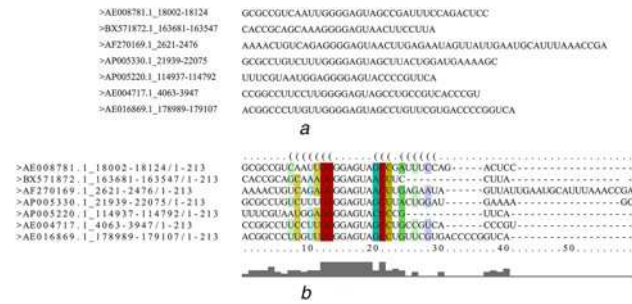


Fig. 1 Multiple sequence alignment

a Unaligned sequences
b Aligned sequences

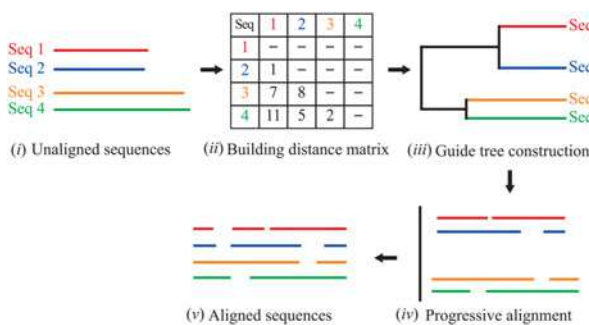


Fig. 2 ClustalW based progressive alignment

1.3 Makespan minimisation

Makespan minimisation is an NP-hard problem that addresses the scheduling of m parallel machines with n independent jobs, formulated as

$$\begin{aligned} \min \{M\} \\ M - \sum_{i=1}^n t_i p_{ij} > 0, \text{ for } j = 1, 2, \dots, m \\ \text{where } M = \text{Max}\{C_j\}, \text{ for } j = 1, 2, \dots, m \end{aligned} \quad (2)$$

here M denotes the makespan and C_j is the completion time of the last job on the machine j . M is maximum from the completion times of last jobs on all machines. For the i th job with processing time t_i , the formulation becomes

$$\sum_{j=1}^m p_{ij} = 1 \quad \text{for } i = 1, 2, \dots, n \quad (3)$$

with

$$p_{ij} = \begin{cases} 1, & \text{if the } i\text{th job is assigned to the } j\text{th machine} \\ 0, & \text{otherwise} \end{cases}$$

Further, the parallel processes are implemented through MPI libraries, which are comprised of function and macros for the programs in C, C++ and FORTRAN. MPI supports more flexible communication method than MapReduce. Hence MPI is implemented here in place of MapReduce or Spark. MPI is basically intended to use for the programs that aim to exploit the presence of multiple processors [19].

1.4 Implementation of B-tree and Ctrie for caching

Cache memories have been a major discovery for reducing the processing time of the frequently occurring operations. Cache-enabled processes are found to be more impressive if work is done with heavy processing. The data structures such as trees and linked lists have implemented them for fast processing. Besides B-tree and Ctrie, other caching approaches include:

- **Hash table:** It is quite an old data structure which has concurrent implementations, but they lock the whole table for read and write operations.
- **Classical tries:** They are not concurrent and thus do not allow us to utilise multiple nodes.
- **Database based caching:** It comes with a lot of overheads, on the top of that most of them use B-tree or hash tables for their implementation.
- **Concurrent hash maps:** Their concurrency is restricted to operations not interfering within the same segments.

B-tree and Ctrie are chosen here for comparison, because, B-tree is the conventionally used caching mechanism, and Ctrie overcomes most of the limitations of B-tree. Effect of caching has been discussed in our previous work for dataset 1 with B-tree [20].

1.4.1 B-tree: B-tree is a balanced multi-way disk tree, which is suitable for managing a large set of strings, hence reduces the number of disk accesses. B-tree is pretty useful while working with data warehouses, high dimensional databases, geographic databases, spatial databases, multimedia databases, text retrieval

systems and so on [21]. Despite the qualities, B-tree has some problems like

- While processing the nodes, the complexity increases, because of the several steps involved in splitting the nodes.
- Performing the operations like prefix searches, skewed access and so on, are inefficient in B-trees.
- As B-tree does many repeated searches, this data structure is not suitable for applications such as search engines.
- Also for concurrent query systems, it is not very efficient due to more overhead.
- Therefore, in this paper, an alternative and the efficient caching mechanism is adopted, namely Ctrie, which is based on a concurrent hash trie with cache-awareness and lock-free structure.

1.4.2 Ctrie: Each hash trie node stores references to sub-tries, thus cache-awareness is ensured by Ctrie in the system. Also, Ctrie is completely lock-free and relies only on the CAS instructions. CAS instructions are already proven more reliable to avoid overheads in the system [22]. Starting from trie trees [23, 24], then moving towards the hash trie [25], followed by its non-blocking implementation has provided cache-aware lock-free concurrent hash tries or simply the Ctries [4].

The concept starts with a concurrent object, which is an abstract data type, which can handle concurrent atomic operations. This object can be implemented in shared memory with its synchronisation primitives such as read, write, fetch-and-add, and CAS. The insertion and removal operations of Ctries are explained as follows:

- If a thread T_1 needs to insert any key K_1 below a node C_1 , the first indirection node I_1 is created (*Inode*) (as shown in Fig. 3b), which is initialised to C_1 and it remains present in the Ctrie even if nodes above and below it change.
- *Inode* has a single field main and it is created whenever a level of nodes is created. The CAS instruction is operated over this *Inode* and the updated node, which has the newly inserted node.
- Similarly, for removing any key from the data structure as shown in Fig. 3c, Tomb node (*TNode*) is used which is again a special type of node that can hold a single key.
- No thread can modify an *Inode*, if it contains a *TNode*. So, the CAS operation is needed to be done on that *Inode* which is set to the Tomb node and then, the thread can contract C_1 .
- Both the operations in Ctrie, are done atomically and no key is lost during the process. Hence, this approach can be applied specifically to the tree-based data structures.

A detailed description regarding the comparative effectiveness of Ctrie and B-tree caching strategies is presented in Section 4.

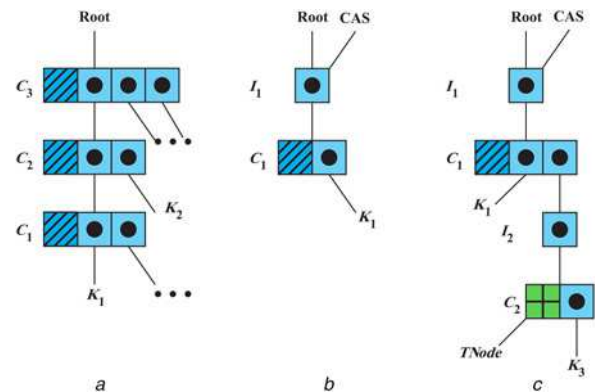


Fig. 3 a Hash array mapped trie; b Insertion operations in Ctrie; c Removal operations in Ctrie

2 Proposed workflow and methodology

Proposed work is so far the first kind of implementation of SI based algorithm, namely DFFA for the addressed problem. Employing Ctrie based caching of MSA is a first-time implementation so far. Also, the proposed hybridised DFFA with Ctrie based strategy pursuit on the parallelly connected machines, scheduled by makespan minimisation is a novel work. Moreover, it is the first comparison of any kind between the performance of Ctrie and B-tree.

Proposed work performs caching on the alignment output of ClustalW. As discussed in Section 1.1, ClustalW performs a progressive alignment with three intermediate outcomes, i.e. acquiring pairwise aligned sequence sets followed by the creation of distance matrix and construction of guide tree. The cache impact on ClustalW has been analysed in our previous work [20]. It was discussed that pairwise alignment takes a big fraction of time, so the pairwise alignment results are cached for performing the query execution of new sequence-sets faster. Fig. 4 presents the proposed work structure of makespan minimisation on parallel machines using DFFA for caching of MSA, which is divided into two levels as follows:

Level 1 is comprised of two steps, i.e.

Step 1: Job analysis is done by measuring complexity (n^2l^2) of pairwise progressive alignment for each sequence set, where n is the number of sequences in a sequence set and l is the average sequence-length of that sequence set. The sets are arranged in descending order of their complexity values.

Step 2: DFFA is run for obtaining the optimal schedule. Details of DFFA are presented in the next subsection.

Level 2 is also comprised of two steps:

Step 1: It handles the job control on the cluster. It distributes the sequence sets on parallel machines and starts the caching server.

Step 2: The whole process of caching in MPI based parallel environment is pursued now. Before performing any pairwise alignment, the slave sends a query to the master server for the decision regarding storing or fetching the cache results. Data is encapsulated in the byte stream for fetch operation. A new thread is created by the master server for extracting the given sequence; it is searched in the cache. The sequence names may or may not be unique; hence, a unique 64-bit hash of sequence is assigned to each sequence. The sequence length is concatenated with the unique 64-bit identifier of that sequence to avoid a hash collision. If a score is returned from the server, then slave need not do any calculation for that pair. If -1 is the returned value, usual calculations are done and the results are sent to the server for future reference. Hence, every pair is calculated only once on the network. The message is sent in the form of a byte stream in message passing approach. Two types of structure are held based on the type of query: if the first byte = 1, then it is a fetch query with structure 1; and if the first byte = 2, then it is a store query with structure 2. The action is chosen based on first bit, i.e. for a fetch query, the sequence is searched through the Ctrie and if a match is found, then the score is sent, otherwise, -1 is sent; for the store query, the given score is saved. Once any node completes its job, all the results are sent back to the master server, which includes aligned sequences and others statistics like cache hit-ratio, the time taken by individual job and the total time of completion of all the jobs.

2.1 Proposed DFFA

The proposed version of FFA is designed to address the makespan problem. Hence, the algebraic operations addition, subtraction, multiplication are designed in a specific way, to obtain the results in terms of the machine number and assignment number. Moreover, one more operation, i.e. the best position is designed here represented as $\odot$. The representation for the other symbols

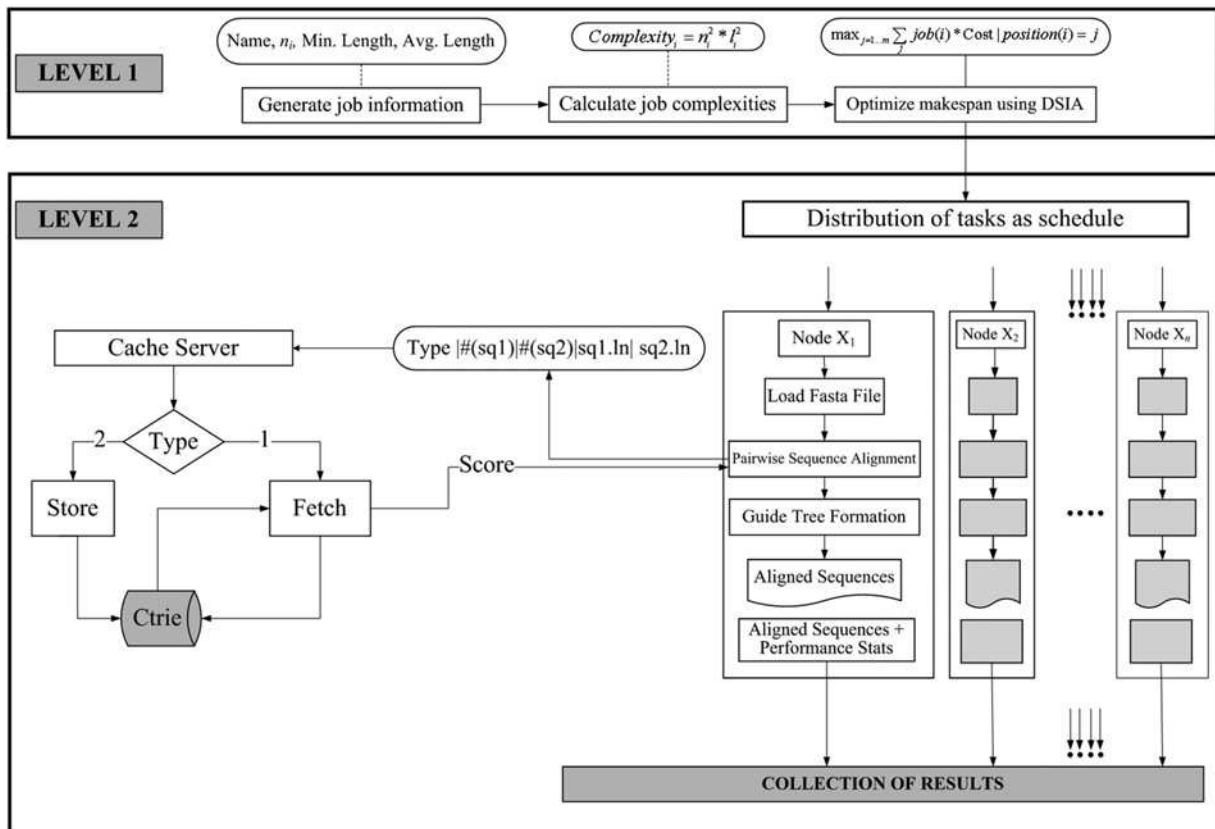


Fig. 4 Proposed work structure

is $\oplus$ for addition; $\ominus$ for subtraction; and $\otimes$ for multiplication. Fig. 5 exemplifies these operations for eight assignments (jobs), i.e. $a_1, a_2, \dots, a_8$ for four machines, i.e. m_1, m_2, m_3 and m_4 . Here, $S_1, S_2, \dots, S_8$ each represents a possible combination of assignments

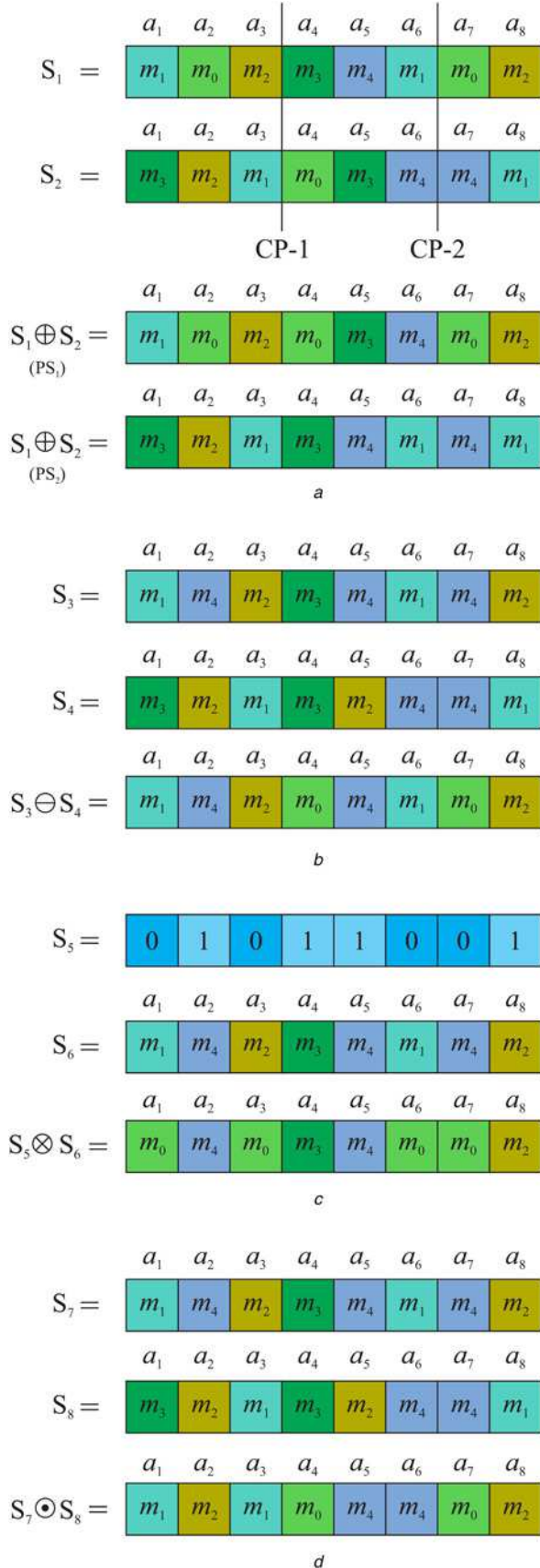


Fig. 5 Illustration of $\oplus$, $\otimes$, $\ominus$, $\odot$ operations

on machines, i.e. a sequence of machines that forms a possible solution.

Figs. 5a–d elaborate the operations applied between two solutions for generating a new solution. Fig. 5a presents the $\oplus$ operation that basically performs a crossover operation between two solutions. This figure contains two cut-points, i.e. CP-1 and CP-2, hence two solutions can be generated, i.e. PS₁ and PS₂. The solutions are obtained by interchanging the jobs between the machines, i.e. in PS₁ first three assignments are obtained from S_1 , the next three assignments are obtained from S_2 and the last two are obtained from S_1 again. Similarly, in PS₂ first three are from S_2 , next three assignments from S_1 and the last two are from S_2 . The algorithm randomly selects a solution from PS₁ and PS₂.

Fig. 5b presents the $\ominus$ operation that evaluates the hamming distance between the best solution (i.e. *pbest* or *gbest*) and the current position. The operation output remains the best one (i.e. from S_3 always), until the machines in the same column (from S_3 and S_4) are not equal. As soon as they are equal, the output becomes m_0 . In that case, the assignment follows the longest processing time (*LPT*) rule. In this rule, the machine with minimal workload iteratively obtains the assignment of the longest processing time. Further, Fig. 5c presents the $\otimes$ operation that is based on performing the Hadamard product between a binary sequence S_5 and a solution sequence S_6 . The product with 1 provides the same assignment, whereas the product with 0 provides m_0 that is assigned by *LPT* rule. In Fig. 5d, the $\odot$ operation is presented that stands for a random assignment out from S_7 and S_8 . Here, $S_7 \odot S_8$ provides the output with random assignments, i.e. sometimes from S_7 and sometimes from S_8 .

Further, these operations are implemented for a proposed discrete version of FFA, i.e. DFFA. All the steps mentioned below for DFFA, are presented in the order of standard FFA to provide a clear description of DFFA.

- The objective function $f(S)$ is formed for $S = (S_1, S_2, \dots, S_d)^T$ for minimisation objective. Here, each S_i presents a possible solution alike $S_1, S_2, \dots, S_8$ in Fig. 5. The objective function taken here is presented by (2).
- Parameters α, β, γ and ϵ are initialised.
- An initial population of n possible solutions is generated.
- The fitness of the initial population is evaluated by (2).
- Then, the light intensity I_i is determined proportionally to S_i .
- while $t < \text{MaxGeneration}$
 - for $i=1: n$
 - for $j=1: n$
 - if $(I_i > I_j)$
 - The position of the i th FF is updated by

$$S_i^{t+1} = \text{LPT}(S_i^t \oplus \beta_0 e^{-\gamma_{ij}^2} \otimes (S_j^t \ominus S_i^t) \oplus \alpha \epsilon_i^t) \quad (4)$$

end if

- Now, the attractiveness is updated concerning the distance r by $e^{-\gamma_{ij}^2}$.
- The new solutions are evaluated by (2) and I_i is updated that is equal to the output.

End for

End for

- The solutions are ranked, and the best solution is updated.
- The iteration counter is updated by $t = t + 1$.
- Numerical value of α is reduced by the damping ratio factor.

end while

- The results are post-processed and visualised.

3 Experimental setup

Nine general purpose computers equipped with Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz running Ubuntu 16.04 with the main memory of 4 GB (DIMM DDR3 Synchronous 1600 MHz) on each of them are dedicated to experimenting. MPI server is built for the datasets with master–slave strategy, one master server and eight general purpose computers as slaves. All the programming part of

SIA is performed in the MATLAB programming environment, whereas MPI based programming is done in C++. Statistical testing by employing one way ANOVA succeeded by Bonferroni post-hoc analysis is performed for determining the significance of the results in SPSS v 23.0. At the part of network bandwidth, packets consist of around 200 bytes of data and those requests were dependent on the number of pairs aligned by per node per second. For ten alignments per second, 2 KBs of data were sent per node per second. This would not be affecting the efficiency until nodes are increased up to thousands.

3.1 Benchmark datasets

As shown in Table 1, dataset 1 that is BALiBASE 4 dataset [9], contains 218 sequences sets with a minimum of 4 sequences in a set and maximum 807 sequences in a set. All the numbers (i.e. average length, average number of sequences) are rounded off to the nearest integer. Parameters, range, and average are measured for all sequences in a sequence set. Minimum sequence length ($L_{\min}$) in dataset 1 is 19 nucleotides and maximum sequence length ($L_{\max}$) is 4895 nucleotides with an average length (L_{avg}) 460. Similarly, the complexity is discussed: $C_{\min}$ as the minimum complexity of dataset, $C_{\max}$ as the maximum complexity, and C_{avg} as the average complexity. The same parameters are presented for dataset 2 that are MUSCLE datasets [10].

The diversity of datasets is maintained as per the need of studying the impact of caching and makespan minimisation. The purpose of employing dataset 1 is to study the impact of caching. Dataset 1 contains a wide range in the number of sequences in a sequence set, i.e. the upper range is 807 sequences in a sequence set. Hence, the number of pairwise alignments will be equal to $^{807}C_2$, which will exhibit the effectiveness of caching these pairwise alignment results. The purpose of employing dataset 2 is to evaluate the impact of makespan minimisation. Dataset 2 contains a large number of sequence sets, i.e. 1681 sequence sets. Scheduling a large number of sequence sets will effectuate the role of makespan minimisation.

3.2 Parameter setting in DSIA

All the DSIA are run for a maximum of 2000 function evaluations. All the algorithms are initialised with a randomly generated solution.

Table 1 Details of dataset 1 and 2

Parameters	Dataset 1	Dataset 2
no. of sequences	218	1681
range (no. of seqs.)	4–807	2–50
average	82	45
$L_{\min}$	19	24
$L_{\max}$	4895	1132
L_{avg}	460	233
$C_{\min}$	1,846,881	25,600
$C_{\max}$	123,792,823,927	1,808,715,841
C_{avg}	4,292,008,673	162,710,567

Table 2 Comparison between Ctrie and B-tree for the store and fetch time at different number of sequences

No. of sequences	Store				Fetch			
	B-tree		Ctrie		B-tree		Ctrie	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
20,000	21,766,188.4 ^b	2,390,554.1	24,118,005.6	7,051,562.8	18,902,740.4 ^a	325,221.7	9,089,797.2	657,549.3
40,000	54,686,835.6 ^a	3,362,447.4	29,323,316.4	6,809,648.0	27,654,883.6 ^a	421,961.7	14,526,542.8	2,501,042.7
60,000	90,431,790.8 ^a	5,437,901.7	34,279,503.6	1,271,415.3	44,093,418.6 ^a	2,403,834.7	16,970,132.8	2,881,572.9
80,000	89,236,187.6 ^a	3,702,840.9	43,019,107.8	3,204,175.1	63,955,107.4 ^a	384,233.4	17,264,361.6	1,282,207.1
1,00,000	120,482,481.4 ^a	9,618,379.8	63,535,051.4	9,105,441.1	88,511,542.2 ^a	550,229.3	20,740,810.6	1,024,188.1

^aSignificant difference.

^bNon-significant difference.

The population size remains 50, and dimension equal to the number of sequence sets. The parameters $\text{var}_{\min}$ and $\text{var}_{\max}$ are decided by the number of minimum and maximum slaves, i.e. 1 and 8; respectively. For DPSO, $c_1 = c_2 = 1/3$; for DSFLA, $m = 5$, $\alpha = 3$; $\beta = 5$; $\gamma = 2$ and 50 frogs in each memplex; for DACO, $\alpha = 1$, $\beta = 12$, $\rho = 0.5$ and in DSMO, local limit count=15; global limit count=30 with $pr=0.1$. For DFFA, the parameters are $\alpha = 0.2$, $\mu = 0.98$, $\beta_0 = 1$ and $\gamma = 1$.

4 Simulation results

The proposed discrete version of FFA is compared with our previously developed versions of SIAs (i.e. DSFLA, DSMO, DACO and DPSO) [26] for makespan minimisation, implemented in MATLAB programming environment. A comparative study is performed between all discrete SIAs at the criteria of time and cost minimisation for the makespan. Here, time represents the time taken by the algorithm to schedule the task (i.e. time taken to optimise the makespan) and the cost (i.e. makespan) represents the number of cycles proportional to the time taken by the system to complete the scheduled MSA jobs that include distribution of jobs on parallel machines, pairwise alignment along with simultaneous caching followed by collecting the aligned sequences on the master server. Tables 2–5 and Figs. 6–9 present the comparative studies between caching mechanisms, as well as of DFFA with other DSIA in performing their assigned processes in proposed work.

Fig. 6 depicts the clarification of replacing B-tree by Ctrie. It can be clearly observed from Fig. 6 that, as soon as the number of sequences increases from 20,000 to 100,000, the time difference between B-tree and Ctrie also gets increased. The detailed results are provided in Table 2, along with the significance testing. In Table 2, it can be observed that Ctrie is always an out-performer than B-tree for all the sequence sizes. The logic supporting the claim is explained in the next lines. Table 2 depicts that the difference between performances of Ctrie and B-tree always remains significant, except the store operation for the smallest sequence size of 20,000 sequences. Ctrie has an overhead of threads which adds an offset to the timing of Ctrie's performance, but, on the longer run, the multiple threads overtake the time taken by B-tree. For Ctrie, the complexity of hashing is the complexity of fetch operations. However, in the case of B-tree, the complexity increases as the height of tree increases. Worst-case complexities of both fetch and store in Ctrie are $O(\log_{32}(n))$ and B-tree is $O(\log_2(n))$, here n is the number of sequences. It is observed for Ctrie that the time difference between store operation is lesser than that of the fetch operation. This is because of the complexities of operation on Ctrie with a high constant factor due to the thread overhead. Ctrie heavily depends on dynamic memory allocation which adds to the complexity of store operation. Hence, store operation shows less performance boost over B-tree than the fetch operation. For 100,000 sequences the differences become preminent.

Figs. 7a and b represent the comparison of DFFA with other discrete SIAs for the average values of time and cost for datasets

Table 3 Bonferroni *post-hoc* analysis results for datasets 1 and 2 at time and cost criteria

Discrete	DSFLA		DSMO		DACO		DPSO		DFFA	
SIAs	<i>T</i>	<i>C</i>	<i>T</i>	<i>C</i>	<i>T</i>	<i>C</i>	<i>T</i>	<i>C</i>	<i>T</i>	<i>C</i>
Dataset 1										
DSFLA	—	—	a	b	a	b	a	b	a	a
DSMO	—	—	—	—	a	b	a	b	a	b
DACO	—	—	—	—	—	—	a	a	a	a
DPSO	—	—	—	—	—	—	—	—	a	b
DFFA	—	—	—	—	—	—	—	—	—	—
Dataset 2										
DSFLA	—	—	a	a	a	a	a	a	a	a
DSMO	—	—	—	—	b	b	a	b	a	b
DACO	—	—	—	—	—	—	a	b	a	b
DPSO	—	—	—	—	—	—	—	—	a	b
DFFA	—	—	—	—	—	—	—	—	—	—

^aSignificant difference.

^bNon-significant difference.

Table 4 Ranking of discrete SIAs at time and cost criteria

SIAs	Time		Cost	
	Dataset 1	Dataset 2	Dataset 1	Dataset 2
DSFLA	2	2	4	5
DSMO	4	3	3	2
DACO	3	4	5	4
DPSO	5	5	2	3
DFFA	1	1	1	1

Table 5 Impact of multi-threading over the performance of the algorithm

Number of threads	Machine utilisation index	Best cost (cycles)
1	1	20,856,093,538
2	1	10,428,046,816
3	1	6,952,047,359
4	0.99999	5,214,053,488
5	0.99997	4,171,323,228
6	0.99996	3,476,169,903
7	0.99994	2,979,631,953
8	0.99991	2,607,245,515
9	0.99987	2,317,636,655
10	0.99983	2,085,963,017
11	0.99976	1,896,463,911
12	0.99963	1,738,647,355
13	0.99974	1,604,732,082
14	0.99947	1,490,507,892
15	0.99939	1,391,259,696
16	0.99941	1,304,281,380
17	0.9993	1,227,689,034
18	0.99939	1,159,383,994
19	0.99926	1,098,498,925
20	0.99932	1,043,516,096
30	0.99817	696,478,700
40	0.99678	523,088,260
50	0.99357	419,821,679
60	0.99303	350,041,583
70	0.92041	323,707,610
80	0.80536	323,707,610
90	0.71588	323,707,610
100	0.64429	323,707,610

1 and 2, respectively. The results for all the SIAs for 2000 function evaluations are tested for statistical significance by one-way ANOVA followed by Bonferroni *post-hoc* analysis summarised in Table 3. Here, *T* represents the time and *C* represents the cost, i.e. makespan time. For dataset 1, it can be observed from Fig. 7a and Table 3 that DFFA is an out-performer at both the criteria, i.e. time and cost, but the difference between the performance of DFFA, DSMO and DPSO is not significant at cost criteria. Whereas at the time criteria the performance of DFFA is significantly better than all the compared DSIs.

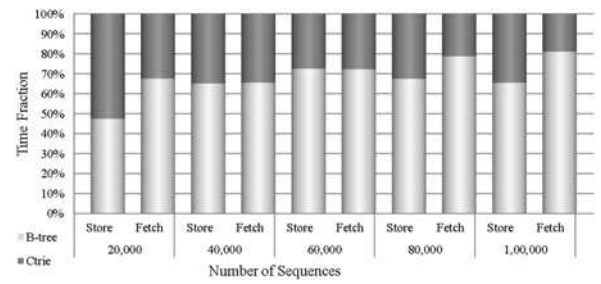


Fig. 6 Comparison for the store and fetch time between Ctrie and B-tree

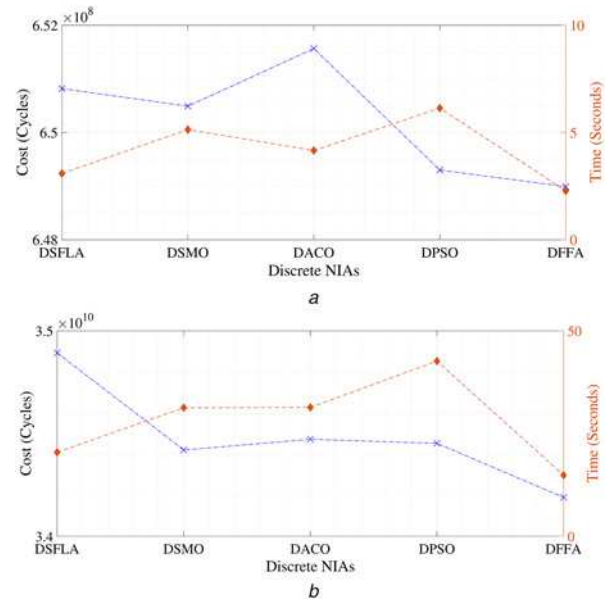


Fig. 7 Cost and time comparison between all SIAs for

a Dataset 1

b Dataset 2

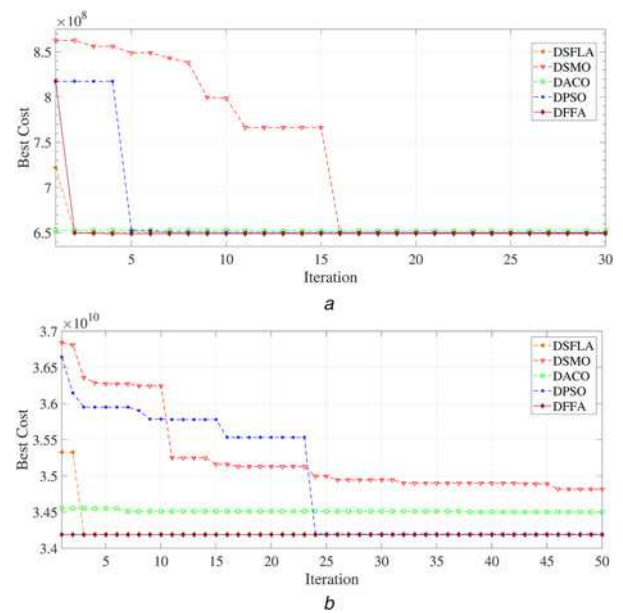


Fig. 8 Convergence plot of all SIAs for

a Dataset 1

b Dataset 2

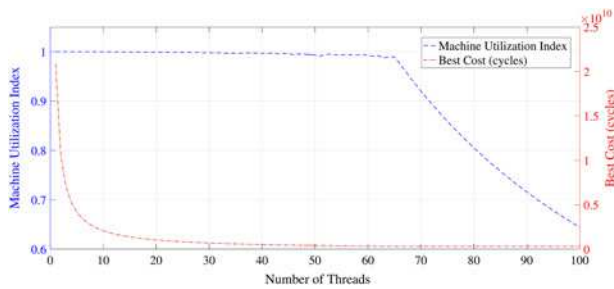


Fig. 9 Impact of the increased number of threads over the performance of the algorithm

At the cost criteria for dataset 2, DFFA performs significantly better than DSFLA, whereas no significant difference is found between the performance of DFFA with DSMO, DPSO and DACO. However, at the time criteria, DFFA is significantly better than all the compared DSIs.

Further, Table 4 shows the ranking of the algorithms at a time and cost criteria for datasets 1 and 2 based at the average. The smallest rank shows the best performer, i.e. the algorithm with rank one is the best performer. Moreover, the ranking shows that at the cost and time criteria for both the datasets, DFFA stands in the first place. At time criteria, the second position is occupied by DSFLA for both the datasets, whereas at the cost criteria DPSO and DSMO remain at second place for datasets 1 and 2, respectively.

The convergence of each algorithm is tested from random initial points for datasets 1 and 2, as shown in Figs. 8a and b, respectively. It can be observed that for both the datasets, DFFA fastly converges towards the solution, followed by DSFLA.

Further, the impact of multi-threading is studied for the best performing algorithm, i.e. DFFA. The algorithm is working in a multi-threaded environment with two shared resources. First is the scheduling part of the algorithm which executes before the initiation of threads. The number of nodes improves the makespan gradually but reduces the node utilisation, which can be observed in Table 5.

The second shared resource for a multi-threaded environment is the caching server running on a Ctrie implementation. The number of threads accessing this server is equal to the number of processing nodes. Ctrie, however, is pretty scalable to handle a large number of threads. However, a large number of threads will increase the probability of the same sequence being aligned concurrently on multiple nodes thus reducing the hit ratio and subsequently reducing the makespan accuracy.

Here, the first kind of multi-threading is studied in detail, as shown in Table 5 with several threads varying from 1 to 100. The effect of increasing the threads over machine utilisation index and the best cost is studied. Since it has been verified that the time predicted by our scheduling algorithm is proportional to the experimental time, on scheduling, the jobs over n nodes will provide the result proportional to the experiment. The 'performance of n machines' is derived out of that scheduling order. Increasing the threads would result in larger bandwidth consumption, which makes the experimental results no better than the theoretical ones.

Here, machine utilisation index stands for the ratio of ideal cost and real cost. The ideal cost is determined by averaging the total cost concerning the number of machines, whereas the real cost is the actual time taken by machine to process the jobs. The cost is measured in terms of cycles and the scaling provides 22 lakh cycles = 1 s (approx). Best cost and machine utilisation index are plotted against the number of threads (machines) in Fig. 9.

It can be clearly observed that the machine utilisation index keeps on getting deteriorated with an increase in the number of threads, whereas, the best cost gets improved consistently. Also, the machine utilisation index gets a sharp deterioration at and after 70 threads, whereas, the best cost gets the last improvement at 70 threads. Hence, the process becomes useless after 70 threads. Here, the best cost with 8 machines is 2,607,245,515 cycles

(~1185 s) and with 70 machines is 323,707,610 (~147 s). Hence, it depends upon the user's need to obtain better cost with a process expensive in terms of machines or to keep a nominal number of threads for a nominal optimal cost.

Conclusively, DFFA is found an out-performer than compared discrete SIAs at all the grounds, i.e. time, cost and convergence rate. Also, Ctrie has been proven a better-caching strategy than B-tree, specially, with the increase in problem complexity.

5 Conclusion

Proposed work acquainted development of DFFA, compared with some popular DSIs, which include DACO, DSFLA, and DSMO for makespan minimisation in a parallel computing environment facilitated by Ctrie based caching to handle queries by multiple clients. Further, a comprehensive comparison was performed among developed DFFA and existing DSIs for the addressed makespan minimisation problem in MPI based parallelisation at the criteria: cost (i.e. makespan time); and the time taken to schedule makespan. Among the compared DSIs, DFFA had been concluded the most competent algorithm for the problem, embellished by statistical testing using one-way ANOVA followed by Bonferroni *post-hoc* analysis.

Further, to improve the execution performance of ClustalW in the multi-client environment, Ctrie based caching was applied on parallel connected machines. Subsequently, a fair comparison between B-tree and Ctrie caching mechanism was performed and through statistical analysis, it was proved that Ctrie is outperforming than B-tree in a parallel computing environment.

As a future scope of the proposed work, multi-level implementation with multi-objective problem formulation, while considering machine idle time as another objective, may be outlined for caching of RNA secondary structure prediction problem.

6 Acknowledgment

The first author (S.L.) gratefully acknowledges the Science & Engineering Research Board, DST, Government of India for the fellowship (PDF/2016/000008). The authors are thankful to Dr. Krishna Mohan from BISR, Jaipur, India for his valuable suggestions throughout the work.

7 References

- [1] Schatz, M.C., Trapnell, C., Delcher, A.L., *et al.*: 'High-throughput sequence alignment using graphics processing units BMC bioinformatics', *BioMed Central*, 2007, **8**, p. 474
- [2] Ma, J., Jiang, X., Gong, M.: 'Two-phase clustering algorithm with density exploring distance measure', *CAAI Trans. Intell. Technol.*, 2018, **3**, pp. 59–64
- [3] Lalwani, S., Kumar, R., Gupta, N.: 'A review on particle swarm optimization variants and their applications to multiple sequence alignment', *J. Appl. Math. Bioinf.*, 2013, **3**, pp. 87–124
- [4] Prokopec, A., Bagwell, P., Odersky, M.: 'Cache-aware lock-free concurrent hash tries', *arXiv:1709.06056*, 2011
- [5] Eusuff, M.M., Lansey, K.E., Pasha, F.: 'Shuffled frog-leaping algorithm: a memetic metaheuristic for discrete optimization', *Eng. Optim.*, 2006, **38**, pp. 129–154
- [6] Bansal, J.C., Sharma, H., Jadon, S.S., *et al.*: 'Spider monkey optimization algorithm for numerical optimization', *Memet. Comput.*, 2014, **6**, pp. 31–47
- [7] Dorigo, M., Birattari, M., Stutzle, T.: 'Ant colony optimization', *IEEE Comput. Intell. Mag.*, 2006, **1**, pp. 28–39
- [8] Kennedy, J.F., Eberhart, R.C.: 'Particle swarm optimization'. Proc. of IEEE Int. Conf. on Neural Networks, Piscataway, NJ, 1995, pp. 1942–1948
- [9] Bahr, A., Thompson, J.D., Thierry, J.C., *et al.*: 'BALiBASE (benchmark alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations', *Nucleic Acids Res.*, 2001, **29**, pp. 323–326
- [10] Edgar, R.C.: 'MUSCLE: multiple sequence alignment with high accuracy and high throughput', *Nucleic Acids Res.*, 2006, **32**, pp. 1792–1797
- [11] Lalwani, S., Kumar, R., Deep, K.: 'Multi-objective two-level swarm intelligence approach for multiple RNA sequence-structure alignment', *Swarm. Evol. Comput.*, 2017, **34**, pp. 130–144
- [12] Lalwani, S., Kumar, R., Gupta, N.: 'A novel two-level particle swarm optimization approach for efficient multiple sequence alignment', *Memet. Comput.*, 2015, **7**, pp. 119–133

- [13] Thompson, J.D., Higgins, D.G., Gibson, T.J.: 'CLUSTAL w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice', *Nucleic Acids Res.*, 1994, **22**, pp. 4673–4680
- [14] Lalwani, S., Kumar, R., Gupta, N.: 'An efficient two-level swarm intelligence approach for multiple sequence alignment', *Comput. Inf.*, 2016, **35**, pp. 963–985
- [15] Du, K.L., Swamy, M.N.S.: 'Search and optimization by metaheuristics techniques and algorithms inspired by nature' (Springer International Publishing, Switzerland, 2016)
- [16] Yang, X.S.: 'Nature-Inspired optimization algorithms' (Elsevier, Massachusetts 02451, USA, 2014, first edn.)
- [17] Karaboga, D.: 'An idea based on honey Bee swarm for numerical optimization', Erciyes University, Engineering Faculty, Computer Engineering Department, 2005
- [18] Yang, X.S.: 'Firefly algorithm, stochastic test functions and design optimisation', *Int. J. Bio-Inspired Comput.*, 2010, **2**, pp. 78–84
- [19] Gropp, W., Lusk, E., Skjellum, A.: 'Using MPI: portable parallel programming with the message-passing interface' (MIT Press, Massachusetts 02142, USA, 1999), 1
- [20] Lalwani, S., Sharma, H., Verma, A., *et al.*: 'Minimization of makespan for parallel machines using PSO to enhance caching of MSA based multi-query processes'. Proc. of 7th Int. Conf. on Soft Computing for Problem Solving (SocProS), Bhubaneswar, India, 2017
- [21] Askitis, N., Zobel, J.: 'B-tries for disk-based string management', *VLDB J.*, 2009, **18**, pp. 157–179
- [22] Prokopec, A., Bronson, N.G., Bagwell, P., *et al.*: 'Concurrent tries with efficient non-blocking snapshots'. Proc. of the 17th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, New Orleans, Louisiana, USA, 2012, vol. 47, pp. 151–160
- [23] Briandais, R.D.L.: 'File searching using variable length keys'. Proc. of the Western Joint Computer Conf., San Francisco, California, 1959, pp. 295–298
- [24] Fredkin, E.: 'Trie memory communications of the ACM', *ACM*, 1960, **3**, pp. 490–499
- [25] Bagwell, P.: 'Ideal hash trees LAMP-EPFL', 2001
- [26] Lalwani, S., Sharma, K.: 'Design and comparison of discrete swarm intelligence algorithms for makespan minimization of parallel machines for complex RNA sequence allocation jobs', *J. Stat. Manag. Syst.*, **21**, (In press)