AIMS Mathematics, 9(5): 11212–11227.
DOI: 10.3934/math.2024550
Received: 29 January 2024
Revised: 10 March 2024
Accepted: 13 March 2024
Published: 21 March 2024

https://www.aimspress.com/journal/Math

Research article

Optimized RNA structure alignment algorithm based on longest arc-preserving common subsequence

Hazem M. Bahig[1,*], Mohamed A.G. Hazber[1] and Tarek G. Kenawy[2]

[1] Department of Information and Computer Science, College of Computer Science and Engineering, University of Ha'il, Hail 81481, KSA
[2] Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt

* **Correspondence:** Email: h.bahig@uoh.edu.sa.

Abstract: Ribonucleic acid (RNA) structure alignment is an important problem in computational biology to identify structural similarity of RNAs. Obtaining an efficient method for this problem is challenging due to the high computational time for the optimal solution and the low accuracy of a heuristic solution. In this paper, an efficient algorithm is proposed based on a mathematical model called longest arc-preserving common subsequence. The proposed algorithm uses a heuristic technique and high-performance computing to optimize the solution of RNA structure alignment, both in terms of the running time and the accuracy of the output. Extensive experimental studies on a multicore system are conducted to show the effectiveness of the proposed algorithm on two types of data. The first is simulated data that consists of 450 comparisons of RNA structures, while the second is real biological data that consists of 357 comparisons of RNA structures. The results show that the proposed algorithm outperforms the best-known heuristic algorithm in terms of execution time, with a percentage improvement of 71% and increasing the length of the output, i.e., accuracy, by approximately 45% in all studied cases. Finally, future approaches are discussed.

Keywords: optimization; RNA alignment; longest common subsequence; parallel algorithm; multicore system
Mathematics Subject Classification: 68W10, 90C27, 92B05, 92C15

1. Introduction

The field of optimization plays an important role in formulating many daily life problems in different fields of science, engineering, medicine, and biology. The main goal of optimization is to find the best possible or optimal solution by taking the maximization or minimization from all possible solutions under certain constraints.

Many techniques have been developed to solve optimization problems such as dynamic programming, greedy technique, integer programming, and metaheuristics. The major challenges for solving many optimization problems are (1) the high computation time that is required to find an optimal solution, and (2) the low accuracy of the approximate solution when manipulating large-size problems. Bioinformatics is a research field that has many problems that can be formulated as optimization tasks. Examples of optimization problems in bioinformatics are molecular docking [1], deoxyribonucleic acid (DNA) motifs [2–4], ribonucleic acid (RNA) structure comparisons [5,6], and RNA structure prediction [7].

We focus on the RNA structure comparison problem. The study of molecular similarity permits the classification of molecules into groups, estimation of their evolutionary history, identification of functional motifs, and thus prediction of their biological function [8].

RNA is a single-stranded polymer that consists of four different nucleotides—adenine (A), guanine (G), cytosine (C), and uracil (U). These nucleotides are connected by phosphodiester bonds. The shape of the RNA structure is determined when the RNA folds back on itself [1]. In this case, a hydrogen bond exists between two interacting nucleotides and forms Watson-Crick (G-C and A-U) and wobble (G-U) base pairs [9].

In general, various computational models have been used to formulate the problem of RNA structure comparison, such as the tree [10–12], arc-annotated sequence (AAS) [13], and alignment-free [12,14]. The research in this paper is interested in the algorithms that solve the RNA structure comparison based on AAS. In this case, the RNA structure comparison problem is defined as follows [13]: Given two RNAs as AASs, the goal is to determine the maximum common subsequence between two AASs under the condition that all arcs connecting the subsequence's nucleotides of RNA are preserved. This goal is named the longest arc-preserving common subsequence (LAPCS).

There are two special cases of LAPCS [15]: c-fragment LAPCS, and c-diagonal LAPCS. In c-fragment LAPCS, is a LAPCS such that the fragment bases in the first sequence are only permitted to match fragment bases at the same location in the second sequence, where each sequence is divided into fragments, each of size c, except the last one, where $c \geq 1$. In c-diagonal LAPCS, is a LAPCS, such that the base a_i is only permitted to match a base in the range $b_{i-c}, \ldots, b_{i+c}$, where $c \geq 0$.

In AAS, many algorithms have been proposed based on different approaches [4,5,13,16–19]: (1) Type of proposed algorithm: exact or approximation. (2) Type or level of RNA structure: Crossing, nested, chain, and plain (see Figure 1). (3) Type of platform used: Sequential or high-performance systems. Table 1 illustrates the different proposed algorithms to solve RNA structure alignment based on the LAPCS model. From Table 1, the following points are observed: (1) The running time for exact algorithms is non-polynomial in the general case of RNA structure level, crossing type. (2) No experimental studies for finding the optimal solution in the general case. (3) The running time for a heuristic solution requires high computational time in the general case. (4) The best-known heuristic algorithm for RNA structure comparison in the general case is the algorithm proposed by Blum and Blesa [5], named the BB algorithm. (5) All proposed algorithms are designed and implemented on a

computer with a single processor and no parallel implementation for any algorithm based on LAPCS.

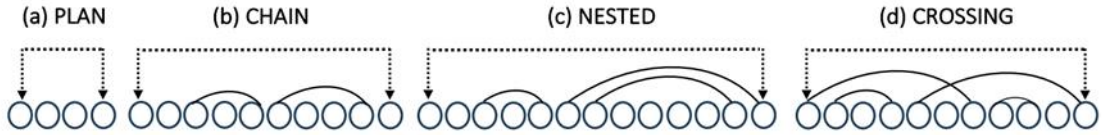

Figure 1. RNA structure levels. (a) Plain: No arc in the sequence. (b) Chain: Any two arcs are not nested and not crossed. (c) Nested: At least two arcs are nested; and no two arcs are crossed. (d) Crossing: At least two arcs are crossed.

Table 1. RNA structure comparison algorithms based on LAPCS*.

Ref	Level of RNA Structure	Exact/Approximation	Time	Technique	Experimental Study
[13]	(Crossing, T)	Exact	NP-hard	MIS	No
[16]	(Nested, Nested)	Exact	NP-hard	MIS	No
[17]	(Nested, Chain)	Exact	$O(l_A l_B^3)$	DP	No
[18]	(Plain, Plain)	Exact	$O(l_A l_B)$	DP	No
[17]	(Crossing, Crossing)	Heuristic	$O(l_A l_B)$	DP and MIS	Yes
[5]	(Crossing,Crossing)	Heuristic	l_A seconds	Generate a set of common subsequences and MIS	Yes
[19]	(Crossing,Crossing)	Heuristic	l_A seconds	Generate a set of common subsequences and Maximum Clique	Yes
[4]	(Crossing, Plain)	Heuristic	$O(l_A l_B)$	DP and Lookup table	Yes

Note: *: T∈{Crossing, Nested, Chain, Plain}, DP: Dynamic programming, MIS: Maximal independent set, NP: non-deterministic polynomial.

The novelty of this research paper focuses on designing a new parallel heuristic algorithm for three goals. The first is reducing the running time of the best-known heuristic algorithm, BB. The second is increasing the accuracy of the output of the best-known heuristic algorithm, BB. The third is to the best of our knowledge, no previous parallel algorithms have been proposed based on LAPCS. To verify these goals, we used the high-performance computing methodology to design a parallel algorithm for solving RNA structure alignment problem. The developed algorithm is based on two levels of parallelism and implemented on a multicore system of 16 threads. The results show that the parallel proposed algorithm outperforms the BB algorithm from running time and accuracy perspectives.

The organization of the paper includes an introduction, four sections, and a conclusion. The mathematical and computer background of the RNA comparison structure problem is given in Section 2. The details of the proposed parallel algorithm are introduced in Section 3. In Section 4, the experimental configurations used in the experiments including simulated and real data are given. The

results of the experiments are discussed and analyzed in Section 5. Finally, the conclusion of using the high-performance system in RNA comparison is given.

2. Mathematical background

In this section, a set of definitions and mathematical formula related to the LAPCS are given as follows.

Assume that the set of nucleotides of RNA is $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{U}\}$ and the set RNA structure type is $T = \{\mathrm{Crossing},\ \mathrm{Nested}, \mathrm{Chain},\ \mathrm{Plain}\}$.

A subsequence $A' = a'_1 a'_2 \dots a'_l$ of a string $A = a_1 a_2 \dots a_{l_A}$ is generated by deleting $l_A - l$ symbols from A. A common subsequence C of two strings *A* and *B* is a subsequence that appears in both strings and is formally represented as:

$$C = \{c_{i,\mathrm{j}} = (i, j),\ \text{s.t. } a_i = b_j,\ \ 1 \le i \le l_A, 1 \le j \le l_B\}. \tag{1}$$

A longest common subsequence (LCS) *C* of two strings *A* and *B* is a common subsequence between *A* and *B* and has a maximal length. Formally,

$$C = \{c_{i,j} = (i, j), \text{s.t.} a_i = b_j,\ 1 \le i \le l_A, 1 \le j \le l_B\} \text{ and } |C| \ge |C'|,$$
$$\forall \text{ common subsequence } C'. \tag{2}$$

The LCS for two substrings $a_1 a_2 \dots a_i$ and $b_1 b_2 \dots b_j$ is computed by the dynamic programming (DP) technique using the following equation.

$$LCS[i,j] = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ LCS[i-1, j-1] + 1 & a_i = b_j \text{ and } i, j \ge 1 \\ \mathrm{Max}\{LCS[i-1, j], LCS[i, j-1]\} & a_i \ne b_j \text{ and } i, j \ge 1 \end{cases} \tag{3}$$

An arc-annotated sequence (AAS) is a pair (A, P_A), where (1) *A* is a string (sequence) of length l_A over Σ, $A = a_1 a_2 \dots a_{l_A}$. (2) P_A is the set of arcs, where each arc represents an unordered pair of any two complementary nucleotides. Formally, the set P_A is defined as follows:

$$P_A = \{(i, j): 1 \le i < j \le l_A, \text{ and } a_i \text{ and } a_j \text{ are complementary}\}. \tag{4}$$

$$\forall\ (i_1, j_1), (i_2, j_2) \in P_A: (i_1 = i_2 \Leftrightarrow j_1 = j_2) \text{ and } i_1 \ne j_2. \tag{5}$$

A common subsequence C of two arc-annotated sequences, (A, P_A) and (B, P_B), is named an arc-preserving common subsequence (APCS) if *C* is a subsequence for A and B and preserves all the arcs that link subsequence nucleotides. Formally,

$$\text{Eq (2) and } \forall\ c_{i_1,j_1},\ c_{i_2,j_2} \in C,\ (i_1, i_2) \in P_A \Leftrightarrow\ (j_1, j_2) \in P_B. \tag{6}$$

The APCS with maximal length is named the longest arc-preserving common subsequence, LAPCS. Formally,

$$C \text{ is } APCS \text{ and } |C| \ge |C'|, \text{ for every } APCS\ C'. \tag{7}$$

The relation between two arcs is as follows [20].

- Two arcs (i_1, j_1) and (i_2, j_2) are crossing iff $i_1 < i_2 < j_1 < j_2$ or $i_2 < i_1 < j_2 < j_1$.
- Two arcs (i_1, j_1) and (i_2, j_2) are nested iff $i_2 < i_1 < j_2 \Leftrightarrow i_2 < j_1 < j_2$, for $i_1 < j_1$ and $i_2 < j_2$.
- Two arcs (i_1, j_1) and (i_2, j_2) are chain iff $i_1 < i_2 \Leftrightarrow j_1 < i_2$, for $i_2 < j_2$.

3. The proposed method

The main purpose of this section is to describe in detail how to use the parallel concept to develop an efficient parallel algorithm based on the best-known algorithm for RNA comparison, the BB algorithm. The proposed algorithm named PBB.

The BB algorithm consists of many steps, where the main two steps are as follows. The first step is generating a number, n_{sol}, of common subsequences, S, from the two strings A and B. The second step uses the Cplex tool [21] to find the APCS for the output of the first step based on the concept of the maximum independent set (MIS). The MIS can be determined by construct a graph G=(V,E), where (1) V is the set of matched pairs that are built from the common subsequences, $V = \{v_{i,j} = (i,j): (i,j) \in S\}$, and (2) $E = \{(v_{i,j}, v_{i',j'}): (i,j)$ and $(i',j')$are breaking arc − preserving conditions, (i,j) and $(i',j') \in S\}$. Then, using integer linear program model, the problem can be formulated as a selection of a maximal number of non-conflicting binary variables [6]. The two main steps are repeated many times which is based on the length of the input string A, l_A seconds. In each iteration, the algorithm updates the set of solutions and determines the best value of the solution by comparing the best old solution with the current values of the solution. Figure 2 illustrates the execution of BB algorithm on two RNAs of length 15 each.

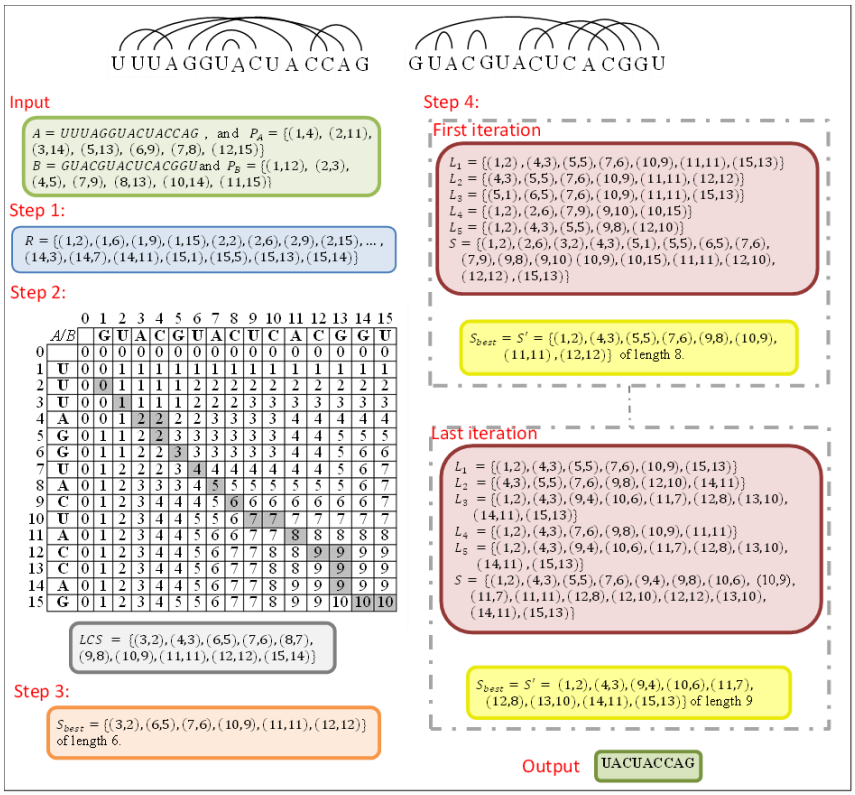

Figure 2. An example for executing BB algorithm.

The BB algorithm uses four parameters [5]: (1) n_{sol} is the number of APCS constructions per iteration. (2) t_{max} is the maximum time allowed to find MIS using Cplex tool when the number of common subsequences is greater than or equal to n_{sol}. (3) d_{rate} and l_{size} are two parameters used in the process of generation of a random common subsequence.

The proposed algorithm is based on using two levels of parallelism as follows. Assume that the number of threads is k, and $k=k_1*k_2$. The reason for writing k as a product of integer numbers is to make the parallelism in two levels. The first level is based on parallelizing the generation of common subsequences from A and B. This means that when generating n_{sol} common subsequences only as in the BB algorithm, the proposed algorithm generates k_1*n_{sol} common subsequences such that each thread generates n_{sol} common subsequences. The second level is how to use parallelization to find APCS using the MIS method.

Additionally, in the BB algorithm, the two main steps are repeated many times based on the length of string A. Therefore, the proposed parallel algorithm reduces this repetition time to approximately l_A/k_1 seconds. Also, the proposed PBB algorithm uses some parallel subroutines such as parallel maximal independent set, PMIS, algorithm [21], parallel binary tree technique [22,23], and parallel LCS, PLCS, algorithm [24,25].

The complete steps for the proposed PBB algorithm, PBB, are as follows.

Algorithm: PBB

Input: Two AASs (A, P_A) and (B, P_B), and three integers k, k_1 and k_2 such that $k=k_1*k_2$ and $k_1, k_2 \geq 2$.

Step 1: Set the values of parameters in a constant time, n_{sol}, t_{max}, d_{rate} and l_{size}, based on the length of A as suggested in [5], where $|A| \geq |B|$, see [5, Table 4].

Step 2: Generate the set of matched pairs between A and B using k threads. This step can be performed by dividing B into k substrings, B_i, of approximately equal size, $|B|/k$. Then each thread finds the set of matched pairs, R_i, between A and B_i. Finally, R is the union set of all matched pairs, $R = \bigcup_{i=1}^{k} R_i$, generated by k threads that can be computed by the parallel binary tree (PBT) paradigm.

Step 3: Find the LCS of A and B by applying the parallel LCS (PLCS) algorithm using k threads. The list L represents the set of match pairs of LCS of A and B.

Step 4: Apply the parallel MIS, PMIS, algorithm on list L using the Cplex tool and set the results to the current solution S_{best}.

Step 5: Each thread t_i, $1 \leq i \leq k_1$, performs the following:

Step 5.1: Initialization: $timer = 0$, $S_{i\_best} = S_{best}$, $S_i = S_{best}$.

Step 5.2: Repeat the following steps until $timer$ is greater than or equal to $|A|/k_1$ seconds:

Step 5.2.1.: Repeat the following n_{sol} iterations:

Step 5.2.1.1: Generate a random common subsequence, say C_i, from the list R as in [5].

Step 5.2.1.2: Apply the PMIS algorithm on C_i using k_2 threads and obtain the list L_{C_i}.

Step 5.2.1.3: Update $S_i = S_i \cup L_{C_i}$.

Step 5.2.2: Apply the PMIS algorithm on S_i using k_2 threads within time limit t_{max} and obtain the list L_{S_i}.

Step 5.2.3: Update S_{i_best} with L_{S_i} if $|L_{S_i}| > |S_{i\_best}|$.

Step 6: Calculate $S_{best} = \bigcup_{i=1}^{k_1} S_{i\_best}$ using k threads and the PBT paradigm.

Step 7: Apply the PMIS algorithm on S_{best} using k threads within time limit t_{max} and obtain the list $LAPCS$.

Output: LAPCS

Note that in the case of using the PBT paradigm using k threads, see Steps 2 and 6. If the value of k_1 is small, then the proposed algorithm will perform these steps sequentially. Also, Figure 3 shows the flow chart of the proposed parallel algorithm.

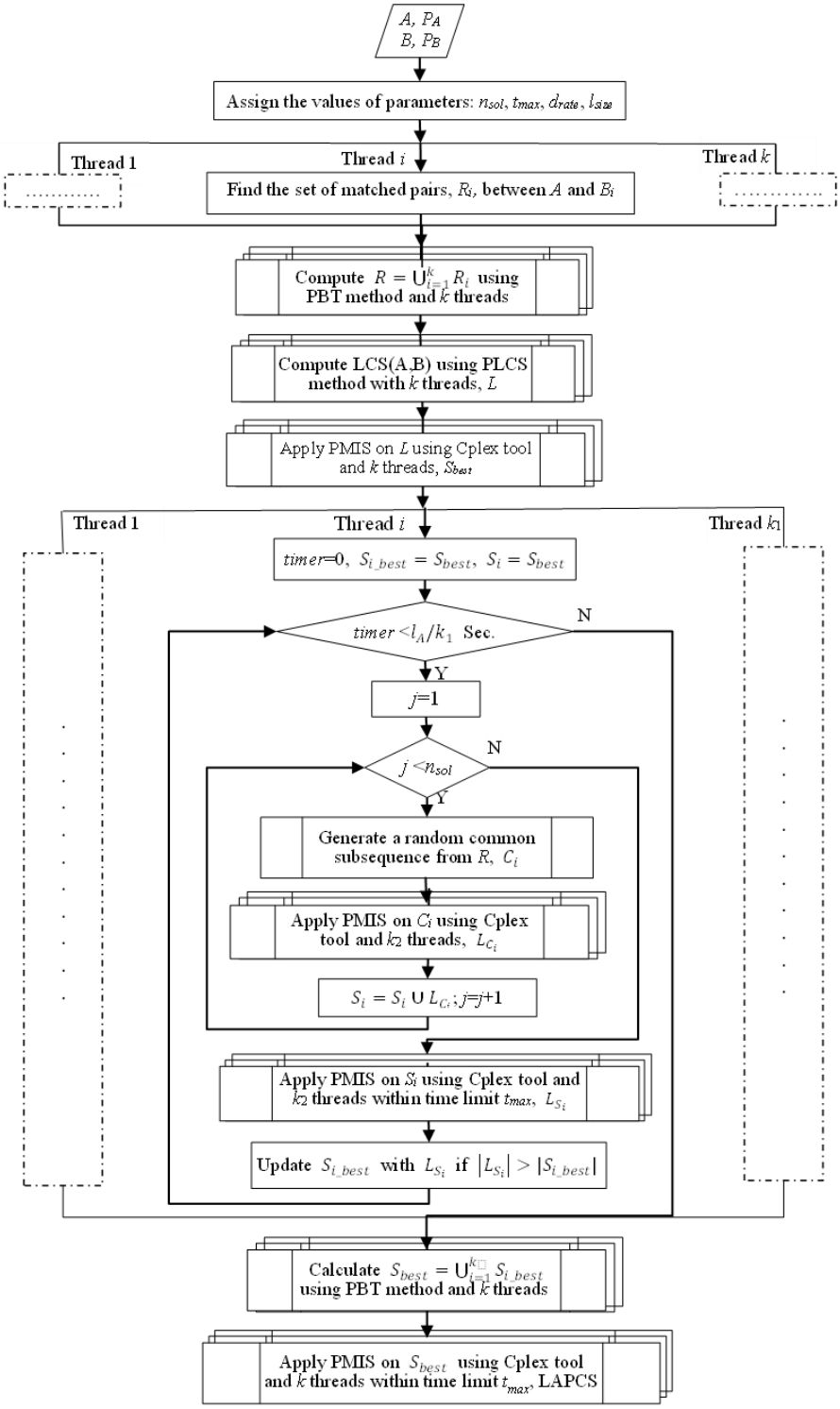

Figure 3. Flow chart for PBB algorithm.

4. Experimental configuration

In this section, we describe the experimental setup used to implement the proposed parallel algorithm on a multicore system. The experimental setup includes the platform used in the implementation, data generation used in the comparison, and the number of threads used in each level of parallelism.

For the platform used in the implementation, the experimental studies are based on a multicore system that can execute 16 threads in parallel using a processor with a speed of 2.4 GHz and a memory capacity of 24 GB. The system works under the Linux operating system. All algorithms were programmed using the Java programming language. The Java thread features were used to implement the parallel region. Additionally, all compared algorithms used IBM ILOG CPLEX v12.8 [21] as a tool to find a good heuristic solution for the MIS problem in sequential and parallel cases

For data generation, experimental comparisons between the algorithms are performed using two different datasets. The first dataset is generated as artificial data that is used for two purposes: (1) Evaluating the proposed algorithm compared to the best-known algorithm for RNA structure comparison, the BB algorithm; and (2) determining the best approach to assign 16 threads for the parallel case as two levels of parallelism. The second dataset is a real biological data for RNA structures that is used to ensure that the proposed parallel algorithm is also efficient for real data in terms of time and accuracy.

For artificial data, two parameters affect the generation of the RNA sequence: The sequence length, n, and the number of arcs, m. For fixed values of n and m, the system will generate an RNA sequence of length n containing m arcs such that the type of RNA structure is crossing. The set of values of n is {100, 200, 300, 400, 500}, while the set of values of m depends on n and is equal to $n/2$, $n/5$, and $n/10$. The sequence will be generated randomly and the appearance of each letter in the sequence is 1/4.

For fixed values of n and m, the running time of the compared algorithms is measured by taking the average value for 30 instances. Therefore, for a fixed value of n, the running time of the algorithm is the average of 90 instances because there are three values of m. Therefore, there are $5 \times 90 = 450$ comparisons of RNA structures. Additionally, the length of APCS is computed for each m to study the effect of the number of arcs on the performance of each algorithm.

To implement the parallel proposed algorithm, PBB, on a multicore system consisting of 16 threads, the number of threads, $k = k_1 * k_2$, can be represented in different ways as follows: (1) $16 = 2 \times 8$, (2) $16 = 4 \times 4$, and (3) $16 = 8 \times 2$. In general, k_1 and k_2 are used for the first and second levels of parallelism, respectively. Therefore, the parallel algorithm can be represented as three parallel versions, PBB1, PBB2, and PBB3 for $16 = 2 \times 8$, $16 = 4 \times 4$, and $16 = 8 \times 2$, respectively. Thus, the PBB1 algorithm uses 2 threads in the first level of parallelism and 8 threads in the second level of parallelism.

For real data of RNA structures, experimental studies focus on different real data as, shown in Table 2. The Ribonuclease P RNA database contains a large number of RNAs; therefore, we selected 12 RNAs only. For Group I introns (Group A, B, and E), we selected all RNAs in the dataset. The details of each selected RNA, such as name, length, and number of arcs, are provided in Appendix A.

For each RNA group that consists of β RNAs, the total number of possible comparisons between each pair is given by:

$$\binom{\beta}{2} = \beta(\beta - 1)/2. \tag{8}$$

Table 2. Real dataset used in the experiments.

RNA Database	Number of Selected RNAs	Total Number of Comparisons	Location	Ref
Ribonuclease P RNA	12 (out of 470)	66	http://www.rnasoft.ca/strand/	[26]
Group i introns (A)	12	66	https://crw2-comparative-rna-web.org/group-i-introns/	[27]
Group i introns (B)	21	210		
Group i introns (E)	6	15		

Therefore, the methodology used in the experimental study applies the two algorithms, sequential and parallel, to all pairs of RNAs. For example, the RNA database named "Group I introns (Group E)" contains 6 RNAs: *M.anisopliae.4*, *C.hypophloia*, *E.nigra*, *H.rubra*, *M.anisopliae.2*, and *C.parasitica*. Therefore, the total number of comparisons is 15 as follows. (1) Five comparisons between *M.anisopliae.4* and every RNA in {*C.hypophloia*, *E.nigra*, *H.rubra*, *M.anisopliae.2*, *C.parasitica*}. (2) Four comparisons between *C.hypophloia* and every RNA in {*E.nigra*, *H.rubra*, *M.anisopliae.2*, *C.parasitica*}. (3) Three comparisons between *E.nigra* and every RNA in {*H.rubra*, *M.anisopliae.2*, *C.parasitica*}. (4) Two comparisons between *H.rubra* and every RNA in {*M.anisopliae.2*, *C.parasitica*}. (5) One comparison between *M.anisopliae.2* and *C.parasitica*.

In general, the experimental studies on real datasets include 357 comparisons to find the LAPCS. For each pair of RNAs in real dataset, two measurements are calculated. The first measure is the length of the APCS, while the second measure is the running time of executing the algorithm.

5. Results and discussion

In this section, the results and analysis of the experimental studies on artificial data and real data are discussed in the next two subsections.

5.1. Results of comparison on artificial data

The results of comparing four algorithms, one sequential and three parallel, are shown in Figure 4 and Table 3. The analysis of data results in the figure and table indicates the following.

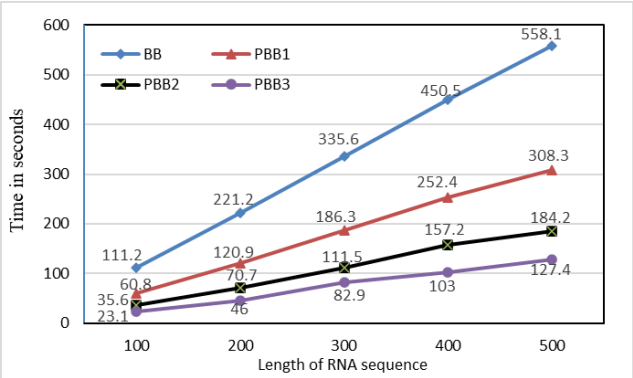

Figure 4. Running time for compared algorithms on simulated data.

Table 3. Length of APCS, average value, for the compared algorithms using different *n* and *m*.

n	m	BB	PBB1	PBB2	PBB3
100	$n/10$	59.3	59.3	59.3	59.3
	$n/5$	56.3	**56.6**	56.5	56.5
	$n/2$	46.2	46.5	46.6	**46.8**
200	$n/10$	121.4	121.3	121.5	**121.6**
	$n/5$	112.4	112.3	113.0	**113.1**
	$n/2$	89.4	89.4	**89.9**	89.1
300	$n/10$	180.5	181.2	**181.5**	181.1
	$n/5$	168.9	**169.4**	169.0	**169.4**
	$n/2$	134.7	135.0	135.3	**135.4**
400	$n/10$	236.5	**238.2**	237.5	237.6
	$n/5$	218.9	**220.6**	219.3	218.6
	$n/2$	166.9	**170.9**	169.6	168.2
500	$n/10$	291.4	294.7	**296.2**	293.8
	$n/5$	264.9	271.1	**273.5**	270.1
	$n/2$	199.3	**207.5**	203.3	203.5

From the running time perspective, the results, in Figure 4, illustrate the following observations. First, the running time of all parallel algorithms is less than the running time of the sequential algorithm. For example, the running time for the BB algorithm is 335.6 second when $n = 300$, whereas the running times for the three parallel algorithms, PBB1, PBB2, and PBB3, are 186.3, 111.5, and 82.0, seconds, respectively. Second, the PBB3 algorithm is faster than the PBB2 algorithm, and the PBB2 algorithm is faster than the PBB1 algorithm. For example, the running time for PBB3 algorithm is 127.4 seconds when $n = 500$, while the running times for the two other algorithms, PBB2, and PBB1, are 184.2 and 308.3 seconds, respectively. Third, the percentage of improvements, on average, for the three parallel algorithms, PBB1, PBB2, and PBB3, are 44.78%, 66.97%, and 77.60%, respectively, where the percentage of improvement is measured by $1\text{-}T_{par}/T_{seq}$. Fourth, the average values of speed up for the parallel algorithms, PBB1, PBB2, and PBB3, are 1.8, 3, and 4.5, respectively, where the speed up is equal to T_{seq}/T_{par}.

From the length of output viewpoint, the results in Table 3 illustrate the following observations. First, the length of the APCS generated by parallel algorithms is approximately equal to the length of the APCS generated by the sequential algorithm when $n \leq 200$. For example, the average length of APCS, for 30 instances, generated by the four algorithms, BB, PBB1, PBB2, and PBB3, are 46.20, 46.53, 46.60, and 46.77, respectively, when $n = 100$, $m = n/2$. Second, the length of the APCS generated by parallel algorithms is greater than the length of the APCS generated by sequential algorithm when $n > 200$. For example, the average length of APCS, for 30 instances, generated by the four algorithms, BB, PBB1, PBB2, and PBB3, are 166.9, 170.9, 169.6, and 168.2, respectively, when $n = 400$, $m = n/2$. Third, the difference between the length of the APCS generated by parallel algorithms and the BB algorithm increases with increasing values of *n*. For example, the average difference between the length of the APCS generated by BB algorithm and PBB1 algorithm is 2.5 when $n = 400$, whereas the difference equal to 5.9 when $n = 500$. Fourth, the length of the APCS generated by the PBB1 and PBB2 algorithms is almost greater than that generated by the PBB3 algorithm.

A non-parametric statistical test known as the Wilcoxon signed-rank test [27] was employed to ascertain whether there exist statistically significant variations in the length of the output for the four algorithms, BB, PBB1, PBB2, and PBB3. The significant level used in the test is equal to 0.05. The results of implementing the test on each pair of algorithms, six pairs of algorithms, show the following observations (see additional file "math-09-05-550-supplementary"). (1) There was a significant difference between all parallel algorithms, PBB1, PBB2, and PBB3, and the sequential algorithm, BB; except in one case, there is no significant difference between BB and PBB1 when $n = 200$. (2) In the case of the two algorithms, PBB1 and PBB2, the PBB2 algorithm is better than the PBB1 algorithm when $n = 200$ and 300, while the PBB1 algorithm is better than the PBB2 algorithm when $n = 400$. Otherwise, there is no significant difference between the two algorithms. (3) In the case of the two algorithms, PBB1 and PBB3, the PBB3 algorithm is better than the PBB1 algorithm when $n = 200$, while the PBB1 algorithm is better than the PBB3 algorithm when $n = 400$ and 500. Otherwise, there is no significant difference between the two algorithms. (4) In the case of the two algorithms, PBB2 and PBB3, the PBB3 algorithm is better than the PBB2 algorithm when $n = 200$, while the PBB2 algorithm is better than the PBB3 algorithm when $n = 400$ and 500. Otherwise, there is no significant difference between the two algorithms.

From the memory required by each algorithm viewpoint, Table 4 illustrates the values of the memories in GB. The results show the following observations: First, the memory required by the BB algorithm is less than all parallel algorithms, PBB1, PBB2, and PBB3. Second, the memory required by the PBB1 algorithm is less than that required by the PBB2 algorithm, and the memory required by the PBB2 algorithm is less than that required by the PBB3 algorithm. The memory required for the PBB3 algorithm is high compared to other parallel algorithms due to the manipulation of 8 APCS simultaneously using the Cplex tool, while the two other algorithms manipulate 4 and 2 APCS.

Table 4. Comparison between different algorithms based on memory requirements in GB.

n	BB	PBB1	PBB2	PBB3
100	0.14	0.16	0.23	0.31
200	0.14	0.15	0.24	0.36
300	0.15	0.19	0.34	0.55
400	0.17	0.18	0.53	0.72
500	0.26	0.28	0.53	0.73

As a result, from the analysis of previous data, the parallel algorithms PBB1 and PBB2 have good performance from the length of output measurement compared to the other algorithm. Additionally, the PBB2 algorithm has better performance than the PBB1 algorithm from a running time perspective, which is more important than memory because the amount of storage is not high for all parallel algorithms. Therefore, the parallel algorithm PBB2 was selected to evaluate the parallelization of the BB algorithm for the real dataset as in the next subsection.

5.2. Results of comparison on real data

In this subsection, a comparison between the sequential algorithm and the selected parallel algorithm, PBB2, is performed to verify that the parallelism enhances the sequential algorithm from the points of view of the length of the output and running time. Table 5 shows the results of two

measurements, time and length of LAPCS, for two algorithms, BB and PBB2, on four datasets of real RNAs.

Table 5. Comparison between the BB and PBB2 algorithms for a real dataset.

RNA Database	Running Time in Seconds			Length of LAPCS				
	BB	PBB2	% of Improvement	Difference (PBB2-BB)	Diff = 0	Diff > 0	CV	
							BB	PBB2
Ribonuclease P RNA	398.1	107.5	73%	[0,7]	50%	50%	0.179	0.175
Group i introns (Group A)	1003.2	259.7	74%	[0,7]	45.4%	54.6%	0.362	0.362
Group i introns (Group B)	1263	343.7	71.4%	[0,10]	68.1%	31.9%	0.633	0.632
Group i introns (Group E)	551.1	183.7	66.6%	[0,5]	53.3%	46.7%	0.163	0.161

For the running time measurement, Table 5 shows the running time of both algorithms and the percentage of improvement in the running for the proposed parallel algorithm PBB2 compared to BB algorithm. For example, the two algorithms, BB and PBB2, were run on the Ribonuclease P RNA dataset and obtained the following results. (1) For 66 cases, the average running times of the BB and PBB2 algorithms are 398.1 and 107.5 seconds, respectively. (2) The PBB2 algorithm outperforms the BB algorithm with a percentage of improvement of 73%. Additionally, the running times for BB algorithm on the two dataset, Group A and B, are higher than the other dataset because the two datasets contain RNA with length greater than 1000 (see Appendix A).

For the length of APCS measurement, Table 5 displays the range of variation between the PBB2 and BB algorithms' outputs for the length of APCS measurement, as well as the percentage of cases where the PBB2 algorithm's generated LAPCS is longer (or equal to) the BB algorithm's generated LAPCS. For example, the two algorithms, BB and PBB2, were run on the Group i introns (Group A) dataset and obtained the following results: (1) The length of LAPCS generated by the PBB2 algorithm is greater than or equal to the output of the BB algorithm, with a difference from 0 to 7. (2) In 45.4% of the comparison cases, both algorithms generate the LAPCS with the same lengths. On the other hand, the PBB2 algorithm generates LAPCS with a length greater than that generated from the BB algorithm, with a percentage of 54.6%. Additionally, the difference between PBB2 and BB algorithms is sometimes large, such as in the Group i introns (Group B) dataset, where the maximum difference is 10. (3) The results of measuring the coefficient of variation (CV) of both algorithms for the length of LAPCS is almost equal. (4) The PBB2 algorithm has a significant difference compared to BB algorithm when we use Wilcoxon signed-rank test for all cases, except one case when the dataset is Group i introns (Group E).

On average, in all cases, the PBB2 algorithm outperforms the BB algorithm in terms of running time, with an improvement of approximately 71%. Additionally, the PBB2 algorithm generates LAPCS with a length greater than that generated by the BB algorithm, with at least 1 in 45.8% of the cases.

6. Conclusions

Identifying the similarity structure between two RNA structures is challenging in bioinformatics due to the high computational time required to find an optimal solution. In this paper, the RNA structure is represented as the longest arc-preserving common subsequence model. Then high-

performance computing paradigms and metaheuristic techniques are utilized to develop an efficient parallel algorithm on a multicore system. The developed parallel algorithm outperforms the best-known sequential algorithm in terms of the running time and length of output.

Additionally, the developed algorithm was tested on artificial and real data to measure the percentage of improvement in the running 71% on average and increasing the length of output by approximately 45% of all cases. On the other side, the proposed algorithm required a little bit more space.

There are many open research questions based on the contributions of this paper: How can we replace the MIS method with another heuristic algorithm to increase the efficiency of the proposed algorithm? (2) How can we use the graphic processing units (GPUs) to implement the parallel proposed algorithm?

Use of AI tools declaration

The authors declare that they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The authors are grateful to the referees for their valuable comments that helped to improve the paper. The authors would like to acknowledge the support provided by Deputy for Research & Innovation, Ministry of Education through Initiative of Institutional Funding at University of Ha'il–Saudi Arabia through project number IFP-22 187.

Funding

This work was supported by Deputy for Research & Innovation, Ministry of Education through Initiative of Institutional Funding at University of Ha'il–Saudi Arabia through project number IFP-22 187.

Conflict of interest

The authors declare that they have no conflicts of interest.

References

1. D. Jereva, P. Alov, I. Tsakovska, M. Angelova, V. Atanassova, P. Vassilev, et al., Application of intercriteria analysis to assess the performance of scoring functions in molecular docking software packages, *Mathematics*, **10** (2022), 2549. https://doi.org/10.3390/math10152549
2. M. M. Abbas, M. Abouelhoda, H. M. Bahig, A hybrid method for the exact planted (l, d) motif finding problem and its parallelization, *BMC Bioinformatics*, **13** (2012), S10. https://doi.org/10.1186/1471-2105-13-S17-S10
3. M. M. Abbass, H. M. Bahig, An efficient algorithm to identify DNA motifs, *Math. Comput. Sci.*, **7** (2013), 387–399. https://doi.org/10.1007/s11786-013-0165-6

4. T. G. Kenawy, M. H. Abdel-Rahman, H. M. Bahig, A fast longest crossing-plain preserving common subsequence algorithm, *Int. J. Inf. Technol.*, **14** (2022), 3019–3029. https://doi.org/10.1007/s41870-022-01038-0
5. M. M. Abbas, H. M. Bahig, M. Abouelhoda, M. M. Mohie-Eldin, Parallelizing exact motif finding algorithms on multi-core, *J. Supercomput.*, **69** (2014), 814–826. https://doi.org/10.1007/s11227-014-1180-3
6. C. Blum, M. J. Blesa, Hybrid techniques based on solving reduced problem instances for a longest common subsequence problem, *Appl. Soft Comput.*, **62** (2018), 15–28. https://doi.org/10.1016/j.asoc.2017.10.005
7. M. S. Islam, M. R. Islam, A hybrid framework based on genetic algorithm and simulated annealing for RNA structure prediction with pseudoknots, *J. King Saud Univ. Comput. Inform. Sci.*, **34** (2022), 912–922. https://doi.org/10.1016/j.jksuci.2020.03.005
8. T. J. X. Li, C. M. Reidys, On the loop homology of a certain complex of RNA structures, *Mathematics*, **9** (2021), 1749. https://doi.org/10.3390/math9151749
9. J. Fallmann, S. Will, J. Engelhardt, B. Grüning, R. Backofen, P. F. Stadler, Recent advances in RNA folding, *J. Biotechnol.*, **261** (2017), 97–104. https://doi.org/10.1016/j.jbiotec.2017.07.007
10. K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM J. Comput.*, **18** (1989), 1245–1262. https://doi.org/10.1137/0218082
11. M. Quadrini, L. Tesei, E. Merelli, An algebraic language for RNA pseudoknots comparison, *BMC Bioinformatics*, **20** (2019), 16. https://doi.org/10.1186/s12859-019-2689-5.
12. F. Wang, T. Akutsu, T. Mori, Comparison of pseudoknotted RNA secondary structures by topological centroid identification and tree edit distance. *J. Comput. Biol.*, **27** (2020),1443–1451. https://doi.org/10.1089/cmb.2019.0512
13. P. A. Evans, *Algorithms and complexity for annotated sequence analysis*, Ph. D Thesis, Canada: University of Victoria, 1999.
14. L. Yang, Y. Liu, X. Hu, P. Wang, X. Li, J. Wu, Graph-based analysis of RNA secondary structure similarity comparison, *Complexity*, **2021** (2021), 8841822. https://doi.org/10.1155/2021/8841822
15. J. Guo, *Exact algorithms for the longest common subsequence problem for arc annotated sequences*, Master's Thesis, Universitat Tubingen, 2002
16. G. Lin, Z. Z. Chen, T. Jiang, J. Wen, The longest common subsequence problem for sequences with nested arc annotations, *J. Comput. Syst. Sci.*, **65** (2002), 465–480. https://doi.org/10.1016/S0022-0000(02)00004-1
17. T. Jiang, G. Lin, B. Ma, K. Zhang, The longest common subsequence problem for arc-annotated sequences, *J. Discrete Algorithms*, **2** (2004), 257–270. https://doi.org/10.1016/S1570-8667(03)00080-7
18. T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, *J. Mol. Biol.*, **147** (1981), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5
19. C. Blum, M. Djukanovic, A. Santini, H. Jiang, C. M. Li, F. Manyà, et al., Solving longest common subsequence problems via a transformation to the maximum clique problem, *Comput. Oper. Res.*, **125** (2021), 105089. https://doi.org/10.1016/j.cor.2020.105089
20. J. Gramm, J. Guo, R. Niedermeier, Pattern matching for arc-annotated sequences, In: *Foundations of software technology and theoretical computer science*, Berlin, Heidelberg: Springer, 2002. https://doi.org/10.1007/3-540-36206-1_17

21. IBM, CPLEX Optimization Studio V12.8.0, Available from: https://www.ibm.com/support/pages/cplex-optimization-studio-v128.
22. G. Blelloch. Prefix sums and their applications. In: *Synthesis of parallel algorithms*, 1990. Available from: http://shelf2.library.cmu.edu/Tech/23445461.
23. H. Bahig, K. A. Fathy, An improved parallel prefix sums algorithm, *Parallel Processing Lett.*, **32** (2022), 2250008. https://doi.org/10.1142/S0129626422500086
24. R. Shikder, P. Thulasiraman, P. Irani, P. Hu, An OpenMP-based tool for finding longest common subsequence in bioinformatics, *BMC Res. Notes*, **12** (2019), 220. https://doi.org/10.1186/s13104-019-4256-6
25. M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon, J. F. Reid, A fast and practical bit-vector algorithm for the longest common subsequence problem, *Inform. Processing Lett.*, **80** (2001), 279–285. https://doi.org/10.1016/S0020-0190(01)00182-X
26. M. Andronescu, V. Bereg, H. H. Hoos, A. Condon, RNA STRAND: The RNA secondary structure and statistical analysis database, *BMC Bioinformatics*, **9** (2008), 340. https://doi.org/10.1186/1471-2105-9-340
27. CRW2: Comparative RNA Web-2. Available from: https://crw2-comparative-rna-web.org/.
28. R. F. Woolson, Wilcoxon signed-rank test, *Wiley encyclopedia of clinical trials*, 2008. https://doi.org/10.1002/9780471462422.eoct979

Appendix A

RNA Database	Name of RNA	length	#arcs
Ribonuclease P RNA	Allochromatium_vinosum	369	119
	Bacteroides_thetaiotaomicron	361	121
	Porphyromonas_gingivalis	398	131
	Haemophilus_influenza	377	124
	Halococcus_morrhuae	475	154
	Haloferax_volcanii	433	142
	Mycoplasma_genitalium	384	119
	Mycoplasma_pneumoniae	369	112
	Serratia_marcescens	378	225
	Shewanella_putrefaciens	354	115
	Streptomyces_bikiniensis	397	135
	Streptomyces_lividans	405	138
Group I intron (Group A)	m.S.cerevisiae	1210	101
	b.Bacteriophage.RB3	1115	69
	b.coliphage.T4.A2.NRDD	1058	92
	c.C.eugametos	1053	82
	b.coliphage.T4.A2.TD	1036	77
	C.reinhardii	953	97
	S.luteus 2449	611	107
	coliphage	607	70
	Bacteriophage.SP0	908	81

Continued on next page

RNA Database	Name of RNA	length	#arcs
Group I intron (Group A)	m.S.cerevisiae	803	100
	S.luteus 2504	373	104
	Bacteriophage.beta	403	87
Group I intron (Group B)	m.P.anserina	2630	66
	c.C.moewusii	1844	110
	m.T.papilionaceus	1780	74
	m.M.senile	1710	86
	m.N.crassa	1180	59
	m.E.nidulans	1105	57
	m.S.cerevisiae	1075	82
	C.albicans	459	127
	S.negevensis	736	93
	S.luteus.1974	623	89
	N.aquatica	416	116
	S.luteus.1923	555	148
	C.pallidostigmatica	924	64
	C.saccharophila	427	101
	A.castellanii	846	75
	C.dubliniensis	621	114
	S.pombe	343	68
	M.senile	880	94
	S.luteus.2500	617	191
	M.grisea	285	66
	W.mrakii	409	72
Group I intron (Group E)	M.anisopliae.4	430	91
	C.hypophloia	526	138
	E.nigra	523	145
	H.rubra	557	136
	M.anisopliae.2	429	86
	C.parasitica	640	124