J Supercomput (2014) 70:880–905
DOI 10.1007/s11227-014-1266-y

Parallelizing and optimizing a hybrid differential evolution with Pareto tournaments for discovering motifs in DNA sequences

David L. González-Álvarez · Miguel A. Vega-Rodríguez · Álvaro Rubio-Largo

Published online: 5 August 2014

Abstract Transcriptional regulation is the main regulation of gene expression, the process by which all prokaryotic organisms and eukaryotic cells transform the information encoded by the nucleic acids (DNA) into the proteins required for their operation and development. A crucial component in genetic regulation is the bindings between transcription factors and DNA sequences that regulate the expression of genes. These specific locations are short and share a common sequence of nucleotides. The discovery of these small DNA strings, also known as motifs, is labor intensive and therefore the use of high-performance computing can be a good way to address it. In this work, we present a parallel multiobjective evolutionary algorithm, a novel hybrid technique based on differential evolution with Pareto tournaments (H-DEPT). To study whether this algorithm is suitable to be parallelized, H-DEPT has been used to solve instances of different sizes on several multicore systems (2, 4, 8, 16, and 32 cores). As we will see, the results show that H-DEPT achieves good speedups and efficiencies. We also compare the predictions made by H-DEPT with those predicted by other biological tools demonstrating that it is also capable of performing quality predictions.

Keywords Parallelism · Hybrid algorithm · Differential evolution · Multiobjective optimization · Motif discovery

D. L. González-Álvarez (✉) · M. A. Vega-Rodríguez · Á. Rubio-Largo
ARCO Research Group, Department Technologies of Computers and Communications, University of Extremadura, Escuela Politécnica, Campus Universitario s/n, 10003 Cáceres, Spain
e-mail: dlga@unex.es

M. A. Vega-Rodríguez
e-mail: mavega@unex.es

Á. Rubio-Largo
e-mail: arl@unex.es

1 Introduction

The emerging need for dealing with complex optimization problems in almost all scientific fields has led to significant technological advances. New and improved multicore architectures are designed every day to try to meet these requirements. These powerful architectures allow researchers to design new parallel strategies or methodologies for addressing complex problems where the solution search space grows exponentially with the size of the input information. Parallel computing allows the exploitation of these powerful systems using multiple processing elements simultaneously. This is accomplished by splitting the problem into different independent parts that are processed in parallel. The development of parallel programs is a difficult task because, among other things, we have to carefully study aspects such as data dependencies. One of the most used programming paradigms for implementing parallel programs is OpenMP [5], which is an application programming interface (API) able to exploit the characteristics of shared-memory architectures by means of a set of compiler directives, library routines, and environment variables. By applying this programming paradigm we parallelize a new algorithm to try to reduce its required runtimes and, so, be able to tackle larger instances in a reasonable time.

In bioinformatics, an important complex optimization problem that can exploit the properties of parallel computing is the motif discovery problem (MDP). MDP aims to identify the binding of transcription factor to short nucleotide sequences. These strings, also known motifs, usually share the same nucleotides and can produce changes in the transcriptional activity. There are many tools specialized in the identification of these DNA patterns; some examples are aligns nucleic acid conserved elements (AlignACE) [38], BioOptimizer [20], BioProspector [28], Consensus [19], Gibbs Motif Sampler [27] [33], MDscan [29], GADEM [25], and MEME [1]. In recent years numerous works that present new techniques based on evolutionary algorithms have been also proposed for finding motifs in DNA sequences; some of them are finding motifs by genetic algorithm (FMGA) [26], a genetic algorithm based on the SAGA [34] operators; structured genetic algorithm (St-GA) [44], and motif discovery using a genetic algorithm (MDGA) [6]. However, genetic algorithm-based motif elicitation (GAME) [48] is one of the techniques that has been most successful. GAME is an optimization algorithm that conducts a more exhaustive search of the space of possible motifs. If we analyze the works presented in the literature that address this optimization problem, the MDP, we observe that most of the proposals are based on genetic algorithms. Additionally, these algorithms have a number of limitations such as following a single-objective formulation (as we will see in the following sections, this methodology does not adequately reflect the biological properties of MDP), or the need of defining a given motif length beforehand. Moving away from the concept of single-objective optimization, we just found a multicriterion alternative in [12] and [13], where the authors optimize the similarity and complexity of the discovered DNA patterns by using weights; and another where a first multiobjective approximation is proposed with the MOGAMOD algorithm [21]. Continuing the research presented in the latter work, we propose a realistic multiobjective formulation that allows us to fill the gaps made by the previously described tools. To demonstrate this, we have incorporated this formulation on a hybrid version of the well-known differential evolution (DE) algo-

rithm [36], hybrid differential evolution with Pareto tournaments (H-DEPT), which also incorporates the Pareto tournaments function based on the Pareto dominance and binary tournament concepts. As we will see, this function aims to facilitate the choice of the best multiobjective solution. We also analyze the parallelization capability presented by H-DEPT by conducting experiments on different multicore systems (2, 4, 8, 16, and 32 cores), considering different OpenMP parameters such as the established thread scheduling policy. As we will see, the proposed hybrid multiobjective evolutionary algorithm is able to predict biological quality solutions, presenting good parallel results. These conclusions demonstrate that this algorithm represents a good tool for discovering motifs in complex instances.

The rest of this paper is organized as follows. Section 2 describes the problem formulation in a formal way, as well as includes an illustrative example of the problem to better understand the main purpose of each defined objective function. In Sect. 3 we present a review of the work related to the application of parallelism to solve the MDP. A description about H-DEPT is presented in Sect. 4. In this section we detail all aspects related to the multiobjective adaptation of the algorithm and we describe the operation of the incorporated local search function. Section 5 is devoted to the analysis of the carried out experiments. A comprehensive analysis of the parallel results and a comparison between the results obtained by our proposed algorithm and those achieved by other well-known biological tools are also included. Finally, Sect. 6 summarizes the conclusions of the paper and discusses possible lines of future work.

2 Motif discovery problem

Identifying new transcription factor binding sites (TFBS) is important and essential for understanding the genetic regulation process together with the mechanisms that control life on our planet. The MDP formulates the discovery of motifs as a complex optimization problem which aims to find small TFBSs in the midst of a huge amount of biological information. To do this, we search over-represented substrings in a sequence of strings $S = \{S_1, S_2, \ldots, S_D\}$ defined on the alphabet $B = \{A, C, G, T\}$. Genetic regulation is essential for all life processes such as cell differentiation, metabolism, cell cycles... Transcriptional regulation, the main genetic regulation, is performed by means of interactions (bindings) of regulatory elements. Although these mechanisms are not even completely understood, numerous efforts are invested in their understanding. What is known is that special proteins called transcription factors (TF) bind to certain DNA substrings forming TFBSs [53]. As a result of these unions, the genetic expression process, the process by which genes are transcribed into RNA, is enabled or disabled. The identification of these TFBSs and other elements that control gene expression and interactions among different TFs may explain the origin of living organisms, providing us important information about its complexity and its evolution. TFBSs and the elements that control gene expression are also known as motifs. For finding them, the MDP defines three conflicting objective functions to be maximized: motif length, support, and similarity. Motif length is the number of nucleotides that compose the pattern. To obtain the values of the other two objectives, we have to build the consensus motif, which is a string abstraction of the motif instances of all

sequences. Only those sequences that achieve a motif instance of certain quality with respect to the consensus motif are taken into account when we generate the final motif. This is indicated by the support (number of sequences taken into account), the second defined objective function. Those candidates which share at least a certain number of nucleotides with the consensus motif will be taken into account in this objective and in the following steps. After several experiments we established this threshold value of support to 50 %. Thus, the candidate motifs that are not able to reach this threshold value will not be taken into account in the support or in the subsequent similarity calculations because we believe that they are distorting the quality of the final solution. After calculating the support value, we continue with the last objective function, the similarity. For this, we must build the position count matrix (PCM), a well-known biological structure that is basically used to count the nucleotide bases that we have in each position of the selected candidate motifs. The dominant nucleotides of each position have to be normalized in the position frequency matrix (PFM), and then we can calculate the similarity value by averaging all the dominance values of each PFM column, as indicated in the following expression:

$$\text{Similarity(Motif)} = \frac{\sum_{i=1}^{l} \max_b \{f(b,i)\}}{l} \tag{1}$$

where $f(b,i)$ is the score of nucleotide b in column i in the PFM and $\max_b\{f(b,i)\}$ is the dominance value of the dominant nucleotide in column i. To better adapt the problem formulation to the real-world biological requirements, we have also incorporated several constraints that should be satisfied by each solution. These constraints (C_1, C_2, and C_3) must be met by all the generated solutions, i.e., if a solution does not meet the three defined constraints, it will be discarded and it will not be part of the population.

First, since motifs are usually very short [11], we have restricted the motif length to the range [6,22], where the minimum is 6 and the maximum is 22 (C_1). Additionally, we have set a minimum support value of 2 for the motifs of the sequence data sets composed of four or less sequences, and of three for the other ones (more than 4 sequences) (C_2). Normally, the binding sites are composed of motifs of all or nearly all sequences, and without this constraint is very easy to predict motifs with a high similarity (even 100 %) formed, for example, by candidates of only one sequence. Finally, we have also applied the complexity concept proposed in [12] and [13] (C_3). As it is explained in [12] and [13], this concept should be considered to avoid the appearance of biologically irrelevant solutions, for example, two candidate motifs: "AAAAAA" and "AAACAA" are very similar, but it is not a meaningful final motif. The average complexity for a final motif represents the total complexity score for each candidate motif. We calculate the complexity of a motif by using the following expression:

$$\text{Complexity} = \log_N \frac{l!}{\prod (n_i!)} \tag{2}$$

where $N = 4$ for DNA sequences, l is the motif length, and n_i is the number of nucleotides of type $i \in \{A, C, G, T\}$. For example, if we consider the motif "AAAA"

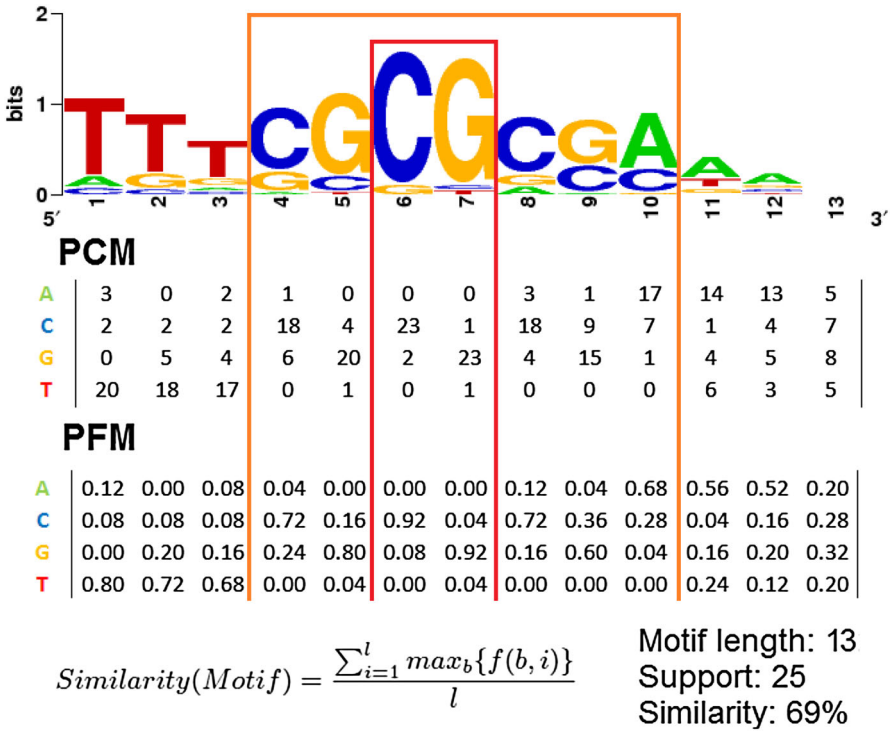


Fig. 1 Real motif discovery problem; the example includes the consensus/final motif, the position count matrix (PCM), the position frequency matrix (PFM), and the objective function values (motif length, support, and similarity)

($n_A = 4$, $n_T = 0$, $n_G = 0$, and $n_C = 0$) we will obtain a minimum complexity since we get the highest value in $\prod (n_i!)$. Otherwise, if we have, for example, the "ACGT" motif ($n_A = 1$, $n_T = 1$, $n_G = 1$, and $n_C = 1$) we will obtain a higher complexity. As we can see in Eq. (2), if we do not normalize the complexities when we compare motifs, the maximum complexity is highly dependent on the motif length. The compositional complexity calculation was revised such that the possible maximum complexity score is calculated for each possible motif length prior to the evolutionary computation. During evolution, each complexity score is rescaled between [0, 1] where the possible maximum complexity score is 1. This removes any potential bias in complexity relative to the motif length, as detailed in [12]. In our algorithm we have established a minimum complexity of 0.5 (50 %).

In Fig. 1 we include a real MDP example of size 13 (motif length = 13) corresponding to a solution of the E2F sequence data set. This instance is composed of 25 sequences and, in this case, all candidate motifs satisfy the support threshold requirement. Therefore, the support is equal to the number of sequences, i.e., support = 25. At this point and, as we have previously explained, we have to build the PCM and PFM to calculate the value of the last objective, the similarity. These two biological structures are also included in Fig. 1. As we can note, PCM indicates the number of

nucleotides of type A, C, G, and T in each motif position. Furthermore, PFM includes the same information but as percentages. These values are calculated by dividing the number of repetitions of each base by the total number of the used candidates, in this case 25. Once we build these structures, we just have to apply the expression indicated in Eq. (1) by using the maximum frequencies of each column to obtain the final similarity value; in this example we obtain a similarity = 69 %.

3 Related work

Although the MDP is a complex optimization problem, there are few works that investigate the application of parallelism to solve it. The first parallel works presented in the literature are dedicated to use the parallelism for accelerating the operation of MEME [1]. The execution of this biological tool includes an initial search phase (starting point search) and another which applies the known expectation maximization (EM) process [10]. The runtimes required for executing both phases increase with the problem size, making necessary the parallelism to speed up execution. In [18], the authors present one of the first works devoted to accelerate the implementation of MEME on distributed memory clusters by using message passing interface (MPI). MEME has also been parallelized by using specialized hardware such as field programmable gate arrays (FPGAs) in [40]. Such techniques have been successfully applied to accelerate the execution of other methods designed to solve other biological problems such as homologies search [50], multiple sequence alignment [35], and phylogenetic inference [32]. Recent improvements in multicore architectures caused the appearance of new effective technologies when parallelizing tools or algorithms, the GPUs (graphics processing units). This led the authors of [7] and [30] to exploit the advantages of GPUs to accelerate the execution of MEME, proposing GPU-MEME and CUDA-MEME, respectively. The authors of this latest work improved their proposal (CUDA-MEME) designing a new version called mCUDA-MEME [31], which combines CUDA, open multi-processing (OpenMP), and MPI. Among all of them, we highlight ParaMEME [18], which uses the most similar methodology to the one followed in this paper. Even with this, a comparison with this parallel biological tool is not possible because its source code is not available, so it is impossible to measure the time required to solve the instances addressed in this work, and the available results are based on an outdated hardware architecture (CPUs that are not currently accessible).

Aside from the research related to MEME, there are few other works such as [47] where the authors propose the parallelization of a method for discovering motifs called PrefixSpan, which is a data mining technique used to extract common patterns from databases. In [2], the authors develop Motif Discovery Toolkit, another algorithm that uses a graph-based parallel algorithm to predict where the genetic regulation can occur. The authors of [37] propose a parallel tool based on the ParSeq application, previously proposed by the same authors. Also, in [4] the parallelism is applied to accelerate the execution of an algorithm based on graph theory [42]. Other papers related to the discovery of motifs using parallelism are [41], where the Boolean matrices algorithm (BMA) model is parallelized on FPGAs; or [51] where the authors parallelize an

algorithm based on Gibbs sampling using GPUs. In short, we can note how there is a lack of works related to the application of parallelism for accelerating the execution of evolutionary algorithms solving the MDP; most of them are dedicated to parallelize existing biological tools such as MEME or Gibbs sampling. For this reason, we could not compare the parallel performance of our proposed evolutionary algorithm with that achieved by other parallel evolutionary proposals.

Regarding differential evolution, the authors of this work proposed a first version of DEPT in [15]. Some years later, a new version of this algorithm was proposed in [16] that incorporates improvements such as the use of the dominance and the crowding distance concepts. Finally, the algorithm presented in this paper (H-DEPT) presents novel improvements such as the new and specialized local search function that allows the algorithm to improve its performance and accuracy when predicting motifs in DNA sequences, and the application of the complexity concept that avoids the discovery of low biologically irrelevant solutions. With regard to parallelism, very preliminary parallel studies have been presented in [14,39]. However, these works have been considerably improved with the analysis of new and more complex instances (from 4 to 8 sequence data sets) and extended to the addressed optimization problem, the MDP. In addition, in this work we have also included a thread scheduling policy study, a comparison among the execution times of the algorithm on multicore machines with more cores (until 32 cores instead of 8), a more detailed study of the obtained results, and a more complete biological analysis of the predictions made, comparing our results with those achieved by other well-known biological tools.

4 Parallel multiobjective evolutionary algorithm

In this section we present the multiobjective evolutionary algorithm proposed for addressing the MDP. First, we describe the multiobjective adaptation of the algorithm (DEPT), also analyzing the operation of the incorporated local search function (H-DEPT). After this, we delve into the explanation of the parallel version.

4.1 Hybrid differential evolution with Pareto tournaments (H-DEPT)

In this work we modify a differential evolution (DE-based multiobjective evolutionary algorithm named DE with Pareto tournaments (DEPT). This algorithm incorporates a novel concept called Pareto tournaments function that combines the Pareto dominance and binary tournament concepts to facilitate the choice of the best multiobjective solution. The new algorithm is an improved version of DEPT, named hybrid DEPT (H-DEPT), thanks to the hybridization with a problem-aware local search.

DE is a simple, yet powerful, population-based evolutionary algorithm which aims to optimize complex problems by maintaining a population of individuals and combining their information to generate new solutions [46]. These new solutions are generated by applying a simple crossover-mutation formulation defined in a set of different selection (crossover/mutation) schemes (see Table 1). We can distinguish two kinds of selection schemes: exponential and binomial. The exponential schemes are similar with the two-point crossover. The first cut is randomly selected and then the algo-

Table 1 Defined DE selection schemes

Scheme	Mutant vector generation
Best/1/exp	$x_{\text{trial}} = x_{\text{best}} + F(x_{r1} - x_{r2})$
Rand/1/exp	$x_{\text{trial}} = x_{r3} + F(x_{r1} - x_{r2})$
Randtobest/1/exp	$x_{\text{trial}} = x_{r3} + F(x_{\text{best}} - x_{r3}) + F(x_{r1} - x_{r2})$
Best/2/exp	$x_{\text{trial}} = x_{\text{best}} + F(x_{r1} + x_{r2} - x_{r3} - x_{r4})$
Rand/2/exp	$x_{\text{trial}} = x_{r5} + F(x_{r1} + x_{r2} - x_{r3} - x_{r4})$
Best/1/bin	$x_{\text{trial}} = x_{\text{best}} + F(x_{r1} - x_{r2})$
Rand/1/bin	$x_{\text{trial}} = x_{r3} + F(x_{r1} - x_{r2})$
Randtobest/1/bin	$x_{\text{trial}} = x_{r3} + F(x_{\text{best}} - x_{r3}) + F(x_{r1} - x_{r2})$
Best/2/bin	$x_{\text{trial}} = x_{\text{best}} + F(x_{r1} + x_{r2} - x_{r3} - x_{r4})$
Rand/2/bin	$x_{\text{trial}} = x_{r5} + F(x_{r1} + x_{r2} - x_{r3} - x_{r4})$

rithm crosses the chromosomes consecutively until a second point determined by the *CR* probability. On the other hand, the binomial schemes applies the crossover probability to all the individual chromosomes. For further information about these crossover/mutation schemes, see [52].

H-DEPT has four important parameters: population size $PopSize$, crossover probability CR, mutation factor F, and selection scheme $Scheme$. For better describing the operation of the H-DEPT algorithm, we include its pseudocode in Algorithm 1.

The main structure of H-DEPT can be presented in three parts: initialization (lines 1–3 of Algorithm 1), new solutions generation (lines 5–18), and the Pareto tournament (line 19 and Algorithm 2). Analyzing in detail the H-DEPT pseudocode, we can note how the first three lines are devoted to the generation and evaluation of the initial population (P), and to the initialization of the solution archive (A) with the non-dominated

Algorithm 1 H-DEPT considering the rand/1/bin selection scheme.

Require: $MaxGenerations$, $PopSize$, CR, F, $Scheme$, and *local search parameters*
Ensure: Solution archive: A
1: $P \leftarrow$ generateRandomPopulation($PopSize$)
2: $A \leftarrow$ nonDominatedSolutions(P)
3: $P \leftarrow$ evaluatePopulation(P)
4: **for** g = 0, g < $MaxGenerations$, g = g + 1 **do**
5: **for** i = 0, i < $PopSize$, i = i + 1 **do**
6: $x_{target} \leftarrow P[i]$
7: $x_{r1} \leftarrow$ selectRandomIndividual(P) //$x_{r1} \neq x_{target}$
8: $x_{r2} \leftarrow$ selectRandomIndividual(P) //$x_{r2} \neq x_{r1}$
9: $x_{r3} \leftarrow$ selectRandomIndividual(P) //$x_{r3} \neq x_{r2}$
10: **for** j = 0, j < $NumberOfSequences$, j = j + 1 **do**
11: **if** CR indicates crossover **then**
12: $x_{\text{trial}}[j] \leftarrow x_{r3}[j] + F \cdot (x_{r1}[j] - x_{r2}[j])$
13: **else**
14: $x_{\text{trial}}[j] \leftarrow x_{target}[j]$
15: **end if**
16: **end for**
17: $x_{\text{trial}} \leftarrow$ applyLocalSearchFunction(x_{trial}, *local search parameters*)
18: $x_{\text{trial}} \leftarrow$ evaluateIndividual(x_{trial})
19: $P[i] \leftarrow$ ParetoTournament(P, x_{target}, x_{trial})
20: $A \leftarrow$ insertSolutionToArchive($P[i]$)
21: **end for**
22: **end for**
23: **return** A

Algorithm 2 H-DEPT Pareto tournament function.

Require: P, x_{target}, x_{trial}
Ensure: Best multiobjective solution: x_{new}
1: $x_{new} \leftarrow x_{\text{target}}$
2: **if** $x_{\text{trial}} \neq x_{\text{target}}$ **then**
3: **if** $x_{\text{trial}}.MOF < x_{\text{target}}.MOF$ **then**
4: $x_{\text{new}} \leftarrow x_{\text{trial}}$
5: **else if** $x_{\text{trial}}.MOF = x_{\text{target}}.MOF$ **then**
6: $P_{CW} \leftarrow$ insertIndividualsBelongToTheSameParetoFront(P, x_{target}, x_{trial})
7: $x_{\text{target}}.CW \leftarrow$ calculateCrowdingDistance(P_{CW}, x_{target})
8: $x_{\text{trial}}.CW \leftarrow$ calculateCrowdingDistance(P_{CW}, x_{trial})
9: **if** $x_{\text{trial}}.CW > x_{\text{target}}.CW$ **then**
10: $x_{\text{new}} \leftarrow x_{\text{trial}}$
11: **end if**
12: **end if**
13: **end if**
14: **return** x_{new}

solutions of the population. After the initialization of these necessary structures, H-DEPT begins to explore the solution space. For doing this, and considering the requirements of the established selection scheme (in this case rand/1/bin), the algorithm selects the necessary solutions (x_{target}, x_{r1}, x_{r2}, and x_{r3}) and applies the corresponding expression to generate a new solution (trial individual): $x_{\text{trial}} = x_{r3} + F \cdot (x_{r1} - x_{r2})$. Then, this new generated solution is improved by using the local search function. The operation of this function is detailed in the following paragraphs (see Algorithm 3). Finally, the last step is to evaluate the new improved solution and check whether it is better than the original one (target individual), the Pareto tournament (lines 19 and Algorithm 2). If both solutions are not the same, we first calculate a multiobjective fitness value (MOF) by using the following expression:

$$MOF(ind) = |isDominated(ind)| \cdot PopSize + |dominates(ind)| \qquad (3)$$

Algorithm 3 H-DEPT local search function.

Require: Individual: ind, Window size: WS, Reference: REF, Direction: DIR
Ensure: Improved individual: $newInd$
1: $newInd \leftarrow ind$
2: $refMotif \leftarrow$ selectReferenceMotif(REF)
3: **for** i = 0, i < $NumberOfSequences$, i = i + 1 **do**
4: $firstNucleotide \leftarrow$ selectStartingNucleotide(WS)
5: $found \leftarrow$ FALSE
6: **for** j = WS, (j ≥ 2 ∧ found = FALSE), j = j - 1 **do**
7: $string \leftarrow$ refMotif.substr($firstNucleotide$, j);
8: $found$, $pos \leftarrow$ searchStringInSequence(i,$string$,DIR)
9: **if** $found$ = TRUE **then**
10: $newInd.startingPos[i] \leftarrow pos$
11: $newInd \leftarrow$ evaluateSolution($newInd$)
12: **if** $newInd$ not dominates ind **then**
13: $newInd.startingPos[i] \leftarrow ind.startingPos[i]$
14: **end if**
15: **end if**
16: **end for**
17: **end for**
18: **return** $newInd$

where $isDominated(ind)$ indicates the number of solutions of the population that dominates ind, and $dominates(ind)$ represents the number of solutions of the population dominated by ind. So, the quality of each solution is obtained considering the quality of the remaining individuals from the population. For further information about this expression see [49]. If both solutions achieve the same MOF, i.e., both solutions belong to the same Pareto front, we have to make a second comparison (lines 5–12 of Algorithm 2). In this second comparison, the algorithm analyzes which solution provides greater dispersion to the population by means of the crowding distance concept [9]. The solution with the highest crowding distance will be the winner of the Pareto tournament and will be chosen for the next generation of the algorithm.

As we have previously exposed, the main feature of the H-DEPT algorithm is its local search function. Its operation is simple and effective at the same time and aims to improve the process of discovering motifs in DNA sequences. Taking this into account, the local search is applied at the end of each generation to optimize the quality of the newly generated solutions. Thus, this function is executed a total of $PopSize$ times at each generation. The local search requires three important parameters: window size (WS), search direction (DIR), and reference string (REF). WS defines the size of the substring that we have to search in the corresponding sequences, DIR defines the direction that we must take to find the selected substring, and finally the REF parameter indicates which motif (among all candidates and the consensus motif) is used as reference. Algorithm 3 shows the operation of the defined local search.

The local search receives as input the three previously defined parameters (WS, DIR, and REF) and the individual that we want to optimize (ind), returning an improved solution ($newInd$). First, it initializes the solution that it will modify (line 1 of Algorithm 3). Then, it selects the motif that will be used as reference taking into account the value of the REF parameter (line 2). After selecting the reference motif, the local search processes all the sequences for finding a small portion of this motif. The size of this portion is defined by the WS parameter. For starting the search process, the local search randomly selects the first string nucleotide considering the value of the WS parameter (line 4). Then, it starts looking for the substring of size WS (lines 6–16). The search is done by considering the direction indicated by the DIR parameter. If we find the string in the sequence (line 9), we will check if the generated solution has improved the previous one; if so, we keep the change, otherwise we reestablish the old starting location. On the other hand, if we do not find the string ($found = FALSE$), we decrease the considered WS parameter value (j variable) and repeat the search but, at this time, finding a smaller substring. All possible values of WS, DIR, and REF are indicated in Table 2.

4.2 Parallel H-DEPT

When we design and implement an algorithm for addressing the MDP, it is practically mandatory to study its parallelization capability. MDP is an NP-hard optimization problem whose complexity increases exponentially with the size of the input data [24]. Therefore, if we have to solve complex instances, parallel computing can be a good alternative to speed up the execution of the developed techniques. In this work,

Table 2 Possible values and functions of the local search parameters

Parameter	Values	Parameter value operation
Windows size (WS)	[2, 7]	Nucleotides that compose the substrings
Reference string (REF)	0	Candidate motif of the first sequence
	1	Candidate of a randomly selected sequence
	2	Consensus motif
	3	Candidate closer to the consensus motif at a nucleotide level
Search direction (DIR)	0	Beginning of the sequence
	1	Random direction (right or left) from the starting location
	2	Both directions (right and left), selecting the best achieved result

we present a parallel version of the H-DEPT algorithm. For parallelizing it, we have used OpenMP [5], a well-known API able to exploit the characteristics of shared-memory architectures by means of a set of compiler directives, library routines, and environment variables.

First of all, we must analyze its scheme and ensure that no data dependencies exist in the parts that we want to parallelize. In this particular case, as we can see in Algorithm 1, H-DEPT is mainly composed of two large loops (lines 4 and 5). Since the results obtained in a generation $g + 1$ depend on the results achieved in the previous generation g, the first loop cannot be parallelized. On the other hand, the second loop seems to have no dependencies because its main function is to generate one trial solution for each target individual. Additionally, for achieving good parallel results, it is also important to establish an appropriate population size ($PopSize$) and study in detail the time required by the algorithm to process each individual. These important aspects are analyzed in the following section.

In conclusion, the Parallel H-DEPT algorithm distributes the trial generation work into different threads. In a formal way, if we assume an N-core system and a population size equal to $PopSize$, then each thread will process $PopSize/N$ solutions, i.e., each thread will execute $PopSize/N$ iterations of those parallelized for loop. Furthermore, a synchronization barrier exists at the end of the for loop that prevents starting the next generation until all threads have finished their parallel executions. In this way, we avoid possible errors due to data dependencies. A graphical example where we describe the operation of our parallel algorithm is shown in Fig. 2. In this example we assume a population size $PopSize = 96$ and a multicore system with 32 cores ($N = 32$). As we can see, each thread is responsible for executing $PopSize/N = 96/32 = 3$ loop iterations, that is, processing 3 individuals of the population. Following this parallel scheme we can execute the algorithm up to 32 times faster (considering an efficiency of 100 %). However, since threads are launched in each generation, the algorithm invests time in creating and eliminating these threads. This time penalizes the achieved parallel results and is an important aspect that will be analyzed in the following sections.

Finally, in Table 3 we present an illustrative distribution of the individuals processed by each thread (Th_i) at each moment by using different multicore systems: 2, 4, 8, 16, and 32 cores. As we can observe, while the sequential version needs 95 additional

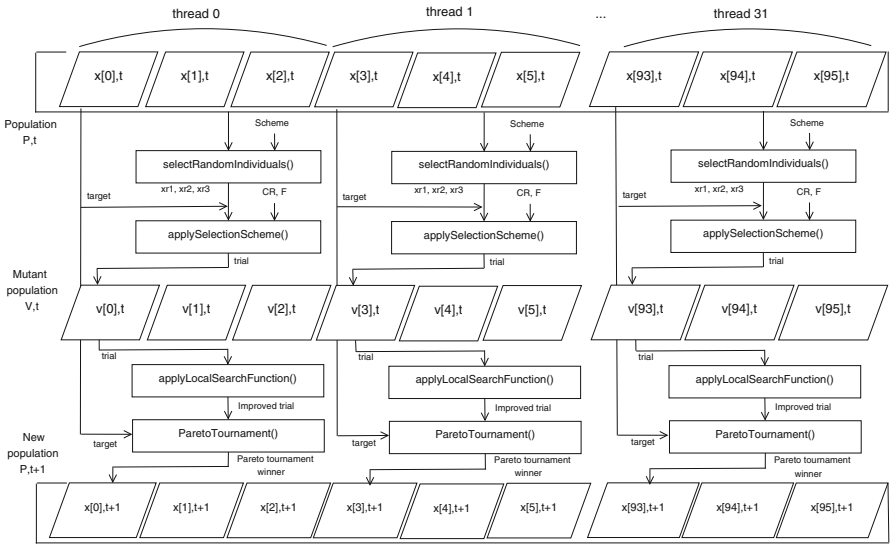


Fig. 2 General outline of the parallel H-DEPT algorithm

time units (t_{95}) to generate the last trial individual and finish the execution of the corresponding generation, the multicore versions only need 47, 23, 11, 5, and 2 time units (t_{47}, t_{23}, t_{11}, t_5, and t_2), respectively. Thus, the most powerful multicore system (32 cores) is able to make the same operations as the sequential version, in 3 time units (t_{0-2}) instead of 96.

5 Experimentation

This section is aimed at presenting the experiments conducted to evaluate the performance of the previously described algorithm. Before starting the study of the obtained results, we explain the methodology followed in the conducted experiments, describe the representation of the individuals, and configure the algorithm to known which parameter values achieve the best results.

5.1 Experimental methodology

For the parameter adjustments, the multicore experiments, and the final biological comparisons, we have followed the same experimental methodology. We have carried out 30 independent runs, using the average results for the comparisons; so we ensure certain statistical significance in the results. In addition, in all experiments we have considered the same individual representation. This representation is shown in Table 4. As we can see, it includes the necessary information for building a possible motif, which is represented by the motif length and the starting locations of each candidate motif in each DNA sequence. With these values, and following the steps described in Sect. 3, we can calculate the value of the three defined objective functions: motif length, support, and similarity.

Table 3 Illustrative distribution of the threads tasks using 1, 2, 4, 8, 16, and 32 cores

	Seq.	2-cores		4-cores				8-cores							
	Th_0	Th_0	Th_1	Th_0	Th_1	Th_2	Th_3	Th_0	Th_1	Th_2	Th_3	Th_4	Th_5	Th_6	Th_7
t_0	0	0	48	0	24	48	72	0	12	24	36	48	60	72	84
t_1	1	1	49	1	25	49	73	1	13	25	37	49	61	73	85
t_2	2	2	50	2	26	50	74	2	14	26	38	50	62	74	86
t_3	3	3	51	3	27	51	75	3	15	27	39	51	63	75	87
t_4	4	4	52	4	28	52	76	4	16	28	40	52	64	76	88
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
t_{10}	10	10	58	10	34	58	82	10	22	34	46	58	70	82	94
t_{11}	11	11	59	11	35	59	83	11	23	35	47	59	71	83	95
	...	...	...	...	...	...	...								
t_{22}	22	22	70	22	46	70	94								
t_{23}	23	23	71	23	47	71	95								
	...	...	...												
t_{46}	46	46	94												
t_{47}	47	47	95												
	...														
t_{94}	94														
t_{95}	95														

	16-cores								32-cores							
	Th_0	Th_1	Th_2	Th_3	...	Th_{12}	Th_{13}	Th_{14}	Th_{15}	Th_0	Th_1	...	Th_{14}	Th_{15}	...	Th_{30}	Th_{31}
t_0	0	6	12	18	...	72	78	84	90	0	3	...	42	45	...	90	93
t_1	1	7	13	19	...	73	79	85	91	1	4	...	43	46	...	91	94
t_2	2	8	14	20	...	74	80	86	92	2	5	...	44	47	...	92	95
t_3	3	9	15	21	...	75	81	87	93								
t_4	4	10	16	22	...	76	82	88	94								
t_5	5	11	17	23	...	77	83	89	95								

Table 4 Individual representation

	Seq. 1	Seq. 2	Seq. 3		Seq. n
Motif length	S_1	S_2	S_3	...	S_n

Furthermore, the value of the algorithm parameters have been adequately adjusted after performing numerous experiments. To configure the algorithm parameters, we first analyzed which of them were the most influential ones in terms of the quality of the results. Once identified, we configure them in that order. For doing this, we select a minimal set of five values uniformly distributed in their possible ranges and then execute the algorithm. Those parameter values that achieve the best results in most

Table 5 Parameterization of the algorithm

Parameterization used in H-DEPT	
Population size ($PopSize$)	96 individuals
Crossover probability (CR)	25 %
Mutation factor (F)	3 % of the individual chromosomes
Selection scheme ($Scheme$)	Rand/1/bin
Parameterization used in the local search function	
Windows size (WS)	6 nucleotides
Reference string (REF)	1, candidate of a randomly selected sequence
Search direction (DIR)	2, both directions, selecting the best one

instances will be established. This process is repeated for all parameters. In Table 5, we show the best configurations found for the H-DEPT algorithm. All these experiments have been performed on an AMD Opteron Processor 6174 (2.20 GHz) with 64 GB of RAM and Scientific Linux 6.1 (64 bits), compiling the software with GCC 4.4.5.

To study the behavior of the parallel algorithm we measure its performance by using three indicators: execution time (T, in seconds), speedup (S_c), and efficiency (E_c). Speedup refers to how much a parallel algorithm is faster than a corresponding sequential algorithm, and to calculate its value we need the sequential time (T_1) and the parallel time using c cores (T_c):

$$S_c = \frac{T_1}{T_c} \tag{4}$$

Furthermore, efficiency is a value between 0 and 1 typically used to estimate how well utilized the processors are in solving a problem. It is measured by using the sequential and parallel times, i.e., the speedup; and the number of cores used for running the parallel application (c):

$$E_c = \frac{S_c}{c} = \frac{T_1}{c \cdot T_c} \tag{5}$$

Ideally, the best speedup is equal to the number of cores ($S_c = c$) and the best efficiency is equal to 1 ($E_c = 1$); however, these results are very difficult to achieve due to several reasons; one example can be that there may be parts of the algorithm that do not accept being parallelized. For further information about these parallel indicators, see [17].

5.2 MDP instances

For the empirical study we have used a total of eight instances. These instances were introduced for evaluating the GAME method in [48] and include motifs of different properties. The CRP data set is a widely tested benchmark [45] containing 23 cyclic

Table 6 Properties of the eight used real instances

Data set	CREB	CRP	E2F	ERE	MEF2	MyOD	SRF	TBP
N	17	18	25	25	17	17	20	95
L	200	105	200	200	200	200	200	200
w	8	22	11	13	7	6	10	6
♯	19	23	27	25	17	21	36	95
MaxGenerations	3,750	1,500	2,500	2,500	7,000	3,750	5,000	1,000
Runtimes (s)	≈170	≈70	≈200	≈200	≈100	≈300	≈250	≈850

N is the number of sequences, L the sequence length, w the motif width, and ♯ is the number of true TFBSs annotated

adenosine monophosphate receptor binding sites of *Escherichia coli*. The estrogen receptor is a ligand-activated enhancer protein which can activate gene expression in response to estradiol. We analyze 25 genomic sequences that contain known estrogen receptor elements (ERE) binding sites [23]. Finally, we examine the regulation of 25 sequences that contain 27 binding sites for transcription factors in the E2F family [22]. We have also used a set composed by five additional instances: CREB, MEF2, MyOD, SRF, and TBP from the ABS database of annotated regulatory binding sites [3]. The number of sequences of these eight instances ranges from 17 to 95, and the size of their sequences from 105 to 200 bases. This allows us to have a set of instances with different properties that can help us to correctly evaluate the performance of our proposal. Their properties are described in Table 6. In this table we also include the runtimes required by the algorithm to obtain quality solutions in each instance. These times depend on the biological complexity of instances, i.e., the number of sequences, the number of nucleotides per sequence, and the size of the included binding sites. On the other hand, the number of generations used in each instance has been calculated according to the times required by the algorithm.

5.3 Parallel results

As explained in the previous sections, parallelizing an algorithm is not a simple task. In Sect. 4.2 we saw how the H-DEPT algorithm has no data dependencies in its second main loop (lines 5–21 of Algorithm 1). This means that if we parallelize it, we could produce significant accelerations. However, first of all we must deepen the tasks performed by each thread. The first operations performed by the threads in the parallelized loop are dedicated to generate the trial individuals by taking advantage of the genetic information of other individuals of the population (lines 6–16 of Algorithm 1), in this case, by using the information of x_{r1}, x_{r2}, and x_{r3}. These operations are the same for all individuals and, consequently, the time spent by each thread is practically identical. Continuing with the analysis of the operations performed by the threads, we can see how the next function to be executed is the local search (line 17 of Algorithm 1). Studying the operation of the local search function (see Algorithm 3), we can note how the number of iterations made in the loop of line 6 may not be equal in all

cases. Some solutions can easily improve in a few iterations and others may require more operations to improve. When parallelizing the algorithm, this aspect should be taken into account since we are assigning tasks with different temporal requirements to the threads. Finally, the same happens for the last operation of the parallelized loop (line 19) corresponding to the Pareto tournament function. If we compare two solutions belonging to the same Pareto front, we have to build the P_{CW} population and calculate the corresponding crowding distances. Otherwise, if we have two solutions belonging to different Pareto fronts, the algorithm will not have to perform these operations.

A loop has logical iterations numbered $0, 1, \ldots, N-1$ where N is the number of loop iterations, and the numbers denote the order in which the iterations would be executed in a single thread. OpenMP defines an schedule clause that specifies how the iterations of a given loop are associated with the different threads. The scheduling kind can be static, dynamic, or guided. When static schedule is specified, iterations are assigned to the threads in a round-robin fashion in the order of the thread number. When dynamic schedule is used, the iterations are distributed to threads as the threads request them. Finally, when guided schedule is considered, the iterations are assigned to the threads in exponentially decreasing block sizes until a minimum size is obtained.

To study what thread scheduling policy allows the algorithm to obtain the best parallel results, we have performed experiments with the three described ones: static, dynamic, and guided. More specifically, we have solved eight real instances (see Sect. 5.2) on different multicore systems composed of 2, 4, 8, 16, and 32 cores. The obtained results are shown in Tables 7 and 8. These tables include the sequential and parallel times using the three possible thread scheduling policies, together with the corresponding speedups and efficiencies. In addition, we also indicate in gray the best results, i.e., the greatest accelerations. In all experiments we have considered the default $chunk\_size$ value. Although we know that this value can be configured, our experience demonstrates that the best performances are generally obtained with the default values.

In Table 7a we include the results obtained on the two-core multicore system. In this particular case, we can see how the results obtained by applying the different thread scheduling policies are similar: an average efficiency of 90.20 % with static, 90.62 % with dynamic, and 90.01 % with guided. Furthermore, the three scheduling policies achieve the best results in some instance: two best results with static, five with dynamic, and one with guided. With this data, we can conclude that, when using a two-core multicore system, the negative aspects of the previously exposed uneven load balancing problem does not influence the results. Therefore, we may use any of the three tested thread schedules. The same happens in the experiments conducted on the four-core multicore system (see Table 7b). The algorithm achieves similar results by applying any thread scheduling policy: an average efficiency of 88.33 % with static, 87.51 % with dynamic, and 88.18 % with guided. This can be due to each thread being assigned a large number of individuals ($96/4 = 24$) and the total work done by each thread being still very similar. The performance of the parallel algorithm will be affected when threads process fewer individuals and some of these solutions are complicated. Thus, the conclusions drawn for this second multicore system (4 cores)

Table 7 Mean runtime (X represents the mean time in seconds and Std the standard deviation), speedup (S), and efficiency (E) obtained by the parallel H-DEPT on different multicore systems

	Sequential	Static			Dynamic			Guided		
	X_{Std}	X_{Std}	S_2	E_2 (%)	X_{Std}	S_2	E_2 (%)	X_{Std}	S_2	E_2 (%)
(a) 2-cores										
CREB	$154.31_{3.65}$	$86.35_{1.75}$	1.79	89.35	$85.42_{1.78}$	1.81	90.33	$85.25_{1.38}$	1.81	90.51
CRP	$68.50_{1.17}$	$37.88_{0.79}$	1.81	90.41	$38.13_{1.02}$	1.80	89.81	$38.30_{1.29}$	1.79	89.43
E2F	$213.45_{6.10}$	$118.15_{2.71}$	1.81	90.33	$117.18_{3.25}$	1.82	91.08	$117.40_{2.82}$	1.82	90.91
ERE	$216.84_{2.03}$	$118.15_{1.14}$	1.84	91.77	$117.18_{1.47}$	1.85	92.52	$120.53_{1.53}$	1.80	89.96
MEF2	$105.11_{2.43}$	$58.86_{0.89}$	1.79	89.29	$58.66_{1.00}$	1.79	89.59	$59.09_{1.67}$	1.78	88.95
MyOD	$288.76_{8.03}$	$157.87_{3.90}$	1.83	91.46	$158.33_{3.74}$	1.82	91.19	$159.86_{4.27}$	1.81	90.38
SRF	$264.39_{5.05}$	$147.67_{3.34}$	1.79	89.52	$146.21_{2.68}$	1.81	90.41	$146.85_{3.04}$	1.80	90.02
TBP	$871.57_{5.41}$	$486.98_{4.75}$	1.79	89.49	$484.13_{4.10}$	1.80	90.01	$484.17_{5.38}$	1.80	90.01
	272.87	151.49	1.80	90.20	150.66	1.81	90.62	151.43	1.80	90.01
	Sequential	Static			Dynamic			Guided		
	X_{Std}	X_{Std}	S_4	E_4 (%)	X_{Std}	S_4	E_4 (%)	X_{Std}	S_4	E_4 (%)
(b) 4-cores										
CREB	$154.31_{3.65}$	$43.57_{0.84}$	3.54	88.54	$52.72_{2.82}$	2.93	73.17	$43.49_{0.59}$	3.55	88.71
CRP	$68.50_{1.17}$	$19.44_{0.26}$	3.52	88.07	$19.20_{0.33}$	3.57	89.17	$19.15_{0.26}$	3.58	89.45
E2F	$213.45_{6.10}$	$60.17_{1.48}$	3.55	88.69	$59.17_{1.37}$	3.61	90.19	$59.87_{1.55}$	3.57	89.13
ERE	$216.84_{2.03}$	$61.53_{0.78}$	3.52	88.11	$60.65_{0.71}$	3.58	89.38	$61.03_{0.52}$	3.55	88.83
MEF2	$105.11_{2.43}$	$29.84_{0.50}$	3.52	88.06	$29.48_{0.47}$	3.56	89.12	$29.30_{0.60}$	3.59	89.68
MyOD	$288.75_{8.03}$	$80.67_{1.87}$	3.58	89.49	$79.73_{1.61}$	3.62	90.55	$80.53_{2.03}$	3.59	89.65
SRF	$264.39_{5.05}$	$75.48_{1.18}$	3.50	87.57	$74.00_{1.08}$	3.57	89.33	$81.42_{9.23}$	3.25	81.19
TBP	$871.57_{5.41}$	$247.29_{2.88}$	3.52	88.11	$244.32_{2.47}$	3.57	89.18	$245.34_{2.56}$	3.55	88.81
	272.87	77.25	3.53	88.33	77.41	3.50	87.51	77.51	3.53	88.18

are the same as those obtained for the previous multicore system (2 cores). The thread scheduling policy does not affect the quality of the parallel results, so that we can apply any of them.

The results achieved on the eight-core multicore system show a slightly different behavior. Bt analyzing the information of Table 8a, we can note how the guided and dynamic thread schedules get better results than those achieved by static. Although it seems that the static and dynamic results are similar (average efficiency of 84.29 and 84.91 %, respectively), the dynamic final result is heavily influenced by the poor performance presented by the algorithm in the first two instances (CREB and CRP). These results show that the parallel algorithm begins to present temporal differences in the processing of the tasks and, consequently, the achieved performance also begins to be influenced. Therefore, when using this multicore system a guided or dynamic thread scheduling policy should be established.

Table 8 Mean runtime (X represents the mean time in seconds and Std the standard deviation), speedup (S), and efficiency (E) obtained by the parallel H-DEPT on different multicore systems

	Sequential	Static			Dynamic			Guided		
	X_{Std}	X_{Std}	S_8	E_8 (%)	X_{Std}	S_8	E_8 (%)	X_{Std}	S_8	E_8 (%)
(a) 8-cores										
CREB	$154.31_{3.65}$	$24.33_{2.49}$	6.34	79.27	$25.21_{3.10}$	6.12	76.52	$25.34_{2.49}$	6.09	76.13
CRP	$68.50_{1.17}$	$10.20_{0.85}$	6.71	83.92	$10.77_{1.62}$	6.36	79.51	$9.92_{0.16}$	6.91	86.34
E2F	$213.45_{6.10}$	$30.86_{1.03}$	6.92	86.46	$30.25_{0.72}$	7.06	88.21	$30.41_{0.77}$	7.02	87.73
ERE	$216.84_{2.03}$	$32.20_{1.43}$	6.73	84.18	$31.24_{0.30}$	6.94	86.77	$31.37_{0.34}$	6.91	86.41
MEF2	$105.11_{2.43}$	$15.70_{0.66}$	6.69	83.66	$15.32_{0.22}$	6.86	85.78	$15.36_{0.31}$	6.84	85.53
MyOD	$288.75_{8.03}$	$41.98_{0.93}$	6.88	85.98	$40.83_{0.94}$	7.07	88.39	$41.28_{1.06}$	6.99	87.44
SRF	$264.39_{5.05}$	$38.89_{0.56}$	6.80	84.99	$38.03_{0.70}$	6.95	86.90	$38.20_{0.69}$	6.92	86.52
TBP	$871.57_{5.41}$	$126.94_{1.52}$	6.87	85.83	$124.94_{1.05}$	6.98	87.20	$124.67_{1.00}$	6.99	87.38
	272.87	40.14	6.74	84.29	39.57	6.79	84.91	39.57	6.83	85.43
	Sequential	Static			Dynamic			Guided		
	X_{Std}	X_{Std}	S_{16}	E_{16} (%)	X_{Std}	S_{16}	E_{16} (%)	X_{Std}	S_{16}	E_{16} (%)
(b) 16-cores										
CREB	$154.31_{3.65}$	$20.10_{26.42}$	7.68	47.97	$12.16_{0.55}$	12.69	79.31	$13.13_{3.77}$	11.76	73.48
CRP	$68.50_{1.17}$	$8.61_{11.94}$	7.95	49.70	$6.16_{1.76}$	11.12	69.47	$6.28_{2.33}$	10.91	68.17
E2F	$213.45_{6.10}$	$20.89_{9.62}$	10.22	63.85	$17.11_{4.09}$	12.48	77.98	$18.28_{5.83}$	11.67	72.96
ERE	$216.84_{2.03}$	$20.65_{8.88}$	10.50	65.65	$16.96_{1.56}$	12.79	79.91	$17.69_{3.68}$	12.26	76.61
MEF2	$105.11_{2.43}$	$11.15_{5.35}$	9.43	58.93	$8.33_{0.67}$	12.61	78.82	$10.10_{3.66}$	10.40	65.02
MyOD	$288.75_{8.03}$	$25.99_{10.60}$	11.11	69.44	$22.63_{1.37}$	12.76	79.77	$25.46_{6.84}$	11.34	70.89
SRF	$264.39_{5.05}$	$23.94_{8.47}$	11.05	69.03	$20.64_{6.83}$	12.81	80.06	$25.63_{10.03}$	10.31	64.46
TBP	$871.57_{5.41}$	$77.69_{34.53}$	11.22	70.11	$65.41_{1.54}$	13.32	83.28	$65.72_{0.99}$	13.26	82.89
	272.87	26.13	9.89	61.84	21.18	12.57	78.58	22.79	11.49	71.81
	Sequential	Static			Dynamic			Guided		
	X_{Std}	X_{Std}	S_{32}	E_{32} (%)	X_{Std}	S_{32}	E_{32} (%)	X_{Std}	S_{32}	E_{32} (%)
(c) 32-cores										
CREB	$154.31_{3.65}$	$8.77_{5.26}$	17.60	55.01	$7.55_{0.78}$	20.44	63.88	$7.81_{1.41}$	19.76	61.75
CRP	$68.50_{1.17}$	$3.96_{1.26}$	17.28	54.00	$3.57_{0.43}$	19.18	59.92	$3.60_{0.58}$	19.02	59.44
E2F	$213.45_{6.10}$	$10.31_{2.72}$	20.70	64.70	$9.67_{1.08}$	22.08	69.00	$9.80_{1.96}$	21.77	68.04
ERE	$216.84_{2.03}$	$10.61_{2.85}$	20.44	63.88	$9.82_{1.18}$	22.09	69.02	$10.36_{2.26}$	20.93	65.41
MEF2	$105.11_{2.43}$	$5.79_{1.93}$	18.17	56.78	$5.13_{0.77}$	20.49	64.03	$5.52_{1.85}$	19.03	59.47
MyOD	$288.75_{8.03}$	$14.23_{2.20}$	20.29	63.39	$14.03_{1.52}$	20.58	64.32	$14.71_{2.94}$	19.63	61.36
SRF	$264.39_{5.05}$	$12.89_{2.39}$	20.51	64.11	$12.60_{1.59}$	20.99	65.59	$12.80_{2.30}$	20.66	64.56
TBP	$871.57_{5.41}$	$43.69_{10.56}$	19.95	62.34	$38.89_{2.65}$	22.41	70.04	$39.50_{2.82}$	22.07	68.95
	272.87	13.78	19.37	60.53	12.66	21.03	65.72	13.01	20.36	63.62

Table 9 Best parallel results of H-DEPT using different multicore systems (S_c is the speedup and E_c the efficiency for c cores)

	Sequential	2-cores		4-cores		8-cores		16-cores		32-cores	
		S_2	E_2 (%)	S_4	E_4 (%)	S_8	E_8 (%)	S_{16}	E_{16} (%)	S_{32}	E_{32} (%)
CREB	154.31	1.81	90.51	3.55	88.71	6.34	79.27	12.69	79.31	20.44	63.88
CRP	68.50	1.81	90.41	3.58	89.45	6.91	86.34	11.12	69.47	19.18	59.92
E2F	213.45	1.82	91.08	3.61	90.19	7.06	88.21	12.48	77.98	22.08	69.00
ERE	216.84	1.85	92.52	3.58	89.38	6.94	86.77	12.79	79.91	22.09	69.02
MEF2	105.11	1.79	89.58	3.59	89.68	6.86	85.78	12.61	78.82	20.49	64.03
MyOD	288.75	1.83	91.46	3.62	90.55	7.07	88.39	12.76	79.77	20.58	64.32
SRF	264.39	1.81	90.41	3.57	89.33	6.95	86.90	12.81	80.06	20.99	65.59
TBP	871.57	1.80	90.01	3.57	89.18	6.99	87.38	13.32	83.28	22.41	70.04
	272.87	1.81	90.75	3.58	89.56	6.89	86.13	12.57	78.58	21.03	65.72

Finally, the results for the multicore systems composed of 16 and 32 cores are included in Table 8b and c. Analyzing these results, we can see how the algorithm achieves the best parallel results in all instances by using the dynamic thread scheduling policy. In addition, the algorithm also achieves considerable improvements in the average efficiency: 78.58 vs. 61.84 and 71.81 % presented by the rest of the policies for 16 cores; and 65.72 vs. 60.53 and 63.62 % for 32 cores. These results confirm the previously raised hypothesis, because now each thread is responsible for processing $96/16 = 6$ and $96/32 = 3$ individuals, respectively, and when this number of individuals is so low, the processing of a complicated solution greatly penalizes the performance of the parallel application. Dynamic scheduling policy solves this work-balancing problem and allows achieving good parallel results. In this latter study, we can conclude that the best thread scheduling policy for multicore systems composed of 16 and 32 cores is the dynamic one.

In conclusion, the information presented in Tables 7 and 8 shows that any thread scheduling policy is good for the systems composed of two or four cores. Guided and dynamic schedules are the most appropriate ones for the 8-core system; and finally, the best results on 16-core and 32-core multicore systems are achieved by defining a dynamic schedule of threads. In Table 9 we summarize the best parallel results achieved on all the multicore systems.

In addition, in Fig. 3 we also include a more graphical and intuitive comparison where we compare the runtimes achieved by the parallel version of H-DEPT with the ideal times obtained by a fully parallel algorithm, i.e., we compare the runtimes that the algorithm should ideally get if its efficiency was always equal to 100 %, with the experimental results achieved by the H-DEPT algorithm on the 2, 4, 8, 16, and 32-core multicore systems. As we can observe, the obtained results are only slightly higher than the ideal times. Thus, we can say that the proposed algorithm presents a scheme that makes it suitable for parallelization. Finally, to sum up, we include in Fig. 4 the mean speedup and efficiency achieved by the H-DEPT algorithm when solving the eight real sequence data sets.

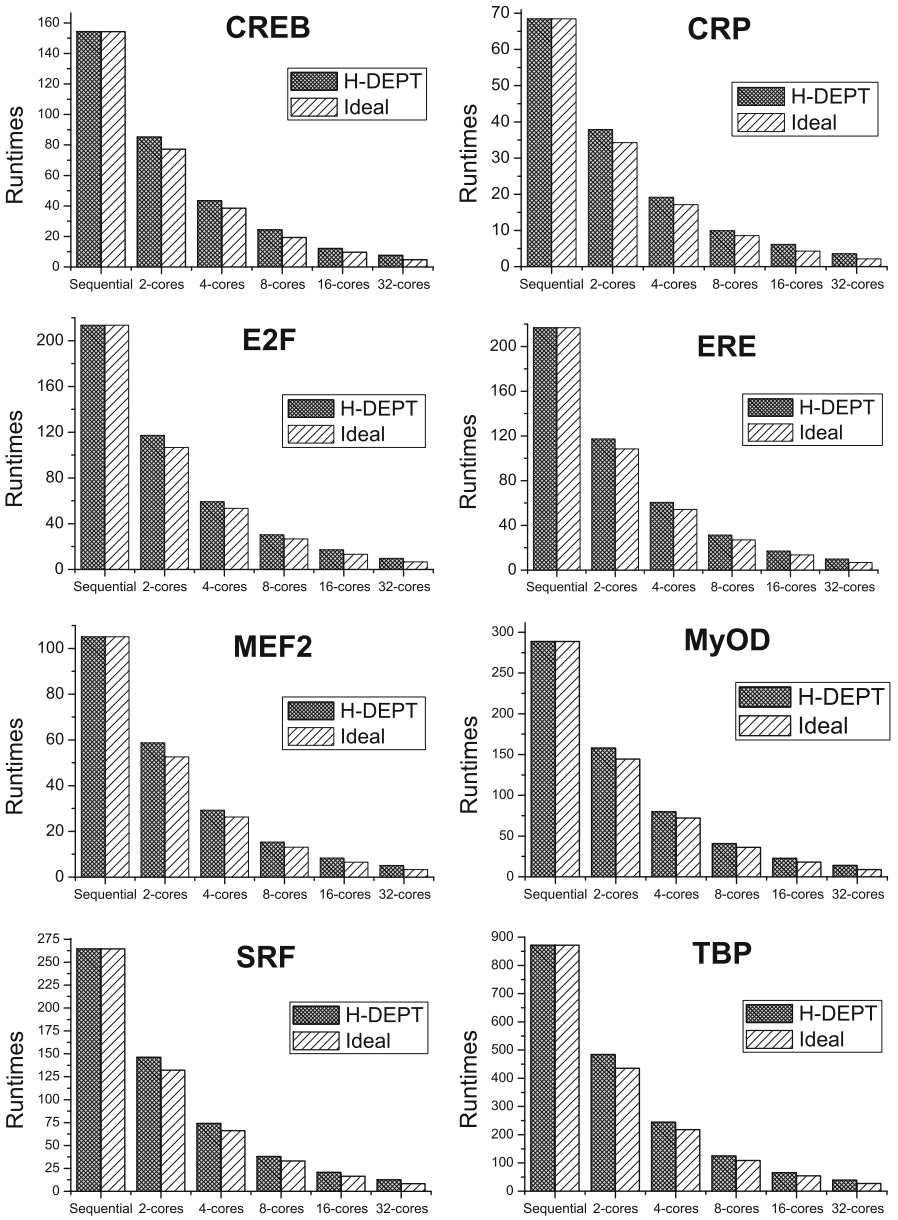


Fig. 3 Comparisons between the H-DEPT and the ideal times in seconds

5.4 Analysis of the problem solutions

In this section we study the biological quality of the solutions predicted by our algorithm. It is important to note that the solutions predicted by the sequential and parallel algorithms are the same, but more rapidly obtained. In this comparison we compare

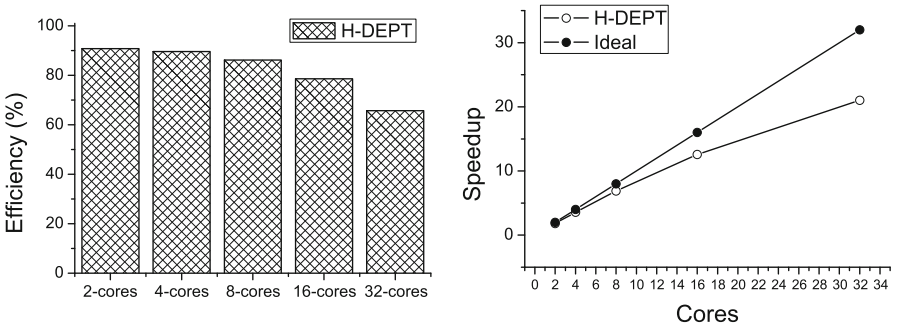


Fig. 4 Mean speedups and efficiencies of H-DEPT in all the solved instances

its predictions with those made by the GAME algorithm [48], MEME [1], and BioProspector [28]. In addition, we also include the solutions resulting from the application of the BioOptimizer [20] optimization program by using the solutions discovered by MEME and BioProspector. To compare the correctness of the predictions made, we use the standard information retrieval metrics of precision and recall [43]:

$$\text{Precision} = \frac{\#\text{ of predicted motif sites that are true sites}}{\#\text{ of predicted motif sites}} \tag{6}$$

$$\text{Recall} = \frac{\#\text{ of predicted motif sites that are true sites}}{\#\text{ of true sites}} \tag{7}$$

To make fair comparisons with these biological methods and to compare our results with those presented in [48], we have used the same biological metrics and followed the same methodology. That is, we consider that a site is correctly predicted when at least three base pairs are correctly predicted. These two biological metrics are combined in the following indicator:

$$F_{score} = 2 \times Precision \times \frac{Recall}{Precision + Recall} \tag{8}$$

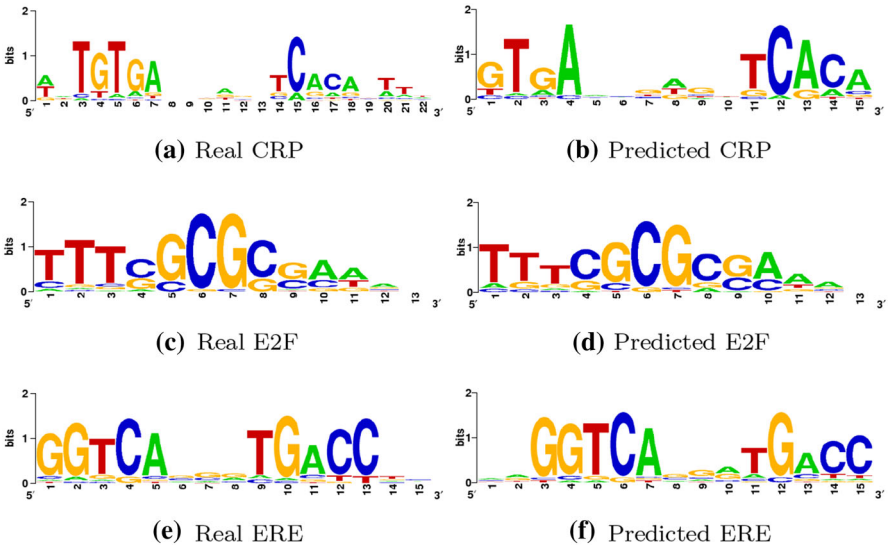
which is a standard method of comparison [43]. High F_{score} values only occur when high precision and recall are achieved. The GAME, MEME, and BioProspector solutions included in this comparative section are the best solutions found by each method. Regarding BioOptimizer, the included solutions are those resulting from the optimization process.

In Table 10 we show the results of this comparison. This table includes, as we have already mentioned, the best solutions obtained by GAME, MEME, BioProspector, and BioOptimizer, as well as some solutions discovered by H-DEPT. After carrying out 30 independent runs of the properly configured algorithm, we selected the best solutions of each execution and each instance. To select these solutions, we measured the biological quality of all solutions belonging to the best Pareto front (non-dominated solutions) of each execution and each instance by calculating the value of the described biological indicators, and selecting those solutions that have higher F_{score}. Once done, we have

Table 10 Comparison among the solutions discovered by H-DEPT, GAME, MEME, BioOptimizer, and BioProspector

Predictor	w	$\|A\|$	Precision	Recall	F score	w	$\|A\|$	Precision	Recall	F score
	CREB					MEF2				
H-DEPT (best)	14	17	17/17	17/19	0.94	9	17	17/17	17/17	1.00
H-DEPT (median)	**14**	**17**	**17/17**	**17/19**	**0.94**	**9**	**17**	**17/17**	**17/17**	**1.00**
H-DEPT (worst)	15	14	14/14	14/19	0.85	10	14	13/14	13/17	0.84
GAME	8	22	15/22	15/19	0.73	9	17	15/17	15/17	0.88
BioOpt. (MEME)	12	15	10/15	10/19	0.59	13	15	14/15	14/17	0.88
BioOpt. (BioPro.)	9	17	12/17	12/19	0.67	11	19	11/19	11/17	0.61
MEME	11	15	10/15	10/19	0.59	9	15	14/15	14/17	0.88
BioProspector	8	20	13/20	13/19	0.67	7	17	12/17	12/17	0.71
	CRP					MyOD				
H-DEPT (best)	15	18	18/18	18/23	0.88	15	15	14/15	14/21	0.78
H-DEPT (median)	**15**	**17**	**17/17**	**17/23**	**0.85**	**14**	**17**	**12/17**	**12/21**	**0.63**
H-DEPT (worst)	8	16	14/16	14/23	0.72	15	10	8/10	8/21	0.52
GAME	19	17	16/17	16/23	0.80	7	21	10/21	10/21	0.48
BioOpt. (MEME)	24	13	12/13	12/23	0.67	10	10	0/10	0/21	0.00
BioOpt. (BioPro.)	24	13	12/13	12/23	0.67	11	11	0/11	0/21	0.00
MEME	24	13	12/13	12/23	0.67	9	8	0/8	0/21	0.00
BioProspector	22	9	9/9	9/23	0.56	6	18	0/18	0/21	0.00
	E2F					SRF				
H-DEPT (best)	11	25	24/25	24/27	0.92	11	20	20/20	20/36	0.71
H-DEPT (median)	**11**	**25**	**23/25**	**23/27**	**0.88**	**11**	**20**	**20/20**	**20/36**	**0.71**
H-DEPT (worst)	12	25	21/25	21/27	0.81	8	20	18/20	18/36	0.64
GAME	11	24	23/24	23/27	0.90	10	47	33/47	33/36	0.80
BioOpt. (MEME)	13	27	20/27	20/27	0.74	14	51	32/51	32/36	0.74
BioOpt. (BioPro.)	13	27	19/27	19/27	0.70	14	50	32/50	32/36	0.74
MEME	13	23	19/23	19/27	0.76	13	48	28/48	28/36	0.67
BioProspector	11	21	11/21	11/27	0.46	10	35	25/35	25/36	0.70
	ERE					TBP				
H-DEPT (best)	14	24	24/24	24/25	0.98	9	93	86/93	86/95	0.91
H-DEPT (median)	**13**	**23**	**22/23**	**22/25**	**0.92**	**9**	**95**	**85/95**	**85/95**	**0.89**
H-DEPT (worst)	15	20	19/20	19/25	0.84	9	84	60/84	60/95	0.67
GAME	13	26	19/26	19/25	0.75	7	91	78/91	78/95	0.84
BioOpt. (MEME)	15	22	17/22	17/25	0.72	12	79	35/79	35/95	0.40
BioOpt. (BioPro.)	16	23	18/23	18/25	0.75	9	78	65/78	65/95	0.75
MEME	15	17	15/17	15/25	0.71	12	50	26/50	26/95	0.36
BioProspector	13	16	14/16	14/25	0.68	6	69	58/69	58/95	0.71

H-DEPT results are indicated in bold
w represents the motif length and $|A|$ the number of candidate motifs



(a) Real CRP **(b)** Predicted CRP

(c) Real E2F **(d)** Predicted E2F

(e) Real ERE **(f)** Predicted ERE

Fig. 5 Sequence logos for the real binding sites of CRP, E2F, and ERE (*left*) compared with the binding sites predicted by H-DEPT (*right*), created using WebLogo [8]

located the best biological solution of each execution and each instance. Among all the solutions discovered in each instance, we have selected the best, the worst, and the solution corresponding to the median solution and included the information in Table 10. As we can observe, the algorithm is able to make good biological predictions, achieving high accuracy (even 100 % in some cases), properly locating most of the real binding sites. This causes good F_{scores} that overcome in most cases the quality of the predictions made by the other biological tools. Among all results we highlight the one obtained in the MEF2 instance, where the algorithm finds all binding sites in more than half of the executions. In short, the results presented in this section demonstrate the biological accuracy of the designed hybrid multiobjective evolutionary algorithm, being able to conclude that H-DEPT represents a good tool for discovering motifs (binding sites) in DNA sequences.

Finally, as an example, in Fig. 5 we use the WebLogo software [8] to show the sequence logos of the best solutions predicted by H-DEPT in the CRP, E2F, and ERE instances. As we can see, the solutions predicted by our algorithm are very consistent with the real motifs, accurately locating all nucleotides that compose the solutions in most cases. These representations support the conclusions drawn in the previous comparison and demonstrate the biological accuracy presented by the proposed algorithm.

6 Conclusions and future work

In this work we study in detail the parallelization capability of a hybrid multiobjective evolutionary algorithm named hybrid differential evolution with Pareto tournaments (H-DEPT). After detailing the designed multiobjective adaptation, and explaining

the operation of the local search function developed for improving the resolution of an important biological problem, the motif discovery problem (MDP), we presented the results obtained after conducting several studies. These studies aim to analyze the parallelization capability and accuracy of the algorithm when it predicts motifs in DNA sequences. To parallelize and accelerate the execution of the proposed algorithm we have used OpenMP and, as differences exist in the execution times of the parallel tasks, we have analyzed the behavior of the algorithm using different thread scheduling policies. These parallel experiments have been carried out on several multicore systems (2, 4, 8, 16, and 32 cores). The obtained results show that any scheduling policy is good for multicore systems composed of two or four cores; dynamic and guided are the best for the eight-core multicore system; and finally, the dynamic scheduling policy achieves the greatest accelerations in multicore systems composed of many cores (16 and 32 cores). In addition to these conclusions and thanks to the presented parallelization, we are able to achieve efficiencies around 91 % for 2 cores, 90 % for 4 cores, 86 % for 8 cores, 79 % for 16 cores, and 66 % for 32 cores.

In the second study, we measure the biological quality of the solutions predicted by the algorithm. For doing this, we use several biological indicators that allow us to measure the correctness of the solutions. The obtained results demonstrate that the H-DEPT algorithm is able to find quality solutions that overcome those predicted by other well-known biological tools presented in the literature such as GAME, MEME, BioProspector, and BioOptimizer.

As future work, we intend to study the parallel behavior of other multiobjective algorithms for this specific problem, and thus compare their results with those presented by the parallel H-DEPT algorithm. Another interesting line of future work is to address the MDP by applying other parallel methodologies such as MPI or CUDA, and thus we would know which one is most suitable for its resolution. Finally, we will also improve the multiobjective problem definition to address in an even more realistic way the MDP.

Acknowledgments This work was partially funded by the Spanish Ministry of Economy and Competitiveness and the ERDF (European Regional Development Fund), under the contract TIN2012-30685 (BIO project). David L. González-Álvarez and Álvaro Rubio-Largo are supported by the postdoc research grant ACCION-III-15 and ACCION-III-13 from the University of Extremadura.

References

1. Bailey TL, Elkan C (1995) Unsupervised learning of multiple motifs in biopolymers using expectation maximization. Mach Learn 21(1–2):51–80
2. Baldwin N, Collins R, Langston M, Symons C, Leuze M, Voy B (2004) High performance computational tools for motif discovery. In: Proceedings of 18th International parallel and distributed processing symposium (IPDPS'04) pp 1–8
3. Blanco E, Farre D, Alba M, Messenguer X, Guigo R (2006) ABS: a database of annotated regulatory binding sites from orthologous promoters. Nucleic Acids Res 34:D63–D67
4. Challa S, Thulasiraman P (2008) Protein sequence motif discovery on distributed supercomputer. Adv Grid Pervasive Comput LNCS 5036:232–243
5. Chapman B, Jost R, van der Pas R (2007) Using OpenMP. MIT Press, Boston
6. Che D, Song Y, Rashedd K (2005) MDGA: motif discovery using a genetic algorithm. Proceedings of the 2005 Conference on genetic and evolutionary computation (GECCO'05), pp 447–452

7. Chen C, Schmidt B, Weiguo L, Müller-Wittig W (2008) GPU-MEME: using graphics hardware to accelerate motif finding in DNA sequences. Pattern Recogn Bioinf LNCS 5265:448–459
8. Crooks GE, Hon G, Chandonia JM, Brenner SE (2004) WebLogo: a sequence logo generator. Genome Res 14(6):1188–1190
9. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
10. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc 39(1):1–38
11. D'haeseleer P (2006) What are DNA sequence motifs? Nat Biotechnol 24(4):423–425
12. Fogel GB, Porto VW, Varga G, Dow ER, Crave AM, Powers DM, Harlow HB, Su EW, Onyia JE, Su C (2008) Evolutionary computation for discovery of composite transcription factor binding sites. Nucleic Acids Res 36(21):e142, 1–14
13. Fogel GB, Weekes DG, Varga G, Dow ER, Harlow HB, Onyia JE, Su C (2004) Discovery of sequence motifs related to coexpression of genes using evolutionary computation. Nucleic Acids Res 32(13):3826–3835
14. González-Álvarez DL, Vega-Rodríguez MA (2013) Parallelizing a hybrid multiobjective differential evolution for identifying cis-regulatory elements. In: Proceedings of the 20th European MPI Users's Group Meeting (EuroMPI 2013), pp 223–228
15. González-Álvarez DL, Vega-Rodríguez MA, Gómez-Pulido JA, Sánchez-Pérez JM (2010) Solving the motif discovery problem by using differential evolution with pareto tournaments. In: IEEE Congress on Evolutionary Computation (CEC'10), pp 4140–4147
16. González-Álvarez DL, Vega-Rodríguez MA, Gómez-Pulido JA, Sánchez-Pérez JM (2012) Predicting DNA motifs by using evolutionary multiobjective optimization. IEEE Trans Syst Man Cybernet Part C: Appl Rev 42(6):913–925
17. Grama A, Gupta A, Karypis G, Kumar V (2003) Introduction to parallel computing, 2nd edn. Pearson Education Limited, Edinburgh
18. Grundy W, Bailey T, Elkan C (1996) ParaMEME: a parallel implementation and a web interface for a dna and protein motif discovery tool. Computer Appl Biosci 12(4):303–310
19. Hertz GZ, Stormo GD (1999) Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics 15(7–8):563–577
20. Jensen S, Liu J (2004) BioOptimizer: a Bayesian scoring function approach to motif discovery. Bioinformatics 20:1557–1564
21. Kaya M (2009) MOGAMOD: multi-objective genetic algorithm for motif discovery. Expert Syst Appl 36(2):1039–1047
22. Kel A, Kel-Margoulis O, Farnham P, Bartley S, Wingender E, Zhang M (2001) Computer-assisted identification of cell cycle-related genes: new targets for E2F transcription factors. J Mol Biol 309(1):99–120
23. Klinge C (2001) Estrogen receptor interaction with estrogen response elements. Nucleic Acids Res 29(14):2905–2919
24. Li M, Ma B, Wang L (2002) Finding similar regions in many sequences. J Computer Syst Sci 65(1):73–96
25. Li L (2009) GADEM: a genetic algorithm guided formation of spaced dyads coupled with an EM algorithm for motif discovery. J Comput Biol 16(2):317–329
26. Liu FFM, Tsai JJP, Chen RM, Chen SN, Shih SH (2004) FMGA: finding motifs by genetic algorithm. In: Fourth IEEE Symposium on bioinformatics and bioengineering (BIBE'04), pp 459–466
27. Liu J, Neuwald A, Lawrence C (1995) Bayesian models for multiple local sequence alignment and gibbs sampling strategies. J Am Stat Assoc 90(432):1156–1170
28. Liu X, Brutlag D, Liu J (2001) Bioprospector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In: Pacific Symposium on biocomputing, pp 127–138
29. Liu X, Brutlag D, Liu J (2002) An algorithm for finding protein-DNA interaction sites with applications to chromatin immunoprecipitation microarray experiments. Nat Biotechnol 20:835–839
30. Liu Y, Schmidt B, Liu W, Maskell D (2010) CUDA-MEME: accelerating motif discovery in biological sequences using CUDA-enabled graphics processing units. Pattern Recogn Lett 31(14):2170–2177
31. Liu Y, Schmidt B, Maskell D (2011) An ultrafast scalable many-core motif discovery algorithm for multiple GPUs. In: IEEE International Symposium on parallel and distributed processing Workshops and Phd Forum, pp 428–434
32. Mak T, Lam K (2004) Embedded computation of maximum-likelihood phylogeny inference using platform FPGA. In: IEEE computational systems bioinformatics conference, pp 512–514

33. Neuwald AF, Liu JS, Lawrence CE (1995) Gibbs motif sampling: detection of bacterial outer membrane protein repeats. Protein Sci 4(8):1618–1632
34. Notredame C, Higgins DG (1996) SAGA: sequence alignment by genetic algorithm. Nucleic Acids Res 24(8):1515–1524
35. Oliver T, Schmidt B, Nathan D, Clemens R, Maskell D (2005) Using reconfigurable hardware to accelerate multiple sequence alignment with clustalw. Bioinformatics 21(16):3431–3432
36. Price K, Storn R, Lampinen J (2006) Differential evolution: a practical approach to global optimization. Springer, Inc. Secaucus, New york, NJ, USA
37. Qin J, Pinkenburg S, Rosenstiel W (2005) Parallel motif search using ParSEQ. In: Parallel and distributed computing and networks, pp 601–607
38. Roth FP, Hughes JD, Estep PW, Church GM (1998) Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. Nat Biotechnol 16(10):939–945
39. Rubio-Largo A, Vega-Rodríguez MA, González-Álvarez DL (2013) Designing a fine-grained parallel differential evolution with pareto tournaments for solving an optical networking problem. In: Concurrency and computation: practice and experience (online available), pp 1–27
40. Sandve G, Nedland M, Syrstad B, Eidsheim L, Abul O, Drabløs F (2006) Accelerating motif discovery: motif matching on parallel hardware. Alg Bioinf LNCS 4175:197–206
41. Schröder J, Wienbrandt L, Pfeiffer G, Schimmler M (2008) Massively parallelized DNA motif search on the reconfigurable hardware platform COPACOBANA. In: Proceedings of the Third IAPR International Conference on pattern recognition in bioinformatics, pp 436–447
42. Setubal J, Meidanis J (1997) Introduction to computational molecular biology. PWS Publishing Company, Boston
43. Shaw W, Burgin R, Howell P (1997) Performance standards and evaluations in ir test collections: cluster-based retrieval models. Inf Process Manag 33(1):1–14
44. Stine M, Dasgupta D, Mukatira S (2003) Motif discovery in upstream sequences of coordinately expressed genes. In: The 2003 Congress on evolutionary computation (CEC'03), pp 1596–1603
45. Stormo G (1988) Computer methods for analyzing sequence recognition of nucleic acids. Annu Rev Biophys Biophys Chem 17(1):241–263
46. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359
47. Sutou T, Tamura K, Mori Y, Kitakami H (2003) Design and implementation of parallel modified prefixspan method. Int Symp High Perform Comput 2858:412–422
48. Wei Z, Jensen S (2006) GAME: detecting cis-regulatory elements using a genetic algorithm. Bioinformatics 22(13):1577–1584
49. Weicker N, Szabo G, Weicker K, Widmayer P (2003) Evolutionay multiobjective optimization for base station transmitter placement with frequency assignment. IEEE Trans Evol Comput 7(2):189–203
50. Yamaguchi Y, Miyajima Y, Maruyama T, Konagaya A (2002) High speed homology search using runtime reconfiguration. Field Progr Logic Appl: Reconfig Comput Going Mainstream LNCS 2438:671–687
51. Yu L, Xu Y (2009) A parallel gibbs sampling algorithm for motif finding on gpu. In: IEEE International Symposium on parallel and distributed processing with applications, pp 555–558
52. Zaharie D (2009) Influence of crossover on the behavior of differential evolution algorithms. Appl Soft Comput 9(3):1126–1138
53. Zare-Mirakabad F, Ahrabian H, Sadeghi M, Hashemifar S, Nowzari-Dalini A, Goli-aei B (2009) Genetic algorithm for dyad pattern finding in DNA sequences. Genes Genet Syst 84(1):81–93