

RESEARCH ARTICLE

Open Access



DDmap: a MATLAB package for the double digest problem using multiple genetic operators

Licheng Wang¹ , Jingwen Suo¹, Yun Pan^{2*} and Lixiang Li¹

Abstract

Background: In computational biology, the physical mapping of DNA is a key problem. We know that the double digest problem (DDP) is NP-complete. Many algorithms have been proposed for solving the DDP, although it is still far from being resolved.

Results: We present DDmap, an open-source MATLAB package for solving the DDP, based on a newly designed genetic algorithm that combines six genetic operators in searching for optimal solutions. We test the performance of DDmap by using a typical DDP dataset, and we depict exact solutions to these DDP instances in an explicit manner. In addition, we propose an approximate method for solving some hard DDP scenarios via a scaling-rounding-adjusting process.

Conclusions: For typical DDP test instances, DDmap finds exact solutions within approximately 1 s. Based on our simulations on 1000 random DDP instances by using DDmap, we find that the maximum length of the combining fragments has observable effects towards genetic algorithms for solving the DDP problem. In addition, a Maple source code for illustrating DDP solutions as nested pie charts is also included.

Background

The physical mapping of DNA is a key problem in computational biology [5]. A large DNA molecule is a long string composed of four nucleotides, A, C, G and T. To understand the structure of DNA molecules, it is of interest to determine the occurrences of short substrings, such as GAATTC, on the DNA. Double digest experiments (DDE for short) are a standard approach for constructing physical DNA maps [2]. Given two restriction enzymes, denoted by \mathcal{A} and \mathcal{B} , this approach cuts a target DNA sequence by using only enzyme \mathcal{A} , only enzyme \mathcal{B} , and both enzymes simultaneously, in three separate and parallel experiments [5]. As a result, we obtain three multisets of short DNA fragments. However, due to certain experimental limitations, only the length information (i.e., The number of nucleotides) of these short fragments can be measured with certain accuracy using certain mature biological techniques, such as gel

electrophoresis. The objective of the double digest problem (DDP) is to reconstruct the original ordering of the fragments in the target DNA molecule.

Since the first successful reconstruction of restriction site mapping in the earlier 1970s [7, 11], the DDP problem has become an intensively studied issue that covers a variety of disciplines [6, 9]. Although the major concerns come from the community of bioinformation, the challenges related to this problem have also attracted attention from the artificial intelligence, algorithmic complexity, and optimization communities. We now know that DDP is strongly NP-complete [1, 2], and many algorithms have been proposed for solving the DDP problem [3–6, 8–10, 12–15]. However, the DDP problem is still far from being resolved. All of the algorithms developed to address this problem have encountered significant difficulties as the number of restriction sites increases. Moreover, even for different DDP instances with the same size, the hardness for finding an exact solution might vary remarkably.

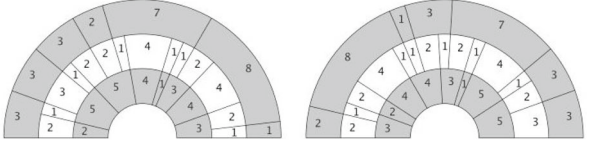
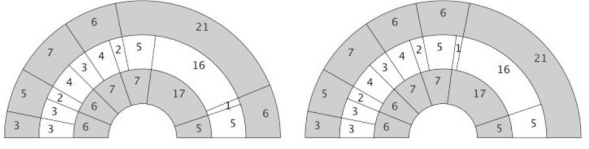
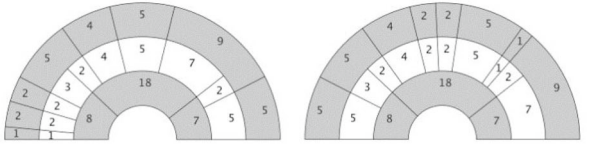
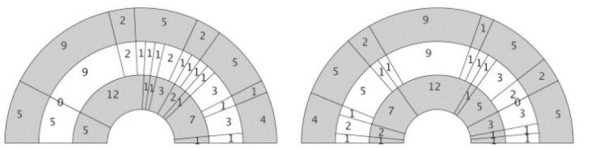
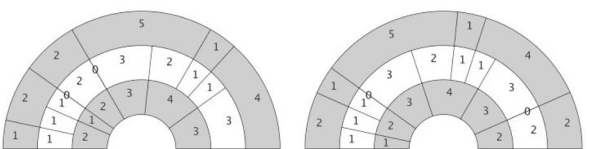
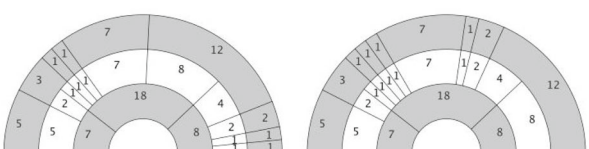
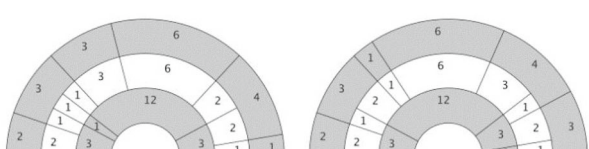
The main motivation of this work comes from three considerations: First, almost all existing formulations of

* Correspondence: pany@cuc.edu.cn

²School of Computer Science, Communication University of China, 1 East of Dingfuzhuang Street, Chaoyang District, Beijing 100024, China
Full list of author information is available at the end of the article



Table 1 Main results: separated and integrated effects of all six genetic operators. Instance 1,3,4,5,7,8 come from [13], instance 2' is derived by using a scaling-rounding-adjusting process towards instance 2, m, n, k are the lengths of the input fragments A, B and C, respectively. There are six genetic operators, RWS is selection operator defined as the well-known roulette wheel algorithm. PCC and RSC are crossing operators, PCC is the combination of two permutations, RSC is Referencing Sorting Crossing, P4X, FLP, CSH are mutation operators, P4X is a four-point mutating, FLP defined as the flipping of the given fragment. CSH defined as the cyclic shifting of the given fragment. The average running time, average evolution generations and success rate are listed in the table. At the right of the table, we draw pie charts of DDmap's two solutions.

Instances		Genetic Operators	AvgTime (seconds)	avgEv oGen	SuccRate (%)	Pie charts of two solutions
No	(m,n,k)					
1	(8,7,14)	RWS + PCC	0.0812	3.41	100	
		RWS + RSC	0.1263	3.70	100	
		RWS + P4X	0.1533	5.75	100	
		RWS + FLP	0.1483	5.40	100	
		RWS + CSH	0.1041	3.99	100	
		RWS + ALL	0.1149	2.91	100	
2'	(6,6,11)	RWS + PCC	1.0865	40.299	100	
		RWS + RSC	1.5291	43.67	100	
		RWS + P4X	1.4401	52.05	100	
		RWS + FLP	1.9856	70.09	100	
		RWS + CSH	1.6777	59.43	100	
		RWS + ALL	1.4489	33.97	100	
3	(3,8,10)	RWS + PCC	0.0159	1.21	100	
		RWS + RSC	0.0201	1.20	100	
		RWS + P4X	0.0251	1.48	100	
		RWS + FLP	0.0181	1.24	100	
		RWS + CSH	0.0295	1.64	100	
		RWS + ALL	0.0182	1.15	100	
4	(9,8,15)	RWS + PCC	0.3041	10.72	100	
		RWS + RSC	0.6469	13.95	100	
		RWS + P4X	0.5760	19.81	100	
		RWS + FLP	0.6411	21.27	100	
		RWS + CSH	0.5906	19.67	100	
		RWS + ALL	0.5297	9.67	100	
5	(6,6,9)	RWS + PCC	0.0110	1.02	100	
		RWS + RSC	0.0104	1.00	100	
		RWS + P4X	0.0103	1.00	100	
		RWS + FLP	0.0103	1.00	100	
		RWS + CSH	0.0107	1.00	100	
		RWS + ALL	0.0110	1.01	100	
7	(3,9,11)	RWS + PCC	0.0510	2.46	100	
		RWS + RSC	0.1056	3.31	100	
		RWS + P4X	0.1343	5.14	100	
		RWS + FLP	0.1191	4.62	100	
		RWS + CSH	0.1288	4.93	100	
		RWS + ALL	0.0902	2.48	100	
8	(4,6,9)	RWS + PCC	0.0116	1.09	100	
		RWS + RSC	0.0194	1.22	100	
		RWS + P4X	0.0177	1.19	100	
		RWS + FLP	0.0184	1.28	100	
		RWS + CSH	0.0159	1.19	100	
		RWS + ALL	0.0168	1.13	100	

the DDP problem use multiset as the basic data structure, while we find that it is even easier to model the DDP problem by using vectors. Second, some recently proposed genetic algorithms [3, 13] for addressing the DDP problem should be improved. Third, it is of interest to develop an open-source package for studying the DDP problem by using easily accessible engineering computation platforms, such as MATLAB.

Our main contributions are summarized as follows:

- A vector-based formulation of the DDP problem is presented and illustrated step-by-step.
- A novel genetic algorithm for solving the DDP problem is proposed by combining six genetic operators, and a MATLAB package, DDmap, is implemented by integrating the proposed genetic algorithm and other necessary supporting and testing widgets. Then, by using DDmap, exact solutions for typical DDP test instances [13] are explicitly derived and depicted. (See the right column of Table 1.)
- A relation between the hardness of certain DDP instances and the maximum length of double digest sequences is revealed based on our simulations of 1000 random DDP instances. Meanwhile, an approximate approach for typical hard DDP instances is conceived based on this relation.

Results

To test the utility of DDmap, eight DDP instances, referred to as $INS_j (j = 1 \cdots 8)$, are taken from [13]. They are shown in the following Table 2:

First, the integrated effects of the six aforementioned genetic operators of DDmap are verified. For the instances $INS_{1, 3, 4, 5, 7, 8}$, DDmap performs considerably well, and the related results are collected in Table 1. For *each* instance, 100 trails were run using DDmap with respect to *each* combination of six genetic operators. Then, the average running time, the average evolution generation and the success rate of finding exact DDP solutions are counted. Two different exact solutions for the instances $INS_{1, 3, 4, 5, 7, 8}$ are also depicted in the right column of Table 1. In addition, the average running time and the average evolution generations of finding exact DDP solutions are depicted in Fig. 1. From Table 1 and Fig. 1. We can see that the genetic operators combination of RWS + PCC performs best in running time, RWS + ALL performs best in evolving generation, while other combinations of different genetic operators perform similarly and equally effective. Moreover, the tendency of running time curve and evolving generation curve are very similar.

However, we find that DDmap performs very poorly for INS_2 and INS_6 . Upon further examination, we find

Table 2 Test instances from [13]. Suppose giving two restriction enzymes, denoted by \mathfrak{A} and \mathfrak{B} , \vec{a} , \vec{b} , \vec{c} are the multisets of short DNA fragments by cuts a target DNA sequence by using enzyme \mathfrak{A} only, enzyme \mathfrak{B} only, and both enzymes simultaneously.

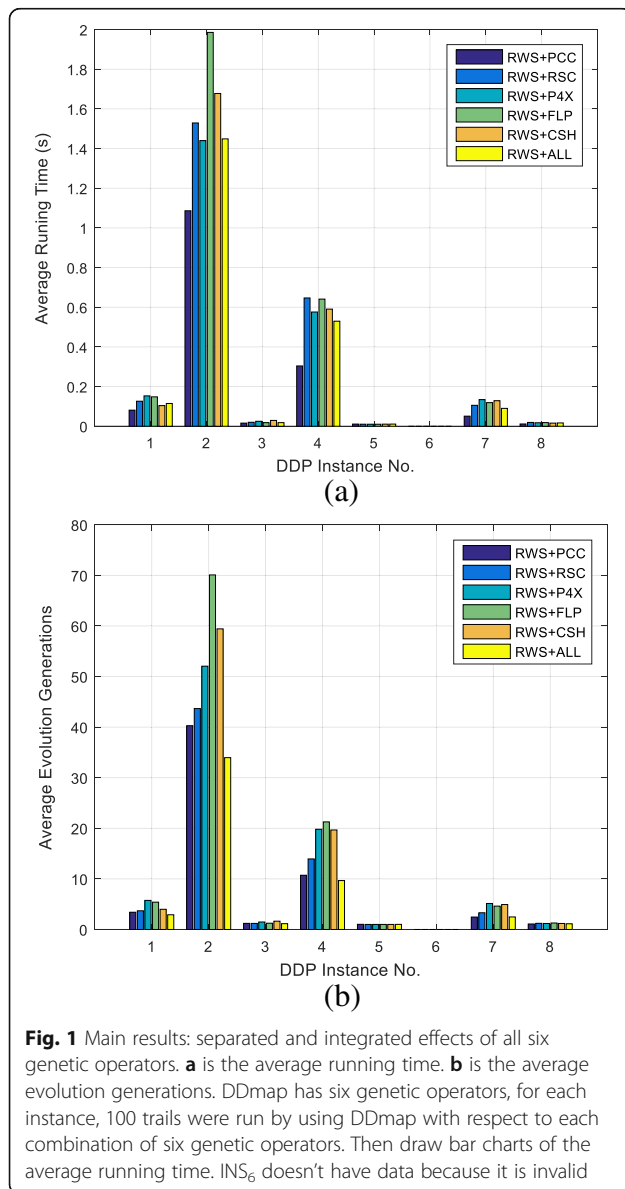
No.	\vec{a}	\vec{b}	\vec{c}
1	1,2,3,3,4, 4,5,5	1,2,3,3,3, 7,8	1,1,1,1,1,1,2,2, 2,2,2,3,4,4
2	5509,562 6,6527,6 766,7233 ,16841	3526,487 8,5643,58 04,7421,2 1230	1120,1868,256 4,2752,3240,3 526,3758,3775 ,4669,5509,15 721
3	7,8,18	1,2,2,4,5, 5,5,9	1,2,2,2,2,3,4,5, 5,7
4	1,1,1,1,2, 3,5,7,12	1,2,2,4,5, 5,5,9	1,1,1,1,1,1,1,1, 1,2,2,3,3,5,9
5	1,2,2,3,3, 4	1,1,2,2,4, 5	1,1,1,1,1,2,2,3, 3
6	1,2,3,4,5, 6,7,8,9	15,15,15	1,1,1,1,1,2,2,2, 3,6
7	7,8,18	1,1,1,1,2, 3,5,7,12	1,1,1,1,1,2,2,4, 5,7,8
8	1,3,3,12	1,2,3,3,4, 6	1,1,1,1,2,2,2,3, 6

that INS_6 is *invalid*. Simple calculation shows that as for INS_6 , we have

$$45 = \sum(\vec{a}) = \sum(\vec{b}) \neq \sum(\vec{c}) = 19$$

because it violates the restriction condition of (5) (See Definition 1).

For INS_2 , we run DDmap 100 trails and successfully obtain exact solutions of INS_2 in 67 trails. But the average running time and evolution generations for reaching the exact solution of INS_2 are 122 s and 3828, respectively, i.e., approximately 1000 times slower than the results of other test instances (see Table 1). Furthermore, we find that these 67 solutions are essentially the same: One solution is depicted in Fig. 2(a), and another solution is just to read out the sequences A, B and C of Fig. 2(a) in an reverse order. It seems that the solution to INS_2 's solutions are very sparse, and thus, DDmap faces the difficulty of escaping from so many local optima.



We deal with the INS_2 by using the scaling-rounding-adjusting approach. As expected, DDmap can find solutions towards INS_2 very efficiently. For each combination of six genetic operators, we run DDmap towards INS_2 100 trials. The average running time is no more than 2 s, the evolution generation is no more than 80, and the success rate for finding exact DDP solutions is always 100%. The results are already contained in Table 1 and Fig. 1. Now, we directly take some INS_2 's solution, $(\mu, \nu) \in S_m \times S_m$, as an approximate solution of INS_2 . The resulted double digest pie charts are depicted in Fig. 2(b). Compared to the exact solution given in Fig. 2(a), we think this kind of approximation is an interesting result in the sense that the relative error, defined as the proportion of total length

of gaps between two miss-aligned fragments, is merely 4.8%, calculated by

$$\frac{115 + 17 + 256 + 171 + 117 + 188 + 280 + 1120}{48502} = 0.0487.$$

Next, via a number of simulations, we find that DDmap's performance is tightly related to the maximum length of a piece in the sequence of C , denoted by $\rho_C = \max c_i$. This is reasonable considering that for a fixed length of sequence C , denoted by $L_C = |C|$, the smaller ρ_C is, the denser the solutions, and thus, the easier for genetic algorithms, such as DDmap, to meet an exact solution during the evolution process. Based on our simulations towards 1000 random DDP instances with different ρ_C , the relationship between the success rate of finding exact DDP solutions with respect to ρ_C is depicted in Fig. 3.

Discussion

♦ Cases of $k \neq m + n - 1$

Note that in both INS_4 and INS_5 , the given two enzymes cut the target DNA molecule at some of the same sites and lead to the case where $k \neq m + n - 1$. At the beginning, DDmap performs very poorly on INS_4 and INS_5 . The performance of DDmap on INS_4 and INS_5 improves remarkably after we adopt the following simple preprocessing strategy:

- If $k < m + n - 1$, then introduce $\delta = (m + n - 1) - k$ fragments with length 0 into the sequence \vec{c} ;
 - Otherwise, if $k > m + n - 1$, then introduce $\delta = k - (m + n - 1)$ fragments with length 0.
- into the shorter sequence among \vec{a} and \vec{b} ;
- Otherwise, do nothing.

An interesting observation is that the newly introduced 0-length fragments will explicitly appear in the pie charts of exact DDP solutions. For instance, Fig. 4(a) shows that a 0-length fragment in sequence \vec{c} of INS_4 appears at the fifteenth site, while Fig. 4(b) shows that two 0-length fragments in sequence \vec{c} of INS_5 appear at the sixth and eighth sites, respectively.

Here, we follow the convention of reading a pie chart from 0° to 180° or 360° .

♦ Comparison

Figure 5(a) and (b) are the comparison of the average running time between DDmap and the algorithm in 2005 [13] and 2012 [3]. Operator 1–5 are the crossover and mutation operator in DDmap. Because the crossover

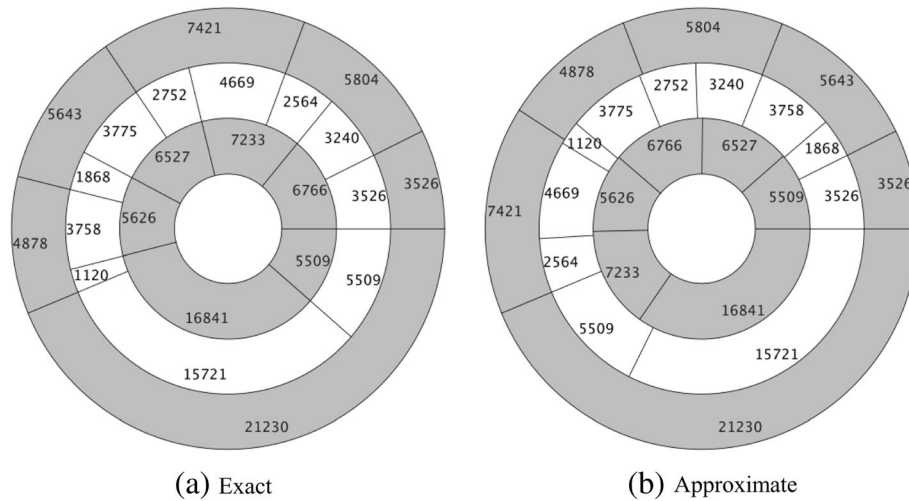


Fig. 2 Effects of scaling-rounding-adjusting method. **a** is an exact solution of INS_2 . **b** is an approximate solution of INS_2 , derived by using the scaling-rounding-adjusting process towards INS_2

operator in [13] is the same as our operator 2 and the two mutation operators in [3] are similar to our operators op4 and op5, so we only implement the mutation operator op6 in [13] and crossover operator op7 in [3]. Eight instances are from the paper [13]. In the comparison experiment, each instance is run 100 times for operators op1–7 respectively, and then we got the average running time and the success rate of finding the exact DDP solution.

Through the experimental data, we found the data of op6 is much larger than that of the other six operators, the data of the other six operators will be neglected in the rectangular coordinate system, so we choose the logarithmic coordinate system. Figure 5(a) is the comparison between DDmap and the algorithm in 2005 [13], the blue line is the average running time of op6, it is higher than the other six lines, our

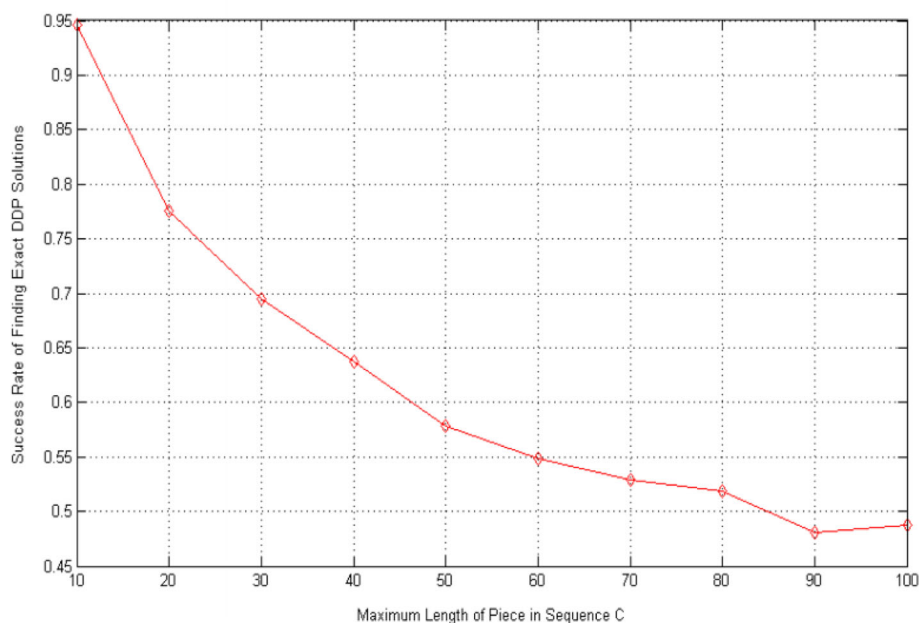
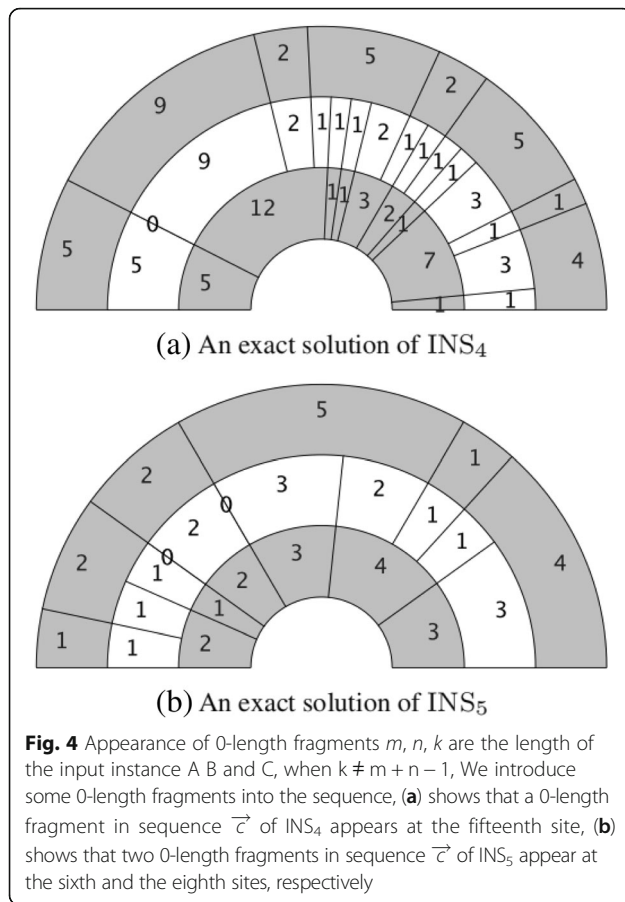


Fig. 3 Success Rate vs. Maximum Length of Piece in C. DDmap's performance is tightly related to the maximum length of piece in C, we generated a series of random double digest instances with the maximum length of C ranging from 10 to 100, then test the DDmap's success rate, the line of success rate changing with the maximum length of C is shown in Fig. 3

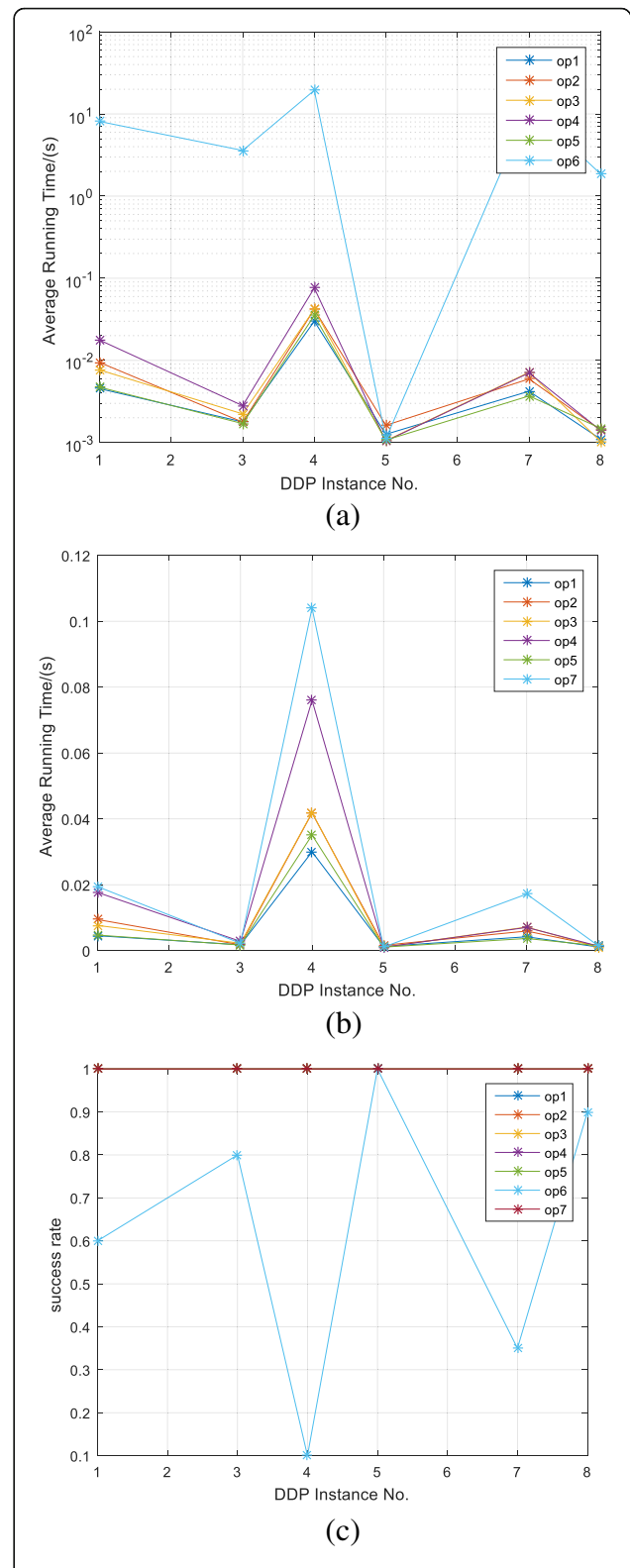


algorithm has a significant time advantage over the [3]'s algorithm. As can be seen from Fig. 5(b), the six lines have little difference, however, the op7's line is always at the top, so our algorithm has a slight advantage over that of [3].

The comparison of success rate is shown in Fig. 5(c). The success rate of operators 1, 2, 3, 4, 5, 7 is 100%, they are all effective for these instances. Operator 6 runs very irregularly and the results are not very good.

Instance 2 and 6 does not appear in Fig. 5. In fact, INS_6 is invalid. As aforementioned, INS_2 is very complex, so we analyze it separately. To reset the maximum evolution generation as large as 100,000, running each operator 10 times towards INS_2 , the average running time and the success rate is shown in Fig. 6(a) and (b), respectively. We can see that the running time of op6 is about 10 times longer than other operators, while the running time of op7 is about twice longer than our operators op1–5. The success rates of our five operators are all 100%, however, op7's success rate is 90%, but op6 does not produce the exact DDP solution.

In conclusion, DDmap is much better than the algorithm in [13] and it is slightly better than [3]'s algorithm.



(See figure on previous page.)

Fig. 5 Comparison of DDmap and algorithm in [3, 13]. Operators 1–5 are the crossover and mutation operators in DDmap, op6 is the mutation operator in [13] and op7 is the crossover operator in [3]. Each instance is run 100 times by using op1–7 respectively. **a** is a logarithmic coordinate system figure, we can see the average running time comparison between DDmap and the algorithm in [13] in (a). **b** is the average running time comparison between DDmap and the algorithm in [3]. **c** is the success rate comparison between DDmap and the algorithm in [3, 13]

Conclusions

An open-source MATLAB package DDmap based on a newly designed genetic algorithm that combines six genetic operators is designed for solving the double digest problem. This algorithm finds exact solutions within approximately 1 s for typical DDP test instances. For some hard DDP instances, DDmap performs very well via a

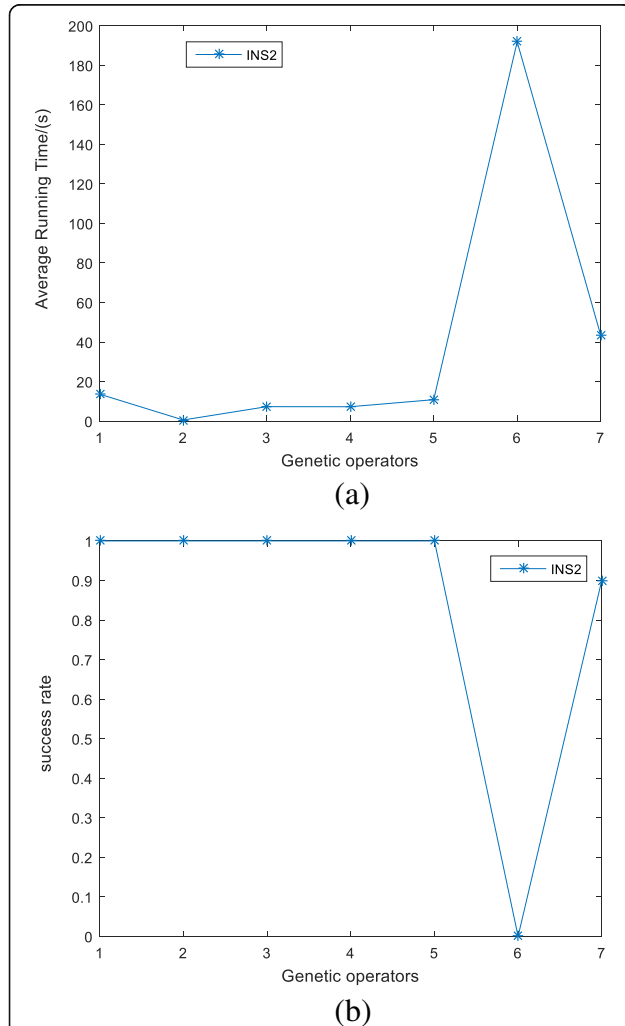


Fig. 6 Comparison of DDmap and algorithm in [3, 13] under the condition of INS₂. The maximum evolution generation is set to 100,000, running each operator 10 times, (a) is the average running time of each operator. (b) is the success rate of each operator

scaling-rounding-adjusting process. The experimental results of our algorithm confirm its efficiency.

Methods

Problem formulation

Let S_m denote the symmetric group on m indices $\{1, 2, \dots, m\}$. Then, for a given permutation $\pi \in S_m$ and a given vector $\vec{a} = (a_1, \dots, a_m)$, the action of π on \vec{a} derives a vector $\vec{a}^\pi = (a_{\pi(1)}, \dots, a_{\pi(m)})$, reassembling of the order of entries of \vec{a} according to π . Further, let us define the *accumulative sum vector* of \vec{a} , denoted by $AS(\vec{a})$, and the *step difference vector* of \vec{a} , denoted by $SD(\vec{a})$, as follows:

$$AS(\vec{a}) = (\Sigma(\vec{a}, 1), \dots, \Sigma(\vec{a}, m)) \quad (1)$$

and

$$SD(\vec{a}) = (\Sigma(\vec{a}, 1), \Sigma(\vec{a}, 2) - \Sigma(\vec{a}, 1), \dots, \Sigma(\vec{a}, m) - \Sigma(\vec{a}, m-1)) \quad (2)$$

where $\Sigma(\vec{a}, j) = \sum_{i=1}^j a_i (j = 1, \dots, m)$ indicates the partial sum of \vec{a} .

Now, the double digest problem (DDP) can be formulated by the following steps:

- Given two vectors $\vec{a} = (a_1, \dots, a_m)$ and $\vec{b} = (b_1, \dots, b_n)$ with the restriction $\Sigma(\vec{a}, m) = \Sigma(\vec{b}, n)$, we define the combining sequence of \vec{a} and \vec{b} , denoted by $\Pi(\vec{a}, \vec{b})$, as the concatenation of vectors $AS(\vec{a})$ and $AS(\vec{b})$ and removing the tail entry. That is,

$$\Pi(\vec{a}, \vec{b}) = (AS(\vec{a})_1, \dots, AS(\vec{a})_m, AS(\vec{b})_1, \dots, AS(\vec{b})_{n-1}) \quad (3)$$

- The sequence $\Pi(\vec{a}, \vec{b})$ can be reassembled to obtain a new sequence according to the nondecreasing order, denoted by $\hat{\Pi}(\vec{a}, \vec{b})$.
- The double digest sequence of \vec{a} and \vec{b} , denoted by $DDS(\vec{a}, \vec{b})$, can be defined as the step difference vector of $\hat{\Pi}(\vec{a}, \vec{b})$. That is,

$$DDS(\vec{a}, \vec{b}) = SD(\hat{\Pi}(\vec{a}, \vec{b})) \quad (4)$$

- Now, we introduce the following definition:

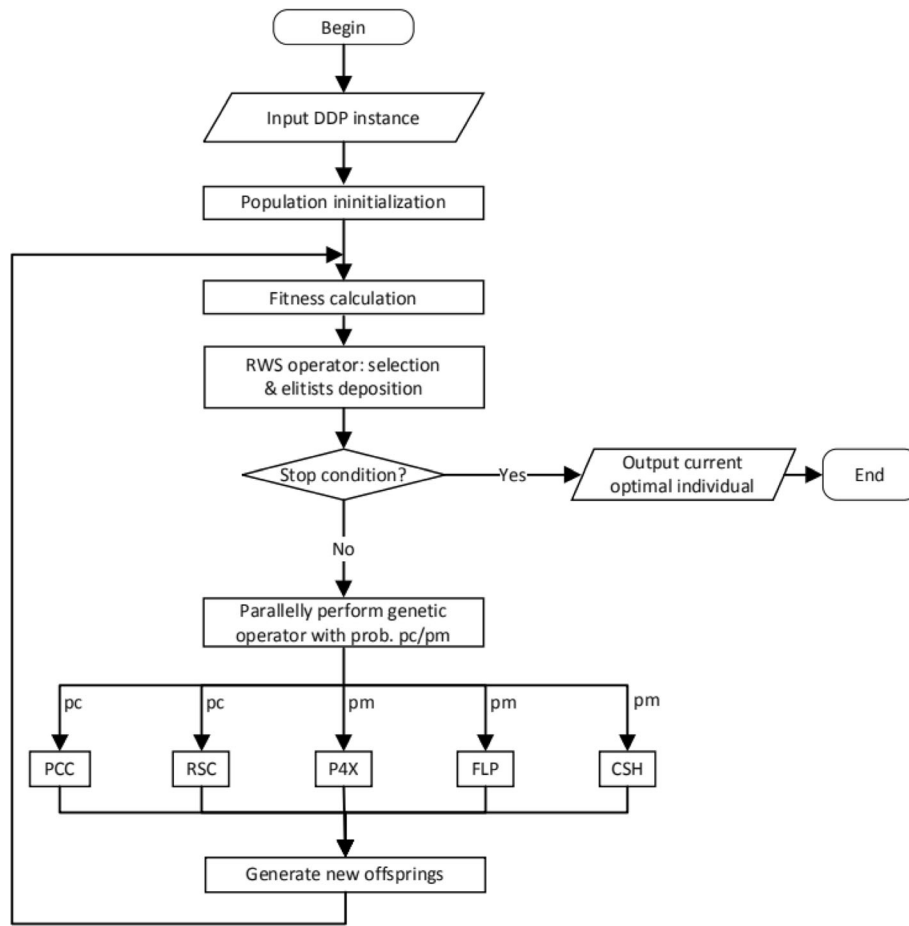


Fig. 7 Flowchart of main GA algorithm of DDmap. The input DDP instance includes the instances in [13] and random instances, after calculating the fitness value, if not satisfied the stop condition, the crossover and mutation operators will be performed probabilistically, then generate new offsprings and recalculate the fitness values

Definition 1

A double digest problem (DDP) instance is specified by three vectors $\vec{a} = (a_1, \dots, a_m)$, $\vec{b} = (b_1, \dots, b_n)$ and $\vec{c} = (c_1, \dots, c_k)$ with the restriction of

$$\Sigma(\vec{a}, m) = \Sigma(\vec{b}, n) = \Sigma(\vec{c}, k) \quad (5)$$

and the objective is to find a pair permutations $(\mu, \nu) \in S_m \times S_n$ such that.

$$DDS(\vec{a}^\mu, \vec{b}^\nu) = \vec{c}^\pi \text{ for some } \pi \in S_m \quad (6)$$

Remark 1

If two enzymes cut a target DNA molecule at disjoint sites, then we have the condition $k = m + n - 1$. It was previously suspected that this case might lead to easier reconstruction problems [2]. (However, our simulation does support this conjecture, and details are given in the supplementary part). However, due to some unavoidable

experimental errors, this condition does not always hold. Thus, in DDmap, we employ a very simple strategy to address the cases of $k = m + n - 1$: Introducing 0-length fragments in sequence A, B, or C if necessary. Our simulation results show that this method is considerably robust.

Remark 2

If we take into consideration possible partial cleavage errors, then the optimization goal (6) should be updated to

$$\min_{\mu \in S_m, \nu \in S_n} \left| \left| DDS(\vec{a}^\mu, \vec{b}^\nu) \oplus \vec{c} \right| \right| \quad (7)$$

where symbol \oplus indicates the set exclusive operation, and the two operands $DDS(\vec{a}^\mu, \vec{b}^\nu)$ and \vec{c} should be regarded as unordered multisets. By doing so, the searching space of the DDP solution is reduced to $S_m \times S_n$ instead of $S_m \times S_n \times S_k$. In fact, π can be easily extracted from any valid solution (μ, ν) . A simple method for obtaining π is

to at first sort $DDS(\vec{a}^\mu, \vec{b}^\nu)$ to obtain a nondecreasing sequence and then let π be the permutation specified by the reverse index of the sorting subscripts. Apparently, this step can be performed within the complexity $O(k \log k)$.

Example 1

For given three vectors $\vec{a} = (1, 2, 3, 5)$, $\vec{b} = (2, 2, 3, 4)$ and $\vec{c} = (1, 1, 1, 2, 2, 2)$ as well as two permutations

$$\mu = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix} \text{ and } \nu = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}, \text{ we can verify that}$$

(μ, ν) is a valid solution for the DDP instance specified by $(\vec{a}, \vec{b}, \vec{c})$. The pie charts of a solution and the corresponding calculation steps and complexities are depicted in Table 3.

The proposed genetic operators

Recall that the basic idea of a genetic algorithm consists of the following concepts: an individual is totally specified by a chromosome; a chromosome is the carrier of a gene, and the position of a gene in a chromosome is called a locus; the gene composition of an individual is called the genotype; and the fitness value, called phenotype, is the result of mutual effects of genotype and external environments. Thus, to design a genetic algorithm for a given optimization problem, we need to specify how to represent a chromosome, evaluate the fitness value, design genetic operators, and determine evolution strategies such as the population size, the maximum evolution generation, the elitism keeping method, the probabilities for each genetic operator, etc.

First, for a given DDP instance $(\vec{a}, \vec{b}, \vec{c})$, we directly use a random pair of permutations $(\mu, \nu) \in S_m \times S_n$ to represent a chromosome, and its fitness value is given by

$$f(\mu, \nu) = \frac{1}{1 + |DDS(\vec{a}^\mu, \vec{b}^\nu) \oplus \vec{c}|} \quad (8)$$

Second, the following 6 genetic operators are employed in this work:

- **RWS**. This is a natural selection operator defined as the well-known roulette wheel algorithm.
- **PCC**. This is a crossing operator defined as a combination of two permutations. Given two chromosomes $(\mu^{(1)}, \nu^{(1)})$ and $(\mu^{(2)}, \nu^{(2)})$, this operator produces two new offspring

$$(\mu^{(1)} \circ \mu^{(2)}, \nu^{(1)} \circ \nu^{(2)})$$

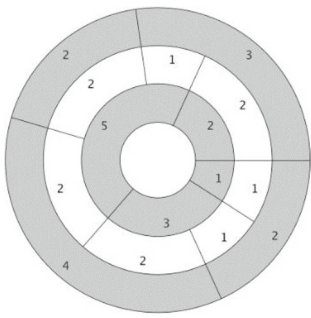
and

$$(\mu^{(2)} \circ \mu^{(1)}, \nu^{(2)} \circ \nu^{(1)})$$

respectively.

- **RSC**. This is a crossing operator defined as the so-called referencing sorting (RS). Given a target sequence \vec{a} and a reference sequence \vec{b} , assuming both are defined over the same alphabet. Then, during the sorting process, the swapping operation

Table 3 Illustration of the proposed formulation. This is the detailed process of solving the double digest problem, The calculation process of example 1 is listed in (a). The pie chart for this example's solution is in (b).

Calculation steps	Values	Complexities	Memo	
\vec{a}^μ	(2,5,3,1)	$O(m)$	Inner piechart	
\vec{b}^ν	(3,2,4,2)	$O(n)$	Outer piechart	
$AS(\vec{a}^\mu)$	(2,7,10,11)	$O(m)$		
$AS(\vec{b}^\nu)$	(3,5,9,11)	$O(n)$		
$\sqcup(\vec{a}^\mu, \vec{b}^\nu)$	(2,7,10,11,3,5,9)	$O(k)$	$k = m + n - 1$	
$\hat{\sqcup}(\vec{a}^\mu, \vec{b}^\nu)$	(2,3,4,7,9,10,11)	$O(k \log k)$		
$DDS(\vec{a}^\mu, \vec{b}^\nu)$	(2,1,2,2,2,1,1)	$O(k)$		
$ DDS(\vec{a}^\mu, \vec{b}^\nu) \oplus \vec{c} $	0	$O(k)$	Exact matched	
π	$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 1 & 5 & 6 & 7 & 2 & 3 \end{pmatrix}$	$O(k \log k)$		
\vec{c}^π	(2,1,2,2,2,1,1)	$O(k)$	Middle piechart	
(a) Calculation and verification				(b) Pie chart of a DDP solution

of two elements in \vec{a} is performed only if they are in the reverse order in the referencing sequence \vec{b} . RS is a generalization of ordinary sorting in the sense that any two elements can be compared even if they do not come from a complete order. RS is inspired by operator precedence grammars. More details about RS and RSC are given in the supplementary section. In fact, RSC is called *order preserving weighted crossover* in [13].

- **P4X**. This is a four-point mutating operator defined as follows: Given a chromosome (μ, ν) , randomly exchange two elements of μ and two elements of ν .
- **FLP**. This is a fragment mutating operator defined as flipping of the given fragment. By flipping a fragment (2, 5, 4, 1), we obtain (1, 4, 5, 2).
- **CSH**. This is a fragment mutating operator defined as cyclic shifting of the given fragment. By cyclically shifting a fragment (2, 5, 4, 1), we obtain (5, 4, 1, 2).

More details about the referenced sorting crossover (RSC) genetic operator.

RSC is in fact the order preserving weighted crossover given in [13]. Suppose two parent chromosomes are

$$p_1 = (1, 3, 2, 1, 3, 4, 2, 2) \text{ and}$$

$$p_2 = (1, 2, 2, 2, 4, 3, 3, 1),$$

and the crossover point is 3. Then, the producing of the offspring is given below:

- (1) p_1 is split into two pieces: $p_{11} = (1, 3, 2)$ and $p_{12} = (1, 3, 4, 2, 2)$, and p_2 is split into two pieces: $p_{21} = (1, 2, 2)$ and $p_{22} = (2, 4, 3, 3, 1)$.
- (2) The piece p_{12} is sorted by taking p_2 as the referenced sequence. Since in p_2 there exists a chain $2 - 2 - 4 - 3 - 1$ this leads to $p'_{12} = (2, 2, 4, 3, 1)$.
- (3) Similarly, p_{22} is sorted by taking p_1 as the referenced sequence. This time, we obtain $p'_{22} = (3, 1, 3, 4, 2)$ since there exists a chain $3 - 1 - 3 - 4 - 2$ in p_1 .
- (4) Two offspring chromosomes are

$$c_1 = p_{11} \parallel p'_{12} = (1, 3, 2, 2, 2, 4, 3, 1) \text{ and } c_2 = p_{21} \parallel p'_{22} = (1, 2, 2, 3, 1, 3, 4, 2).$$

Among the above 6 genetic operators, RWS is widely used in most genetic algorithms, and RSC was first used in [13] to solve the DDP problem. Four other genetic operators, although being easily conceived, are new to DDP-oriented genetic algorithms, as far as we know.

Third, the evolution strategies in this work refer to [13]. That is, the population size and maximum

evolution generation are set to 50 and 10,000, respectively. Elitists in each generation are kept, and the crossing probability is set to 0.85. The linearly adaptive mutation probability in [13] is also used in our work, but with a slight modification to ensure the cyclic increment of mutation probability is nonnegative. The details are as follows:

We follow the suggestion given in [13] by letting the mutation probability vary linearly in cycles of 200 iterations. However, in the original paper, this cycle varies from $\frac{2}{m+n}$ to 0.45, while in our work, the cycle varies from $\frac{2}{m+n}$ to 0.55, considering that in the case of $m = n = 2$, the start point would be 0.5, which is larger than 0.45.

Scaling-rounding-adjusting approach

Based on the above observation, we try to deal with the instance INS_2 in another way. A new test instance, INS_2 , is derived by using a scaling-rounding-adjusting process on INS_2 . The details of this process are as follows:

- **Scaling and rounding**. Because the minimum length of pieces in sequence \vec{c} of INS_2 is 1120, we take 0.001 as the scaling factor. That is, we multiply the sequences \vec{a} , \vec{b} , \vec{c} by 0.001 and then round them. By doing so, we obtain

$$\vec{a}' = (6, 6, 7, 7, 7, 17)$$

$$\vec{b}' = (4, 5, 6, 6, 7, 21)$$

$$\vec{c}' = (1, 2, 3, 3, 3, 4, 4, 4, 5, 6, 16)$$

- **Adjusting**. Next, we find that

$$\sum(\vec{a}') = 50 \neq \sum(\vec{b}') = 49 \neq \sum(\vec{c}') = 51$$

That is, $(\vec{a}', \vec{b}', \vec{c}')$ is an invalid DDP instance. Intuitively, this occurs because the round operation, $\text{round}(\cdot)$, introduces more errors. Thus, we try to adjust the rounding operation in the previous step according to the so-called rounding-up and rounding-down strategies:

- **Rounding-up**: $\text{round}(x)$ is replaced by $x' = \text{round}(x + 0.1)$, and we obtain

$$\vec{a}'' = (6, 6, 7, 7, 7, 17)$$

$$\vec{b}'' = (4, 5, 6, 6, 8, 21)$$

$$\vec{c}'' = (1, 2, 3, 3, 3, 4, 4, 4, 5, 6, 16)$$

This DDP instance is again invalid since

$$\sum(\vec{a}) = 50 = \sum(\vec{b}) \neq \sum(\vec{c}) = 51$$

- Rounding-down: $\text{round}(x)$ is replaced by $x' = \text{round}(x - 0.1)$

$$\vec{a}' = (5, 6, 6, 7, 7, 17)$$

$$\vec{b}' = (3, 5, 6, 6, 7, 21)$$

$$\vec{c}' = (1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 16)$$

Now, the DDP instance may be valid since

$$\sum(\vec{a}') = \sum(\vec{b}') = \sum(\vec{c}') = 48$$

Note that the constant 0.1 in rounding-up/rounding-down is a value defined by experience. A reasonable domain of this constant would be the interval [0.0001, 0.4999].

• Now, the newly derived DDP instance INS_2 , is given by the three vectors $(\vec{a}', \vec{b}', \vec{c}')$.

Finally, we would like to mention that all simulations in this work are conducted on a X1 Carbon laptop with Windows(TM) 8, Intel(R)Core(TM)i5 – 4300U CPU @ 1.90 GHz/2.49 GHz and 8GB RAM. The complete genetic algorithm for solving the DDP problem is implemented as a MATLAB package, DDmap, and a Maple source code for drawing DDP solutions as nested pie charts is also included in this package.

The package DDmap consists of

- 13 MATLAB algorithms:
- permGA.m, the MATLAB genetic algorithm for solving the DDP problem. This is the main algorithm, and its flowchart is depicted in Fig. 7. Note that this file also contains the definitions of five genetic operators — RWS, PCC, P4X, FLP, CSH and related MATLAB functions for calculating the fitness values.
- referIndexSort.m, the MATLAB algorithm for implementing the so-called referenced sorting (based on index).
- opPermCross.m, the MATLAB algorithm for implementing the RSC genetic operator.
- getInstance.m, the auxiliary MATLAB algorithm for outputting test DDP instances in [Sur-Kolay S. et al., 2005].
- randDDPinstance.m, the auxiliary MATLAB algorithm for producing a valid DDP instance according the given parameters.

- strABC.m, the auxiliary MATLAB algorithm for producing Maple commands for reading data before calling the Maple algorithm DDdraw.mws.
- simu1004.m, simu1004plots.m, simu1007.m, and simu1008.m, the auxiliary MATLAB algorithms for organizing simulations and producing the related figures.
- Trans.m, the MATLAB algorithm for implementing Scaling-rounding-adjusting approach for Cases of INS_2 .
- Plot1.m, the auxiliary MATLAB algorithms for comparison of DDmap and algorithm in [3, 13].
- Plot2.m, the auxiliary MATLAB algorithms for comparison of DDmap and other algorithms under the condition of INS_2 .
- 1 Maple algorithm, DDdraw.mws, is used for drawing the DDP solution in nested pie charts, with inputs A, B and C that are assigned by using Maple commands produced by strABC.m.
- 43 Data files: 42 of them are named as $\text{INS}_{xx-ggg} \cdot \text{TEX}$, where $xx \in \{01, 03, 04, 05, 07, 08, 10\}$ and $ggg \in \{pcc, rsc, p4x, flp, csh, all\}$, and the last is named as $\text{INS}_{02-rs-1008} \cdot \text{TEX}$. These data files are in fact the running records of our simulations towards the 7 valid DDP test instances given in [13]. In these running records, many exact DDP solutions are provided.

Abbreviations

CSH: Cyclic shifting; DDE: Double digest experiments; DDP: Double digest problem; FLP: Flipping; P4X: Four-point exchange; PCC: Permutations combination crossing; RSC: Referencing sorting crossing; RWS: Roulette wheel selection

Acknowledgements

We thank the anonymous reviewers for giving us valuable suggestions.

Funding

This work was supported by the National Key R&D Program of China (Grant No. 2016YFB0800602), the Shandong provincial Key R&D Program of China (Grant No. 2018CXGC0701), and the 111 Project (No. B08004). The experimental environments and the editing services are supported by the Shandong provincial Key R&D Program of China (Grant No. 2018CXGC0701) and the 111 Project (No. B08004), respectively. The publication cost is funded by National Key R&D Program of China (Grant No. 2016YFB0800602). The funders had no role in the design of the study and collection, analysis, and interpretation of data and in writing the manuscript.

Availability of data and materials

DDmap is open source and available at <https://github.com/wanglc2012/DDmap>.

Authors' contributions

LW proposed the main concept of the presented technique, implemented the main framework. JS finished the main experimental study, as well as analysis on related work. YP conceived the main idea about this work and implemented the algorithms about different genetic operators. LL verified the main method, checked the correctness of the presented technique. All authors reviewed the manuscript and approved the final manuscript.

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, 10 West Tucheng Road, Haidian District, Beijing 100876, China. ²School of Computer Science, Communication University of China, 1 East of Dingfuzhuang Street, Chaoyang District, Beijing 100024, China.

Received: 19 February 2019 Accepted: 29 April 2019

Published online: 18 June 2019

References

1. Blazewicz J, Burke E, Jaroszewski M, Kasprzak M, Paliswiat B, Pryputniewicz P. On the complexity of the double digest problem. *Control Cybern.* 2004;33(1):133–40.
2. Cieliebak M, Eidenbenz S, Woeginger GJ. Double digest revisited: complexity and Approximability in the presence of Noisy data. In: *International conference on Computing & Combinatorics*: Springer-Verlag; 2003. p. 519–27.
3. Ganjtabesh M, Ahrabian H, Nowari-Dalini A, Moghadam ZRK. Genetic algorithm solution for double digest problem. *Bioinformation.* 2012; 8(10):453–6.
4. Goldstein L, Waterman MS. Mapping DNA by stochastic relaxation. *Adv Appl Math.* 1987;8:194–207.
5. Kao MY, Samet J, Sung WK. The enhanced double digest problem for dna physical mapping. *J Comb Optim.* 2003;7(1):69–78.
6. Karp R. Mapping the genome: some combinatorial problems arising in molecular biology. In: *ACM symposium on theory of computing*; 1993. p. 278–85.
7. Nathans D, Smith HO. Restriction endonucleases in the analysis and restructuring of DNA molecules. *Annu Rev Biochem.* 1975;44:273–93.
8. Pevzner PA. DNA physical mapping, flows in networks and minimum cycles mean in graphs. In: Gindikin SG, editor. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 8: Mathematical Methods of Analysis of Biopolymer Sequences*, pages 99–112. Providence: American Mathematical Society; 1992.
9. Pevzner PA. DNA physical mapping and alternating Eulerian cycles in colored graphs. *Algorithmica.* 1995;13(1/2):77–105.
10. Schmitt W, Waterman MS. Multiple solutions of DNA restriction mapping problems. *Adv Appl Math.* 1991;12:412–27.
11. Smith HO, Wilcox KW. A restriction enzyme from *Hemophilus influenzae*. I. Purification and general properties. *J Mol Biol.* 1970;51(2):379–91.
12. Stefik M. Inferring DNA structure from segmentation data. *Artif Intell.* 1978;11:85–114.
13. Sur-Kolay S, Banerjee S, Mukhopadhyaya S, et al. Genetic algorithm for double digest problem. In: *International conference on Pattern Recognition & Machine Intelligence*. Berlin Heidelberg: Springer; 2005.
14. Waterman MS, Griggs JR. Interval graphs and maps of DNA. *Bull Math Biol.* 1986;48(2):189–95.
15. Wu Z, Zhang Y. Solving large double digestion problems for DNA restriction mapping by using branch-and-bound integer linear programming. *Int J Bioinforma Res Appl.* 2008;4(4):351.

Ready to submit your research? Choose BMC and benefit from:

- fast, convenient online submission
- thorough peer review by experienced researchers in your field
- rapid publication on acceptance
- support for research data, including large and complex data types
- gold Open Access which fosters wider collaboration and increased citations
- maximum visibility for your research: over 100M website views per year

At BMC, research is always in progress.

Learn more biomedcentral.com/submissions

