A parallel multistart algorithm for the closest string problem

Article *in* Computers & Operations Research · November 2008

DOI: 10.1016/j.cor.2007.04.002 · Source: DBLP

CITATIONS
25

READS
133

4 authors:

Fernando Carvalho-Gomes
Universidade Federal do Ceará
36 PUBLICATIONS **238** CITATIONS
SEE PROFILE

Claudio Meneses
Universidade Federal do ABC (UFABC)
19 PUBLICATIONS **318** CITATIONS
SEE PROFILE

Panos Pardalos
University of Florida
1,719 PUBLICATIONS **43,523** CITATIONS
SEE PROFILE

Gerardo Valdisio R. Viana
Universidade Federal do Ceará
11 PUBLICATIONS **119** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 3636–3643
computers & operations research
www.elsevier.com/locate/cor

A parallel multistart algorithm for the closest string problem☆

Fernando C. Gomes[a,*], Cláudio N. Meneses[b,1], Panos M. Pardalos[b], Gerardo Valdisio R. Viana[a,c]

[a]*Department of Computer Science, Federal University of Ceara, Brazil*
[b]*Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32611, USA*
[c]*Computer Science Department, State University of Ceara, Brazil*

Available online 19 April 2007

Abstract

In this paper we describe and implement a parallel algorithm to find approximate solutions for the *Closest String Problem* (CSP). The CSP, also known as *Motif Finding* problem, has applications in Coding Theory and Computational Biology. The CSP is NP-hard which motivates us to think about heuristics to solve large instances. Several approximation algorithms have been designed for the CSP, but all of them have a poor performance guarantee. Recently some researchers have shown empirically that integer programming techniques can be successfully used to solve moderate-size instances (10–30 strings each of which is 300–800 characters long) of the CSP. However, real-world instances are larger than those tested. In this paper we show how a simple heuristic can be used to find near-optimal solutions to that problem. We implemented a parallel version of this heuristic and report computational experiments on large-scale instances. These results show the effectiveness of our approach.

Keywords: Sequence analysis; Parallel algorithms; Closest String Problem

1. Introduction

String selection problems are among the most important faced by researchers in computational biology. These problems are important in finding sequence conserved regions, genetic drug target identification, and genetic probes in molecular biology. Combinatorial optimization is one possible approach to solve String Selection Problems. Recently, some authors have demonstrated empirically that integer programming techniques are suitable to solve moderate-size instances of the Closest String Problem (CSP) [1,2], but real-world size instances are much larger and difficult.

☆ This work has been partially supported by NSF, NIH and CRDF grants. A preliminary version of this paper appeared at the proceedings of the I International Symposium on Mathematical and Computational Biology, November 2004, Brazil.
* Corresponding author.
E-mail addresses: carvalho@lia.ufc.br (F.C. Gomes), claudio@ufl.edu (C.N. Meneses), pardalos@ufl.edu (P.M. Pardalos), valdisio@lia.ufc.br (G.V.R. Viana).
[1] Supported in part by the Brazilian Federal Agency for Higher Education (CAPES)—Grant no. 1797-99-9.

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2007.04.002

In this paper we are concerned with reporting speed up obtained by a parallel algorithm for string selection. More precisely, we present a parallel algorithm for the CSP and report computational results for large-scale instances. We aim at showing that parallelization can play an important role in solving String Selection Problems.

In Section 2, we define the CSP, and a parallel algorithm is presented and analyzed. In Section 3, computational issues about the parallel algorithm are presented. In Section 4, we present the experimentation setup, the results and the discussion. Conclusions are given in Section 5.

2. Problem and algorithms

Let s and t be two strings with equal length (i.e., $|s| = |t|$). The number of mismatched positions between those two strings is the Hamming distance $d_H(s, t)$ of them. For example, if s="ACT" and t="CCA", then $d_H(s, t) = 2$. We use s_j^i to denote the character in the jth position of string s^i. Through out the text the terms closest and farthest are used with respect to the maximum Hamming distance.

The CSP can be defined as:

Instance: Given a finite set $\mathscr{S}_c = \{s^1, s^2, \ldots, s^n\}$ of n strings of length m over an alphabet $\mathscr{A}$.

Objective: Find a string x of length m over $\mathscr{A}$ minimizing d_c such that for every string s^i in $\mathscr{S}_c$, $d_H(x, s^i) \leqslant d_c$.

Example 1. Let $\mathscr{S}_c = \{$"*ACGT*", "*TTAC*", "*CCGC*"$\}$. String "*TCGC*" is an optimal solution to this instance and it has d_c equal to 2.

The CSP is NP-hard and has applications in computational biology [3]. There exist approximation algorithms for the CSP (see e.g., [3–5]). In [1] it has been shown empirically the practical use of integer programming techniques to solve moderate-size instances of the CSP. By moderate-size instances we mean instances with 10–30 strings each of which is 300–800 characters long.

In [3], for example, an algorithm with performance guarantee of $\frac{4}{3}(1 + \varepsilon)$, for any small constant $\varepsilon > 0$, is presented and analyzed, with a similar result appearing also in [6]. A PTAS for the CSP was presented in [5]. These algorithms make use of integer linear programming and so to solve large-size instances of the CSP using them may take a long time.

In this paper we take a different direction and design a parallel heuristic that is a combination of an approximation algorithm and local search strategies. In this way, we first generate feasible solutions that are guaranteed to have a performance factor from the optimum value, and then we apply local search strategies to improve the solutions. We do the local search using several processors in a parallel machine, and thus we get a parallel implementation.

The heuristic is described in Algorithm 1. This algorithm is a slightly different version of the one presented in [1]. The different point is on step 3 (local search procedure). Step 3 of the algorithm in [1] runs for exactly N (parameter) iterations, whereas step 3 in Algorithm 1 may restart itself as long as it finds a solution better than the incumbent (current best solution).

The heuristic consists of taking a string in $\mathscr{S}_c$ and modifying it until a locally optimal solution is found. Recall that selecting any string in $\mathscr{S}_c$ as a solution gives a 2-approximation algorithm (see [4]). Thus, all solutions generated by our heuristic have a performance guarantee of 2. We decided to use that simple 2-approximation algorithm as part of our heuristic, instead of the best approximation algorithm for the CSP, because the former runs always fast.

In the first step, the algorithm searches for a string $s \in \mathscr{S}_c$ that is the closest to all other strings in $\mathscr{S}_c$. In the second step, the distance d_c between s and the remaining strings is computed. In the last step of Algorithm 1, a local search procedure is applied as described below.

Let s^b be a string in $\mathscr{S}_c$ such that $d_H(s^b, s)$ is maximum. If $s_i \neq s_i^b$, with $i \in \{1, \ldots, m\}$, then replace s_i by s_i^b if the replacement makes the solution better. If the replacement occurs, update the Hamming distances from s to all strings in $\mathscr{S}_c$. After having scanned all m positions, select $s^b \in \mathscr{S}_c$ where s^b is the farthest string from the resulting s, and repeat the process.

The idea behind the local search is to make the current solution s look like the string s^b. The number of iterations of the local search is controlled by the parameter N. The details of this algorithm are presented in Algorithm 2. We use an auxiliary n-size array, d', to recompute rapidly the distances between the current solution s and the string s^b.

Algorithm 1. Heuristic for the CSP.

Input: instance $\mathscr{S}_c$, N
Output: string s, distance d_c
1 $s \leftarrow$ string in $\mathscr{S}_c$ closest to the other $s^i \in \mathscr{S}_c$
2 $d_c \leftarrow \max_{i\in\{1,\ldots,n\}} d_H(s^i, s)$
3 improve_solution($\mathscr{S}_c$, s, d_c, N)

The parallel algorithm is described in Algorithm 3.

Algorithm 2. Step 3 of Algorithm 1.

Input: instance $\mathscr{S}_c$, current solution s, distance d_c, and parameter N
Output: resulting solution s and distance d_c
for $k \leftarrow 1$ **to** n **do**
  $d_k \leftarrow d_H(s^k, s)$
  $d'_k \leftarrow d_k$
end
$OldMax \leftarrow d_c$
$NoImprov \leftarrow 0$
While $NoImprov \leqslant N$ **do**
  $b \leftarrow i$ such that $d_H(s^i, s) = d_c$ /* break ties randomly */
  for $j \leftarrow 1$ **to** m such that $s^b_j \neq s_j$ **do**
    $max \leftarrow -1$
    for $k \leftarrow 1$ **to** n such that $k \neq b$ **do**
      if $(s_j = s^k_j)$ **and**$(s^b_j \neq s^k_j)$ **then** $d_k \leftarrow d_k + 1$
      else if $(s_j \neq s^k_j)$ **and** $(s^b_j = s^k_j)$ **then** $d_k \leftarrow d_k - 1$
      if $(max < d_k)$ **then** $max \leftarrow d_k$
    end
    if $d_c \geqslant max$ /* this is not worse */ **then**
      $d_c \leftarrow max$; $s_j \leftarrow s^b_j$
      for $k \leftarrow 1$ **to** n **do** $d'_k \leftarrow d_k$
    else
      for $k \leftarrow 1$ **to** n **do** $d_k \leftarrow d'_k$
    end
  end
  if $OldMax \neq d_c$ **then**
    $OldMax \leftarrow d_c$
    $NoImprove \leftarrow 0$
  else
    $NoImprov \leftarrow NoImprov + 1$
  end
end

3. Implementation issues

In the implementation of the parallel algorithm we use a *multiple independent-thread strategy* which is depicted in Fig. 1 and consists of: (a) p processors are available; (b) iterations are evenly distributed over the p processors; (c) each processor keeps a copy of data and algorithms; (d) one processor acts as the master handling seeds, data and iteration counter; (e) each processor performs N iterations (N will be explained in Section 4.2).

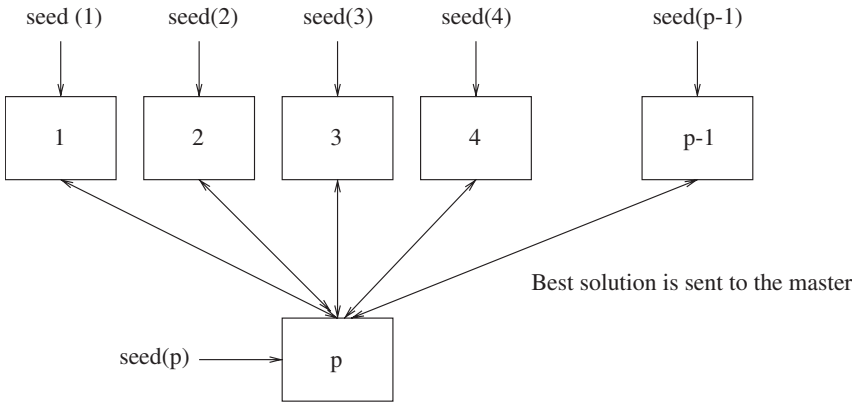

Fig. 1. Parallel independent implementation strategy.

4. Computational experiments

We now present the computational experiments carried out with the algorithm described in the previous section. Initially, we describe the set of instances used in the tests. Then, in Section 4.2 the results obtained by the parallel algorithm are shown.

Algorithm 3. Parallel algorithm for the CSP.

Input: instance $\mathscr{S}_c$, N (number of iterations),
NTasks (number of parallel tasks)
Output: solution *Smin*, distance *Dmin*
/* **MASTER PROCESS** */
Start NTasks parallel processes
Send data ($\mathscr{S}_c$, N) to the Slave Process
$Dmin \leftarrow +\infty$ (upper bound on an optimal solution value)
for $i = 1$ **to** *NTasks* **do**
  /* wait for results from the slave processes */
  receive results D_i, S_i
  show partial solution D_i
  if $Dmin > D_i$ **then**
    $Dmin \leftarrow D_i$
    $Smin \leftarrow S_i$
  end
end
/* **SLAVE PROCESS** */
Receive data from the Master Process
Call Heuristic for the CSP (Algorithm 1)
Receive results s, d
Send results s, d to the Master Process

4.1. Instances and test environment

We use actual and simulated biological data. For the simulated data the instances were randomly generated in the following way. Given parameters n (number of strings), m (string length), and an alphabet $\mathscr{A}$, randomly choose (using a uniform distribution) a character from $\mathscr{A}$ for each position in the resulting string.

We also test simulated instances using the alphabet $\mathscr{A} = \{A, C, G, T\}$ with strings having the GC content (the % of G's and C's) equals to 72%, which is the content for the Actinobacteria Streptomyces coelicolor.

In the simulated data we test three classes of instances. The first class consists of strings with an alphabet of size two; the second one involves strings that use an alphabet of size four and the third class contains strings that adopt an alphabet of size 20. Instances with alphabets of size two find applications in Coding Theory, whereas instances using alphabets with sizes four and 20 appear in applications involving DNA and amino acid sequences, respectively.

The algorithm used for random-number generation is an implementation of the multiplicative linear congruential generator [7], with parameters 16 807 (multiplier) and $2^{31} - 1$ (prime number).

All tests were executed on a parallel machine with 28 nodes Intel Xeon DP, dual processed cluster, each node having 1 Gbyte RAM, and the data network being Gigabit Ethernet. The heuristic algorithm was implemented in the C++ language and PVM (Parallel Virtual Machine) [8] was used to implement the parallel version of the algorithm. The CPLEX 7.0 [9] was used for solving the integer programming models that compute the optimal solution values to instances of the CSP (see [1]).

The master process was implemented in FORTRAN and the slaves in C++. To the best of our knowledge, this is the first time that an application of PVM using two different programming languages was developed.

Since the integer programs for the large instances tested in this paper are relatively large, the time limit to the CPLEX solver was set to 1 h. We emphasize that we are mostly interested in analyzing the quality of the solutions delivered by the parallel algorithm and *not* in comparing CPU times spent by the exact and the parallel algorithms.

4.2. Discussion

Seventeen actual instances were tested: six use alphabet of size four (McClure instances [10]) and 11 use alphabet with 20 characters (GenBank [11]). The McClure instances are protein sequences frequently used to test string-comparison algorithms. The sequences within these instances are fairly diverse. In the instances obtained from GenBank, seven out of 11 instances have sequences fairly similar.

The simulated data consists of 135 instances, with 45 instances for each alphabet.

For each of the actual instances considered, the size of the strings (m) is equal to the length of the smallest string in the set (this was necessary since the instances have strings of different lengths). We removed the last characters for strings with length greater than the minimum.

Tables 1 and 2 show the results for actual biological data. In these tables, the first three columns give the instance name, the number of strings and their lengths. Columns Val. and Time show the solution value and CPU time (in seconds) to compute it. The column Ratio is explained below.

Tables 3–6 show the results over simulated data. Each row in these tables correspond to three instances. The columns are interpreted as follows. The first two columns give the number of strings (n) and the lengths of the strings (m) in an instance. Columns Min, Avg. and Max represent the minimum, the average and the maximum solution values, while Time column describes the average time over three instances of same size. All times are expressed in seconds. The column labeled Ratio expresses the relative ratio between the parallel algorithm solution value and the optimal solution

Table 1
Results for actual instances over the alphabet with four characters

Instance			Exact algorithm		Ratio	Parallel algorithm	
Name	n	m	Val.	Time	Val.	Val.	Time
16SrRNAFasta	8	457	3	0.01	1.0	3	0.1
16SrRNAFasta2	8	457	3	0.01	1.0	3	0.1
1021Fasta	10	1051	58	0.02	1.0	58	0.1
1051Fasta	3	1537	4	0.01	1.0	4	0.1
2521Fasta	4	2563	867	0.19	1.0	867	0.1
2641Fasta	6	2678	964	0.59	1.060	1022	0.1
HistoneH3-aFasta	21	122	6	0.01	1.0	6	0.1
HistoneH3Fasta	21	539	204	0.80	1.001	206	2.0
HistonebFasta	21	324	151	0.18	1.013	153	1.0
Zhing1997Fasta	10	212	36	0.01	1.0	36	0.1
ZhingS2-1780-5	5	1775	488	0.13	1.020	498	0.1

Table 2
Results for McClure instances over the alphabet of 20 characters

Instance			Exact algorithm		Ratio	Parallel algorithm	
Name	*n*	*m*	Val.	Time	Val.	Val.	Time
mc582.10	10	141	97	0.09	1.041	101	0.1
mc582.12	12	141	97	0.09	1.031	100	0.1
mc582.6	6	141	95	0.04	1.063	101	0.1
mc586.6	6	100	72	0.03	1.069	77	0.1
mc586.10	10	98	75	0.06	1.040	78	0.1
mc586.12	12	98	76	0.08	1.039	79	0.1

Table 3
Results for simulated data with an alphabet of four characters and GC content of 72%

Instance		Exact algorithm				Ratio	Parallel algorithm			
n	*m*	Min	Avg.	Max	Time	Avg.	Min	Avg.	Max	Time
10	1000	579	580.7	583	0.6	1.013	585	588.5	591	2.7
10	2000	1117	1126.0	1132	3.1	1.011	1127	1138.6	1145	5.3
10	3000	1615	1622.7	1628	1.7	1.010	1628	1638.4	1642	8.5
10	4000	2081	2087.7	2096	6.6	1.010	2097	2107.6	2120	11.7
10	5000	2496	2500.7	2508	7.8	1.009	2512	2522.5	2529	17.6
20	1000	633	633.0	633	364.5	1.028	648	650.6	655	4.4
20	2000	1218	1222.7	1227	1258.3	1.027	1245	1255.2	1266	9.0
20	3000	1763	1765.7	1768	1202.2	1.023	1797	1805.8	1815	19.3
20	4000	2260	2261.7	2263	1203.8	1.021	2302	2309.8	2326	25.8
20	5000	2713	2723.3	2731	2480.6	1.021	2761	2780.2	2761	33.1
30	1000	655	655.3	656	2560.4	1.032	671	676.2	687	6.3
30	2000	1262	1265.0	1267	3600.0	1.028	1293	1300.6	1308	16.3
30	3000	1828	1830.7	1833	3600.0	1.027	1874	1880.3	1894	30.3
30	4000	2342	1347.3	2354	3600.0	1.025	2397	2405.9	2412	46.3
30	5000	2816	2819.3	2821	3600.0	1.024	2877	2886.0	2894	61.8

Table 4
Results for simulated data with an alphabet of four characters

Instance		Exact algorithm				Ratio	Parallel algorithm			
n	*m*	Min	Avg.	Max	Time	Avg.	Min	Avg.	Max	Time
10	1000	578	581.3	584	0.9	1.016	588	590.7	593	1.2
10	2000	1159	1163.0	1165	2.8	1.013	1175	1178.3	1180	4.8
10	3000	1731	1737.7	1741	18.3	1.013	1753	1760.0	1768	6.9
10	4000	2313	2317.7	2323	3.3	1.011	2337	2342.7	2348	14.7
10	5000	2899	2901.7	2906	25.1	1.010	2928	2931.0	2934	18.9
20	1000	630	632.0	635	1382.1	1.030	649	651.0	655	3.3
20	2000	1258	1261.7	1265	3601.8	1.027	1293	1295.3	1299	10.6
20	3000	1886	1889.7	1895	48.6	1.025	1933	1937.7	1944	21.2
20	4000	2520	2523.3	2528	1205.6	1.026	2585	2588.3	2594	29.9
20	5000	3152	3154.0	3155	3600.0	1.025	3232	3233.3	3235	45.0
30	1000	654	655.3	656	3600.0	1.031	673	675.3	678	7.1
30	2000	1306	1307.3	1310	3600.0	1.030	1344	1347.0	1353	17.9
30	3000	1954	1956.0	1960	3600.0	1.029	2010	2013.7	2020	32.3
30	4000	2610	2612.0	2613	3600.0	1.031	2692	2692.3	2693	49.7
30	5000	3265	3267.0	3269	3600.0	1.030	3358	3364.0	3369	72.8

Table 5
Results for simulated data with an alphabet of two characters

Instance		Exact algorithm				Ratio	Parallel algorithm			
n	*m*	Min	Avg.	Max	Time	Avg.	Min	Avg.	Max	Time
10	1000	371	376.0	379	1200.1	1.009	375	379.3	382	0.1
10	2000	752	755.0	757	2400.1	1.009	761	761.7	783	1.0
10	3000	1129	1130.0	1131	2400.4	1.006	1134	1136.7	1145	2.1
10	4000	1496	1507.3	1517	1205.3	1.008	1504	1519.3	1532	3.2
10	5000	1889	1891.3	1893	1204.9	1.005	1895	1900.3	1906	5.3
20	1000	412	412.3	413	1215.2	1.028	421	424.0	427	1.0
20	2000	817	824.0	831	1249.6	1.093	874	882.3	898	2.0
20	3000	1232	1235.3	1240	2438.1	1.029	1255	1270.7	1291	3.4
20	4000	1643	1647.7	1653	253.2	1.041	1697	1716.0	1742	4.0
20	5000	2058	2061.7	2067	2406.4	1.025	2101	2112.3	2114	5.2
30	1000	431	433.3	436	3600.0	1.035	443	448.3	455	1.0
30	2000	857	862.7	869	3600.0	1.043	885	900.0	921	3.0
30	3000	1292	1295.0	1300	3600.0	1.032	1328	1337.0	1359	4.2
30	4000	1711	1713.0	1714	3600.0	1.048	1764	1794.7	1825	6.1
30	5000	2140	2141.7	2143	3600.0	1.044	2231	2237.0	2248	8.0

Table 6
Results for simulated data with an alphabet of 20 characters

Instance		Exact algorithm				Ratio	Parallel algorithm			
n	*m*	Min	Avg.	Max	Time	Avg.	Min	Avg.	Max	Time
10	1000	781	783.0	785	1.2	1.043	815	817.0	820	3.1
10	2000	1560	1564.3	1569	4.6	1.034	1610	1617.7	1622	10.4
10	3000	2344	2345.3	2347	5.2	1.028	2409	2411.7	2416	19.2
10	4000	3124	3129.0	3133	7.0	1.024	3192	3204.7	3216	30.7
10	5000	3910	3912.0	3913	15.1	1.022	3999	3999.7	4001	41.3
20	1000	837	838.0	839	95.1	1.065	889	892.3	896	6.3
20	2000	1675	1677.3	1680	1206.9	1.059	1771	1775.7	1779	20.2
20	3000	2510	2511.7	2513	1222.5	1.056	2651	2653.3	2658	39.1
20	4000	3355	3357.0	3359	1220.6	1.053	3531	3533.3	3535	69.3
20	5000	4190	4192.0	4194	1222.2	1.049	4395	4399.0	4405	112.1
30	1000	860	861.7	863	2401.5	1.064	916	916.7	917	6.9
30	2000	1722	1723.0	1724	1210.7	1.060	1825	1827.0	1830	22.4
30	3000	2581	2582.0	2583	2411.0	1.059	2734	2734.7	2735	48.8
30	4000	3447	3448.0	3449	1226.9	1.058	3646	3648.3	3650	67.0
30	5000	4305	4306.0	4307	2488.9	1.057	4550	4553.3	4556	103.7

value, and is calculated as h/i, where h is the average solution value obtained by the parallel algorithm and i is the average optimal solution value over three instances of same size.

The parallel algorithm uses 20 processors. Communication overhead did not affect the overall performance of the algorithm. The stopping condition for the parallel algorithm was the maximum number of iterations between two improvements and this parameter was set to 2000 iterations. This is the parameter N mentioned in Section 2.

The performances of the exact and parallel algorithms are striking for instances derived from actual biological data. For simulated data we can see a clear distinction between the behavior of the algorithms. The exact algorithm performs very well on instances with small value of n, whereas the parallel algorithm seems to have a uniform performance over all instances tested.

The results produced by the parallel algorithm are quite impressive. On average it yields solution values within 2.0%, 2.3%, 2.9% and 4.9% of the optimum values for simulated instances with alphabets of sizes four (GC content 72%), four, two and 20, respectively. For actual data the results are even better. On the other hand, the exact algorithm

fails to deliver optimal solutions to several instances in the time limit of 1 h. For these instances we report the values of the best feasible solutions found within 1 h of CPU time. Another attractive aspect of our algorithm is the CPU times to find high-quality solutions. All solutions were obtained in less than 2 min. An interesting finding is that our algorithm produced very quickly near optimal solutions to instances with an alphabet of two characters, whereas the exact algorithm struggled to find optimal solutions. Clearly, the parallel algorithm would be a decent way to speed the exact algorithm by providing upper bounds when searching for optimal solutions. Note that this is applicable to all instances tested in this paper.

5. Concluding remarks

In this paper we proposed a parallel algorithm for the CSP. The algorithm was implemented and the computational experiments showed that our approach was very effective in solving large-scale instances.

We have not noticed communication overhead when the number of processors exceeds 10. We believe that parallelization of existing algorithms shall be an important research avenue.

References

[1] Meneses CN, Lu Z, Oliveira CAS, Pardalos PM. Optimal solutions for the closest string problem via integer programming. INFORMS Journal on Computing 2004;16(4):419–29.
[2] Meneses CN, Oliveira CAS, Pardalos PM. Optimization techniques for string selection and comparison problems in genomics. IEEE Engineering in Medicine and Biology Magazine 2005;24(3):81–7.
[3] Lanctot K, Li M, Ma B, Wang S, Zhang L. Distinguishing string selection problems. Information and Computation 2003;185(1):41–55.
[4] Ben-Dor A, Lancia G, Perone J, Ravi R. Banishing bias from consensus sequences. In: Apostolico A, Hein J, editors. Proceedings of the 8th annual symposium on combinatorial pattern matching. Lecture notes in computer science, vol. 1264. Aarhus, Denmark: Springer; 1997. p. 247–61.
[5] Li M, Ma B, Wang L. On the closest string and substring problems. Journal of the ACM 2002;49(2):157–71.
[6] Gasieniec L, Jansson J, Lingas A. Efficient approximation algorithms for the hamming center problem. In: Proceedings of the 10th ACM-SIAM symposium on discrete algorithms. 1999. p. S905–6.
[7] Park S, Miller K. Random number generators: good ones are hard to find. Communications of the ACM 1988;31:1192–201.
[8] PVM user's guide and reference manual. OAK Ridge National Laboratory; Geist A, Beguelin A, Dangarra J, Jiang W, Machek R, Sunderan V. 1994.
[9] ILOG Inc. CPLEX 7.0 User's Manual; 2003.
[10] McClure M, Vasi T, Fitch W. Comparative analysis of multiple protein-sequence alignment methods. Molecular Biology and Evolution 1994;11:571–92.
[11] GenBank. ⟨http://www.ncbi.nlm.nih.gov/genbank/index.html⟩ [last accessed on January 30, 2007].