Solving the Double Digestion Problem as a Mixed-Integer Linear Program*

Zhijun Wu† and Yin Zhang‡

August, 2001

Abstract. The double digestion problem for DNA restriction mapping is known to be NP-complete. Several approaches to the problem have been used including exhaustive search, simulated annealing, branch-and-bound. In this paper, we consider a mixed-integer linear programming formulation of the problem and show that with this formulation the problem can be solved efficiently to a fairly large size using state-of-the-art integer programming techniques. In particular, we present computational results obtained by using the CPLEX mixed-integer linear programming software on a set of randomly generated, large-scale double digestion problems.

Key Words. DNA sequencing, restriction mapping, double digestion, NP-complete problems, mixed-integer linear programming

1 Introduction

The double digestion problem arises in the construction of restriction mappings for DNA sequences. Two restriction enzymes are applied to a DNA sequence to cut the sequence into small fragments. The lengths of the fragments can be measured by using the electrophoretic gel, but how the fragments are connected or ordered in the original sequence is unknown. Let the

*Computational results reported in this paper were obtained on an SGI Origin2000 computer at Rice University acquired in part with support from NSF Grant DMS-9872009.

†Department of Mathematics, Iowa State University, Ames, IA, 50010. Work supported in part by DOE/LANL Contract 03891-99-23.

‡Department of Computational and Applied Mathematics, Rice University, Houston, Texas, 77005. Work supported in part by DOE Grant DE-FG03-97ER25331 and DOE/LANL Contract 03891-99-23.

two restriction enzymes be denoted by A and B. Let $a = \{a_1, a_2, \ldots, a_n\}$ be the set of the fragments obtained after applying enzyme A to the DNA sequence, and $b = \{b_1, b_2, \ldots, b_m\}$ be the set obtained after applying enzyme B. Let $c = \{c_1, c_2, \ldots, c_l\}$ be the set of the fragments obtained after first applying enzyme A and then enzyme B. The fragments in a or b alone do not suggest any orders. However, given the fragments in c, a and b must be in certain orders so that c can indeed be produced when the two enzymes for a and b are combined. The double digestion problem is to find the right orders for a and b given the fragments in c as the result of the sequence being digested by the two enzymes a and b together. For a general review on the problem and its biological applications, readers are referred to a standard text book by Waterman [13].

The double digestion problem is solved routinely in molecular biology labs for constructing the restriction maps for newly cloned DNA sequences. The problem can be difficult to solve, however, if the number of fragments is large. In general, it is intractable from the viewpoint of computational complexity as shown in Goldstein and Waterman [3]. Several approaches to the problem have been proposed, such as searching for all possible permutations [9, 11], simulated annealing [3, 4], and fragment matching [2, 7, 12]. Software packages based on these approaches have been developed and used in practice [14]. Moreover, in real applications, the lengths of the DNA fragments are usually subject to measurement errors, and there are often multiple solutions to the double digestion problem as well while the correct order of the fragments in a given DNA sequence should be unique [10].

In this paper, we study some fundamental issues related to the formulation and solution of the double digestion problem. We approach the problem from a more mathematical point of view and formulate a crucial part of the problem, called fragment matching, as a pair of mixed-integer linear programs. We show that the fragment matching problem can then be solved directly by using available mixed-integer programming software. More specifically, we try to match the double-digested fragments to each of the single-digested ones and then determining an order for each set of single-digested fragments. The fragment matching problem is formulated as mixed-integer linear programs that minimize the matching errors in either l_1 or l_∞ norm. We then use the CPLEX mixed-integer linear programming software to solve the problem. In this approach, we have been able to obtain exact solutions to a set of randomly-generated digestion problems with up to several hundreds of fragments.

We describe the mixed-integer linear programming formulation in greater detail in Section 2 and introduce the CPLEX mixed-integer linear programming software in Section 3. Computational results are presented and discussed in Section 4. Comments and remarks are given in Section 5.

2 MIP Formulation

Let A and B be two restriction enzymes. Let $a = \{a_1, a_2, \ldots, a_n\}$ be the set of the fragments obtained after applying enzyme A to the DNA sequence and $b = \{b_1, b_2, \ldots, b_m\}$ the set obtained after applying enzyme B. Let $c = \{c_1, c_2, \ldots, c_l\}$ be the set of the fragments obtained after applying enzyme A to the given sequence followed by enzyme B to the fragments in a, or in the other way around, applying enzyme B to the given sequence followed by enzyme A to the fragments in b. In order to find correct orders for a and b, we need to find for every element in c which element in a it comes from, and the same thing in b. We call this problem the fragment matching problem. It is an important part of the double digestion problem. Once this problem is solved, a solution to the double digestion problem can be constructed easily using for example a fragment ordering algorithm as described in Chapter 2 of Waterman [13].

Let X be a $n \times l$ matrix and $x_{i,j}$ the i,j-element of X, and

$$x_{i,j} = \{ \begin{array}{ll} 1, & c_j \subset a_i, \\ 0, & otherwise. \end{array}$$

Let Y be a $m \times l$ matrix and $y_{i,j}$ the i,j-element of X, and

$$y_{i,j} = \{ \begin{array}{ll} 1, & c_j \subset b_i, \\ 0, & otherwise. \end{array}$$

Since the elements c_j in c that come from the ith fragment of a add to a_i and that from the ith fragment of b add to b_i,

$$a_i = \sum_{j=1}^{l} x_{i,j} c_j, \qquad i = 1, \ldots, n,$$

and similarly,

$$b_i = \sum_{j=1}^{l} y_{i,j} c_j, \qquad i = 1, \ldots, m.$$

In matrix form the equations are equivalent to

$$a = Xc, \qquad b = Yc.$$

Note that each element in c may come from only one element in a or b. Therefore,

$$\begin{array}{ll} \sum_{i=1}^{n} x_{i,j} = 1, & j = 1, \ldots, l, \\ \sum_{i=1}^{m} y_{i,j} = 1, & j = 1, \ldots, l. \end{array}$$

Clearly the fragment matching problem for a triple sets a, b, and c is basically to find X and Y so that the above equations hold. As stated in Waterman [13], the problem can be formulated as a linear integer program,

$$\begin{array}{rl} \min_{\alpha,\beta,X,Y} & \alpha + \beta \\ \textit{subject to} & \alpha \geq \displaystyle\sum_{j=1}^{l} x_{i,j} c_j - a_i \geq -\alpha, \qquad i = 1, \ldots, n \\ & \beta \geq \displaystyle\sum_{j=1}^{l} y_{i,j} c_j - b_i \geq -\beta, \qquad i = 1, \ldots, m \\ & \displaystyle\sum_{i=1}^{n} x_{i,j} = 1, \qquad j = 1, \ldots, l \\ & \displaystyle\sum_{i=1}^{m} y_{i,j} = 1, \qquad j = 1, \ldots, l \\ & x_{i,j},\ y_{i,j} \in \{0, 1\}. \end{array}$$

The program is separable and equivalent to two separate sub-programs,

$$\begin{array}{rl} \min_{\alpha,X} & \alpha \\ \textit{subject to} & \alpha \geq \displaystyle\sum_{j=1}^{l} x_{i,j} c_j - a_i \geq -\alpha, \qquad i = 1, \ldots, n \\ & \displaystyle\sum_{i=1}^{n} x_{i,j} = 1, \qquad j = 1, \ldots, l \\ & x_{i,j} \in \{0, 1\}. \end{array}$$

and

$$\min_{\beta,Y} \qquad \beta$$

$$\begin{aligned} \textit{subject to} \quad & \beta \geq \sum_{j=1}^{l} y_{i,j} c_j - b_i \geq -\beta, \qquad i = 1, \ldots, m \\ & \sum_{i=1}^{m} y_{i,j} = 1, \qquad j = 1, \ldots, l \\ & y_{i,j} \in \{0, 1\}. \end{aligned}$$

The two sub-programs have the same mathematical structure. It suffices to consider only one of them. We put it in the following general form,

$$\begin{aligned} (\mathrm{P}_\infty) \qquad \min\nolimits_{t,X} \qquad & t \\ \textit{subject to} \quad & t \geq \sum_{j}^{l} x_{i,j} c_j - a_i \geq -t, \qquad i = 1, \ldots, n \\ & \sum_{i}^{n} x_{i,j} = 1, \qquad j = 1, \ldots, l \\ & x_{i,j} \in \{0, 1\}. \end{aligned}$$

This program is to find an appropriate assignment matrix X such that the errors for the equations

$$a_i = \sum_{j}^{l} x_{i,j} c_j, \qquad i = 1, \ldots, n$$

are minimized in l_∞ norm. We therefore name the problem P_∞. Of course, the errors can also be minimized in l_1 norm, and the program then becomes

$$\begin{aligned} (\mathrm{P}_1) \qquad \min\nolimits_{t,X} \qquad & \sum_{i}^{n} t_i \\ \textit{subject to} \quad & t_i \geq \sum_{j}^{l} x_{i,j} c_j - a_i \geq -t_i, \qquad i = 1, \ldots, n \\ & \sum_{i}^{n} x_{i,j} = 1, \qquad j = 1, \ldots, \\ & x_{i,j} \in \{0, 1\}. \end{aligned}$$

Here we name the problem P_1. Minimizing l_1 or l_∞ norm may have different effects on the solution to the problem. If the errors are not equal to zero at the solution, minimizing l_1 norm reduces total errors even if there are a few big errors, while minimizing l_∞ norm keeps the biggest error as small as possible. If an exact solution is found where the errors are all equal to zero, there are no differences in the solution qualities with l_1 or l_∞ norms. However, the solutions may still be different if there are multiple solutions, and the algorithm may perform quite differently as well. In the following sections, we show how the problems P_1 and P_∞ can be solved by using the CPLEX mixed-integer linear programming software.

3 CPLEX MIP Solver

CPLEX is a widely-used software package for solving integer, linear and quadratic programming problems. It was originally developed by Bixby [1] and later commercialized.

A linear programming problem usually is referred to as a optimization problem with a linear objective function and linear constraints. The following is a typical linear program:

$$\begin{array}{rl}
\min_{x_1,x_2,\ldots,x_n} & c_1x_1 + c_2x_2 \cdots + c_nx_n \\
\textit{subject to} & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\
& \cdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \\
& l_1 \leq x_1 \leq u_1 \\
& l_2 \leq x_2 \leq u_2 \\
& \cdots \\
& l_n \leq x_n \leq u_n
\end{array}$$

where $x_1, x_2, \ldots, x_n$ are continuous variables in general. The problem becomes a mixed-integer linear program if some variables are required to be integers. CPLEX has solvers for linear programs and, in particular, a mixed-integer solver for mixed-integer linear programs (MIP). Linear programming problems with continuous variables (LP) can be solved in polynomial time, while mixed-integer linear programs can be hard, including many NP-complete problems. The CPLEX MIP solver is built upon its LP solvers and uses a branch-and-bound strategy (see for example [8]), with the aid of cutting plane techniques, to find integer solutions. Two types of cuts, based on maximal cliques and minimal covers (see for example [5]), are generated when necessary to cut off non-integral solutions.

In the branch-and-bound algorithm for solving a MIP problem, a series of LP sub-problems is solved. A tree of subproblems is generated with each subproblem being a node of the tree. The root node is the LP relaxation of the original MIP. If the solution to the relaxation has integer variables taking fractional values, then one of such variables is chosen for branching, and two new subproblems are generated, each with more restrictive bounds for the branching variable. For example, for binary variables, one node will have the variable fixed at zero, and the other at one. The subproblem can result in an all-integer solution, an infeasible problem, or another fractional solution. If the solution is fractional, the process is repeated. A node is "cut off" when the objective function value of the subproblem associated with the node becomes higher than a known upper bound on the optimal value of the original MIP. Note that every all-integer solution gives such an upper bound. Also note that the node is "cut off" because further branching from the node will always generate subproblems with the same or even higher objective function values, and therefore, the whole branch can be removed.

The branch-and-bound algorithm searches for an integer solution for the MIP problem in the subproblem tree. In the worst case, it may need to examine the entire tree, which can be computationally prohibitive since there are exponentially many nodes in the tree with respect to the integer variables. In practice, the performance of the algorithm depends on specific search strategies such as how to select nodes for branching, how to obtain tighter bounds, etc. The CPLEX MIP solver provides several ways for users to choose among best possible search strategies. Readers are referred to the CPLEX Users' Manual [6] for more detailed descriptions.

For mixed-integer programming, in addition to branch-and-bound, cutting plane is another important solution technique. It is often used in combining with branch-and-bound. A cutting plane, or a cut, is an inequality added to a problem that restricts or cuts off non-integral solutions. Adding cuts usually reduces the number of branches that are needed to solve a mixed-integer program. The CPLEX MIP solver generates two types of cuts derived from maximal cliques and minimal covers (see [6] for more details).

4 Computational Results

We used the CPLEX MIP solver (CPLEX 6.5.2) to obtain the solutions for problems P_1 and P_∞. The program was run on a multi-processor SGI Origin 2000 computer with sixteen 300MHZ R12000 processors and 10GB main memory. However, only one processor was used at a time. In order to see how double digestion problems can be solved by our approach, we generated a set of random test problems by following a procedure suggested by Goldstein and Waterman in [3]. We assumed a set of DNA sequences of 100, 200, 300, 400, and 500 units. The unit here may correspond to one or more than one base pair. It does not matter what the actual unit is since the difficulty of the problem is correlated to the number and sizes of fragments, which is independent of the size of the unit. For each sequence, we simulate the first enzyme restriction by cutting the sequence at every unit with a probability p_1, and the second enzyme restriction with a probability p_2. After combining the two, we obtain a triple set of fragments a, b, and c. In order to cover certain amount of possibilities, we generated different sets of problems with $p_1 = p_2 = 0.2$, $p_1 = p_2 = 0.3$, $p_1 = p_2 = 0.4$, and $p_1 = p_2 = 0.5$. Obviously, more fragments are generated with increasing probabilities p_1 and p_2, and the difficulties of the problems may vary. We list the sizes of the fragment sets for all generated problems in Table 1. Note that if the fragments in c is to be matched to those in a, the corresponding integer program has $n \times l$ binary variables, where n is the number of fragments in a and l the number of fragments in c. In our test problems, the number of fragments in a or b ranges from 18 to 250, while that in c from 35 to 376. This implies that the corresponding integer programming problems can have up to 94,000 binary variables. An integer program with this many binary variables is often considered too large to be solvable even on powerful computers. However, as we can see in Table 2 to 5, the integer programs for our test problems have been solved very well by the CPLEX MIP solver.

Tables 2 to 5 contain the results obtained from using the CPLEX MIP solver on the test problems we have generated in Table 1. Note that for every triple sets a, b, and c, there are two integer programs to solve, one for matching c to a and another for c to b. We denote the two programs by $c \to a$ and $c \to b$. For each of these problems, we list in the tables the number of branch-and-bound nodes used in the CPLEX MIP solver and the time to obtain a solution to the problem. Table 2 contains the results for the problems solved as P_1 type programs. Table 3 contains the results for the problems solved as P_∞ type programs. These two sets of results were

Table 1: randomly generated problems

# units	$p_1 = p_2 = 0.2$	$p_1 = p_2 = 0.3$	$p_1 = p_2 = 0.4$	$p_1 = p_2 = 0.5$
100:	$\|a\| = 18$ $\|b\| = 22$ $\|c\| = 35$	$\|a\| = 27$ $\|b\| = 33$ $\|c\| = 49$	$\|a\| = 36$ $\|b\| = 42$ $\|c\| = 62$	$\|a\| = 48$ $\|b\| = 55$ $\|c\| = 75$
200:	$\|a\| = 33$ $\|b\| = 39$ $\|c\| = 61$	$\|a\| = 45$ $\|b\| = 59$ $\|c\| = 86$	$\|a\| = 66$ $\|b\| = 77$ $\|c\| = 115$	$\|a\| = 96$ $\|b\| = 102$ $\|c\| = 146$
300:	$\|a\| = 55$ $\|b\| = 58$ $\|c\| = 104$	$\|a\| = 79$ $\|b\| = 89$ $\|c\| = 148$	$\|a\| = 109$ $\|b\| = 114$ $\|c\| = 181$	$\|a\| = 145$ $\|b\| = 149$ $\|c\| = 227$
400:	$\|a\| = 71$ $\|b\| = 84$ $\|c\| = 137$	$\|a\| = 103$ $\|b\| = 124$ $\|c\| = 191$	$\|a\| = 142$ $\|b\| = 157$ $\|c\| = 241$	$\|a\| = 192$ $\|b\| = 197$ $\|c\| = 298$
500:	$\|a\| = 90$ $\|b\| = 115$ $\|c\| = 189$	$\|a\| = 135$ $\|b\| = 165$ $\|c\| = 256$	$\|a\| = 184$ $\|b\| = 211$ $\|c\| = 319$	$\|a\| = 242$ $\|b\| = 250$ $\|c\| = 376$

obtained with the initial fragment sets in the original orders. However, we have also tested the problems with the given fragment sets sorted in an ascending order. The results are listed in Tables 4 and 5.

From Tables 2 to 5 we can see that the CPLEX MIP solver solved the problems very well although many of their corresponding integer programs have large numbers of integer variables. The problems were solved within reasonable amount of time except for one that took about 10 hours. Comparing the results for P_1 and P_∞ types problems, the timings are comparable. The choice between solving P_1 or P_∞ should depend on practical needs. On the other hand, the results for problems with or without sorting the fragment data seem quite different. Most of the problems were solved faster when the fragment sets were sorted except for a couple of cases.

In general, the more nodes are there in the branch-and-bound procedure, the longer time the program will run. However, as we can see in Tables 2 to 5, the number of branch-and-bound nodes does not always correlate with the running time. Some problems required fewer nodes but longer time. The reason is that the cost of solving linear programs corresponding to different nodes can vary greatly.

5 Concluding Remarks

In this paper, we have considered a mixed-integer linear programming formulation for the double digestion problem and showed that with this formulation, the problem can be solved efficiently by using state-of-the-art mixed-integer programming techniques. In particular, we used the CPLEX mixed-integer linear programming software and obtained the exact solutions for a set of randomly-generated test problems.

We have focused on the problem of matching the double-digested fragments to the single-digested ones. The problem is formulated as a mixed-integer linear program minimizing the matching errors in either l_1 or l_∞ norm. The program is separated into two sub-programs. We have used the CPLEX MIP solver to solve both l_1 and l_∞ types of problems. The test problems were generated randomly to cover possible sizes of problems, lengths of fragments, and random fragment distribution. We have tested problems with hundreds of fragments and tens of thousands of binary variables in the corresponding integer programs. Many of the problems were solved in only a few minutes on a single SGI R12000 processor.

Several issues yet to be resolved in future work. One of them is the handling of the errors in the experimental data. If the measured lengths for the fragments are still integers, the mixed-integer programs will essentially be at the same level of difficulty as those we have tested. If the lengths are real numbers, however, the solutions to the mixed-integer programs may take longer time to find, We plan to conduct more computational experiments to study this and other issues.

Another issue is the multiple solutions to the mixed-integer programs and hence the double digestion problem. As discussed in literature, this problem cannot be solved unless additional knowledge of the sequence is given. In any case, the solutions to the two sub-programs for fragment matching show only how the single and double-digested fragments match but not how they should be ordered. The fragments need to be examined together and put into sensible orders. Fortunately, this can be done in an efficient and systematic fashion [13].

When using the CPLEX MIP solver for the test problems, we have so far not taken advantages that CPLEX provides for improving the performance of the solver, such as using user-supplied initial solutions, heuristic upper bounds, and better node selection orders, etc. We have only used default settings for our test. It is highly probable that by taking these advantages, one should be able to exploit the problem structure more effectively and achieve even better computational results than what we have just observed.

References

[1] R. E. Bixby, *Implementing the Simplex Method: The Initial Basis*, Technical Report TR90-32, Department of Computational and Applied Mathematics, Rice University, Houston, TX, 1990.

[2] W. M. Fitch, T. F. Smith, and W. W. Ralph, *Mapping the Order of DNA Restriction Fragments*, Gene, 22, 1983, pp. 19-29.

[3] L. Goldstein and M. S. Waterman, *Mapping DNA by Stochastic Relaxation*, Adv. Appl. Math., 8, 1987, pp. 194-207.

[4] A. V. Grigorjev and A. A. Mironov, *Mapping DNA by Stochastic Relaxation: A New Approach to Fragment Sizes*, Applic. Biosci., , pp. 107-111.

[5] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithm and Combinatorial Optimization*, Springer, 1987.

[6] ILOG Inc., *Using the CPLEX Callable Library*, ILOG Inc., 1997.

[7] M. Krawczak, *Algorithms for the Restriction-Site Mapping of DNA Molecules*, Proc. Natl. Acad. Sci. USA, 85, 1988, pp. 7298-7301.

[8] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, Inc., 1988.

[9] W. R. Pearson, *Automatic Construction of Restriction Site Maps*, Nucleic Acids Res., 10, 1984, 217-227.

[10] W. Schmitt and M. S. Waterman, *Multiple Solutions of DNA Restriction Mapping Problems*, Adv. Appl. Math., 12, 1991, pp. 412-427.

[11] M. Stefik, *Inferring DNA Structures from Segmentation Data*, Artif. Intell., 11, 1978, pp. 85-114.

[12] P. Tuffery, P. Dessen, C. Mugnier, and S. Hazout, *Restriction Map Construction Using Complete Sentences Compatibility Algorithm*, Comput. Applic. Biosci., 4, pp. 103-110.

[13] M. S. Waterman, Introduction to Computational Biology: Sequences, Maps, and Genomes, Chapman Hall, 1995.

[14] L. W. Wright, J. B. Lichter, J. Reinitz, M. A. Shifman, K. K. Kidd, and P. L. Miller, *Computer-Assisted Restriction Mapping: An Integrated Approach to Handling Experimental Uncertainty*, CABIOS, Vol. 10, No. 4, 1994, pp. 443-450.

Table 2: Test results for P_1 type problems

# units	$p_1 = p_2 = 0.2$	$p_1 = p_2 = 0.3$	$p_1 = p_2 = 0.4$	$p_1 = p_2 = 0.5$
100:	$c \to a$ $b\&b : 45$ $time : 1s$	$c \to a$ $b\&b : 23$ $time : 1s$	$c \to a$ $b\&b : 21$ $time : 2s$	$c \to a$ $b\&b : 21$ $time : 2s$
	$c \to b$ $b\&b : 88$ $time : 1s$	$c \to b$ $b\&b : 33$ $time : 1s$	$c \to b$ $b\&b : 38$ $time : 2s$	$c \to b$ $b\&b : 20$ $time : 3s$
200:	$c \to a$ $b\&b : 430$ $time : 8s$	$c \to a$ $b\&b : 206$ $time : 11s$	$c \to a$ $b\&b : 73$ $time : 13s$	$c \to a$ $b\&b : 145$ $time : 28s$
	$c \to b$ $b\&b : 718$ $time : 12s$	$c \to b$ $b\&b : 762$ $time : 27s$	$c \to b$ $b\&b : 191$ $time : 31s$	$c \to b$ $b\&b : 76$ $time : 24s$
300:	$c \to a$ $b\&b : 529$ $time : 32s$	$c \to a$ $b\&b : 400$ $time : 1m10s$	$c \to a$ $b\&b : 463$ $time : 2m5s$	$c \to a$ $b\&b : 193$ $time : 2m35s$
	$c \to b$ $b\&b : 4156$ $time : 2m4s$	$c \to b$ $b\&b : 602$ $time : 1m56s$	$c \to b$ $b\&b : 2369$ $time : 7m0s$	$c \to b$ $b\&b : 65$ $time : 1m9s$
400:	$c \to a$ $b\&b : 2390$ $time : 3m13s$	$c \to a$ $b\&b : 1097$ $time : 6m23s$	$c \to a$ $b\&b : 541$ $time : 6m49s$	$c \to a$ $b\&b : 160$ $time : 5m7s$
	$c \to b$ $b\&b : 2669$ $time : 3m36s$	$c \to b$ $b\&b : 1228$ $time : 6m24s$	$c \to b$ $b\&b : 845$ $time : 9m49s$	$c \to b$ $b\&b : 324$ $time : 9m19s$
500:	$c \to a$ $b\&b : 1235$ $time : 3m33s$	$c \to a$ $b\&b : 7648$ $time : 46m36s$	$c \to a$ $b\&b : 662$ $time : 15m47s$	$c \to a$ $b\&b : 144$ $time : 7m43s$
	$c \to b$ $b\&b : 6616$ $time : 17m14s$	$c \to b$ $b\&b : 18862$ $time : 1h30m42s$	$c \to b$ $b\&b : 14661$ $time : 1h43m13s$	$c \to b$ $b\&b : 1238$ $time : 38m5s$

Table 3: Test results for P_∞ type problems

# units	$p_1 = p_2 = 0.2$	$p_1 = p_2 = 0.3$	$p_1 = p_2 = 0.4$	$p_1 = p_2 = 0.5$
100:	$c \to a$ $b\&b : 11$ $time : 1s$	$c \to a$ $b\&b : 43$ $time : 1s$	$c \to a$ $b\&b : 28$ $time : 2s$	$c \to a$ $b\&b : 21$ $time : 2s$
	$c \to b$ $b\&b : 97$ $time : 2s$	$c \to b$ $b\&b : 50$ $time : 1s$	$c \to b$ $b\&b : 37$ $time : 2s$	$c \to b$ $b\&b : 21$ $time : 3s$
200:	$c \to a$ $b\&b : 226$ $time : 4s$	$c \to a$ $b\&b : 225$ $time : 9s$	$c \to a$ $b\&b : 182$ $time : 37s$	$c \to a$ $b\&b : 87$ $time : 35s$
	$c \to b$ $b\&b : 4070$ $time : 38s$	$c \to b$ $b\&b : 1086$ $time : 39s$	$c \to b$ $b\&b : 148$ $time : 44s$	$c \to b$ $b\&b : 81$ $time : 33s$
300:	$c \to a$ $b\&b : 545$ $time : 25s$	$c \to a$ $b\&b : 177$ $time : 1m20s$	$c \to a$ $b\&b : 138$ $time : 1m16s$	$c \to a$ $b\&b : 97$ $time : 1m50s$
	$c \to b$ $b\&b : 198$ $time : 18s$	$c \to b$ $b\&b : 237$ $time : 2m4s$	$c \to b$ $b\&b : 391$ $time : 4m17s$	$c \to b$ $b\&b : 78$ $time : 1m16s$
400:	$c \to a$ $b\&b : 2293$ $time : 2m27s$	$c \to a$ $b\&b : 269$ $time : 2m42s$	$c \to a$ $b\&b : 268$ $time : 5m1s$	$c \to a$ $b\&b : 280$ $time : 18m35s$
	$c \to b$ $b\&b : 3751$ $time : 4m13s$	$c \to b$ $b\&b : 1727$ $time : 24m6s$	$c \to b$ $b\&b : 1071$ $time : 19m36s$	$c \to b$ $b\&b : 324$ $time : 11m34s$
500:	$c \to a$ $b\&b : 208$ $time : 53s$	$c \to a$ $b\&b : 334$ $time : 10m13s$	$c \to a$ $b\&b : 401$ $time : 40m31s$	$c \to a$ $b\&b : 144$ $time : 10m46s$
	$c \to b$ $b\&b : 2193$ $time : 12m10s$	$c \to b$ $b\&b : 1520$ $time : 1h24m42s$	$c \to b$ $b\&b : 414$ $time : 17m11s$	$c \to b$ $b\&b : 821$ $time : 2h44m25s$

Table 4: Test results for P_1 type problems with sorted fragments

# units	$p_1 = p_2 = 0.2$	$p_1 = p_2 = 0.3$	$p_1 = p_2 = 0.4$	$p_1 = p_2 = 0.5$
100:	$c \to a$ $b\&b : 232$ $time : 2s$	$c \to a$ $b\&b : 52$ $time : 1s$	$c \to a$ $b\&b : 26$ $time : 2s$	$c \to a$ $b\&b : 33$ $time : 3s$
	$c \to b$ $b\&b : 1198$ $time : 5s$	$c \to b$ $b\&b : 33$ $time : 1s$	$c \to b$ $b\&b : 41$ $time : 3s$	$c \to b$ $b\&b : 19$ $time : 2s$
200:	$c \to a$ $b\&b : 446$ $time : 7s$	$c \to a$ $b\&b : 380$ $time : 15s$	$c \to a$ $b\&b : 196$ $time : 30s$	$c \to a$ $b\&b : 42$ $time : 20s$
	$c \to b$ $b\&b : 198$ $time : 5s$	$c \to b$ $b\&b : 461$ $time : 21s$	$c \to b$ $b\&b : 162$ $time : 25s$	$c \to b$ $b\&b : 21$ $time : 18s$
300:	$c \to a$ $b\&b : 668$ $time : 37s$	$c \to a$ $b\&b : 169$ $time : 50s$	$c \to a$ $b\&b : 83$ $time : 58s$	$c \to a$ $b\&b : 57$ $time : 1m12s$
	$c \to b$ $b\&b : 1251$ $time : 1m5s$	$c \to b$ $b\&b : 1115$ $time : 2m58s$	$c \to b$ $b\&b : 104$ $time : 1m5s$	$c \to b$ $b\&b : 91$ $time : 1m40s$
400:	$c \to a$ $b\&b : 1462316$ $time : 9h59m31s$	$c \to a$ $b\&b : 1345$ $time : 6m28s$	$c \to a$ $b\&b : 415$ $time : 5m13s$	$c \to a$ $b\&b : 94$ $time : 4m14s$
	$c \to b$ $b\&b : 4000$ $time : 4m57s$	$c \to b$ $b\&b : 2341$ $time : 15m48s$	$c \to b$ $b\&b : 437$ $time : 5m50s$	$c \to b$ $b\&b : 199$ $time : 7m0s$
500:	$c \to a$ $b\&b : 1120$ $time : 4m8s$	$c \to a$ $b\&b : 1694$ $time : 1m21s$	$c \to a$ $b\&b : 159$ $time : 5m24s$	$c \to a$ $b\&b : 119$ $time : 7m19s$
	$c \to b$ $b\&b : 992$ $time : 4m10s$	$c \to b$ $b\&b : 846$ $time : 13m32s$	$c \to b$ $b\&b : 342$ $time : 10m18s$	$c \to b$ $b\&b : 289$ $time : 15m9s$

Table 5: Test results for P_∞ type problems with sorted fragments

# units	$p_1 = p_2 = 0.2$	$p_1 = p_2 = 0.3$	$p_1 = p_2 = 0.4$	$p_1 = p_2 = 0.5$
100:	$c \to a$ $b\&b : 25$ $time : 1s$	$c \to a$ $b\&b : 51$ $time : 1s$	$c \to a$ $b\&b : 21$ $time : 2s$	$c \to a$ $b\&b : 21$ $time : 2s$
	$c \to b$ $b\&b : 102$ $time : 1s$	$c \to b$ $b\&b : 77$ $time : 2s$	$c \to b$ $b\&b : 41$ $time : 2s$	$c \to b$ $b\&b : 27$ $time : 3s$
200:	$c \to a$ $b\&b : 76$ $time : 2s$	$c \to a$ $b\&b : 191$ $time : 8s$	$c \to a$ $b\&b : 87$ $time : 18s$	$c \to a$ $b\&b : 54$ $time : 19s$
	$c \to b$ $b\&b : 152$ $time : 4s$	$c \to b$ $b\&b : 584$ $time : 29s$	$c \to b$ $b\&b : 52$ $time : 13s$	$c \to b$ $b\&b : 50$ $time : 21s$
300:	$c \to a$ $b\&b : 212$ $time : 12s$	$c \to a$ $b\&b : 154$ $time : 33s$	$c \to a$ $b\&b : 83$ $time : 51s$	$c \to a$ $b\&b : 70$ $time : 1m14s$
	$c \to b$ $b\&b : 453$ $time : 27s$	$c \to b$ $b\&b : 254$ $time : 1m27s$	$c \to b$ $b\&b : 107$ $time : 51s$	$c \to b$ $b\&b : 106$ $time : 1m27s$
400:	$c \to a$ $b\&b : 308$ $time : 40s$	$c \to a$ $b\&b : 572$ $time : 5m15s$	$c \to a$ $b\&b : 193$ $time : 5m39s$	$c \to a$ $b\&b : 69$ $time : 4m8s$
	$c \to b$ $b\&b : 741$ $time : 1m36s$	$c \to b$ $b\&b : 198$ $time : 2m47s$	$c \to b$ $b\&b : 175$ $time : 6m13s$	$c \to b$ $b\&b : 133$ $time : 6m42s$
500:	$c \to a$ $b\&b : 222$ $time : 1m16s$	$c \to a$ $b\&b : 321$ $time : 11m56s$	$c \to a$ $b\&b : 197$ $time : 13m33s$	$c \to a$ $b\&b : 124$ $time : 13m5s$
	$c \to b$ $b\&b : 1342$ $time : 6m32s$	$c \to b$ $b\&b : 261$ $time : 15m42s$	$c \to b$ $b\&b : 182$ $time : 9m4s$	$c \to b$ $b\&b : 131$ $time : 17m8s$