MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Use of Primal-Dual Technique in the Network Algorithm for Two-Way Contingency Tables

Taiji SUZUKI, Satoshi AOKI,
and Kazuo MUROTA

METR 2004–28 May 2004

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: http://www.i.u-tokyo.ac.jp/mi/mi-e.htm

The METR technical reports are published as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

Use of Primal-Dual Technique in the Network Algorithm for Two-Way Contingency Tables *

Taiji Suzuki[†‡], Satoshi Aoki[†§] and Kazuo Murota[†¶]

May, 2004

Abstract

The network algorithm of Mehta and Patel is one of the most efficient algorithms to execute a generalized Fisher's exact test in two-way contingency tables. In this article an efficient algorithm for solving the longest path problem in the network algorithm is proposed. The efficiency of the proposed algorithm relies on the primal dual technique for convex network flow problems and the use of maximum likelihood estimate as the initial value. The algorithm is more efficient than previous algorithms, especially for large contingency tables (say, $r \times c > 50$). This work is intended to indicate a fruitful interplay between statistics and combinatorical optimization.

1 Introduction

The network algorithm of Mehta and Patel [7] is currently the most efficient algorithm for computing the exact p-value of the generalized Fisher's exact test in two-way contingency tables. The algorithm is basically a dynamic programming algorithm representing the recurrence relation in the form of a network, and the algorithm avoids useless enumerations by solving certain subproblems, shortest-path and longest-path problems in the network. Some methods for approximately solving these subproblems have been proposed successfully by Mehta and Patel [7] and Aoki [2]. Joe's method [6] for computing the exact optimal values of these subproblems has resulted in a further improvement of efficiency in computation time and required memory

*This work is supported by a Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan. The third author is supported by PRESTO, JST.

[†]Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Tokyo 113-8656, Japan

[‡]s-taiji@sat.t.u-tokyo.ac.jp

[§]aoki@stat.t.u-tokyo.ac.jp

[¶]murota@mist.i.u-tokyo.ac.jp

of the network algorithm. This is implemented as FEXACT, a source code in Fortran77, by Clarkson, Fan and Joe [3].

In this article we propose a new algorithm that exactly solves the longest-path problem in the network algorithm. Our algorithm consists of solving a min-cost integer-flow problem with a discrete convex cost function by the standard primal-dual framework [1] [9]. The algorithm sets the initial flow and potential with the aid of a statistical knowledge of the maximum likelihood estimate. This is a key for the efficiency of the algorithm as they are guaranteed to be near the optimal values. Our method for the longest-path problem compares favorably with Joe's method both in computation time and required memory. Whereas the required computer memory and CPU time grow exponentially with the problem size in Joe's algorithm, this is not the case with our algorithm, in which the CPU time grows almost linearly with the problem size. It is noted, however, that our algorithm and Joe's enumerate the same set of paths since both algorithms perform trimming on the basis of the exact longest-path length.

In this article two different kinds of networks are involved. One is the network due to Mehta and Patel to represent the dynamic programming structure and the other is a network, to be introduced in Section 3.1, for computing the longest-path length with the aid of the maximum likelihood estimate. This work is intended to indicate a fruitful interplay between statistics and combinatorial optimization.

The outline of this paper is as follows. In Section 2 a brief outline of the network algorithm is given. In Section 3, the proposed algorithm is described, whereas the detail of the validity is explained in Appendix A. Section4 presents computational results, and Section 5 provides conclusion and discussion.

2 Network Algorithm of Mehta and Patel

In this section we show a brief outline of the network algorithm of Mehta and Patel. We refer the reader to Mehta and Patel [7] [8], Joe [6], Clarkson, Fan and Joe [3] for technical details of the algorithm.

Let $X = (x_{ij})_{1 \le i \le r, 1 \le j \le c}$ be an $r \times c$ contingency table which is assumed to have nonnegative elements. Let $R_i = \sum_{j=1}^{c} x_{ij}$ be the ith row sum, and $C_j = \sum_{i=1}^{r} x_{ij}$ be the jth column sum, and put $N = \sum_{i=1}^{r} R_i = \sum_{j=1}^{c} C_j$. We denote by $\mathcal{F}$ the set of all $r \times c$ contingency tables with the same row and column sums as the given X. Under the null hypothesis of row and column independence, the conditional probability of observing $Y = (y_{ij}) \in \mathcal{F}$ is described as

$$P(Y) = \frac{\prod_{i=1}^{r} R_i! \prod_{j=1}^{c} C_j!}{N! \prod_{i=1}^{r} \prod_{j=1}^{c} y_{ij}!} = D \left(\prod_{i=1}^{r} \prod_{j=1}^{c} y_{ij}! \right)^{-1},$$

where $D = (\prod_{i=1}^{r} R_i! \prod_{j=1}^{c} C_j!)/N!$. Freeman and Halton [4] defined the p-value for the conditional test of independence as $p = \sum_{Y \in \mathcal{T}} P(Y)$, where $\mathcal{T} = \{Y \in \mathcal{F} \mid P(Y) \leq P(X)\}$ for the observed table X. If $p \leq \alpha$, for a prespecified critical point α (for example, $\alpha = 0.05$), we reject the hypothesis of independence.

In the network algorithm, $\mathcal{F}$ is identified with a network consisting of $c+1$ stages labelled successively $c, c-1, \cdots, 0$. Stage k contains nodes labelled by $(k : \boldsymbol{R}_k) = (k : R_{1k}, \cdots, R_{rk})$, which represents the stage number k and the row sum vector $\boldsymbol{R}_k$ from the first to the kth column of a certain element in $\mathcal{F}$. Stage c has node $(c : R_1, \cdots, R_r)$ only and Stage 0 has node $(0 : 0, \cdots, 0)$ only. A directed arc emanates from node $(k : R_{1k}, \cdots, R_{rk})$ to node $(k-1 : R_{1,k-1}, \cdots, R_{r,k-1})$ if and only if $R_{ik} - R_{i,k-1} \geq 0$ for all $i = 1, \cdots, r$. Then a path from $(c : R_1, \cdots, R_r)$ to $(0 : 0, \cdots, 0)$ has a one-to-one correspondence with an element in $\mathcal{F}$, say, $Y = (y_{ij})$ with $y_{ik} = R_{ik} - R_{i,k-1}$. The length of an arc with $(y_{1k}, \cdots, y_{rk})$ is defined as $(\prod_{i=1}^{r} y_{ik}!)^{-1}$ and the length of a path is defined to be the product of the lengths of the arcs on the path. In particular, the length of a path from $(c : R_1, \cdots, R_r)$ to $(0 : 0, \cdots, 0)$ that is identified with $Y \in \mathcal{F}$ equals $P(Y)/D$. Figure 1 shows an example of the constructed network for an observed table $X = \begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 2 & 2 & 2 \\ \hline \end{array}$, which is taken from [7]. The dotted path in Figure 1 represents X.

The network algorithm proceeds visiting nodes along arcs from the start node $(c : R_1, \cdots, R_r)$. At each visited node, say $N_k = (k : R_{1k}, \cdots, R_{rk})$, let $\mathcal{P}$ be the path that is traversed from the start node $(c : R_1, \cdots, R_r)$ to N_k, and PAST be the length of $\mathcal{P}$. The algorithm also calculates the shortest path length SP and longest path length LP of a path from node N_k to $(0 : 0, \cdots, 0)$. By using SP, LP and PAST, useless traversing of arcs can be eliminated and hence the computation time can be reduced as follows. If $\mathrm{LP} \cdot \mathrm{PAST} \cdot D \leq P(X)$, all the paths sharing the traversed path $\mathcal{P}$ contribute to the p-value, with the total contribution being equal to $\mathrm{PAST} \cdot D \cdot \frac{(\sum_{j=1}^{k} C_j)!}{\prod_{i=1}^{r} R_{ik}! \cdot \prod_{j=1}^{k} C_j!}$. If $\mathrm{SP} \cdot \mathrm{PAST} \cdot D > P(X)$, none of such paths contribute to the p-value. In either case, the further path traversing beyond $\mathcal{P}$ may be *trimmed*. Note that PAST is not determined by N_k alone but depends on the traversed path $\mathcal{P}$, whereas SP and LP are associated with N_k.

For the exactness of the calculated p-value we may use an upper bound of LP or a lower bound of SP in place of the exact values of LP or SP. The degree of approximation, however, much affects the computational efficiency. That is, tighter bounds lead to higher computational efficiency as a result of earlier path trimming. A number of methods have been proposed for computing SP and LP or their bounds [2], [6], [7]. In particular, Joe's

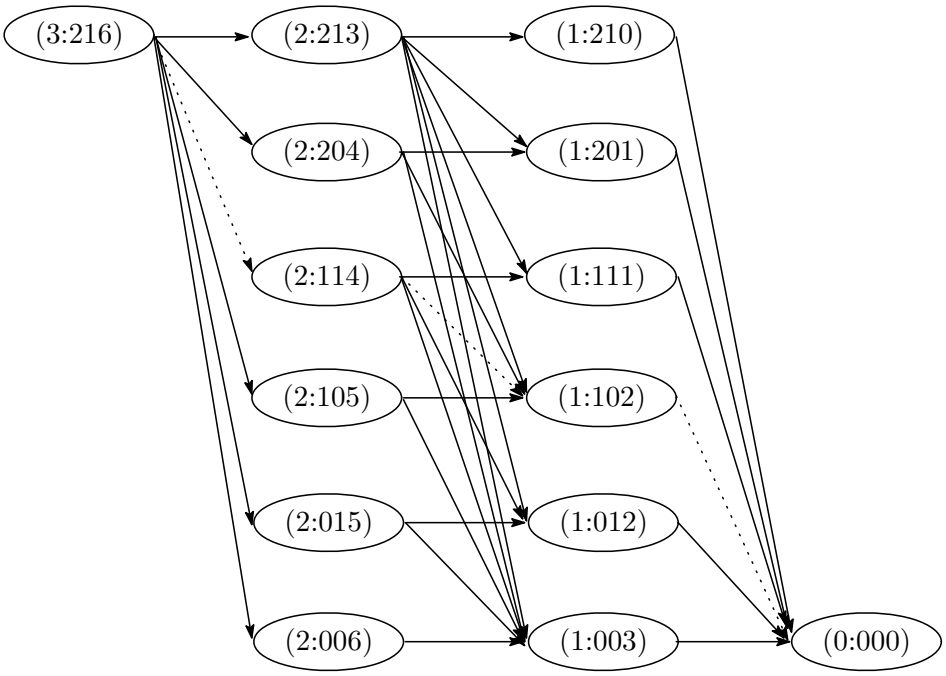

Figure 1: Example [7] of the network with $R_1 = 2, R_2 = 1, R_3 = 6, C_1 = 3, C_2 = 3, C_3 = 3$.

methods [6] for computing exact SP and LP are implemented as FEXACT in [3], in which LP is computed by judicious enumeration of submatrices that satisfy a necessary condition of optimality. Although FEXACT (f3xact, the subroutine of FEXACT for computing LP, more precisely) is sufficiently fast for smaller contingency tables, the required computer memory and CPU time grow exponentially with the problem size, as the method is based on enumeration of candidate submatrices. In contrast, our algorithm, to be described in Section 3.1, is free from such exponentially increasing resource requirement.

3 New Algorithm Based on Min-Cost Flow

In this section a new algorithm for computing the exact value of the longest-path length LP is proposed. It consists of solving a sequence of min-cost integer-flow problems by the primal-dual method. The outline of the algorithm is as follows. A good initial pair of flow and potential is constructed from the maximum likelihood estimate under the model of no interaction between the row and column. The pair satisfies the complementarity condition for optimality, although the flow does not necessarily meet the supply-demand requirement. Then, according to the primal-dual framework, the

algorithm repeats augmenting flows and updating potentials while maintaining the complementarity condition. By virtue of the good initial value the number of iterations in the primal-dual algorithm is bounded by $r \times c$.

3.1 Formulation to a Min-Cost Flow Problem

Let $\mathbf{R}$ be the set of real numbers, $\mathbf{Z}$ be the set of integer numbers, and $\mathbf{Z}_+$ be the set of nonnegative integers. We describe the algorithm at node $(c : R_1, \cdots, R_r)$ at stage c, where the adaptation to other nodes at general stages should be obvious.

The problem of computing LP is written in the following form:

Problem 1

$$\begin{aligned}
\text{Max.} \quad & \tilde{\Gamma}(Y) = \left(\sum_{i=1}^{r} \sum_{j=1}^{c} y_{ij}! \right)^{-1}, \\
\text{s. t.} \quad & \sum_{i=1}^{r} y_{ij} = C_j, \quad \sum_{j=1}^{c} y_{ij} = R_i, \\
& y_{ij} \in \mathbf{Z}_+ \quad (1 \le i \le r, 1 \le j \le c),
\end{aligned}$$

where $Y = (y_{ij})$. We assume without loss of generality that $R_i > 0$ $(1 \le \forall i \le r)$ and $C_j > 0$ $(1 \le \forall j \le c)$.

To formulate the above problem to a convex optimization problem we convert the objective function $\tilde{\Gamma}(Y)$ to $\Gamma(Y) = -\log(\tilde{\Gamma}(Y)) = \sum_{i=1}^{r} \sum_{j=1}^{c} f(y_{ij})$ with $f : \mathbf{Z} \to \mathbf{R} \cup \{+\infty\}$ defined by

$$f(y) = \begin{cases} \log(y!) & (y \ge 0), \\ +\infty & (\text{otherwise}). \end{cases}$$

Note that f is a convex function over integers in the sense that

$$f(y+1) + f(y-1) - 2f(y) = \log \frac{y+1}{y} > 0 \qquad (y \ge 1). \tag{1}$$

Then Problem 1 is equivalent to the following convex integer programming problem:

Problem 2

$$\text{Min.} \quad \Gamma(Y) = \sum_{i=1}^{r} \sum_{j=1}^{c} f(y_{ij}), \tag{2}$$

$$\text{s. t.} \quad \sum_{i=1}^{r} y_{ij} = C_j, \quad \sum_{j=1}^{c} y_{ij} = R_i, \tag{3}$$

$$y_{ij} \in \mathbf{Z} \quad (1 \le i \le r, 1 \le j \le c). \tag{4}$$

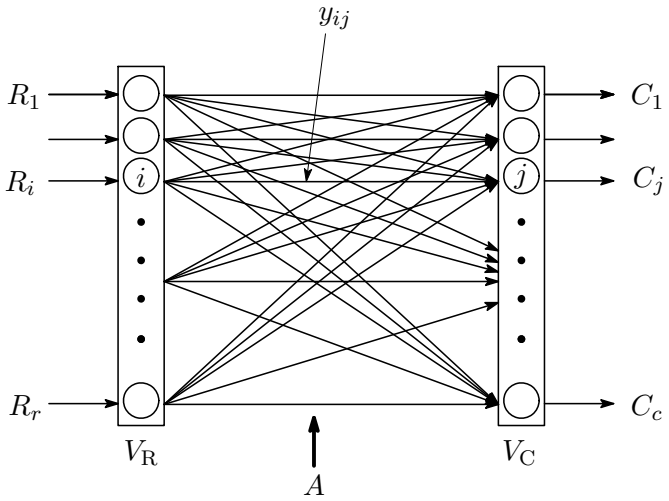

Figure 2: Network $G(V, A)$

This problem can easily be identified as a convex integer transportation problem, i.e., a min-cost integer-flow problem on a bipartite graph with a convex cost function. The underlying graph, say, $G(V, A)$ is a complete bipartite graph on the vertex set $V = V_{\mathrm{R}} \cup V_{\mathrm{C}}$ that consists of the set of the row numbers $V_{\mathrm{R}} = \{1, 2, \cdots, r\}$ and the set of the column numbers $V_{\mathrm{C}} = \{1, 2, \cdots, c\}$ of the contingency table. The arcs are directed from V_{R} to V_{C}, i.e., $A = \{(i, j) \mid i \in V_{\mathrm{R}}, j \in V_{\mathrm{C}}\}$. The graph $G(V, A)$ is illustrated in Figure 2. Each arc $(i, j) \in A$ has cost $f(y_{ij})$ for the flow vector $y = (y_{ij})_{(i,j)\in A}$. Our aim is to minimize the total cost $\sum_{(i,j)\in A} f(y_{ij})$ under the supply-demand condition (3).

A necessary and sufficient condition for optimality, so-called *potential criterion*, is available in network flow theory. By a potential we mean $p = (p_{\mathrm{R}}, p_{\mathrm{C}})$ with $p_{\mathrm{R}} : V_{\mathrm{R}} \to \mathbf{R}$ and $p_{\mathrm{C}} : V_{\mathrm{C}} \to \mathbf{R}$. The *subdifferential* of $f : \mathbf{Z} \to \mathbf{R}$ at $z \in \mathbf{Z}_+$ is defined as

$$\partial f(z) = \{q \in \mathbf{R} \mid q \cdot d + f(z) \leq f(z + d),\ \forall d \in \mathbf{Z}\}.$$

We put $\log 0 = -\infty$ by convention.

Lemma 1 *For $z \in \mathbf{Z}_+$,*

$$\partial f(z) = \{q \in \mathbf{R} \mid \log z \leq q \leq \log(z + 1)\}. \tag{5}$$

Proof. By the convexity (1), we see that $q \in \partial f(z)$ if and only if

$$q \cdot d + f(z) \leq f(z + d) \quad \text{for} \quad d = \pm 1.$$

Substituting (1) yields (5). ■

The optimality condition for Problem 2 is given as follows.

Theorem 1 (Potential criterion) *For a feasible integer flow* $y : A \to \mathbf{Z}$ *in Problem 2, the two conditions* (OPT) *and* (POT) *below are equivalent.*

- (OPT) y *is an optimal integer flow.*
- (POT) *There exists a potential* $p = (p_{\mathrm{R}}, p_{\mathrm{C}}) : V_{\mathrm{R}} \cup V_{\mathrm{C}} \to \mathbf{R}$ *such that* $p_{\mathrm{C}}(j) - p_{\mathrm{R}}(i) \in \partial f(y_{ij})$ *for all* $(i, j) \in A$.

Proof. This fact, implicit in Minoux [9], seems to be a folklore. This follows, for example, from *Integrality Theorem for Flows* and *Network Equilibrium Theorem* in Rockafellar [11]. This theorem is also a special case of Theorem 9.16 in Murota [10]. ■

3.2 Initial Value

Good initial values of y and p that satisfy (POT) can be constructed from the maximum likelihood estimate under the hypothesis of independence. These initial values play an important role in this algorithm because these are guaranteed to be near the optimal value.

Lemma 2 (Initial Value) *A pair of flow* y *and potential* $p = (p_{\mathrm{R}}, p_{\mathrm{C}})$ *defined by*

$$y_{ij} = \left\lfloor \frac{R_i C_j}{N} \right\rfloor, \quad p_{\mathrm{R}}(i) = -\log \frac{R_i}{N}, \quad p_{\mathrm{C}}(j) = \log C_j \quad (i \in V_{\mathrm{R}}, j \in V_{\mathrm{C}}) \tag{6}$$

satisfies the condition (POT).

Proof. Obviously $p_{\mathrm{C}}(j) - p_{\mathrm{R}}(i) = \log(R_i C_j / N)$ lies between $\log y_{ij} = \log(\lfloor R_i C_j / N \rfloor)$ and $\log(y_{ij} + 1) = \log(\lfloor R_i C_j / N \rfloor + 1)$. This implies (POT) by Lemma 1. ■

The initial y in (6) does not necessarily meet (3), but satisfies $\sum_{j=1}^{c} y_{ij} \leq R_i$, $\sum_{i=1}^{r} y_{ij} \leq C_j$. Then it is natural to increase the flow and reduce the residual $N - \sum_{i,j} y_{ij}$ while keeping y and p satisfying (POT).

Remark 1 The initial values (6) are derived from the maximum likelihood estimate under the model of no interaction between the row and column. The detailed statistical discussion is found in Aoki [2]. Joe [5] also makes a convincing argument about the relevance of the maximum likelihood estimate with reference to Schur concavity.

3.3 Flow Augmentation

Starting with the initial value of y and p given by (6), the algorithm iterates the flow augmentation by the standard primal-dual method [1] adapted to convex cost functions. The flow augmentation process reduces the flow deficiency by more than one unit at every iteration. If $N - \sum_{i,j} y_{ij} = 0$, then the algorithm stops.

In the primal-dual method we employ an auxiliary network (G_{yp}, l_{yp}). The node set of the graph G_{yp} is defined as $\tilde{V} = \{s\} \cup V$, where s is a source node. The arc set is $A_{yp} = A^*_{yp} \cup B^*_{yp} \cup C^*_{yp}$, where

$$
\begin{aligned}
A^*_{yp} &= \{(i,j) \mid i \in V_{\mathrm{R}}, j \in V_{\mathrm{C}}\},\\
B^*_{yp} &= \{(j,i) \mid y_{ij} > 0, i \in V_{\mathrm{R}}, j \in V_{\mathrm{C}}\},\\
C^*_{yp} &= \{(s,i) \mid \sum_{j=1}^{c} y_{ij} < R_i, i \in V_{\mathrm{R}}\}.
\end{aligned}
$$

The function $l_{yp} : A_{yp} \to \mathbf{R}$, representing arc lengths, is defined by

$$
l_{yp}(a) = \begin{cases}
\log(y_{ij}+1) + p_{\mathrm{R}}(i) - p_{\mathrm{C}}(j) & (a = (i,j) \in A^*_{yp}),\\
-\log(y_{ij}) - p_{\mathrm{R}}(i) + p_{\mathrm{C}}(j) & (a = (j,i) \in B^*_{yp}),\\
0 & (a \in C^*_{yp}).
\end{cases}
$$

The procedure can be described as follows.

Step 0 Set the initial values of flow y and potential $p = (p_{\mathrm{R}}, p_{\mathrm{C}})$ by:

$$
y_{ij} = \left\lfloor \frac{R_i C_j}{N} \right\rfloor,\ p_{\mathrm{R}}(i) = -\log \frac{R_i}{N},\ p_{\mathrm{C}}(j) = \log C_j
$$
$(\forall i \in V_{\mathrm{R}}, \forall j \in V_{\mathrm{C}})$.

Step 1 Set $Q = \{j \in V_{\mathrm{C}} \mid \sum_{i=1}^{r} y_{ij} < C_j\}$. If $Q = \emptyset$, then output y and p as optimal flow and optimal potential, and exit. Otherwise, construct the auxiliary network (G_{yp}, l_{yp}).

Step 2 In (G_{yp}, l_{yp}) solve the single source shortest path problem from the start node s, to obtain the shortest distance $d_{\mathrm{R}}(i)$ for $i \in V_{\mathrm{R}}$ and $d_{\mathrm{C}}(j)$ for $j \in V_{\mathrm{C}}$. Let T be the set of arcs that can be contained in some shortest path tree from s.

Step 3 Modify the potential $p = (p_{\mathrm{R}}, p_{\mathrm{C}})$ as follows:

$$
\begin{cases}
p_{\mathrm{R}}(i) & \leftarrow p_{\mathrm{R}}(i) + d_{\mathrm{R}}(i) \quad (\forall i \in V_{\mathrm{R}}),\\
p_{\mathrm{C}}(j) & \leftarrow p_{\mathrm{C}}(j) + d_{\mathrm{C}}(j) \quad (\forall j \in V_{\mathrm{C}}).
\end{cases}
$$

Step 4 Let $v \in Q$ be a node which is reachable from s along T. Let P_v be a shortest path from s to v, and augment a unit flow along P_v as follows:

$$y_{ij} \leftarrow \begin{cases} y_{ij} + 1 & ((i,j) \in P_v \cap A^*_{yp}), \\ y_{ij} - 1 & ((j,i) \in P_v \cap B^*_{yp}), \\ y_{ij} & (\text{otherwise}). \end{cases}$$

Then delete arcs from T as follows:

$$T \leftarrow T \setminus (P_v \cap (A^*_{yp} \cup B^*_{yp})).$$

If $\sum_{i=1}^{r} y_{iv} = C_v$, delete v from Q. If there remains a node $j \in Q$ which is reachable from s along T, repeat Step 4, else go to Step 1.

The number of iterations in our algorithm is bounded by the size of problem as follows.

Lemma 3 *The number of iterations is no more than $r \times c$.*

Proof. Let α be the number of iterations. Since the algorithm augments at least one unit flow, we have

$$\alpha \leq N - \sum_{i=1}^{r} \sum_{j=1}^{c} \lfloor R_i C_j / N \rfloor.$$

Using $\sum_{i=1}^{r} R_i = \sum_{j=1}^{c} C_j = N$ we obtain

$$\alpha \leq \sum_{i=1}^{r} \sum_{j=1}^{c} (R_i C_j / N - \lfloor R_i C_j / N \rfloor) \leq \sum_{i=1}^{r} \sum_{j=1}^{c} 1 = r \times c.$$

■

In actual executions of the algorithm, the number of iterations is often much smaller than $r \times c$.

Remark 2 In the network algorithm we have to solve Problem 2 for many different sets of marginals $\boldsymbol{R}$ and $\boldsymbol{C}$. For two problems with near marginals, we can make use of the solution of one of the two problems in solving the other problem. Let us consider the special case that two problems have identical column marginals $\boldsymbol{C}$ and near row marginals $\boldsymbol{R}$ and $\tilde{\boldsymbol{R}}$ that satisfy

$$\tilde{R}_i = \begin{cases} R_i + 1 & (i = u) \\ R_i - 1 & (i = v) \\ R_i & (\text{otherwise}) \end{cases}$$

for some u and v. In this case, if the optimal flow y and the optimal potential p under row marginals $\boldsymbol{R}$ and column marginals $\boldsymbol{C}$ is known, only one flow augmentation is required to compute LP under row marginals $\tilde{\boldsymbol{R}}$ and column marginals $\boldsymbol{C}$: augment a unit flow from u to v.

It may be worth mentioning that, when $R_u = 0$, the potential $p_{\mathrm{R}}(u)$ is not defined in our description of the algorithm because of the assumption of $R_i > 0$ ($\forall i \in V_{\mathrm{R}}$). In this case $p_{\mathrm{R}}(u)$ should be set to satisfy (POT), for example, as $p_{\mathrm{R}}(u) = \max\{p_{\mathrm{C}}(j) \mid j \in V_{\mathrm{C}}, C_j \neq 0\}$.

4 Computational Results

We compared the CPU time for solving Problem 2 between our algorithm and FEXACT which is a source code of FORTRAN77 implemented by Clarkson, Fan and Joe [3] based on the theoretical analysis of Joe [6]. FEXACT is adopted for the comparison because it is known to be more efficient than previously published methods using approximate upper bounds of LP; see [3] for comparison with Mehta and Patel's method [7], and [12] for that with Aoki's [2]. The programs were coded in C or C++ and compiled by C++ compiler. We have converted by f2c the FORTRAN77 source code of FEXACT to a C source code. Both algorithms have been executed on a machine with Intel Pentium M processor 1300MHz and 512MB memory. The results are displayed in Figure 3, in which the CPU time for solving Problem 2 by our algorithm and FEXACT and the stack size required in FEXACT, both in log-scale, are plotted against $r \times c$, the size of problems. The observed data with a common row size r are connected by line segments.

Figure 3 shows that, while FEXACT requires CPU time of almost exponential order with respect to the size $r \times c$, our algorithm requires CPU time of almost linear order. For other test data, the two algorithms behave in a similar way, but with different multiplicative factors. Our algorithm is uniformly faster, although there are some other cases in which FEXACT is fast at small sized problems (for example $r \times c \leq 50$). At $r \times c > 50$, in particular, our algorithm is almost always faster in all our computational experiments. According to Figure 3, for example, at $r \times c = 60$ our algorithm is about 100 times faster and at $r \times c = 140$ about 10^7 times faster than FEXACT.

As for the memory requirement, our algorithm has a decisive advantage over FEXACT (Figure 3). FEXACT uses a stack for computing LP, and the size of the stack increases exponentially with the problem size; for example, the size of the stack is about 1000 at $r \times c = 70$ and about 10^5 at $r \times c = 140$. FEXACT uses a stack to enumerate contingency tables via dynamic programming. Though the enumeration is kept minimum by exploiting the theoretical results of Joe [6], the stack gets exponentially large with the problem size, and even worse, the size of the stack cannot be predicted in

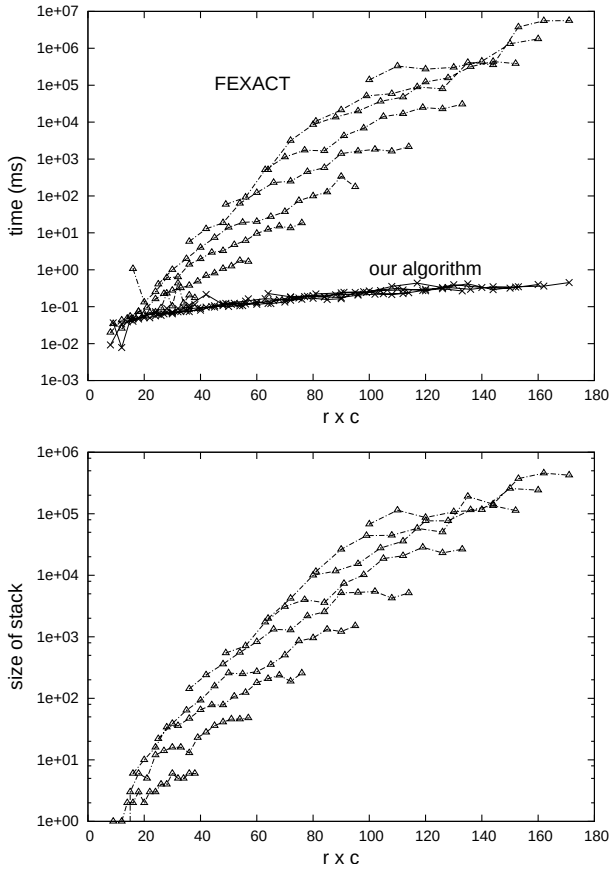

Figure 3: CPU time for solving Problem 2 by FEXACT and our algorithm and required stack size in FEXACT, plotted against problem size $r \times c$.

advance. Our algorithm, on the other hand, does not involve any dynamic memory allocation and hence is much stabler in memory requirement.

As for the overall CPU time of network algorithm, the difference between our algorithm and FEXACT is small in the range of the problem sizes which the network algorithm can treat. This is because in this range the CPU time for computing LP does not differ so much between the two algorithms and because it is not dominant in the overall CPU time (Table 1). In view of the above computational results, it can be said, however, that when the treatable problem size is made much larger by the development of computer power in the future, the use of our algorithm is obviously more effective with respect to CPU time and memory requirement.

5 Discussion

In the cases that the difference among row (or column) sums is small, FEXACT obtains LP immediately by a heuristic method based on Theorem 4 of Joe [6]. It is easily suspected that a hybrid algorithm of our algorithm and FEXACT may be useful in such cases. That is, at the entrance of the algorithm, if the difference among row or column sums is small, we attempt at applying the heuristic method, and if the heuristic method is not applicable, we use our algorithm. Our additional computational experiments showed, however, that whereas this hybrid algorithm is effective in solving Problem 2, it does not yield improvement in the overall CPU time of network algorithm. This is because the time saved by the heuristic method is often less than the time consumed by testing whether the heuristic method is applicable or not. We have thus confirmed the use of our algorithm is most practical with respect to memory requirement and CPU time.

The efficiency of our algorithm relies on the fact that the maximum likelihood estimate, when discretized, serves as a good initial value for the maximization problem in discrete variables. We believe, with optimism, that this is also the case in fairly general situations, for example, in maximizing a Schur concave function that has a statistical meaning; see Joe [6].

Appendix A: Primal-Dual Method

This appendix explains the detail of the primal-dual method.

For a given flow y, we call a node $i \in V_{\mathrm{R}}$ (or $j \in V_{\mathrm{C}}$) *saturated* if $\sum_{j=1}^{c} y_{ij} = R_i$ (or $\sum_{i=1}^{r} y_{ij} = C_j$), else we call it *non-saturated*.

Concerning the auxiliary network (G_{yp}, l_{yp}) the next lemma holds.

Lemma 4 *y and p satisfy* (POT)*, if and only if $l_{yp}(a) \geq 0$ $(\forall a \in A_{yp})$.*

We call the process to augment a flow keeping satisfying (POT) *augmentation process*. Before we state the detail of the augmentation process, we

Table 1: CPU time of the network algorithm using our algorithm and FEXACT

Problems	Contingency table	p-value	CPU time (s)	
			Our algorithm	FEXACT
1	0 2 3 4 1 1 4 5 0 4 4 2 3 0 2 4 5 4 2 4 3	0.2599	0.035	0.036
2	3 0 4 0 2 0 5 3 0 1 5 0 2 2 2 2 0 1 0 3 0 0 4 1	0.0116	0.059	0.060
3	8 3 3 2 2 1 3 8 9 1 1 2 2 1 2 3 7 3 3 1 0 1 0 3 1 0 2 1	0.0460	4.739	4.708
4	2 1 0 2 3 1 2 2 1 2 3 2 2 1 0 0 2 1 2 1 0 1 2 2 1 0 0 1 0 0 0 3 0 1 0 1 1 2 1 0 0 0	0.8296	0.748	1.001
5	7 15 2 1 0 2 1 9 0 3 2 1 0 1 2 3 2 2 2 0 1 1 1 1 3 2 2 1 3 1 1 3 0 3 3	0.0004	232.524	232.606
6	3 3 2 3 3 1 0 1 2 1 2 1 0 0 0 3 0 0 1 0 1 2 2 3 0 3 2 0 1 3 3 2 2 3 0 2 0 2 3 2 2 0 2 0 1 2 2 2 1 1 0 1 2 0 0 1	0.0337	835.306	840.291
7	2 3 0 3 0 0 1 3 0 0 3 1 0 1 2 1 0 1 1 1 1 3 2 3 1 1 2 3 2 3 1 3 1 2 3 0 0 1 0 3 2 0 2 1 2 0 1 2 0 0 0 0 1 2 1 2	0.0821	188.567	198.416
8	3 1 3 1 3 3 0 0 1 2 0 2 1 1 0 1 0 2 0 3 0 3 0 0 3 1 0 0 1 3 3 3 3 0 1 1 2 1 0 3 0 0 2 0 1 0 1 1 0 1 1 0 0 2 0 2	0.0029	171.598	176.218
10	1 3 3 1 0 1 0 3 1 4 2 1 3 1 1 3 2 1 3 0 1 3 0 3 0 1 2 1 2 5 6 3 2 0 0 1 2 6 0 2 0 0 2 1 0 0 2 0 2 0 2 0 0 1 0 2	0.0008	5473.700	5511.070

show the outline of the process. The augmentation process consists of two phases: a potential update and a flow update. A potential update is a pre-process that enables us to augment a flow keeping y and p satisfying (POT). In a potential update process the algorithm obtains basically a shortest path tree T on (G_{yp}, l_{yp}), the root of which is the source node s, and at each node $v \in V$, $p(v)$ is added by a distance from s to v in order to satisfy (POT). Next in a flow update process the algorithm augments a unit flow from s to each non-saturated node in V_{C} along T. For a potential update process it is guaranteed that (POT) is still satisfied after flow update (see Lemma 5). Then the algorithm modifies (G_{yp}, l_{yp}) according to the updated y and p, and iterates the same process on the modified (G_{yp}, l_{yp}) until all the nodes in V_{R} (and V_{C}) are saturated.

Next, we state the detail of the augmentation process. Let T be the shortest path tree on (G_{yp}, l_{yp}) from s. When y and p satisfy (POT), by Lemma 4, a single-source shortest path problem from s can be solved by an efficient algorithm such as Dijkstra's. Then the algorithm modifies the potential $p = (p_{\mathrm{R}}, p_{\mathrm{C}})$ as follows:

$$\begin{cases} p_{\mathrm{R}}(i) & \leftarrow p_{\mathrm{R}}(i) + d_{\mathrm{R}}(i) \quad (\forall i \in V_{\mathrm{R}}), \\ p_{\mathrm{C}}(j) & \leftarrow p_{\mathrm{C}}(j) + d_{\mathrm{C}}(j) \quad (\forall j \in V_{\mathrm{C}}), \end{cases} \tag{7}$$

where $d_{\mathrm{R}}(i)$ and $d_{\mathrm{C}}(j)$ are distances from s to $i \in V_{\mathrm{R}}$ and $j \in V_{\mathrm{C}}$ respectively.

Next the algorithm enters the flow update phase. It picks a non-saturated node $v \in V_{\mathrm{C}}$ which is reachable from s along T, and augments a unit flow along the shortest path P_v from s to v. To augment a unit flow along P_v is described concretely as

$$y_{ij} \leftarrow \begin{cases} y_{ij} + 1 & ((i,j) \in P_v \cap A^*_{yp}), \\ y_{ij} - 1 & ((j,i) \in P_v \cap B^*_{yp}), \\ y_{ij} & (\text{otherwise}). \end{cases} \tag{8}$$

Then delete the arcs from T as follows:

$$T \leftarrow T \setminus (P_v \cap (A^*_{yp} \cup B^*_{yp})). \tag{9}$$

Until there exists no non-saturated node $v \in V_{\mathrm{C}}$ which is reachable by traversing T from s, the algorithm repeats the flow augmentation and the arc deletion from T.

Lemma 5 *The modified y and p also satisfy the condition* (POT).

Proof. Let y and y' be the flows before and after modification. Let $p = (p_{\mathrm{R}}, p_{\mathrm{C}})$ and $p' = (p'_{\mathrm{R}}, p'_{\mathrm{C}})$ be the potential before and after modification, where $p'_{\mathrm{R}}(i) = p_{\mathrm{R}}(i) + d_{\mathrm{R}}(i)$ and $p'_{\mathrm{C}}(j) = p_{\mathrm{C}}(j) + d_{\mathrm{C}}(j)$.

(a) For $(i,j) \in A^*_{yp}$ such that $y'_{ij} = y_{ij}$. Since $d_{\mathrm{R}}(i)+l_{yp}(i,j)-d_{\mathrm{C}}(j) \geq 0$, we have

$$\begin{aligned} & d_{\mathrm{R}}(i) + l_{yp}(i,j) - d_{\mathrm{C}}(j) \geq 0 \\ \Leftrightarrow & \log(y_{ij}+1) + p_{\mathrm{R}}(i) - p_{\mathrm{C}}(j) + d_{\mathrm{R}}(i) - d_{\mathrm{C}}(j) \geq 0 \\ \Leftrightarrow & \log(y'_{ij}+1) + p'_{\mathrm{R}}(i) - p'_{\mathrm{C}}(j) \geq 0. \end{aligned} \tag{10}$$

If $(j,i) \in B^*_{yp} (\Leftrightarrow y_{ij} > 0)$, since $d_{\mathrm{C}}(j) + l_{yp}(j,i) \geq d_{\mathrm{R}}(i)$, we have

$$\begin{aligned} & d_{\mathrm{C}}(j) + l_{yp}(j,i) - d_{\mathrm{R}}(i) \geq 0 \\ \Leftrightarrow & -\log(y_{ij}) - p_{\mathrm{R}}(i) + p_{\mathrm{C}}(j) + d_{\mathrm{C}}(j) - d_{\mathrm{R}}(i) \geq 0 \\ \Leftrightarrow & -\log(y'_{ij}) - p'_{\mathrm{R}}(i) + p'_{\mathrm{C}}(j) \geq 0, \end{aligned} \tag{11}$$

else if $(j,i) \notin B^*_{yp} (\Leftrightarrow y_{ij} = 0)$,

$$-\log(y'_{ij}) - p'_{\mathrm{R}}(i) + p'_{\mathrm{C}}(j) = \infty > 0. \tag{12}$$

(b) For $(i,j) \in A^*_{yp}$ such that $y'_{ij} = y_{ij}+1$. In this case, $d_{\mathrm{R}}(i)+l_{yp}(i,j) = d_{\mathrm{C}}(j)$ because $(i,j) \in P_v$. Then

$$\begin{aligned} & d_{\mathrm{R}}(i) + l_{yp}(i,j) - d_{\mathrm{C}}(j) = 0 \\ \Leftrightarrow & \log(y_{ij}+1) + p_{\mathrm{R}}(i) - p_{\mathrm{C}}(j) + d_{\mathrm{R}}(i) - d_{\mathrm{C}}(j) = 0 \\ \Leftrightarrow & -\log(y'_{ij}) - p'_{\mathrm{R}}(i) + p'_{\mathrm{C}}(j) = 0, \end{aligned} \tag{13}$$

and the last expression obviously implies

$$\log(y'_{ij}+1) + p'_{\mathrm{R}}(i) - p'_{\mathrm{C}}(j) \geq 0. \tag{14}$$

(c) For $(j,i) \in B^*_{yp}$ such that $y'_{ij} = y_{ij} - 1$.
In this case $d_{\mathrm{C}}(j) + l_{yp}(j,i) = d_{\mathrm{R}}(i)$ because $(j,i) \in P_v$. Then

$$\begin{aligned} & d_{\mathrm{C}}(j) + l_{yp}(j,i) - d_{\mathrm{R}}(i) = 0 \\ \Leftrightarrow & -\log(y_{ij}) - p_{\mathrm{R}}(i) + p_{\mathrm{C}}(j) - d_{\mathrm{R}}(i) + d_{\mathrm{C}}(j) = 0 \\ \Rightarrow & -\log(y'_{ij}) - p'_{\mathrm{R}}(i) + p'_{\mathrm{C}}(j) \geq 0, \end{aligned} \tag{15}$$

the expression of which obviously implies

$$\log(y'_{ij}+1) + p'_{\mathrm{R}}(i) - p'_{\mathrm{C}}(j) = 0. \tag{16}$$

From (a),(b) and (c) we obtain

$$\begin{aligned} & \forall (i,j) \in A, \\ & \log y'_{ij} \leq p'_{\mathrm{C}}(j) - p'_{\mathrm{R}}(i) \leq \log(y'_{ij}+1) \\ \Leftrightarrow & p'_{\mathrm{C}}(j) - p'_{\mathrm{R}}(i) \in \partial f(y'_{ij}). \end{aligned} \tag{17}$$

■

The algorithm reconstructs the auxiliary network (G_{yp}, l_{yp}) based on modified y and p, and repeats the augmentation process until all nodes in V are saturated. When all nodes in V are saturated, the algorithm stops and $Y = (y_{ij})$ is an optimal contingency table.

The following lemma guarantees the validity of the proposed algorithm.

Lemma 6 *All nodes in V are reachable from s on (G_{yp}, l_{yp}) till the algorithm stops. So the potential is always finite at each node.*

Proof. A non-saturated node $i \in V_{\mathrm{R}}$ is reachable from s because $(s, i) \in C^*_{yp}$. If there exists a node $i \in V_{\mathrm{R}}$ which is non-saturated, any $j \in V_{\mathrm{C}}$ is reachable from s because for any $j \in V_{\mathrm{C}}$ there is an arc $(i, j) \in A^*_{yp}$. For a saturated node $i \in V_{\mathrm{R}}$ there exists $j \in V_{\mathrm{C}}$ which satisfies $y_{ij} > 0$ because $\sum_{j=1}^{c} y_{ij} = R_i > 0$. Then there exists $(j, i) \in B^*_{yp}$. Since any $j \in V_{\mathrm{C}}$ is reachable, any saturated $i \in V_{\mathrm{R}}$ is also reachable.

Appendix B: Further Computational Results

We show further computational results in this appendix. In the following figures, CPU time for solving Problem 2 by FEXACT and our algorithm and required stack size in FEXACT, both in log-scale, are plotted against problem size $r \times c$. We generated test problems randomly.

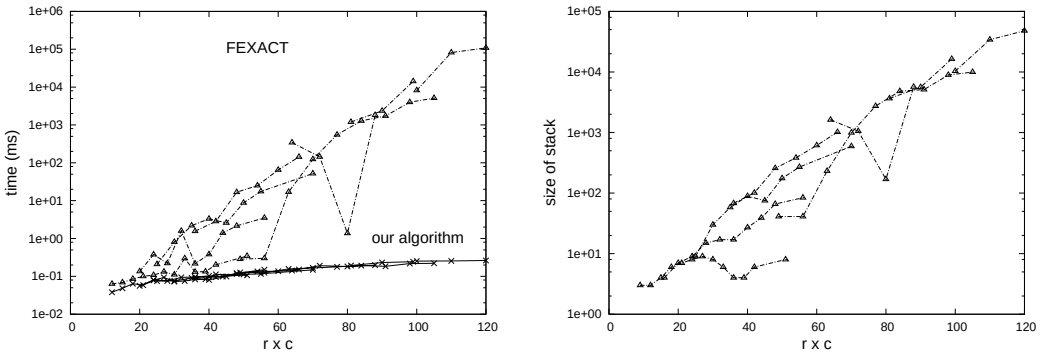

Figure A1: CPU time and size of stack for problem set 1.

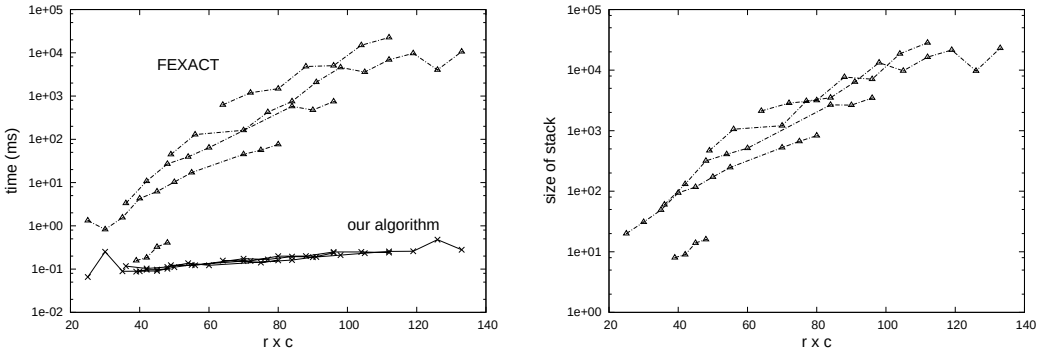

Figure A2: CPU time and size of stack for problem set 2.

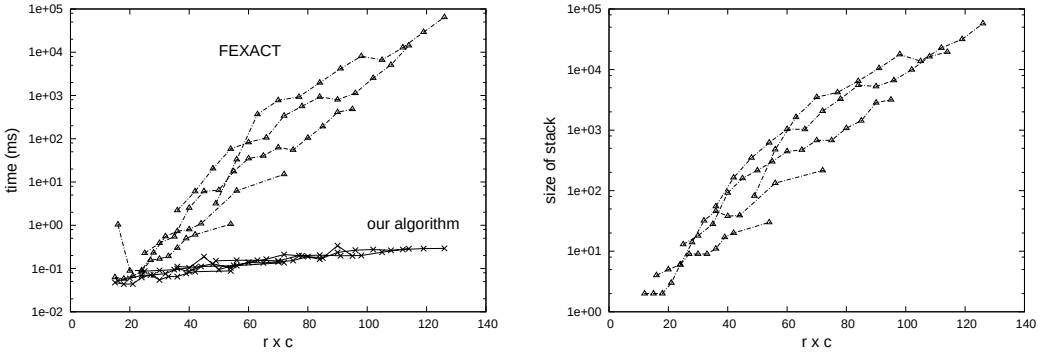

Figure A3: CPU time and size of stack for problem set 3.

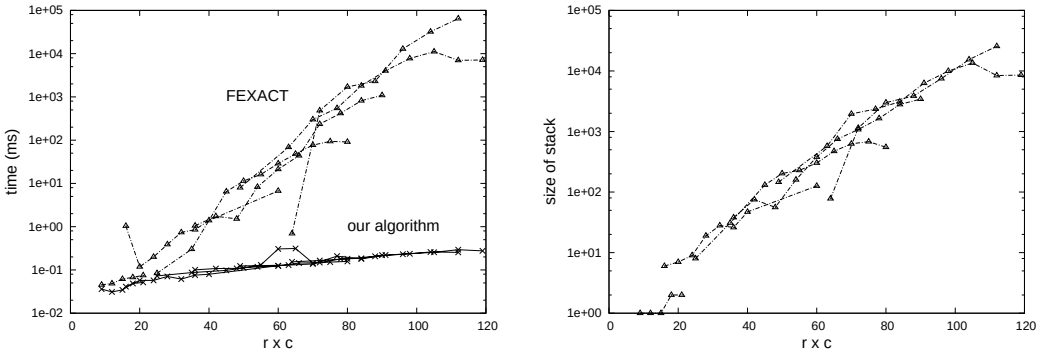

Figure A4: CPU time and size of stack for problem set 4.

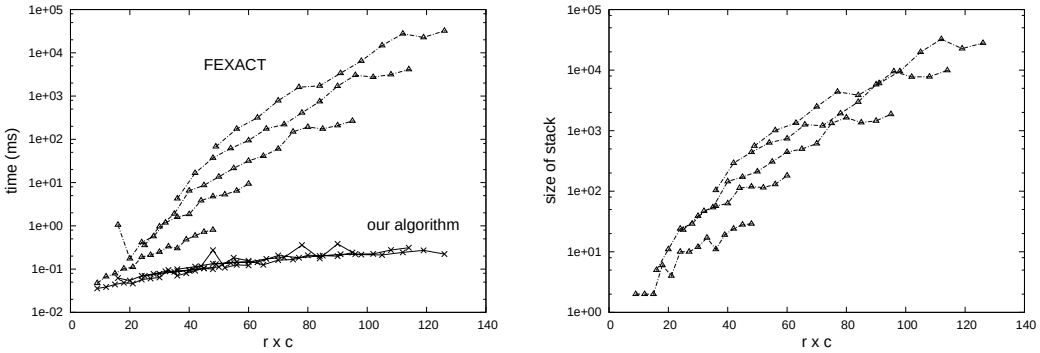

Figure A5: CPU time and size of stack for problem set 5.

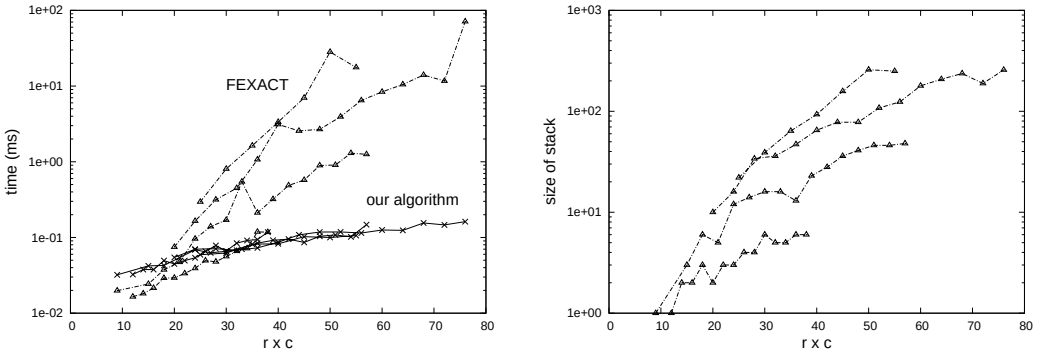

Figure A6: CPU time and size of stack for problem set 6.

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin: *Network Flows —Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, 1993.

[2] S. Aoki: Improving path trimming in a network algorithm for Fisher's exact test in two-way contingency tables, *Journal of Statistical Computation and Simulation*, **72** (2002), 205–216.

[3] D. B. Clarkson, Y. Fan and H. Joe: A remark on Algorithm 643: FEXACT: an algorithm for performing Fisher's exact test in $r \times c$ contingency tables, *ACM Transactions on Mathematical Software*, **19** (1993), 484–488.

[4] G. H. Freeman and J. H. Halton: Note on an exact treatment of contingency, goodness of fit and other problems of significance, *Biometrika*, **38** (1951), 141–149.

[5] H. Joe: An ordering of dependence for contingency tables, *Linear Algebra and Its Applications*, **70** (1985), 89–103.

[6] H. Joe: Extreme probabilities for contingency tables under row and column independence with application to Fisher's exact test, *Communications in Statistics—Theory and Methods*, **17** (1988), 3677–3685.

[7] C. R. Mehta and N. R. Patel: A network algorithm for performing Fisher's exact test in $r \times c$ contingency tables, *Journal of the American Statistical Association*, **78** (1983), 427–434.

[8] C. R. Mehta and N. R. Patel: Algorithm 643 FEXACT: a FORTRAN subroutine for Fisher's exact test on unordered $r \times c$ contingency tables, *ACM Transactions on Mathematical Software*, **12** (1986), 154–161.

[9] M. Minoux: Solving integer minimum cost flows with separable convex cost objective polynomially, *Mathematical Programming Study*, **26** (1986), 237–239.

[10] K. Murota: *Discrete Convex Analysis*, SIAM, Philadelphia, PA, 2003.

[11] R. T. Rockafellar: *Network Flows and Monotropic Optimization*, John Wiley and Sons, New York, 1984.

[12] T. Suzuki: *Analysis of Contingency Tables by Algorithms in Discrete Convex Analysis* (in Japanese), Graduation thesis, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, 2004.