数理解析研究所講究録 1375 巻 2004 年 120-125

データの論理的解析における正関数発見の並列化

片岡 博幸 (Hiroyuki Kataoka)* 小野 廣隆 (Hirotaka Ono)†
定兼 邦彦 (Kunihiko Sadakane)† 山下 雅史 (Masafumi Yamashita)†

1 はじめに

与えられた大量のデータから意味のある情報を関数として取り出すことはデータマイニング, 知識発見 [3], データの論理的解析 (LAD)[5] などの基本的なテーマである. しかし, 大容量デバイスなどの普及により, 我々が取得可能な情報の量は爆発的に増加し, その処理時間が膨大となり問題となってきている. これを解決するために近年研究されているのが処理の並列化である [1, 2]. 本稿では, 正データである n 次元 0-1 ベクトル x に対しては $f(x) = 1$, 負データベクトル x に対しては $f(x) = 0$ を満たす論理関数, f が正関数の性質を満たすかどうかの判定問題に注目し, これを解く処理を並列化したアルゴリズムを提案・評価する. さらにこれを PC クラスタ上で実装し, ランダムデータに対する計算実験を行なうことにより, 性能評価を行なう. また, データが正性 (単調性) を満たさない場合についても, データに最も適合する正関数を発見する並列アルゴリズムを提案する.

2 準備

以下では論理関数の知識を仮定する. n 変数論理関数 f が, 任意の $x, y \in \{0,1\}^n$ に対し, $x \leq y \Rightarrow f(x) \leq f(y)$ を満すとき**正関数** (関数のクラス : $\mathcal{C}_P$) であると言い, 様々な現象の構造を示すのによく用いられる. 正データの集合 $T \subseteq \{0,1\}^n$ と負のデータ集合 $F \subseteq \{0,1\}^n$ の対 (T, F) を**部分定義論理関数** (pdBf) と呼ぶ. 論理関数 $f : \{0,1\}^n \to \{0,1\}$ が $f(v) = 1$ を満たすベクトル $v \in \{0,1\}^n$ を f の**真ベクトル**, $f(v) = 0$ を満たすベクトル v を f の**偽ベクトル**といい, その集合をそれぞれ $T(f)$, $F(f)$ と記す. 定義より, $T(f) \cap F(f) = \emptyset$ かつ $T(f) \cup F(f) = \{0,1\}^n$ である. さらに条件 $T(f) \supseteq T, F(f) \supseteq F$ を満すとき, f を (T, F) の**拡大** (extension) であるという. 拡大は一般に多数存在ため, 可能な全ての拡大の中から, (T, F) の論理的説明としてどの拡大を選ぶかは重要な問題である. 選択基準の一つとして, ある関数の性質を拡大が満足するか (ある関数のクラスに属するか) どうかといったことが挙げられる. 関数のクラスを限定することは次のような意味を持つ : 第一に, あらかじめ分析したい現象に何らかの構造が存在することがわかっている場合, そのような構造を反映したクラスに属する関数を拡大として選ぶのは自然である. 第二に, 得られた関数自体をさらに解析することにより, 新たな知識を得ようとする試みがあるが, 関数のクラスを限定することにより, この解析が容易になるなどの利点がある.

例えば, 関数のクラスを $\mathcal{C}$ としたとき, ここでの問題は次のようになる.

Problem EXTENSION($\mathcal{C}$)	
Input:	A pdBf (T, F), ただし $T, F \subseteq \{0,1\}^n$.
Question:	(T, F) の拡大 $f \in \mathcal{C}$ は存在するか?

当然のことながら, この問題に常に解が存在するとは限らない. このような場合, EXTENSION($\mathcal{C}$) をより一般化した, 最も適合する拡大を求める問題が考えられる.

Problem BEST-FIT($\mathcal{C}$)	
Input:	A pdBf (T, F).
Output:	次の性質を満たす (T^*, F^*) とその拡大
	1. $T^* \cap F^* = \emptyset, T^* \cup F^* = T \cup F$
	2. (T^*, F^*) は拡大 $f \in \mathcal{C}$ を持つ
	3. $\lvert T^* \cap F \rvert + \lvert T \cap F^* \rvert$ は最小

データには通常, 完全に信頼できるものではなく測定ミスや判断ミスなどがある場合や, 様々な例外を含むこと

*九州大学 大学院システム情報科学府情報工学専攻 (Department of Computer Science and Communication Engineering,Graduate School of Information Science and Electrical Engineering, Kyushu University), hiroyuki@tcslab.csce.kyushu-u.ac.jp
†九州大学 大学院システム情報科学研究院情報工学部門 (Department of Computer Science and Communication Engineering,Graduate School of Information Science and Electrical Engineering, Kyushu University), {ono, mak, sada}@csce.kyushu-u.ac.jp

が多い. この問題は, そういった状況下でのデータ解析を考える上で重要な意味を持つ.

本稿では, 3節で EXTENSION($\mathcal{C}_\mathcal{P}$) について, PC クラスタを用いた並列化によって解いた計算時間を示し, 台数効果などの結果を示す. 4節では BEST-FIT($\mathcal{C}_\mathcal{P}$) を解く 2 近似の並列アルゴリズムを説明する.

3 EXTENSION($\mathcal{C}_\mathcal{P}$) の並列化

3.1 PARA-COMPARE(T, F)

本節では EXTENSION($\mathcal{C}_\mathcal{P}$) を並列化して解くことを議論する. EXTENSION($\mathcal{C}_\mathcal{P}$) について次の定理が知られている.

定理 1 [6] pdBf(T, F) が拡大 $f \in C_P$ を持つための必要十分条件は, $a \leq b$ を満たす $a \in T, b \in F$ が存在しないことである.

この定理から, 全ての $a \in T, b \in F$ の対を比較することにより, EXTENSION($\mathcal{C}_\mathcal{P}$) の解が得られる.

Algorithm COMPARE(T, F)

1. 全ての $a \in T, b \in F$ に対して, $a \leq b$ であるかどうかを判定. 存在すれば No, しなければ Yes を出力.

COMPARE(T, F) の計算量は $O(n|T||F|)$ である.

この問題を N 台の計算機から成る PC クラスタを用いて解くことを考える. PARA-COMPARE(T, F) は (T, F) を N 等分したものを比較後, 他のプロセッサと交換する.

このアルゴリズムの正当性は定理 1 の条件を全てチェックしていることから明らかである. また計算時間は以下の通りである : P_i についての各 COMPARE(T_i, F_i) で $O(n|T||F|/N^2)$ かかり, それが N 回呼び出されることから, 全体の計算時間は $O(n|T||F|/N)$ + (通信時間) となる. 元の COMPARE(T, F) の計算時間が $O(n|T||F|)$ であることから, 通信時間が相対的に十分小さい場合, この並列アルゴリズムにより最適な台数効果が見込めることとなる.

Algorithm PARA-COMPARE(T, F)

1. P_1 は T, F を $T_i, F_i (T = \bigcup T_i, F = \bigcup F_i)$ へ N 等分し, 各 $P_i (i = 1, \ldots, N)$ に振り分ける.
2. j:=i;
3. 各 P_i は必要ならば P_{i-1} からデータを受け取り, COMPARE($T_i, F_{j'}$) (ただし $j' \equiv i + j \pmod N$) を解き, $j := j + 1$.
4. $j = N$ となったら停止. $j < N$ なら 3 へ戻る.
5. COMPARE($T_i, F_{j'}$) の返り値が全て Yes なら Yes, それ以外は No を出力.

PARA-COMPARE(T, F) に従って実装した実験結果を示す. 本研究での詳細な実験環境と実験方法を次の通りである. 実験方法の (注) にもある通り, データ数が台数で割り切れない場合は実装上の理由からデータ数を削っている. これはデータ数全体からみれば, ほとんど無視できる数なので結果への影響はないと見なしている.

実験環境	
OS	Debian 2.20
CPU	Pentium 2.26GHz
メモリ	512MB
通信ライブラリ	pvm3.4
コンパイラ	gcc 2.95.2

実験方法	
データ数	4000000($\|T\| = \|F\| = 2000000$)
変数の数	22
クラスタ台数	最大 10 台
計測方法	実時間 (関数 gettimeofday を使用)
試行回数	5 回

(注) データ数がクラスタ台数で割り切れない場合 (例えば 7 台), 実装上の理由から 2000000 以下の最大倍数を使用 (7 台の時は 1999998).

使用されるデータは以下で示すように単調増加の性質を満たしかつランダムに作成した. ただし, T, F, L はデータの集合とする.

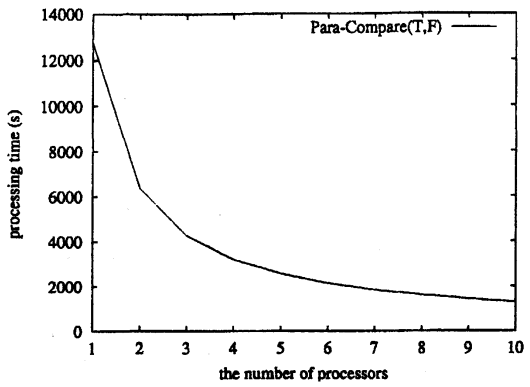

図 1: algorithm1:台数-処理時間

データ作成アルゴリズム

1. $T = F = L = \emptyset$.
2. ランダムにデータベクトル c を作成
3. $l \in L$ と比較

3-1. $c > l$ ならば $T = T \cup \{c\}$

3-2. $c < l$ ならば $F = F \cup \{c\}$

3-3. それ以外ならば $L = L \cup \{c\}$

** 2-3 を必要回数繰り返し終了

4. T, F を出力する.

図 1 はプロセッサ台数に応じて PARA-COMPARE(T, F) の 5 回の試行実行時間の平均をグラフ化したものである. グラフでは N 台に対し 1/N 程度の効果があらわれており, 見積り通り最適に並列化がおこなわれていることがわかる. しかし, PARA-COMPARE(T, F) では正関数の性質を利用せず, 単純な比較のみをおこなっているために計算量が膨大となっている. これを改善したものが PARA-COMPAREMINMAX(T, F) である.

3.2 PARA-COMPAREMINMAX(T, F)

正関数に関しては定理 1 から以下の系を示すことができる.

Corollary 1 (T, F) に対し, T の極小ベクトル集合を T^m, F の極大ベクトル集合を F^M とする. このとき, (T, F) に対して $f \in C_P$ が存在する必要十分条件は $a \leq b$ を満たす $a \in T^m$ と $b \in F^M$ の対が存在しないことである. □

PARA-COMPAREMINMAX(T, F) ではこのことをふまえ, (T, F) を N 等分後, 各プロセッサ P_i で極小値並びに極大値を求め, PARA-COMPARE(T, F) と同様に比較する.

このアルゴリズムの正当性は定理 1 の条件を全てチェックしていることから明らかである. この計算時間の見積りは次の通りである：各プロセッサにおいて, T_i^m, F_i^M の作成に $O(n(|T_i^m||T_i| + |F_i^M||F_i|)$ 時間かかるため, $\alpha = \max_i\{|T_i^m|\}, \beta = \max_i\{|F_i^M|\}$ を用いて, ステップ 2 終了までに $O(n(\alpha|T| + \beta|F|)/N)$ かかる. ステップ 4 における COMPARE$(T_i^m, F_{j'}^M)$ の計算時間は 1 回あたり $O(n\alpha\beta)$ であり, これが N 回繰り返されるから $O(N \cdot n\alpha\beta)$ である. よって, 全体の計算時間は $O(n(\alpha|T| + \beta|F|)/N + N \cdot n\alpha\beta)+$ (通信時間) となる.

Algorithm PARA-COMPAREMINMAX(T, F)

1. P_1 は T, F を $T_i, F_i (T = \bigcup T_i, F = \bigcup F_i)$ へ N 等分し, 各 $P_i (i = 1, \ldots, N)$ に振り分ける.
2. P_i は T_i の極小ベクトルの集合 T_i^m と F_i の極小ベクトルの集合 F_i^M を作成.
3. j:=i;
4. 各 P_i は必要ならば P_{i-1} からデータを受け取り, COMPARE$(T_i^m, F_{j'}^M)$ (ただし $j' \equiv i + j \pmod N$) を解き, $j := j + 1$.
5. $j = N$ となったら停止. $j < N$ なら 3 へ戻る.
6. COMPARE$(T_i^m, F_{j'}^M)$ の返り値が全て Yes なら Yes, それ以外は No を出力.

図 2 は Para-Algorithm2 に対してのプロセッサ台数に応じて, 5 回の試行実行時間の平均をグラフ化したものである. 図 2 のグラフでは 5 台以上になると, 台数の効果が現れにくくなる. これは PARA-COMPAREMINMAX(T, F) が T の極大値, 並びに F の極小値の数に依存するために生じる現象である. 図 3

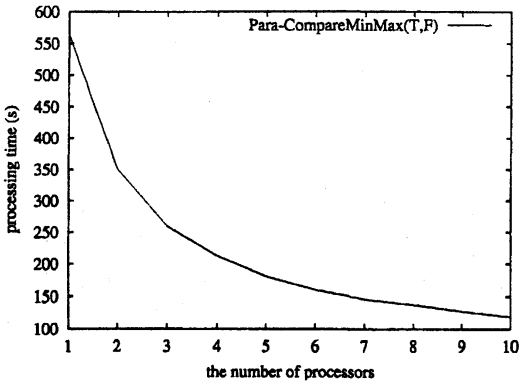

図 2: Para-Algorithm2:台数-処理時間

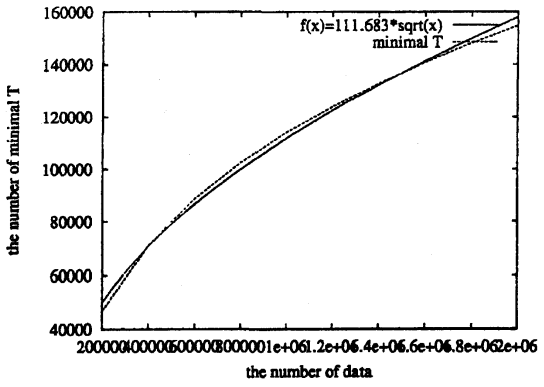

図 3: T の極小値数

はデータ数に対する T の極小値数, 並びに図 4 は F の極大値数であり, データを近似した関数はどちらも $y = c\sqrt{x}$ (ただし, c:定数であり, $0 \leq x \leq 2^{n-1}$) となる. この関数より PARA-COMPAREMINMAX(T, F) の計算時間における, おおよその見積りができるのではないかと考えている. また, 図 3, 4 よりわかるようにデータ数が小さくなると, このデータの場合データ数に占める極大値並びに極小値の割合が大きくなる. また, 台数が大きくなればなるほど $|T|, |F|$ が小さくなるために, 計算時間への極大・極小値の数の影響は相対的に強くなっている. これらから計算時間への極大・極小値の数が台数効果を反映しにくくしていることがわかる. 最後に, 図 5 に PARA-COMPARE(T, F) と PARA-COMPAREMINMAX(T, F) の台数に対する効率を示す. グラフの横軸がプロセッサ台数 k, 縦軸が (1 台の処理時間)/(k 台の並列処理時間) である.

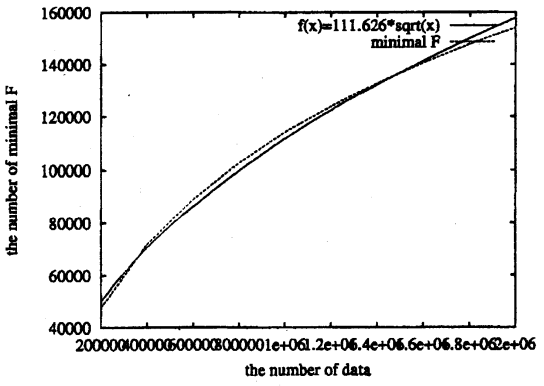

図 4: F の極大値数

4 BEST-FIT$(\mathcal{C}_\mathcal{P})$ の並列化

本節では BEST-FIT$(\mathcal{C}_\mathcal{P})$ に対する並列化アルゴリズムを提案する. 本来, BEST-FIT$(\mathcal{C}_\mathcal{P})$ は多項式時間の厳密アルゴリズムが存在するため, 並列化環境でも厳密解を求めるアルゴリズムを構築することは可能である. しかし, 本研究では高速アルゴリズム設計の観点から, 精度よりも計算量を優先し, 2 近似の並列アルゴリズムを提案する.

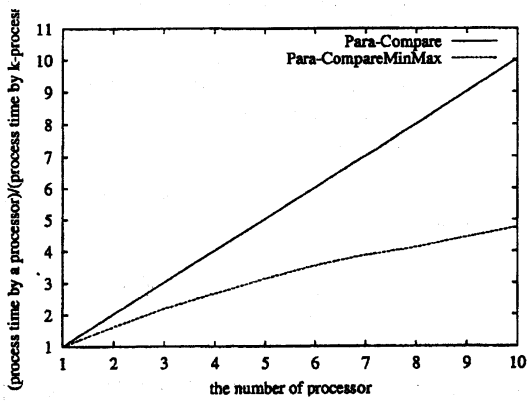

図 5: 台数効率

4.1 1台の時のアルゴリズム

準備として, この逐次厳密アルゴリズムのアイデアを簡単に紹介する. 以下に BEST-FIT($\mathcal{C}_P$) の多項式アルゴリズムを示す. (T,F) が $f \in C_P$ を持たないとすれば, $a \leq b$ となる $a \in T, b \in F$ が存在する (定理 1). そこで, 次の 2 部グラフ $G = (V_T, V_F, E)$ を定義する.

$$\begin{aligned} V_T &= \{a \mid a \leq b, a \in T, b \in F\}, \\ V_F &= \{b \mid a \leq b, a \in T, b \in F\}, \\ E(T,F) &= \{(a,b) \mid a \leq b, a \in T, b \in F\} \end{aligned}$$

この $G = (V,E)$ の最小点カバー U は 1. BEST-FIT $(\mathcal{C}_P)$ の解 (T^*,F^*) に対し $|T^* \cup F| + |F^* \cup T| \geq |U|$, 2. $T^* = (T-U) \cap (F \cup U), F^* = (T \cup U) \cap (F-U)$ と定義した時, (T^*,F^*) は $f \in C_P$ を持ち, $|T^* \cup F| + |F^* \cup T| = |U|$ が成立, の二つを満たす.

すなわち, G の最小点カバー U より (T^*,F^*) が求まる. 2 部グラフ上の最小点カバーが最大マッチングと双対関係にあることを利用する (König-Egerváry の定理) と, この問題は $O((|T|+|F|)^{5/2})$ で解くことができ [4], グラフ構成の手間を併せて $(n|T||F| + (|T|+|F|)^{5/2}))$ の時間で解を得ることができる.

4.2 2近似の並列アルゴリズム

この節では 4.1 節のアルゴリズム (最大マッチングに基づくアルゴリズム) を並列化することを考える. 3 節で紹介した PARA-COMPARE(T,F) を素直に拡張することにより, このアルゴリズムを並列化することは可能であるが, 最終的に 1 台のプロセッサで最大マッチングを解くことになる. ここでは正確さより速度を重視し, 2 近似の並列アルゴリズム PARA-BESTFIT を次に示す. ただし, 条件として $T \cap F = \emptyset$ を仮定する. 本アルゴリズムは MASTER 部と SLAVE 部に分けられており, MASTER は P_1 が SLAVE とともに兼任する. このとき, 次の定理が与えられる.

定理 2 Para-Algorithm3-2 は 2 近似アルゴリズムである.

証明. PARA-BESTFIT の Slave のステップ 5 終了時に得られるマッチング $M = \bigcup M_i$ は極大マッチング (集合の包含関係の元で極大なマッチング) である. 頂点集合 C はマッチングの両端の頂点であるから, C でカバーされないような辺は一つもない. あったとするならば, そのような辺は M に追加できるために M の極大性に反する. よって, C は頂点カバーである. また, 明らかに $|M| \leq OPT$ であり, $C = 2|M|$ であるから, $C \leq 2OPT$ となる. □

条件として $T \cap F = \emptyset$ を仮定した理由は, PARA-BESTFIT 終了時に a, $b \in C$, $a = b$ である任意の頂点 $a \in T, b \in F$ が存在した場合, PARA-BESTFIT の出力として $a = b$ である $b \in T^*, a \in F^*$ が存在することになり, 正関数の条件を満たさないためである.

PARA-BESTFIT の計算時間は, T_i^m, F_i^M の作成には $\alpha = \max_i\{|T_i^m|\}$, $\beta = \max_i\{|F_i^M|\}$ を用いて, $O(n(\alpha|T| + \beta|F|)/N)$ かかり, $V_i^{(1)}, V_i^{(2)}$ を作成する計算時間は $O(n(\alpha|F| + \beta|T|))$ かかる. また, 極大マッチングは辺を 1 つずつ選んでいきながら, 辺の両端の頂点, 並びにそれらに接続する全ての辺を除くことを, 辺がなくなるまで繰り返すことで得られるから, 最終的な M_i を求める計算時間は $O(|T||F|/N)$ である. 頂点カバーより (T^*,F^*), 並びにその拡大を求める計算時間が $O(n|T||F|/N)$ であるから, 全体では $O(n|T||F|/N)$+(通信時間) かかる.

Algorithm PARA-BESTFIT(MASTER)

1. T,F を T_i, F_i $(T = \bigcup T_i, F = \bigcup F_i)$ へ N 等分し, $P_i (i = 1,\ldots,N)$ に振り分ける.

** SLAVE の 2 まで終了後

2. $V^{(1)} = \bigcup V_i'^{(1)}$, $V^{(2)} = \bigcup V_i'^{(2)}$ を $V_i^{(1)}$, $V_i^{(2)}$ $(V^{(1)} = \bigcup V_i^{(1)}$, $V^{(2)} = \bigcup V_i^{(2)})$ へ N 等分し, 各 $P_i (i = 1,\ldots,N)$ に振り分ける.

** SLAVE の 6 まで終了後

3. $C = \bigcup C_i$ とし, C より (T^*,F^*), 並びにその拡大を求め出力.

Algorithm PARA-BESTFIT(SLAVE P_i)

** MASTER の 1 まで終了後

1. T_i の極小ベクトルの集合 T_i^m と F_i の極小ベクトルの集合 F_i^M を探索. 各 $P_j(j \neq i)$ と通信することにより,

$$V_i'^{(1)} = \{a \mid a \leq b, a \in T_i, b \in F_j^M, j = 1, \ldots, N\},$$

$$V_i'^{(2)} = \{b \mid a \leq b, a \in T_j^m, b \in F_i, j = 1, \ldots, N\}$$

を構築.

** ここではプロセッサ間で F_j^M, T_j^m を交換する.

2. $V_i'^{(1)}, V_i'^{(2)}$ を MASTER に送信.

** MASTER の 2 まで終了後

3. 辺 $e = \{(a,b) \mid a \leq b, a \in V_i^{(1)}, b \in V_i^{(2)}\}$ が存在している間, $M_i = M_i \cup \{(a,b)\}$, $V_i^{(1)} = V_i^{(1)} \backslash \{a\}$, $V_i^{(2)} = V_i^{(2)} \backslash \{b\}$ を繰り返す.

4. 以上が N 回繰り返されていれば 5 へ. そうでなければ, 自分の $V_i^{(2)}$ を P_{i+1} へ送信し, P_{i-1} から送信されてきたものを新たな V_i^2 とする (ただし, P_N は P_1 へ) . 3 へ戻る.

5. $C_i = \{u, v \mid (u,v) \in M_i\}$ を MASTER に送信.

5 まとめと今後の課題

本稿では, データ (T, F) から正拡大を発見することの並列化と解析, さらにデータの誤りがある場合についても 2 近似の並列アルゴリズムを与えた. 今後は, EXTENSION($\mathcal{C}_\mathcal{P}$) に対しては実際のデータとのずれ, すなわち, データに極端な偏りが存在する場合などについての検証, BEST-FIT($\mathcal{C}_\mathcal{P}$) に対しては, $T \cap F \neq \emptyset$ でも PARA-BESTFIT を実行可能にする改良, よりよい近似精度の並列アルゴリズムの開発, 計算実験による評価などが挙げられる.

参考文献

[1] R. Agrawal and J.C.Sharfer. Parallel mining of association rules. IEEE Transactions on Knowledge and Data Eng., 8(6)962-969, December 1996.

[2] D.W. Cheung, S.D. Lee and Y. Xiao, Effect of Data Skewness and Workload Balance in Parallel Data Mining. IEEE Transaction on Knowledge and Data Engineering, IEEE Computer Society, V14 N3, pp. 498-513, May 2002.

[3] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press, 1995.

[4] J. E. Hopcroft and R. M. Karp, An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM Journal on Computing, 2 (1973), pp. 225–231.

[5] 茨木俊秀, "データの論理的解析とブール関数," "離散構造とアルゴリズム V," 近代科学社, 147-202, 1998.

[6] Y. A. Zuev, Approximation of a partial Boolean function by a monotonic Boolean function, U.S.S.R. Computational Mathematics and Mathematical Physics, 18 (1979) 212-218.