[解説]

プロセスマイニング・サーベイ (第04回: アルゴリズム (1))

飯島 正 †, 田端 啓一 ‡, 斎藤　忍 ‡

1　はじめに

今回 (第 04 回) はプロセス発見 (Process Discovery) のためのアルゴリズムについて解説する．当初の予定では，様々なアルゴリズムの解説を一回分で簡潔する予定だったが，ページ数が嵩みバランスを欠くことからことから，今回は多様なアルゴリズムの概観的な分類を試みた後，α アルゴリズム [1][2] と呼ばれる基本的なアルゴリズムと，その直系の改良版 (α^+ アルゴリズム [3]，α^{++} アルゴリズム [4]，$\alpha^\sharp$ アルゴリズム [5]) についてのみ，詳述するものとする．その他のアルゴリズムや，各アルゴリズムの歴史的展望，横並びで比べた比較や評価については，連載を当初の予定から拡大し第 06 回以降に解説するものとする．

プロセス発見という課題に対しては，これまで多様な分野で開発されてきた手法が適用されている．順序関係の演繹的な推論，例文から文法を獲得する文法推論 (Grammatical Inference) や帰納論理プログラミング (Inductive Logic Programming;ILP) といった帰納的な推論手法最適化問題や解探索問題のための数理計画法 (整数計画法 (Integer Linear Programming;ILP)) やメタヒューリスティック手法 (遺伝的アルゴリズム (Genetic Algorithm;GA) や遺伝的プログラミング (Genetic Programming;GP) などの進化的手法 (Evolutionary Approach) や焼きなまし法 (Simulated Annealing;SA))，計算知能 (Computational Intelligence) におけるファジー論理 (Fuzzy Logic) やニューラルネットワーク (主にリカレントニューラルネットワーク (Reccurent Neural Network;RNN)) なども適用されてきた．こうした手法は，それぞれプロセス発見への適用に値する特徴を有している．

そこで，今回は，プロセス発見アルゴリズム全体を見渡す分類軸としては，主に採用した手法の分類に基づき，単純に，(1) 演繹的推論手法，(2) 帰納的推論手法，(3) ソフトコンピューティング手法，の三つに大きく分けることとする[1]．

今回，主に取り上げる手法である，α アルゴリズム・ファミリーは，今回の分類では，(1) 演繹的推論手法に属し，プロセスマイニング研究を強力に推進している van der Aalst のグループによるものである[2]．

2　プロセス発見アルゴリズムの分類

プロセス発見アルゴリズムは，基本的には，実行ログ (イベント発火系列ないしトレース多重集合) を入力とし，そのログを生成しうるプロセスのモデルを出力する関数と考えることができる．入力の要素である実行トレースは，典型的には，実行されたタスク (アクティビティ) の直線的な順序列 (イベント発火系列) である (タイムスタンプなど付帯情報を伴うこともある)．出力は，いわゆるビジネスプロセスのモデリング記法で表現されたモデルであるが，典型的には，ペトリネットのサブクラスである WF-net を想定している．但し，木構造でも表現可能な，構造化された (structured) モデルに限定することもある．

各アルゴリズムは，それぞれ提案に値する特徴を有しているが，それが分類軸として採用すべものかどうかについて判断が必要となる[3]．たとえば，ノイズに強いという特徴をもったアルゴリズム (Heuristaic Miner，Flexible Heuristic Miner など)，多くのアルゴリズムがフラットなモデルを生成する中で，ファジークラスタリングによって，

A Survey on Business Process Mining
— 04: Algorithm (1) —
Tadashi Iijima†, Keiichi Tabata‡, and Shinobu Saito‡
†Faculty of Science and Technology, Keio University
‡Nippon Telegraph and Telephone Corporation
† 慶應義塾大学・理工学部
‡ 日本電信電話株式会社
[解説] 2017 年 9 月 30 日受付.

[1] 複数の手法を組み合わせたハイブリッド手法も考えられるが，その場合は個々の手法ごとに考える．

[2] 解説中で使用しているモデルやログの例の多くは，文献 [6] から借用している．

[3] 以降，今回詳述しないアルゴリズムの名称も現れるが，煩雑さを避けるために，文献情報は，次回以降のそのアルゴリズムに関して詳述する際に記載する

階層的なモデルを生成できるアルゴリズム (Fuzzy Miner) もある．まず低水準なモデル (遷移系やマルコフモデル) を獲得し，高水準なプロセスモデルへ変換する二段階 (two-phase) アプローチといった観点もある．遷移系 (Transition System) や有限状態オートマトン (Finite State Automaton)，もしくは言語の文法からペトリネットモデルを合成する問題としてとらえる手法 (Region-ベースアルゴリズム) もある．ログを例文として捉えモデルを例文を認識もしくは生成できる文法とみなすと，文例から文法を獲得する文法推論 (Grammatical Inference) は，プロセス発見に相当することとなる．そうした文法推論や，タスクの順序関係に関する制約条件集合から機械学習する手法は，部分的に与えられた不完全情報から帰納的な推論を行うことで漸進的にモデルを構築していくプロセス発見手法といえる．このとき，実働しているシステムやレガシーシステムの出力した実行ログであれば，入力として負例 (negative example) が存在することは考えにくいが，プロセス発見の入力は，順序関係の制約条件なのだと考えれば，正例 (positive example) だけでなく負例を受け付けることに支障は生じない．また，解探索問題や最適化問題としてとらえると，遺伝的アルゴリズム (GA)/遺伝的プログラミング (GP) のような進化的手法 (Evolutionary Approach) の導入 (たとえば，Genetic Miner や Evolutionary Tree Miner) も考えられるが，その手法の特性からノイズ耐性が高いという特徴を有する．

基本的には，プロセス発見アルゴリズムの特徴は，それがもとにしている手法に大きく依存するので，今回，プロセス発見のためのアルゴリズムを分類するにあたっては，アルゴリズム自身が有するこれらの特徴には言及するとはいえ，概観する分類としては，基礎とする手法に基づいて，大きく，

(1) 演繹的推論手法，
(2) 帰納的推論手法，
(3) ソフトコンピューティング手法，

の 3 つにわけることとする．例えばノイズ耐性に強いという性質は，基礎とする手法が全く異なる，ヒューリスティック・マイナーでも，進化的手法 (Evolutionary Approach) でも持ちうるが，それらの特性は，個々の分類の中で論じるものとする．

3 演繹的推論手法: α アルゴリズム

今回は，(1) 演繹的推論手法というカテゴリに分類したアルゴリズムとして，α アルゴリズム [1, 2] とその欠点を克服する直系の拡張アルゴリズム群を紹介する．まず，van der Aalst らのグループによる α アルゴリズムとその問題点について述べる．

元々，ペトリネットをベースとしたビジネスプロセスモデルと，そのパターンに関する研究を進めてきた van der Aalst らのグループは，この α アルゴリズムの開発とその改良から始まり，多くのプロセスマイニングのアルゴリズムの研究を牽引してきた．そして，その多くは（既に標準的には使われなくなってきたものもあるが），既に紹介している ProM プラットフォームの上で，プラグインとして利用し試すことができるように開発されている．もちろん，α アルゴリズム・ファミリー (α^{++} や $\alpha^{\sharp}$) も試すことができる．

α アルゴリズムの基本的は発想は，ログからタスク間の順序関係のパターンを見つけ出し，特定のパターンに対して対応するモデル構成要素を出力して，つなぎ合わせるものである．アルゴリズム自体は，8 ステップ程で記述できる極めて単純なものである．

α アルゴリズムは並列構造に対応しているが，ログ内容全体から順序関係のパターンを抽出し一覧表 (足跡行列) を生成するため，本質的にノイズや不完全なログに耐性がないという欠点を持つ．それだけではなく幾つかの部分構造のパターンに対応していないという問題点があるため，後年，その点を改良した直系の改訂版アルゴリズムが提案されている．今回は，それら α アルゴリズムファミリーを特に取り上げて解説している．

タスクの集合 A から構成されるイベントログ L を入力とする．このイベントログに対し，表**??**の 4 つの $x, y \in A$ の順序関係 (Log-based ordering relation) を定義する．直接先行関係 $x \succ_L y$ を，よりフォーマルに定義すると，$t_i = x, t_{i+1} = y$ (但し，$i \in \{1, \ldots, n-1\}$) を含むようなトレース $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in L$ が存在することに相当する．

直接先行関係　$x >_L y \iff x$がyに直接先行するトレースが存在する

単方向順序関係　$x \rightarrow_L y \iff (x >_L y) \wedge (y \not>_L x)$

直接関係未定義　$x \#_L y \iff (x \not>_L y) \wedge (y \not>_L x)$

並行関係　$x ||_L y \iff (x >_L y) \wedge (y >_L x)$

α アルゴリズムの大まかな手順は，以下の通りである．

(1) 基本順序関係 $\succ_L$,$\rightarrow_L$,$\#_L$,$||_L$ を抽出する
  (1-1) まず直接先行関係 $\succ_L$ を抽出する．
  (1-2) そこから，単方向順序関係 $\rightarrow_L$，直接関係未定義 $\#_L$，並行関係 $||_W$ を導出する
(2) 足跡行列 (footprint matrix) と呼ばれるタスク間の関係表を構成する (表 1).
(3) 足跡行列からパターン (図 1 に典型的なものを示す) に基づいて，WF-net の断片を作り，結合する．

例を示す．ログ $L_1 = \{\langle a,b,c,d,e,f,b,d,c,e,g\rangle, \langle a,b,d,c,e,g\rangle^2, \langle a,b,c,d,e,d,f,b,c,d,e,f,b,c,e,g\rangle\}$ が与えられたとき，α アルゴリズムは，そのログ L_1 から，まず足跡行列 (表 1) を生成し，モデル (図 2 を構築する．

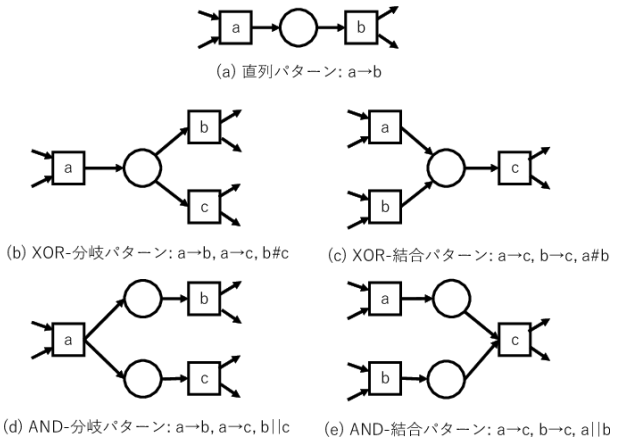

図 1: 足跡行列からモデルの断片を構築するための典型的なパターン

このアルゴリズムをフォーマルに記述するとアルゴリズム 1 のように示すことができる．得られるモデルは，プレースの集合 P，トランジションの集合 T，アークの集合 F の組で表される WF-net とする．

表 1: ログ L_1 の足跡行列

	a	b	c	d	e	f	g
a	#	→	#	#	#	#	#
b	←	#	→	→	#	←	#
c	#	←	#	||	→	#	#
d	#	←	||	#	→	#	#
e	#	#	←	←	#	→	→
f	#	→	#	#	←	#	#
g	#	#	#	#	←	#	#

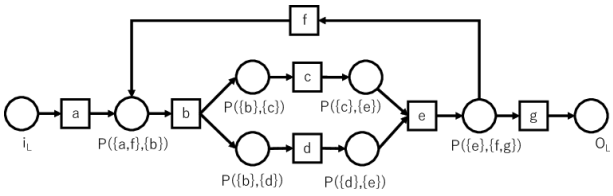

図 2: ログ L_1 から α アルゴリズムで生成された WF-net

Algorithm 1 α アルゴリズム $\alpha(L)$

(1) タスク集合
$T_L = \{t \in T \mid \exists_{\sigma \in L} t \in \sigma\}$
(2) 開始タスク集合
$T_I = \{t \in T \mid \exists_{\sigma \in L} t \in first(\sigma)\}$
(3) 終了タスク集合
$T_O = \{t \in T \mid \exists_{\sigma \in L} t \in last(\sigma)\}$
(4) $X_L = \{(A,B) \mid A \subseteq T_L$
$\wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset$
$\wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_L b$
$\wedge \forall_{a_1,a_2 \in A} a_1 \#_L a_2$
$\wedge\ forall_{b_1,b_2 \in B} b_1 \#_L b_2\}$
(5) $Y_L = \{(A,B) \in X_L |$
$\forall_{(A',B') \in X_L} A \subseteq A'$
$\wedge B \subseteq B' \Rightarrow\}$
$(A,B) = (A',B')\}$
(6) $P_L = \{p\,(A,B) \mid (A,B) \in Y_L\} \cup \{i_L, o_L\}$
(7) $F_L = \{(a, p\,(A,B)) \mid (A,B) \in Y_L \wedge a \in A\}$
$\cup \{(p\,(A,B), b) \mid (A,B) \in Y_L \wedge b \in B\}$
$\cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$
(8) 出力 $\alpha\,(L) = (P_L, T_L, F_L)$

4 「短いループ」への対応: α^+ アルゴリズム

α アルゴリズムでは，長さ 1 ならびに 2 の「短い (short) ループ」を生成できない (長さ 3 以上は問題ない) ことが知られていて，その問題に対処する拡張も提案されている [3]．それが α^+ アルゴリズムである．

ログ $L_2 = \{\langle a,c\rangle^2, \langle a,b,c\rangle^3, \langle a,b,b,c\rangle^2, \langle a,b,b,b,b,c\rangle\}$ が与えられたとすると，このモデルには長さ 1 のループが含まれていると判断することが妥当となる．すなわち，図 3 のような WF-net が生成されることが期待できるが，α アルゴリズムで生成される WF-net は図 4 である．

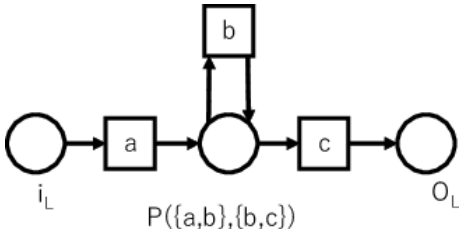

図 3: L_2 に対する妥当な WF-net の一つ

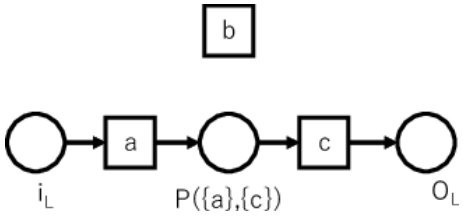

図 4: L_2 に対して α アルゴリズムが生成する WF-net

また，ログ $L_3 = \{\langle a,b,d\rangle^3, \langle a,b,c,b,d\rangle^2, \langle a,b,c,b,c,b,d\rangle\}$ が与えられたとすると，このモデルには長さ 2 のループが含まれていると判断することが妥当となる．すなわち，図 5 のような WF-net が生成されることが期待できるが，α アルゴリズムで生成される WF-net は図 6 である．

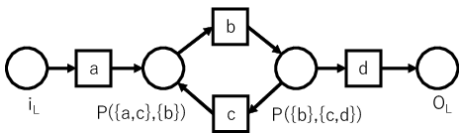

図 5: L_3 に対する妥当な WF-net の一つ

α-アルゴリズムの拡張である α^+-アルゴリズムでは，長さ 1 ならびに 2 の「短い」ループについてのみ特別に前処理ならびに後処理で対応する．

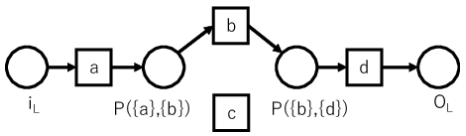

図 6: L_3 に対して α アルゴリズムが生成する WF-net

長さ 1 のループは，トレース中に同じイベントが連続すればすぐ分かるので，前処理として，一旦，それを一つのイベントへ縮退させておき，最終的に，そのイベントを意味するノードに自己ループを付与すればよい．長さ 2 のループに関する問題は，並行関係 $x||_W y$ と，双方向にに付与された単方向順序関係 $(x \rightarrow_W y) \wedge (y \rightarrow_W x)$ とが区別できないことに起因している．そこで，対象とするトレースを W としたとき，長さ 2 のループを識別するための関係 $\diamondsuit_W$ を導入し，イベント間の直接的な基本順序関係を再定義する．

まず，長さ 2 のループの存在を以下の関係で表現する．

$$x \triangle_W y \iff \text{ログ} W \text{中に} xyx \text{という部分系列を持つトレースが存在する}$$
$$x \diamondsuit_W y \iff (x \triangle_W y) \wedge (y \triangle_W x)$$

この関係を用いて，長さ 2 のループを，並行関係 $x||_W y$ から除外する．

$$x \rightarrow_W y \iff (x >_W y) \wedge (y \not>_W x \vee x \triangle_W y)$$
$$x \#_W y \iff (x \not>_W y) \wedge (y \not>_W x)$$
$$x||_W y \iff (x >_W y) \wedge (y >_W x) \wedge x \not\diamondsuit_W y$$

これにより，長さ 2 のループが含まれた WF-net を得ることができる．

α^+ アルゴリズムをアルゴリズム 2 に示す．ここで，$L1L$ は「長さ 1 のループ (length-one-loop)」を持つタスクを意味し，W^{-L1L} は「長さ 1 のループ」を取り除いたログを意味する．関数 $eliminateTask$ は，与えられたトレースから特定のタスクを取り除いたトレースを返す関数である．

5 非自由選択な構造への対応: α^{++} アルゴリズム

α アルゴリズムでも対応できていないモデル構造が幾つか知られている．そのうちの一つである「長さ 1 ないし 2 の『短いループ (short loop)』構造を検出できない」，という問題点は，α^+ アルゴリズムで対処された．しかし，α アルゴリズム

Algorithm 2 α^+ アルゴリズム

$T_{log} = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$
$L1L = \{t \in T_{log} \mid$
$\quad \exists_{\sigma=t_1 t_2 \ldots t_n \in W; i \in \{1,2,\ldots n\}} t = t_{i-1} \wedge t = t_i\}$
$T' = T_{log} \backslash L1L$
$F_{L1L} = \emptyset$
for each $t \in L1L$ **do**
$\quad A = \{a \in T' \mid a >_W t\}$
$\quad B = \{b \in T' \mid t >_W b\}$
$\quad F_{L1L} := F_{L1L} \cup$
$\quad\quad \left\{\left(t, p_{(A\backslash B, B\backslash A)}\right), \left(p_{(A\backslash B, B\backslash A)}, t\right)\right\}$
end for
$W^{-L1L} = \emptyset$
for each $\sigma \in W$ **do**
$\quad \sigma' = \sigma$
$\quad$ **for each** $t \in L1L$ **do**
$\quad\quad \sigma' := \mathrm{eliminateTask}(\sigma', t)$
$\quad$ **end for**
$\quad W^{-L1L} := W^{-L1L} \cup \sigma'$
end for
$(P_{W^{-L1L}}, T_{W^{-L1L}}, F_{W^{-L1L}}) = \alpha\left(W^{-L1L}\right)$
$P_W = P_{W^{-L1L}}$
$T_W = T_{W^{-L1L}} \cup L1L$
$F_W = F_{W^{-L1L}} \cup F_{L1L}$
$\alpha^+(W) = (P_W, T_W, F_W)$

に内在し，α^+ アルゴリズムでも対処されないまま残されていた問題点に,「非自由選択な構成要素 (non-free choice construct) に起因する『暗黙的な依存関係 (implicit dependency)』を検出できない」というものがある．この問題点にアプローチしているのが α^{++} アルゴリズム [4] である．

図 8 のような WF-net は，非局所依存性を持っており，a が発火したら d が，b が発火したら e が発火する．ログ $L_4 = \{\langle a,c,d\rangle^{45}, \langle b,c,e\rangle^{42}\}$ が与えられたとすると，α^+ アルゴリズムで生成される WF-net は図**??**となってしまう．これは，ログ $L_5 = \{\langle a,c,d\rangle^{45}, \langle b,c,d\rangle^{42}, \langle a,c,e\rangle^{38}, \langle b,c,e\rangle^{22}\}$ を入力して，α^+ アルゴリズムで生成される WF-net でもある．しかし，α^{++} アルゴリズムであれば，ログ L_4 から，図 8 の WF-net を生成することができる．

この問題点は，タスク間の非局所的な依存関係を十分に扱えていないことを意味する．α アルゴリズムは，基本的に前後二つの連続するタスクの関係 (直接先行関係) を元に，全タスク間の基本順序関係を足跡行列にまとめている．そ改良版である α^+ アルゴリズムは，(1) α アルゴリズムに長さ 2 のループを検出するための基本順序関係を導入することと，(2) 前処理として長さ 1 のループを検出し縮退させておき，後処理として自己ループをモデルに追加する，という対応を行っている．すなわち，まず局所的な関係に着目し，そこから出発して全タスク間の関係を導出している点に変わりはない．その際に,「タスク間の非局所的な依存関係を十分に検出するに至っていない」ということになる．

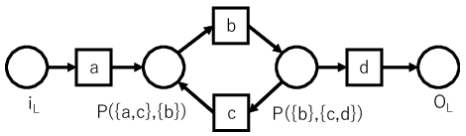

図 7: L_4 に対する妥当な WF-net の一つ (局所依存性を持つ WF-net)

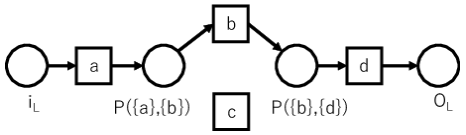

図 8: L_4 に対して α アルゴリズムが生成する WF-net

α^{++} アルゴリズムでのアプローチの概略を簡潔に紹介する．詳細は，煩雑となるので文献を参照されたい．ここで，長さ 1 のループへの対応は，α^+ アルゴリズムの手法に任せるので，以下，長さ 1 のループは縮退させたものとして考える．

まず,「暗黙的な依存関係」を持つモデルの中で，α アルゴリズムでは，その「暗黙的な依存関係」を検出できないパターンを分析することで,「暗黙的な依存関係」を導入するプレースを 2 つ生成してしまうパターン 2 種類，幾つかのアークを欠損してしまうパターン 4 種類,「暗黙的な依存関係」を導入するプレースを検出できないパターン 1 種類，を見出した．これらの各パターンに対応する，三つの「暗黙的な依存関係」$a \mapsto_{W^1} b$, $a \mapsto_{W^2} b$, $a \mapsto_{W^3} b$ を定義している．それぞれ，以下のような依存関係である．$a \mapsto_{W^1} b$ は，マイニングされるモデルに，二つの前後するタスクに接続するプレースがあり，後続タスクが 1 つ以上の入力プレースを持つとき，常にそのタスクは，先行タスクの直後に実行される可能性を持つことを保証するものである．$a \mapsto_{W^2} b$ は，タスクが複数の並行分岐のうちの一つからトークンを得るなら，同時に他の並行分岐もトークンを消費しなければな

らないことを保証するものである．$a \mapsto_{W^3} b$ は，二つの排他的分岐が異なる並行分岐の集合につながっており，その二つの集合がともにある条件を満たすならば，マイニングされた WF-net が健全であることを保証するものである．α^{++} アルゴリズムの基本発想は，これらの三つの「暗黙的な依存関係」を過不足なく検出し，他の基本順序関係と同様，モデル生成に活用することにある．

そこで，次に，基本順序関係を以下の 9 種類に拡大し，それによって三つの「暗黙的な依存関係」を定義し検出できるようにする (定義は省く)．

$a >_L b \iff a$がbに直接先行するトレースが存在する

$a \triangle_W b \iff$ ログW中に$\langle aba \rangle$という部分系列を持つトレースが存在する

$a \rightarrow_W b \iff (a >_W b) \wedge (b \not> _W a \vee a \triangle_W b \vee b \triangle_W a)$

$a \#_W b \iff (a \not> _W b) \wedge (b \not> _W a)$

$a || _W b \iff (a >_W b) \wedge (b >_W a) \wedge \neg(a \triangle_W b \vee b \triangle_W a)$

$a \triangleleft_W b \iff (a \#_W b)$かつ $(c \in W \wedge c \rightarrow_W a \wedge c \rightarrow_W b)$ なるタスクtが存在する

$a \triangleright_W b \iff (a \#_W b)$かつ $(c \in W \wedge a \rightarrow_W c \wedge a \rightarrow_W c)$ なるタスクtが存在する

$a \gg_W b \iff (a \not> _W b)$かつ $((\sigma \in W) \wedge (i < j) \wedge t_k = a \wedge t_k = b \wedge$ (全ての$k \in \{i+1, \ldots, j-1\}$が $(t_k \neq a \wedge t_k \neq b \wedge \neg(t_k \triangleleft_W a \vee t_k \triangleright_W a)$を満たす トレース$\sigma \in W$と$i, j \in \{1, \ldots, n\}$ が存在する

$a \succ_W b \iff a \rightarrow_W b \vee a \gg_W b$

最初の 5 個は α^{+} アルゴリズムのものと同じである．順序関係 $a \triangleleft_W b$ と $a \triangleright_W b$ は，それぞれ，XOR-分岐 (XOR-Split) と XOR-結合 (XOR-Join) に対応する．順序関係 $a \gg_W b$ は，b が a に間接的に後続することを意味し，$a \succ_W b$ は，b が a に直接的もしくは間接的に後続することを意味する．これらの順序関係を追加することで，上記三つの「暗黙的な依存関係」を検出することができる．各「暗黙的な依存関係」の間の相互関係を考慮すると，その検出順序が決まる．詳細は省くが，W_1, W_2, W_3 の順に検出し，W_3 の検出の前に，全ての W_2 を W として扱うことが必要である．したがって，アルゴリズムでは，この順序に従って，各「暗黙的な依存関係」を検出する．

但し，ここで，検出された「暗黙的な依存関係」の中には，他の「暗黙的な依存関係」から導くことできる，冗長な「暗黙的な依存関係」(redundant implicit dependency) が含まれている．そうした冗長な「暗黙的な依存関係」を削除するルールも二つ定義されている (この詳細も省くが，後述するアルゴリズムでは，$eliminateRdByRule1$, $eliminateRdByRule2$ としている)．

以上により，検出された三つの「暗黙的な依存関係」を満たすモデルを生成するアルゴリズムが，α^{++} アルゴリズムである．アルゴリズム 3 に示す．

Algorithm 3 α^{++} アルゴリズム

$T_{log} = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$
$L1L = \{t \in T_{log} \mid \exists_{\sigma = t_1 t_2 \ldots t_n \in W; i \in \{1,2,\ldots n\}} t = t_{i-1} \wedge t = t_i\}$
$T' = T_{log} - L1L$
$F_{L1L} = \emptyset$
$X_W = \{(A, B, C) \mid A \subseteq T' \wedge B \subseteq T'$
$\wedge C \subseteq L1L$
$\wedge \forall_{a \in A} \forall_{c \in C} (a >_W c \wedge \neg(c \triangle_W a))$
$\wedge \forall_{b \in B} \forall_{c \in C} (c >_W b \wedge \neg(c \triangle_W b))$
$\wedge \forall_{a \in A} \forall_{b \in B} (a \not|_W b)$
$\wedge \forall_{a_1, a_2 \in A} (a_1 \#_W a_2)$
$\wedge \forall_{b_1, b_2 \in B} (b_1 \#_W b_2)$
$W^{-L1L} = \emptyset$
for each $\sigma \in W$ **do**
  $\sigma' = \sigma$
  for each $t \in L1L$ **do**
    $\sigma' :=$ eliminateTask(σ', t)
  end for
  $W^{-L1L} := W^{-L1L} \cup \sigma'$
end for
$ID_{W^1} = \{(a, b) \mid a \in T' \wedge b \in T' \wedge a \mapsto_{W^1} b\}$
$(P_{W^{-L1L}}, T_{W^{-L1L}}, F_{W^{-L1L}}) = \alpha(W^{-L1L})$
各 $a \mapsto_{W^1} b \in ID_{W^1}$ を $a \rightarrow_W b$ として扱い，
$ID_{W^2} = \{(a, b) \mid a \in T' \wedge b \in T' \wedge a \mapsto_{W^2} b\}$
$ID_{W^2} = eliminateRdByRule1(ID_{W^2})$
$X_W = \{(A \cup A_2), (B \cup B_2) \mid P(A, B) \in P_{W^{-L1L}}$
$\wedge A_2 \cup B_2 \neq \emptyset$
$\wedge A \cup A_2 \neq \emptyset$
$\wedge B \cup B_2 \neq \emptyset$
$\wedge \forall_{a \in A} \forall_{b \in B_2} (a \mapsto_{W^1} b \vee a \mapsto_{W^2} b)$
$\wedge \forall_{a \in A_2} \forall_{b \in C \cup B_2} (a \mapsto_{W^1} b \vee a \mapsto_{W^2} b)$
$\wedge \forall_{a_1 \in A} \forall_{a_2 \in A_2} (a_2 \#_W a_1 \wedge a_2 \not\gg_W A_1)$
$\wedge \forall_{b_1 \in B} \forall_{b_2 \in B_2} (b_1 \#_W b_2 \wedge b_1 \not\gg_W b_2)$

$$
\begin{aligned}
&Y_W = \{(A,B) \mid ((A,B) \in X_W \\
&\qquad \vee p(A,B) \in P_{W^{L1L}}) \\
&\quad \wedge \forall_{(A',B') \in X_W \vee p(A',B') \in P_{W-L1L}} \\
&\quad (A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B'))\} \\
&\text{各 } a \mapsto_{W^2} b \in ID_{W^2} \text{ を } a \rightarrow_W b \text{ として扱い,} \\
&ID_{W^3} = \{(a,b) \mid a \in T' \wedge b \in T' \wedge a \mapsto_{W^3} b\} \\
&ID_{W^3} = eliminateRdByRule2(ID_{W^3}) \\
&X_W = \{(A,B) \mid A \subseteq T' \wedge B \subseteq T' \\
&\quad \wedge \forall_{a \in A} \forall_{b \in B} a \mapsto_{W^3} b \\
&\quad \wedge \forall_{a_1,a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1,b_2 \in B} b_1 \#_W b_2\} \\
&Z_W = \{(A,B) \in W_W \mid \\
&\quad \forall_{A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \\
&\qquad \Rightarrow (A,B) = (A',B')\} \\
&P_W = \{p(A,B) \mid (A,B) \in Y_W \cup Z_W\} \\
&\quad - \{p(A,B) \mid \\
&\qquad \exists_{(A',B',C') \in L_W} A' = A \wedge B' = B\} \\
&\quad \cup \{p(A \cup C, B \cup C) \mid (A,B,C) \in L_W\} \\
&T_W = T_{W-L1L} \cup L1L \\
&F_W = \{(a, p(A,B)) \mid (A,B) \in P_W \wedge a \in A\} \\
&\quad \cup \{(p(A,B), b) \mid (A,B) \in P_W \wedge b \in B\} \\
&\alpha^{++}(W) = (P_W, T_W, F_W)
\end{aligned}
$$

6 不可視タスクへの対応: $\alpha^\sharp$ アルゴリズム

これまでの α アルゴリズムへの拡張と同じく，$\alpha^\sharp$ アルゴリズムでも，α アルゴリズムが対応できなかった構造を持つモデルを生成することができる．しかし，$\alpha^\sharp$ アルゴリズムで対応しているのは，元々のモデルには含まれているが，ログの中には含まれない無名のタスクを追加することである．これらのタスクは，不可視タスク (Invisible Task; IT) と呼ばれる．$\alpha^\sharp$ アルゴリズムに関しては，煩雑になるので手順の詳細に立ち入らず，どのような不可視タスクが対応できるようになったかを示すにとどめる．

まず，不可視タスクのうち，主要不可視タスク (Prime Invisible Task) を定義する．不可視タスクの中にはモデルの挙動に影響を与えないものもあるが，主要不可視タスクはモデルの挙動に影響を与えるものである．不可視タスクを含む WF-net が以下の 3 要件を満たすとき，健全である (sound) といい，そこに含まれている不可視タスクは主要不可視タスクである．

(1) 包囲性 (Surround) $\cdots$ 各不可視タスクが，少なくとも一つの直接先行する可視タスクと，一つの直接後続する可視タスクを持つ
(2) 連続性 (Successtion) $\cdots$ 二つの可視タスク a と b が，不可視基本パス (Invisible Elementary Path; あるタスクから，不可視タスクだけを，同じタスクを 2 度通らないように辿って別のタスクへ到達する経路）で接続されているなら，a の直後に b が出現するトレースが存在する
(3) 必要性 (Necessity) $\cdots$ 可視タスクが，可視タスクとの間に新たに依存関係を導入することなしに，入力と出力のプレースをマージすることで直接削除できない

主要でない不可視タスクの中には，前処理として入力イベントと出力イベントを導入することで，主要不可視タスクとできるものもある．

$\alpha^\sharp$ アルゴリズムは，主要不可視タスクを分析し，SKIP, REDO, SWITCH という 3 つのタイプの主要不可視タスクに対し，それらを作り出す ConIT アルゴリズム (Construction alghrithm) を定義したものである．ここでは，ConIT アルゴリズムの手順の詳細の説明は省く．

SKIP は，あるタスクを選択的にスキップするために導入される不可視タスクのタイプであり，LONG SKIP は複数のタスクをスキップする．REDO は，特定のタスクを反復的に実行するために導入される不可視タスクのタイプであり，LONG REDO は複数のタスクを反復的に実行する．SWITH は，選択的な実行のために導入される不可視タスクのタイプである (図 9 に示す)．

図 9 では，N_1〜N_5 がそれぞれ SKIP, LONG SKIP, REDO, LONG REDO, SWITCH として機能している不可視タスク (黒塗りされていて，トレースに含まれていない) である．これに対し，L_1〜L_5 のログを与えて α アルゴリズムを実行すると，残念ながら N_1〜N_5 は生成されず，代わりに N_1'〜N_5' が生成されてしまう．これは，ログに含まれていないタスクが α アルゴリズムでは導入されないことに由来するので，$\alpha^\sharp$ アルゴリズムでは，必要に応じてそれらの不可視タスクを構成するような ConIT アルゴリズムを導入することで対応している．

ここで取り上げている主要不可視タスク SKIP, REDO, SWITCH は，実際に非常によく使われるパターンであり，$\alpha^\sharp$ アルゴリズムとして，これらのパターンを扱うことができるように α アルゴリズムが拡張されたことは，非常に意義のあることといえる．

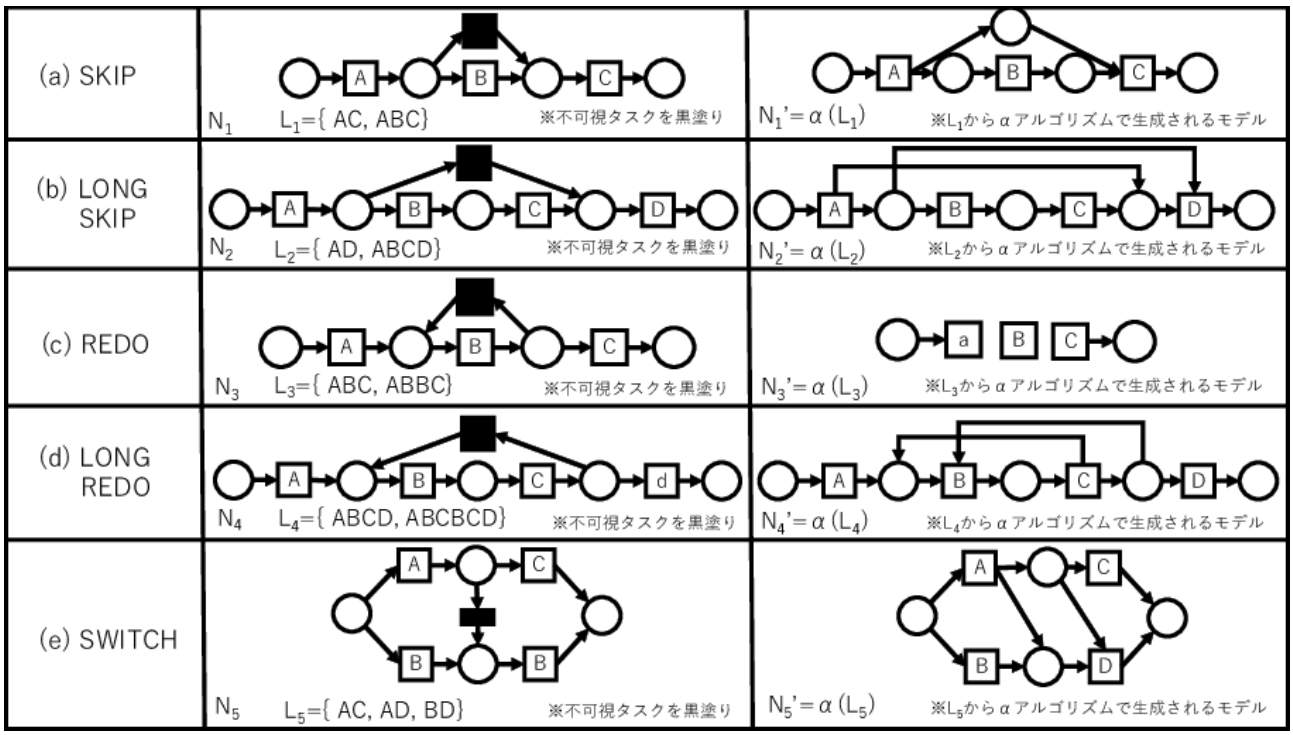

図 9: 主要不可視タスクを含む WF-net

以上，並列構造を扱うことのできる α アルゴリズムを出発点として，扱うことのできなかった構造を分析し，タスク間の関係を拡張したり，前処理や後処理として追加的な処理を補うことで，地道に対応できる範囲が拡大されてきた経緯を解説した．具体的には，α^{+} アルゴリズムは「短いループ (長さ 1 ないし 2)」への対応，α^{++} アルゴリズムは「非自由選択構造に伴うような長距離依存関係」への対応，$\alpha^{\sharp}$ アルゴリズムは「不可視タスク」への対応であった．

7 おわりに

本稿では，プロセス発見アルゴリズムを分類し，簡単な解説を加えることを試みた．まず，プロセス発見アルゴリズムを，大きく (1) 演繹的推論手法，(2) 帰納的推論手法, (3) ソフトコンピューティング手法の三つに分類した．更に，タスク間の順序関係から演繹的にモデルを構成する α アルゴリズムとその直系の改良版アルゴリズムを取り上げ，解説した．α アルゴリズムで対応できなかった対象を，一つ一つ分析し，それらに対処していく地道なアプローチで改良されていった流れをつかむことができる．しかし，プロセス発見のためのアルゴリズムは，演繹的手法に基づく α アルゴリズム・ファミリーだけではない．帰納的推論やソフトコンピューティング/計算知能の膨大な研究で培われた多様な手法が，プロセスマイニングに適用されている．これらの手法は，その基礎となった手法の特性を引継ぎ，ノイズや不完全情報に強いという特徴を有している．

元々は，この回で，多様なプロセス発見アルゴリズムを一通り概観する予定であったが，紙数の都合上，今回は見送ることとした．次回，第 05 回は応用事例の解説を予定しているので，α アルゴリズム以外のアルゴリズムについては，連載回数を当初予定の全 5 回から拡大し，第 06 回以降で解説することとする．その際には，まずは，帰納的推論やソフトコンピューティング/計算知能を基礎とせずに，ノイズや不完全情報を扱えるようにアプローチしたヒューリスティック・マイナーから解説を始める予定である．

今回の執筆分担:
本稿の執筆分担は飯島が担当し，田端・斎藤が内容を確認した．

謝辞

アルゴリズム記述の一部の入力の際に，沖田勇馬氏 (慶應義塾大学・大学院・理工学研究科・修士課程 1 年在学中) のご協力をいただいている．

文献

[1] W. van der Aalst, A. Weijters, and L. Maruster, “Workflow Mining: Which Processes can be Rediscovered?,” BETA Working Paper Series, WP 74, 2002, http://wwwis.win.tue.nl/~wvdaalst/publications/p169.pdf.

[2] W. van der Aalst, A. Weijters, and L. Maruster, “Workflow Mining: Discovery Process Models from Event Logs ,” IEEE Transactions on Knowledge and Data Engineering, Vol.16, No.9, pp.1128–1142, 2004, http://wwwis.win.tue.nl/~wvdaalst/publications/p245.pdf.

[3] A. de Medeiros, B. van Dongen, W. van der Aalst, and A. Weijters, “Process Mining for Ubiquitous Mobile Systems: An Overview and a Concrete Algorithm,” Ubiquitous Mobile Information and Collaboration Systems (UMICS 2004), (eds.) L. Baresi, S. Dustdar, H. Gall, and M. Matera, pp.156–170, 2004, http://wwwis.win.tue.nl/~wvdaalst/publications/p245.pdf.

[4] L. Wen, W. van der Aalst, J. Wang, and J. Sun, “Mining process models with non-free-choice constructs,” Data Mining and Knowledge Discovery, Vol.15, No.2, pp.145–180, 2007, http://wwwis.win.tue.nl/~wvdaalst/publications/p394.pdf.

[5] L. Wen, J. Wang, W. van der Aalst, B. Huang, and J. Sun, “Mining Process Models with Prime Invisible Tasks,” Data and Knowledge Engineering, Vol.69, No.10, pp.999–1021, 2010, http://wwwis.win.tue.nl/~wvdaalst/publications/p575.pdf.

[6] W. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer, 2011,2016, http://www.springer.com/jp/book/9783642193446, http://www.springer.com/jp/book/9783662498507, http://www.processmining.org/book/start.

著者略歴

[1] 飯島 正 (いいじま ただし)

慶應義塾大学 理工学部 専任講師 (管理工学科 所属)．慶應義塾大学 理工学部 計測工学科卒業 (1986 年)，同大学院 理工学研究科 修士課程 修了 (1988 年)， 同博士課程 単位取得退学 (1991 年)．1990 年より (株) 東芝勤務を経て，1992 年より 現所属 (助手を経て現職)．博士 (工学)．情報システム学会 元理事 (2007–2008 年 大会担当理事，2009–2013 年 理事)．

[2] 田端 啓一 (たばた けいいち)

2012 年早稲田大学 大学院 修士課程修了，同年， 日本電信電話 (株) に入社． 以来，ソフトウェアイノベーションセンタにて ソフトウェア工学の研究に従事．専門分野: プログラム自動並列化，コンピュータアーキテクチャ，ソフトウェアテスト．

[3] 斎藤 忍 (さいとう しのぶ)

2001 年 慶應義塾大学 大学院 修士課程修了，同年 NTT データに入社．2015 年 日本電信電話株式会社に転籍． 現在，ソフトウェアイノベーションセンタに所属．2016 年よりカリフォルニア大学アーバイン校 客員研究員．ソフトウェア工学，要求工学に関する研究開発に従事．2007 年 慶應義塾大学 大学院 博士課程修了． 博士（工学）．