



SOKA University

Faculty of Science and Engineering
Department of Information Systems Science

創価大学 理工学部 情報システム工学科 卒業論文

2022年度

論文題目

OTMM を用いた糖鎖の解析手法
Glycan Analysis Method Using OTMM

学籍番号

1958207

氏名

戸塚 健人

指導教員

篠宮 紀彦

目次

第 1 章	序論	1
1.1	生命科学の概要	1
1.2	糖鎖の概要	3
第 2 章	糖鎖とコンピューターサイエンス	12
2.1	糖鎖と木構造	12
2.2	糖鎖と確率	14
第 3 章	研究の概要	15
3.1	研究背景	15
3.2	研究目的	16
3.3	研究内容	16
第 4 章	先行研究	18
4.1	HTMM	18
4.2	PSTMM	19
4.3	profile PSTMM	20
4.4	OTMM	20
4.5	各確率モデルの計算量	21
第 5 章	本研究で用いる確率モデル	22
5.1	OTMM の概要	22
第 6 章	実験	27
6.1	実験の内容	27
6.2	実験結果 I	29
6.3	実験結果 II	35

第 7 章	考察	50
7.1	正確性に関する考察	50
7.2	効率性に関する考察	50
第 8 章	結論	54
8.1	結論	54
8.2	今後の展望	54
謝辞		55
参考文献		56
付録 A	レクチンと OTMM	59
A.1	レクチンが認識する糖鎖の学習方法	59
付録 B	実装で生じたエンジニアリング的な問題	60
B.1	対数変換	60
B.2	float 型と decimal 型	64
付録 C	OTMM のソースコード	65

第 1 章

序論

1.1 生命科学の概要

1.1.1 生命の仕組み

生命は謎に満ちている。その謎を 1 つ 1 つ解き明かしても、また新たな謎が出現する。本研究によって、このような謎が少しでも解き明かされ、生命現象への解像度が上がることを期待する。

生命は絶妙なバランスを保ちながら変化し続ける。福岡 [1] によると、生命とは「動的平衡 (*dynamic equilibrium*)」の流れであるという。動的平衡とは、図 1.1 のように「身体の構成物質が絶え間なく入れ替わり、一種の平衡状態になること」である。このような「流れ」によって、生命はその秩序を保っている。

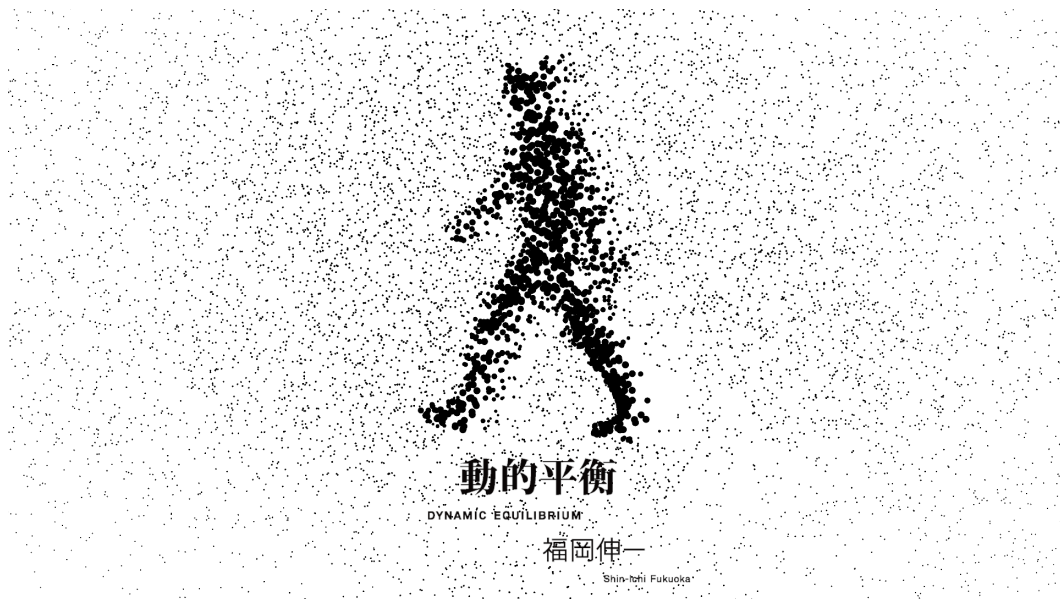


図 1.1 動的平衡のイメージ [2]

生物は、「動的平衡」によって支えられている。そもそも、タンパク質など生物の構成物質は、時間が経過するにつれて劣化し、最終的には崩壊する。この現象は「エントロピー増大の法則」と呼ばれ、エントロピー (劣化による乱雑さ) が蓄積することで生物は死に至る。つまり、生物が生き続けるには構成物質の劣化を抑えて、体内の秩序を保ち続けなければならない。動的平衡は、この劣化によるエントロピーの増加を抑える仕組みである。具体的には、劣化が生じる前に自ら先回りして身体の構成要素を分解し、新たな物質でその要素を入れ替えることを指す。このようにすることで、体内にエントロピーが蓄積することを防ぎ、正常な生命活動を維持できるようにする。すなわち、ダイナミックに古い物質を新しい物質に入れ替えることで、生物は生き長らえることができるのである。

また、動的平衡において、形質を維持するためには、平衡状態 (物質が入れ替わっても見かけ上変化が無い安定な状態) を維持する必要がある。具体的には、「適切な場所に適切な物質を入れ替えること」が必要である。これを支えるのが「相補性」という概念である。例えば、タンパク質の構造は鍵と鍵穴の関係になっており、体内の適切な場所に組み込まれるようになっている。つまり、生命の構成物質は、基本的にジグソーパズルのような構造をとり、相補的に結合しあっている。これによって、ある物質を分解し、組織に穴が開いても、即座に適切な物質を入れることができるのである。この「相補性」は、動的平衡だけでなく、様々な生命現象においても重要な役割を果たす。言い換えれば、物質の「構造」は、生命現象での役割 (機能) を決定する。以上のようなことから、生命現象を理解するには、その物質の「構造」を理解することが重要である。

1.1.2 生物学の変遷

糖鎖研究の概要を理解するには、近代生物学の歴史を理解する必要がある。まず、近代生物学では、DNA・タンパク質・糖鎖の順で研究が進められてきた [3]。そもそも、DNA は遺伝情報を担う物質であり、A, C, G, T と表される「ヌクレオチド」という物質で構成されている。この DNA が RNA という物質に写し取られ (転写), RNA の情報を基にしてタンパク質が合成される (翻訳)。さらに、タンパク質に「糖鎖」が付くことで様々な機能が付与される。糖鎖は、翻訳後にタンパク質を修飾することから、「翻訳後修飾子」と呼ばれている。図 1.2 に、糖タンパク質が作られるまでの流れを示す。

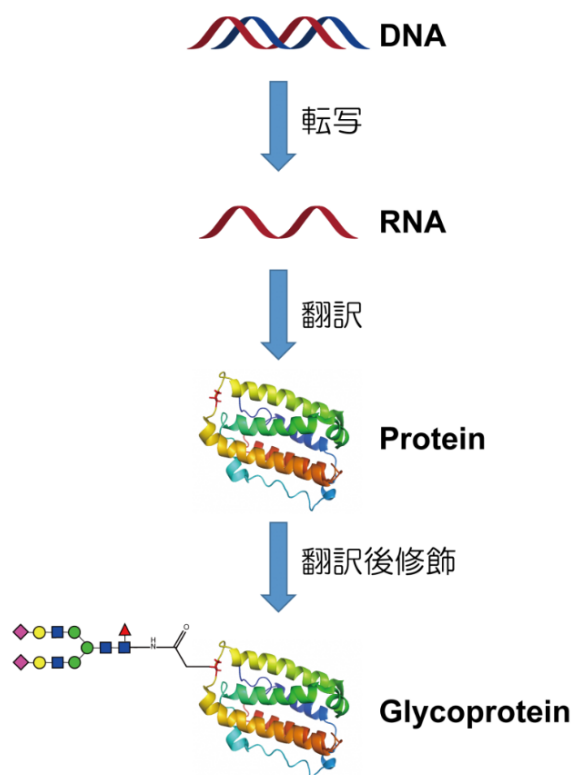


図 1.2 糖鎖によるタンパク質の修飾 [4]

近代生物学の歴史も同様の流れを辿る．まず，1953 年，科学誌「ネイチャー」にワトソンとクリックの *DNA* の 2 重らせん構造に関する論文が掲載された．その後，*RNA* の研究やヒトゲノム計画などが行われ，核酸に関する様々な事実が解き明かされた．そして，研究の標的はタンパク質に移った．タンパク質は，20 種類のアミノ酸から構成される物質であり，現在もその立体構造や性質に関する研究が進められている．このように，*DNA*・タンパク質の研究が進められるにつれて，「糖鎖」という物質の重要性が理解されるようになった．そして現在，糖鎖研究が行われている．

1.2 糖鎖の概要

1.2.1 糖鎖

糖鎖は，ヌクレオチドの鎖である *DNA*・アミノ酸の鎖であるタンパク質に次ぐ「第 3 の生命鎖」と呼ばれている．この糖鎖は，グルコースやガラクトースなど分子量が 200 前後の糖が繋がって生成される「糖の鎖」であり，多種多様な単糖が分岐などを経て繋がっているため，非常に複雑な構造を持つ．

1.2.2 糖鎖の性質

糖鎖は原核生物や哺乳類、植物に至るまであらゆる生物種に存在することがわかっている [4]。また糖鎖は、図 1.3 や図 1.4 のように細胞表面のタンパク質や脂質に結合しており、多くの生命現象に関与していることがわかっている。具体的には、発生、免疫、神経伝達、細胞間コミュニケーション、タンパク質の品質管理などに糖鎖が利用される。また、その機能 (構造) は糖鎖単体では決まらず、タンパク質や脂質などの「糖鎖が結合する相手」や時期や環境などの「状況」に応じて大きく変化する。このように、糖鎖は結合している物質の状態を表せるので、「情報分子」とも呼ばれている [5]。

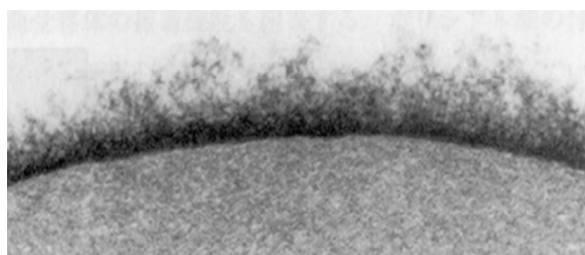


図 1.3 細胞表面に毛のように付着する糖鎖 [6]

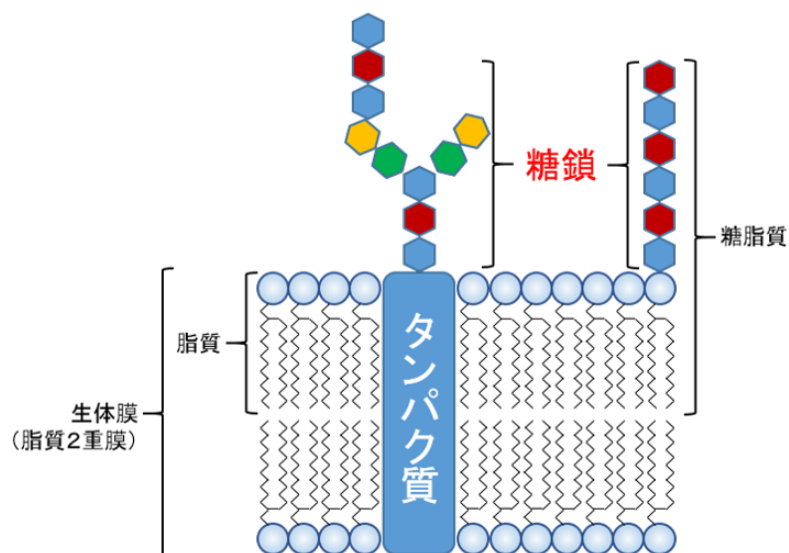


図 1.4 複合糖鎖 [7]

1.2.3 単糖

そもそも、糖とは「炭水化物から派生する一連の化合物」[8]である。炭水化物は、一般式で $C_nH_{2n}O_n$ と表され、文字通り炭素と水が化合した物質である。また、それ以上加

水分解できない糖を単糖と呼ぶ。図 1.5 に代表的な単糖を挙げる。

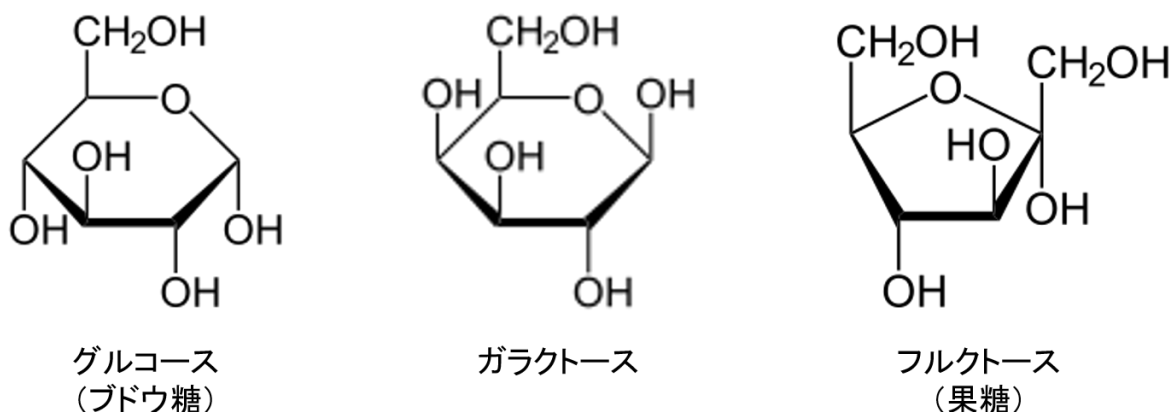


図 1.5 代表的な単糖

左からそれぞれ、グルコース、ガラクトース、フルクトースであり、化学式はいずれも $C_6H_{12}O_6$ で表現される。単糖はこのような環状の形式 (ハース投影式) で表現される。このハース投影式では、環状構造を形成する炭素は省略される。ただし、自然界においては、単糖は直鎖状と環状の2種類の平衡状態で存在し、必ずしもハース投影式のように環状で観測できるとは限らない。さらに、グルコースなどの六員環構造における各炭素原子は、 sp^3 混成軌道からなる。したがって、コンフォメーション (立体構造) は、熱力学的に安定な「椅子型」や「舟型」になることがあり、ハース投影式とはその見た目が一致しない可能性が高い [8]。

1.2.4 結合様式

次に、図 1.6 にアノマーの関係性のグルコースを挙げる。アノマーとは環状構造の不斉炭素に起因する立体異性体のことであり、 α と β の2種類が存在する [8]。ここで、不斉炭素とは「結合している原子または原子団が四つともすべて異なっている炭素原子」であり、立体異性体とは「同じ分子式であるが、その立体構造が異なる化合物」である。また、単糖の炭素には番号が付けられており、例えばグルコースでは1位から6位までの6つの番号が付けられる。

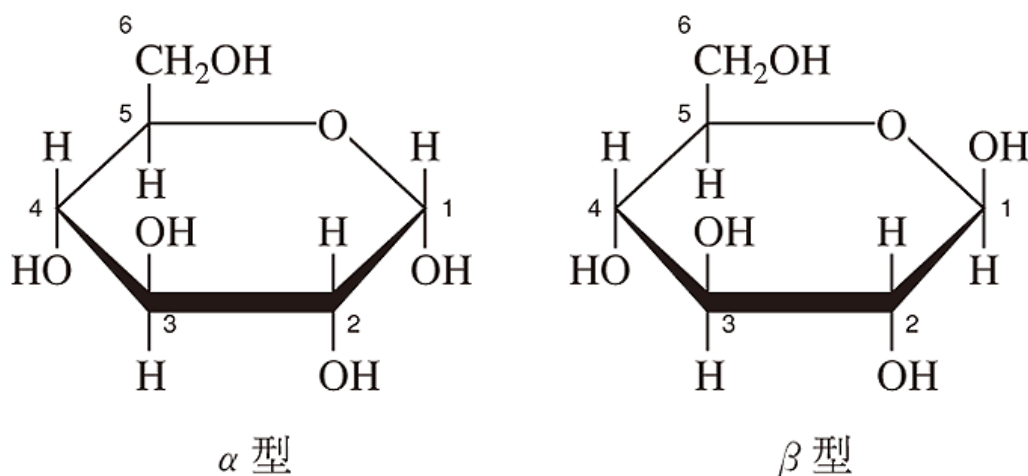


図 1.6 アノマーの関係性のグルコース [9]

図 1.6 のグルコースは、1 位の不斉炭素に結合している水酸基 (OH) の位置の違いによって、 α か β に分類される。このように、単糖はその立体構造の違いによって、2 種類の形態に分類される。

以上を踏まえて、糖の結合様式は、「炭素の番号」と「アノマーの種類」で決まる。例えば、 α -グルコースの 1 位と α -グルコースの 4 位が結合 (脱水縮合してグリコシド結合を形成) した場合、その結合様式は「 $\alpha 1-4$ 」と表される。他の単糖同士が結合した場合も、同様の考え方で結合様式を表現することができる。

1.2.5 糖鎖研究の難しさ

前述の通り、糖鎖は単糖などの「糖」が分岐を経て繋がり、木構造を形成する物質である。また、多種多様な構造を取り、その本質的な機能は未だに解明できていない。糖鎖は「細胞の顔」とも言われており、細胞の状態によってその形状が変化する。さらに、糖鎖の細胞への付加率は一定ではなく、結合する際の構造も常に同じではない [8]。このような「糖鎖の不均一さ」によって、糖鎖解析は困難になっている。加えて、DNA、タンパク質はシーケンス (配列) で表せるデータなのに対し、糖鎖は「木構造」でしか表せないデータであるので、コンピューターで扱いにくいという側面もある。以上のような理由から、糖鎖研究は困難を極める。また、糖鎖は「糖転移酵素」という酵素によって、逐次的に単糖が結合して伸長していく。この糖転移酵素は、糖鎖構造を認識して特異的に糖を結合させるため (基質特異性)、そのメカニズムの理解には結合様式などを含めた詳細な構造の把握が必要となる。

1.2.6 レクチンとは

糖鎖はレクチンと総称されるタンパク質に特異的に認識されることがわかっている [8]. すなわち, レクチンが糖鎖の多様な働きに関わっていると考えられている. 図 1.7 にレクチンが糖鎖を認識する様子を示す.

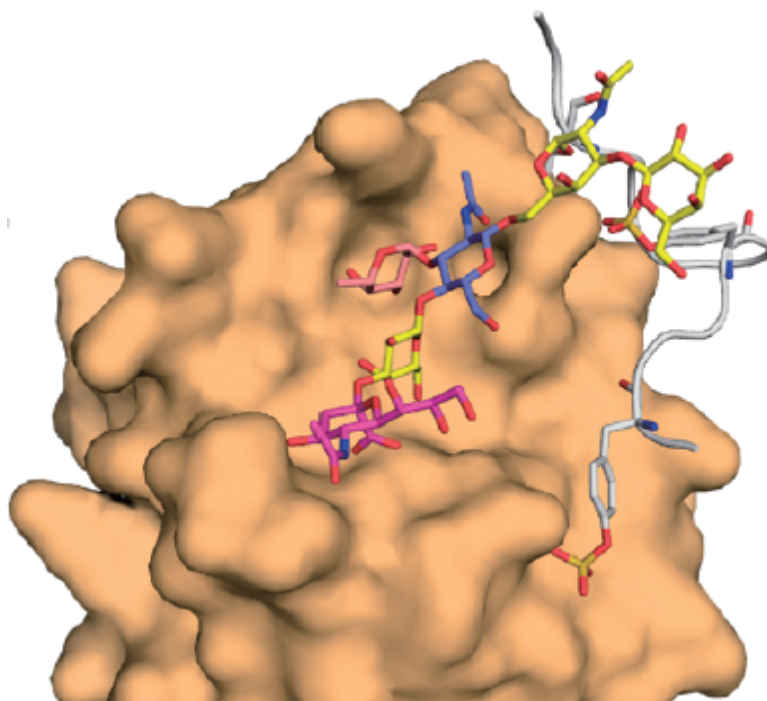


図 1.7 レクチン (橙色) の糖鎖認識 [5]

このようにレクチンは, 特定の糖鎖構造にパズルピースのようにピッタリとはまるので, 多様な糖鎖から特定の糖鎖を同定することができる [5]. しかしながら, 一部のレクチンは様々な糖鎖構造を幅広く認識する. 例えば, レクチンの一種である「ガレクチン」はガラクトースを含む糖鎖を幅広く認識し結合する. これは, ある機能が使えなくなっても動的に対処するために, 生命が用意した冗長性であると考えられる. したがって, レクチンが認識する糖鎖を理解するには, 認識される糖鎖構造の「共通パターン」を調べる必要がある.

1.2.7 糖鎖の利用

糖鎖は様々な生命現象と関連している。以下では、糖鎖が関連している生命現象を3つ紹介する。

1つ目は、血液型の区別である。血液型は、赤血球の表面上の糖鎖の違いによって生じる [10]。図 1.8 に赤血球上の糖鎖構造を示す。

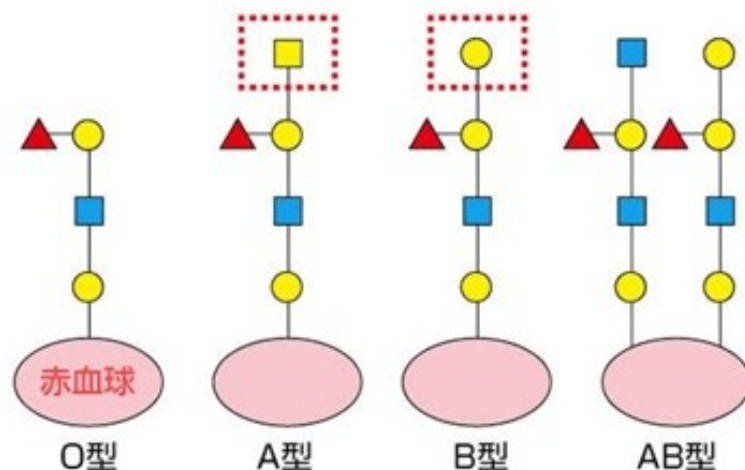


図 1.8 血液型と糖鎖の関係 [10]

O 型の糖鎖が基本となって、A 型、B 型、AB 型の糖鎖に派生していることがわかる。このように、糖鎖によって様々な区別が可能となるため、糖鎖は「細胞の顔」とも呼ばれている。

2つ目は、インフルエンザウイルスの感染である。インフルエンザウイルスは、その表面に「シアル酸」という糖を認識するレクチンを備えており、そのレクチンが宿主の細胞表面にある糖鎖に結合することで感染を引き起こす。また、ヒトインフルエンザウイルスは「 $\alpha 2-6$ 」結合のシアル酸を含む糖鎖を認識するのに対して、トリインフルエンザウイルスは「 $\alpha 2-3$ 」結合のシアル酸を含む糖鎖を認識することが知られている [8]。このように、糖鎖中の単糖だけでなく、その結合様式も生命現象に影響を与える。また、インフルエンザ薬「リレンザ」は、糖鎖を活用してウイルスの増殖を防いでいる。

3つ目は、病気との関係である。糖鎖はアルツハイマー病とも関連しており、理化学研究所 [11] によると、ある特定の糖鎖を持たないマウスでは、認知症の原因となるアミロイド β というタンパク質の蓄積が激減したという。また、産業技術総合研究所では、癌細胞の表面に出現する糖鎖をレクチンで認識し、癌細胞に薬をダイレクトに作用させる手法 [12] を開発している。このように、糖鎖はバイオマーカーとしても機能するため、医療分野でもその活用が期待されている。

1.2.8 N型糖鎖

糖鎖は1兆を超える種類があるといわれている [6]. したがって、本研究では「N-結合型糖鎖」に種類を絞って研究を行う. N-結合型糖鎖とは、タンパク質を構成するアミノ酸「アスパラギン (Asn)」に N-アセチルグルコサミン (GlcNAc) が β 結合した糖鎖である. 図 1.9 に結合の様子を示す.

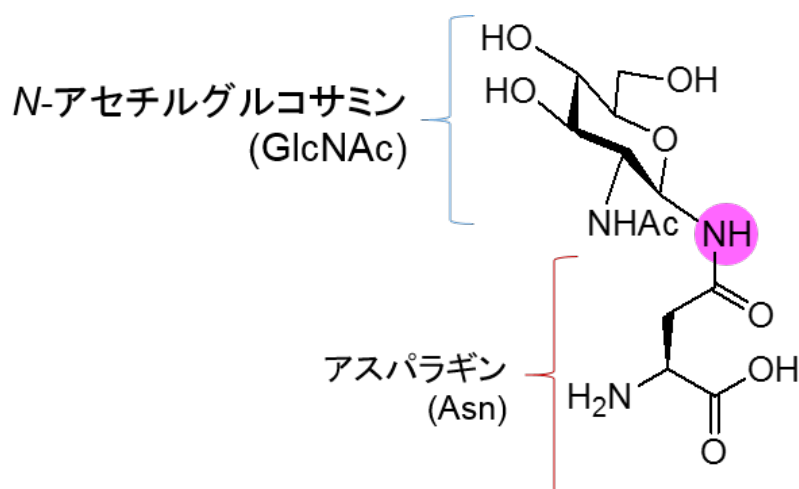


図 1.9 N-結合型糖鎖の根元部分 [13]

アスパラギン (Asn) 残基に N-アセチルグルコサミン (GlcNAc) が結合するので、糖鎖の根元は必ず GlcNAc となる. また、根元の GlcNAc から、単糖が伸長していき、最終的に3種類のタイプに分類できる.

ここで、図 1.10 に糖鎖構造中の単糖の記号を示す.

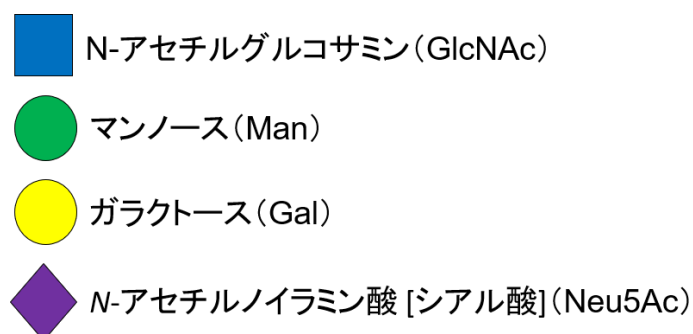


図 1.10 単糖の記号

また、辺上にな書かれている文字は結合様式を示す. 以降、糖鎖表示法は、*Symbol Nomenclature for Glycans (SNFG)* [14] を用いる.

1つ目のタイプは, *High mannose* 型である.

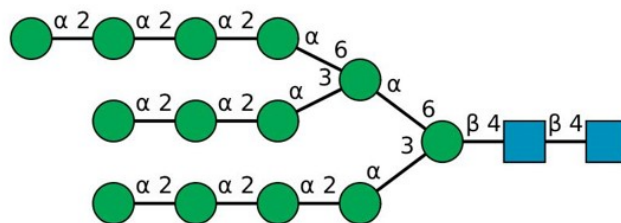


図 1.11 High mannose 型

図 1.11 より, *High mannose* 型の *N*-結合型糖鎖は末端付近にマンノース (*Man*) が多数存在することが特徴である. このタイプの糖鎖は, 細胞内タンパク品質管理で重要な役割を果たしている [13].

2つ目のタイプは, *Complex* 型である.

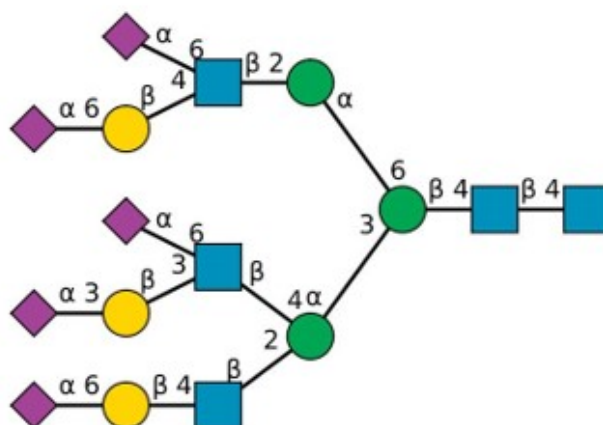


図 1.12 Complex 型

図 1.12 より, *Complex* 型の *N*-結合型糖鎖は末端付近に *N*-アセチルグルコサミン (*GlcNAc*), ガラクトース (*Gal*), シアル酸 (*Sia*) が存在することが特徴である. このタイプの糖鎖は, 抗体, 糖タンパクなどのバイオ医薬品で利用されている [13].

第 2 章

糖鎖とコンピューターサイエンス

2.1 糖鎖と木構造

糖鎖は複雑な構造を取るので，グラフ理論における「木」を用いてその構造を表す．「木」とは，図 2.1 のような「連結かつサイクルがないグラフ」のことである．ここで，連結とは「グラフのどの 2 頂点についても，その 2 頂点を結ぶ歩道が存在すること」であり，サイクルとは「始点と終点一致し，それ以外の頂点は全て異なる歩道のこと」である．

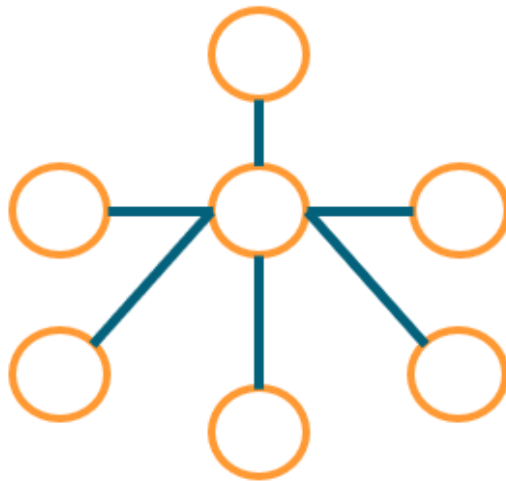


図 2.1 木の例

木はコンピューターサイエンスにおける重要な概念であり，データの格納，データのソート，情報の探索などで良く用いられる [15]．本研究では，糖鎖構造のデータを格納する際に「木」を用いる．

ここで、糖鎖と木構造の対応関係を整理する。木における「ノード」に相当するのが糖鎖を構成する「単糖」であり、「エッジ」に相当するのが単糖同士の「結合」である。図 2.2 に木構造で表した糖鎖の例を示す。

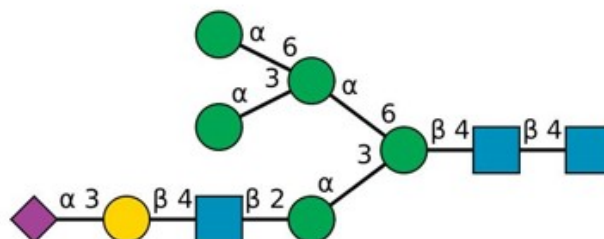


図 2.2 木と糖鎖の関係性

右端のノードが木における根、左端のノードが木における葉であり、各ノードが単糖に相当する。また、各ノードを繋ぐ辺が、単糖同士の結合を表す。このように、糖鎖構造は木で表すことができる。

なお、実際の糖鎖構造のデータはテキストデータであり、図 2.2 の構造のデータは以下のよう表される。

$Neu5Ac(a2 - 3)Gal(b1 - 4)GlcNAc(b1 - 2)Man(a1 - 3)[Man(a1 - 3)[Man(a1 - 6)]Man(a1 - 6)]Man(b1 - 4)GlcNAc(b1 - 4)GlcNAc$

ただし、糖鎖構造のデータは右から左に読むのが通例となっており、上記のデータも右から左に読むことで、木構造の根から葉に向かって「深さ優先探索 (DFS)」の順序で読むことができる。また、本研究では *IUPAC Condensed* というフォーマットのデータを用いる。

ここで、本研究で用いる木構造の定義を行う。ラベル付き順序木 (*Labeled ordered tree*) とは、図 2.3 のような、深さに応じて順序が割り振られた木である。この順序は、幅優先探索の順序に対応している。また、兄弟 (*sibling*) ノードは左に行くほど (長男に近づくほど) 番号が若くなる。すなわち、長男ノードは同じ親を持つノードで最も番号が若いノードとなる。

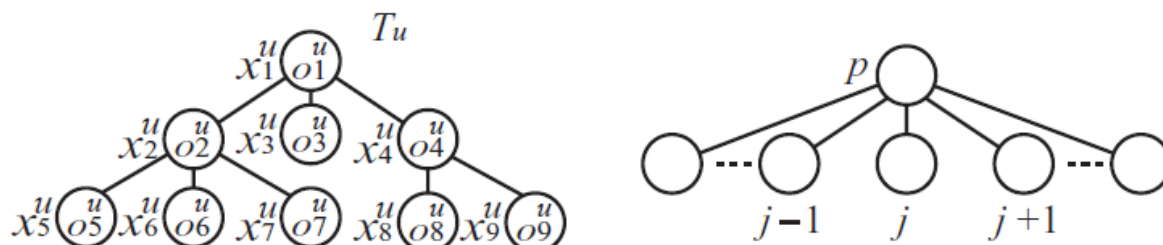


図 2.3 ラベル付き順序木の例 [16]

2.2 糖鎖と確率

前述の通り、一部のレクチンは、特定の糖鎖を特異的に認識するのではなく、ある構造を持った糖鎖を幅広く認識する。したがって、あるレクチンが認識する糖鎖を分析する際は、「どの位置に」、「どのような単糖が」出現しやすいかを確率的に考える必要がある。また、生命現象を扱う特性上、実験で常に同じ結果が得られるとは限らない。すなわち、本研究で使用するデータは「曖昧性 (ノイズ)」を持つ。さらに、実験データはある生命現象の一部を切り取った「離散的なデータ」であるので、その本質を捉えるには、様々なデータを学習してその傾向・特徴を捉える必要がある。このようなことから、本研究では確率を使用する人工知能的なアプローチを用いる。

第 3 章

研究の概要

3.1 研究背景

大規模なデータセットから頻出パターンを取り出すことは、データマイニングにおける重要な課題である。従来のデータマイニングでは、データが構造化されており、シーケンスなどの比較的単純なパターンがマイニングの対象となっていた。しかし、近年、テキストマイニング、Web マイニング、バイオインフォマティクス [17, 18] などの分野で、より複雑で非構造化（または半構造化）された木やグラフなどのデータセットが登場するようになった。具体的には、Web でよく利用される XML などが挙げられ、この領域でのデータのマイニングは重要度が増している。この XML 文書は木構造テキスト形式であり、半構造化データに分類される。より具体的には、「ラベル付き順序木」のデータセットであり、近年はカーネル関数 [19] や頻出パターンマイニング [20] に基づくマイニング手法も登場している。

糖鎖は、DNA、タンパク質に続く第 3 の生体分子であり [21]、糖鎖情報学の登場により、近年、糖鎖構造データ及びそのアノテーションデータが増加している [17, 18]。また、糖鎖は木構造であり、そのノードは単糖でラベル付けされている。さらに、糖鎖構造中で同じ親に結合する単糖は順序が決まっているので、糖鎖は「ラベル付き順序木 (*labeled ordered trees*)」とみなすことができる。この糖鎖構造における葉の並び方は、認識やシグナル伝達など様々な生命現象で利用されていることが知られている [21]。しかしながら、技術的な課題により、糖鎖の構造を詳細に決定することは困難である。このようなことから、糖鎖構造について学ぶべきことは大量に存在する [22]。

3.2 研究目的

生命現象は、様々な物質が相補的に構造を認識し合うことで成り立っている。したがって、生命現象を理解するには物質の「構造」を捉えることが重要である。糖鎖も、その構造が酵素やウイルスなどの識別子として利用されており [23]、糖鎖構造には何らかの複雑なパターンが内在していると考えられる。したがって、本研究では「糖鎖構造の共通パターン」を既存の研究よりも正確に取得することを目指す。これによって、複雑な糖鎖構造からその特徴を正確に抽出することが可能になり、糖鎖が関係する生命現象のメカニズムをより詳細に理解することに繋がる。

以上をまとめると、本研究ではコンピュータサイエンス知識と糖鎖生物学の知識を融合し、これまでの研究では見出せなかった糖鎖に潜む法則性・必然性を発見することを目指す。そして、そのような発見が「糖鎖の本質的な機能の解明」、「糖鎖を目印とした病気の発見」、「創薬・ワクチン開発」などで役立てられることを期待する。

3.3 研究内容

糖鎖には未だに不明な点が多く、その解析は困難を極める。したがって、本研究では、糖鎖の種類を N -結合型糖鎖に絞って解析を行う。また、本研究では、 $OTMM$ という確率モデルに新しい形式の糖鎖構造のデータを入力することで、既存の研究よりも正確な解析結果を得ることを目指す。具体的には、既存の研究では「単糖」のみのデータを学習していたが、本研究では「単糖」と「結合様式」を組み合わせた新しいデータ (以降、単糖 + 結合様式) を学習する。研究の手順は以下の通りである。

1. N -結合型糖鎖のデータを確率モデル ($OTMM$) に入力する。
2. 入力された糖鎖構造のデータを $OTMM$ で学習する。学習では EM アルゴリズムを用いる。
3. 状態遷移図を得る。この状態遷移図には学習結果が反映されている。
4. $Viterbi$ アルゴリズムを用いて、糖鎖構造の解析を行う (学習結果をわかりやすくビジュアル化する)。

アルゴリズムの設計手法としては、動的計画法と機械学習が用いられている [24]。

図 3.1 に学習で得られる状態遷移図の例を示す．この図では，「各状態にどのような単糖が出現しやすいか」や「どのような状態に遷移しやすいか」を読み取ることができる．ここで，状態とは図 3.1 における番号を指す．また，この状態遷移図は木構造における状態遷移を表すので，実質的に糖鎖構造の共通パターン (特徴) を得たことに相当する．

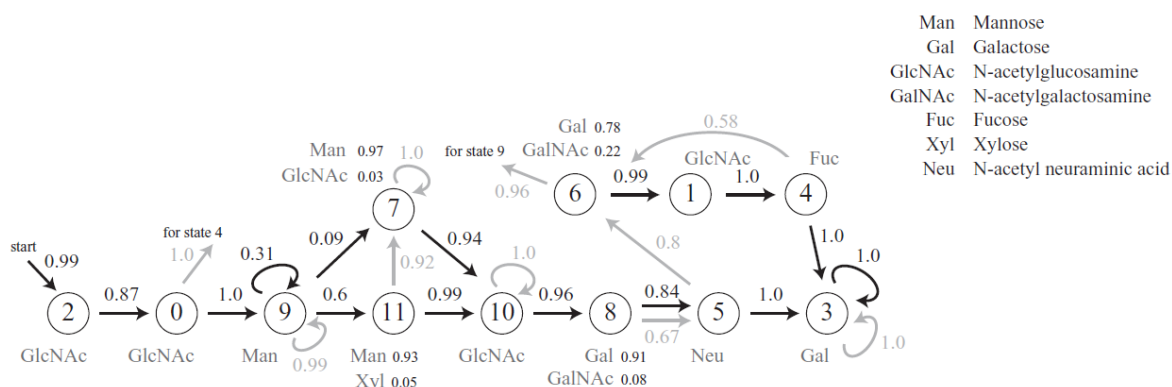


図 3.1 OTMM で得られる状態遷移図の例 [16]

前述の通り，糖鎖の働きの全容を捉えるには，糖鎖とレクチンの相互作用を考えることが重要である．上記と同じ手法を用いて，レクチンの種類別に糖鎖を解析することで，レクチンが認識する糖鎖構造の共通パターン (特徴) も得ることができる (付録 A 参照)．

第 4 章

先行研究

糖鎖に関するデータなど，生物学的な実験から得られるデータにはノイズ (曖昧性) が含まれる．確率モデルはノイズに強いため，生物学的なデータを用いる機械学習やデータマイニングにおいて非常に効率的なアプローチとなる．本研究で使用する確率モデルは隠れマルコフモデル (*HMM*) [24, 25] がベースとなっており，音声認識やバイオインフォマティクスなど多くの分野で適用されている．しかし，*HMM* は *DNA* やタンパク質などの 1 次元の配列データにしか適用できない．このようなことから，現在，木構造 (糖鎖) で使用できる様々なモデルが提案されている．本章で紹介する先行研究は，*HMM* をベースにした機械学習を用いることで，糖鎖構造のデータからその共通パターンを状態遷移図として取得することができる．

4.1 HTMM

Hidden tree Markov model (HTMM) [26] はラベル付き木の確率モデルであり，木構造における親子関係を扱うことができる．しかし，兄弟間の依存関係を扱えないため，ラベル付き順序木には適用できない．したがって，ラベル付き順序木である「糖鎖」を扱う本研究には適さないモデルである．図 4.1 に *HTMM* の例を示す．

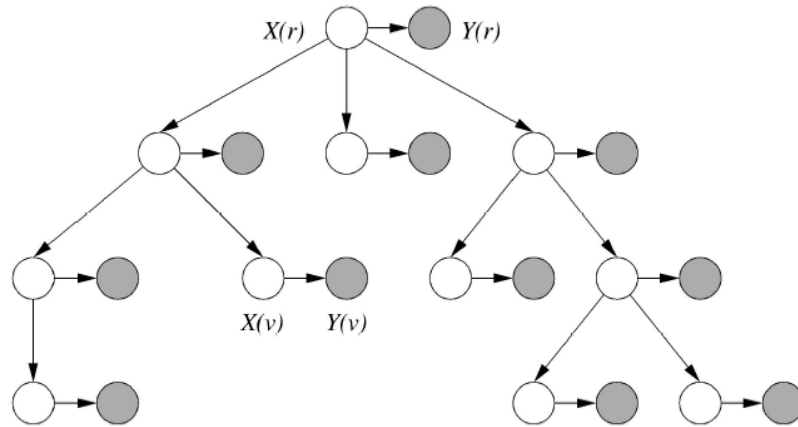


図 4.1 HTMM の例 [26]

4.2 PSTMM

親子の依存関係と兄弟の依存関係の両方を含むラベル付き順序木のモデルとして、*Probabilistic sibling – dependent tree Markov model*(*PSTMM*)[27] がある．図 4.2 に *PSTMM* の例を示す．

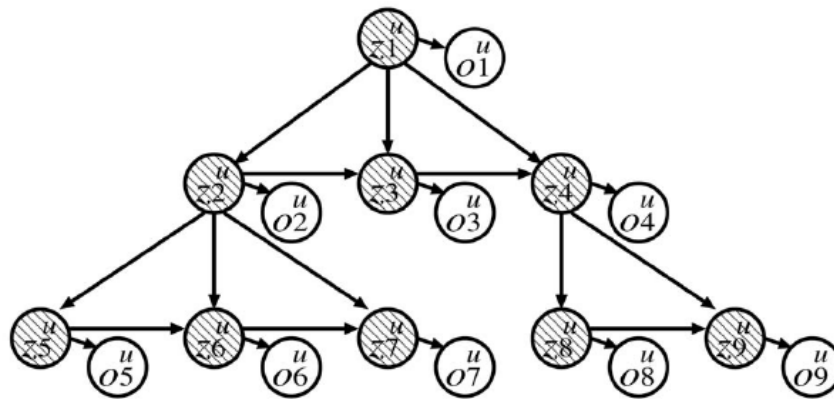


図 4.2 PSTMM の例 [27]

このモデルは、親子や兄弟でノードが依存しあっているため、糖鎖構造を詳細に表現できる．しかしながら、*PSTMM* の時間計算量と空間計算量は、おおよそ *HTMM* の状態数に比例して増加する．この増加は、2つの深刻な問題を引き起こす．1つ目は、*PSTMM* を大規模なデータに適用すると、時間計算量と空間計算量が大きくなり、実用性がなくなるとのことである．2つ目は、糖鎖生物学で用いるような小さなサイズのデータでは、過学習を起こしてしまうことである．これら2つの理由から、*PSTMM* を現実の問題に適用するためには、その複雑さを軽減する必要がある．

4.3 profile PSTMM

PSTMM を改良した確率モデルが *profile PSTMM*[28] である．図 4.3 に *profile PSTMM* の例を示す．

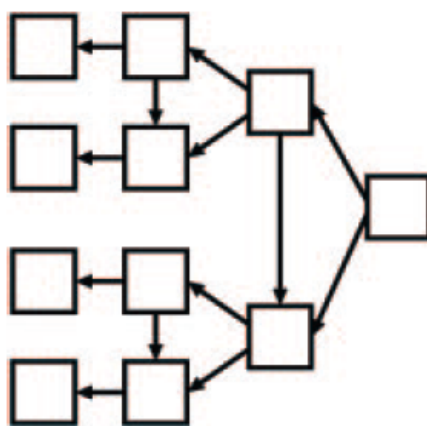


図 4.3 *profile PSTMM* の状態遷移図 [28]

profile PSTMM では，状態遷移図 (学習結果) を事前に固定しておくことで，糖鎖構造の共通パターン (糖鎖 profile) を容易に取得することができる．言い換えれば，*profile PSTMM* では事前に欲しい糖鎖構造を設定する必要がある．

4.4 OTMM

HTMM や *PSTMM* を改善したモデルが，*Ordered tree Markov model(OTMM)* である．具体的には，*HTMM* では扱えなかった兄弟関係を扱えるようにし，*PSTMM* のような複雑さを軽減したモデルである．特に，*OTMM* の親子の依存関係は長男とその親の間に限定される．このようなことから，*OTMM* は 1 階マルコフ連鎖モデルとみなすことができる．図 4.4 に *OTMM* の例を示す．

[16] では，*OTMM* の有効性を，人工的に生成したデータと，実際の糖鎖構造のデータを用いて評価した．その結果，*OTMM* の学習時間は *PSTMM* の学習時間に比べて大幅に短縮されることがわかった．さらに，*OTMM* は *PSTMM* の予測能力を維持したまま，学習データの過学習の問題を改善できることもわかった．

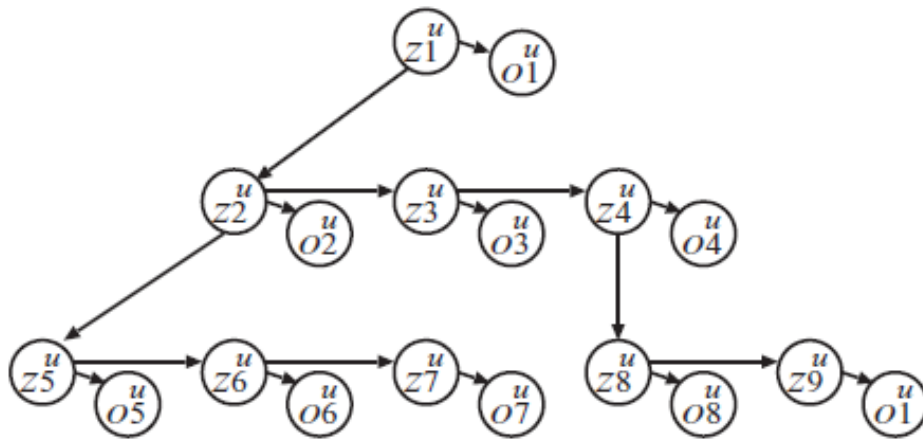


図 4.4 OTMM の例 [16]

4.5 各確率モデルの計算量

表 4.1, 4.2 に, 各確率モデルの時間計算量と空間計算量を *Big-O* 記法 [24] で示す.

表 4.1 確率モデルの時間計算量

	Time
HTMM	$O(T \cdot S ^2 \cdot V)$
PSTMM	$O(T \cdot S ^3 \cdot V \cdot C)$
OTMM	$O(T \cdot S ^2 \cdot V)$

表 4.2 確率モデルの空間計算量

	Space
HTMM	$\max\{O(S \cdot V), O(S ^2), O(S \cdot \Sigma)\}$
PSTMM	$\max\{O(S \cdot V), O(S ^2), O(S \cdot \Sigma)\}$
OTMM	$\max\{O(S ^2 \cdot V), O(S ^3), O(S ^2 \cdot \Sigma)\}$

HTMM と OTMM は, 時間・空間計算量で PSTMM よりも効率的なアルゴリズム [24] であることがわかる. このようなことから, 本研究では, 正確性と効率性が最も優れている OTMM を用いて研究を行う.

第 5 章

本研究で用いる確率モデル

5.1 OTMM の概要

本研究では，確率モデル *OTMM*(*Ordered tree Markov model*) を用いて実験を行う．

5.1.1 OTMM の定義

以下，*OTMM* は 5 つの要素で定義され，観測可能なラベルと観測できない状態が存在するのが特徴である [16]．

OTMM

$S = \{s_1, \dots, s_{|S|}\}$: (観測できない) 状態の集合

$\Sigma = \{\sigma_1, \dots, \sigma_{|\Sigma|}\}$: (観測可能な) ラベルの集合

$\Pi = \{ \pi(l) \}$: 初期状態確率分布

初期状態とはルートの状態を意味する

例： $\pi(l)$ はルート (根) の状態が s_l である確率

a : 状態遷移確率分布 (2 種類)

$\alpha[q, m]$: 状態 q の親から状態 m の子へ遷移する確率

$\beta[l, m]$: 状態 l の兄から状態 m の弟へ遷移する確率

b : ラベル出力確率分布

例： $b[l, \sigma_h]$ はノードの状態が s_l の時，ラベル σ_h が出力される確率

さらに, $OTMM$ で用いる様々な用語の定義を行う.

- $\mathbf{T}=\{T_1, \dots, T_{|\mathbf{T}|}\}$ をラベル付き順序木の集合とする.
 - ただし, $T_u = (V_u, E_u)$ とする.
 - 本研究では, T_u が1つの「糖鎖構造のデータ」に対応する.
- 頂点集合 V_u を $V_u=\{x_1^u, \dots, x_{|V_u|}^u\}$ とする.
 - 本研究では, x_i^u は糖鎖構造における1つのノードに対応する.
- 辺集合 E_u を $E_u \subset V_u \times V_u$ (頂点集合の直積) とする.
- ノード x_j^u はラベル $o_j^u \in \Sigma$ を持つ.
 - ただし, $\Sigma=\{\sigma_1^u, \dots, \sigma_{|\Sigma|}^u\}$ とする.
 - 本研究では, ラベル o_j^u が「単糖」や「単糖 + 結合様式」に対応する.
- θ を確率モデルのパラメータのセットとする.
 - 具体的には, 初期状態確率分布、状態遷移確率分布、ラベル出力確率分布の3つの確率分布を指す.
- $L(\mathbf{T})$ を確率モデルに対する入力データ全体の尤度とする.
 - 具体的には, モデル (パラメータ) がどの程度入力データにフィット (適合) しているかを示す指標である.

また, 各ノードは, 直接観測することができない隠れた「状態」を持つ. そして, 各状態は (観測可能な) ラベルを確率的に出力する. 例えば, ノード x_3^2 には「状態1」という状態が割り振られ, 状態1はラベル「*Man*」を70%の確率で, ラベル「*GlcNAc*」を30%で出力するという具合である. 図5.1では, z がノード (状態) を, o が出力されたラベル (単糖など) を表す.

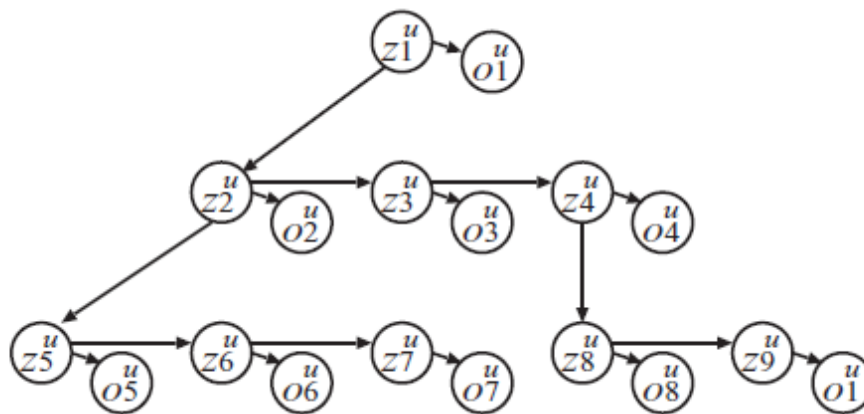


図 5.1 $OTMM$ における状態とラベル [16]

$OTMM$ の学習では, 糖鎖構造のデータ (観測可能なラベルの列) から, 直接観測することができない糖鎖構造の共通パターン (状態遷移図) を得ることを目指す.

5.1.2 OTMM のアルゴリズム

確率モデルを実世界の問題に適用する場合、パラメータの学習 (*Learning*), *Most likely state path* の発見 (*Parsing*), 尤度の計算 (*Likelihood*) の 3 つの問題を解決する必要がある。

まず, *Learning* では *EM*(*Expectation – Maximization*) アルゴリズム [16](最尤推定) を用いる。これは *HMM* の *Baum – Welch*(*Forward – Backward*) アルゴリズムを拡張したアルゴリズムである。このアルゴリズムでは, 糖鎖構造のデータを入力し, 初期状態確率分布、状態遷移確率分布、ラベル出力確率分布 (3 種類の確率分布) を出力する。この 3 種類の確率分布を用いて状態遷移図を作成することで, 実質的に糖鎖構造の共通パターン (特徴) を抽出したことになる。図 5.2 に, 学習アルゴリズムの疑似コードを示す。

```

1: for each  $\theta_{i,j}$  do Initialize  $\theta$ ;
2: Calculate  $L^{(0)}(\mathbf{T})$  using the initial  $\theta$ ;
3:  $t := 0$ ;
4: repeat
5:   for each  $\theta_{i,j}$  do  $\mu_{\mathbf{T}}(\theta_{i,j}) := 0$ ;
6:   for  $u := 1$  to  $|\mathbf{T}|$  do
7:     for  $k := |V_u|$  downto 1 do /* B-up & R-to-L DP */
8:       for each  $q \in S$  do Calculate  $U_u(q, k)$ ;
9:       for each  $m \in S$  do Calculate  $B_u(m, k)$ ;
10:    for  $k := 1$  to  $|V_u|$  do /* T-down & L-to-R DP */
11:      for each  $q \in S$  do Calculate  $D_u(q, k)$ ;
12:      for each  $m \in S$  do Calculate  $F_u(m, k)$ ;
13:    for each  $\theta_{i,j}$  do Calculate  $\mu_u(\theta_{i,j})$ ;
14:    for each  $\theta_{i,j}$  do  $\mu_{\mathbf{T}}(\theta_{i,j}) := \mu_{\mathbf{T}}(\theta_{i,j}) + \mu_u(\theta_{i,j})$ ;
15:    for each  $\theta_{i,j}$  do Update  $\theta_{i,j}$  using  $\mu_{\mathbf{T}}(\theta_{i,j})$ ;
16:     $t := t + 1$ ;
17:  Calculate  $L^{(t)}(\mathbf{T})$  using the current  $\theta$ ;
18: until  $|L^{(t)}(\mathbf{T}) - L^{(t-1)}(\mathbf{T})| < \epsilon$ 
19: output  $\theta$ ;

```

図 5.2 OTMM における状態とラベル [16]

疑似コードでは, パラメータ θ を初期化した後, *Upward probability*(U), *Backward probability*(B), *Forward probability*(F), *Downward probability*(D) という 4 つの条件付き確率を求める。これら 4 つの確率は, 木構造の特定の部分 (部分木) でラベルが出力される確率を表す。例えば, *Upward probability* の $U(q, p)$ は, 「ノード p の状態が q のとき, ノード p をルートとする部分木 (*sub tree*) の全てのラベルが出力される確率」を表す。

図 5.3 の塗りつぶされている部分のラベル (単糖など) が出力される確率を求めるイメージである。

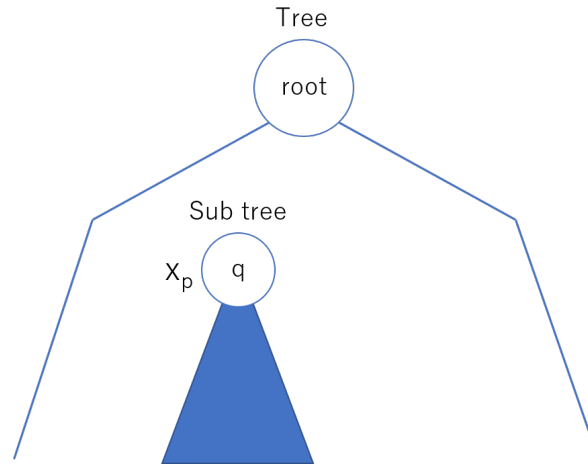


図 5.3 Upward probability のイメージ

そして, U , B , F , D を用いて, 期待値 μ を計算する (E ステップ). この期待値は, 全部で 4 種類あり, それぞれ $\mu(\alpha[q, m])$, $\mu(\beta[l, m])$, $\mu(b[l, \sigma_h])$, $\mu(\pi(l))$ と表される. ここで, 期待値とは「ある出来事が起こる平均回数」と言い換えることができる. したがって, 期待値 $\mu(\alpha[q, m])$ は, 「状態 q から状態 m へ親子間の遷移をする平均回数」と考えることができる. 他の期待値も同様に, $\mu(\beta[l, m])$ は「状態 l から状態 m へ兄弟間の遷移をする平均回数」, $\mu(b[l, \sigma_h])$ は「ノードの状態が s_l の時, ラベル σ_h が出力される平均回数」, $\mu(\pi(l))$ は「ルート (根) の状態が s_l である平均回数」と言い換えることができる.

したがって, パラメータ (3 種類の確率分布) は以下のように求めることができる (M ステップ).

$$\begin{aligned}\hat{\alpha}[q, l] &= \frac{\sum_u \mu(\alpha[q, l])}{\sum_u \sum_{l'} \mu(\alpha[q, l'])} \\ \hat{\beta}[q, l] &= \frac{\sum_u \mu(\beta[q, l])}{\sum_u \sum_{l'} \mu(\beta[q, l'])} \\ \hat{b}[m, \sigma_h] &= \frac{\sum_u \mu(b[m, \sigma_h])}{\sum_u \sum_{\sigma_i} \mu(b[m, \sigma_i])} \\ \hat{\pi}[m] &= \frac{\sum_u \mu(\pi[m])}{\sum_u \sum_k \mu(\pi[k])}\end{aligned}$$

上記のパラメータを求める式は全て、

$$\frac{\text{求めたい事象の回数}}{\text{全体の事象の回数}}$$

となっており、確率の定義そのものになっている。よって、上記の式でパラメータ (3 種類の確率分布) を求めることができる。

また、*EM* アルゴリズムは、期待値を求める *E* ステップとパラメータを更新する *M* ステップを交互に繰り返し、尤度の変化が一定の値 (しきい値) 以下になると学習が終了する。

次に、*Parsing* の紹介を行う。*OTMM* において、糖鎖構造のデータ (観測可能な出力ラベル列) から、その構造の特徴 (*Most likely state path*) を得る際は、*Parsing* を用いる。具体的には、*Viterbi* アルゴリズム [16] を使用する。このアルゴリズムでは、学習結果 (状態遷移図) を基に糖鎖構造を解析し、その構造の特徴 (*Most likely state path*) を得ることで「モデルが学習したことは何か」を間接的に理解する。言い換えれば、「ある糖鎖構造の特徴」が学習結果に基づいてビジュアル化される。また、*Parsing* は、「ある糖鎖構造上で最も可能性の高い状態の遷移 (*Most likely state path*)」を求めるので、*Likelihood* のアルゴリズムの Σ (総和) の部分を全て *max* にしたアルゴリズムになっている (ソースコードは付録 C を参照)。また、*Parsing* では、*Upward probability*(*U*) と *Backward probability*(*B*) を用いて計算を行う。

Likelihood アルゴリズムは文字通り、入力データ全体に対するモデルの尤度を計算するアルゴリズムである。本研究で使用する確率モデルは、本質的に初期状態確率分布、状態遷移確率分布、ラベル出力確率分布の 3 つの確率分布 (パラメータ) で定義できる。したがって、*Likelihood* は学習で得られた「パラメータ」がどのくらい入力データにフィット (適合) してるかを求める事と同義である (ソースコードは付録 C を参照)。

また、*OTMM* のアルゴリズムの設計手法は、動的計画法 (*Dynamic programming*) と機械学習である [24]。動的計画法 (*DP*) とは、問題を小さな部分問題に分け、部分問題の解を組み合わせて、問題全体を解く手法である。本研究では、条件付き確率 *F*, *B*, *U*, *D* を求める際などに使用している。また、機械学習とは、データを統計的に分析し、その結果に基づいて問題を解く手法である。本研究では、機械学習で糖鎖構造のデータを分析し、その構造の特徴を得ている。

第 6 章

実験

6.1 実験の内容

本研究では，観測可能な糖鎖構造のデータから，隠れている糖鎖構造の共通パターン(状態遷移図)を既存の研究よりの正確に得ることが目的である．実験では，「単糖」をラベルにした糖鎖構造のデータと「単糖 + 結合様式」をラベルにした糖鎖構造のデータをそれぞれ *OTMM* に入力し，結果の比較を行った．

実験に使用したデータは，*GlyCosmos* の *GlyCosmos Glycans*[29] で取得した *N*-結合型糖鎖の構造データである．このデータは，*IUPAC Condensed* というフォーマットで記述されており，データベースからランダムに集めたデータである．また，前処理を施して，*IUPAC Condensed* の構造中に「?」の表記が含まれない *Linkage defined saccharide* の糖鎖を使用した．すなわち，曖昧な構造がある糖鎖構造のデータは学習データから除外した．このようなことから，データ数は最大で 5195 ある．さらに，コンピューターのリソース的に扱えるデータ数には限界があり，一部実験では糖鎖構造のデータ数を制限している．具体的には，データが格納されている *csv* ファイルの上位 1000 個のデータを使用して一部の実験を行った．

表 6.1 に、データ数と糖鎖構造のノード数の関係を示す。

表 6.1 糖鎖構造のデータ数とノード数の関係

糖鎖構造のデータ数 (個)	平均ノード数 (個)	ノード数の中央値 (個)
100	11.850	11
500	12.398	12
1000	12.402	12
2000	12.462	12
3000	12.457	12
4000	12.447	12
5195 (max)	12.453	12

表 6.1 より、学習したデータ数が変化しても、糖鎖構造中の平均ノード数は 12 付近になることがわかる。また、中央値もデータ数が 100 の場合を除いて 12 個で一致している。したがって、データ数の違いが糖鎖構造のノード数に与える影響は低いと考える。

また、本研究の糖鎖構造は *OTMM* の木構造と同じルールで作成される。参考のため、*OTMM* の木構造を図 6.1 に再掲する。

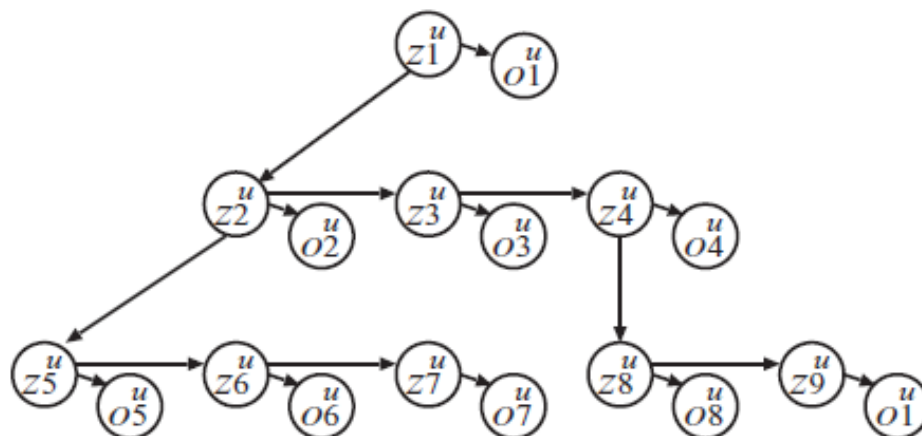


図 6.1 *OTMM* の例 (再掲)[16]

このように、*OTMM* では、親子間、兄弟間の繋がりが 1 対 1 になっている。特に、親ノードと長男ノードのみが親子関係になっているのが特徴である。

さらに，図 6.2 に糖鎖構造中の単糖の記号を示す．

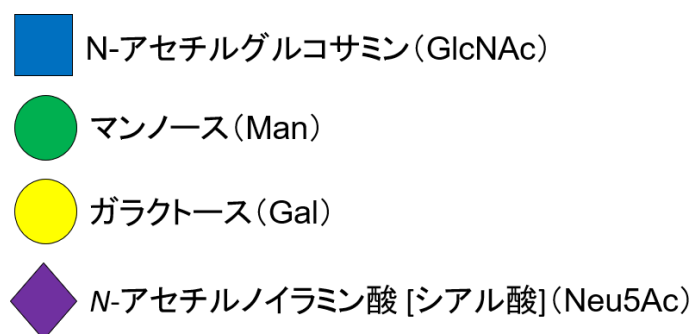


図 6.2 単糖の記号 (再掲)

以降の実験結果では，この単糖の表記に従うこととする．また，糖鎖構造を表すグラフの辺上に記述されている文字は結合様式を表す．

6.2 実験結果 I

まず，データ数が 5195，状態数が 6，しきい値が 1.0 で実験を行った．

6.2.1 単糖のみを学習した場合

単糖のみを学習した場合，ラベル数は 59 であった．まず，学習済みのモデルで *N*-結合型糖鎖を解析 (*Parsing*) した結果を示す．これによって，学習結果 (状態遷移図) が解析に反映されるので，「モデルは何を学習したのか」を間接的に理解することができる．

図 6.3 に *High mannose* 型の解析結果を示す．

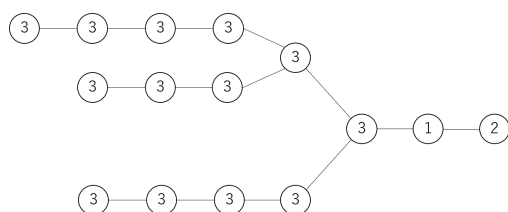


図 6.3 Parsing した結果
(Most likely state path)

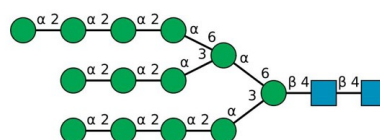


図 6.4 High mannose 型糖鎖

Man が多数存在する葉付近には，状態 3 が割り振られていることがわかる．このようなことから，同じ単糖 (*Man*) を認識できていることがわかる．また，コア 5 糖については，他のタイプの糖鎖と状態が一致していることもわかる．図 6.4 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は，対数表示で -8.826 となった．

図 6.5 に *Complex* 型の解析結果を示す.

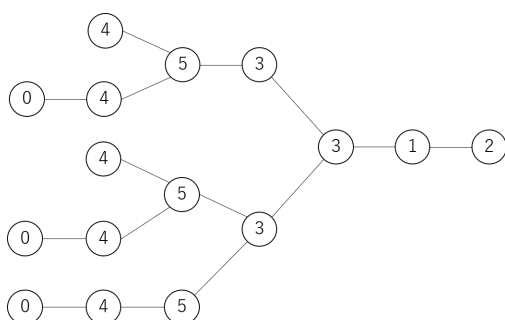


図 6.5 Parsing した結果
(Most likely state path)

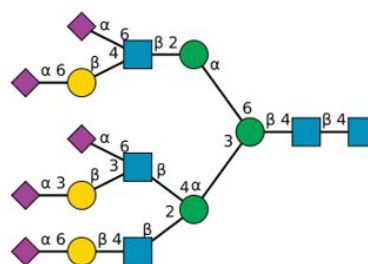


図 6.6 *Complex* 型糖鎖

葉の部分において、「状態 5 → 状態 4 → 状態 0」という *GlcNAc* → *Gal* → *Neu5Ac* に相当する状態が割り振られていることがわかる。また、コア 5 糖については、他のタイプの糖鎖と状態が一致していることもわかる。図 6.6 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -16.143 となった。

図 6.7 に *Hybrid* 型の解析結果を示す。

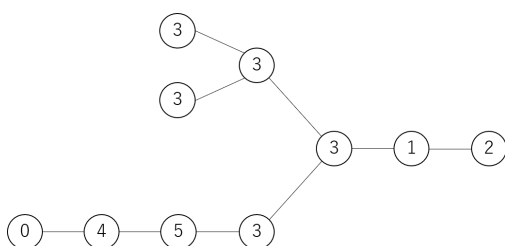


図 6.7 Parsing した結果
(Most likely state path)

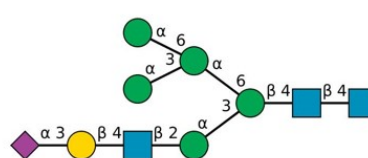


図 6.8 *Hybrid* 型糖鎖

High mannose 型の特徴である「状態 3 が葉付近で連続する部分」と *Complex* 型の特徴である「状態 5 → 状態 4 → 状態 0」という遷移を併せ持っている。これは、*Hybrid* 型の特徴と一致する。また、コア 5 糖については、他のタイプの糖鎖と状態が一致していることもわかる。図 6.8 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -3.934 となった。

次に、図 6.9 に学習結果 (状態遷移図) を示す。この状態遷移図は、前述の 3 種類の確率分布 (初期状態確率分布, 状態遷移確率分布, ラベル出力確率分布) を基にして作成された。青色の実線が親子関係を、橙色の破線が兄弟関係を表す。また、辺上の数値が遷移確率を、ラベル (単糖) 横の数値がラベルの出力確率を表す。さらに、初期確率と遷移確率は基本的に 0.3 未満を省略した。加えて、ラベル出力確率は基本的に 0.1 未満を省略した。

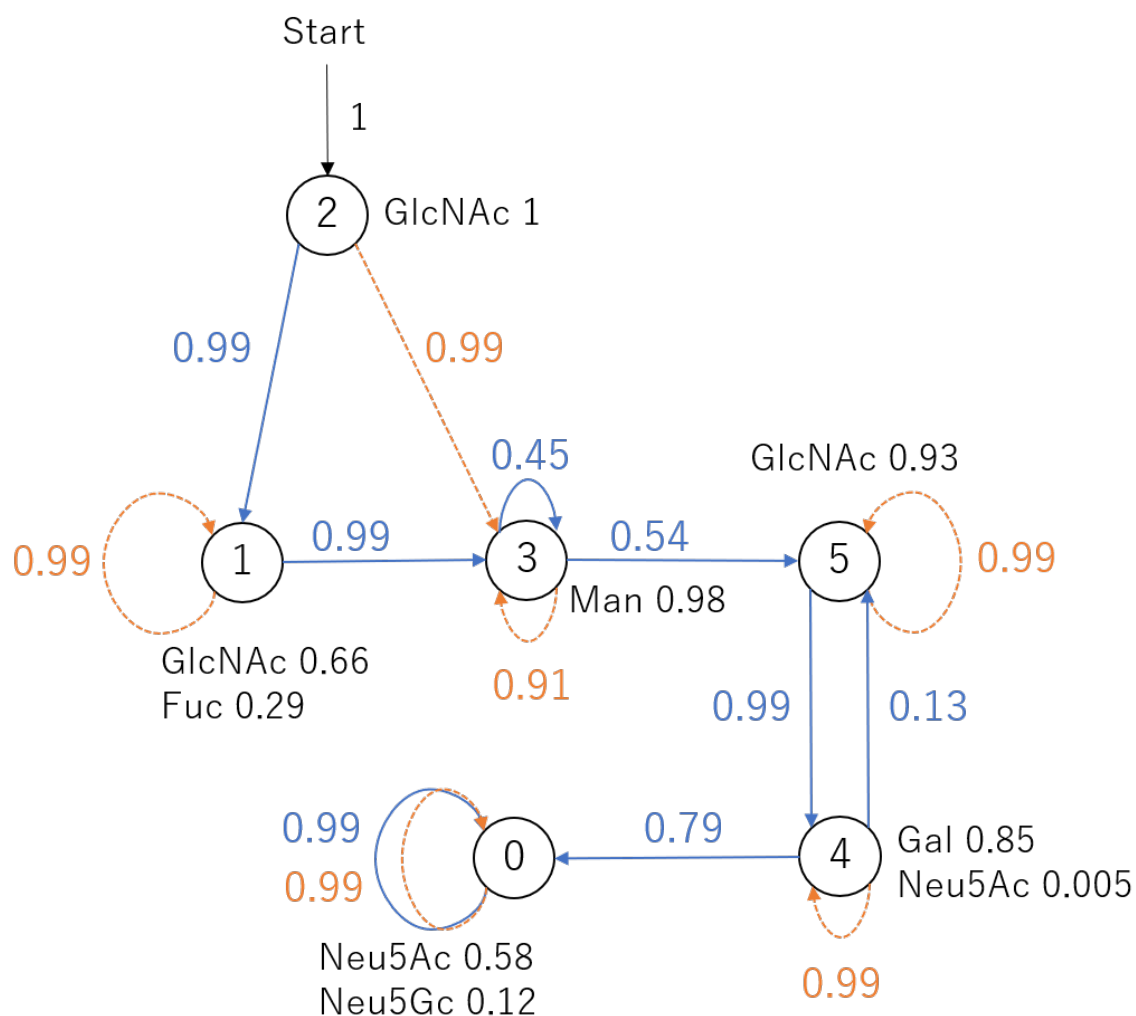


図 6.9 単糖のみを学習した時の状態遷移図

「状態 2 → 状態 1 → 状態 3」と遷移した後に、状態 3 を繰り返す場合は *High mannose* 型、状態 5 に遷移する場合は *Complex* 型になると考えられる。また、根に相当する初期状態 2 のラベルには *GlcNAc* が現れ、*N*-結合型糖鎖の構造の特徴を適切に表している。

6.2.2 単糖 + 結合様式を学習した場合

単糖と結合様式を組み合わせ学習した場合、ラベル数は 159 であった。まず、学習済みのモデルで *N*-結合型糖鎖を解析 (*Parsing*) した結果を示す。

図 6.10 に *High mannose* 型の解析結果を示す。

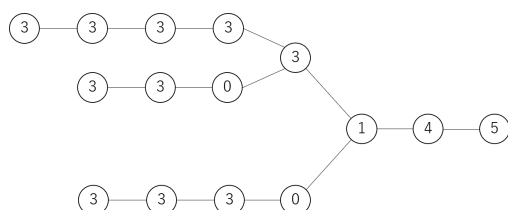


図 6.10 Parsing した結果
(Most likely state path)

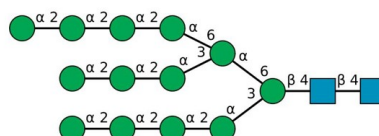


図 6.11 High mannose 型糖鎖

Man が多数存在する葉付近には、状態 3 が割り振られていることがわかる。また、*Man*(a1-3) には状態 0 が割り振られており、 α 1-3 結合を持つ *Man* を区別できていることがわかる。すなわち、結合様式の違いを捉えて、構造の特徴を学習できている。図 6.11 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -36.270 であり、単糖のみのデータより低い確率となった。

図 6.12 に *Complex* 型の解析結果を示す。

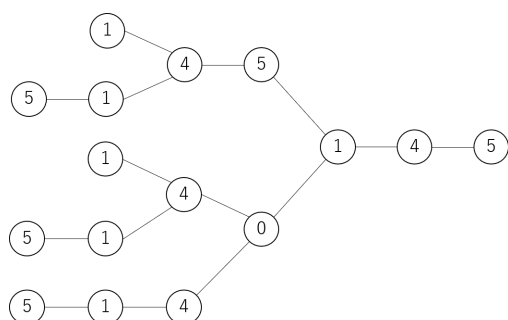


図 6.12 Parsing した結果
(Most likely state path)

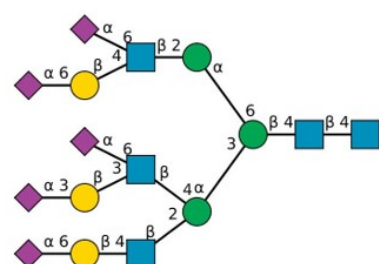


図 6.13 Complex 型糖鎖

葉の付近において、「状態 4 → 状態 1 → 状態 5」という *GlcNAc* → *Gal* → *Neu5Ac* に相当する状態が割り振られていることがわかる。また、コア 5 糖は *High mannose* 型と異なっている。全体的な結果としては、単糖のみの解析結果と大きく変わらない。図 6.13 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -39.364

であり、単糖のみのデータより低い確率となった。

図 6.14 に *Hybrid* 型の解析結果を示す.

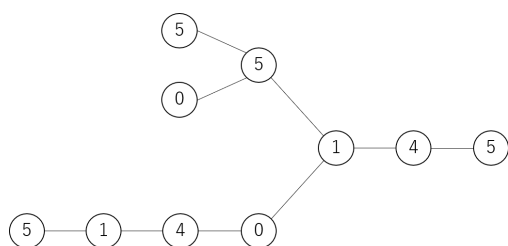


図 6.14 Parsing した結果
(Most likely state path)

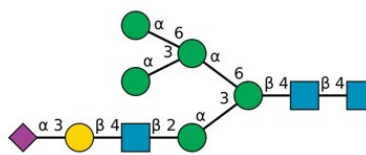


図 6.15 Hybrid 型糖鎖

Complex 型の特徴である「状態 4 → 状態 1 → 状態 5」という遷移を持っているが、*High mannose* 型の特徴である状態 0 と状態 3 は見られない。これは、*Hybrid* 型の特徴と一致しない。また、コア 5 糖は *Complex* 型と一致している。したがって、状態遷移は *Complex* 型と同じ特徴を持つことが考えられる。図 6.15 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -21.434 であり、単糖のみのデータより低い確率となった。

次に、図 6.16 に学習結果 (状態遷移図) を示す。この状態遷移図は、前述の 3 種類の確率分布を基にして作成された。青色の実線が親子関係を、橙色の破線が兄弟関係を表す。また、辺上の数値が遷移確率を、ラベル (単糖 + 結合様式) 横の数値がラベルの出力確率を表す。

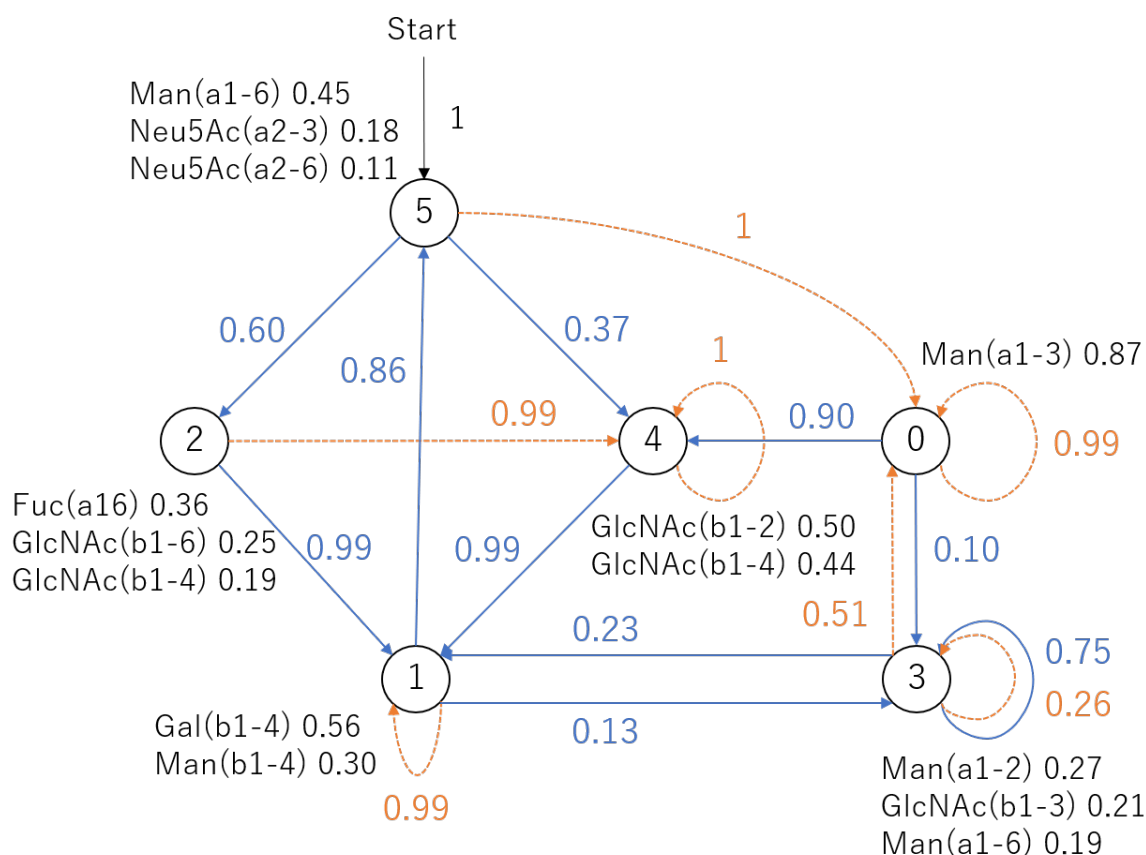


図 6.16 単糖 + 結合様式を学習した時の状態遷移図

「状態 5 → 状態 4 → 状態 1」と遷移した後に、状態 3 や状態 0 を繰り返す場合は *High mannose* 型、状態 5 に戻る場合は *Complex* 型になると考えられる。また、根に相当する初期状態 5 では、*Man(a1-6)* などの *GlcNAc* 以外のラベルが高い確率で出力されることがわかる。これは、必ず根に *GlcNAc* が存在する *N*-結合型糖鎖の特徴と一致しない。さらに、単糖のみの状態遷移図と比べて、1 つの状態が出力するラベルの種類が多くなり、その出力確率も分散している。これは、1 つの状態が複数のラベル (単糖 + 結合様式) を出力することを意味する。

6.2.3 まとめ I

「単糖のみ」のデータを用いた場合は、おおむね適切に糖鎖構造中の単糖の区別ができしており、*N*-結合型糖鎖の 3 つのタイプの特徴も学習できていた。したがって、*N*-結合型糖鎖の構造に共通するパターンを適切に学習できていたとわかる。

「単糖 + 結合様式」のデータを用いた場合は、*High Mannose* 型で結合様式を考慮した糖鎖構造の学習ができていた。しかしながら、「単糖のみ」のデータと比べてラベル数が非常に多くなるため、1 つの状態が複数の種類のラベルを出力することになった。この影響により、*Hybrid* 型では適切に糖鎖構造の特徴を表現できなかった。また、解析した糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は「単糖のみ」のデータと比べて低くなった。これは、「単糖 + 結合様式」のデータを用いた学習は、「単糖のみ」のデータを用いた学習よりも結果が曖昧になることを意味する。ただし、この確率はあくまで評価指標の 1 つであり、実際の解析結果を生物学的な視点から定量的に評価することも非常に重要な指標となる。特に「単糖 + 結合様式」のデータを用いた学習では、元々のラベル数が「単糖のみ」のデータに比べて大幅に多くなるため、この確率が低くなる傾向があることに注意が必要である。以上のようなことから、「単糖 + 結合様式」のデータを学習する場合は、状態数を増やす必要があるとわかった。

6.3 実験結果 II

実験結果 I を受けて、「単糖 + 結合様式」のデータを学習する際は状態数を増やした。具体的には、「単糖のみ」のデータを用いる時は状態数を 6 にし、「単糖 + 結合様式」のデータを用いるときは状態数をその 2.5 倍の 15 にした。その理由としては、「単糖のみ」のデータを学習する場合はラベルの数が 40 になり、「単糖 + 結合様式」のデータを学習する場合はラベルの数が 102 になることが挙げられる。すなわち、「単糖 + 結合様式」のデータは「単糖のみ」のデータの約 2.5 倍のラベルを持っているので、状態数もそれに合わせて 2.5 倍にした。また、前述の表 4.1 より、状態数を増やすと 2 乗のオーダーで時間計算量が増加する。したがって、処理時間を抑えるため、データ数を 1000 に減らす。しきい値はこれまで通り 1.0 とする。

6.3.1 単糖のみを学習した場合 (状態数 6)

単糖のみ学習した場合は、ラベルの数が 40 個であった。まず、学習済みのモデルで *N*-結合型糖鎖を解析 (*Parsing*) した結果を示す。

図 6.17 に *High mannose* 型の解析結果を示す。

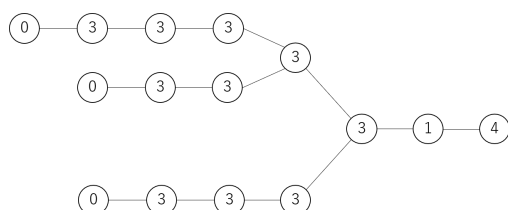


図 6.17 Parsing した結果
(Most likely state path)

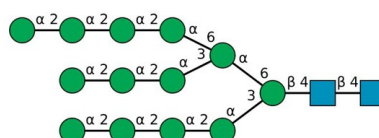


図 6.18 High mannose 型糖鎖

Man が多数存在する葉付近には、状態 3 が割り振られている。また、葉の *Man* のみ状態 0 が割り振られており、連続する *Man* の最後の部分を認識できていることがわかる。さらに、根から 3 つ目までのノードについては、他のタイプの糖鎖と状態が一致している。図 6.18 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -20.665 であった。

図 6.19 に *Complex* 型の解析結果を示す。

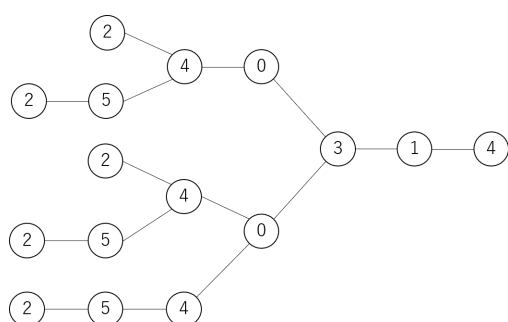


図 6.19 Parsing した結果
(Most likely state path)

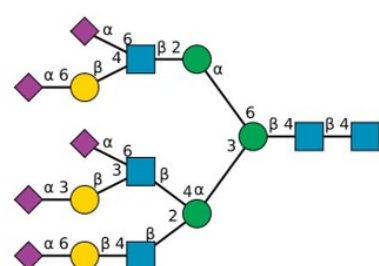


図 6.20 Complex 型糖鎖

葉の付近において、「状態 4 → 状態 5 → 状態 2」という *GlcNAc* → *Gal* → *Neu5Ac* に相当する状態が割り振られていることがわかる。また、根から 3 つ目までのノードについては、他のタイプの糖鎖と状態が一致している。図 6.20 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -16.143 となった。

図 6.21 に *Hybrid* 型の解析結果を示す.

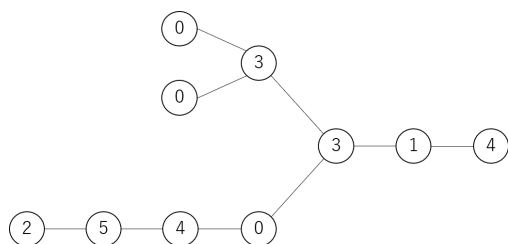


図 6.21 Parsing した結果
(Most likely state path)

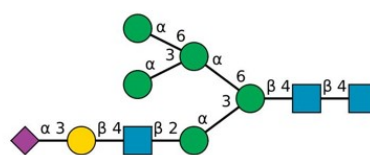


図 6.22 *Hybrid* 型糖鎖

上部の分岐では葉付近で状態 3 と状態 0 が見られ、下部の分岐では「状態 4 → 状態 5 → 状態 2」という遷移が見られる。したがって、この解析結果は *High mannose* 型と *Complex* 型の特長を併せ持っているといえる。これは、*Hybrid* 型の特徴と一致している。また、図 6.22 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -6.291 となった。

次に、図 6.23 に学習結果 (状態遷移図) を示す。この状態遷移図は、前述の 3 種類の確率分布を基にして作成された。青色の実線が親子関係を、橙色の破線が兄弟関係を表す。また、辺上の数値が遷移確率を、ラベル (単糖) 横の数値がラベルの出力確率を表す。

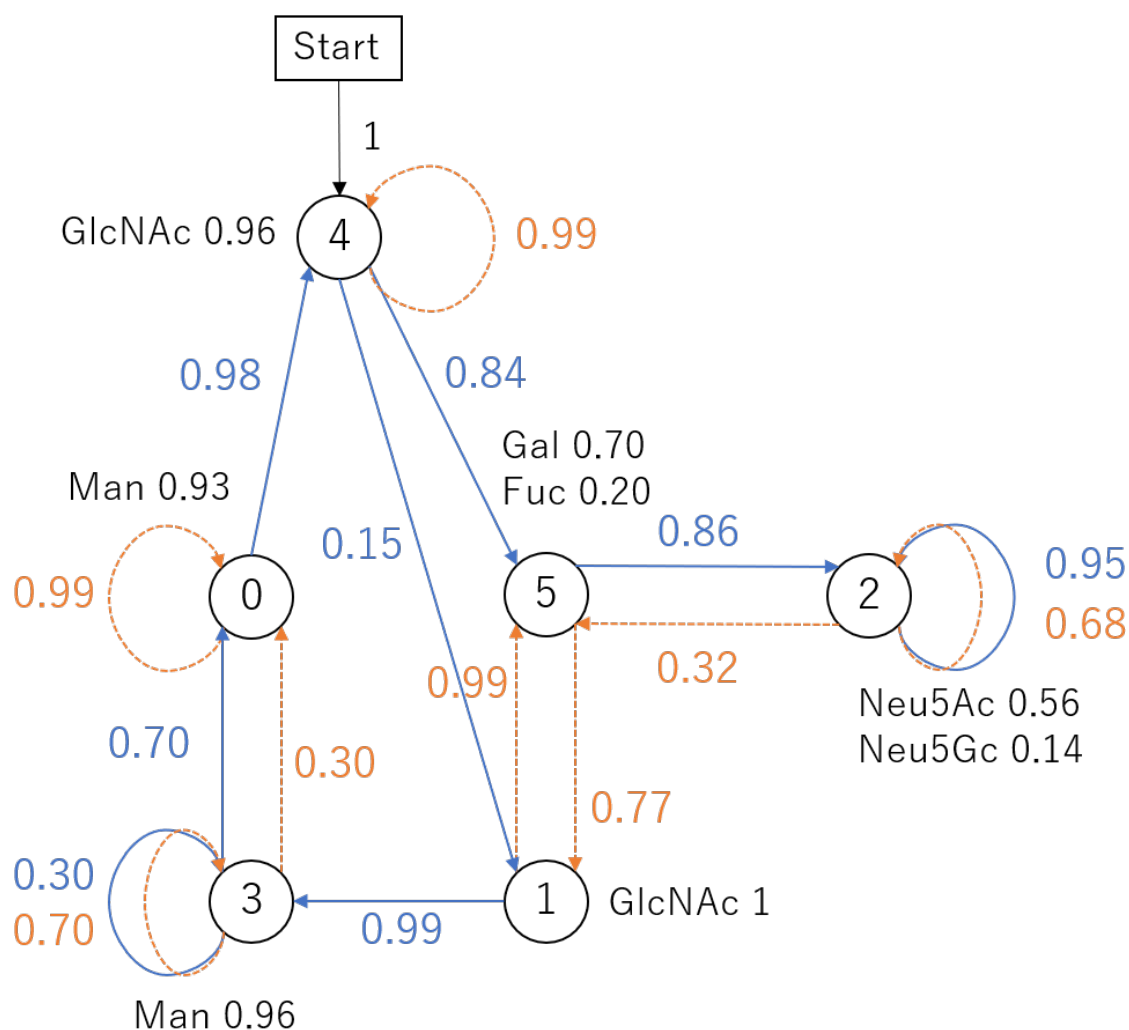


図 6.23 単糖のみを学習した時の状態遷移図 (状態数 6)

「状態 4 → 状態 1 → 状態 3」と遷移した後に、状態 3 を繰り返す場合は *High mannose* 型、状態 0 に遷移する場合は *Complex* 型になると考えられる。また、初期状態 (根) の 4 のラベルには *GlcNAc* が現れ、*N*-結合型糖鎖の構造の特徴を適切に表していることがわかる。

6.3.2 単糖 + 結合様式を学習した場合 (状態数 6)

単糖と結合様式を組み合わせ学習した場合は、ラベルの数が 102 個であった。まず、学習済みのモデルで *N*-結合型糖鎖を解析 (*Parsing*) した結果を示す。

図 6.24 に *Higmannose* 型の解析結果を示す。

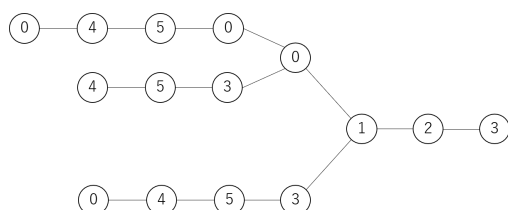


図 6.24 Parsing した結果
(Most likely state path)

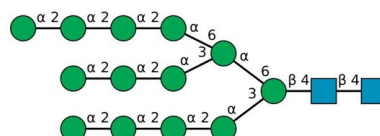


図 6.25 High mannose 型糖鎖

同じ「単糖 + 結合様式」のラベルであっても異なる状態が割り振られている。具体的には、葉付近の *Man*($\alpha 1-2$) が多数存在する位置には、状態 0, 状態 4, 状態 5 が割り振られている。特に、下部の分岐における「状態 3 → 状態 5 → 状態 4 → 状態 0」という遷移は *Complex* 型にも見られる遷移になっている。また、*Man*($\alpha 1-3$) には状態 3 が割り振られており、 $\alpha 1-3$ 結合を持つ *Man* を区別できていることがわかる。さらに、コア 5 糖が他のタイプの糖鎖の解析結果と一致していることもわかる。図 6.25 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -44.884 であり、状態数が 6 の時の単糖のみのデータより低い確率となった。

図 6.26 に *Complex* 型の解析結果を示す。

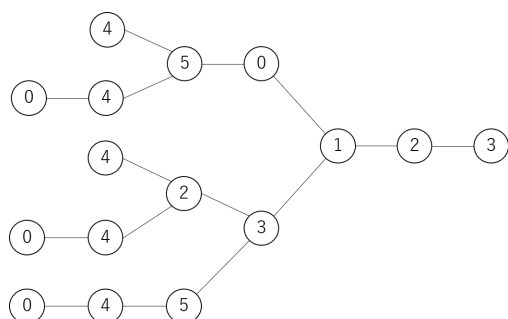


図 6.26 Parsing した結果
(Most likely state path)

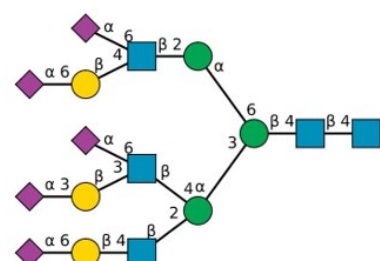


図 6.27 Complex 型糖鎖

根を除くと, $GlcNAc(b1-4)$ には状態 2 が, $GlcNAc(b1-2)$ には状態 5 が割り振られていることがわかる. すなわち, 結合様式の違いを捉えて, 糖鎖構造を表現できている. また, コア 5 糖は他のタイプの糖鎖の解析結果と一致していることもわかる. 図 6.27 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は, 対数表示で-35.457 であり, 状態数が 6 の時の単糖のみのデータより低い確率となった.

図 6.28 に *Hybrid* 型の解析結果を示す.

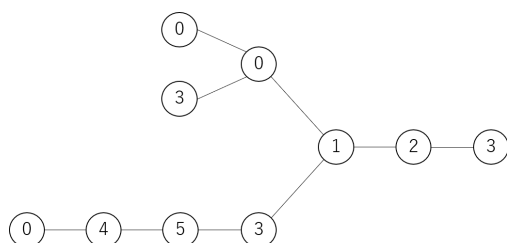


図 6.28 Parsing した結果
(Most likely state path)

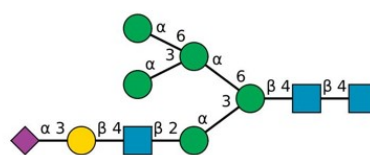


図 6.29 Hybrid 型糖鎖

上部の分岐では, *High mannose* 型の特徴である状態 0 と状態 3 が割り振られ, 下部の分岐では, *Complex* 型の特徴である「状態 3 → 状態 5 → 状態 4 → 状態 0」という遷移を持っている. これは, *Hybrid* 型の特徴と一致する. また, コア 5 糖は他のタイプの糖鎖の解析結果と一致していることもわかる. 図 6.29 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は, 対数表示で-16.001 であり, 状態数が 6 の時の単糖のみのデータより低い確率となった.

次に、図 6.30 に学習結果 (状態遷移図) を示す。この状態遷移図は、前述の 3 種類の確率分布を基にして作成された。青色の実線が親子関係を、橙色の破線が兄弟関係を表す。また、辺上の数値が遷移確率を、ラベル横の数値がラベルの出力確率を表す。

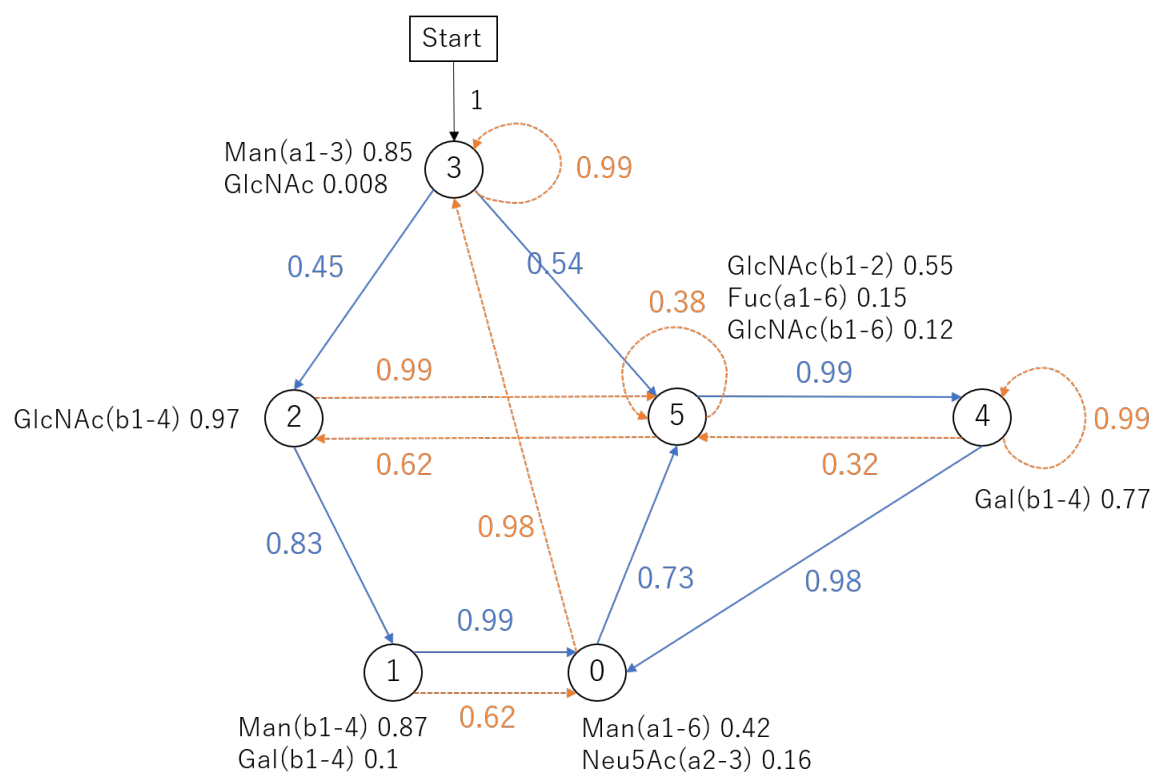


図 6.30 単糖 + 結合様式を学習した時の状態遷移図 (状態数 6)

「状態 3 → 状態 2 → 状態 1」と遷移した後に、状態 0 や状態 3 を繰り返す場合は *High mannose* 型、状態 5 や状態 4 に遷移する場合は *Complex* 型になると考えられる。また、データ数が 5195 個の時と同様に、1 つの状態が出力するラベルの種類が多くなっており、その出力確率も分散している。これは、1 つの状態が複数のラベル (単糖 + 結合様式) を出力することを意味する。このようなことから、*High Mannose* 型の下部の分岐の状態と *Complex* 型の下の分岐の状態が同じになったと考えられる。解析結果より、全く異なる 2 つのタイプの糖鎖で同じ状態列が見られるため、適切に糖鎖構造の特徴を学習できたとはいえない。

6.3.3 単糖のみを学習した場合 (状態数 15)

単糖のみ学習した場合は、ラベルの数が 40 個であった。まず、学習済みのモデルで *N*-結合型糖鎖を解析 (*Parsing*) した結果を示す。

図 6.31 に *High mannose* 型の解析結果を示す。

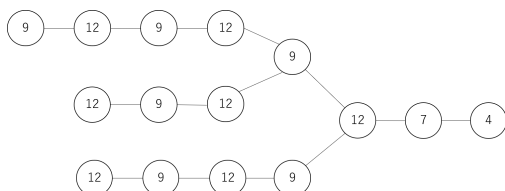


図 6.31 Parsing した結果
(Most likely state path)

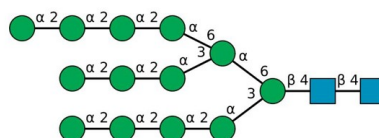


図 6.32 High mannose 型糖鎖

葉付近の *Man* が多数存在する位置には、状態 9 と状態 12 が交互に割り振られている。図 6.37 の状態遷移図を見ると、*Man* の親子関係では、状態 9 と状態 12 の間で交互に遷移をすることがわかる。また、コア 5 糖は他のタイプの糖鎖の解析結果と一致していることもわかる。図 6.32 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -12.137 となった。

図 6.33 に *Complex* 型の解析結果を示す。

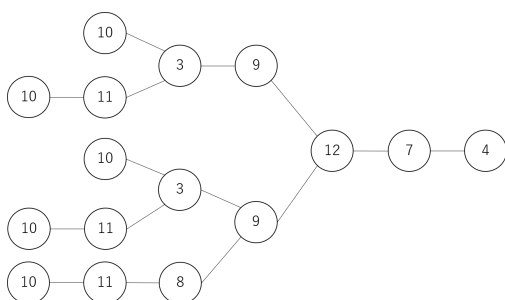


図 6.33 Parsing した結果
(Most likely state path)

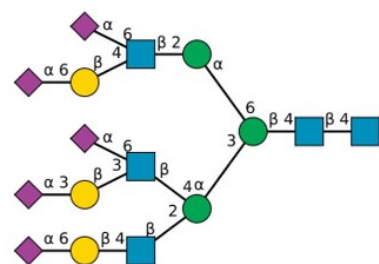


図 6.34 Complex 型糖鎖

葉の付近において、「状態 3 → 状態 11 → 状態 10」という *GlcNAc* → *Gal* → *Neu5Ac* に相当する状態が割り振られていることがわかる。また、*GlcNAc* には、状態 3, 状態 4, 状態 7, 状態 8 の 4 つの状態が割り振られていることがわかる。さらに、コア 5 糖は他のタイプの糖鎖の解析結果と一致していることもわかる。図 6.34 の糖鎖構造のラベルが *Most*

likely state path に沿って出力される確率は、対数表示で-18.702 となった.

図 6.35 に *Hybrid* 型の解析結果を示す.

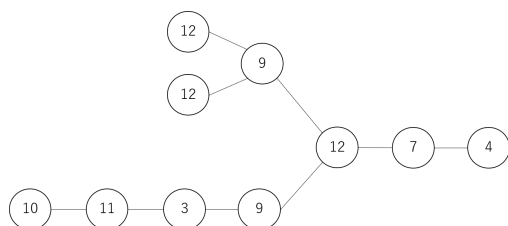


図 6.35 Parsing した結果
(Most likely state path)

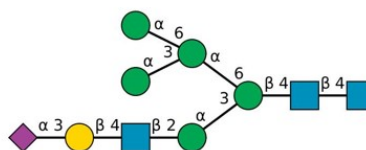


図 6.36 Hybrid 型糖鎖

上部の分岐には *High mannose* 型で見られる状態 9 と状態 12 があり，下部の分岐には *Complex* 型に見られる「状態 3 → 状態 11 → 状態 10」という遷移を併せ持っている．これは，*Hybrid* 型の特徴と一致している．また，コア 5 糖は他のタイプの糖鎖の解析結果と一致していることもわかる．図 6.36 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は，対数表示で -4.786 となった．

次に、図 6.37 に学習結果 (状態遷移図) を示す。この状態遷移図は、前述の 3 種類の確率分布を基にして作成された。青色の実線が親子関係を、橙色の破線が兄弟関係を表す。また、辺上の数値が遷移確率を、ラベル (単糖) 横の数値がラベルの出力確率を表す。

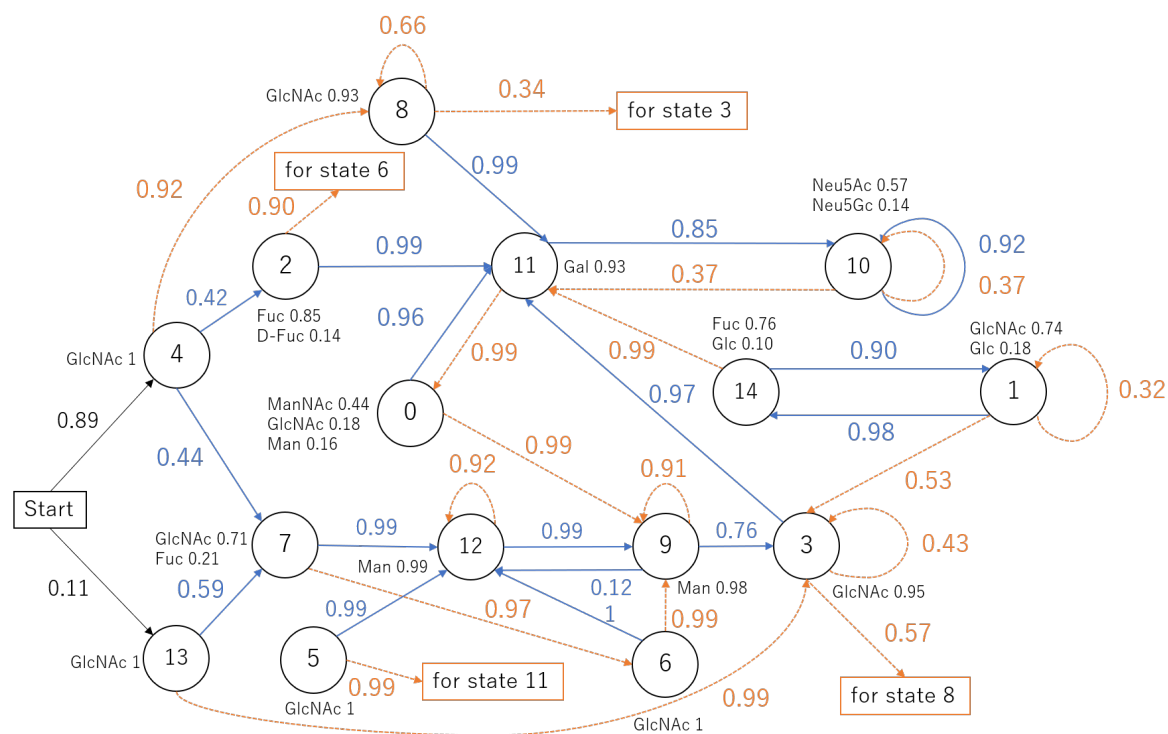


図 6.37 単糖のみを学習した時の状態遷移図 (状態数 15)

「状態 4 → 状態 7 → 状態 12 → 状態 9 → 状態 9」がコア 5 糖の状態遷移であると考えられる。その後、状態 12 や状態 9 を繰り返すのが *High mannose* 型であり、「状態 3 → 状態 11 → 状態 10」と遷移するのが *Complex* 型であると考えられる。また、状態 1, 状態 3, 状態 4, 状態 5, 状態 6, 状態 7, 状態 8, 状態 13 では *GlcNAc* が、状態 9 と状態 12 では *Man* が出力しやすくなっている。言い換えれば、状態数が多すぎて各状態に同じ単糖が出現している。このようなことから、同じ単糖ラベルでも様々な状態が割り振られてしまい、糖鎖構造の特徴を適切に表すことが困難になっている。

6.3.4 単糖 + 結合様式を学習した場合 (状態数 15)

単糖と結合様式を組み合わせ学習した場合は、ラベルの数が 102 個であった。まず、学習済みのモデルで *N*-結合型糖鎖を解析 (*Parsing*) した結果を示す。

図 6.38 に *Higmannose* 型の解析結果を示す。

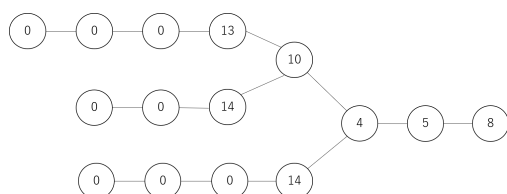


図 6.38 Parsing した結果
(Most likely state path)

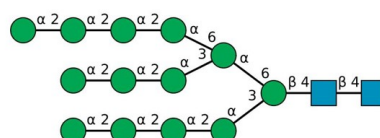


図 6.39 High mannose 型糖鎖

葉付近の $Man(a1-2)$ が多数存在する位置には、状態 0 が割り振られている。また、 $Man(a1-3)$ には状態 14 が、 $Man(a1-6)$ には状態 10 と状態 13 が割り振られている。すなわち、*Man* の結合様式の違いをほぼ完璧に捉えており、構造の特徴を正確に表現できている。さらに、コア 5 糖は他のタイプの糖鎖の解析結果と一致していることもわかる。図 6.39 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -11.704 であり、状態数が 15 の時の単糖のみのデータより高い確率となった。すなわち、この解析結果は、状態数が 15 の時の単糖のみのデータの解析結果より曖昧性が低いといえる。

図 6.40 に *Complex* 型の解析結果を示す。

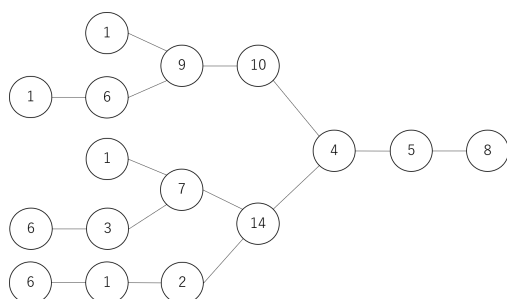


図 6.40 Parsing した結果
(Most likely state path)

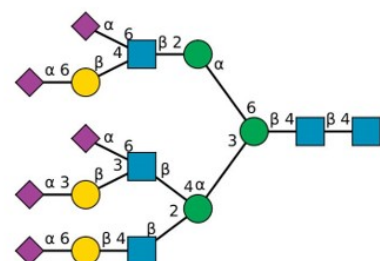


図 6.41 Complex 型糖鎖

一見、様々な状態が割り振られ、法則性が無いように見えるが、葉の付近に注目するとかなり詳細な構造を捉えていることがわかる。図 6.42 に、学習でとらえた詳細な構造を示す。

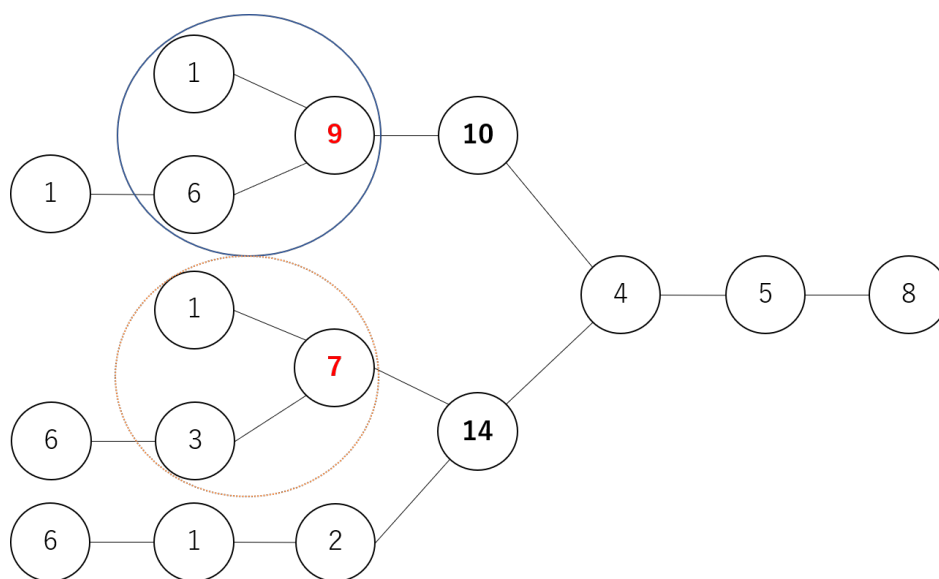


図 6.42 単糖 + 結合様式の学習で捉えられる詳細な構造

図 6.45 の状態遷移図より、状態 1 は 0.99 の確率で（兄弟ノードの）状態 3 に遷移することがわかる。つまり、通常は橙色の破線の丸のように状態 1 から状態 3 へと遷移するはずである。一方で、非常に低い確率で青い実線の丸のように状態 1 から状態 6 へと遷移することがある。これは、「親の状態」が違うために生じる違いであると考えられる。具体的には、橙色の破線の丸では親の状態が 9 であるのに対し、青い実線の丸では親の状態が 7 となる（赤色の太字）。すなわち、同じ単糖 (*GlcNAc*) であっても結合様式 (*b1-2* 結合と *b1-4* 結合) が違うため、異なる状態が割り振られるのである。加えて、状態 9 と状態 7 の違いは、その親である状態 10 と状態 14 の違いに起因する（黒い太字）。この違いも、結合様式の違いによるものである。このような違いによって、一見似ている構造であっても違いが生じる。したがって、単糖 + 結合様式の学習によって、「注目しているノードの周りの結合様式の状況」まで考慮できるようになったと考えられる。すなわち、詳細な構造の違いまで捉えられるようになったと考えられる。また、*Gal(b1-4)* には状態 6 が、*Gal(b1-3)* には状態 3 が割り振られ、*Gal* の結合様式の違いを捉えていることがわかる。さらに、コア 5 糖は他のタイプの糖鎖の解析結果と一致していることがわかる。図 6.41 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は、対数表示で -54.719 であり、状態数が 15 の時の単糖のみのデータより低い確率となった。

図 6.43 に *Hybrid* 型の解析結果を示す.

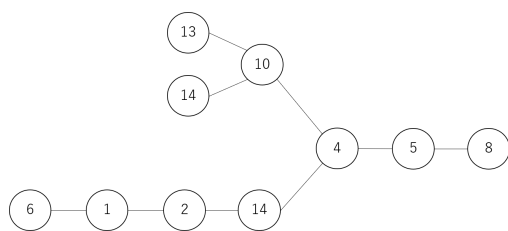


図 6.43 Parsing した結果
(Most likely state path)

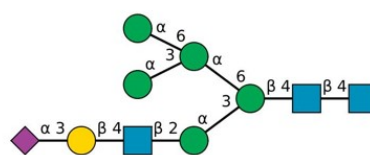


図 6.44 *Hybrid* 型糖鎖

分岐の上部において, *High mannose* 型の特徴である「親に状態 10, 子に状態 13 と状態 14」という遷移を持っている. 加えて, 分岐の下部において, *Complex* 型で見られる「状態 2 → 状態 1 → 状態 6」という遷移も持っている. これは, *Hybrid* 型の特徴と一致する. *High mannose* 型と *Complex* 型で見られる状態遷移を持っていることから, *Hybrid* 型でも結合様式を捉えて, 糖鎖構造を詳細に表現できていることがわかる. また, コア 5 糖は他のタイプの糖鎖の解析結果と一致している. 図 6.44 の糖鎖構造のラベルが *Most likely state path* に沿って出力される確率は, 対数表示で -6.578 であり, 状態数が 15 の時の単糖のみのデータより低い確率となった.

次に、図 6.45 に学習結果 (状態遷移図) を示す。この状態遷移図は、前述の 3 種類の確率分布を基にして作成された。青色の実線が親子関係を、橙色の破線が兄弟関係を表す。また、辺上の数値が遷移確率を、ラベル横の数値がラベルの出力確率を示す。

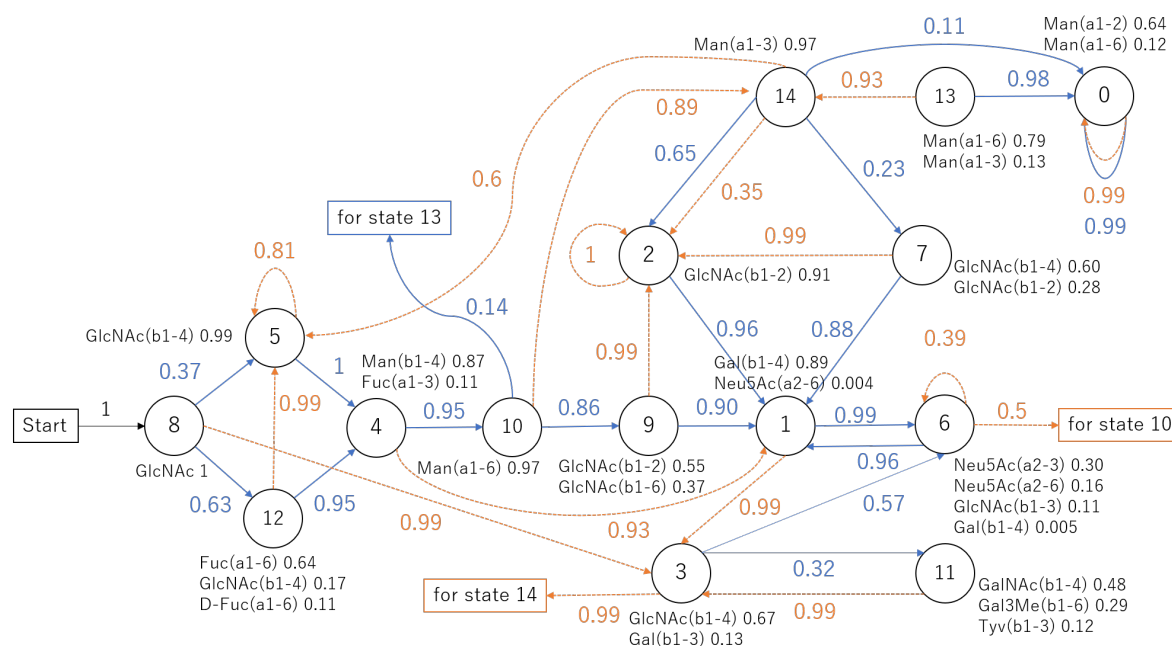


図 6.45 単糖 + 結合様式を学習した時の状態遷移図 (状態数 15)

「状態 8 → 状態 4 → 状態 5 → 状態 10 → 状態 14」という遷移がコア 5 糖の遷移である。その後、右上付近の状態 14, 状態 13, 状態 0 を繰り返す場合は *High mannose* 型となる。また、中心付近の状態 7, 状態 1, 状態 9, 状態 6 を繰り返す場合は *Complex* 型になると考えられる。さらに、*Complex* 型の特徴として、「*Neu5Ac* をラベルとして持つ状態 6 にたどり着くためには、*Gal* をラベルとして持つ状態 1 か状態 3 を経由する必要があること」が挙げられる。加えて、*Man(a1-6)* をラベルとして持つ状態 10 の後に、*Man(a1-3)* をラベルとして持つ状態 14 のが兄弟ノードとして登場するのは生物学的にも妥当である。そして、状態数を増やしたため、1 つの状態が持つラベルの種類が少なくなった。これは、1 つの状態が特定の種類のラベル (単糖 + 結合様式) を出力できることを意味する。具体的には、状態 10 は *Man(a1-6)* を、状態 14 は *Man(a1-3)* を非常に高い確率で出力できることなどが挙げられる。このようなことによって、結合様式まで考慮した詳細な構造を捉えることが可能になったと考えられる。

6.3.5 まとめ II

「単糖のみ」のデータを用いた場合、状態数が 6 では正確に糖鎖構造中の単糖の区別ができた。しかしながら、状態数が 15 が増えると、同じ単糖に複数の状態が割り振られて、糖鎖構造の特徴を正確に表現することが困難になった。したがって、「単糖のみ」のデータは、状態数が多すぎると糖鎖構造を適切に学習できないことがわかった。

「単糖 + 結合様式」のデータを用いた場合、状態数が 6 ではラベルに対して状態数が少なすぎるため、正確に糖鎖構造の区別ができなかった。一方で、状態数が 15 が増えると、結合様式まで考慮した糖鎖構造を正確に学習することができた。

したがって、状態数を少なくした場合 (状態数が 6 など) は、「単糖のみ」のデータを用いた方が正確に糖鎖構造の特徴を捉えることができ、状態数を多くした場合 (状態数が 15 など) は、「単糖 + 結合様式」のデータを用いた方が正確に糖鎖構造の特徴を捉えることができた。特に、「単糖 + 結合様式」のデータを用いることで、結合様式まで考慮した「より正確な」糖鎖構造の解析が可能となった。

第 7 章

考察

7.1 正確性に関する考察

実験結果から、「単糖 + 結合様式」のデータを用いた学習によって、結合様式まで考慮した微細な構造の違いを捉えることができた。また、このような細かい構造の違いは、「単糖のみ」の学習では捉えることができなかった。したがって、「単糖 + 結合様式」のデータを用いた学習によって、より正確に糖鎖構造の特徴 (共通パターン) を得ることができるとわかった。

ただし、「単糖 + 結合様式」のデータを用いても、*Complex* 型や *Hybrid* 型などの一部の糖鎖では、その構造の特徴を完璧に捉えきけることは難しい。したがって、今後は学習データの質やしきい値を調整するなどして、より正確に糖鎖構造の特徴が得られるような条件を探る必要がある。

7.2 効率性に関する考察

コンピューターサイエンスにおいて、アルゴリズムの効率性を考えることは非常に重要である。

まず、「単糖のみ」のデータと「単糖 + 結合様式」のデータについて、学習に要する時間の比較を行う。ここで、「入力データを何回繰り返して学習したか」を表す指標を「エポック数」と呼ぶ。例えば、入力データを 5 回繰り返して学習した場合、エポック数は 5 となる。また、使用したマシン (環境) は *Mac Pro*(2019) であり、スペックは表 7.1 の通りである。

表 7.1 Mac Pro(2019) のスペック

	Spec
OS	macOS Monterey バージョン 12.6
プロセッサ	2.7GHz 24 コア Intel Xeon W
メモリ	48GB 2933MHz DDR4
グラフィックス	AMD Radeon Pro W5500X 8GB

図 7.1 に 1 エポック当たりの処理時間の比較を示す.

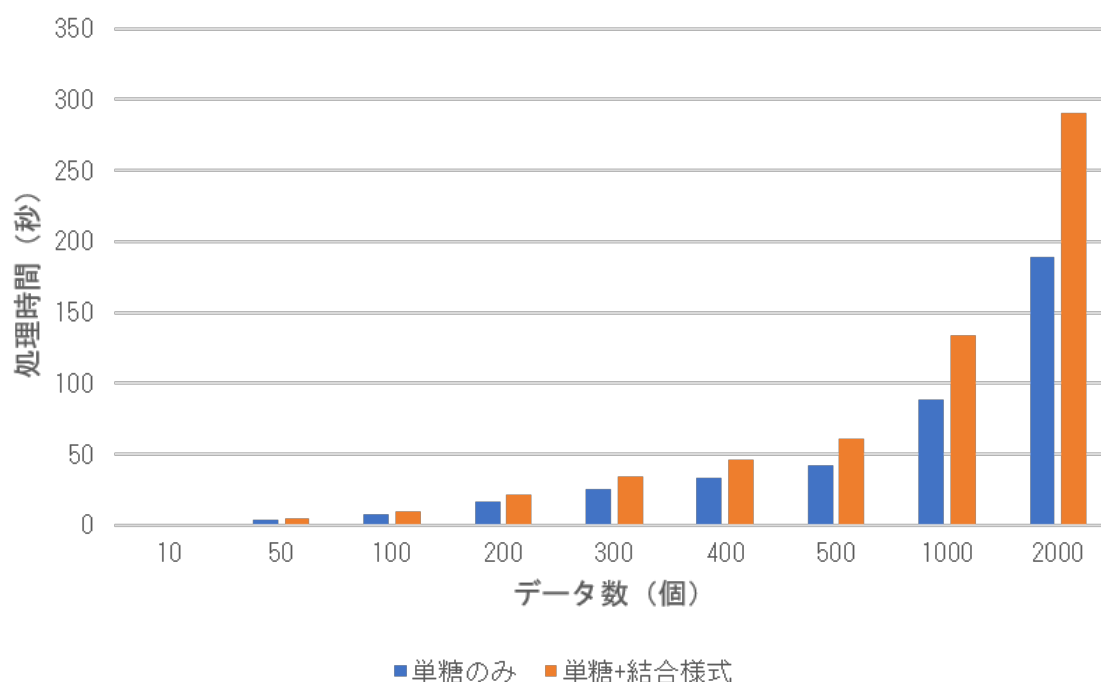


図 7.1 1 エポック当たりの処理時間の比較

「単糖のみ」の糖鎖構造データを用いた場合と比べて、「単糖 + 結合様式」の糖鎖構造データを用いた場合は処理時間が長くなった。これは、「単糖 + 結合様式」の糖鎖構造データは、必ず「単糖のみ」の糖鎖構造データ以上のラベル数を持つためであると考えられる。つまり、「単糖 + 結合様式」のデータは「単糖のみ」のデータよりも学習するデータの量が多くなるので、必然的に処理時間が長くなったと考えられる。また、データ数が多くなるほど、処理時間の差が大きくなることがわかる。これは、データ数が増加すると、「単糖 + 結合様式」の組み合わせ (ラベルの種類) が大幅に増加するためであると考えられる。逆に、「単糖のみ」の場合は、データ数が増加しても「単糖 + 結合様式」ほどラベル数は増加しない。

次に、状態数を変えて処理時間の比較を行う．具体的には、状態数が 6 の場合と状態数が 15 の場合の学習に要する時間を比較する．ただし、使用した環境は *Google Colaboratory* であり、スペックは表 7.2 の通りである．

表 7.2 Google Colaboratory の環境

	Spec
OS	Ubuntu バージョン 18.04.6 LTS (Bionic Beaver)
メモリ	54.8GB
GPU	NVIDIA Tesla T4

図 7.2 に「単糖のみ」のデータを学習した時の 1 エポック当たりの処理時間の比較を示す．

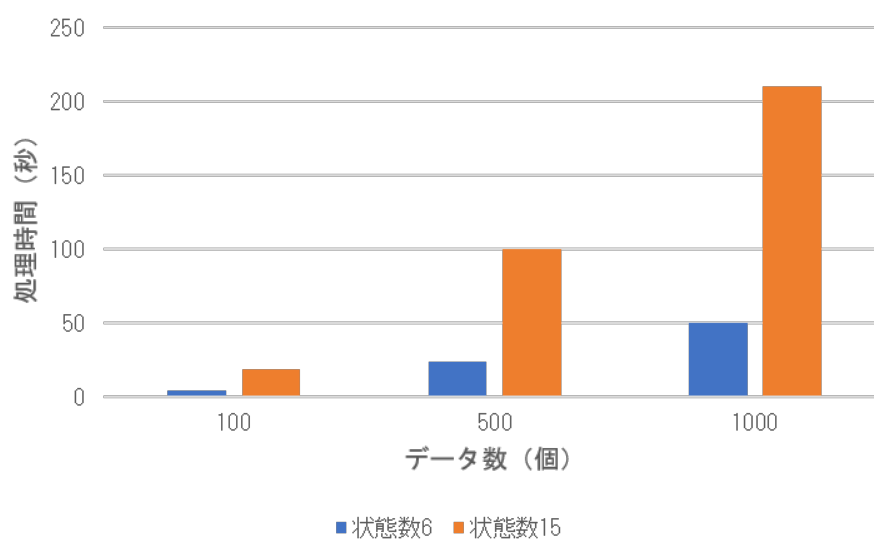


図 7.2 単糖のみを学習した時の 1 エポック当たりの処理時間

図 7.3 に「単糖 + 結合様式」のデータを学習した時の 1 エポック当たりの処理時間の比較を示す。

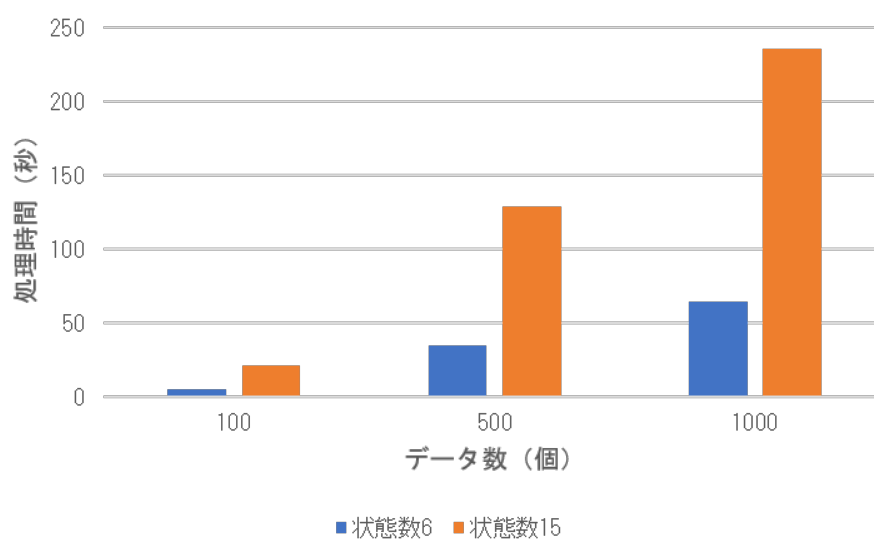


図 7.3 単糖 + 結合様式を学習した時の 1 エポック当たりの処理時間

このように、「単糖のみ」データを用いても「単糖 + 結合様式」のデータを用いても、状態数が多い方が処理時間は長くなる。これは、時間計算量に関する表 4.1 から明らかである。

前述の通り、「単糖 + 結合様式」のデータを用いる場合、状態数が多くないと適切に学習ができない。したがって、図 7.1, 図 7.3 から、「単糖 + 結合様式」のデータを用いる場合は、単糖のみのデータを用いる場合と比べて、処理時間が大幅に長くなることが考えられる。

第 8 章

結論

8.1 結論

実験結果から、単糖と結合様式を組み合わせた糖鎖構造データを学習することによって、微細な糖鎖構造を捉えることができるとわかった。具体的には、結合様式まで考慮した糖鎖構造の共通パターンを得ることができるとわかった。したがって、「単糖 + 結合様式」の学習によって、より正確に糖鎖構造の特徴を取得することができると結論付ける。しかしながら、処理時間は「単糖のみ」の学習の方が短くなるため、正確性と効率性がトレードオフの関係にあることもわかった。

糖鎖構造を正確に捉えられると、生物学的に様々な利点がある。例えば、レクチンは糖鎖構造を認識して結合する。また、糖鎖が生成されるときは「糖転移酵素」というタンパク質が、糖鎖構造を認識して、どの単糖を付加するか決める。このように、糖鎖に関わる生命現象では「構造」が重要な役割を果たす。したがって、糖鎖構造の特徴を正確に捉えることは生物学的に非常に重要であり、様々な生命現象のメカニズム解明に繋がる。

8.2 今後の展望

今後は、*OTMM* を *profile OTMM* というモデルに改良する。これによって、*OTMM* では困難であった「容易に糖鎖構造の共通パターン (糖鎖 *profile*) を取得すること」が可能になる。また、レクチンの種類別に糖鎖構造のデータを学習することで、各レクチンが認識する糖鎖構造を取得することも目指す。本研究が完成することで、糖鎖インフォマティクスや糖鎖の基礎研究を進展させることができ、糖鎖科学の発展に寄与できると確信している。

謝辞

本研究に取り組むにあたり、多くの方々にご指導ご鞭撻を賜りました。指導教官の創価大学理工学部情報システム工学科 篠宮紀彦教授には終始適切なご指導を賜りました。研究内容に留まらず、海外挑戦や進路の相談でも心のこもったご指導を頂き、知見を広げることができました。ここに深謝の意を表します。共生創造理工学科教授 木下フローラ 聖子先生、並びに同学科助教授 細田正恵先生には、本論文の作成にあたり、適切なご助言を賜りました。糖鎖に関する様々な知識を教えて頂けなければ、本論文を完成することはできませんでした。感謝申し上げます。

また、学術文章作法Ⅲの授業では、創価大学学習支援センター助教授 小田玲子先生、教職大学院非常勤講師 小田勝己先生から見分を広める様々な知識を頂きました。頂いた知識は本論文でも活用しております。厚く御礼申し上げます。

そして、様々なアドバイスを頂きました篠宮研究室の先輩方、特に、宮下正明氏、芝田貴之氏、若松篤史氏に感謝いたします。また、同研究室の研究グループ IDA の岸添翔希氏、高梨恭四郎氏には、様々な議論をして頂き、楽しく研究を行うことができました。さらに、2年間共に成長することができた、同期の穂山翔氏、井上颯人氏、林恩智氏、高橋ひめの氏、吉上城大氏のおかげで、卒業研究をやり遂げることができました。改めまして、同研究室の方々に感謝申し上げます。加えて、寺島研究室の小林悠生氏、戸田研究室の塩田俊吾氏には研究に関する意見交換をして頂き、様々な刺激を頂きました。ありがとうございます。

また、日々私を応援してくれる両親のおかげで、自由に挑戦ができ、大学4年間成長することができました。

最後に、いつも暖かく見守ってくださっている創立者池田先生・奥様に深く感謝申し上げます。そして、大学院進学後も勉学に励み、関わった全ての方々への感謝を忘れずに、恩返しができる人材へと成長して参ります。

参考文献

- [1] 福岡伸一, 生物と無生物のあいだ, 講談社現代新書, 2007.
- [2] 佐藤卓, 中村勇吾, 大河遼太, 林里佳子, 動的平衡, THA LTD., 2018-12, <https://tha.jp/8826> (閲覧日 2023-01-09).
- [3] 遺伝学年表, 遺伝学電子博物館, 2003, <https://www.nig.ac.jp/museum/history13.html> (閲覧日 2023-01-09).
- [4] 成地健太郎, ～なぜ糖鎖～ 糖鎖の視点, 医化学創薬株式会社, <https://soyaku.co.jp/point-of-view/> (閲覧日 2023-01-09).
- [5] 長江雅倫, 山口芳樹, 糖鎖の多様性に対応するレクチンの認識システムとシグナリング, 生化学第90巻第5号, 2018, pp.651–663.
- [6] 阿部裕, 糖鎖テクノロジー, MITSUI & CO. STRATEGIC STUDIES INSTITUTE, 2019-04, https://www.mitsui.com/mgssi/ja/report/detail_icsFiles/afeldfile/2020/01/30/1904t_abe_1.pdf (閲覧日 2022-07-30).
- [7] 成地健太郎, 第3話 糖が鎖のように連なった物質「糖鎖」, 医化学創薬株式会社, 2016-06, <https://soyaku.co.jp/column/976/> (閲覧日 2023-01-09).
- [8] 平林淳, 糖鎖とレクチン, 日刊工業新聞社, 2016.
- [9] グルコース, 農業技術事典, <http://lib.ruralnet.or.jp/nrpd/#koumoku=11442> (閲覧日 2023-01-09).
- [10] 木下聖子教授, 創価大学, <https://www.soka.ac.jp/science/symbiosis/feature/lab/kinoshita> (閲覧日 2023-01-09).
- [11] アルツハイマー病を進行させる糖鎖を発見, 理化学研究所, 2015-01, https://www.riken.jp/press/2015/20150115_3/index.html (閲覧日 2023-01-09).
- [12] 膵がん細胞表面の糖鎖をレクチン融合薬で狙い撃ち, 産業技術総合研究所, 2017-09, https://www.aist.go.jp/aist_j/press_release/pr2017/pr20170926/pr20170926.html (閲覧日 2022-07-30).
- [13] 成地健太郎, 第4話 N-結合同型糖鎖, 医化学創薬株式会社, 2016-07, <https://soyaku.co.jp/column/1000/> (閲覧日 2023-01-09).
- [14] Symbol Nomenclature for Glycans (SNFG), National Center for Biotechnology Infor-

- mation, 2022-10, <https://www.ncbi.nlm.nih.gov/glycans/snfg.html> (閲覧日 2023-01-12).
- [15] 小林みどり, あたらしいグラフ理論入門, 牧野書店, 2013.
- [16] Kosuke Hashimoto, Kiyoko Flora Aoki-Kinoshita, Nobuhisa Ueda, Minoru Kanehisa, and Hiroshi Mamitsuka. 2008. A new efficient probabilistic model for mining labeled ordered trees applied to glycobiology, *ACM Transactions on Knowledge Discovery from Data*, vol.2, no.1.
- [17] Aoki, K. F., Ueda, N., Yamaguchi, A., Akutsu, T., Kanehisa, M., and Mamitsuka, H. 2004. Managing and analyzing carbohydrate data. *ACM SIGMOD Rec.* 33, 2, 33–38.
- [18] Aoki, K. F., Yamaguchi, A., Ueda, N., Akutsu, T., Mamitsuka, H., Goto, S., and Kanehisa, M. 2004. KCaM (KEGG Carbohydrate Matcher): A software tool for analyzing the structures of carbohydrate sugar chains. *Nucleic Acids Resear.* 32, W267–W272.
- [19] Kashima, H. and Koyanagi, T. 2002. Kernels for semi-structured data. In *Proceedings of the 19th International Conference on Machine Learning*. Morgan Kaufmann, 291–298.
- [20] Zaki, M. J. 2005. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Trans. Knowl. Data Engin.* 17, 8, 1021–1035.
- [21] Varki, A., Cummings, R., Esko, J., Freeze, H., Hart, G., and Marth, J., Eds. 1999. *Essentials of glycobiology*. Cold Spring Harbor Laboratory Press, New York.
- [22] Bertozzi, C. and Keissling, L. 2001. Chemical glycobiology. *Science* 291, 2357–2364.
- [23] Brooks, S., Dwek, M., and Schumacher, U. 2002. *Functional and molecular glycobiology*. BIOS Scientific.
- [24] Jones N, Pevzner P.(2007). バイオインフォマティクスのためのアルゴリズム入門 (An Introduction to Bioinformatics Algorithm), 渋田哲郎・坂内英夫訳, 共立出版株式会社.
- [25] Daniel Jurafsky and James H. Martin. (2021). *Hidden Markov Models*, <https://web.stanford.edu/jurafsky/slp3/A.pdf> (閲覧日 2022-07-30).
- [26] Michelangelo Diligenti, Paolo Frasconi, and Marco Gori. (2003). Hidden tree markov models for document image classification.
- [27] Nobuhisa Ueda, Kiyoko F. Aoki-Kinoshita, Member, IEEE, Atsuko Yamaguchi, Tatsuya Akutsu, and Hiroshi Mamitsuka. 2005. A probabilistic model for mining labeled ordered trees: capturing patterns in carbohydrate sugar chains, *IEEE transactions on knowledge and data engineering*, vol.17, no.8.
- [28] Kiyoko F. Aoki-Kinoshita, Nobuhisa Ueda, Hiroshi Mamitsuka and Minoru Kanehisa. 2006. ProfilePSTMM: Capturing tree-structure motifs in carbohydrate sugar chains, *Oxford University Press*, vol.22, no.14.

- [29] GlyCosmos Portal, <https://glycosmos.org/glycans> (閲覧日 2023-01-10).
- [30] Chris Smith, The Logarithm of a Sum, Medium, 2021-03, <https://cdsmithus.medium.com/the-logarithm-of-a-sum-69dd76199790> (閲覧日 2023-01-6).
- [31] 木下聖子, 西原祥子, 木確率モデルを用いた知識発見よりのタンパク質の糖鎖認識部位予測, 2008, http://lifesciencedb.jp/houkoku/pdf/A-44_2008.pdf (閲覧日 2022-07-30).

付録 A

レクチンと OTMM

本研究では行わなかったが、参考のため、レクチンが認識する糖鎖構造の共通パターンを得る手順を紹介する。

A.1 レクチンが認識する糖鎖の学習方法

糖鎖の働きの全容を捉えるには、糖鎖とレクチンの相互作用を考えることが重要である。以下の手順で研究を行うことで、レクチンが認識する糖鎖構造の共通パターンを得ることができる。

1. レクチンと結合強度 (親和性) の高い糖鎖構造のデータを入力する。ただし、糖鎖とレクチンの結合の強さは *Average RFU(Relative Fluorescence Units)* という値で判断する。
2. 入力された糖鎖構造のデータを確率モデル (OTMM) で学習する。学習では EM アルゴリズムを用いる。
3. 状態遷移図を得る。この状態遷移図には学習結果が反映されている。
4. *Viterbi* アルゴリズムを用いて学習結果をビジュアル化し、レクチンが認識する糖鎖構造の共通パターン (特徴) を理解する。

レクチンは糖鎖が機能するために生み出された存在 [8] といわれており、糖鎖とレクチンの関係を理解することは糖鎖の性質の深い理解に繋がる。

付録 B

実装で生じたエンジニアリング的な問題

B.1 対数変換

B.1.1 和の計算

本研究では、尤度などが非常に小さな数値になるので、全ての数値を対数変換して計算を行っている。対数変換をすることで、「その数はネイピア数 e の何乗か」が求まり、非常に小さな値でも絶対値が大きな負の数に変換することができる。このようなことから、対数変換をした状態であれば、コンピューターでも容易に小さな値を取り扱うことができる。

また、以下の対数法則により、「掛け算」を「足し算」として扱うことができる。

$$\log(x \times y) = \log x + \log y$$

しかしながら、本研究のアルゴリズムでは、総和 (Σ) を取るので、対数変換を施した状態で真数同士を足し算する必要がある。以下では、

$$\log(x + y)$$

を対数変換した状態で行う方法 [30] を紹介する。

これによって、常に対数変換をした状態で総和 (Σ) を取ることができ、誤差が少ないプログラムが作成できる。

まず、 $x + y$ は

$$x + y = x\left(1 + \frac{y}{x}\right)$$

と因数分解できる (ただし、 $x \neq 0$)。

したがって、対数法則を用いて以下のような恒等式を得られる。

$$\log(x + y) = \log x + \log\left(1 + \frac{y}{x}\right) \quad (1)$$

式 (1) は対称式であり、 x と y を入れ替えても結果は変わらない。

さらに、式 (1) を対数のみを用いて表そうとすると、以下のような式変形ができる。

$$\begin{aligned} \log(x + y) &= \log x + \log\left(1 + e^{\log \frac{y}{x}}\right) \\ &= \log x + \log\left(1 + e^{\log y - \log x}\right) \end{aligned}$$

ここで、

$$\text{smoothmax}(x, y) = x + \log(1 + e^{y-x}) \quad (2)$$

とおく。

式 (2) より、

$$\log(x + y)$$

は対数を用いて以下のように表現することができる。

$$\begin{aligned} \log(x + y) &= \text{smoothmax}(\log x, \log y) \\ \text{smoothmax}(x, y) &= x + \log(1 + e^{y-x}) \end{aligned}$$

以上より、対数を用いて $\log(x + y)$ を計算することができた。

また、*smoothmax* 関数は、実質的に和の演算記号のように振る舞っている。実際に [30] より、*smoothmax* 関数は和の演算記号が持つべき代数的な性質を持っていることが確かめられている。

B.1.2 プログラムへの適用

本研究のプログラムでは,

$$\begin{aligned}\log(x + y) &= \text{smoothmax}(\log x, \log y) \\ \text{smoothmax}(x, y) &= x + \log(1 + e^{y-x})\end{aligned}$$

という式を利用することで, 対数変換をした状態で総和 (Σ) の計算を実行している.

また, 対数関数や指数関数の計算では *Python* の *math* モジュールを使用している. このとき, *smoothmax* 関数の引数の差 ($y - x$) が非常に大きいと, $\log()$ の計算や *exp()* の計算でエラーが生じる. したがって, 引数の差が非常に大きく正の値を取る場合は,

$$\log(1 + e^{y-x}) \approx y - x$$

と近似して,

$$\text{smoothmax}(x, y) = y$$

としている.

また, 引数の差が非常に大きく負の値を取る場合は,

$$\log(1 + e^{y-x}) \approx 0$$

と近似して,

$$\text{smoothmax}(x, y) = x$$

としている.

以下に実際の *smoothmax* 関数を示す.

ソースコード B.1 *smoothmax* 関数

```
1 def smoothmax(x, y):
2     try:
3         return x + Decimal(str(math.log(Decimal(str(1)) + Decimal(str(math
4             .exp(y-x))))))
5     except:
6         if y-x > 0:
7             return copy.deepcopy(y)
8         elif y-x < 0:
9             return copy.deepcopy(x)
```

本研究のプログラムでは, 上記のような近似を用いて計算をしているため, 厳密な計算結果を得ることはできない. しかしながら, 本研究の目的は糖鎖構造の共通パターン (特徴) の「大まかな傾向」を得ることなので, 厳密性の高い計算は必要がないと判断した.

B.1.3 smoothmax の由来

まず,

$$h(x) = x + \log(1 + e^x)$$

という関数を考える.

この関数 h は *smoothmax* 関数と本質的に性質が同じであり,

$$\log(x + y) = \log x + h(\log y - \log x) \quad (3)$$

と表すことができる.

また, 図 A. 1 より, 関数 h は正規化線形関数 (*Rectified linear function*) と類似 (漸近) しており, 正規化線形関数を $x=0$ 付近で *smooth*(滑らか) にした関数と言える. したがって, 名称に *smooth* が入ることが理解できる.

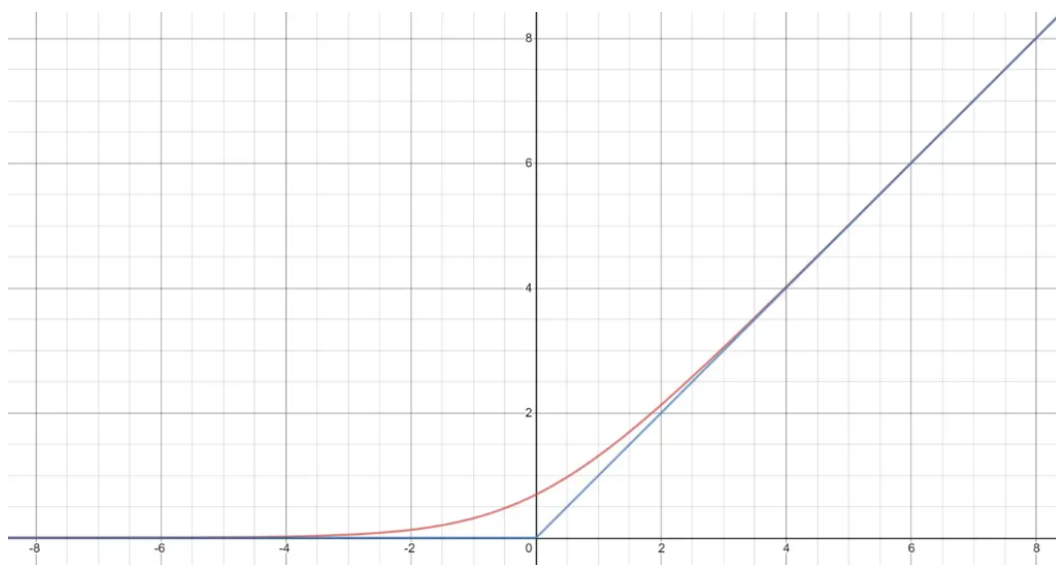


図 B.1 関数 h (赤色) と正規化線形関数 (青色)[30]

次に, 式 (3) の関数 h を正規化線形関数におきかえると,

$$\log(x + y) \approx \begin{cases} \log x & (x \geq y) \\ \log y & (x \leq y) \end{cases}$$

と近似することができる.

上記の近似より, $\log(x + y)$ は x か y のどちらか大きな方を出力する関数といえる. すなわち, 「最大値 (*max*)」を出力する関数と言える. したがって, 名称に *max* が入ることが理解できる. 以上の議論より, $\log(x + y)$ を計算する関数は, *smoothmax* という名称になる. なお, 正規化線形関数 (*ReLU*) は, ニューラルネットワークの活性化関数などで頻繁に用いられる.

B.2 float 型と decimal 型

コンピュータは 2 進数しか扱えないため、10 進数の小数を扱おうとすると僅かに誤差が生じる。Python で 10 進数の小数を扱う場合、特に型指定をしないと *float*(浮動小数点) 型が付与される。この *float* 型は、小数を正確に表すことができないため、計算を繰り返すと誤差が生じてしまう。

このようなことから、本研究のプログラムでは *Decimal*(固定小数点) 型を用いて計算を行う。具体的には、*decimal* モジュールの *Decimal* 型を利用することで、小数の計算における誤差を *float* 型よりも抑えることが可能になる。

以下に、データ数 100, しきい値 1.0, 状態数 6, 単糖のみの糖鎖構造のデータを使用した時の学習の回数 (エポック数) を示す。

表 B.1 データ型が与える学習への影響

環境	エポック数 (float 型)	エポック数 (Decimal 型)
Mac Pro	19	31
Surface Laptop 2	21	31
Google Colaboratory	23	31

表 B.1 からわかるように、*float* 型を用いると、環境によってエポック数が変化することがわかる。すなわち、*float* 型を用いると環境の差による誤差が発生し、結果が変わる可能性がある。

これに対して、*Decimal* 型を用いた場合はエポック数は変わらない。したがって、*Decimal* 型によって、環境の差による誤差が抑えられていることがわかる。以上のことから、*Decimal* 型は環境が変わっても結果が変わる可能性は低く、安定的な動作が期待できる。

付録 C

OTMM のソースコード

以下，本研究で使⽤したプログラムのソースコードを示す．

プログラム C.1

ソースコード C.1 結合様式を含めた糖鎖構造の読み取り (preprocessor_novelty.py)

```

1 import re
2 import copy
3
4 """Class Node"""
5 class Node:
6     def __init__(self, no, name, child, child_num):
7         # nodes リストのインデックスが order になっているので order は属性に入れない
8         self.no = no # position of the text glycan data
9         self.name = name # 単糖 + 結合様式
10
11         self.elder = None # left (immediately elder sibling)
12         self.elder_num = None
13
14         self.younger = None # right (immediately younger sibling)
15         self.younger_num = None
16
17         self.parent = None
18         self.parent_num = None
19
20         self.child = [] # down
21         self.child.append(child)
22
23         self.child_num = []
24         self.child_num.append(child_num)

```

```
25
26 def add_order(self, order):
27     self.order = order
28
29 def add_elder(self, elder, elder_num):
30     self.elder = elder
31     self.elder_num = elder_num
32
33 def add_younger(self, younger, younger_num):
34     self.younger = younger
35     self.younger_num = younger_num
36
37 def add_parent(self, parent, parent_num):
38     self.parent = parent
39     self.parent_num = parent_num
40
41 def add_child(self, child, child_num):
42     self.child.append(child)
43     self.child_num.append(child_num)
44
45 # タンパク質を消去
46 def delete_PROT(row):
47     if re.fullmatch(r'\(a[0-9]-|\(b[0-9]-$', row["IUPAC Condensed
        "][-4:]): # [-4:]は文字列の後ろから 4文字まで
48         row["IUPAC Condensed"] = row["IUPAC Condensed"][:-4] # [:-4]は後
            ろから 4文字を省いたテキスト
49     return row
50
51 # 糖鎖のテキストデータを行ごとに分ける
52 def separate_structure(row):
53     text = row["IUPAC Condensed"]
54
55     stack = [] # stack を利用
56     result = []
57     first_flag = False # 最初の単糖を処理したかどうか
58     for i, word in enumerate(text):
59         if i == 0: # 最初の単糖
60             stack.append(0)
61             first_flag = True
62
63         if word == '(':
```

```
64     if first_flag == True: #最初の単糖
65         result.append(text[0:i])
66         first_flag = False
67         stack.append(i)
68     else:
69         result.append(text[stack.pop():i]) # 単糖
70         stack.append(i) # 括弧
71
72     elif word == ')':
73         result.append(text[stack.pop():i + 1]) # 括弧
74         if text[i+1] != '[':
75             stack.append(i + 1) # 単糖
76
77     elif word == '[':
78         result.append(text[i]) # 角括弧
79         stack.append(i + 1) # 単糖
80
81     elif word == ']':
82         result.append(text[i]) # 角括弧
83         if text[i+1] != '[':
84             stack.append(i + 1) # 単糖
85
86     # 途中で括弧などが終わっていてもスタックをすべて使いきれるようにする
87     if i == len(text)-1: # "文字数-1"が最後の文字のインデックス
88         result.append(text[stack.pop():i + 1])
89
90     row["IUPAC Condensed"] = result
91     return row
92
93 # 親子関係を抽出(この時点では1人の親が複数の子供を持つ)
94 def get_structure(result):
95     nodes = [] # add nodes of glycan (tree)
96     parent_index = set()
97
98     i = len(result)-1
99     while i >= 0:
100         # ルートのみ例外処理(親がないので結合情報を入れない)
101         if i == len(result)-1:
102             if re.fullmatch(r'\([a-z][0-9]-[0-9]\)', result[i-1]) or re.
                fullmatch(r'\([a-z][0-9]-[0-9]\/[0-9]\)', result[i-1]) or
                re.fullmatch(r'\([a-z][0-9]-[a-zA-Z]-[0-9]\)', result[i
```

```

-1]): # 結合様式→子供が 1人だけの時
103     nodes.append(Node(i, result[i], result[i-2], i-2)) # nodes
        [0]==root
104     parent_index.add(i)
105     # print("root:", result[i-2], i-2, "←", result[i], i, "☆
        bond:", result[i-1])

106
107     elif result[i-1] == "]: # 分岐の時
108         nodes.append(Node(i, result[i], result[i-3], i-3)) # nodes
            [0]==root
109         parent_index.add(i)
110         # print("root:", result[i-3], i-3, "←", result[i], i, "☆
            bond:", result[i-2])

111
112     # 通常の処理
113     elif re.fullmatch(r'[0-9a-zA-Z]+' , result[i]) or re.fullmatch(r'[
        a-zA-z]-[0-9a-zA-Z]+' , result[i]) or re.fullmatch(r'[0-9a-zA
        -Z]+[0-9]\/[0-9][a-zA-Z]' , result[i]): # 単糖の場合
114         # 1:1の親子関係を認識
115         if re.fullmatch(r'\([a-z][0-9]-[0-9]\)', result[i-1]) or re.
            fullmatch(r'\([a-z][0-9]-[0-9]\/[0-9]\)', result[i-1]) or
            re.fullmatch(r'\([a-z][0-9]-[a-zA-Z]-[0-9]\)', result[i
            -1]): # 結合様式
116             nodes.append(Node(i, result[i]+result[i+1], result[i-2], i
                -2)) # +result[i+1]が新規性
117             parent_index.add(i)
118             # print(result[i-2], i-2, "←", result[i], i, "☆bond:",
                result[i-1])
119         # 分岐の始まりの場合
120         elif result[i-1] == "]:
121             parent = result[i] # 分岐元の親を保存(最初の分岐の親)。必ず"["の
                直後に対応する"]"が来るので 1つの変数に適時入れるだけで問題な
                い
122             i_parent = i
123             i, parent_index = branch(result, i-1, nodes, parent_index,
                parent, i_parent) # i は"["に対応する"]の 1つ前のインデック
                ス
124             # print("<branch 関数終わり>")
125
126         elif i == 0: #末尾(葉)の場合(親はいない)
127             nodes.append(Node(i, result[i]+result[i+1], None, None)) # +

```

```

        result[i+1]が新規性
128
129     # 分岐の始まりの場合
130     elif result[i] == "]":
131         if result[i+1] == "[":
132             i, parent_index = branch(result, i, nodes, parent_index,
                                     parent, i_parent)
133         else:
134             parent = result[i+1]
135             i_parent = i+1
136             i, parent_index = branch(result, i, nodes, parent_index,
                                     parent, i_parent)
137
138     # 分岐の終わりの場合
139     elif result[i] == "[":
140         # 分岐が連続する場合
141         if result[i-1] == "]":
142             # 最初の分岐の親がこの分岐の親
143             # print("<新たなbranch>")
144             for node in nodes:
145                 if node.no == i_parent:
146                     node.add_child(result[i-3], i-3)
147             # print(result[i-3], i-3, "←", parent, i_parent, "☆bond:",
148                 result[i-2]) # その他 1:1の親子関係については通常の処理で対
149                 応可能
150
151     # その階層の分岐が終わる場合(最後の分岐)
152     elif re.fullmatch(r'\([a-z][0-9]-[0-9]\)', result[i-1]) or re.
153         fullmatch(r'\([a-z][0-9]-[0-9]\/[0-9]\)', result[i-1]) or
154         re.fullmatch(r'\([a-z][0-9]-[a-zA-Z]-[0-9]\)', result[i
155         -1]): # 結合様式
156         # print("<新たなbranch>")
157         for node in nodes:
158             if node.no == i_parent:
159                 node.add_child(result[i-2], i-2)
160             # print(result[i-2], i-2, "←", parent, i_parent, "☆bond:",
161                 result[i-1])
162             # print("<branch 終わり>")
163         i -= 1
164
165     return nodes

```



```
160
161 # get_structure で使う関数
162 def branch(result, j, nodes, parent_index, parent, j_parent):
163     # print("<branch 始まり>")
164     start = j
165     while result[j] != "[":
166         # 分岐中の親子関係を記述
167         if j == start: # ]のとき
168             if j+1 in parent_index: # 一つ右のノードが既出の親の場合
169                 if result[j+1] != result[-1]: # ルートノードではないとき(ルート
                    ノードは既に子どもを登録済み)
170                     for node in nodes:
171                         if node.no == j+1: # j-2の親が見つかったら
172                             node.add_child(result[j-2], j-2)
173             elif result[j+1] == "[": # 1つ右が"["の場合
174                 for node in nodes:
175                     if node.no == j_parent:
176                         node.add_child(result[j-2], j-2)
177                 # print(result[j-2], j-2, "←", parent, j_parent, "☆bond
                    :", result[j-1]) # その他 1:1の親子関係については通常の処
                    理で対応可能
178             else: # 初めて親が出た時
179                 nodes.append(Node(j+1, result[j+1]+result[j+2], result[j-2],
                    j-2)) # +result[j+2]が新規性
180                 parent_index.add(j+1)
181                 # print(result[j-2], j-2, "←", result[j+1], j+1, "☆bond:",
                    result[j-1])
182
183     # 分岐中の 1:1の親子関係を認識
184     elif re.fullmatch(r'[0-9a-zA-Z]+', result[j]) or re.fullmatch(r'[
        a-zA-Z]-[0-9a-zA-Z]+', result[j]) or re.fullmatch(r'[0-9a-zA
        -Z]+[0-9]\/[0-9][a-zA-Z]', result[j]): # 単糖
185         if re.fullmatch(r'\([a-z][0-9]-[0-9]\)', result[j-1]) or re.
            fullmatch(r'\([a-z][0-9]-[0-9]\/[0-9]\)', result[j-1]) or
            re.fullmatch(r'\([a-z][0-9]-[a-zA-Z]-[0-9]\)', result[j
            -1]): # 結合様式
186             nodes.append(Node(j, result[j]+result[j+1], result[j-2], j
                -2)) # +result[j+1]が新規性
187             parent_index.add(j)
188             # print(result[j-2], j-2, "←", result[j], j, "☆bond:",
                result[j-1])
```

```

189
190     # 分岐の中の分岐に対応
191     elif result[j] == "]:
192         if result[j+1] == "[": # 3つ以上の分岐のときの2つ目以上の分岐
193             parent = copy.deepcopy(parent) # 今の親を維持
194             j_parent = copy.deepcopy(j_parent) # 今の親を維持
195         else: # 1つ目の分岐のとき
196             parent = result[j+1] # 1つ右が親(参照渡しを利用して
197                 global に影響を与える)
198             j_parent = j+1 # 1つ右が親(参照渡しを利用して
199                 global に影響を与える)
200             # print("<新たなbranch>")
201             j, parent_index = branch(result, j, nodes, parent_index, parent,
202                 j_parent) # 再帰
203             j -= 1 # この処理で再帰後のresult[j]は[になり、次の
204                 if文に引っかからなくなる
205
206     # 一つ左が結合様式の時(j_parent_branchの子どもの時)
207     if re.fullmatch(r'\([a-z][0-9]-[0-9]\)', result[j-1]) or re.
208         fullmatch(r'\([a-z][0-9]-[0-9]\/[0-9]\)', result[j-1]) or
209         re.fullmatch(r'\([a-z][0-9]-[a-zA-Z]-[0-9]\)', result[j
210             -1]): # 結合様式
211         if re.fullmatch(r'[0-9a-zA-Z]+', result[j-2]) or re.fullmatch
212             (r'[a-zA-Z]-[0-9a-zA-Z]+', result[j-2]) or re.fullmatch(r
213                 '[0-9a-zA-Z]+[0-9]\/[0-9][a-zA-Z]', result[j-2]): # 単糖
214             for node in nodes:
215                 if node.no == j_parent:
216                     node.add_child(result[j-2], j-2)
217
218     # 1つ左が"]"のとき(分岐が3つ以上になっているとき or "]"["のように
219         になっているとき)
220     if result[j-1] == "]:
221         if re.fullmatch(r'\([a-z][0-9]-[0-9]\)', result[j-2]) or re.
222             fullmatch(r'\([a-z][0-9]-[0-9]\/[0-9]\)', result[j-2])
223             or re.fullmatch(r'\([a-z][0-9]-[a-zA-Z]-[0-9]\)', result
224                 [j-2]): # 結合様式
225             if re.fullmatch(r'[0-9a-zA-Z]+', result[j-3]) or re.
226                 fullmatch(r'[a-zA-Z]-[0-9a-zA-Z]+', result[j-3]) or re.
227                 fullmatch(r'[0-9a-zA-Z]+[0-9]\/[0-9][a-zA-Z]', result[
228                     j-3]): # 単糖
229             for node in nodes:

```

```
214         if node.no == j_parent:
215             node.add_child(result[j-3], j-3)
216         # print("<branch 関数終わり>")
217
218     j -= 1 # 前へ
219
220     # 葉にも対応
221     if result[j] == "[":
222         nodes.append(Node(j+1, result[j+1]+result[j+2], None, None)) #
            +result[j+2]が新規性
223
224         if result[j-1] == "]": # 1つ上の階層の親に行かないようにする
225             pass
226
227     return j+1, parent_index #"["の1つ前
228
229     """OTMM 独自の処理"""
230     # OTMM 用に兄弟関係を抽出 (1 人の親が 1 人の子供を持つようにする)
231     def create_siblings(nodes):
232         for node in nodes:
233             if len(node.child) >= 2: # 子供が 2人以上いるとき
234                 parent = node
235                 for i in range(0, len(parent.child)):
236                     if i == 0: # 先頭の子供が i=0 (すなわち、eldest (長男) のとき)
237                         child = parent.child[i] # 子供を入れる
238                         child_num = parent.child_num[i]
239                         sibling = parent.child[i+1]
240                         sibling_num = parent.child_num[i+1]
241                         for node in nodes:
242                             if node.no == child_num:
243                                 node.add_younger(sibling, sibling_num) # 若い
244                                 node.add_parent(parent.name, parent.no) # 親の処理
245                             elif node.no == sibling_num:
246                                 node.add_elder(child, child_num) # 年上
247
248                     elif i == len(parent.child)-1: # youngest のとき
249                         youngest = parent.child[i]
250                         youngest_num = parent.child_num[i]
251                         elder = parent.child[i-1]
252                         elder_num = parent.child_num[i-1]
253                         for node in nodes:
```

```
254         if node.no == youngest_num:
255             node.add_elder(elder, elder_num) # 年上のみ
256
257     else:
258         elder = parent.child[i]
259         elder_num = parent.child_num[i]
260         younger = parent.child[i+1]
261         younger_num = parent.child_num[i+1]
262         for node in nodes:
263             if node.no == elder_num:
264                 node.add_younger(younger, younger_num) # 若い
265             elif node.no == younger_num:
266                 node.add_elder(elder, elder_num) # 年上
267
268     #更新する
269     del parent.child[1:len(parent.child)] # 最も近い子供以外を消去
270     parent.child = parent.child[0]
271     del parent.child_num[1:len(parent.child_num)] # 最も近い子供以外
272     #を消去
273     parent.child_num = parent.child_num[0]
274
275     elif len(node.child) == 1: # 子供が1人の時
276         node.child = node.child[0]
277         node.child_num = node.child_num[0]
278
279     elif node.child[0] == None:
280         node.child = None
281         node.child_num = None
282
283     return nodes
284
285 # 親を登録(上記の兄弟の処理が終わっている前提)
286 def create_parent(nodes):
287     for node in nodes:
288         if node.child != None:
289             parent = node
290             child = node.child
291             child_num = node.child_num
292             for node in nodes:
293                 if node.no == child_num:
294                     node.add_parent(parent.name, parent.no)
```

```
294
295     return nodes
296
297     """単糖のインスタンスをNodeの属性 (parent、child、elder、younger) に代入
        """
298
299     # 単糖の名前ではなくNodeインスタンスを入れる
300     def set_instance(nodes):
301         for node in nodes:
302             # parentをNodeインスタンスに
303             if node.parent != None:
304                 for instance in nodes:
305                     if instance.no == node.parent_num:
306                         node.parent = instance # 親のインスタンスを入れる
307             # childをNodeインスタンスに
308             # if node.child[0] != None: # childのみリスト
309             if node.child != None: # childのみリスト
310                 for instance in nodes:
311                     # if instance.no == node.child_num[0]:
312                     if instance.no == node.child_num:
313                         # node.child[0] = instance # 子のインスタンスを入れる
314                         node.child = instance
315             # youngerをNodeインスタンスに
316             if node.younger != None:
317                 for instance in nodes:
318                     if instance.no == node.younger_num:
319                         node.younger = instance # youngerのインスタンスを入れる
320             # elderをNodeインスタンスに
321             if node.elder != None:
322                 for instance in nodes:
323                     if instance.no == node.elder_num:
324                         node.elder = instance # elderのインスタンスを入れる
325
326     return nodes
327
328     # 末端の葉に印をつける
329     def find_mark(row):
330         i = 0
331         glycan = row["IUPAC Condensed"]
332         for node in glycan:
333             if node.child == None and node.younger == None: # これが厳密な末端
```

の葉

```

334     node.leaf_order = i
335     i += 1
336     row["IUPAC Condensed"] = glycan
337
338     return row

```

プログラム C.2

ソースコード C.2 OTMM の学習 (algorism_decimal.py)

```

1  """
2  主にlikelihoodとlearningで使用するアルゴリズム
3  新規性のある研究も既存の研究もこの部分は同じ
4  小数点の加算と減算しか処理が無いのでDecimal(固定小数点)で誤差を無くす
5  """
6
7  import math
8  import pandas as pd
9  import copy # Pythonでは関数においてミュータブルな変数は参照渡しのため
10 from decimal import Decimal
11
12 """
13 log(x+y)の近似
14 sum of logs
15 """
16 # math.exp(0) = 1なので条件分岐する必要ない
17 # log(1+e^(y-x))をすることで、e^(y-x)が非常に小さい場合log(1+e^(y-x))
    ≐ 0を出力できる→ほぼxが返される
18 def smoothmax(x, y):
19     try:
20         return x + Decimal(str(math.log(Decimal(str(1)) + Decimal(str(math
            .exp(y-x)))))) # exp(± 710)くらいでエラーが発生
21     except:
22         # logsumexpと同じ処理
23         if y-x > 0: # e^(y-x)が非常に大きくなるので1+ e^(y-x) ≐ e^(y-x)と
            する
24             return copy.deepcopy(y)
25         elif y-x < 0:
26             return copy.deepcopy(x)
27
28 """Calculate upward and backward"""

```

```
29 def calc_up(q, p, back, a_a, b, state_set):
30     #calculate up
31     if p.child == None:
32         return b.at[q, p.name]
33     else:
34         total = Decimal(str(0))
35         for m in state_set:
36             if back.at[m, p.child.no] == 100: # あり得ない値(100)を発見
37                 print("100(unexpected value) found at up") # 計算していない
38                     back を使っている状態
39             # j is the p's eldest children (thus, j == p.child)
40             if total == 0 and m == state_set[0]: # 最初の値はそのまま代入
41                 total += a_a[q][m] + back.at[m, p.child.no] # 対数なので足し算
42                     は掛け算
43             else:
44                 total = copy.deepcopy(smoothmax(total, a_a[q][m] + back.at[m,
45                     p.child.no])) #
46                     total は和なので smoothmax に total を代入すればよい
47         return b.at[q, p.name] + total
48
49 def calc_back(m, j, up, back, a_b, state_set):
50     #calculate back
51     if j.younger == None: # youngest child and root
52         if up.at[m, j.no] == 100:
53             print("100(unexpected value) found at back") # 計算していない
54                 up を使っている状態
55         return up.at[m, j.no]
56     else:
57         total = Decimal(str(0))
58         for l in state_set:
59             if total == 0 and l == state_set[0]: # 最初の値はそのまま代入
60                 total += a_b[m][l] + back.at[l, j.younger.no] # 対数なので足し
61                     算は掛け算
62             else:
63                 total = copy.deepcopy(smoothmax(total, a_b[m][l] + back.at[l,
64                     j.younger.no])) #
65                     total は和なので smoothmax に total を代入すればよい
66         return up.at[m, j.no] + total
67
68 def calc_likelihood(df, pi, a_a, a_b, b, state_set):
69     i = 0
```

```
62 likelihood = []
63 for row in df.itertuples():
64     # count how many glycans did we use
65     if i == 0:
66         print("Number of glycans in likelihood", i)
67     elif (i+1)%100 == 0:
68         print("Number of glycans in likelihood", i+1) # 0からインデックスが始まるため
69
70     glycan = row[2]
71
72     # 識別子で区別する
73     sugar_id = []
74     for sugar in glycan:
75         sugar_id.append(sugar.no)
76     sugar_id = sorted(sugar_id)
77
78     # make upward prob
79     up = [[Decimal(str(100))] * len(glycan) for i in [Decimal(str(100))] * len(state_set)]
80     up = pd.DataFrame(up, index=state_set, columns=sugar_id)
81
82     # make backward prob
83     back = [[Decimal(str(100))] * len(glycan) for i in [Decimal(str(100))] * len(state_set)]
84     back = pd.DataFrame(back, index=state_set, columns=sugar_id)
85
86     # calculate up and back
87     for node in reversed(glycan):
88         for state in state_set:
89             up.at[state, node.no] = copy.deepcopy(calc_up(state, node,
90                 back, a_a, b, state_set))
91             back.at[state, node.no] = copy.deepcopy(calc_back(state, node,
92                 up, back, a_b, state_set))
93
94     like = Decimal(str(0))
95     for l in state_set:
96         if like == 0 and l == state_set[0]:
97             like += pi[l] + up.at[l, sugar_id[-1]] # 最もインデックスが大きい数([-1])は必ずルート
98         else:
```



```
197         like = copy.deepcopy(smoothmax(like, pi[l] + up.at[l, sugar_id
198             [-1]]))
199     likelihood.append(like)
200     i += 1
201     return likelihood
202
203     """
204     Learning(EM アルゴリズム)
205     """
206
207     """Forward probability"""
208     def calc_forward(l, j, down, forward, up, a_a, a_b, b, pi, state_set
209         ):
210         #calculate forward
211         if j.parent == None and j.elder == None and j.younger == None: #
212             when j == root
213             return Decimal(str(0)) # 既存の論文で言及されていない部分, log(1) =
214                 0
215         elif j.elder == None: # eldest children
216             total = Decimal(str(0))
217             for q in state_set:
218                 if down.at[q, j.parent.no] == 100:
219                     print("100(unexpected value) found at forward!")
220                 if total == 0 and q == state_set[0]:
221                     total += a_a[q][l] + down.at[q, j.parent.no] + b.at[q, j.
222                         parent.name]
223             else:
224                 total = copy.deepcopy(smoothmax(total, a_a[q][l] + down.at[q,
225                     j.parent.no] + b.at[q, j.parent.name]))
226             return total
227         else:
228             total = Decimal(str(0))
229             for m in state_set:
230                 if forward.at[m, j.elder.no] == 100:
231                     print("100(unexpected value) found at forward!!")
232                 if total == 0 and m == state_set[0]:
233                     total += a_b[m][l] + forward.at[m, j.elder.no] + up.at[m, j.
234                         elder.no] # 対数なので足し算は掛け算
235             else:
236                 total = copy.deepcopy(smoothmax(total, a_b[m][l] + forward.at[
237                     m, j.elder.no] + up.at[m, j.elder.no]))
```

```
130     return total
131
132     """Downward probability"""
133 def calc_down(l, j, forward, back, pi, a_b, state_set):
134     #calculate downward
135     if j.parent == None and j.elder == None and j.younger == None: #
136         when j == root
137         return pi[l]
138     elif j.younger == None:
139         if forward.at[l, j.no] == 100:
140             print("100(unexpected value) found at down!")
141         return forward.at[l, j.no]
142     else:
143         total = Decimal(str(0))
144         for m in state_set:
145             if forward.at[l, j.no] == 100:
146                 print("100(unexpected value) found at down!!")
147             if total == 0 and m == state_set[0]:
148                 total += a_b[l][m] + back.at[m, j.younger.no] # 対数なので足し
149                                     算は掛け算
150             else:
151                 total = copy.deepcopy(smoothmax(total, a_b[l][m] + back.at[m,
152                                     j.younger.no]))
153         return forward.at[l, j.no] + total
154
155     """Learning (EM アルゴリズム)"""
156 def EM(df, pi_original, a_a_original, a_b_original, b_original,
157     state_set, label_set, L, epsilon):
158     t = 0
159     L_all = [] # 全体の期待値
160     print("likelihood before learning", L, "\n")
161     L_all.append(L)
162
163     # 参照渡しのため値を別のアドレス(変数)にコピー
164     pi = copy.deepcopy(pi_original)
165     a_a = copy.deepcopy(a_a_original)
166     a_b = copy.deepcopy(a_b_original)
167     b = copy.deepcopy(b_original)
168
169     while True:
170         print("t(epoc)=", t)
```

```
167     # 5 of pseudo code
168     # initialize mu of T_u
169     mu_aa_t = [[Decimal(str(0))] * len(state_set) for i in [Decimal(
        str(0))] * len(state_set)]
170
171     mu_ab_t = [[Decimal(str(0))] * len(state_set) for i in [Decimal(
        str(0))] * len(state_set)]
172
173     B_t = [[Decimal(str(0))] * len(label_set) for i in [Decimal(str
        (0))] * len(state_set)]
174     mu_b_t = pd.DataFrame(B_t, index=state_set, columns=label_set)
175
176     mu_pi_t = [Decimal(str(0))] * len(state_set)
177
178     # 6 of pseudo code
179     count = 0
180     for row in df.itertuples():
181         if count == 0:
182             print("Number of glycans in learning", count)
183         elif (count+1)%100 == 0:
184             print("Number of glycans in learning", count+1) # 0からインデ
                ックスが始まるため
185
186     # 7 of pseudo code
187     glycan = row[2]
188
189     # initialize mu of the glycan
190     mu_aa_u = [[Decimal(str(0))] * len(state_set) for i in [Decimal
        (str(0))] * len(state_set)]
191
192     mu_ab_u = [[Decimal(str(0))] * len(state_set) for i in [Decimal
        (str(0))] * len(state_set)]
193
194     B_u = [[Decimal(str(0))] * len(label_set) for i in [Decimal(str
        (0))] * len(state_set)]
195     mu_b_u = pd.DataFrame(B_u, index=state_set, columns=label_set)
196
197     mu_pi_u = [Decimal(str(0))] * len(state_set)
198
199     # 識別子で区別する
200     sugar_id = []
```

```
201     for sugar in glycan:
202         sugar_id.append(sugar.no)
203     sugar_id = sorted(sugar_id)
204     # make upward prob
205     up = [[Decimal(str(100))] * len(glycan) for i in [Decimal(str
        (100))] * len(state_set)]]
206     up = pd.DataFrame(up, index=state_set, columns=sugar_id)
207
208     # make backward prob
209     back = [[Decimal(str(100))] * len(glycan) for i in [Decimal(str
        (100))] * len(state_set)]]
210     back = pd.DataFrame(back, index=state_set, columns=sugar_id)
211
212     # make forward prob
213     forward = [[Decimal(str(100))] * len(glycan) for i in [Decimal(
        str(100))] * len(state_set)]]
214     forward = pd.DataFrame(forward, index=state_set, columns=
        sugar_id)
215
216     # make downward prob
217     down = [[Decimal(str(100))] * len(glycan) for i in [Decimal(str
        (100))] * len(state_set)]]
218     down = pd.DataFrame(down, index=state_set, columns=sugar_id)
219
220     # U, Bを計算する(7 to 9 of pseudo code)
221     # reversed(glycan)...bottom-up and right-to-left dynamic
        programming procedure
222     for node in reversed(glycan):
223         for state in state_set:
224             up.at[state, node.no] = copy.deepcopy(calc_up(state, node,
                back, a_a, b, state_set))
225             back.at[state, node.no] = copy.deepcopy(calc_back(state,
                node, up, back, a_b, state_set))
226
227     # D, Fを計算する(10 to 12 of pseudo code)
228     # glycan...top-down and left-to-right dynamic programming
        procedure
229     for node in glycan:
230         for state in state_set:
231             forward.at[state, node.no] = copy.deepcopy(calc_forward(state
                , node, down, forward, up, a_a, a_b, b, pi, state_set))
```

```

232         down.at[state, node.no] = copy.deepcopy(calc_down(state,
233             node, forward, back, pi, a_b, state_set))
234
235     # calculate L(T_u) (尤度)
236     L_u = Decimal(str(0)) # 入力データ 1つ 1つの尤度を保存
237     i = glycan[0].no # optional
238     for l in state_set:
239         if L_u == 0 and l == state_set[0]:
240             L_u += up.at[l, i] + down.at[l, i]
241         else:
242             L_u = copy.deepcopy(smoothmax(L_u, up.at[l, i] + down.at[l,
243                 i]))
244
245     # 期待値の計算 (13 of pseudo code)
246     # calculate mu_aa_u[q][l]
247     for q in state_set:
248         for l in state_set:
249             exist_child = False # 該当するchildがいるかどうか
250             total = Decimal(str(0))
251             for p in glycan:
252                 if p.child != None:
253                     exist_child = True
254                     # p.child.no means j in the thesis of OTMM
255                     if total == 0:
256                         total += down.at[q, p.no] + b.at[q, p.name] + a_a[q][
257                             l] + back.at[l, p.child.no]
258                     else:
259                         total = copy.deepcopy(smoothmax(total, down.at[q, p.no]
260                             + b.at[q, p.name] + a_a[q][l] + back.at[l, p.
261                             child.no]))
262             if exist_child == True or total != 0: # 本来
263                 total × L_u なので total=0の時は 0 (対数変換によって掛け算が足
264                     算になる)
265                 mu_aa_u[q][l] = total - L_u # 対数の引き算は割り算
266             else:
267                 mu_aa_u[q][l] = Decimal(str(0)) # 0にしてパラメータの更新に
268                     影響が出ないようにする
269
270     # calculate mu_ab_u[q][l]
271     for q in state_set:
272         for l in state_set:

```

```

265         exist_younger = False # younger sibling があるかどうか
266         total = Decimal(str(0))
267         for j in glycan:
268             if j.younger != None: # X(j) != empty set
269                 exist_younger = True
270                 # p.child.no means j in the thesis of OTMM
271                 if total == 0:
272                     total += forward.at[q, j.no] + a_b[q][1] + back.at[1,
273                                     j.younger.no] + up.at[q, j.no]
274                 else:
275                     total = copy.deepcopy(smoothmax(total, forward.at[q, j.
276                                     no] + a_b[q][1] + back.at[1, j.younger.no] + up.
277                                     at[q, j.no]))
278             if exist_younger == True or total != 0:
279                 mu_ab_u[q][1] = total - L_u # 対数の引き算は割り算
280             else:
281                 mu_ab_u[q][1] = Decimal(str(0))
282
283 # calculate mu_b_u[m][o_h]
284 for m in state_set:
285     for o_h in label_set:
286         exist_oh = False # whether the sugar(o_h) exists or not
287         total = Decimal(str(0))
288         for i in glycan:
289             if i.name == o_h:
290                 exist_oh = True
291                 if total == 0:
292                     total += down.at[m, i.no] + up.at[m, i.no]
293                 else:
294                     total = copy.deepcopy(smoothmax(total, down.at[m, i.no]
295                                     + up.at[m, i.no]))
296             if exist_oh == True or total != 0:
297                 mu_b_u.at[m, o_h] = total - L_u # 対数の引き算は割り算
298             else:
299                 mu_b_u.at[m, o_h] = Decimal(str(0)) # 入力データには単糖
300                 o_h が存在しないので期待値は 0
301
302 # calculate mu_pi_u[m]
303 for m in state_set:
304     mu_pi_u[m] = (pi[m] + up.at[m, sugar_id[-1]]) - L_u # 最も大
305     さい識別子がルートノード

```

```

300
301     # for each param, do mu_t = mu_t + mu_u (14 of pseudo code)
302     # mu_aa_t = mu_aa_t + mu_aa_u
303     for q in state_set:
304         for l in state_set:
305             if mu_aa_u[q][l] == 0:
306                 mu_aa_t[q][l] += Decimal(str(0))
307             else:
308                 if mu_aa_t[q][l] == 0:
309                     mu_aa_t[q][l] += copy.deepcopy(mu_aa_u[q][l])
310                 else:
311                     mu_aa_t[q][l] = copy.deepcopy(smoothmax(mu_aa_t[q][l],
312                                                                mu_aa_u[q][l]))
312
313     # mu_ab_t = mu_ab_t + mu_ab_u
314     for q in state_set:
315         for l in state_set:
316             if mu_ab_u[q][l] == 0:
317                 mu_ab_t[q][l] += Decimal(str(0))
318             else:
319                 if mu_ab_t[q][l] == 0:
320                     mu_ab_t[q][l] += copy.deepcopy(mu_ab_u[q][l])
321                 else:
322                     mu_ab_t[q][l] = copy.deepcopy(smoothmax(mu_ab_t[q][l],
323                                                                mu_ab_u[q][l]))
323
324     # mu_b_t = mu_b_t + mu_b_u
325     for m in state_set:
326         for o_h in label_set:
327             if mu_b_u.at[m, o_h] == 0:
328                 mu_b_t.at[m, o_h] += Decimal(str(0))
329             else:
330                 if mu_b_t.at[m, o_h] == 0:
331                     mu_b_t.at[m, o_h] += copy.deepcopy(mu_b_u.at[m, o_h])
332                 else:
333                     mu_b_t.at[m, o_h] = copy.deepcopy(smoothmax(mu_b_t.at[m,
334                                                                o_h], mu_b_u.at[m, o_h]))
334
335     # mu_pi_t = mu_pi_t + mu_pi_u
336     for m in state_set:
337         if mu_pi_u[m] == 0:

```

```

338         mu_pi_t[m] += Decimal(str(0))
339     else:
340         if mu_pi_t[m] == 0:
341             mu_pi_t[m] += copy.deepcopy(mu_pi_u[m])
342         else:
343             mu_pi_t[m] = copy.deepcopy(smoothmax(mu_pi_t[m], mu_pi_u[m]
344                                                  ]))
345     count += 1
346
347     # update a_a
348     for q in state_set:
349         denominator = Decimal(str(0))
350         for l_dash in state_set:
351             if denominator == 0 and l_dash == state_set[0]:
352                 denominator += copy.deepcopy(mu_aa_t[q][l_dash])
353             else:
354                 denominator = copy.deepcopy(smoothmax(denominator, mu_aa_t[q]
355                                                         ][l_dash]))
356         for l in state_set:
357             numerator = copy.deepcopy(mu_aa_t[q][l])
358             a_a[q][l] = numerator - denominator # 対数の引き算は割り算
359
360     # update a_b
361     for q in state_set:
362         denominator = Decimal(str(0))
363         for l_dash in state_set:
364             if denominator == 0 and l_dash == state_set[0]:
365                 denominator += copy.deepcopy(mu_ab_t[q][l_dash])
366             else:
367                 denominator = copy.deepcopy(smoothmax(denominator, mu_ab_t[q]
368                                                         ][l_dash]))
369         for l in state_set:
370             numerator = copy.deepcopy(mu_ab_t[q][l])
371             a_b[q][l] = numerator - denominator
372
373     # update b
374     for m in state_set:
375         denominator = Decimal(str(0))
376         for o_i in label_set:
377             if denominator == 0 and o_i == label_set[0]:
378                 denominator += copy.deepcopy(mu_b_t.at[m, o_i])

```



```

376         else:
377             denominator = copy.deepcopy(smoothmax(denominator, mu_b_t.at[
378                 m, o_i]))
379         for o_h in label_set:
380             numerator = copy.deepcopy(mu_b_t.at[m, o_h])
381             b.at[m, o_h] = numerator - denominator
382
383     # update pi
384     for m in state_set:
385         numerator = copy.deepcopy(mu_pi_t[m])
386         denominator = Decimal(str(0))
387         for k in state_set:
388             if denominator == 0 and k == state_set[0]:
389                 denominator += copy.deepcopy(mu_pi_t[k])
390             else:
391                 denominator = copy.deepcopy(smoothmax(denominator, mu_pi_t[k
392                     ])))
393         pi[m] = numerator - denominator
394
395     t += 1
396     # calculate L_t(T) using the param updated (17 of the pseudo code)
397     likelihoods = calc_likelihood(df, pi, a_a, a_b, b, state_set)
398     L_all.append(sum(likelihoods)) # 対数(likelihood) の足し算は掛け算
399
400     # L_all.append(sum(L_T)) # 現在のパラメータを使って計算をしたいので
401     print("Likelihood", L_all[t])
402     print("Difference of the likelihoods", L_all[t] - L_all[t-1], "\n
403         ")
404     if abs(L_all[t] - L_all[t-1]) < epsilon:
405         return pi, a_a, a_b, b, L_all

```

プログラム C.3

ソースコード C.3 糖鎖構造の解析 (parsing.py)

```

1  """
2  Parsing (Retrieve what was learned by retrieving the most likely state
3  paths)
4  Viterbi アルゴリズムで経路探索
5  """
6  import numpy as np

```

```
7 import pandas as pd
8
9 # 識別子で区別する
10 def parse_glycan(glycan, state_set, pi, a_a, a_b, b):
11     sugar_id = []
12     for sugar in glycan:
13         sugar_id.append(sugar.no)
14     sugar_id = sorted(sugar_id)
15
16     """Make data variable for phi"""
17     phi_up = np.zeros((len(state_set), len(glycan)))
18     phi_up = pd.DataFrame(phi_up, index=state_set, columns=sugar_id)
19
20     phi_back = np.zeros((len(state_set), len(glycan)))
21     phi_back = pd.DataFrame(phi_back, index=state_set, columns=
        sugar_id)
22
23     """Calculate phi"""
24     # bottom-up and right-to-left dynamic programming procedure
25     for node in reversed(glycan):
26         for state in state_set:
27             phi_up.at[state, node.no] = calc_phi_up(state, node,
                phi_back, a_a, b, state_set)
28             phi_back.at[state, node.no] = calc_phi_back(state, node,
                phi_up, phi_back, a_b, state_set)
29
30     p = most_likely_prob(pi, phi_up, state_set, sugar_id)
31     print("The probability that all labels are outputted along the most
        likely state transition:", p)
32     z = most_likely_state(glycan, state_set, sugar_id, pi, a_a, a_b, b
        , phi_up, phi_back)
33     for node in glycan:
34         print("Node:", node.name, node.no, "Most likely state:", z[node
            .no])
35     print()
36
37     """
38     Calculate psi
39     top-down and left-to-right manner
40     """
41     def calc_phi_up(q, p, phi_back, a_a, b, state_set):
```

```
42 #calculate phi up
43 if p.child == None:
44     return b.at[q, p.name]
45 else:
46     prob = []
47     for m in state_set:
48         if phi_back.at[m, p.child.no] == 0:
49             print("Unexpected value at calc phi up") # 計算していない
               phi_back を使っている状態
50     #  $\Sigma$  の部分をmax に
51     # j is the p's eldest children (thus, j == p.child)
52     prob.append(a_a[q][m] + phi_back.at[m, p.child.no]) # 対数なので
               足し算は掛け算
53     return b.at[q, p.name] + max(prob)
54
55 def calc_phi_back(m, j, phi_up, phi_back, a_b, state_set):
56     #calculate phi back
57     if j.younger == None: # youngest
58         if phi_up.at[m, j.no] == 0:
59             print("Unexpected value at calc phi back") # 計算していない
               phi_up を使っている状態
60         return phi_up.at[m, j.no]
61     else:
62         prob = []
63         for l in state_set:
64             #  $\Sigma$  の部分をmax に
65             prob.append(a_b[m][l] + phi_back.at[l, j.younger.no]) # 対数なので
               足し算は掛け算
66         return phi_up.at[m, j.no] + max(prob)
67
68 # returns the most likely state for the given node
69 def psi_up(q, p, phi_back, a_a, state_set):
70     if p.child == None: #  $\psi$ 
               _up において p は必ず子供を持つのでこの条件は該当しないはず
71         print("ERROR! node p should have a child!")
72     else:
73         prob = np.empty(0)
74         for m in state_set: # m proceeds from 0(state_set[0]) to state_set
               [-1]
75             if phi_back.at[m, p.child.no] == 0:
76                 print("Unexpected value at psi up") # 計算していない
```

```

        back を使っている状態
77     #  $\Sigma$  の部分をargmax に
78     # j is the p's eldest children (thus, j == p.child)
79     prob = np.insert(prob, m, a_a[q][m] + phi_back.at[m, p.child.no
        ]) # 対数なので足し算は掛け算
80
81     return np.argmax(prob) # m represents a state and return it
82
83 # returns the most likely state for the given node
84 def psi_back(m, j, phi_back, a_b, state_set):
85     if j.younger == None: #  $\Psi$ 
86         _back において j は必ず younger を持つのでこの条件は該当しないはず
87         print("ERROR! j should have a younger sibling!")
88     else:
89         prob = np.empty(0)
90         for l in state_set: # l proceeds from 0(state_set[0]) to state_set
91             #  $\Sigma$  の部分をargmax に
92             prob = np.insert(prob, l, a_b[m][l] + phi_back.at[l, j.younger.
93                 no]) # 対数なので足し算は掛け算
94
95         return np.argmax(prob) # l represents a state and return it
96
97 def most_likely_prob(pi, phi_up, state_set, sugar_id):
98     likely = []
99     for l in state_set:
100         likely.append(pi[l] + phi_up.at[l, sugar_id[-1]])
101     most_likely = max(likely)
102
103     return most_likely
104
105 """Calculate the most likely state for the given node"""
106 def most_likely_state(glycan, state_set, sugar_id, pi, a_a, a_b, b,
107     phi_up, phi_back):
108     z = dict()
109     # top-down and left-to-right manner
110     for node in glycan:
111         if node.no == sugar_id[-1]: # root
112             prob = np.empty(0)
113             for l in state_set:
114                 prob = np.insert(prob, l, pi[l] + phi_up.at[l, node.no]) #

```

```

        node.no==sugar_id[-1]
112     z[node.no] = np.argmax(prob) # return the most likely state of
        the root node
113     elif node.elder == None and node.parent != None:
114         z[node.no] = psi_up(z[node.parent.no], node.parent, phi_back,
            a_a, state_set)
115     else:
116         z[node.no] = psi_back(z[node.elder.no], node.elder, phi_back,
            a_b, state_set)
117
118     return z

```

プログラム C.4

ソースコード C.4 main 関数 (otmm_decimal_novelty.py)

```

1 # -*- coding: utf-8 -*-
2
3 import numpy as np
4 import pandas as pd
5 import sys
6 import time
7 import os
8 import csv
9 from decimal import Decimal
10
11 import preprocessor_novelty as pp
12 import algorism_decimal as alg
13 import parsing
14
15 """Class Node"""
16 class Node:
17     def __init__(self, no, name, child, child_num):
18         # nodes リストのインデックスが order になっているので order は属性に入れない
19         self.no = no # position of the text glycan data
20         self.name = name # 結合情報を入れる
21
22         self.elder = None # left (immediately elder sibling)
23         self.elder_num = None
24
25         self.younger = None # right (immediately younger sibling)
26         self.younger_num = None

```

```
27
28     self.parent = None
29     self.parent_num = None
30
31     self.child = [] # down
32     self.child.append(child)
33
34     self.child_num = [] # to from OTMM structure
35     self.child_num.append(child_num)
36
37 def add_order(self, order):
38     self.order = order
39
40 def add_elder(self, elder, elder_num):
41     self.elder = elder
42     self.elder_num = elder_num
43
44 def add_younger(self, younger, younger_num):
45     self.younger = younger
46     self.younger_num = younger_num
47
48 def add_parent(self, parent, parent_num):
49     self.parent = parent
50     self.parent_num = parent_num
51
52 def add_child(self, child, child_num):
53     self.child.append(child)
54     self.child_num.append(child_num)
55
56 """前処理を行う関数"""
57 def make_OTMM(row):
58     glycan = row["IUPAC Condensed"]
59     glycan = pp.get_structure(glycan)
60     glycan = pp.create_siblings(glycan)
61     glycan = pp.create_parent(glycan)
62     glycan = pp.set_instance(glycan)
63
64     row["IUPAC Condensed"] = glycan
65
66     return row
67
```

```
68 def main(argv):
69     num_data = input('Number of data. Input the number or "max":')
70     epsilon = input("Epsilon:")
71     n = input("Number of states:")
72
73     """データの前処理"""
74     # データをdf に読み込み
75     # 必ずGoogleColab に glycan_data.csv をアップロードすること
76     df = pd.read_csv("glycan_data.csv") # 木下研から頂いたN-結合型糖鎖の
        データ
77
78     # データが無い行を消去
79     df = df.dropna(subset=['IUPAC Condensed'])
80
81     drop_index = [] #消去する糖鎖のindex を格納する
82
83     # ?が入っている糖鎖は構造が不明な糖鎖
84     for row in df.itertuples(): # itertuples()で行ごとにタプルで取り出す
85         if "?" in row[2]:
86             drop_index.append(row[0])
87
88     # ?が含まれる糖鎖構造を消去
89     df = df.drop(drop_index)
90
91     # index を振りなおす
92     df = df.reset_index(drop=True)
93
94     #1行ごとにdelete_PROT()を呼び出す
95     df = df.apply(pp.delete_PROT, axis=1) # axis=1とすることで1行ずつの
        処理になる
96
97     #1行ごとにseparate_structure()を呼び出す
98     df = df.apply(pp.separate_structure, axis=1) # axis=1とすることで1
        行ずつの処理になる
99
100     """データ量の制限"""
101     if num_data == "max":
102         df = df # データ全部
103     else:
104         df = df.head(int(num_data))
105     print("Number of data:", len(df))
```

```
106
107 # プログラム的にIUPAC Condensed が 2 番目に来ること! (row[2]と記述してい
    る部分があるから)
108 df = df.drop(['Motifs', 'Subsumption Level', 'ChEBI', 'Monoisotopic
    Mass', 'Species'], axis=1)
109
110 df = df.apply(make_OTMM, axis=1)
111
112 df = df.apply(pp.find_mark, axis=1)
113
114 """アルゴリズム"""
115 label_set = set()
116
117 for row in df.itertuples():
118     glycan = row[2]
119     for node in glycan:
120         label_set.add(node.name)
121
122 # pandas1.5以上はlist に
123 label_set = list(label_set)
124 # local 環境ではソートしないと実行するごとに順番が変わってしまう
125 # Google colab では set 型でも順番は一定
126 label_set = sorted(label_set)
127
128 print("Number of labels:", len(label_set))
129
130 n = int(n)
131 state_set = []
132 for i in range(n):
133     state_set.append(i)
134
135 # pandas1.5以上はlist に
136 state_set = list(state_set)
137
138 print("Number of states:", len(state_set))
139 # print(state_set)
140
141 """パラメータのinitialize"""
142 """初期状態確率分布  $\pi$ """
143 np.random.seed(seed=0)
144 pi = np.random.rand(len(state_set))
```



```
145 pi = pi/pi.sum()
146 """対数変換"""
147 pi = np.log(pi)
148 """Decimal 型に"""
149 pi = pi.tolist()
150 for i in range(len(pi)):
151     pi[i] = Decimal(str(float(pi[i])))
152
153
154 状態遷移確率 A
155 parent - node(me) ... A_a
156 """
157 np.random.seed(seed=1)
158 a_a = np.random.rand(len(state_set), len(state_set))
159 a_a = a_a/a_a.sum(axis=1).reshape(-1, 1)
160 """対数変換"""
161 a_a = np.log(a_a)
162 a_a = a_a.tolist()
163 for i in range(len(a_a)):
164     for j in range(len(a_a[i])):
165         a_a[i][j] = Decimal(str(float(a_a[i][j])))
166
167
168 状態遷移確率 A
169 elder - node(me) ...A_b
170 """
171 np.random.seed(seed=2)
172 a_b = np.random.rand(len(state_set), len(state_set))
173 a_b = a_b/a_b.sum(axis=1).reshape(-1, 1)
174 """対数変換"""
175 a_b = np.log(a_b)
176 a_b = a_b.tolist()
177 for i in range(len(a_b)):
178     for j in range(len(a_b[i])):
179         a_b[i][j] = Decimal(str(float(a_b[i][j])))
180
181
182 """ラベル出力確率分布 B"""
183 np.random.seed(seed=3)
184 matrix_B = np.random.rand(len(state_set), len(label_set))
185 matrix_B = matrix_B/matrix_B.sum(axis=1).reshape(-1, 1)
186 """対数変換"""
```

```
186 matrix_B = np.log(matrix_B)
187 matrix_B = matrix_B.tolist()
188 for i in range(len(matrix_B)):
189     for j in range(len(matrix_B[i])):
190         matrix_B[i][j] = Decimal(str(float(matrix_B[i][j])))
191 b = pd.DataFrame(matrix_B, index=state_set, columns=label_set)
192
193 """初期の尤度"""
194 print("\nStart time.perf_counter()")
195 start = time.perf_counter() # time.time()よりtime.perf_counter()の方
    が精度が高い
196 likelihood = alg.calc_likelihood(df, pi, a_a, a_b, b, state_set)
197 L = sum(likelihood) #  $\pi$  (全てを掛け合わせる)なのでsumでよい
198
199 """しきい値"""
200 epsilon = float(epsilon)
201
202 print("\nLearning")
203 new_pi, new_a_a, new_a_b, new_b, L_all = alg.EM(df, pi, a_a, a_b, b
    , state_set, label_set, L, epsilon)
204 end = time.perf_counter()
205 print()
206
207 print("\nEnd time.perf_counter()\n")
208 print("pi updated\n", new_pi)
209 print(" $\alpha$  updated\n", new_a_a)
210 print(" $\beta$  updated\n", new_a_b)
211 print("b updated\n", new_b)
212
213 print("\nThe processing time in learning:", end-start, "seconds")
214
215 # 結果を格納するディレクトリを作成
216 dir_path = 'result'+ '_' + 'novelty'+ '_' +str(len(df))+ '_' +str(epsilon
    )+ '_' +str(len(state_set))+ '_' +str(len(label_set))
217 os.makedirs(dir_path, exist_ok=True)
218 os.chdir(dir_path) # 相対パス
219
220 # output (新規性)
221 time_name = 'time'+ '_' +str(len(df))+ '_' +str(epsilon)+ '_' +str(len(
    state_set))+ '_' +str(len(label_set))+ '.txt'
222 with open(time_name, 'w') as f:
```

```
223     f.write("Time in learning: "+str(end-start)+ " seconds")
224
225     pi_name = 'pi'+ '_' +str(len(df))+ '_' +str(epsilon)+ '_' +str(len(
        state_set))+ '.csv'
226     np.savetxt(pi_name, new_pi, delimiter=',')
227     a_a_name = 'a_a'+ '_' +str(len(df))+ '_' +str(epsilon)+ '_' +str(len(
        state_set))+ '_' +str(len(state_set))+ '.csv'
228     np.savetxt(a_a_name, new_a_a, delimiter=',')
229     a_b_name = 'a_b'+ '_' +str(len(df))+ '_' +str(epsilon)+ '_' +str(len(
        state_set))+ '_' +str(len(state_set))+ '.csv'
230     np.savetxt(a_b_name, new_a_b, delimiter=',')
231     b_name = 'b'+ '_' +str(len(df))+ '_' +str(epsilon)+ '_' +str(len(
        state_set))+ '_' +str(len(label_set))+ '.csv'
232     new_b.to_csv(b_name)
233
234     # Likelihood を出力
235     likelihood_name = 'likelihood'+ '_' +str(len(df))+ '_' +str(epsilon)+ '_'
        +str(len(state_set))+ '_' +str(len(label_set))+ '.csv'
236     with open(likelihood_name, 'wt', encoding='utf-8') as f:
237         # ライター(書き込み者)を作成
238         writer = csv.writer(f)
239         # ライターでデータ(リスト)をファイルに出力
240         writer.writerow(L_all)
241
242     os.chdir("..")
243
244     """
245     Parsing
246     Viterbi アルゴリズムで経路探索
247     """
248     print("\nParsing")
249     # 解析する糖鎖を 1つ選ぶ(今回はcsv ファイルの先頭の糖鎖)
250     glycan = df["IUPAC Condensed"][0] # glycan[0]
251     # Parsing に Decimal が扱えない処理があるので float に変換
252     new_pi = np.float_(new_pi)
253     new_a_a = np.float_(new_a_a)
254     new_a_b = np.float_(new_a_b)
255     new_b = new_b.astype('float64')
256     parsing.parse_glycan(glycan, state_set, new_pi, new_a_a, new_a_b,
        new_b)
257
```

```
258     return 0
259
260 if __name__ == '__main__':
261     sys.exit(main(sys.argv))
```
