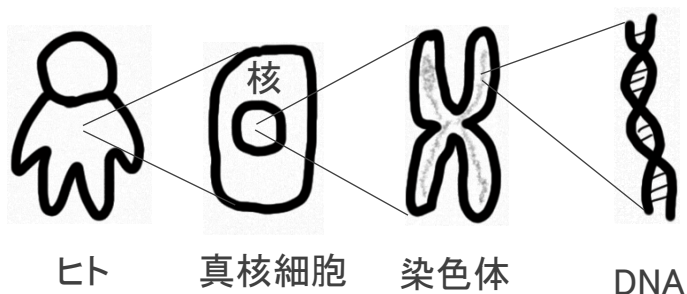


応用レベル(ゲノム解析):テキスト

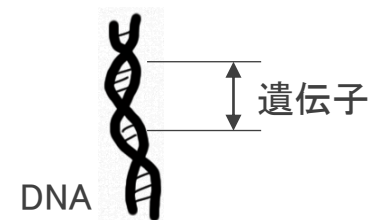
1 DNAと染色体

- ヒトの体は、ある設計図に従って作られている。その設計図にあたるのが「DNA」である。
- ヒトに限らず、生物はみな「細胞」によって形作られる。細胞には「原核細胞」(細菌類、藍藻類など)と、「真核細胞」(ヒトなど)の2種類が存在する。
- DNAは細胞の中にある。原核細胞では細胞内にそのまま、真核細胞では細胞内の核の中に畳まれて、それぞれ格納されている。
- DNAはひものような形をしている。ヒトの場合、ひとつの核の中にあるDNAをすべてつなげると、その全長は2メートルほどにもなる。
- DNAは染色体という塊になっており、ヒトの染色体は23対46本からなる。23本で1セットとなっているのは、両親それぞれから、23本ずつの染色体を受け継ぐためである。

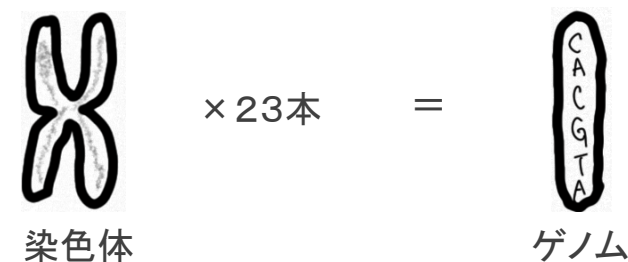


2 遺伝子とゲノム

- DNAは設計情報が蓄えられた化学物質のようなもので、この設計情報によって様々なタンパク質が作られる。
- DNAには異なるタンパク質を作るための暗号のようなものがいくつも並んでいる。この暗号の一部のことを遺伝子と呼ぶ。



- 染色体の1セットのことをゲノムという。
- ヒトの場合は、23本の1セットがゲノムである。特に、ヒトのゲノムのことをヒトゲノムと呼ぶ。
- ゲノムという言葉は、遺伝子 (gene) と染色体 (chromosome) を合成したものである。



3 ゲノム解析とは

- ゲノム解析とは、生物のゲノムがもつ**遺伝情報を総合的に解析すること**である。具体的なプロセスとしては、ゲノムを構成するDNA分子の**塩基配列**を決めることから始まる。
- 塩基とは、DNAの一部であり、遺伝情報に該当する部分を指す。**4種類の塩基**(A: アデニン、G: グアニン、C: シトシン、T: チミン)が対となって複雑に並ぶことで、**設計情報**を表現している。
- 塩基配列データを眺めるだけでは、どこにどのような遺伝子があるかは分からないため、生物種間での塩基配列比較など、いくつかの手法によってデータの解析を行うことになる。
- ゲノム解析における**データ量は膨大**なものとなる。ヒトゲノムではおよそ31億塩基対もある。そのため、コンピュータの使用は不可欠である。



4 新型コロナウイルスとゲノム解析

- 新型コロナウイルスは2019年末ごろから急速に流行し、社会に大きなインパクトをもたらした。ウイルスは**変異**を繰り返すことが知られており、新型コロナウイルスでは、**約2週間で一か所程度**の速度で変異していると考えられている。
- 新型コロナウイルスの変異の追跡のため、日々、ゲノム解析が行われている。国立感染症研究所では解析結果を「懸念される変異株(VOC)」、「注目すべき変異株(VOI)」、「監視下の変異株(VUM)」に分類し発表している。
- ゲノム解析にかかる時間およびコストは、技術開発によって大きく改善されてきている。しかし、解析手法としてより優位と考えられているロングリード(より長い遺伝子の塩基対)を用いた手法では、エラー率や膨大な計算量といった障壁が存在している。そのため、既存手法のさらなる改善や、新たな解析手法の確立が期待されている。

5 参考資料

Newton 2011年11月号(ニュートンプレス)

図解雑学 DNAとRNA(ナツメ社)

DNA・上(ブルーバックス)

[独立行政法人 製品評価技術基盤機構](#)

[厚生労働省 新型コロナウイルス感染症対策推進本部](#)

[国立研究開発法人 日本医療研究開発機構](#)

1 遺伝学とゲノム解析の歴史

- 遺伝学の基礎を築いたのは、**エンドウ**を使って研究を行ったオーストラリアの**メンデル**である。メンデルは1866年「植物雑種の研究」として、遺伝に関する**3つの法則**(**優性の法則**、**分離の法則**、**独立の法則**)を発表したが、この時点では評価がなされなかった。その後、1900年にこの法則が再発見され、ここから近代遺伝学が始まったとされている。
- 1953年、**DNAの二重らせんモデル**(ワトソン、クリックによる)が発表されたことにより、DNAが遺伝子の本体であることが明らかとなった。
- 1988年、国際的な連絡・協力組織である「ヒトゲノム解析機構」が創設され、ヒトゲノム解析の流れが加速した。この計画は、ヒトゲノムの塩基配列—30億すべての塩基配列—を明らかにすることでヒトの生命現象の仕組みに迫り、最終的には人類の福祉(病気の予防や治療など)に結びつけようというものであった。
- ヒトゲノム計画の歴史
1986年 ヒトゲノム計画提言(米)
1991年 ヒトゲノム計画開始(日米英仏EC)
1996年 ヒトゲノムシーケンス※プロジェクト開始(日米英独)
2000年 ドラフトシーケンス終了
2003年 ヒトゲノム全シーケンス完了
2005年 ヒトゲノム全遺伝子の機能解明

※ シーケンス:塩基配列を決定すること。

- 解読された全ヒトゲノムの上製本

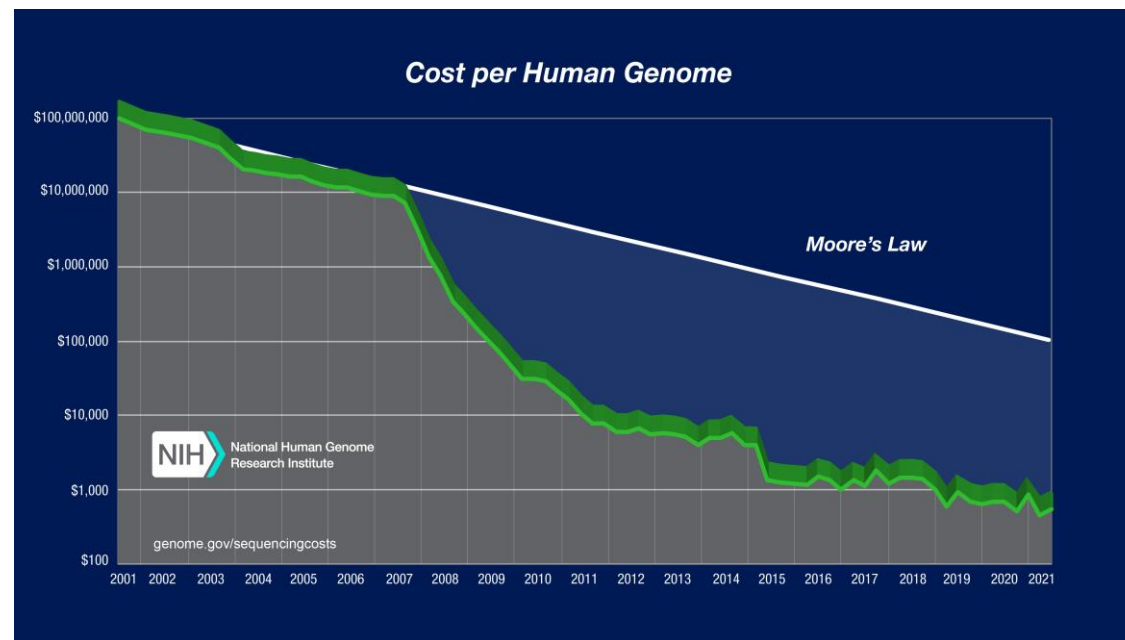


https://commons.wikimedia.org/wiki/File:Wellcome_genome_bookcase.png

2 ゲノム解析のコスト

- ヒトゲノム計画では、2003年に全シーケンスが完了したが、解読に要したコストは30億ドルにもなった。
- 当初は 階層的ショットガン法 と呼ばれる手法を用いて解析を行っていたが、非常に手間と時間のかかる手法であった。そのため、1996年に開始されたヒトゲノムのシーケンスは1999年時点で15%ほどしか完了していなかった。しかし、ここで登場した キャピラリー（毛細管）方式 によって解析効率が向上し、その後の15か月で90%まで解析が進んだ。このエピソードから、ゲノム解析のスピード（およびコスト）は、解析手法によって大きく異なることが分かる。
- 2006年には、一般での個人ゲノム解読が可能となった。しかし、この時に適用されていた サンガーシケンス という手法では、1度にひとつのフラグメント（DNAの断片）しかシーケンスできず、手間がかかることから2000万ドルものコストを要した。
- その後、新たに登場した 次世代シーケンサー（NGS） では何百万ものフラグメントを並列シーケンスできるようになった。コストは2007年に200万ドル、2008年に20万ドル、2010年に1万ドル、2014年に1000ドルという驚異的なスピードでの削減を実現した。この削減スピードは、ムーアの法則を凌駕しており、遺伝学において多大な貢献を果たすことになった。

- ヒトゲノムの解析コストのグラフ
ムーアの法則を量がしたコスト削減を実現していることがわかる。



<https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data>

3 ゲノム解析の課題

- ゲノム解析のプロセスのひとつに **配列アセンブリ** がある。これは、断片化したDNAを元の長い塩基配列に再構築することである。
- 現在の解析手法では、一度に読めるリード（塩基配列）の長さが20から1000残基に留まるため、断片化したDNAのリードを読んだあとに、もとの長い状態に戻すプロセスとして配列アセンブリを行う必要がある。
※ 前項ではゲノム解析のコストに触れたが、このコストには配列アセンブリのプロセスは含まれていない。
- NGS解析では、配列アセンブリを用いると多くの計算時間が必要になるという理由から、代替手段として**リファレンスゲノム**を使用した方式を採用している。リファレンスゲノムを「型」として、断片化したリードをマッピングすることでゲノム解析を行っており、この方法によって時間短縮を実現している。
- しかし、リファレンスゲノムを用いた方式では、個体独自の変異が消えている可能性が指摘されており、必ずしも最適な方式とはいえない。
- 配列アセンブリの手法のひとつに、「**ゲノムアセンブリ**」がある。断片化された大量の塩基配列をつなぎあわせる作業になるため、**ゲノム再編集に膨大な時間がかかる**という課題がある。

4 量子コンピュータへの期待

- 量子コンピュータは膨大な計算処理を並列的に行うことで計算時間を短縮することができると考えられている。
- ゲノムアセンブリにおいては、量子コンピュータを用いることで**ゲノム再編集の時間短縮が実現できる可能性**がある。
もし実現することができれば、より精度の高いゲノム解析をより短時間で行えるため、コストのさらなる削減や、福祉へのさらなる貢献（ex. パンデミックにおけるウイルス解析や、解析に基づく予防の確立など）が期待できる。

5 参考資料

高校の生物が根本からわかる本 細胞・代謝・発生・遺伝編（中経出版）
つくばサイエンス・アカデミー設立総会・記念講演会資料（[1](#), [2](#)）
DNAがわかる本（岩波ジュニア新書、中内光昭）
新版 絵でわかるゲノム・遺伝子・DNA（講談社）
イルミナ株式会社（[1](#)）
Cliffhanger株式会社（[1](#)）
[Wikipedia（配列アセンブリ）](#)

1 概要

- ここでは、ゲノム解析から範囲を拡げ、生命科学等における量子コンピュータの適用事例を見ていく。

2 民間病院への量子コンピュータ提供 (IBM)

- IBMは米国時間2021年3月30日、非営利の学術医療センターである米クリーブランドクリニックと10年間の提携を結び、量子コンピューティングと人工知能 (AI) を応用してヘルスケアとライフサイエンスを研究する「Discovery Accelerator」センターを設立すると発表した。
- IBMは提携の一環として、オハイオ州クリーブランドにある同クリニックの敷地内に、米国の民間部門向けで同社初となる量子コンピューター「IBM Quantum System One」を設置する計画だ。
- クリーブランドクリニックの最高経営責任者 (CEO) を務めるTom Mihaljevic博士によると、量子コンピューターは「医療を変革する」のに役立つという。同氏は発表の中で、「こうした新たなコンピューティング技術は、ライフサイエンス分野における発見を大きく進化させるのに役立つ」と述べた。

- 計画している研究分野には、ゲノム科学、化学薬品と医薬品の開発、単細胞トランスクリプトミクス (遺伝子発現解析の一種)、集団健康管理、臨床応用などがある。研究者らはまた、プライバシーを保護しながらビッグデータを活用し、新型コロナウイルス感染症など世界的な医療危機への対応と患者の治療を向上させることを目指す。
- IBM Quantum System One



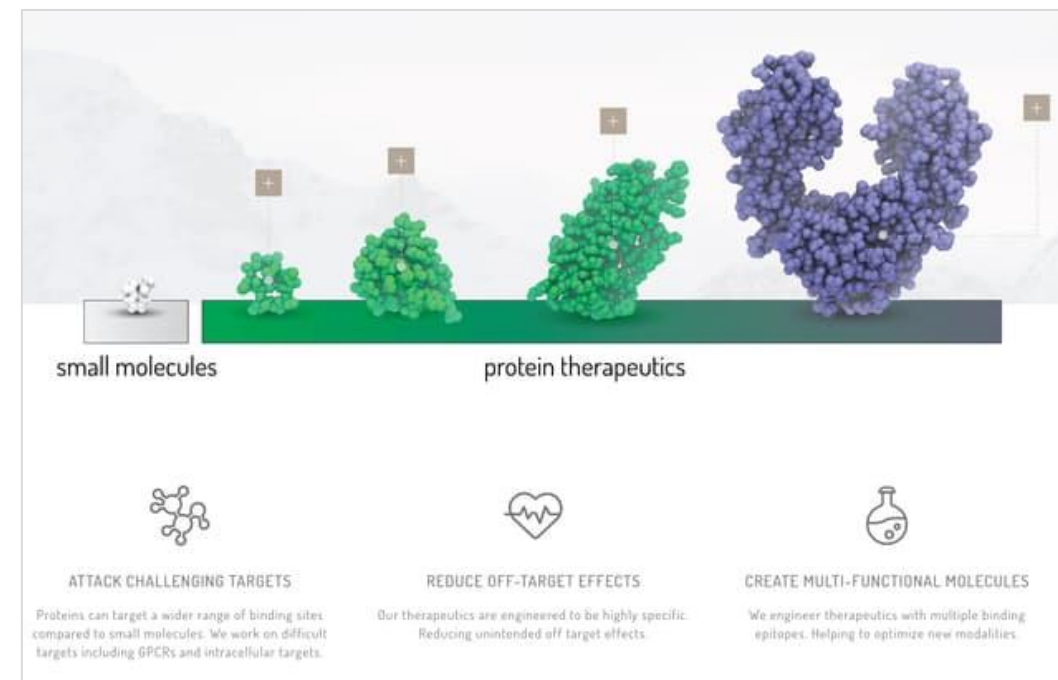
3 量子コンピューターによる医療への影響 (Google)

- 米グーグルは、量子コンピューターを使って化学反応をシミュレーションすることに成功したと発表した。これはまだ開発の初期段階にある量子技術の一里塚となった。
- この相互作用は比較的単純で、現在広く使われているいわゆる「古典コンピューター」でもモデル化できるが、未来の量子コンピューターは従来型のコンピューターよりもずっと正確に複雑な分子の相互作用をシミュレーションできるようになる。
- これにより薬剤候補の効果を予測しやすくなり、創薬の取り組みが加速する可能性がある。

4 量子コンピューターによるたんぱく質構造解析 (プロテインキュア)

- 量子コンピューティングで活発になる創薬の分野は、たんぱく質の構造解析だ。カナダのスタートアップ、プロテインキュア (ProteinQure) は既に現行の量子コンピューターを使ってたんぱく質が体内でどう立体構造になるかを予測している。
- これは従来型のコンピューターでは極めて難しいタスクだが、量子コンピューティングを使って対応すれば有効なたんぱく質に基づく薬をもっと簡単に開発できるようになるだろう。

● たんぱく質に基づく創薬のイメージ



5 参考資料

- IBM、量子コンピューターを初めて民間病院に提供へーゲノム解析や創薬に活用 : <https://japan.cnet.com/article/35168625/>
- 量子コンピューターが変革する9領域 : <https://www.nikkei.com/article/DGXZQODZ2369X0T20C21A3000000/>

1 ゲノム解析のプロセス

- ゲノム解析は、これまで説明したように、ゲノムに含まれる遺伝子の情報を明らかにすること(DNAシーケンシング、塩基配列の読み取り)が重要なプロセスである。そして、その前後にもいくつかのプロセスがある。
- プロセスは大きく、以下のように整理することができる。

DNAサンプルの入手

サンプルの断片化

→ 断片化されたDNA

DNAシーケンシング (塩基配列の読み取り)

→ 断片化されたDNAから読み取った塩基配列データ

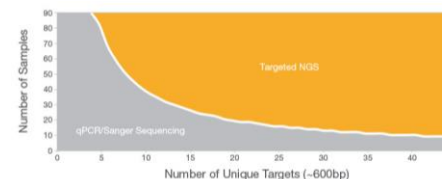
塩基配列データの統合 (ゲノムアセンブリ、マッピング)

→ DNAサンプルの塩基配列決定

データ解析

2 DNAシーケンシング

- ここでは、ゲノム解析のプロセスにおいて特に重要な役割を果たすDNAシーケンシングに注目し、その手法を確認する。
- 1977年、塩基配列の読み取り方法として **マクサム・ギルバート法**、**サンガー法** と呼ばれる2つの手法が発表された。
※各手法の呼び方は様々あり、前者は「化学分解法」、後者は「ジデオキシ法」、「チェーンターミネーター法」、「酵素法」などがある。
- 現在ではサンガー法をもとにした、様々な改良法が用いられている。ヒトゲノム計画ではサンガー法をもとにした「**サーマルサイクル塩基配列決定法**」(サイクルシーケンス法)がよく用いられた。
- 特定個人の全ゲノム解析は、2007年ごろに話題となった。このときの解析手法のひとつが「**ピロシーケンス法**」で、期間2か月強、コスト約1億円での解読を実現した。
- その後、DNAシーケンシングの手法は驚くべき速さで進歩し、「**次世代シーケンサー**」(NGS)の登場により、解析期間・コストが大幅に短縮された。なお、NGSは従来のサンガー法とは原理的に異なる手法である。また、NGSはサンガー法に代わるものではなく、サンプル数やゲノムターゲット数によって使い分けられている。



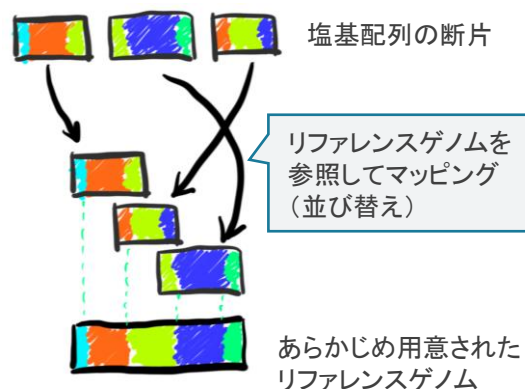
右上の領域がNGS、以外の領域がサンガー法に適するとされている。

出典:illumina

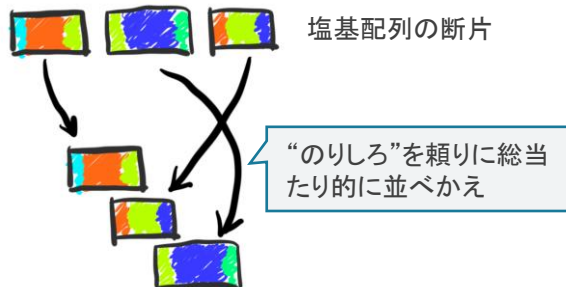
3 塩基配列データの統合

- 断片化された塩基配列データの統合には主に2種類の方法がある。ひとつは**マッピング**、もうひとつは**アセンブリング**である。
- **マッピング**は、データベースで公開されているリファレンス配列（ある種の型）に対して、断片化された塩基配列データをマッピング（型あわせ）することで解析を行う。しかし型を用いる手法という特性上、個体独自の変異が消失してしまう可能性がある。
- **アセンブリング**は、断片化された塩基配列データを、リファレンス配列を用いずに統合する手法である。マッピングのように個体独自の変異は損なわれないとされる一方、大量の塩基配列の統合には膨大な時間（計算量）が必要となり、課題となっている。

マッピング



アセンブリング



4 アセンブリングと量子コンピュータ

- アセンブリングは、前述のとおり膨大な時間（計算量）を要する。具体的には、「大量の塩基配列データをきれいに繋ぎ合わせられる組み合わせ」を、「無数の組み合わせの中から探し出すこと」に膨大な計算処理を要することになる。
- 量子コンピュータは、「組み合わせ最適化問題」に代表されるような、多くの組み合わせの中から最適な解を探し出す計算処理が得意とされている。
ゲノム解析のアセンブリングでは、古典コンピュータでは膨大な計算処理を要する問題であることから、量子コンピュータを適用することで、効率的に最適解（＝きれいに繋がった塩基配列データ）を得られることが期待されている。

5 参考資料

- 生態学者が書いたDNAの本
- [農研機構: ハイテク用語集](#)
- [慶應義塾大学\(榊原康文\): バイオインフォマティクス](#)
- [極東極楽: シーケンスとアノテーション](#)
- [Illumina](#)
- [bioinformatics: マッピング](#)
- [PRTIMES: Cliffhanger](#)

1 サンガーシーケンス

- **サンガーシーケンス**は、1970年代にフレデリック・サンガーが開発した塩基配列の決定法である。
- 限られた数のサンプルやゲノムターゲット(20以下)においてDNAの小さな領域を調べる場合に適した方法とされている。

メリット・デメリット	概要
メリット	少数のターゲット(1～20ターゲット)の場合、迅速でコスト効率が高い
	使いやすいワークフロー
デメリット	低い感度
	低い発見力
	多数のターゲット(20ターゲット以上)の場合、コスト効率があまり高くない
	サンプルインプット要件の増加による低い拡張性

2 次世代シーケンス(NGS)

- **次世代シーケンシング(NGS)**は、数千から数百万ものDNA分子を同時に配列決定することができる強力なシーケンシング手法である。
- 複数個体を同時に配列決定できるなど、高度かつ高速な処理が可能であることから、個の医療、遺伝性疾患および臨床診断学といった分野に変革をもたらしている。

メリット・デメリット	概要
メリット	より高いシーケンス深度が実現するさらに高い感度(1%まで)
	より高い発見力・より高い変異解像度
	同じ量のインプットDNAでより多くのデータが得られる
	より高いサンプルスループット
デメリット	少数のターゲット(1～20ターゲット)の場合、コスト効率が低い
	少数のターゲット(1～20ターゲット)の場合、時間がかかる

3 参考資料

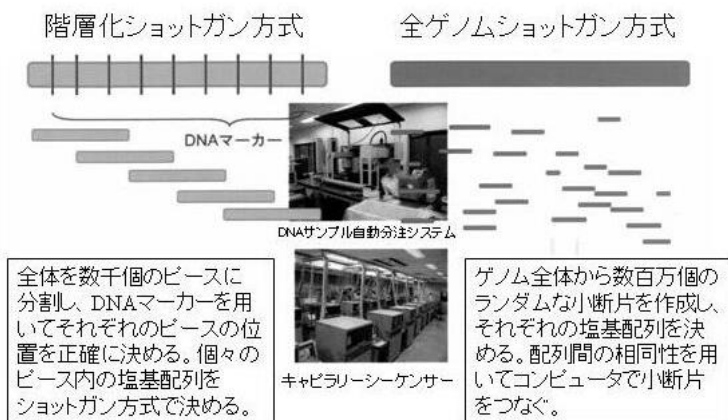
- [illumina](#)
- [コスモ・バイオ株式](#)

1 階層的ショットガン法

- BAC(細菌人工染色体)などのゲノムクローンによる物理地図の作成を行った後、クローン単位でゲノムの塩基配列を決定し、クローンの配列を連結することによりゲノム全体の塩基配列を決定する方法。全ゲノムショットガン法と比較して精度の高い配列データが得られるため、反復配列が豊富な領域のゲノム解析に適している。

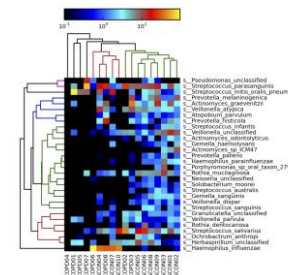
2 全ゲノムショットガン法

- ゲノム配列の決定法として現在広く用いられている手法。染色体の端から端までを正確に切れ目無く1塩基単位で読みとるDNAシーケンサーは今のところ存在していない。そこで、長い染色体をランダムに超音波や水流の剪断力などの物理的な力で裁断し、短くなった大量のDNA断片に対してDNAシーケンサーを用いてその配列を決定する手法。



3 メタショットガン解析

- 次世代シーケンサーの登場により、大量の塩基配列が加速度的に産出されるようになった。そして、環境微生物群集を対象に、新たな大規模メタゲノム解析への次世代シーケンサーの利用が注目を集めている。
- メタゲノム解析の目的としては、「環境中の微生物群集からの有用新機構その探索」や、「腸内・口腔環境や土壌・水質等の環境中に棲む微生物群集の調査」などが挙げられる。



4 参考資料

- [実験医学online 階層的ショットガン法](#)
- [実験医学online全ゲノムショットガン法](#)
- [GeneBay メタショットガン解析](#)
- [GeneBayメタ16S解析](#)

1 量子アニーリングとは

- 量子アニーリングとは、量子の揺らぎを利用することで、最適化問題に特化した問題を高速かつ高精度で解く手法である。
- 1998年に東京工業大学の西森教授らによって提唱され、2011年カナダのD-Wave社が実現した。
- この手法を用いて最適経路探索問題である「巡回セールスマン問題」や、要員の最適化配置を行う「シフトスケジューリング問題」など、さまざまな最適化問題を解くことができる。

2 量子アニーリングの仕組み

- 量子アニーリングでは、超伝導で実現した量子ビットを格子状に配置し、最適化に応じたイジングモデルを量子回路内で実現している。そのエネルギーを基底状態にさせることで、最適値を得る。

3 量子アニーリングの実行ステップ

1. 最適化問題に対するイジング式、またははqubo式と呼ばれる多項式の数式の係数を、行列2次元行列として作成する。
 - ✓ イジング式の場合は1と-1、qubo式の場合は0と1でそれぞれ表現する。
 - ✓ イジング式もしくはqubo式を最小化することと、最適値を求めることは等価である。
2. その行列を量子アニーリングマシン(D-Wave)に投入する。
3. 最小化に近い多項式の変数の値がベクトルとして得られる。

$$H = \sum_i w_i Z_i + \sum_{i < j} w_{ij} Z_i Z_j$$

イジング式は上記の通り。ハミルトニアンHを最小化する形でアニーリングの計算(時間発展)が量子デバイス上で進む。



4 量子アニーリングマシンについて

- 現在、カナダのD-Wave社が5000量子ビットの量子アニーリングマシンを商用にて提供している。
- 量子アニーリングでは必ずしも最適解が求まる訳ではなく、最適解に近い近似解である可能性がある。
- 類似のソリューションとして、既存の手法をFPGAやCMOSなどのハードウェアで行う量子インスパイアード手法を用いたソリューションが、国産の富士通、東芝、日立から提供されている。



<https://dwavejapan.com/system/>

5 参考資料

- [D-Wave Japan](#)
- [D-Wave提供の動画\(Youtube\)](#)

1 QAOAとは

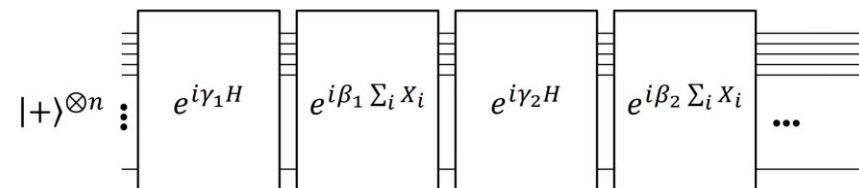
- QAOA (Quantum Approximate Optimization Algorithm) は、量子近似最適化アルゴリズムと呼ばれる、量子ゲート方式による組合せ最適化問題の解を求めるためのアルゴリズムである。
- 『A Quantum Approximate Optimization Algorithm』 (Edward Farhi, Jeffrey Goldstone, Sam Gutmann) にて提唱された。
※ <https://arxiv.org/abs/1411.4028>
- QAOAでは量子アニーリングで利用されるイジングモデルのハミルトニアンを基底をゲート方式にて解くものである

2 QAOAの仕組み

- QAOAはNISQ方式の組合せ最適化アルゴリズムであり、量子化学計算であるVQEや多くの量子機械学習アルゴリズムと同様のVQA (Variational Quantum Algorithm) である。
- QAOAの手順の概要について、まずパラメータ付きの量子回路を初期化する。この量子回路でハミルトニアンの計算を行って結果を得たのち、古典計算でハミルトニアンを小さくする方向にパラメータを更新する。そして、新たなパラメータを与えた量子回路でハミルトニアンを再計算し結果を得る。このような計算プロセスをハミルトニアンが収束するまで繰り返す。

3 QAOA実行のステップ

- QAOAでは以下のステップで組合せ最適化問題を解く。
 1. 離散化問題を基底状態の問題、つまりハミルトニアンに変換する。(古典)
 2. QAOAのansatz、つまり量子回路である $U(\beta, \gamma)$ によってパラメータ β, γ で決まる状態 $\langle \psi(\beta, \gamma) | H | \psi(\beta, \gamma) \rangle$ を用意する。(量子)
 3. 以下をハミルトニアンの値が収束するまで繰り返す。
 - ① $\langle H(\beta, \gamma) \rangle = \langle \psi(\beta, \gamma) | H | \psi(\beta, \gamma) \rangle$ を求める。(量子)
 - ② $\langle H(\beta, \gamma) \rangle$ を小さくする方向に β, γ を更新する。(古典)
- Ansatz回路を以下に示す。ansatz回路が無限に深いものは量子アニーリングと等価となる。



4 参考資料

- A Quantum Approximate Optimization Algorithm (by Edward Farhi, Jeffrey Goldstone, Sam Gutmann)
- 量子コンピュータの応用 – QAOAの仕組みとこれを応用した組合せ最適化問題のプログラミング(blueqat編)

1 アセンブリングと量子コンピュータ

- アセンブリングは、2-1-4のとおり「大量の塩基配列データをきれいに繋ぎ合わせられる組み合わせを」、「無数の組み合わせの中から探し出すこと」に膨大な計算処理を要することになる。
- 量子コンピュータは多くの組み合わせの中から最適な解を探し出す計算処理が得意とされており、ゲノム解析のアセンブリングでは、古典コンピュータでは膨大な計算処理を要する問題であることから、量子コンピュータを適用することで、効率的に最適解(=きれいに繋がった塩基配列データ)を得られることが期待されている。
- 今回は先行研究となる論文からアニーリングによる手法を確認する。本論文ではDe Bruijn graph と Acyclic graphを用いた2つのアプローチについて解説されているが、今回のテキストではゲノムアセンブリの基礎となるDe Bruijn graph について焦点を当てて解説を行う。
- 当項目での説明の流れは以下のとおりである。

手法確認

実装コード
確認コードの参照先
確認

まとめ

- Genome assembly using quantum and quantum-inspired annealing

Cornell University

We gratefully acknowledge support from the Simons Foundation and member institutions.

arXiv > quant-ph > arXiv:2004.06719

Search... All fields Search

Help | Advanced Search

Quantum Physics

[Submitted on 14 Apr 2020 (v1), last revised 24 Jun 2021 (this version, v3)]

Genome assembly using quantum and quantum-inspired annealing

A.S. Boev, A.S. Rakitko, S.R. Usmanov, A.N. Kobzeva, I.V. Popov, V.V. Ilinsky, E.O. Kiktenko, A.K. Fedorov

Recent advances in DNA sequencing open prospects to make whole-genome analysis rapid and reliable, which is promising for various applications including personalized medicine. However, existing techniques for (lit de novo) genome assembly, which is used for the analysis of genomic rearrangements, chromosome phasing, and reconstructing genomes without a reference, require solving tasks of high computational complexity. Here we demonstrate a method for solving genome assembly tasks with the use of quantum and quantum-inspired optimization techniques. Within this method, we present experimental results on genome assembly using quantum annealers both for simulated data and the ϕ X 174 bacteriophage. Our results pave a way for an increase in the efficiency of solving bioinformatics problems with the use of quantum computing and, in particular, quantum annealing. We expect that the new generation of quantum annealing devices would outperform existing techniques for (lit de novo) genome assembly. To the best of our knowledge, this is the first experimental study of de novo genome assembly problems both for real and synthetic data on quantum annealing devices and quantum-inspired techniques.

Comments: 9 pages, 4 figures
Subjects: Quantum Physics (quant-ph)
Cite as: arXiv:2004.06719 [quant-ph]
(or arXiv:2004.06719v3 [quant-ph] for this version)
<https://doi.org/10.48550/arXiv.2004.06719>

Journal reference: Sci. Rep. 11, 13183 (2021)
Related DOI: <https://doi.org/10.1038/s41598-021-88321-5>




Submission history
From: Aleksey Fedorov [view email]
[v1] Tue, 14 Apr 2020 15:41:27 UTC (464 KB)
[v2] Wed, 22 Apr 2020 13:30:29 UTC (464 KB)
[v3] Thu, 24 Jun 2021 08:34:52 UTC (2,933 KB)

Download:
• PDF
• Other formats (license)

Current browse context: quant-ph
< prev | next >
new | recent | 2004

References & Citations
• INSPIRE HEP
• NASA ADS
• Google Scholar
• Semantic Scholar

Export BibTeX Citation

Bookmark
  

Bibliographic Tools Code & Data Related Papers About arXiv Labs

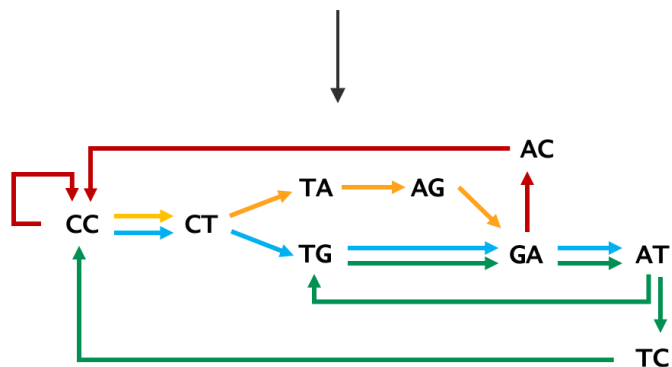
Bibliographic and Citation Tools

<https://arxiv.org/abs/2004.06719>

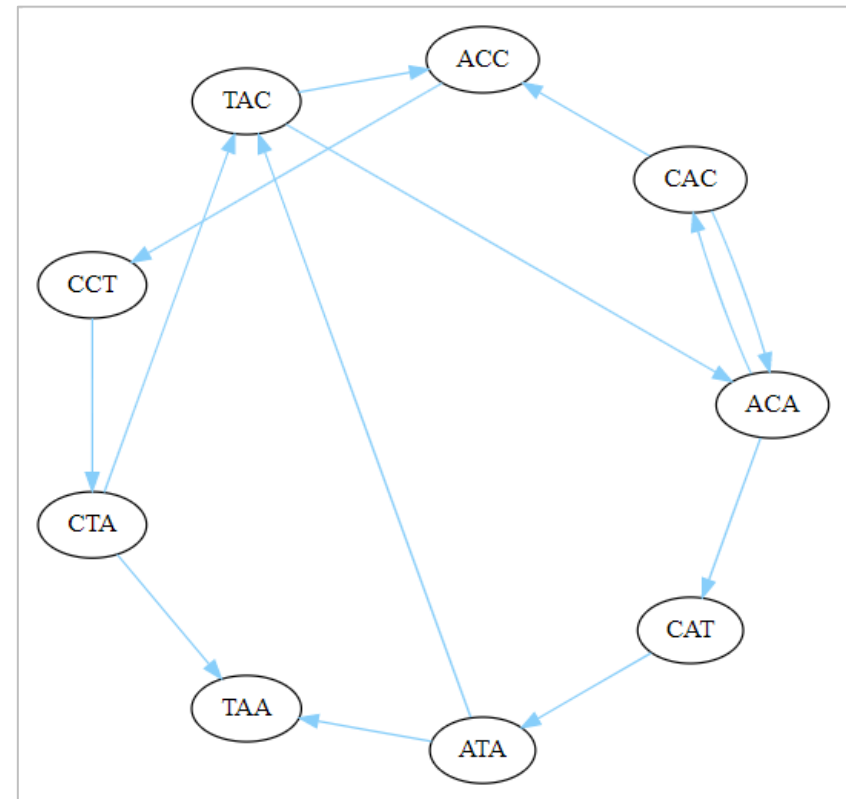
1 De Bruijn graph とは

- De Bruijn graph は生物のDNA配列を決定する際に用いられるデータ構造である。De Bruijn graph は多くのゲノムアセンブラの基礎となっている。
- DNAの断片となるreadを長さ k のDNAの部分配列となる k -mer の k で分割。2-mer だと、read の 1~2, 2~3, 3~4, ... というように、2塩基ずつに分割する。それぞれの分割されたものを頂点とし、隣り合う頂点どうしを辺で結び、グラフを作る。
- 例えば、CCTAGA、ATGATCC、CCTGAT、GACCC の 4 リードの 2-mer で De Bruijn graph を構築すると次の図のようになる。

CCTAGA	CC	→	CT	→	TA	→	AG	→	GA		
CCTGAT	CC	→	CT	→	TG	→	GA	→	AT		
ATGATCC	AT	→	TG	→	GA	→	AT	→	TC	→	CC
GACCC	GA	→	AC	→	CC	→	CC				



- 今回解説するコードの実行例



1 Hamiltonian path De Bruijn graph ソースコードの概要

- 前提
Githubで公開されているライブラリを使用
D-Waveはシュミレータでの実行
- 解説の範囲
解説については、今回のテキスト用に作成したNotebookファイルのみを対象とする。対象およびその他関連ファイルは以下を参照。

教材添付: de_bruijn_graph.ipynb , de_bruijn_graph.pdf

Github: https://github.com/wg-quantum/quantum_genome/ ※対象および関連ファイル

2 Hamiltonian path De Bruijn graph ソースコードの処理の流れ

- 今回解説を行うソースコードの処理の流れは右記の通りである。
- 前処理から始まり、D-Waveを用いない通常のパターンのソースコードを解説し、その後、D-Waveを用いたパターンのソースコードの解説を行う。

メインの解説エリア



右記の処理の流れの縮小図を掲載

説明資料のイメージ

【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の糊付けする個数定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

Graphvizの円形グラフで描画

3 シーケンスの定義

- サンプルデータとなるシーケンスの定義を行う。

```
#ツール読み込み
import dimod

seq = 'CATACACCTAA'
kmer_len, suffix_len = 3, 2
adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)

g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
g.engine = 'circo'
Q = DeBruijnDNA.to_qubo(adj)

g
```

【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の照付けする位置定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

Graphvizの円形グラフで描画。

4 シーケンスの分割単位・糊付けの個数定義

- シーケンスの分割単位と結合時の糊付けする個数を定義している。
- この数字を変更することにより、分割単位・糊付けの個数の定義を変更することができる。

```
#ツール読み込み
import dimod

seq = 'CATACACCTAA'
kmer_len, suffix_len = 3, 2
adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)

g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
g.engine = 'circo'
Q = DeBruijnDNA.to_qubo(adj)

g
```

【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の糊付けする個数定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

Graphvizの円形グラフで描画。

5 De Bruijn graphの作成

- De Bruijn graphを作成する。
- DeBruijnDNAの詳細については以下を参照。

https://github.com/USM-F/quantum_genome/blob/main/DeBruijnDNA.py

```
#ツール読み込み
import dimod

seq = 'CATACACCTAA'
kmer_len, suffix_len = 3, 2
adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)

g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
g.engine = 'circo'
Q = DeBruijnDNA.to_qubo(adj)

g
```

【前処理】

サンプルサイズとなるシーケンスの定義

シーケンスの分割単位と結合時の離付けする距離定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

Graphvizの円形グラフで描画。

6 De Bruijn graphの作成

- De Bruijn graphの作成処理の戻り値の確認。
- 戻り値はグラフを表すための隣接行列、隣接行列に連動したシーケンスのラベル。

Jupyter Notebookで開いた場合のイメージ

```
1  
2 #ツール読み込み  
3 import dimod  
4  
5 seq = 'CATACACCTAA'  
6 kmer_len, suffix_len = 3, 2  
7 adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)  
8  
9 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)  
10 g.engine = 'circo'  
11 Q = DeBruijnDNA.to_qubo(adj)  
12  
13 g
```

戻り値(adj, node_labels)の中身

```
adj  
[[0. 1. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 1. 0. 0. 0. 0. 0. 1.]  
 [0. 0. 0. 1. 0. 1. 0. 0. 0.]  
 [1. 0. 0. 0. 1. 0. 0. 0. 0.]  
 [0. 0. 0. 1. 0. 1. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 1. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 0. 1. 0.]  
 [0. 0. 1. 0. 0. 0. 0. 0. 1.]  
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]  
node_labels  
{0: 'CAT', 1: 'ATA', 2: 'TAC', 3: 'ACA', 4: 'CAC', 5: 'ACC', 6: 'CCT', 7: 'CTA', 8: 'TAA'}
```

【前処理】

サンプルデータとなるシーケンスの定義
シーケンスの分割単位と結合時の照合する位数定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成
グラフ描画
Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理
グラフ描画
Graphvizの円形グラフで描画。

7 グラフ描画

- グラフを描画する。
- DeBruijnDNAの詳細については以下を参照。
https://github.com/USM-F/quantum_genome/blob/main/DeBruijnDNA.py

```
1
2 #ツール読み込み
3 import dimod
4
5 seq = 'CATACACCTAA'
6 kmer_len, suffix_len = 3, 2
7 adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)
8
9 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
10 g.engine = 'circo'
11 Q = DeBruijnDNA.to_qubo(adj)
12
13 g
```

【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の隙付けする整数定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

Graphvizの円形グラフで描画。

8 グラフ描画

- グラフを描画する関数の戻り値の確認。

```
1
2 #ツール読み込み
3 import dimod
4
5 seq = 'CATACACCTAA'
6 kmer_len, suffix_len = 3, 2
7 adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)
8
9 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
10 g.engine = 'circo'
11 Q = DeBruijnDNA.to_qubo(adj)
12
13 g
```

戻り値 (digraph , node) の中身

```
digraph {
  CAT
  CAT -> ATA [label="" color=lightskyblue fontcolor=red fontsize=20]
  ATA
  ATA -> TAC [label="" color=lightskyblue fontcolor=red fontsize=20]
  ATA -> TAA [label="" color=lightskyblue fontcolor=red fontsize=20]
  TAC
  TAC -> ACA [label="" color=lightskyblue fontcolor=red fontsize=20]
  TAC -> ACC [label="" color=lightskyblue fontcolor=red fontsize=20]
  ACA
  ACA -> CAT [label="" color=lightskyblue fontcolor=red fontsize=20]
  ACA -> CAC [label="" color=lightskyblue fontcolor=red fontsize=20]
  CAC
  CAC -> ACA [label="" color=lightskyblue fontcolor=red fontsize=20]
  CAC -> ACC [label="" color=lightskyblue fontcolor=red fontsize=20]
  ACC
  ACC -> CCT [label="" color=lightskyblue fontcolor=red fontsize=20]
  CCT
  CCT -> CTA [label="" color=lightskyblue fontcolor=red fontsize=20]
  CTA
  CTA -> TAC [label="" color=lightskyblue fontcolor=red fontsize=20]
  CTA -> TAA [label="" color=lightskyblue fontcolor=red fontsize=20]
  TAA
}
nodes
{}
```

【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の隣付けする図数定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

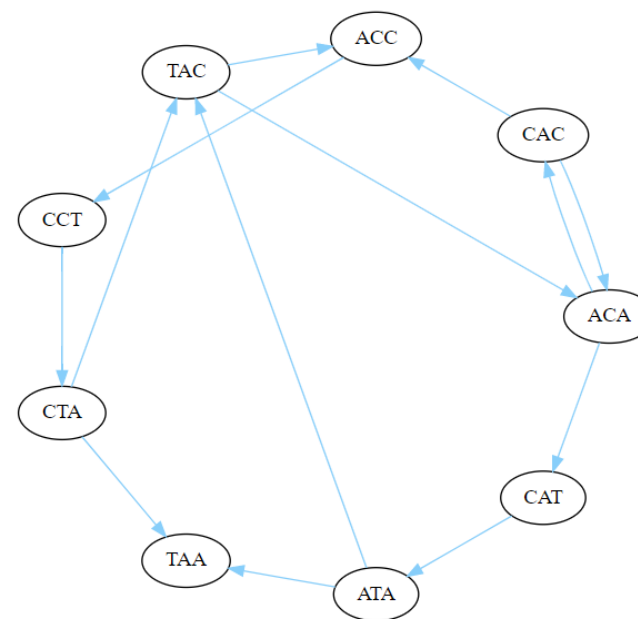
グラフ描画

Graphvizの円形グラフで描画。

9 Graphvizの円形グラフで描画

- オープンソースのグラフ描画ツールのGraphvizで円形グラフを描画。

```
1
2 #ツール読み込み
3 import dimod
4
5 seq = 'CATACACCTAA'
6 kmer_len, suffix_len = 3, 2
7 adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)
8
9 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
10 g.engine = 'circo'
11 Q = DeBruijnDNA.to_qubo(adj)
12
13 g
```



【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の糊付けする接頭辞定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

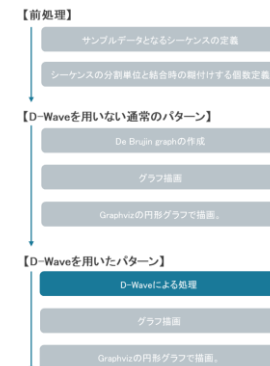
Graphvizの円形グラフで描画。

10 D-Waveによる処理

- D-Waveで処理するために、隣接行列をQUBOに変換する。

```
1
2 #ツール読み込み
3 import dimod
4
5 seq = 'CATACACCTAA'
6 kmer_len, suffix_len = 3, 2
7 adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)
8
9 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
10 g.engine = 'circo'
11 Q = DeBruijnDNA.to_qubo(adj)
12
13 g
```

隣接行列をQUBOに変換

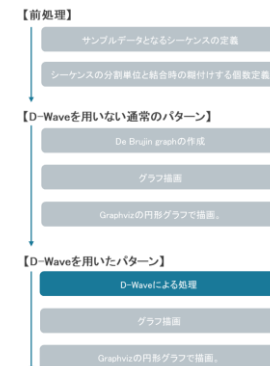


11 D-Waveによる処理

- D-Waveによる処理の実行。

```
target_energy = -np.sqrt(len(Q))
# solution = solve_dwave(Q, API_KEY=API_KEY)
solution = solve_dwave(Q)
spins, energy = [solution[0][i] for i in solution[0].keys()], solution[1]

g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=spins, kmer_len=kmer_len)
g.engine = 'circo'
g
```



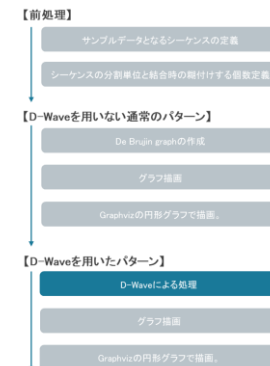
12 D-Waveによる処理

- D-Waveによる処理の戻り値の確認。

```
target_energy = -np.sqrt(len(Q))
# solution = solve_dwave(Q, API_KEY=API_KEY)
solution = solve_dwave(Q)
spins, energy = [solution[0][i] for i in solution[0].keys()], solution[1]

g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=spins, kmer_len=kmer_len)
g.engine = 'circo'
g
```

```
solution[
{0: 1, 1: 1, 2: 0, 3: 0, 4: 1, 5: 1, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0, 12: 1}, -14.0,
{0: 0, 1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1, 8: 0, 9: 1, 10: 1, 11: 0, 12: 1}, -12.0,
{0: 1, 1: 1, 2: 0, 3: 1, 4: 1, 5: 1, 6: 1, 7: 0, 8: 0, 9: 1, 10: 0, 11: 0, 12: 1}, -12.0,
{0: 0, 1: 0, 2: 1, 3: 0, 4: 1, 5: 1, 6: 1, 7: 0, 8: 0, 9: 1, 10: 1, 11: 1, 12: 0}, -12.0,
{0: 1, 1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 1, 7: 0, 8: 0, 9: 1, 10: 1, 11: 1, 12: 1}, -12.0,
{0: 1, 1: 1, 2: 0, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 0, 12: 1}, -12.0,
{0: 0, 1: 1, 2: 0, 3: 1, 4: 0, 5: 1, 6: 1, 7: 1, 8: 0, 9: 1, 10: 1, 11: 0, 12: 0}, -10.0,
~省略~
```



13 グラフ描画

- グラフ描画のためのデータ準備。

```
target_energy = -np.sqrt(len(Q))
# solution = solve_dwave(Q, API_KEY=API_KEY)
solution = solve_dwave(Q)
spins, energy = [solution[0][i] for i in solution[0].keys()], solution[1]

g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=spins, kmer_len=kmer_len)
g.engine = 'circo'
g
```

```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. -2.]
target_energy-3.605551275463989
solution[1]-14.0
solution[{0: 1, 1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1, 8: 0, 9: 1, 10: 1, 11
spins[1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0]
energy-14.0
/usr/local/lib/python3.7/dist-packages/dimod/core/sampler.py:291: SamplerUnknown
```

【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の繰り付ける整数定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

Graphvizの円形グラフで描画。

14 グラフ描画

- グラフ描画。

```
target_energy = -np.sqrt(len(Q))
# solution = solve_dwave(Q, API_KEY=API_KEY)
solution = solve_dwave(Q)
spins, energy = [solution[0][i] for i in solution[0].keys()], solution[1]

g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=spins, kmer_len=kmer_len)
g.engine = 'circo'
g
```

【前処理】

サンプルデータとなるシーケンスの定義

シーケンスの分割単位と結合時の継付けする整数定義

【D-Waveを用いない通常のパターン】

De Bruijn graphの作成

グラフ描画

Graphvizの円形グラフで描画。

【D-Waveを用いたパターン】

D-Waveによる処理

グラフ描画

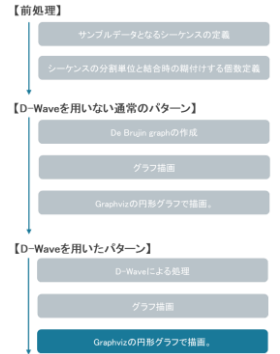
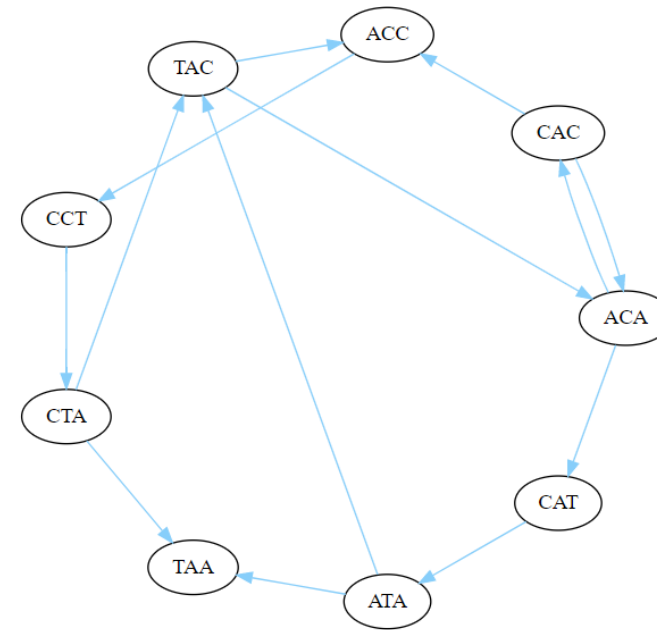
Graphvizの円形グラフで描画。

15 Graphvizの円形グラフで描画

- Graphvizの円形グラフで描画。

```
target_energy = -np.sqrt(len(Q))
# solution = solve_dwave(Q, API_KEY=API_KEY)
solution = solve_dwave(Q)
spins, energy = [solution[0][i] for i in solution[0].keys()], solution[1]

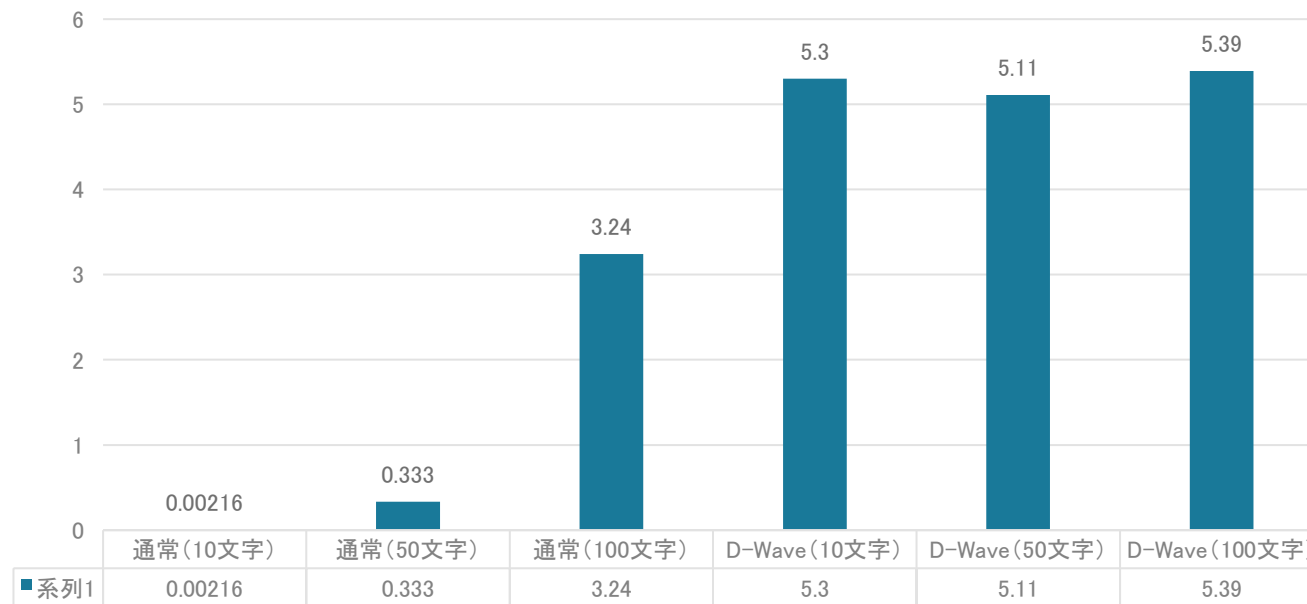
g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=spins, kmer_len=kmer_len)
g.engine = 'circo'
g
```



1 まとめ

- 初めての取り組みとなるゲノム解析だったが、想定していたよりも容易に実装できるということがわかった。
- 今回のDe Bruijn graph 以外にもDirected acyclic graph (DAG) のような実装もある。
- サンプルデータとなるテストシーケンスを10文字、50文字、100文字に分けてD-Waveを用いない通常パターンとD-Waveを用いたパターンで実行時間を測定した。実行時間の測定結果については以下の結果となった。D-Waveを用いないパターンの方が処理速度が速い結果となった。
これはD-Waveを用いたパターンはD-Waveを用いることによるオーバーヘッドの影響で時間がかかっていると考えられる。しかし、D-Waveを用いているため、文字数が増えても実行時間に大きな影響はない結果となった。

実行時間の比較

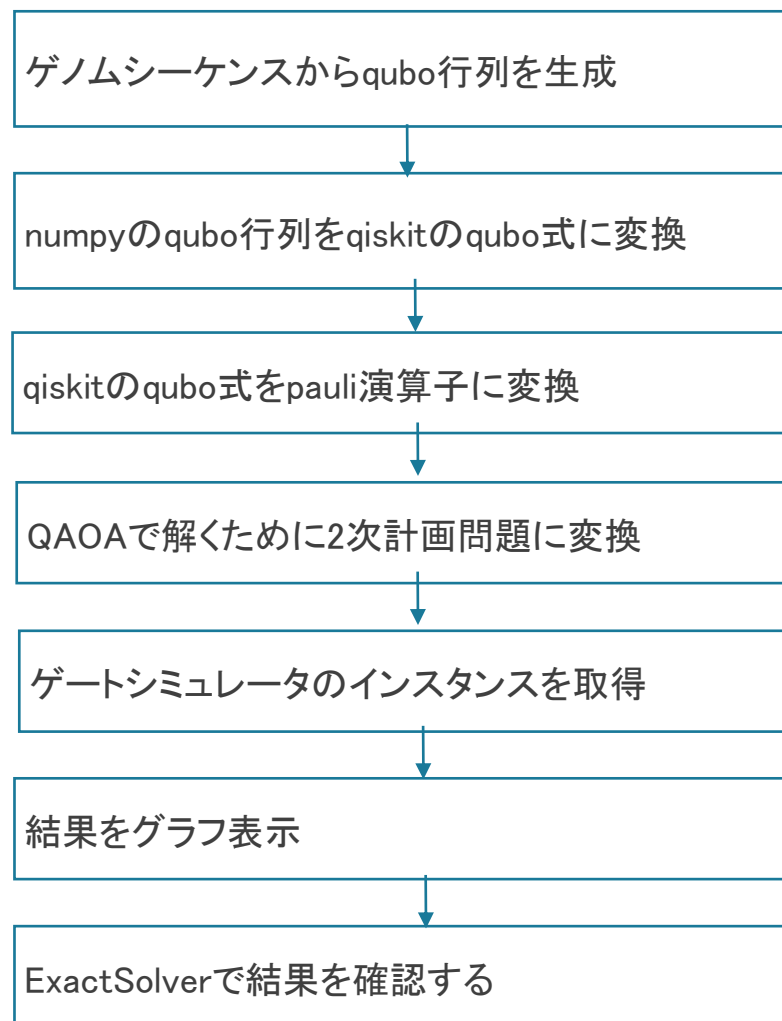


1 QAOAを用いた最適化によるアセンブリング

- 前項ではゲノムアセンブリングをハミルトニアンパス問題に帰結させ、最適化問題を量子アニーリングにて解いた。
- 当項では、ゲート方式の最適化アルゴリズムであるQAOAでこの問題を解いていく。
- Qiskitには最適化を解くためのライブラリとして `qiskit.optimization` が提供されており、QAOAのソルバーが利用できる。
- ハミルトニアンパスではシーケンスの断片、つまりノードの2乗の量子ビット数が必要になる。ノード数が n であれば、 $n \times n$ のqubo行列が必要になるためである。
- 前項の量子アニーリング方式で試行したシーケンスはノード数が9のため、81個の量子ビットが必要ということになる。しかし、ゲート方式のシミュレータで81量子ビットの計算を行うことは現実的でない。
- 上記を踏まえ、当項のQAOAの実験では4ノード、つまり16量子ビットで実行する方針とした。

2 QAOAを用いた処理の流れ

- QAOAでの大まかな処理の流れは以下の通りである。



1 ソースコードの概要

- 解説の範囲

解説については、今回のテキスト用に作成したNotebookファイルのみを対象とする。
対象およびその他関連ファイルは以下を参照。

教材添付: example-QAOA.ipynb, example-QAOA.pdf

Github: https://github.com/wg-quantum/quantum_genome/ ※対象および関連ファイル

2 モジュールのインポート

- 最初に、DeBruijnDNA および 描画ライブラリのインポートを行う。

```
1 # !pip install numpy
2 # !pip install matplotlib
3 # !pip install graphviz
4 # !pip install pymetis
5
6 import numpy as np
7
8 from graphviz import Digraph
9 import networkx as nx
10 import networkx.algorithms as nxa
11 import re
12 import DeBruijnDNA
13 import matplotlib.pyplot as plt
```

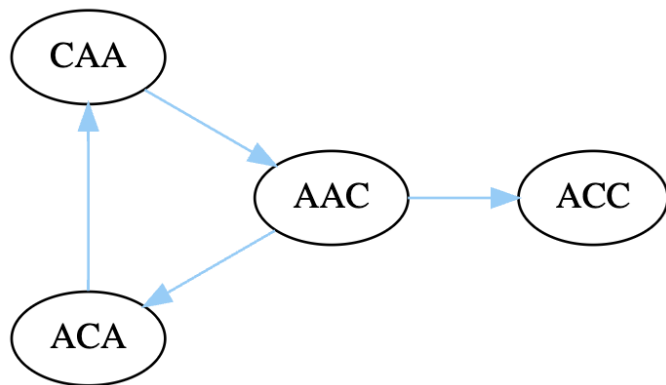
3 ハミルトニアンパスの作成

- 次に、De Bruin graph のハミルトニアンパスを作成する。

De Bruijn graphのハミルトニアンパスを作成

```
1 #seq = 'CATACACCTAA' # ゲノムシーケンス
2 #seq = 'CATACA'
3 seq = 'ACAACC'
4
5 kmer_len, suffix_len = 3, 2 # 文字数は3、サフィックスは2バイト
6 # ゲノムシーケンスから隣接グラフとノードレベルを生成する
7 adj, node_labels = DeBruijnDNA.make_debr(seq, kmer_len=kmer_len, suffix_len=suffix_len)
8 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, [], kmer_len = kmer_len)
9 g.engine = 'circo'
10 Q = DeBruijnDNA.to_qubo(adj) # 隣接グラフからquboを生成する
11 print('solve', len(Q), 'ising problem')
12 g # グラフを描画する
```

solve 16 ising problem



4 ライブラリのインポート

- QAOAで実行するために必要なライブラリをインポートする。

```
1 # !pip install qiskit
2 # !pip install qiskit.optimization
3 # qiskitの必要なライブラリをimportする
4 from qiskit import BasicAer
5 from qiskit.utils import QuantumInstance
6 from qiskit.algorithms import QAOA
7 from qiskit_optimization.algorithms import MinimumEigenOptimizer
8 from qiskit_optimization import QuadraticProgram
```

5 qubo式の生成

- qubo行列からQiskit用のqubo式を生成する。

```
1 # qiskitで必要なqubo式を生成する
2 qubo = QuadraticProgram()
3
4 term_list = ['x_'+str(i) for i in range(len(Q))]
5 for term in term_list:
6     qubo.binary_var(term)
7
8 # numpyのqubo行列から線形項、2次項を取り出して、qiskit用のqubo式を作成する
9
10 def get_terms(Q, term_list):
11     linear = []
12     quadratic = {}
13     for i in range(len(Q)):
14         for j in range(len(Q)):
15             if i==j:
16                 linear.append(Q[i][j])
17             else:
18                 quadratic[(term_list[i],term_list[j])] = Q[i][j]
19     return linear, quadratic
20
21 linear, quad = get_terms(Q, term_list)
22 qubo.minimize(linear=linear, quadratic=quad)
23 print(qubo.prettyprint()) # qubo式を表示する
```

6 変換結果

- 変換した結果の出力を確認。

Problem name:

Minimize

$$\begin{aligned} &2*x_0*x_1 + 2*x_0*x_12 + 2*x_0*x_13 + 2*x_0*x_2 + 2*x_0*x_3 + 2*x_0*x_4 \\ &+ 2*x_0*x_8 + 2*x_0*x_9 + 2*x_1*x_10 + 2*x_1*x_12 + 2*x_1*x_13 + 2*x_1*x_14 \\ &+ 2*x_1*x_2 + 2*x_1*x_3 + 2*x_1*x_4 + 2*x_1*x_5 + 2*x_1*x_9 + 2*x_10*x_11 \\ &+ 2*x_10*x_13 + 2*x_10*x_14 + 2*x_11*x_14 + 2*x_11*x_15 + 2*x_12*x_13 \\ &+ 2*x_12*x_14 + 2*x_12*x_15 + 2*x_13*x_14 + 2*x_13*x_15 + 2*x_14*x_15 \\ &+ 2*x_2*x_10 + 2*x_2*x_11 + 2*x_2*x_13 + 2*x_2*x_14 + 2*x_2*x_15 + 2*x_2*x_3 \\ &+ 2*x_2*x_5 + 2*x_2*x_6 + 2*x_3*x_11 + 2*x_3*x_14 + 2*x_3*x_15 + 2*x_3*x_6 \\ &+ 2*x_3*x_7 + 2*x_4*x_12 + 2*x_4*x_13 + 2*x_4*x_5 + 2*x_4*x_6 + 2*x_4*x_7 \\ &+ 2*x_4*x_8 + 2*x_5*x_12 + 2*x_5*x_13 + 2*x_5*x_14 + 2*x_5*x_6 + 2*x_5*x_7 \\ &+ 2*x_5*x_8 + 2*x_5*x_9 + 2*x_6*x_10 + 2*x_6*x_13 + 2*x_6*x_14 + 2*x_6*x_15 \\ &+ 2*x_6*x_7 + 2*x_6*x_9 + 2*x_7*x_10 + 2*x_7*x_11 + 2*x_7*x_14 + 2*x_7*x_15 \\ &+ 2*x_8*x_10 + 2*x_8*x_11 + 2*x_8*x_12 + 2*x_8*x_9 + 2*x_9*x_10 + 2*x_9*x_11 \\ &+ 2*x_9*x_12 + 2*x_9*x_13 - x_0 - x_1 - x_10 - x_11 - x_12 - x_13 - x_14 \\ &- x_15 - x_2 - x_3 - x_4 - x_5 - x_6 - x_7 - x_8 - x_9 \end{aligned}$$

Subject to

No constraints

Binary variables (16)

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_10 x_11 x_12 x_13 x_14 x_15

7 Pauli演算子への変換

- qubo式をPauli演算子に変換する。

```
1 # qubo式をパウリ演算子からなるイジングモデルのハミルトニアンに変換する
2 op, offset = qubo.to_ising()
3 print("offset: {}".format(offset))
4 print("pauli operator:")
5 print(op) # パウリ演算子からなるハミルトニアンを表示する
```

```
offset: 28.0
pauli operator:
-3.5 * IIIIIIIIIIIIIIIIZ
- 4.5 * IIIIIIIIIIIIIIIIZI
- 4.5 * IIIIIIIIIIIIIIIIZII
- 3.5 * IIIIIIIIIIIIIIIIZIII
- 3.5 * IIIIIIIIIIIIIIIIZIIII
- 4.5 * IIIIIIIIIIIIIIZIIIII
- 4.5 * IIIIIIIIIIIIZIIIIII
- 3.5 * IIIIIIIIIIZIIIIIII
- 3.0 * IIIIIIIIZIIIIIIII
- 4.0 * IIIIIIIIZIIIIIIIII
- 4.0 * IIIIIIZIIIIIIIIIII
```

(以降続く)

8 二次計画問題への変換

- QAOAで解くため、さらに二次計画問題に変換する。

```
1 # イジングモデルのハミルトニアンを2次計画問題に変換
2 qp = QuadraticProgram()
3 qp.from_ising(op, offset, linear=True)
4 print(qp.prettyprint())
```

Problem name:

Minimize

$$\begin{aligned} & 2*x_0*x_1 + 2*x_0*x_{12} + 2*x_0*x_{13} + 2*x_0*x_2 + 2*x_0*x_3 + 2*x_0*x_4 + 2*x_0*x_8 \\ & + 2*x_0*x_9 + 2*x_1*x_{10} + 2*x_1*x_{12} + 2*x_1*x_{13} + 2*x_1*x_{14} + 2*x_1*x_2 + 2*x_1*x_3 \\ & + 2*x_1*x_4 + 2*x_1*x_5 + 2*x_1*x_9 + 2*x_{10}*x_{11} + 2*x_{10}*x_{13} + 2*x_{10}*x_{14} + 2*x_{11}*x_{14} \\ & + 2*x_{11}*x_{15} + 2*x_{12}*x_{13} + 2*x_{12}*x_{14} + 2*x_{12}*x_{15} + 2*x_{13}*x_{14} + 2*x_{13}*x_{15} \\ & + 2*x_{14}*x_{15} + 2*x_2*x_{10} + 2*x_2*x_{11} + 2*x_2*x_{13} + 2*x_2*x_{14} + 2*x_2*x_{15} + 2*x_2*x_3 \\ & + 2*x_2*x_5 + 2*x_2*x_6 + 2*x_3*x_{11} + 2*x_3*x_{14} + 2*x_3*x_{15} + 2*x_3*x_6 + 2*x_3*x_7 \\ & + 2*x_4*x_{12} + 2*x_4*x_{13} + 2*x_4*x_5 + 2*x_4*x_6 + 2*x_4*x_7 + 2*x_4*x_8 + 2*x_5*x_{12} \\ & + 2*x_5*x_{13} + 2*x_5*x_{14} + 2*x_5*x_6 + 2*x_5*x_7 + 2*x_5*x_8 + 2*x_5*x_9 + 2*x_6*x_{10} \\ & + 2*x_6*x_{13} + 2*x_6*x_{14} + 2*x_6*x_{15} + 2*x_6*x_7 + 2*x_6*x_9 + 2*x_7*x_{10} + 2*x_7*x_{11} \\ & + 2*x_7*x_{14} + 2*x_7*x_{15} + 2*x_8*x_{10} + 2*x_8*x_{11} + 2*x_8*x_{12} + 2*x_8*x_9 + 2*x_9*x_{10} \\ & + 2*x_9*x_{11} + 2*x_9*x_{12} + 2*x_9*x_{13} - x_0 - x_1 - x_{10} - x_{11} - x_{12} - x_{13} - x_{14} - x_{15} \\ & - x_2 - x_3 - x_4 - x_5 - x_6 - x_7 - x_8 - x_9 \end{aligned}$$

Subject to

No constraints

Binary variables (16)

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}

9 シミュレータ生成

- Qiskitのローカルシミュレータを生成する。

```
1 # qiskitのローカルシミュレータのインスタンスを生成する
2 quantum_instance = QuantumInstance(BasicAer.get_backend('qasm_simulator'))
```

10 QAOAインスタンス生成

- QiskitのQAOAインスタンスを生成する。

```
1 # qaoaのインスタンスを生成する
2 qaoa_mes = QAOA(quantum_instance=quantum_instance, initial_point=[0.0, 0.0])
3 qaoa = MinimumEigenOptimizer(qaoa_mes) # using QAOA
```

11 実行・結果表示

- Qiskitのシミュレータで実行し、結果を表示する。

```
1 # QAOAを実行して結果を表示する
2 qaoa_result = qaoa.solve(qubo)
3 print(qaoa_result.prettyprint())
```

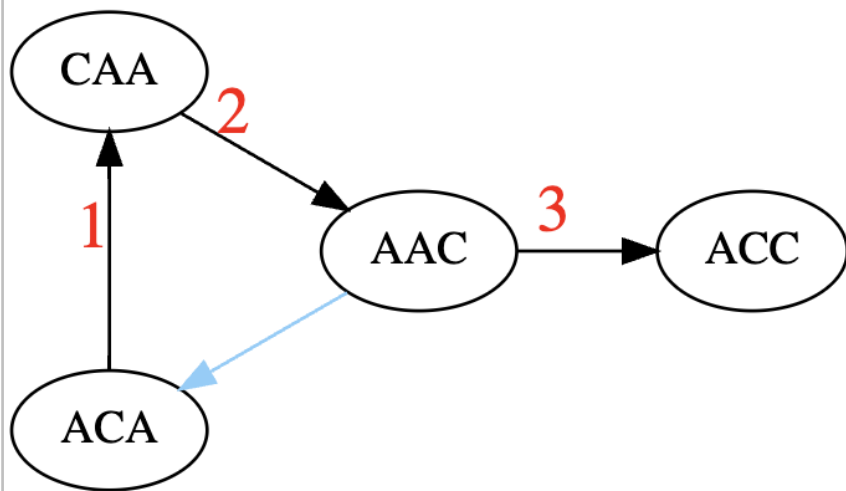


```
objective function value: -3.0
variable values: x_0=0.0, x_1=1.0, x_2=0.0, x_3=0.0, x_4=0.0, x_5=0.0, x_6=1.0, x_7=0.0, x_8=1.0, x_9=0.0, x_10=0.0, x_11=0.0, x_12=0.0, x_13=0.0, x_14=0.0, x_15=0.0
status: SUCCESS
```

12 結果グラフ表示

- 結果をグラフ表示する。

```
1 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=ret.tolist(), kmer_len=kmer_len)
2 g.engine = 'circo'
3 g
```



13 結果確認

- ExactSolver(総当たり)で結果を確認する。

exactsolverで結果を確認する

```
1 # ExactSolverのインスタンスを生成する
2 exact_mes = NumPyMinimumEigensolver()
3 exact = MinimumEigenOptimizer(exact_mes) # using the exact classical numpy minimum eigen solver
```

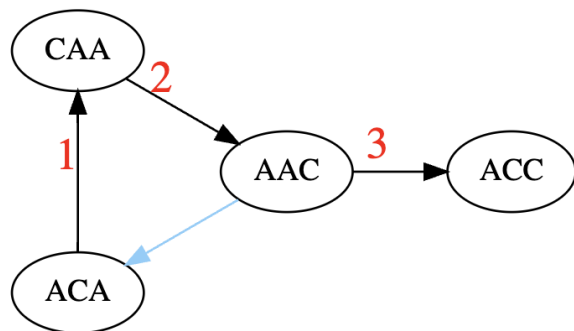
```
1 # ExactSolverを実行して結果を表示する
2 exact_result = exact.solve(qubo)
3 print(exact_result.prettyprint())
```

objective function value: -4.0

variable values: x_0=1.0, x_1=0.0, x_2=0.0, x_3=0.0, x_4=0.0, x_5=1.0, x_6=0.0, x_7=0.0, x_8=0.0, x_9=0.0, x_10=1.0, x_11=0.0, x_12=0.0, x_13=0.0, x_14=0.0, x_15=1.0

status: SUCCESS

```
1 # 隣接グラフを描画する
2 g, nodes = DeBruijnDNA.draw_graph(adj, node_labels, path_spins=exact_result.x.tolist(), kmer_len=kmer_len)
3 g.engine = 'circo'
4 g
```



1 まとめ

- 今回、アニーリング方式で行った最適化をQAOAでも行った。
- ハミルトニアンパス問題においては、シーケンスの文字数 n に対し、糊付けは $(n-2)$ となるため $(n-2)^2$ の量子ビットが必要となる。
- 今回、シーケンスを6文字としたため、 $4^2=16$ の量子ビット数が必要となる。
実行時間は4～5分程度であった。

```
# QAOAを実行して結果を表示する
```

```
%time qaoa_result = qaoa.solve(qubo)
print(qaoa_result.prettyprint())
```

```
CPU times: user 5min 41s, sys: 747 ms, total: 5min 42s
```

```
Wall time: 5min 42s
```

```
objective function value: 0.0
```

```
variable values: x_0=0.0, x_1=0.0, x_2=0.0, x_3=0.0, x_4=0.0, x_5=0.0, x_6=0.0, x_7=1.0, x_8=0.0, x_9=0.0, x_10=0.0, x_11=0.0, x_12=0.0, x_13=0.0, x_14=0.0, x_15=1.0
```

```
status: SUCCESS
```

- シーケンスを7文字にすると、 $5^2=25$ の量子ビット数が必要となる。
このケースでは、qasm_simulator でエラーとなり、実行することができなかった。

```
# QAOAを実行して結果を表示する
```

```
%time qaoa_result = qaoa.solve(qubo)
print(qaoa_result.prettyprint())
```

```
FAILURE: Can not get job id, Resubmit the qobj to get job id. Terra job error: 'Number of qubits 25 is greater than maximum (24) for "qasm_simulator".'
```

- 今後、より大きなシーケンスで実行できる方法を調査する予定である。
- 少なくともゲート方式では、実用的な問題サイズでの最適化計算を行うことが困難であることが分かった。

本資料の著作権は、日本アイ・ビー・エム株式会社（IBM Corporationを含み、以下、IBMといいます。）に帰属します。

ワークショップ、セッション、および資料は、IBMまたはセッション発表者によって準備され、それぞれ独自の見解を反映したものです。それらは情報提供の目的のみで提供されており、いかなる参加者に対しても法律的またはその他の指導や助言を意図したものではなく、またそのような結果を生むものでもありません。本資料に含まれている情報については、完全性と正確性を期するよう努力しましたが、「現状のまま」提供され、明示または暗示にかかわらずいかなる保証も伴わないものとします。本資料またはその他の資料の使用によって、あるいはその他の関連によって、いかなる損害が生じた場合も、IBMまたはセッション発表者は責任を負わないものとします。本資料に含まれている内容は、IBMまたはそのサプライヤーやライセンス交付者からいかなる保証または表明を引きだすことを意図したものでも、IBMソフトウェアの使用を規定する適用ライセンス契約の条項を変更することを意図したものでもなく、またそのような結果を生むものでもありません。

本資料でIBM製品、プログラム、またはサービスに言及していても、IBMが営業活動を行っているすべての国でそれらが使用可能であることを暗示するものではありません。本資料で言及している製品リリース日付や製品機能は、市場機会またはその他の要因に基づいてIBM独自の決定権をもっていつでも変更できるものとし、いかなる方法においても将来の製品または機能が使用可能になると確約することを意図したものではありません。本資料に含まれている内容は、参加者が開始する活動によって特定の販売、売上高の向上、またはその他の結果が生じると述べる、または暗示することを意図したものでも、またそのような結果を生むものでもありません。パフォーマンスは、管理された環境において標準的なIBMベンチマークを使用した測定と予測に基づいています。ユーザーが経験する実際のスループットやパフォーマンスは、ユーザーのジョブ・ストリームにおけるマルチプログラミングの量、入出力構成、ストレージ構成、および処理されるワークロードなどの考慮事項を含む、数多くの要因に応じて変化します。したがって、個々のユーザーがここで述べられているものと同様の結果を得られると確約するものではありません。

記述されているすべてのお客様事例は、それらのお客様がどのようにIBM製品を使用したか、またそれらのお客様が達成した結果の実例として示されたものです。実際の環境コストおよびパフォーマンス特性は、お客様ごとに異なる場合があります。

IBM、IBM ロゴは、米国やその他の国におけるInternational Business Machines Corporationの商標または登録商標です。他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。現時点でのIBMの商標リストについては、ibm.com/trademarkをご覧ください。