汎用組み合わせ最適化アルゴリズム

内容

汎用組み合わせ最適化アルゴリズムの狙い

人工知能が説明変数（特徴量）から、目的変数を推測する場合を考える。ある目的変数が、ある一つの説明変数のみの関数とは限らず、複数の説明変数が組み合わさって影響している。すなわち、影響する説明変数の組み合わせを選ぶ、最適化問題と解釈できる。汎用 AI においては、計算開始から任意の時間が経過した時点で、最適でなくてもよいので、できるだけ良い回答をしなければならない。そのため、深さ優先探索や、幅優先探索はできない。ランダムな組み合わせを試し続けて、現時点までの最適解を出すことなら出来るが、汎用的だが効率が良いとはいえない。組み合わせ最適化問題では、動的計画法やメタヒューリスティクスのような効率化する方法があるが、使用できる場合が限定されていたり、問題によって向き・不向きがあったりする。そこで、どんな問題にでも汎用的に対応でき、任意の計算時間で回答でき、総当たりやランダムな手法より効率の良いアルゴリズムを考案した。問題を解くのに必要な情報が与えられており、演繹的に解を求められる場合だけでなく、必要な情報が欠けており、帰納的にしか予測できない場合についても同じアルゴリズムで対応可能とする。また、P 問題か、NP 問題かといった区別をする必要はなく、多項式時間で解けるものは多項式時間で解くことができる。また、多項式時間で解けないものでも、徐々に正解へ近づき、正解があれば必ず正解へ到達する。

組み合わせ最適化アルゴリズムの原理（演繹的）

まずは、既存の一般的な組み合わせ最適化アルゴリズムがどうして有効なのか、その原理を考える。ヒューリスティックではない方法としては、分岐限定法や動的計画法、バックトラックなどがあるが、これは途中までの計算で、残りは計算しなくても分かるため省略している。また、動的計画法では、計算済みの部分は、再度計算する必要がないとして省略している。これらの方法は、与えられた問題をもとに、演繹的に計算を省いているといえる。

不要な計算を省略するためには、問題を分割する必要がある。例えば、充足可能性問題(SAT)であれば、項毎に分けられる。ある項が制約を満たさなければ、他の項は検査する必要はない。また、回答の選択肢についても分割することができる。ナップザック問題で、品が 8 つならば、回答の選択肢は 256 択だが、それぞれの品を入れるかどうかの 2 択×8 品という積の形で表される。ある品を入れると、サイズオーバーしてしまうと分ければ、他の品を入れるかどうかを検査する必要はない。どのように、問題・回答を分割するかは、プログラマーが問題をどう解釈するかに依存する。汎用的であるためには、自動的に最適な解釈をする仕組みが必要である。例えば、問題が論理式ならば、和積標準形に変換してしまえば一律な解釈となる。また、選択肢は 2 の累乗の形に無理やり分割することができる。しかし、例えば、グー、チョキ、パーを選ぶ 3 択を、「グー or グー以外」、「チョキ or パー」というように、分割してしまうのは合理的ではなく、制約条件を考慮して、分割しなければならない。

組み合わせ最適化アルゴリズムの原理（ヒューリスティック）

一方、ヒューリスティックな方法として、山登り法やタブーサーチ、焼きなまし法のような局所探索法があるが、これは不要な計算を省くのではなく、良さそうな所を優先的に探索するものである。悪い所の近辺よりも、良い所の近辺の方が、さらに良い所があるだろうという経験に基づいている。ある所から僅かに離れた所は、そこより僅かに高いか低いかだろうと推測できる。距離が 0 に近づくほど、高低差も 0 に近づくからである。y=f(x)の x の差が小さいほど、y の差も小さいだろうという帰納的推論の基本的な原理に基づいているといえる。その推論は必ずしも正しいわけではないために、局所解に陥ってしまうため、それを回避する方法の違いで、アルゴリズムにバリエーションが存在する。また、遺伝的アルゴリズムもまたヒューリスティックな方法であり、同様の原理に基づいている。高い所と高い所の中点辺りも平均的には同じくらいの高さで、より高い所もあるだろうと推測している。汎用的であるためには、プログラマーが決める必要があるハイパーパラメータがあってはならないが、局所探索法では、どういったものを近傍のものとするかを指定しなければならない。説明変数 A[m]、説明変数 B[kg]があったとして、1m 離れたものと、1kg 離れたものも、同様に近傍とするといったような指定が必要になる。遺伝的アルゴリズムなら、2 点を間を取るだけなので、そういったハイパーパラメータは不要である。ある 1 点の近傍を調べたいなら、2 点の間のなかでも、片側に寄った所を調べればよい。もし、既知のデータが 1 点しかないなら、近傍を調べたり、突然変異したりしようにも、説明変数のスケールが分からないため、1cm 動かせばいいのか、1km 動かせばいいのかわからないだろう。比較的良い複数のデータの混ぜ具合を変えることで、近傍へ移動したり、突然変異したりすることができる。局所探索法は、突然変異だけを行う遺伝的アルゴリズムとも捉えることができる。経験的（ヒューリスティック）なアルゴリズムはさまざまなものがあるが、全く同じ原理に基づいている。原理以外の部分のバリエーションにより、得意・不得意な問題が発生している。汎用的にするためには、原理以外の処理は固定せず、状況に応じて最適な処理をしなければならない。

近傍 or ランダムの最適な比率

局所解に陥らないために、局所探索では、ランダムな複数箇所から開始したり、遺伝的アルゴリズムでは、突然変異したりする。近傍に対して、ランダムな探索をどの程度加えるかは、プログラマーの経験で決めている。汎用的なアルゴリズムにするためには、その度合いを、ケースバイケースで自動的に最適値を選ばなければならない。近傍だけの探索ではいけない理由を考えよう。逐次、その時点での、最も正解に近い（期待値が最大の）手を 1 つ選ぶのを繰り返すという戦略は、一見正しいように思えるが、例外がある。例として二つの箱のどちらかからクジを引く場合を考えよう。何度かのくじ引きの結果、箱 A は 90%が当たり、箱 B は 10%が当たりだった。次にクジを引くなら、箱 A(期待値 0.9)を選ぶ方が良いだろう。もしそれで外れてしまっても、その次は、箱 A の期待値は 0.9 より下がるが、箱 B の期待値 0.1 より大きい限りは、箱 A からクジを引き続けるのが最良の戦略と思える。しかし、実際には箱 B の当たりは 95%でありながら、初めに低確率のハズレを連発した可能性もある。目的が、クジを無限回

引いたときの利益だとすると、時間が掛かってでも最も当たりの多い箱を突き止める必要がある。最適解への到達が目的とすると、期待値が最高の手を選び続けるのは最適な戦略ではない。では、どのくらい他の手を選ぶのが最適か考えよう。例えば、ある時点で、箱 A が最も当たりクジの多い確率が 90%、箱 B は 10%の場合を考える。乱数（0〜1）を見て、0.1 以上なら箱 A、0.1 未満なら箱 B を選ぶ戦略はどうだろうか。しかし、クジが何回引くことができるかは不明であるとして、常にできるだけ多く当たりを引きたいとしよう。仮に乱数 0.1 未満が出たからといって、次がラストのクジかもしれないのに、箱 B を選ぶのは愚策である。次が最後なら、期待値の高い箱 A を選ぶべきである。では、10 枚クジを引ける場合を考えよう。9 枚は箱 A、1 枚は箱 B から選べば、実際には箱 B の方が、当たりが多いという稀なケースにも対応できるだろう。ここで、1 枚ずつ乱数を使って、A か B か選ぶのは愚策である。乱数が偏り、期待値の低い B ばかりを選んでしまう可能性がある。しかし、何枚まとめてクジが引けるか分かるとは限らないので、1 枚ずつクジを引く場合の戦略を考えよう。引きたいクジの分布、引いた（加えてこれから引く）クジの分布を比較する。引きたいクジの分布は箱 A が 90%、箱 B が 10%だとする。1 回目にクジを A または B から引いた場合の分布を考えると、A が 100%または B が 100%となる。引きたい分布と引いた分布が、出来るだけ近くなるように引くクジを選べばよい。1 枚目は A を選ぶことになる。これを繰り返し 10 枚選んだ後は、A を 9 枚、B を 1 枚引く結果となる。乱数を使う必要なく、逐次、最適な選択ができた。乱数を使う必要があるのは、同値の場合のみである。

一般的な場合として、高さが最高の地点を探索する場合を考えよう。低い所の近傍よりも、高い所の近傍に、より高い所があるとヒューリスティックに予想できるが、それはどのくらいの確率だろうか。低い所をどのくらい探索するべきかを決めるのに、その確率分布が必要である。ある程度探索が進めば、統計的に求めることも出来るが、それでも探索が不十分な領域は、うまく予測できないだろう。極端な場合として、高さが既知の点が 1 点しかない場合、他の場所がそこより高いとも低いともいえない。遺伝的アルゴリズムで交叉する相手もいない。そこで、「不明」という値を導入しよう。既知の点は「不明」ではない。既知でない点は「不明」ではあるが、近傍の既知の点から推測は可能である。推測値は確率分布で表され、既知の点から離れるほど、「不明」の確率が上がる。距離がゼロなら、それは既知の点であるため、「不明」は 0%、無限に離れていれば全く推測不明なため、「不明」が 100%である。探索を始めた直後で、既知の点がない場合は、全て「不明」である。全て「不明」なら、ランダムに探索するしかない。探索が進むにつて「不明」の確率分布が減少する。「不明」でない部分については、高い所と低い所の近傍をどのくらい選べばよいのか求めることができる。例えば、「不明」が 50%の場合は、50%分は、ランダムな探索を行い、残りの 50%分はヒューリスティックな予測に基づいて探索すればよい。初めはランダムな探索から始まり、徐々に、最適解がありそうなところを優先して探索するようになる。

ヒューリスティックな問題分割

動的計画法のように一定の条件を満たす場合に限って、精度の保証を保ちながら問題を分割する方法があるが、精度の保証のないヒューリスティックな方法でも、ないよりはマシである。

貪欲法はその一つだが、人間は無意識にもう少し良い方法を行っている。理想的には、分割したある問題が別の問題に影響しないよう分割するのが望ましいが、現実には、問題を解くのに必要な情報が全て与えられるとは限らず、理想的な分割はできない。しかし、いい加減な分割でも、しないよりはマシである。どのような形であれ、問題を分割することで、最適解にどれだけ近づいているか知り易くなる。例えば、充足可能性問題は、充足可能か否という 2 つの答えしかない。報酬は 0 または 1 であるといえる。ディープラーニングや強化学習を使って、この報酬の期待値が最大になるように学習させようとしても、ランダムな探索と差がないだろう。なぜなら、偶然、報酬 1 に到達するまで、報酬 1 にどれだけ近づいているかという情報が一切得られないからである。そこで、充足可能性問題を和積標準形にして、AND で繋がれた 10 個の項（部分問題）に分割したとしよう。例えば、3 個の項（部分問題）のみ満たすものと、7 個の項を満たすものでは、どちらも報酬 0 だが、後者の方が報酬 1 に近づいていると感じるだろう。しかし、問題の表現の仕方、分割の仕方を変えると、何個の項を満たしているかは、逆転する場合もある。満たしている部分問題の数の大小が、最適化への近さとの大小と必ずしも一致しないとはいえ、他に情報がなければ、部分問題をより満たしている方が最適解に近いとヒューリスティックに判断できる。特別な場合として、項 A、項 B を満たすものと、項 A、項 B、項 C を満たすものでは、論理的に包括関係にあるため、必ず、後者の方が最適解に近い。満たす部分問題を一つずつ増やしていけば、必ず最適解に近づくが、最終的に最適解に到達する保証はない。その方法はいわゆる貪欲法である。部分問題の解を選択すると、制約伝播により、他の部分も問題で有効な選択肢が減ってしまい、詰んでしまう場合がある。その場合、バックトラッキングといった方法で、まだ試していない組み合わせの中で、出来るだけ現在の組み合わせに近く、矛盾が解決するものを選ぶことが有効である。次に試すものをランダムに選ぶのではなく、近傍を探索するとも表現できる。変化させる項の数が少ないほど、近いといえるだろう。貪欲法や深さ優先探索の欠点として、どの順番で探索するかによって、最終到達点や掛かる時間が大きく変わってしまう。途中で問題の分割方法を変えることも、それまでが無駄になってしまうためできない。また、問題として、選択肢がいくつあるのか明示されていれば良いが、現実の問題では、選択肢がいくつあるのかわからない。例えばある製品の不良が発生する条件を探索する場合、加工温度と加工速度だけを選択肢とすればよいのか、それとも気温や湿度までも選択肢として加えなければならないのかは、わからない。いわゆるフレーム問題が起こる。何が影響するか分からないからといって、考えられるすべてのことを考慮したら、選択肢の数は無限になってしまう。無限にある部分問題を端から一つずつ解こうとすると、実際にはほとんど関係ないことばかりを考えてしまうかもしれない。フレームを決め打ちすれば、計算はできるが、最適解にどこまで近づくかは、フレームの決め方で決まってしまう。汎用的であるためには、どう問題を分割し、分割したどの問題について探索するかを、逐次、決めなければならない。それは、現時点までに得られている情報をすべて使い、統計的にどうするべきか推測し、分からなければランダムに決めるしかない。例として、体操の競技で高得点を出したい場合を考えよう。得点はいくつかの項目の得点の合計だと分かっていれば、各項目のそれぞれについて、得点が大きくなるように演技すればよい。ある方向に演技を修正すれば、ある項目の得点は増えるが、別の項目の得点が減ったり、別の項目の得点を増やすための修正は行えなくなったりする場合もある。人間は、最適解を求めているわけではないが、演技のどの

要素が、どの項目に効くかを推測している。演技の要素の最適な組み合わせを探索しなければならないのだが、演技をどう要素に分けるかも考えなければならない。演技の要素の分け方は無数にあるが、どの演技の要素が、どの評価項目の要素に効くかという目的に沿って、分けなければならない。例えば、評価に関係ないことについては、演技を要素に分ける必要はない。ただし、現実では、評価基準が不明で、何が何に効くのかわからない状態から推論を開始しないといけない場合が大半である。例えば、5 つのランプと、10 つのスイッチがある装置で、ランプを全て点灯させたいとしよう。これが、数学的な組み合わせ最適化問題なら、それぞれのランプについて、どのスイッチが ON または OFF であるかという条件が示されるだろう。しかし現実には、回路図は知ることができず、手探りで推測しなければならない場合もあるだろう。回路が分からないからといって、1024 通り全ての組み合わせを二進数で 0 から 1023 の順に試すのは効率が悪いだろう。なぜなら、あるランプは、ある一つのスイッチのみと単純に繋がっている場合もあるからである。一つずつスイッチを操作し、ランプが点灯すればポジティブな効果、消灯すればネガティブな効果があると認知するだろう。何通りか調べれば、どのスイッチが、どのランプに、ポジティブまたはネガティブな効果、または関係がないことを徐々に推測できる。どのランプに対してもポジティブまたは影響ないランプのみ ON に固定し、ポジティブな影響とネガティブな影響が混在するボタンだけについて、ON/OFF の組み合わせを試せば効率的である。それでダメなら、まだ試していない組み合わせを試し、どれがどれに影響するかという推測の精度を上げていけばよい。初めはランダムな探索からはじまり、徐々に統計的に良さそうなものを試すようになり、最終的には総当たりとなる。ランダムな探索では、探索済みの組み合わせを再度探索してしまう可能性がある。探索が進むほどその可能性は高くなり、9 割が探索済みの場合は、9 割は無駄な探索となる。後半での重複による探索効率の悪化を防ぐためには、探索済みのものをメモしておけばよい。一定以上が探索済みになったら、残ったものをしらみ潰しに探索する方法に切り替えればよい。しかし、組み合わせが多いと、探索済みのメモがメモリに入りきらないだろう。そんな場合でも、複数の組み合わせをグループ化して、グループ単位で探索済みのメモを取ればよい。2 の 10 乗で 1024 通りの組み合わせがある場合、探索済みのメモに 1024 ビット必要そうだが、2 ビットずつを組みにして、4 の 5 乗とすれば、32 ビットで済む。1 つ探索する毎に、探索済みのメモをして、次のどこを探索するか決めるよりも、4 つずつまとめて探索した方が、多くの場合で効率的だろう。理想的な汎用アルゴリズムであるためには、どうグループ化するか、どのタイミングでしらみ潰しに切り替えるか、適宜判断する必要がある。

汎用的に多項式時間で解ければ解くアルゴリズム

組み合わせ最適化問題は、多項式時間で解けるものと解けないものがあるが、そのどちらであるかは、情報として与えてはもらえないだろう。多項式時間で解けない問題用のアルゴリズムは、解ける問題にも使えるが、効率が悪い。多項式時間で解ける問題かどうかを判別して、アルゴリズムを切り替えても良いが、判別は難しい。また、汎用的に多項式時間で解ければ解くアルゴリズムがなければ、多項式時間で解ける問題でも、解けない。そこで、汎用的に多項式時間で解ければ解くアルゴリズムを考えよう。さまざまな P 問題に対して、それぞれ個別に

多項式時間で解くアルゴリズムが知られているが、それらを一般化してしまおう。例として2SAT 問題を考える。解法は、x と¬x をそれぞれ代入し、他の変数で選択の余地がなくなればそれを選び、連鎖的に式を簡略化していく。片方の式が矛盾しているのが分かれば、他方の値を固定して、次は別の変数について同様の操作をする。片方に矛盾が見つかる必要は無く、x に固定した場合にに解が見つからなければ、¬x に固定した場合にも解は見つからないという包括関係が分かれば、x に固定することができる。この検証が多項式時間でできるため、2SAT は多項式時間で解ける。2SAT は、1 節に 2 リテラルしかないため、2 リテラルの内の 1 つの値を False と仮定すれば、残りの 1 項が自動的に True に決まる。3SAT では、3 リテラルの内、1 つを False と仮定しても、残りの 2 リテラルのどちらかが、True になればよい。2 リテラルそれぞれについて、True とした場合を仮定しなければ、矛盾があるか検証できない。2 つに場合分けして考える必要がある。片方の場合についてある変数を True と仮定すれば、連鎖的に他の節についても同様の場合分けが必要になる。再帰的に、2 分岐がさらに 2 分岐するため、2 の累乗の計算量が必要になる。ようするに、再帰的に場合分けをする必要がある計算をするには、指数時間が必要である。別の例として 2 部グラフのマッチング問題を考える。男女のペアを作るような問題である。この問題は P 問題だが、2SAT の場合のように、一人ずつ、仮の相手を仮定するという方法では、矛盾が導けず解けない。解法は、適当にペアを作れるものから作ってしまい、組める相手が居ない人が発生したときに、その矛盾を解決していく。A は B または C としかペアを組めないが、B は D と、C も E と既にペアを組んでいるとしよう。A は、D から B を奪うか、E から C を奪わなければならない。D または E とペアを組むのが可能で、まだペアになっていない人が居れば良いが、居なければ、他の既存のペアから奪わなければならない。連鎖的なペアの変更を考えなければいけないが、場合分けして考える必要がなく多項式時間で解ける。初めに「A」とペアが組めて空いている人を探し、いなければ、「A、D、E」とペアが組めて空いている人を探す。D、E についても居なければ、「A、D、E、F、G、H、I」というように連鎖的にリストを増やしていく。リスト内の誰か一人でもペアが組めて空いている人が居ればよく、居なければ、リスト内の誰でもいいので、ペアを奪える相手をリストに追加していけばよい。この方法は 3 部グラフのマッチングでは使えない。(A と B) または(C と D)から、ペアを奪わないといけない形になるため、一つのリストにはならず、場合分けしなければならない。このマッチング問題を解くアルゴリズムは、2SAT も解くことができるため、より汎用的なアルゴリズムである。より一般的な場合を考えると、まずは、乱択法や貪欲法で選択肢を埋めていき、それ以上、最適解へ近づけない状態に行き詰ったときに、そこまでの選択肢を修正しようとする。選択肢を変えたときの影響が、他の選択肢で場合分けしなくても分かる場合は、多項式時間で解ける。帰納的推論の場合は、どう影響するか問題として与えられないが、統計的に予測する。場合分けが必要なら、指数時間を掛けて、各場合を検証するしかない。しかし、その場合でもヒューリスティックな手法が使える。ある選択肢を変えたときの二次的な影響は無視して、一時的な影響だけを見て、優先的に調べる場合を決めれば、ランダムよりは良い。このアルゴリズムなら、あらかじめ多項式時間で解けるかどうかを区別する必要がない。問題の種別を判別して使用するアルゴリズムを切り替える必要がなく、一つの汎用的なアルゴリズムで対応できる。

P≠NP 問題の曖昧さ

P≠NP 問題として、一般的には P≠NP と信じられているが証明はされていない。上述の、多項式時間で解ければ解けるアルゴリズムで解ければ P だが、解けないものは本当に解く方法はないのだろうか。直感的には、再帰的に場合分けが必要な問題なら指数時間が掛かると感じるだろう。この問題を難しくしている原因は大きく分けて 3 つ存在する。

P≠NP 問題の証明を難しくしている原因の一つは、数学的な定理が全て明白にはなっておらず、未知の定理の存在を否定できないからである。思いもよらなかったトリッキーな定理が見つかり、不可能と思っていたことが可能になるかもしれない。特に、素数についてはまだ解明され尽くされてはおらず、数字を使う以上、全てが明らかとはいえない。ただし、ブール代数のみを扱う問題では、そうでもない。論理式を変形すると解き易くなることはあるが、変形できる形は有限であり、全て調べることも可能だ。複雑に見えても、単純な論理の組み合わせに過ぎず、未知の定理で介入する余地はない。P≠NP 問題を考えるなら、充足可能性問題を考えれば、余計なことを考えずに済むだろう。

P≠NP 問題の証明を難しくしている 2 つ目の原因は、入力の情報量に対する計算量というのが、計算の難しさと合わないからである。例えば、NP 問題であっても、問題にゴミデータを付加して、入力 bit 数を水増しすれば、入力 bit 数に対して、計算時間は、多項式時間となる。ゴミデータではなく、問題を解くのに必要なデータでも同様である。例えば、ロスタイム辞書付き巡回サラリーマン問題を考えてみよう。ロスタイム辞書は、全てのルートの組み合わせの一つずつについて、ロスタイムをいくら加えるかが記載されている巨大（指数スケール）な辞書であり、問題として与えられる。ロスタイム付きの問題を解くには、ロスタイム無しの問題を解いた上で、さらにロスタイムを調べなければならない。ロスタイムなしでのアルゴリズム＋追加の処理になり、論理的包括関係にあるため、ロスタイム付きは、ロスタイムなしよりも必ず計算量が多く、すなわち難しい。しかし、ロスタイム付きは P であり、より易しい問題であるロスタイムなしは NP である。この場合、P より NP の方が、計算量が少なく易しい問題となってしまう。P または NP というのは計算量を表すのではなく、同じ量の計算をする問題で、情報量をどこまで圧縮できるかという表現の方が適当だろう。計算の複雑さを議論するなら、情報量ではなく、組み合わせの数で考えるべきだろう。

P≠NP 問題の証明を難しくしている 3 つ目の原因は、どうなれば多項式時間で解けたといえるかという解釈にブレがあるからである。実は、NP 問題と信じられている全ての問題は多項式時間で解くことができる。そのアルゴリズムは簡単で、あらかじめ全ての組み合わせについて計算して辞書を作っておく。問題が示されたら 2 分探索すれば、n ビットの問題なら n 回の条件分岐で、解を得られる。この方法には 3 つの疑念が存在する。一つ目の疑念は解の辞書を見るのは、解いているといえないのではないかだ。しかし、アルゴリズムは変形することができるため、あらかじめ保存されていた 0 または 1 という値を代入する行為と、値の代入はせず、単に条件分岐するプログラムは全く区別できない。逆に、真っ当なアルゴリズムと思っていたものでも、変形して最適化していけば、単に辞書を読み出すだけのものになるだろう。2 つ目の疑念として、この方法は、問題が示されてから解くまでは多項式時間だが、その下準備としてプログラムを組むときに指数時間が掛かってしまう。とはいえ、問題を解くアルゴリズムの

速さを競うのに、プログラムするのに掛かる時間は加えるべきではないだろう。仮に加えるとしても、プログラムするのにいくら時間が掛かったかを示すのは困難である。例えば、偶然頭に思い浮かんだコードを書いたら、指数時間かけないと分からないはずの答えの辞書ができるかもしれない。あるプログラムのアイデアが必然的に得られるのにどれだけ時間が掛かるかを論ずるのは困難である。3 つ目の疑念は、与えられた問題のビット数が、想定して作っておいた辞書を超えたら使えないが、解けたといえるのだろうか。最大〇bit という制約がある問題なら、多項式時間で解けるといえよう。入力 bit 数が無限なら解けないが、それはどんなアルゴリズムでも解けない。実用的には、解をあらかじめ求めておく方法は、どんな量子コンピューターよりも速いが、記憶領域を使い過ぎるため、難しい問題には使えない。