グラフ構造によるアルゴリズムの表現を用いた AutoML-Zero の提案

三嶋隆史

2025 年 1 月 27 日

第 1 章

序論

1.1 研究の背景と目的

AutoML は, 機械学習のモデルを自動で最適化する手法として注目されてきた. AutoML 登場以前は, 機械学習のモデルを最適化を人間の手でしていたため, 高度な専門知識と膨大な時間を必要としていた. AutoML はこの膨大にかかる最適化プロセスをコンピュータに置き換えることを目指してきた分野であり [1][4][2], 今後の機械学習の進歩において非常に重要である. これまでの AutoML に関する研究の多くは, 計算コストを抑えるために人間のデザインに大きく依存した制約付きの空間を探索している. 例えば, ニューラルネットワークの構造探索では, 事前に専門家が用意した経験則的に性能が高くなる層を構成要素として使うことで, 最適化の対象を構成要素の組合わせやハイパーパラメータに限定したり, 重みの更新方法として常に誤差逆伝搬法を用いることで探索空間を制限している [13][8][12].

Esteban らが言及しているように, 探索空間を限定する既存の AutoML には, 主に 2 つの問題点が存在する [9]. 1つ目は, 人間がデザインした探索空間にはバイアスがかかってしまい, 人間がまだ発見していないより良いアルゴリズムを見つけられる可能性が減少してしまう点である. 実際, Elsken らによって, 探索空間の制限は性能に影響を大きく影響を与える観点が無視されることがあることが報告されている [6]. 2つ目は探索空間を限定する際は, 極めて慎重に行う必要があり [14][11][7], 結果的に研究者に負担が掛かってしまう点である. これは, AutoML の本来の目的である人間の介入を最小限にするという点に反する.

Esteban らは, これらの AutoML における課題を解決するため, 人間からの入力を最小限に抑えた機械学習アルゴリズムの探索手法 AutoML-Zero を提案した [9]. AutoML-Zero は, Regularized Evolution (RE) を世代交代モデルとして採用した進化計算により, 与えられた機械学習タスクの集合内の各タスクに対する適合度を最大化する機械学習アルゴリズムを探索する. 機械学習タスクに対するアルゴリズムの適合度は, 学習データを用いて学習した後, 検証データに対する損失の平均を $[0, 1]$ の範囲に変換した値として定義される. AutoML-Zero では, 機械学習アルゴリズムを仮想メモリ上で動作するプログラムとして表現する. プログラムは Setup, Predict, Learn の 3 つの関数で構成され, 各関数内では高校数学レベルの基本的な演算のみが許可されている. アルゴリズムの改善は, RE による突然変異を通じて行われる. 具体的な突然変異操作として, 命令の追加・削除, 変数アドレスの書き換え, および関数全体のランダムな初期化が実装されている. このように, 人間の事前知識をほとんど使用しない設計にもかかわらず, AutoML-Zero は勾配降下法や ReLU 関数の再発明に成功するなど, 注目すべき成果を示している [9]. また, Esteban らの貢献をベースとして, 様々な AutoML-Zero の拡張が提案されている. 例えば Guha らは, 精度と計算コストのトレードオフ構造を考慮した多目的の AutoML-Zero (Moaz) を提案している [3]. AutoML-Zero の最適化対象が適合度のみであるこ

とに対して, Moaz では計算コストも考慮した多目的の最適化を行っている．また, Hao らによって, AutoML-Zero の探索手法を利用して損失関数のみを探索する手法も提案されている [5].

しかしながら, Esteban らの AutoML-Zero は人間の事前知識や介入を最小限に抑えつつ機械学習アルゴリズムを自動探索するという画期的な成果を示した一方で, その探索効率には重要な課題が残されている．例えば, ReLU 関数の再発明に成功し, CIFAR-10 や MNIST データセットに対する分類アルゴリズムを発見した Section 4.2 の実験では, 膨大な計算リソースが必要とされた．具体的には, 1 秒間あたり 10,000 モデルの評価が可能な CPU を搭載したマシンを 10,000 台使用し, 約 5 日間, 評価回数にして 10^{12} オーダーの計算を実行している．このような大規模な計算リソースの要求は, 現実的な環境下での応用可能性を制限していると考えられる．

我々は, AutoML-Zero において探索効率を低下させ得る 3 つの要因に着目した．第一に, 探索空間の冗長性が挙げられる．既存手法では, 機械学習アルゴリズムとして非妥当なアルゴリズムまでもが探索対象に含まれている．例えば, 予測ラベルが代入されないアルゴリズム、入力ベクトルが予測に利用されていないアルゴリズム、逐次更新される学習パラメータが存在しないアルゴリズム, 学習時に正解ラベルを利用していないアルゴリズム、予測に無関係な変数が存在するアルゴリズムが探索対象となっている。また, 既存手法は本質的に同じアルゴリズムでも, 変数名が異なると別のアルゴリズムとして, 扱われてしまう冗長性も存在する．これらの要因により, 既存手法では無意味な評価が多く発生し, 探索効率を低下すると考えられる．第二に, 突然変異操作に関する 2 つの問題が挙げられる．一つは, 非妥当な突然変異が多く発生し, 探索が非効率になる問題である．ここで非妥当な突然変異とは, 親個体が妥当なアルゴリズムであるにも関わらず, 非妥当なアルゴリズムに変化してしまう突然変異を意味する．また既存手法では, 関数全体をランダムに再構成する極めて破壊的な突然変異が高確率で発生する．故に, 関数中の良質な部分命令列を保存しつつ, 関数を逐次改善することが難しいと考えられる．第三に, 集団の多様性維持に関する問題が挙げられる．既存手法で採用されている世代交代モデル RE は, 佐藤らの研究が示唆するように, 集団の多様性維持が困難であると考えられる [10]．さらに, 集団内における個体の重複や希少度が考慮されていないため, 同一もしくは類似したアルゴリズムが集団内に過度に増加してしまう傾向があると考えられる．既存手法は, 上述した多様性維持を困難にする要因により, 探索序盤で発見された局所最適なアルゴリズムに早期収束してしまい, 探索効率の低下を招いていると考えられる．

本研究の目的は, 人間の事前知識や介入を最小限に抑えて機械学習アルゴリズムの探索が可能であるという AutoML-Zero の利点を保持しつつ, 探索効率を大幅に改善した新たな手法を提案することである．具体的には, 既存手法の問題点に対処した手法を提案し, 評価回数を揃えて探索を行った時に, 既存手法よりも高い適合度を持つアルゴリズムの発見できることを確かめる．提案手法では, グラフ構造を用いてアルゴリズムを表現する手法（アルゴリズムグラフ）を導入する．その上で, 機械学習アルゴリズムとして満たすべき条件（妥当なアルゴリズムの条件）を明確に規定する．これにより, 条件を満たさないアルゴリズムを探索空間から除外し, 探索空間の冗長性の削減を目指す．また, 変数名が異なるアルゴリズムでも, グラフ構造は同一になるので, 提案手法では変数名による冗長性も排除できると考えられる．提案手法では, アルゴリズムグラフに対する突然変異を導入することで, 非妥当な突然変異が起こらないように設計する．また, 部分グラフの再構成を用いた新たな突然変異により, アルゴリズムの関数内の部分命令列を局所的に変更することを実現する．突然変異の工夫により, 近傍探索性能が向上し, より効率的な探索ができると期待される．さらに, 集団の多様性維持のため, Minimal Generation Gap (MGG) [10] による世代交代モデル, 集団内の同一個体の重複排除, および希少度に基づく生存選択を導入する．これにより, 提案手法は集団の多様性が維持されやすくなり, 探索の初期段階で局所最適なアルゴリズムに収束することを防げると考えられる．

1.2 本論文の貢献

本研究の主要な貢献は以下の 4 点である.

- グラフ構造によるアルゴリズム表現の提案: 機械学習アルゴリズムをグラフ構造で表現し, 満たすべき条件を明確に規定した. これにより, 条件を満たさないアルゴリズムを探索空間から排除し, 探索空間を xx% 以上削減することに成功した. また, アルゴリズムをグラフ構造で表現したことで, 変数名によるアルゴリズムの冗長性の排除をすることができた.
- グラフ構造に基づく突然変異操作の提案: グラフ構造を活用した突然変異により, 既存手法で xx% 発生していた非妥当な突然変異を完全に発生しないようにすることができた. さらに, 部分グラフに対する突然変異を導入することで, 特定の変数計算に必要な部分命令列のみを変更可能な突然変異を実現した. 突然変異の改善により, 同一評価回数内で発見されるアルゴリズムの適合度を x% 以上向上させることに成功した.
- 集団の多様性維持のための手法の提案: Minimal Generation Gap (MGG) による世代交代モデル, 集団内の同一個体の排除, および希少度を考慮した生存選択を導入した. これらの改善により, 集団の多様性が維持され, 早期収束が抑制された結果, 同一評価回数内で発見されるアルゴリズムの適合度が x% 以上向上した.
- 提案手法の有効性の検証: ローカル PC という限られた計算リソース環境において, 主要な回帰問題と分類問題を用いた比較実験を実施した. その結果, 同一評価回数内で発見されるアルゴリズムの適合度が xx% 以上向上することを確認した. また, 線形回帰問題において, 適合度が 0.999 以上のアルゴリズムを発見するまでの評価回数を比較した結果, 提案手法は既存手法の 1/x で発見に成功した.

1.3 本論文の構成

本論文では, 第 1 章で研究の背景と目的および貢献, 第 2 章で AutoML-Zero の問題設定と Esteban らが提案した既存手法の説明と問題点, 第 3 章で既存手法の問題点に対処した提案手法の詳細な説明, 第 4 章で提案手法の有効性を検証するための実験, 第 5 章で実験結果の考察や提案手法の各工夫の有効性の検証, 第 6 章で本研究のまとめと今後の課題について述べる.

第 2 章

問題の所在

2.1 はじめに

本章では, 第 2.2 節で我々が研究対象とする AutoML-Zero の問題設定の定義を説明した上で, 第 2.3 節で AutoML-Zero の既存手法である RE-AutoML-Zero について説明する. その後, 第 2.4 節で, 探索空間の冗長性に関する問題, 突然変異に関する問題, 集団の多様性時に関する問題について述べる.

2.2 問題の定義

本研究で対象とする AutoML-Zero は, 与えられた機械学習タスク集合 $\mathcal{T}$ 内の各機械学習タスクに対して, 最も高い適合度を達成可能なアルゴリズム $a^* \in \mathcal{A}$ を, $\mathcal{T}$ の代表的な部分集合 $\mathcal{T}_{\text{search}} \subset \mathcal{T}$ から探索する問題である. ここで, $\mathcal{A}$ はアルゴリズム全体の集合である. 以下, 第 2.2.1 節でタスク集合, 第 2.2.2 節で適合度, 第 2.2.3 節において最適なアルゴリズムの詳細な説明を行う.

2.2.1 タスク集合

タスク集合 $\mathcal{T}$ は, 複数の機械学習タスクによって構成される. 各機械学習タスク $T \in \mathcal{T}$ は, 入力ベクトル $\boldsymbol{x}_j$ と正解ラベル y_j の順序対の集合であり, 以下のように定義される.

$$T = \left\{ (\boldsymbol{x}_j, y_j) \mid \boldsymbol{x}_j \in \mathbb{R}^d,\ y_j \in \mathbb{R},\ j \in \mathbb{N},\ 1 \leq j \leq N \right\}$$

ここで, N および d はそれぞれタスクごとに定まるデータの個数とタスクの次元である. また, タスク T には学習用データ $D_{\text{train}} \subset T$ および検証用データ $D_{\text{valid}} \subset T$ が定められており, $D_{\text{train}} \cup D_{\text{valid}} = T$, $D_{\text{train}} \cap D_{\text{valid}} = \emptyset$ を満たす.

2.2.2 適合度

アルゴリズム a のタスク T に対する適合度 $F(a, T)$ は, 学習データ D_{train} をアルゴリズム a で学習させた上で, 検証データ D_{valid} に対する損失を $[0, 1]$ に変換することで計算される. 適合度は, 1 に近いほどタスク T に適合している, 言い換えれば T の検証データに対する損失が小さいことを意味する. 損失や損失を $[0, 1]$ に変換する関数はユーザによって与えられる. 例えば, 線型回帰のタスクであれば, D_{valid} に対する予測ラベルと正解ラベルの平均二乗誤差 l が損失として利用され, 適合度は $1 - \frac{2}{\pi} \arctan\left(\sqrt{l}\right)$ を用いて, $[0, 1]$ の区間に変換することで算出される.

2.2.3 最適なアルゴリズム

本研究における最適なアルゴリズム $a^* \in \mathcal{A}$ は, 以下のように定式化される.

$$a^* = \arg\max_{a \in \mathcal{A}} \frac{1}{|\mathcal{T}_{\text{eval}}|} \sum_{T \in \mathcal{T}_{\text{eval}}} F(a, T) \tag{2.1}$$

一般に, $\mathcal{T}$ は無限集合であり, $\mathcal{T}$ の全タスクに対する適合度の計算は困難である. そのため, $\mathcal{T}$ の有限部分集合 $\mathcal{T}_{\text{eval}}$ 内のタスクに対する適合度の平均を最大化するアルゴリズムを最適なアルゴリズムと定義する. ここで, $\mathcal{T}_{\text{eval}}$ に含まれるタスクは $\mathcal{T}_{\text{search}}$ と同様にタスクの種類に偏りが生じないよう構成する必要がある. また, $\mathcal{T}_{\text{search}}$ への過適合を防ぐため, $\mathcal{T}_{\text{search}}$ と $\mathcal{T}_{\text{eval}}$ は互いに独立に構成する.

最適なアルゴリズムについて具体例を挙げて説明する. $\mathcal{T}$ が線形回帰タスク全体 $\mathcal{T}_{\text{LinReg}}$ である場合, $\mathcal{T}_{\text{LinReg}}$ 内の全タスクに対して誤差 0 で回帰可能な線型回帰アルゴリズム $a_{\text{LinReg}} \in \mathcal{A}$ が最適なアルゴリズムとなる. 一方, $\mathcal{T}$ が一般の回帰問題全体の集合 $\mathcal{T}_{\text{Reg}}$ である場合, $\mathcal{T}_{\text{Reg}}$ には線型回帰タスクに加えて非線形回帰タスクも含まれる. このとき, 線型回帰アルゴリズム a_{LinReg} では非線形回帰問題に対して十分な精度を得られないため, 最適なアルゴリズムとはならない. したがって, 本問題設定では, タスク集合として $\mathcal{T}_{\text{LinReg}}$ が与えられた場合は線型回帰アルゴリズム $a_{\text{LinReg}} \in \mathcal{A}$ が, $\mathcal{T}_{\text{Reg}}$ が与えられた場合は非線形回帰タスクにも対応可能なアルゴリズム $a_{\text{Reg}} \in \mathcal{A}$ が得られることが求められる.

2.3 既存手法 RE-AutoML-Zero

本節では, Esteban らが提案した AutoML-Zero 手法について述べる. Esteban らが提案した手法は, Regularized Evolution (RE) を使った AutoML-Zero 手法であるため, 以降では RE-AutoML-Zero と呼ぶ. 以下, 第 2.3.1 項では RE-AutoML-Zero におけるアルゴリズムの表現方法, 第 2.3.2 項では RE-AutoML-Zero におけるアルゴリズムの評価方法, 第 2.3.3 項では RE による世代交代, 第 2.3.4 項では突然変異による個体生成について述べる.

2.3.1 アルゴリズムの表現方法

RE-AutoML-Zero において機械学習アルゴリズムは, 小さな仮想メモリで動作するプログラムとして表される. 仮想メモリには, スカラー, d 次元ベクトル, $d \times d$ 次元行列を複数個格納できる. ここで, d はタスク集合 $\mathcal{T}_{\text{search}}$ に含まれるタスク T の入力ベクトルの次元である. 以降, スカラーを格納する変数を $s0, s1, \cdots$, ベクトルを格納する変数を $v0, v1, \cdots$, 行列を格納する変数を $m0, m1, \cdots$ と表す. $s0$, $s1$, $v0$ は, それぞれ正解ラベル, アルゴリズムによる予測ラベル, 入力ベクトルを格納する先として使われる特別な変数である. その他の変数は学習対象のパラメータを格納したり, 計算結果を一時的に保存する用途で用いられる. 変数の個数の上限はスカラー, ベクトル, 行列それぞれに対してユーザが指定する必要がある.

アルゴリズムは, Code.2.1 に示したように, Setup, Predict, Learn の 3 つの関数で表現される. 各関数は Table.B.1 に示した 64 個の命令の列で構成される. 命令は, 人間のバイアスを与えすぎないようにするため, 高校数学で学ぶ程度の演算のみを使用し, 機械学習のアルゴリズムの概念や行列の分解等の演算は含まれていない. また, 命令に与える引数は, 基本的には仮想メモリに格納されているスカラー $s1, s2, \cdots$, ベクトル $v1, v2, \cdots$, 行列 $m1, m2, \cdots$ のいずれかである. 一部例外として, 正規分布による乱数生成の命令等では, μ, σ 等の定数が入力されることもある.

Code. 2.1　アフィン回帰と解釈可能なアルゴリズム

```
def Setup():
  s3 = 0.01 // 学習率の設定

def Predict():
  s2 = dot(v6, v0) // 学習対象の重みと入力ベクトルの内積を計算
  s1 = s7 + s2 // 切片を加算

def Learn():
  s4 = s0 - s1 // 予測ラベルと正解ラベルの誤差を計算
  s6 = s3 * s4 // 学習率の適用
  v3 = s6 * v0 // 傾きの更新するための差分を計算
  v6 = v6 + v3 // 傾きを更新
  s7 = s7 + s6 // 切片を更新
```

2.3.2　アルゴリズムの評価方法

既存手法のアルゴリズムの探索では, 以下の式で定義されるタスク集合 $\mathcal{T}_{\text{search}}$ に対する適合度 $F(a, \mathcal{T}_{\text{search}})$ を評価指標として用いる.

$$F(a, \mathcal{T}_{\text{search}}) = \frac{1}{|\mathcal{T}_{\text{search}}|} \sum_{T \in \mathcal{T}_{\text{search}}} F(a, T)$$

この適合度 $F(a, \mathcal{T}_{\text{search}})$ を最大化するアルゴリズムは, $\mathcal{T}_{\text{eval}}$ のタスクに対しても高い適合度を示すことが期待される.

タスク集合内の 1 つのタスク T に対するアルゴリズムの評価は, Algorithm 1 に示した流れで行われる. Algorithm 1 の入力は評価対象のアルゴリズム $a \in \mathcal{A}$, タスク $T \in \mathcal{T}_{\text{search}}$ の学習用データ D_{train} および検証用データ D_{valid} であり, 出力は評価対象のアルゴリズムのタスク T に対する適合度である.

Algorithm 1 タスク T に対するアルゴリズムの評価方法

Input: 評価対象のアルゴリズム $a = (\text{Setup}, \text{Predict}, \text{Learn})$, タスク $T = D_{\text{train}} \cup D_{\text{valid}}$
Output: タスク T に対するアルゴリズムの適合度 $F(a, T)$

initialize_memory()
Setup()
for $e = 0, 1, \cdots, N_{\text{epochs}}$ **do**
  for all $(\boldsymbol{x}_j, y_j) \in D_{\text{train}}$ **do**
    $v0 \leftarrow \boldsymbol{x}_j$
    Predict()
    $s1 \leftarrow \text{Normalize}(s1)$
    $s0 \leftarrow y_j$
    Learn()
  end for
end for
$l_{\text{sum}} = 0.0$
for all $(\boldsymbol{x}_j, y_j) \in D_{\text{valid}}$ **do**

14: $v0 \leftarrow \boldsymbol{x}_j$
15: Predict()
16: $s1 \leftarrow \mathrm{Normalize}(s1)$
17: $l_{\mathrm{sum}} \leftarrow l_{\mathrm{sum}} + \mathrm{Loss}(y, s1)$
18: **end for**
19: $l_{\mathrm{mean}} \leftarrow l_{\mathrm{sum}} / |D_{\mathrm{valid}}|$
20: fitness = $\mathrm{Rescale}(l_{\mathrm{mean}})$
21: **return** fitness

Algorithm 1 の各行の詳細な説明を以下に示す.

1 行目 仮想メモリをすべて 0 で初期化する.
2 行目 Setup 関数を実行する.
3-11 行目 学習用のループを実行する. N_{epochs} はエポック数であり, 学習用データを使う回数を表す.
4-10 行目 e 番目のエポックに対応する学習を行う. エポックごと $(\boldsymbol{x}_j, y_j) \in D_{\mathrm{train}}$ を取り出す順番は異なる.
5 行目 $v0$ に入力ベクトル $\boldsymbol{x}_j$ を代入する.
6 行目 Predict 関数の実行を行う. 正解ラベル y_j は事前に代入されていないので使うことが出来ない. また, Predict 関数実行後は, $s1$ に予測結果が含まれているものとして扱う.
7 行目 $s1$ に格納されている予測結果を Normalize 関数を用いて正規化する. Normalize 関数は, 回帰タスクの場合には恒等関数, 二値分類タスクの場合には sigmoid 関数などが使われる.
8 行目 正解ラベルを $s0$ に代入する.
9 行目 Learn 関数を実行する.
12 行目 検証用データにおける損失の合計を計算するための変数 l_{sum} を初期化する.
13-18 行目 検証用データ D_{valid} を用いて検証用ループを実行する.
14 行目 学習用ループと同様に $v0$ に入力ベクトル $\boldsymbol{x}_j$ を代入する.
15 行目 学習用ループと同様に Predict 関数の実行を行う.
16 行目 学習用ループと同様に予測結果を正規化する.
17 行目 学習用ループとは異なり, Learn 関数の実行はせず予測結果の損失を Loss 関数を使って計算する. Loss 関数は, 回帰タスクの場合には二乗誤差を返す関数, 二値分類タスクの場合にはの場合には予測ラベルと正解ラベルが一致しているときに 1, それ以外のときは 0 を返す関数である.
19 行目 損失の平均値 l_{mean} を計算する.
20 行目 損失を Rescale 関数を用いて適合度に変換する. Rescale 関数は, 回帰タスクの場合は $\mathrm{Rescale}(l) = 1 - \frac{2}{\pi} \arctan \sqrt{l}$, 二値分類タスクの場合は $\mathrm{Rescale}(l) = 1 - l$ などが用いられる.
21 行目 適合度を返却する.

1 行目で行ったメモリの初期化以降, 特別な変数 $s0$, $s1$, $v0$ 以外の変数への代入は, Setup, Predict, Learn の中以外で行われることがない. そのため, 評価対象のアルゴリズムの各関数 Setup, Predict, Learn では, $s0$, $s1$, $v0$ 以外の変数に学習対象のパラメータを格納することで, 初期化時から検証時まで当該パラメータの値を引き継ぐことができる.

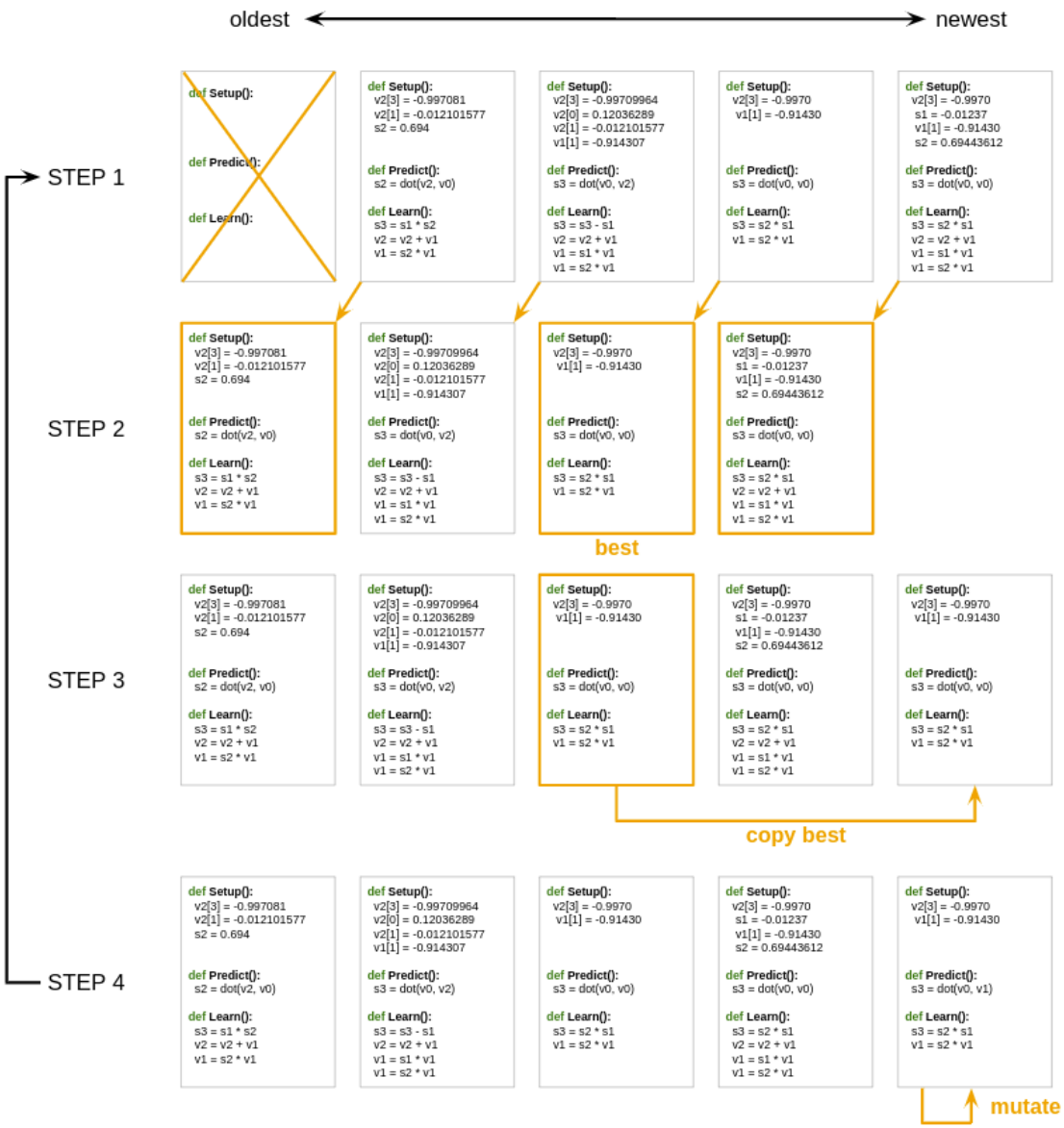

Fig.2.1　RE-AutoML-Zero[9] の世代交代モデル．STEP1 から STEP4 を繰り返すことで最適なアルゴリズムの発見を目指す．STEP1 で最も古い個体を削除した後に, STEP2 でトーナメント選択を行う．その後, STEP3 でトーナメント選択によって選ばれた個体をコピーし, STEP4 で一定確率 p_{mutate} で突然変異を行う．

2.3.3　RE による世代交代

Esteban らが提案した RE-AutoML-Zero では, N_{pop} 個のアルゴリズムをランダムに初期集団として生成した後に, Fig.2.1 の STEP1 から STEP4 に示した Regularized Evolution (RE) を繰り返し行うことで, 最適なアルゴリズムの探索を行う．RE では, STEP1 で最も古い個体を削除した後に, STEP2 でトーナメント選択, すなわち $K(< N_{\text{pop}})$ 個の個体を非復元抽出した上で最も適合度の高い個体を選択を行う．その後, STEP3 でトーナメント選択によって選ばれた個体

をコピーし, STEP4 で一定確率 p_{mutate} で突然変異を行う. 集団サイズ N_{pop}, トーナメントサイズ K, 突然変異確率 p_{mutate} はユーザパラメータである.

RE-AutoML-Zero の詳細なアルゴリズムを Algorithm 2 に示す. Algorithm 2 の入力はタスク集合 $\mathcal{T}_{\mathrm{search}}$, 集団サイズ N_{pop}, トーナメントサイズ K, 突然変異確率 p_{mutate}, 最大評価回数 N_{eval} である. また, 出力は探索結果のアルゴリズムである.

Algorithm 2 Regularized Evolution (RE) のアルゴリズム

Input: タスク集合 $\mathcal{T}_{\mathrm{search}}$, 集団サイズ N_{pop}, トーナメントサイズ K, 突然変異確率 p_{mutate}, 最大評価回数 N_{eval}
Output: 発見されたアルゴリズム

```
1: eval_num ← 0
2: P ← ∅
3: for i = 1 to N_pop do
4:    algorithm ← generate_algorithm()
5:    fitness ← F(a, T_search)
6:    eval_num ← eval_num + 1
7:    individual ← (algorithm, fitness)
8:    P ← P ∪ {individual}
9: end for
10: while eval_num < N_eval do
11:    oldest = get_oldest(P)
12:    P ← P \ {oldest}
13:    (base_algorithm, fitness) ← tournament_select(P, K)
14:    algorithm ← copy(base_algorithm)
15:    if rand(0, 1) < p_mutate then
16:       algorithm ← mutate(algorithm)
17:       fitness ← F(a, T_search)
18:       eval_num ← eval_num + 1
19:    end if
20:    individual ← (algorithm, fitness)
21:    P ← P ∪ {individual}
22: end while
23: (best_algorithm, best_fitness) ← get_best_fitness(P)
24: return best_algorithm
```

Algorithm 2 の各行の説明を以下に示す.

1 行目　評価回数のカウンターを初期化する.
2 行目　集団 P を初期化する.
3-9 行目　集団 P に N_{pop} 個のアルゴリズムをランダムに生成し, 適合度を計算する.
　　4 行目　アルゴリズムをランダムに生成する関数 generate_algorithm() を実行する.
　　5 行目　タスク集合 $\mathcal{T}_{\mathrm{search}}$ に対するアルゴリズムの適合度を計算する.
　　6 行目　評価回数をインクリメントする.
　　7 行目　アルゴリズムと適合度のペアである個体を作成する.
　　8 行目　集団 P に個体を追加する.
10-22 行目　評価回数が上限を超えるまで RE による世代交代を繰り返す.

11 行目　集団 P から最も古いアルゴリズム (個体) を取り出す関数 get_oldest(P) を実行する.

12 行目　集団 P から最も古いアルゴリズムを消去する.

13 行目　集団 P からトーナメントサイズ K でトーナメント選択する関数 tournament_select(P, K) を実行する.

14 行目　トーナメント選択されたアルゴリズムをコピーする.

15-19 行目　一定確率 p_{mutate} で突然変異を行う.

16 行目　アルゴリズムを突然変異させる関数 mutate(algorithm) を実行する.

17 行目　タスク集合 $\mathcal{T}_{\text{search}}$ に対するアルゴリズムの適合度を計算する.

18 行目　評価回数をインクリメントする.

20 行目　アルゴリズムと適合度のペアである個体を作成する.

21 行目　集団 P に個体を追加する.

23 行目　集団 P から最も適合度の高いアルゴリズムを取得する.

24 行目　最も適合度の高いアルゴリズムを返却する.

2.3.4　突然変異による子個体生成

RE-AutoML-Zero の突然変異は図 2.2 に示した 3 種類が存在する. それぞれの突然変異手法の以下で説明する.

(1) Setup 関数, Predict 関数, Learn 関数のいずれかを一様ランダムに選択した上で, 選択された関数に対して命令の追加または削除を行う突然変異. 追加または削除する場所も一様ランダムに決まる.

(2) Setup 関数, Predict 関数, Learn 関数のいずれかを, ランダムな命令列で書き換える突然変異. 命令数は元の個数と変更しない.

(3) Setup 関数, Predict 関数, Learn 関数のいずれかを選択した上で, その関数の 1 つの命令の入力, 出力もしくは即値のいずれかをランダムに変更する突然変異.

突然変異はユーザが与える制限を満たす範囲でのみ行うことができる. RE-AutoML-Zero のユーザは, Setup 関数, Predict 関数, Learn 関数のそれぞれに対して, 命令数の下限および上限, 命令の種類を指定することができる, また, スカラー, ベクトル, 行列それぞれの仮想メモリのアドレス数も指定可能である. そのため, (1) によってユーザが設定した命令数の上限を超えた追加や下限を下回る削除を行ったり, (2) によって許可されていない命令を関数に追加することは出来ない. また, 仮想メモリ上のスカラーの変数の個数が 3 個と決まっている場合, すなわち $s0$ から $s2$ まで使える場合に, (3) で命令の入力変数として $s4$ に設定することはできない.

2.4　既存手法の問題点

Esteban らの AutoML-Zero は人間の事前知識や介入を最小限に抑えつつ機械学習アルゴリズムを自動探索するという画期的な成果を示した一方で, その探索効率には重要な課題が残されている. 例えば, ReLU 関数の再発明に成功し, CIFAR-10 や MNIST データセットに対する分類アルゴリズムを発見した Section 4.2 の実験では, 膨大な計算リソースが必要とされた. 具体的には, 1 秒間あたり 10,000 モデルの評価が可能な CPU を搭載したマシンを 10,000 台使用し, 約 5 日間, 評価回数にして 10^{12} オーダーの計算を実行している. このような大規模な計算リソースの要求は, 現実的な環境下での応用可能性を制限していると考えられる.

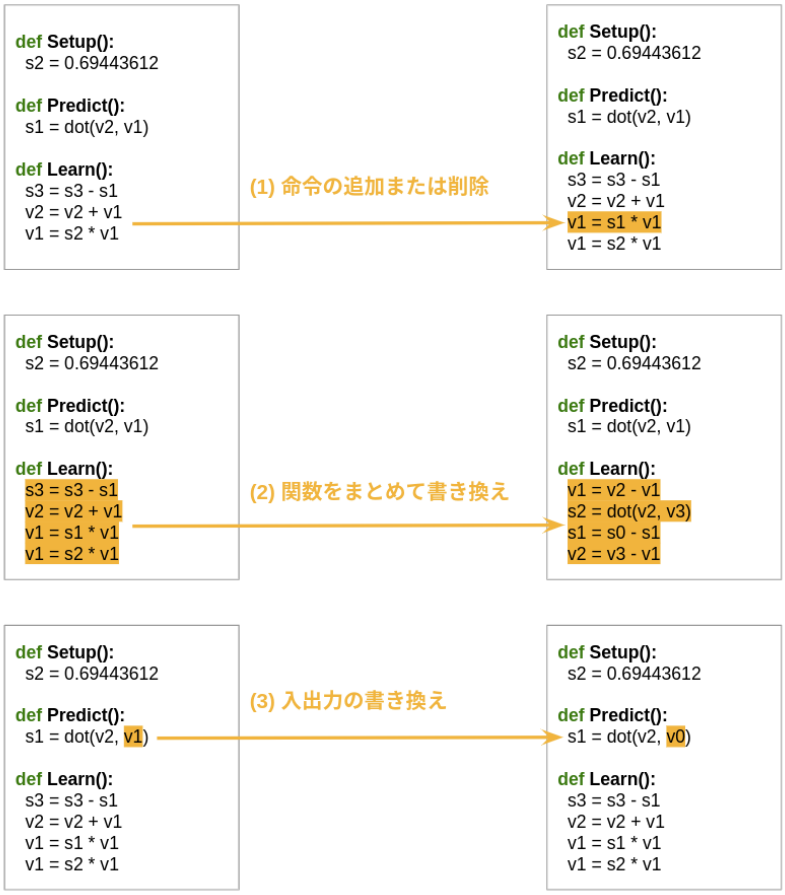

Fig.2.2　突然変異の種類 [9]．(1) アルゴリズムの Setup, Predict, Learn のいずれかをランダムに選択した上で，ランダムに命令を追加または削除する．(2) アルゴリズムの Setup, Predict, Learn のいずれかをすべてランダムな命令列で書き換える．(3) アルゴリズム内のいずれかの命令の入出力をランダムに変更する．

我々は，既存手法の探索効率を低下させ得る問題として，探索空間の冗長性に関する問題，突然変異に関する問題，集団の多様維持に関する問題に着目した．以下，第 2.4.1 節で探索空間の冗長性に関する問題，第 2.4.2 節で突然変異に関する問題，第 2.4.3 節で集団の多様維持に関する問題について説明する．

2.4.1　探索空間の冗長性に関する問題

AutoML-Zero では，アルゴリズムの評価をする際に，$\mathcal{T}_{\text{search}}$ に含まれる全ての機械学習タスクを解く必要があり多くの時間を要するため，冗長な探索空間は探索効率を大きく低下させる要因になる．既存手法には，変数名の違いによる冗長性，命令の実行順の違いによる冗長性，非妥当なアルゴリズムによる冗長性の 3 つの冗長性が存在すると考えられる．本節では，それぞれの冗長性について詳しく説明する．

変数名の違いによる冗長性

既存手法には, 変数名の違いによる冗長性が存在すると考えられる. 例えば, アフィン回帰と解釈可能なアルゴリズムである Code.2.1 において, $s6$ という変数が $s8$ に変わったり, $v6$ が $v1$ に変わっても実行結果は同じである. 既存手法では, 実行結果に違いがなく, 変数名が異なるアルゴリズムを区別して探索しているため, 探索空間が冗長になっていると考えられる.

命令の実行順の違いによる冗長性

既存手法には, 命令の実行順の違いによる冗長性が存在すると考えられる. 例えば, アフィン回帰と解釈可能なアルゴリズムである Code.2.1 において, 12 行目と 13 行目が入れ替わっても実行結果は同じである. 既存手法では, 実行結果に違いがなく, 命令の順序が異なるアルゴリズムを区別して探索しているため, 探索空間が冗長になっていると考えられる.

非妥当なアルゴリズムによる冗長性

既存手法は, 機械学習アルゴリズムとして妥当ではないアルゴリズムが探索対象となっており, 探索空間が冗長になっているという問題が存在する. ここで, 妥当ではないアルゴリズムとは, 以下の妥当なアルゴリズムの条件のいずれかを満たさないアルゴリズムである.

1. Learn 関数で逐次更新される全ての学習パラメータが以下の条件を満たすこと.
   (a) 1 ステップ前の自分自身の値に依存して更新されていること
   (b) Predict 関数で利用されている場合は, 正解ラベル $s0$ に依存して更新されていること.
   (c) Predict 関数で利用されている場合は, 予測ラベル $s1$ に依存して更新されていること.
2. Predict 関数で予測ラベル $s1$ の算出に, 以下が利用されていること.
   (a) 入力ベクトル $v0$
   (b) 1 つ以上の Learn 関数で逐次更新される学習パラメータ
3. 全ての変数が最終的に予測ラベル $s1$ の算出に寄与すること.

これらの条件は、機械学習タスクを高い適合度で解くために、タスクの種類によらず普遍的に必要不可欠な性質である。そのため、妥当なアルゴリズムの条件を満たさないアルゴリズムを探索することは非効率的である。しかし、既存手法ではこれらの条件を考慮せずに, 全てを探索しているため、探索空間に冗長性が生じていると考えられる。

妥当なアルゴリズムの例を Code.2.2 に, 既存手法で探索対象となっている妥当ではないアルゴリズムの例と妥当ではない理由を Code.2.3 から 2.9 に示す.

Code. 2.2　妥当な機械学習アルゴリズムの例

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v1, v0)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v1 + v2
```

Code. 2.3　妥当なアルゴリズムの条件 4 を満たさない非妥当なアルゴリズム

```
def Setup():
  s2 = 0.01

def Predict():
  s2 = dot(v1, v0)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v1 + v2
```

Code. 2.4　妥当なアルゴリズムの条件 2 の (a) を満たさない非妥当なアルゴリズム

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v1, v3)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v1 + v2
```

Code. 2.5　妥当なアルゴリズムの条件 2 の (b) を満たさない非妥当なアルゴリズム

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v3, v0)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v1 + v2
```

Code. 2.6　妥当なアルゴリズムの条件 1 の (a) を満たさない非妥当なアルゴリズム

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v1, v0)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v0 + v2
```

Code. 2.7　妥当なアルゴリズムの条件 1 の (b) を満たさない非妥当なアルゴリズム

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v1, v0)

def Learn():
  s3 = s2 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v1 + v2
```

Code. 2.8　妥当なアルゴリズムの条件 1 の (c) を満たさない非妥当なアルゴリズム.

```
1    def Setup():
2      s2 = 0.01
3
4    def Predict():
5      s1 = dot(v1, v0)
6
7    def Learn():
8      s3 = s0 - s2
9      s3 = s2 * s3
10     v2 = s3 * v0
11     v1 = v1 + v2
```

Code. 2.9　妥当なアルゴリズムの条件 3 を満たさない非妥当なアルゴリズム

```
1    def Setup():
2      s2 = 0.01
3
4    def Predict():
5      s1 = dot(v1, v0)
6
7    def Learn():
8      s3 = s0 - s1
9      s4 = s2 * s3
10     v2 = s3 * v0
11     v1 = v1 + v2
```

2.4.2　突然変異に関する問題

既存手法の突然変異には, 2 つの問題が存在すると考えられる. 一つは, 親個体が妥当なアルゴリズムであっても, 既存手法の突然変異によって非妥当なアルゴリズムに変化してしまう確率が高い点である. もう一つは突然変異の局所性に関する課題である.

非妥当なアルゴリズムに高確率で突然変異する問題

既存手法では, 親個体が妥当なアルゴリズムであったとしても, 妥当ではないアルゴリズムに変化させてしまう妥当ではない突然変異が頻発する問題が存在する. 実際, 予備実験の結果により, 妥当な線形回帰アルゴリズムを突然変異させると, 98% 以上の確率で妥当ではないアルゴリズムに変化することが確かめられている. これによって, 妥当ではないアルゴリズムが大量に評価されてしまい, 探索効率が低下していると考えられる.

突然変異の局所性に関する問題

既存手法では, (1) の単一命令の追加・削除や (3) の変数アドレスの変更といった局所的な突然変異と (2) の関数全体をランダムに変更する破壊的な突然変異が実装されている. 局所的な突然変異のみでは, アルゴリズム構造を大きく変更することはない一方で, 十分な探索空間を探索することが難しい. しかしながら, (2) のような関数内のアルゴリズムを全て変える破壊的な突然変異は, 関数中の良質な部分命令列が存在していたとしても, 子個体にその良質な部分命令列が継承ができなくなってしまう可能性が高い. 故に, 既存手法の子個体生成方法では, 関数中の良質な部分命令列を保存しつつ, 関数を逐次改善することが難しいと考えられる.

2.4.3　集団の多様性維持に関する問題

既存手法には, 多様性維持に関する問題として, 世代交代モデル RE の問題, 集団に同一個体が存在する問題, 個体の希少性が無視される問題が存在すると考えられる. 多様性維持の困難さにより, 既存手法は探索序盤で発見された局所最適なアルゴリズムに早期収束してしまい, 探索効率の低下を招いていると考えられる.

世代交代モデル RE に関する問題

既存手法の RE-AutoML-Zero で採用されている世代交代モデルである RE は, 集団の多様性を失いやすいと考えられる. 佐藤らの論文 [10] の考察を踏まえると, RE が集団の多様性を低下させる要因として, 主に以下の 3 点が考えられる.

1. Fig.2.1 の STEP1 (Algorithm 2 の 11 行目) において, 無条件で元の集団の個体が淘汰されている点
2. Fig.2.1 の STEP2 (Algorithm 2 の 12 行目) における複製選択で, 強い選択圧が掛かるトーナメント選択が用いられている点
3. Fig.2.1 の STEP3 および STEP4(Algorithm 2 の 13 行目から 19 行目) における生存選択で選ばれる個体が淘汰される個体と無関係である点

1 つめは, 希少な構造を持つ個体も無条件で淘汰されることを意味しており, 多様性の低下を招くと考えられる. 2 つめは, 探索序盤で得られた局所最適解に対しても選択圧が強く掛かることを意味しており, 十分な探索をする前に多様性を低下させる恐れがある. 3 つめは, 淘汰される個体の形質が次世代に引き継げなくなることを意味しており, 集団の多様性を低下させる要因になると考えられる. 実際, これらの要因によって, 集団の多様性維持が困難になり, 探索に要す評価回数が増加したり, 局所最適解に陥りやすくなることが佐藤らの論文 [10] でも示されている.

集団に同一個体が存在する問題

既存手法の RE-AutoML-Zero は, 既に集団内に存在する個体と同等の個体が初期個体や子個体生成時に追加される問題が存在する. 集団内に同等のアルゴリズムが追加されると, 相対的に希少な構造を持つ個体が淘汰される可能性が高まり, 集団の多様性が維持しにくくなると考えられる.

個体の希少性が考慮されていない問題

既存手法では, 個体の希少度を考慮せずに世代交代を行なっており, 仮に改善の見込みがある良質な構造を持つアルゴリズムが生成されたとしても, 他の数が多い局所最適なアルゴリズムによって淘汰されてしまい, 集団の多様性が低下してしまうと考えられる.

第 3 章

提案手法

3.1 基本的な考え方

提案手法では, AutoML-Zero の利点である人間の事前知識や介入を最小限に抑えて機械学習アルゴリズムの探索が可能である点を保持しつつ, 探索効率の大幅な改善を目指す. 具体的には, 既存手法の問題点として挙げた探索空間の冗長性, 突然変異に関する問題, 集団の多様性維持に関する問題に対処した手法を提案する.

探索空間の冗長性については, グラフ構造によるアルゴリズムの表現（アルゴリズムグラフ）を導入することで対処する. アルゴリズムグラフを導入することで, 単純な命令列でアルゴリズムを表現している既存手法に比べて依存関係の解析が容易となる. そのため, アルゴリズムグラフに対して, 妥当なアルゴリズムの条件を規定することで, 非妥当なアルゴリズムを探索空間から除外することが可能となる. 加えて, 変数名のみが異なるアルゴリズムは, グラフ構造は同一となるので, 既存手法の問題である変数名の違いによる探索空間の冗長性も排除できると考えられる.

突然変異に関する問題には, アルゴリズムグラフを用いた突然変異によって対処する. 提案手法の突然変異では, アルゴリズムの妥当性を引き継げない非妥当な突然変異が起こらないように保証する. また, 部分グラフの再構成を用いた新たな突然変異を導入することで, 関数内の特定の変数の計算に利用する部分命令列のみを変化させる子個体生成を実現する. これによって, 突然変異による近傍探索性能が向上し, 少ない評価回数で高い適合度のアルゴリズムを発見できることが期待できる.

集団の多様性維持に関する問題には, Minimal Generation Gap (MGG) [10] による世代交代モデル, 集団内の同一個体の排除, および希少度に基づく生存選択を導入することで対処する. 佐藤らの考察を踏まえると, MGG は既存手法の RE に比べて, 集団の多様性を維持しやすい世代交代モデルと考えられる [10]. また, 集団内に同一個体や類似個体が増えてしまうと, 探索の初期段階で局所最適なアルゴリズムに陥ってしまうため, 同一個体が集団に追加されないようにした上で, 個体の希少度を考慮した世代交代を実現する. これにより, 探索が進んでいない段階で, 有望な個体を淘汰させる可能性が減少し, 局所最適なアルゴリズムに陥る問題を解決できると考えられる.

本章では, 第 3.2 節で, アルゴリズムグラフの定義と評価, 探索空間を削減するために必要となる妥当なアルゴリズムの条件, 個体の生成や世代交代に必要となるアルゴリズムグラフの深さや同一性の概念について説明する. その後, 第 3.3 節で, 提案手法における初期集団の生成方法, 第 3.4 節で既存手法の問題点に対処した突然変異を導入した子個体生成方法について説明する. 最後に, 第 3.5 節で, 集団の多様性を維持するために必要な世代交代モデル MGG, 集団内の同一個体の排除, 希少度に基づく生存選択について述べる.

3.2 アルゴリズムグラフ (AG)

本節では、提案手法の機械学習アルゴリズムの表現形式であるアルゴリズムグラフ (Algorithm Graph, AG) について論じる。AG は, 閉路を持たない順序付き有向グラフであり, 各ノードには、命令、定数値、入力ベクトル、正解ラベル、および学習過程で逐次更新される学習パラメータが割り当てられる。命令ノードの子ノードに命令の入力に対応するノードを配置することで、アルゴリズムの依存関係を表現する。AG を用いると, 変数名や実行順序の異なる同値なアルゴリズムを一意に表現でき、探索空間の冗長性を削減可能である。さらに, AG における妥当なアルゴリズムの条件を定式化し, 妥当なアルゴリズムグラフ (Valid Algorithm Graph, VAG) のみを探索対象とすることで, 非妥当なアルゴリズムに起因する冗長性も合わせて削減する。

本節では, 第 3.2.1 節で AG の定義, 第 3.2.2 節で AG の評価, 第 3.2.3 節で AG で削減できる探索空間の冗長性, 第 3.2.4 節で VAG について述べる. その後, 第 3.2.5 節で, VAG の個体を生成するために必要となるノードの NLP 媒介数を定義する. 最後に, 第 3.2.6 節で提案手法の世代交代モデルで必要となる AG の同一性と構造的同一性について述べる.

3.2.1 AG の定義

本論文では, アルゴリズムグラフを非巡回順序付き有向グラフ $G = (U, E)$ として定式化する. ここで, U はノードの集合, $E \subset U \times \mathbb{N} \times U$ は順序エッジの集合である. 順序エッジ (u_1, n, u_2) は通常の有向グラフのエッジとは異なり, 始点 u_1 と終点 u_2 の他に, u_1 の何番目のエッジであるかを表す順序値 $n \in \mathbb{N}$ を持つ. 非巡回順序付き有向グラフの詳細は, 付録 A を参照されたい.

アルゴリズムグラフ $G = (U, E)$ のノード集合 U に属する各ノードは, 以下の 4 つの型のいずれかに分類される.

定数ノード
: 定数値を表すノードでスカラー, ベクトル, 行列のいずれかの定数値が格納されている.

データノード
: 学習データや検証データの代入先となるノードであり, 以下の 2 つが存在する.

  入力ベクトルノード: u_{f}
  : 学習データや検証データの入力ベクトルを代入するノード. U の中にただ一つのみ存在し, u_{f} と表す. 第 2.3.1 節で述べた既存手法のアルゴリズムの表現における $v0$ に対応する.

  正解ラベルノード: u_{c}
  : 学習データの正解ラベルを代入するノード. U の中にただ一つのみ存在し, u_{c} と表す. 第 2.3.1 節で述べた既存手法のアルゴリズムの表現における $s0$ に対応する.

LP ノード: $u_{\mathrm{lp},i}$
: 学習時に逐次更新される学習パラメータ (Learning Parameter, LP) を表すノード. 学習時の最初のステップでは, 当該学習パラメータの初期値が格納されており, それ以降のステップでは, 1 ステップ前で更新された値が格納されている. 一般に複数存在するため, 順序をつけて $u_{\mathrm{lp},i}$ と表す.

命令ノード
: Table.B.1 に示した命令セット内の命令が割り当てられたノード. 命令の入力に対応するノードを子ノードとして持ち, 命令の出力値が格納されている. n 番目の子ノードが第 n 引数に対応する. また, 出力の利用用途が特殊な命令ノードとして, 以下の予測ノードと

NLP ノードが存在する.

予測ノード: u_p

出力が予測ラベルとして利用される特殊な命令ノード. ノード集合の U 内にただ一つのみ存在し, u_p と表す. 第 2.3.1 節で述べた既存手法のアルゴリズムの表現における $s1$ に対応する. また, 予測に正解ラベルを利用することはできないため, 予測ノードの子孫ノードに正解ラベルノードを含めることはできない.

NLP ノード: $u_{\mathrm{nlp},i}$

出力が次のステップの特定の学習パラメータ (Next Learning Parameter, NLP) の値として利用される特殊な命令ノード. NLP ノードは, U 内に LP ノードと同様の個数存在し, NLP ノードと LP ノードは 1 対 1 で対応するため, 添え字を対応させて $u_{\mathrm{nlp},i}$ と表す.

これらのノードのうち, 子ノードを持つノードは命令ノードのみであり, それ以外のノードは子ノードを持たない. そのため, 定数ノード, 入力ベクトルノード, 正解ラベルノード, LP ノードをまとめて終端ノードと呼ぶ. また, 各ノードにはスカラー型, ベクトル型, 行列型のいずれかが割り当てられており, 命令ノードの子ノードを割り当てる際は, 命令の入力の型の整合性が保たれる必要がある.

AG における 1 つの命令ノードとそのノードを始点とするエッジは, 第 2.3.1 節で述べた既存手法のアルゴリズムの表現における 1 行と対応する. 例えば, Fig.3.1 に示した命令ノード $u_{\mathrm{op}} \in U$ と 2 つのエッジ $(u_{\mathrm{op}}, 1, u_{\mathrm{c}})$ と $(u_{\mathrm{op}}, 2, u_{\mathrm{p}})$ は, Code.3.1 の 9 行目の命令 $s4 = s0 - s1$ と対応しており, u_{op} は $s4$, $s0$ は正解ラベルノード u_{c}, $s1$ は予測ノード u_{p} に対応する.

アフィン回帰と解釈可能なアルゴリズム Code.2.1 の AG を Fig.3.2 に示す. 図において, 命令ノードは円形, 定数ノードは四角形, データノードはひし形, LP ノードは台形で表現されている. 特殊な命令ノードである予測ノードと NLP ノードは二重丸で示した. また, 対応関係にある LP ノードと NLP ノードは同じ色で着色した. 命令ノードには, 割り当てられている命令の演算子, 定数ノードには代入されている定数値, LP ノードには学習パラメータの初期値が記載されている. 子ノードを持つノードの場合は, 図の上部にある子ノードを順序値が小さいものとする. また, アルゴリズムグラフでは不要ではあるが, Code.2.1 との対応関係を明確にするために, 各ノードには代入先の変数名を記載している.

3.2.2 AG の評価

AG は, 既存手法の命令列の形式に変換した上で, 第 2.3.2 節で述べた既存手法と同様の手法で評価を行う. 変換する際は, 定数ノードの定数の代入と LP ノードの初期値の代入を Setup 関数に割り当てて, 予測ノードおよびその子孫の命令ノードを Predict 関数に割り当てる. また, NLP ノードおよびその子孫の命令ノードのうち, Predict 関数に割り当てていないノードを Learn 関数に割り当てる. 命令ノードを割り当てる際は, 命令の実行順が付録 A に示したトポロジカルソートの逆順になるようにする. これにより, AG で表される依存関係を保持して, 既存手法の表現形式で命令の実行を行うことができる. また, 中間変数に関しては, 特殊な変数 $s0$, $s1$, $v0$ 以外のアドレスを順番に利用する.

機械学習タスクを実行する際の 1 ステップ分 (1 データ分) の処理の流れを, AG に対応させて説明する. Setup 関数実行時は, Fig.3.3 に赤色で示した定数ノードの値と LP ノードの初期値が代入される. 予測時には, Fig.3.4 で示したように, 入力ベクトルノードに入力ベクトルが代入された後で, 赤色で示した命令ノードが数字の順に実行される. 予測ラベルノード u_{p} の出力値が, 既存手法における予測ラベル $s1$ の値として利用される. 学習時には, Fig.3.5 で示したように, 正

Code. 3.1　アフィン回帰と解釈可能なアルゴリズム (Code.2.1 と同様)

```
1  def Setup():
2    s3 = 0.01 // 学習率の設定
3
4  def Predict():
5    s2 = dot(v6, v0) // 学習対象の重みと入力ベクトルの内積を計算
6    s1 = s7 + s2 // 切片を加算
7
8  def Learn():
9    s4 = s0 - s1 // 予測ラベルと正解ラベルの誤差を計算
10   s6 = s3 * s4 // 学習率の適用
11   v3 = s6 * v0 // 傾きの更新するための差分を計算
12   v6 = v6 + v3 // 傾きを更新
13   s7 = s7 + s6 // 切片を更新
```

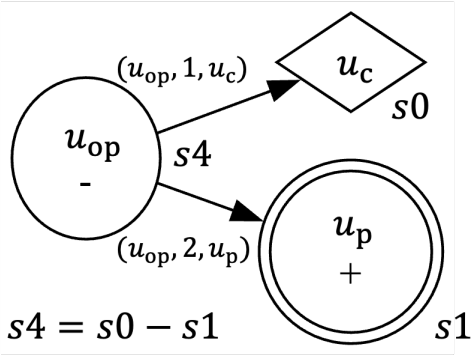

Fig.3.1　Code.3.1 における 9 行目の命令 $s4 = s0 - s1$ に対応する AG の命令ノード．u_{op} は $s4$, $s0$ は正解ラベルノード u_{c}, $s1$ は予測ノード u_{p} に対応する．後述する Fig.3.2 の AG の例と整合性を取るためにノードを表す図形を区別して利用している．また, 対応関係を分かりやすくするために, 変数名をノードに併記している．

解ラベルノードに正解ラベルが代入された後で, 赤色で示した命令ノードが数字の順に実行される．各 NLP ノードの出力値は, 次のステップの対応する LP ノードの値として利用される．

3.2.3　AG による冗長性の削減

本節では, AG によって削減可能な探索空間の冗長性について説明する．AG を導入することで, 既存手法の探索空間の冗長性の問題のうち, 変数名の違いによる冗長性と命令の実行順の違いによる冗長性に対処することができる．既存手法では, 実行結果が同等で変数名のみが異なるアルゴリズムが区別されて扱われていた．一方で, 提案手法の AG には変数名の情報は含まれていないため, 実行結果が同等で変数名のみが異なるアルゴリズムは同一視することができる．また, 既存手法では実行結果が同等で実行順序が異なるアルゴリズムがトポロジカルソートの数分だけ区別されて扱われてしまう．一方で, 提案手法の AG では依存関係のみを扱っており, 評価時にトポロジカルソートを 1 つ選んで実行するため, 実行結果が同等で実行順序が異なるアルゴリズムを同

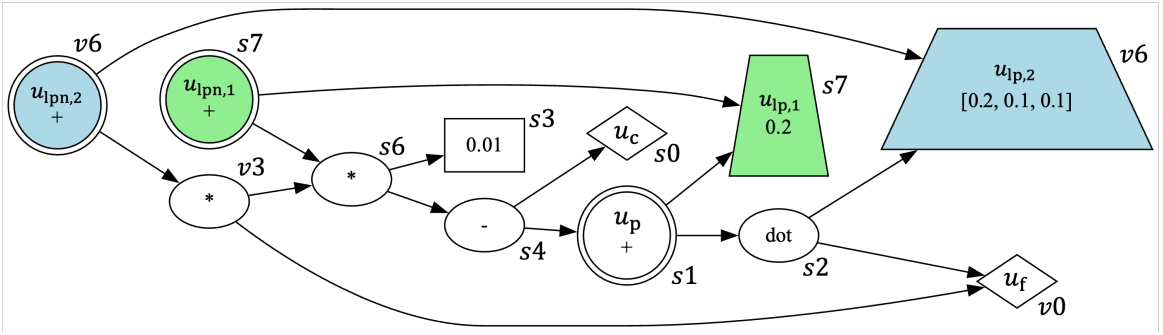

Fig.3.2　アフィン回帰と解釈可能なアルゴリズム Code.3.1 の AG. 命令ノードは円形, 定数ノードは四角形, データノードはひし形, LP ノードは台形で表現されている. 特殊な命令ノードである予測ノードと NLP ノードは二重丸で示した. また, 対応関係にある LP ノードと NLP ノードは同じ色で着色した. 命令ノードには, 割り当てられている命令の演算子, 定数ノードには代入されている定数値, LP ノードには学習パラメータの初期値が記載されている. 子ノードを持つノードの場合は, 図の上部にある子ノードを順序値が小さいものとする. また, AG では不要ではあるが, Code.3.1 との対応関係を明確にするために, 各ノードには代入先の変数名を併記している.

Setup関数実行時に定数ノードの値とLPノードの初期値を設定

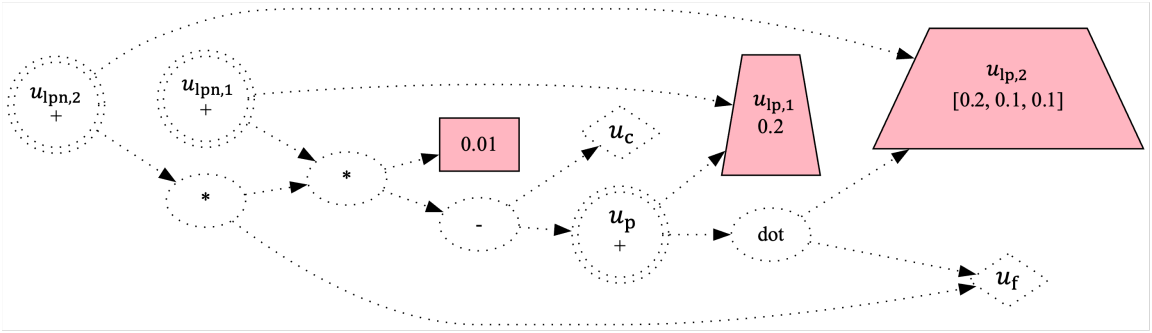

Fig.3.3　Setup 関数を実行時の AG の様子. Setup 関数では定数ノードの値と LP ノードの初期値が代入される. ピンク色で着色したノードが Setup 関数内で代入されるノードである.

一視することができる.

非妥当なアルゴリズムに関する冗長性は, 依存関係が構造化された AG に対して, 次節（第 3.2.4 節）で述べる妥当なアルゴリズムの条件を規定し 条件を満たす AG のみを探索対象とすることで削減することができる.

3.2.4　妥当なアルゴリズムグラフ (VAG)

本研究では, 以下の妥当なアルゴリズムの条件を満たすアルゴリズムグラフ $G = (U, E)$ を, 妥当なアルゴリズムグラフ (Valid Algorithm Graph, VAG) と定義する.

1. 全ての LP ノード $u_{\mathrm{lp},i} \in U$ と NLP ノード $u_{\mathrm{nlp},i} \in U$ に対して, 以下が満たされる.
   (a) $u_{\mathrm{lp},i} \in D_G(u_{\mathrm{nlp},i})$
   (b) $u_{\mathrm{lp},i} \in D_G(u_{\mathrm{p}}) \Rightarrow u_{\mathrm{c}} \in D_G(u_{\mathrm{nlp},i})$

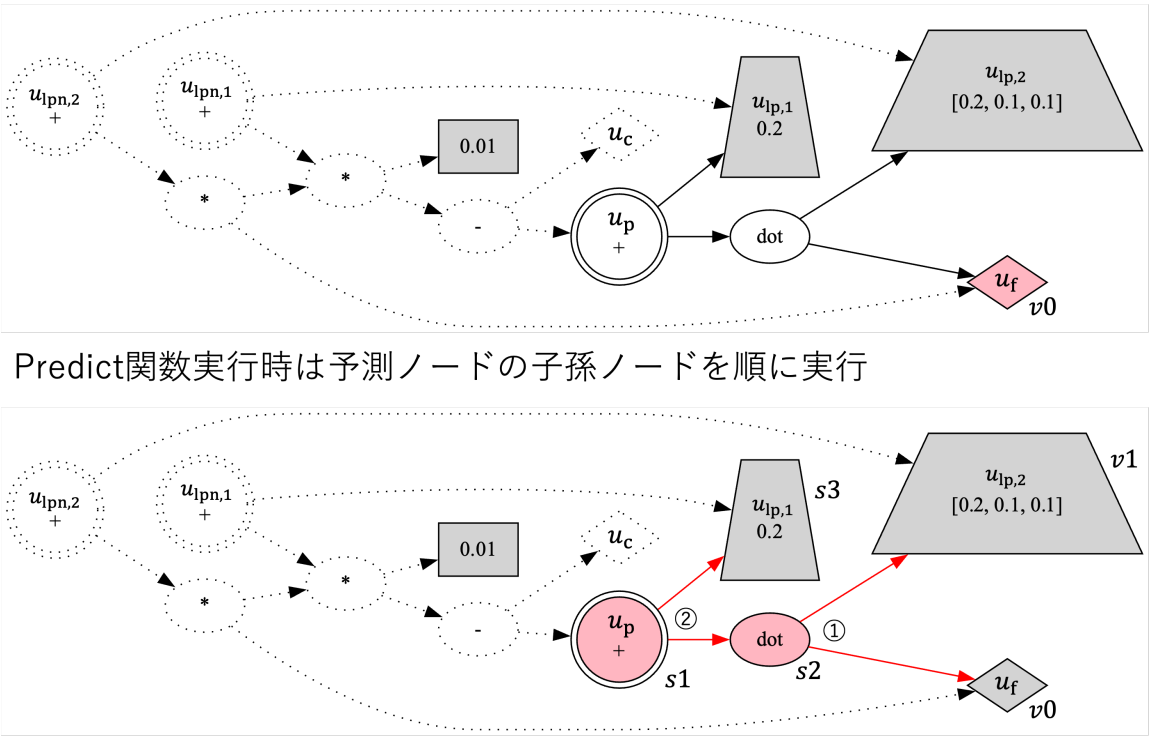

Fig.3.4　予測時の AG の様子．予測時は入力ベクトルノードに入力ベクトルが代入された後で, 着色したノードが処理される．まるで囲まれた数字は Predict 関数における実行順を意味する．予測ラベルノード u_{p} の出力値が, 既存手法における予測ラベル $s1$ の値として利用される．

(c) $u_{\mathrm{lp},i} \in D_G(u_{\mathrm{c}}) \Rightarrow u_{\mathrm{c}} \in D_G(u_{\mathrm{nlp},i})$

2. $u_{\mathrm{p}} \in U$ に関して以下が満たされる．
   (a) $v0 \in D_G(u_{\mathrm{p}})$
   (b) $\exists u_{\mathrm{lp},i}, u_{\mathrm{lp},i} \in D_G(u_{\mathrm{p}})$
3. 順序付き有向グラフ G が付録 A に示した弱連結を持ち, 任意のノード $u \notin D_G(u_{\mathrm{p}})$ に対して, 以下の条件を満たす NLP ノードの系列 $\pi : \{1, 2, 3, \cdots, m\} \to \{i \in \mathbb{N} \ : \ 1 \leq i \leq N_{\mathrm{lp}}\}$ が存在する．ここで, N_{lp} は LP ノードの総数である．
   - $u_{\mathrm{lp},\pi(1)} \in D_G(u_{\mathrm{p}})$
   - $u_{\mathrm{lp},\pi(k+1)} \in D_G(u_{\mathrm{nlp},\pi(k)}),\ k \in \mathbb{N},\ 1 \leq k \leq m-1$
   - $u \in D_G(u_{\mathrm{nlp},\pi(m)}) \vee u = u_{\mathrm{nlp},\pi(m)}$

ここで, $D_G(u)$ は u を始点として, 予測ノード u_{p} を経由点として持たない経路で到達可能なノードの集合である．つまり, $u' \in U$ であれば u を始点, u' を終点とする経路で, u_{p} を経由点として持たない経路が存在する．経路や経由点の詳細な定義は付録 A を参照されたい．本条件は, 第 2.4.1 節で述べた妥当なアルゴリズムの条件を, AG を用いて定式化したものである．条件の番号は, 既存手法で述べた妥当なアルゴリズムの条件の番号と対応している．

条件 3 については, 具体例を挙げて説明する．Fig.3.6 上に条件 3 を満たす AG, Fig.3.6 下に条件 3 を満たさない AG を示す．図において NG は, ノードグループで複数のノードの集合である．上下の 2 つのグラフの違いは, NG1 から $u_{\mathrm{lp},3}$ へのエッジが存在するかどうかである．$u_{\mathrm{lp},3}$ への

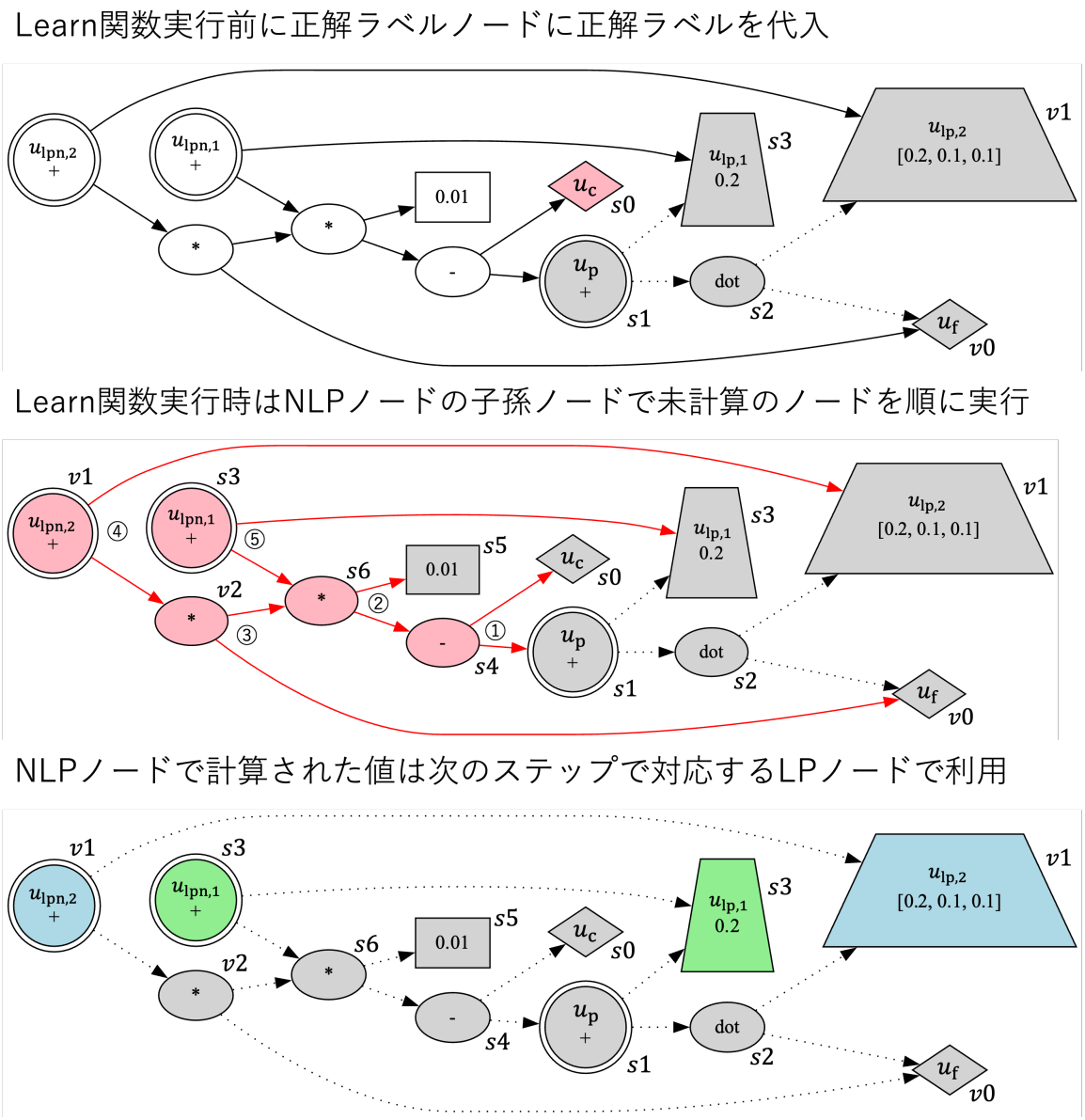

Fig.3.5　学習時の AG の様子. 学習時は正解ラベルノードに正解ラベルが代入された後で, 着色したノードが処理される. まるで囲まれた数字は Learn 関数における実行順を意味する. 各 NLP ノードの出力値は, 次のステップの対応する LP ノードの値として利用される.

エッジが存在する上の AG では条件 3 を満たす. 実際, $D_G(u_{\mathrm{p}})$ に属さない各ノードに対して, 条件 3 を満たす系列 π を構築ができる. 例えば, $\forall u \in \mathrm{NG3}$ であれば, $\pi(1) = 2, \pi(2) = 3, \pi(3) = 4$ とすれば, $u_{\mathrm{lp},2} \in D_G(u_{\mathrm{p}})$, $u_{\mathrm{lp},3} \in D_G(u_{\mathrm{nlp},2})$, $u_{\mathrm{lp},4} \in D_G(u_{\mathrm{nlp},3})$, $u \in D_G(u_{\mathrm{nlp},4})$ を満たす. また, $u_{\mathrm{nlp},2}$ であれば, $\pi(1) = 2$ とすれば, $u_{\mathrm{lp},2} \in D_G(u_{\mathrm{p}})$, $u_{\mathrm{nlp},\pi(1)} = u_{\mathrm{nlp},2}$ を満たす. しかし, NG1 から $u_{\mathrm{lp},3}$ へのエッジが存在しない Fig.3.6 下の AG では, $u_{\mathrm{nlp},2}$ から $u_{\mathrm{lp},3}$ への経路が存在しないため, $u_{\mathrm{lp},3} \notin D_G(u_{\mathrm{nlp},2})$ となり, NG3 に含まれるノードに対して条件が満たされない.

提案手法では, この妥当なアルゴリズムの条件を満たす VAG のみを探索することで, 既存手法の非妥当なアルゴリズムが探索対象となっている問題に対処する. VAG のみを探索対象とするために, 初期個体や突然変異による子個体の生成時にも, VAG のみが生成されるようにする必要が

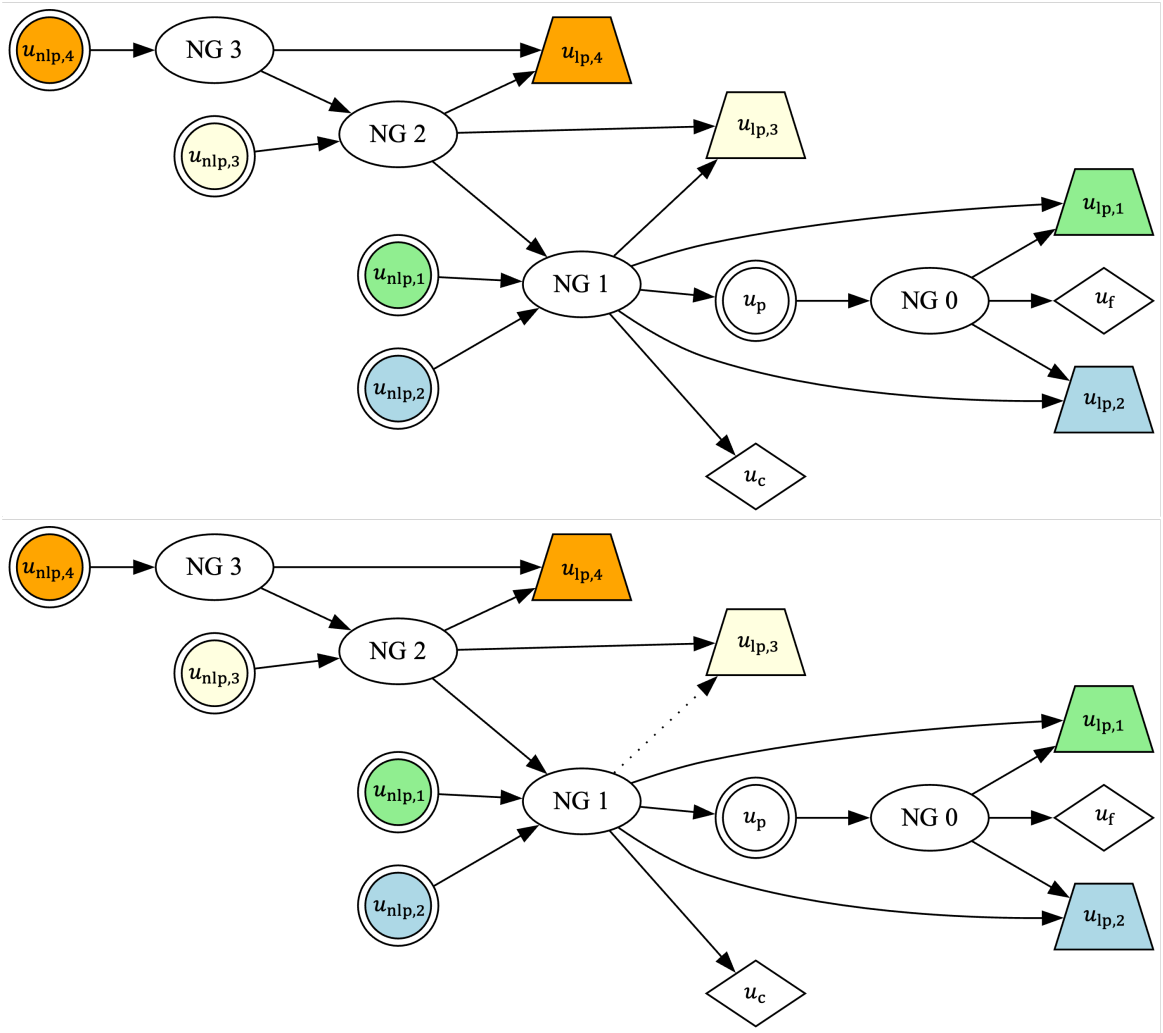

Fig.3.6　妥当なアルゴリズムの条件 3 を満たす AG（上）と満たさない AG（下）．図において NG は，ノードグループで複数のノードの集合である．両者の AG の違いは，NG1 から $u_{\mathrm{lp},3}$ へのエッジが存在するかどうかのみで，その他は同等の AG である．上のグラフでは，$\forall u \in$ NG3 に対して，$\pi(1) = 2, \pi(2) = 3, \pi(3) = 4$ とすれば，$u_{\mathrm{lp},2} \in D_G(u_{\mathrm{p}})$，$u_{\mathrm{lp},3} \in D_G(u_{\mathrm{nlp},2})$，$u_{\mathrm{lp},4} \in D_G(u_{\mathrm{nlp},3})$，$u \in D_G(u_{\mathrm{nlp},4})$ を満たす．しかし，NG1 から $u_{\mathrm{lp},3}$ へのエッジが存在しない下の AG では，$u_{\mathrm{nlp},2}$ から $u_{\mathrm{lp},3}$ への経路が存在しないため，$u_{\mathrm{lp},3} \notin D_G(u_{\mathrm{nlp},2})$ となり，NG3 に含まれるノードに対して条件が満たされない..

ある．これらの詳細は，第 3.3 節と第 3.4 節で述べる．

3.2.5　VAG のノードの NLP 媒介数

VAG のノード $u \in U$ における NLP 媒介数 $C_{\mathrm{nlp}}(u)$ を，妥当なアルゴリズムの条件 3 を満たす最小の系列の長さと定義する．ただし，u_{p} と $D_G(u_{\mathrm{p}})$ 内のノード u に対しては，$C_{\mathrm{nlp}}(u) = 0$ と定義する 例えば，Fig.3.6 上の VAG では，$\forall u \in$ NG3 に対しては，$\pi(1) = 2, \pi(2) = 3, \pi(3) = 4$ とすれば，$u_{\mathrm{lp},2} \in D_G(u_{\mathrm{p}})$，$u_{\mathrm{lp},3} \in D_G(u_{\mathrm{nlp},2})$，$u_{\mathrm{lp},4} \in D_G(u_{\mathrm{nlp},3})$，$u \in D_G(u_{\mathrm{nlp},4})$ であり，条件 3 を満たす最小の系列となるため，$C_{\mathrm{nlp}}(u) = 3$ となる．

3.2.6 AG の同一性と構造的同一性

本節では, AG に対する同一性と条件を弱めた構造的同一性について述べる. 2 つのアルゴリズムが同一であるとは, 定数ノード, 入力ベクトルノード, 正解ラベルノード, LP ノード, 命令ノード, 予測ノード, NLP ノードを完全に保存する同型写像が存在することである. また, 構造的に同一であるとは定数ノードの値と LP ノードの初期値のみが異なることを許容した同一性である. 本節では, それぞれの同一性について厳密に定義して, 具体例を挙げて説明する.

同一性

2 つのアルゴリズムグラフ $G^A = (U^A, E^A)$, $G^B = (U^B, E^B)$ が同一である ($G^A \equiv G^B$) とは, 以下の条件を満たす同型写像 $f : U_1 \to U_2$ が存在することと定義する.

1. f が全単射である.
2. $\forall u_1, u_2 \in U^A, \forall n \in \mathbb{N}$ に対して, $(u_1, n, u_2) \in E^A \Leftrightarrow (f(u_1), n, f(u_2)) \in E^B$ が成り立つ.
3. G^A の入力ベクトルノードを u^A_{f}, G^B の入力ベクトルノードを u^B_{f} とすると, $f(u^A_{\mathrm{f}}) = u^B_{\mathrm{f}}$ が成り立つ.
4. G^A の正解ラベルノードを u^A_{c}, G^B の正解ラベルノードを u^B_{c} とすると, $f(u^A_{\mathrm{c}}) = u^B_{\mathrm{c}}$ が成り立つ.
5. G^A の予測ノードを u^A_{p}, G^B の予測ノードを u^B_{p} とすると, $f(u^A_{\mathrm{p}}) = u^B_{\mathrm{p}}$ が成り立つ.
6. G^A の任意の定数ノード $u^A_{\mathrm{const}} \in U$ に対して以下が成立する.
   (a) $f(u^A_{\mathrm{const}})$ は G^B の定数ノードである.
   (b) u^A_{const} と $f(u^A_{\mathrm{const}})$ に割り当てられている定数値が等しい.
7. G^A の任意の命令ノード $u^A_{\mathrm{op}} \in U$ に対して以下が成立する.
   (a) $f(u^A_{\mathrm{op}})$ は G^B の命令ノードである.
   (b) u^A_{op} と $f(u^A_{\mathrm{op}})$ に割り当てられている命令が等しい.
8. G^A の任意の LP ノード $u^A_{\mathrm{lp}} \in U$ に対して以下が成立する.
   (a) $f(u^A_{\mathrm{lp}})$ は G^B の LP ノードである.
   (b) u^A_{lp} と $f(u^A_{\mathrm{lp}})$ に割り当てられている学習パラメータの初期値が等しい.
9. G^A の任意の NLP ノード $u^A_{\mathrm{nlp}} \in U$ に対して以下が成立する.
   (a) $f(u^A_{\mathrm{nlp}})$ は G^B の NLP ノードである.
   (b) NLP ノード u^A_{nlp} に対応する LP ノードを u^A_{lp} とすると, G^B において NLP ノード $f(u^A_{\mathrm{nlp}})$ に対応する LP ノードは $f(u^A_{\mathrm{lp}})$ である.

AG が同一である関係は AG 全体の集合 $\mathcal{G}$ 上の同値関係となる. すなわち, AG の同一性は反射律, 対称律, 推移律を満たす. 実際, 反射律は恒等写像、対称律は同型写像の逆写像、推移律は二つの同型写像の合成写像を考えることで容易に示される。

構造的同一性

AG の同一性から定数ノードの値と学習パラメータの初期値の同一条件を外した構造的同一性を定義する. 2 つのアルゴリズムグラフ $G^A = (U^A, E^A)$, $G^B = (U^B, E^B)$ が構造的に同一である ($G^A \equiv_{\mathrm{struct}} G^B$) とは, 同型写像の条件のうち 6 (b) と 8 (b) を除いた条件を満たす写像 $f : U_1 \to U_2$ が存在することと定義する. 構造的同一性の関係も同一性の関係と同様に, $\mathcal{G}$ 上の同値関係となる.

構造的同一性に関して, 具体例を挙げて説明する. Fig.3.7 に, Fig.3.2 の AG と構造的同一の AG を示す. 2 つの AG の違いは, 定数ノードの値, LP ノードの初期値のみである. また, Fig.3.8

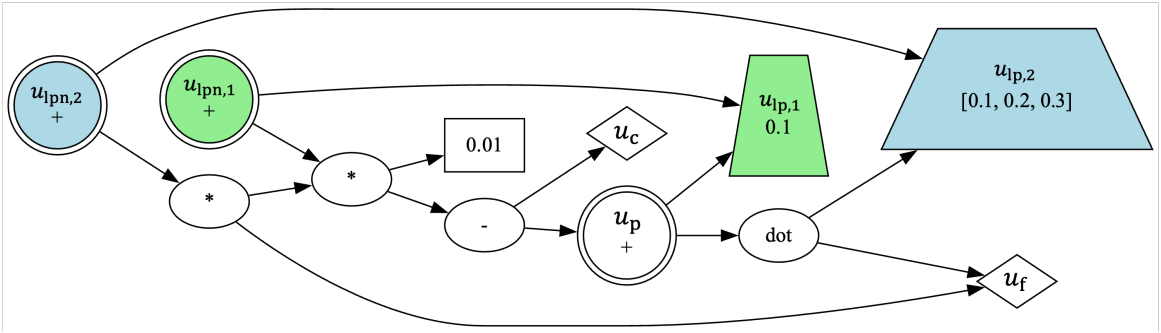

Fig.3.7　Fig.3.2 の AG と構造的同一の AG. 2 つの AG の違いは, 定数ノードの値, LP ノードの初期値のみである

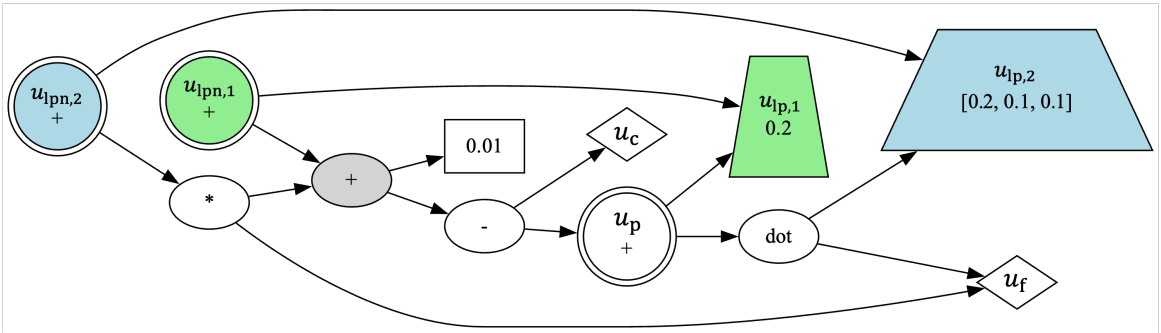

Fig.3.8　Fig.3.2 の AG と構造的同一ではない AG. 灰色で着色された命令ノードに割り当てられている命令が異なる.

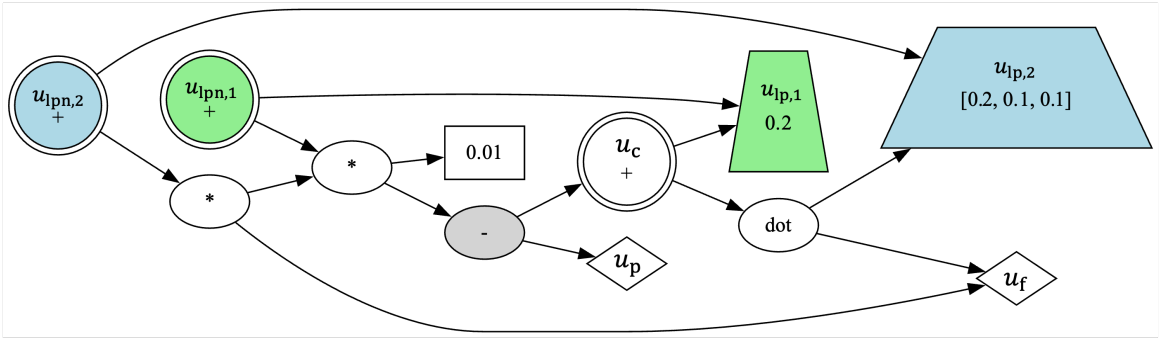

Fig.3.9　Fig.3.2 の AG と構造的同一ではない AG. 灰色で命令ノードに割り当てられている子ノードの順序が異なる.

と 3.9 に, Fig.3.2 の AG と構造的同一性を持たない AG を示す. Fig.3.8 の AG は命令ノードに割り当てられている命令が異なり, Fig.3.9 の AG は命令ノードに割り当てられている子ノードの順序が異なる.

3.3　初期個体の生成

提案手法では, VAG のみを探索対象とするために, 予測ノード u_{p} を起点として, 妥当なアルゴリズムの条件を反映させながら, グラフを段階的に構築していくことで VAG の生成を行う. 具

体的には, 予測ノード u_{p} と $D_G(u_{\mathrm{p}})$ を構築した後で, ノードの NLP 媒介数が小さい順で NLP ノード $u_{\mathrm{nlp},i}$ と $D_G(u_{\mathrm{nlp},i})$ を構築する. 本節では, 第 3.3.1 節で初期個体を生成するために必要なノードの基本操作について説明した後で, 第 3.3.2 節で u_{p} と $D_G(u_{\mathrm{p}})$ を構築する方法, 第 3.3.3 節で NLP ノード $u_{\mathrm{nlp},i}$ と $D_G(u_{\mathrm{nlp},i})$ を構築する方法について説明する.

3.3.1 ノードの基本操作

以下に, 特定のノード v とその子孫ノード $D_G(v)$ の構築をするために必要な基本操作を示す.

命令ノード構築操作 (add_op_node)
命令ノード構築操作は指定された型 type に基づいて命令セット OP から一様ランダムに命令を選択して, 入力（子ノード）に空きがある命令ノード v を構築する操作である. 構築されたノードは, アルゴリズムグラフ G に追加される. 入力はアルゴリズムグラフ G, 作成する命令ノードの型 type, 命令セット OP で, 出力は構築された命令ノード v である.

命令ノード拡張操作 (extend_op_node)
命令ノード拡張操作は, $\{v\} \cup D_G(v)$ に含まれる命令ノードのうち, 入力（子ノード）に空きがあるノードを一様ランダムに 1 つ選択し, 型が整合的な新たな命令ノードを命令ノード構築操作で構築して割り当てる操作である. 構築されたノードと接続関係は, アルゴリズムグラフ G に追加される. 入力は, アルゴリズムグラフ G, 命令ノード拡張操作の対象ノード v, 命令セット OP であり, 出力は拡張して構築された命令ノードである.

接続操作 (connect_node)
接続操作は, 命令ノード構築操作や命令ノード拡張操作で追加された命令ノード v の入力を一定確率 p_{conn} で既存の型が整合的な他のノードと一様ランダムに接続する操作である. 構築された接続関係は, アルゴリズムグラフ G に追加される. ただし, 閉路ができるようなノードや $\mathrm{depth}(v)$ よりも深いノードと接続することはできない. 入力は, アルゴリズムグラフ G, 接続対象の命令ノード v, 接続確率 p_{conn} であり, 出力は接続先のノードである.

ノード指定接続操作 (connect_specifed_node)
ノード指定接続操作は, $\{v\} \cup D_G(v)$ に含まれる命令ノードのうち, 入力（子ノード）に空きがあるノードを一様ランダムに 1 つ選択し, 指定されたノード u を強制的に割当てる操作である. ここで, 割当先の入力と u の型は整合的になるように選択される. 構築された接続関係は, アルゴリズムグラフ G に追加される. 入力は, アルゴリズムグラフ G, ノード指定接続操作の対象となるノード v, 接続するノード u であり, 出力は存在しない.

LP ノード構築操作 (add_lp_node)
LP ノード構築操作は, $\{v\} \cup D_G(v)$ に含まれる命令ノードのうち, 入力（子ノード）に空きがあるノードを一様ランダムに 1 つ選択し, 選択されたノードの入力と整合的な型を持つ LP ノードを初期化して割り当てる操作である. ここで, LP ノードを初期化する際には, 学習パラメータの初期値は標準正規分布に従ってランダムに初期化される. 構築されたノードは, アルゴリズムグラフ G に追加される. 入力は, アルゴリズムグラフ G, LP ノード構築操作の対象となるノード v であり, 出力は構築した LP ノードである.

終端接続操作 (connect_terminal_nodes)

終端接続操作は, $\{v\} \cup D_G(v)$ に含まれる命令ノードのうち, 入力に空きがあるノード（子ノード）を順に 1 つずつ選択し, 一定確率 $\hat{p}_{\text{conn}}$ でアルゴリズムグラフ G に含まれる既存の終端ノードと一様ランダムに接続する操作である. 構築された接続関係は, アルゴリズムグラフ G に追加される. 入力は, アルゴリズムグラフ G, 終端接続操作の対象となるノード v, 接続確率 $\hat{p}_{\text{conn}}$ であり, 出力は存在しない.

定数割当操作 (add_const_nodes)

定数割当操作は, $\{v\} \cup D_G(v)$ に含まれる命令ノードのうち, 入力に空きがあるノード（子ノード）を順に 1 つずつ選択し, 定数ノードを初期化して割り当てる操作である. ここで, 定数ノードを初期化する際には, スカラー型の場合は標準正規分布に従ってランダムに初期化され, ベクトル型や行列型の場合は各要素が標準正規分布に従ってランダムに初期化される. 構築された定数ノードと接続関係は, アルゴリズムグラフ G に追加される. 入力は, アルゴリズムグラフ G, 定数割当操作の対象となるノード v であり, 出力は存在しない.

3.3.2 u_{p} と $D_G(u_{\text{p}})$ の構築

$s1$ ノードの構築方法を Algorithm 3 に示す. $s1$ ノードは, 命令数 $N_{\text{OP}}^{(0)}$, 学習パラメータの最大数 $N_{\text{LP,max}}^{(0)}$, 接続確率 $p_{\text{conn}}^{(0)}$, 終端接続確率 $\hat{p}_{\text{conn}}^{(0)}$ を入力として受け取り, $s1$ 構築済みのアルゴリズムグラフ G, 深さ 0 の学習パラメータの集合 S_{LP} を出力する. 詳細なアルゴリズムの説明を以下に示す.

Algorithm 3 $s1$ ノードとその子孫ノードの構築

Input:

- 命令セット $\text{OP}^{(0)}$
- 命令数 $N_{\text{OP}}^{(0)}$
- 学習パラメータの数 $N_{\text{LP}}^{(0)}$
- 接続確率 $p_{\text{conn}}^{(0)}$
- 終端接続確率 $\hat{p}_{\text{conn}}^{(0)}$

Output:

- 深さ 0 まで構築済みのアルゴリズムグラフ $G^{(0)}$
- 深さ 0 の LP ノードの集合 $S_{\text{LP}}^{(0)}$

$V_G \leftarrow \{v0\}$
$E_G \leftarrow \emptyset$
$G^{(0)} \leftarrow (V_G, E_G)$
$S_{\text{LP}}^{(0)} \leftarrow \emptyset$
$s1 \leftarrow \text{initalize\_op\_node}(G, \text{SCALAR}, \text{OP}^{(0)})$
$\text{connect\_node}(G, s1, p_{\text{conn}}^{(0)}, \text{OP})$
for $i \leftarrow 2$ to $N_{\text{OP}}^{(0)}$ **do**
  $v \leftarrow \text{extend\_node}(G, s1, \text{OP}^{(0)})$
  $\text{connect\_node}(G, v, p_{\text{conn}}^{(0)}, \text{OP}^{(0)})$
end for

11: **if** $v0 \notin D_G(s1)$ **then**
12: connect_specified_node$(G, s1, v0)$
13: **end if**
14: **for** $i \leftarrow 1$ to $N_{\mathrm{LP}}^{(0)}$ **do**
15: $v \leftarrow$ add_lp_node$(G, s1)$
16: $S_{\mathrm{LP}}^{(0)} \leftarrow S_{\mathrm{LP}}^{(0)} \cup \{v\}$
17: **end for**
18: connect_terminal_nodes$(G, s1, \hat{p}_{\mathrm{conn}}^{(0)})$
19: add_const_nodes$(G, s1)$
20: **return** $G^{(0)}$, $S_{\mathrm{LP}}^{(0)}$

以下に, Algorithm 3 の詳細な説明を示す.

1 行目 アルゴリズムグラフのノードの集合に $v0$ を入れて初期化する.
2 行目 アルゴリズムグラフの辺の集合には何も追加せずに初期化する.
3 行目 ノードと辺からアルゴリズムグラフを初期化する.
4 行目 深さが 0 の LP ノードの集合を空集合で初期化する.
5 行目 初期化操作によって $s1$ ノードを作成する.
6 行目 作成された $s1$ ノードに対して接続操作を適用する.
7-10 行目 所与の命令数になるまで命令ノード拡張操作と接続操作を繰り返す.
11-13 行目 予測ラベル $s1$ に $v0$ が使われていない場合は, ノード指定接続操作で $v0$ を接続する.
14-17 行目 $N_{\mathrm{LP}}^{(0)}$ 個の学習パラメータを LP ノード構築操作で終端ノードに追加する.
18 行目 終端接続操作によって, 空きがあるノードを一定確率で既存の終端ノードと接続する.
19 行目 終端接続操作を適用後にも埋まらなかったノードを定数割当操作によって, 定数ノードで埋める.
20 行目 NLP ノードの構築ステップで利用するために, 深さ 0 まで構築したアルゴリズムグラフと LP ノードの集合を返却する.

3.3.3 NLP ノードとその子孫ノードの構築

NLP ノードとその子孫ノードの構築は, $S_{\mathrm{LP}}^{(k)} = \emptyset$ になるまで繰り返す. 深さ k の NLP ノードとその子孫ノードの構築方法を Algorithm 4 に示す.

Algorithm 4 深さ k の NLP ノードとその子孫ノードの構築

Input:

- 命令セット $\mathrm{OP}^{(k)}$
- 命令数 $N_{\mathrm{OP}}^{(k)}$
- 学習パラメータの数 $N_{\mathrm{LP}}^{(k)}$
- 接続確率 $p_{\mathrm{conn}}^{(k)}$
- 終端接続確率 $\hat{p}_{\mathrm{conn}}^{(k)}$
- 深さ $k-1$ まで構築済みのアルゴリズムグラフ $G^{(k-1)}$
- 深さ $k-1$ の LP ノードの集合 $S_{\mathrm{LP}}^{(k-1)}$

Output:

- 深さ k まで構築済みのアルゴリズムグラフ $G^{(k)}$

- 深さ k の LP ノードの集合 $S_{\mathrm{LP}}^{(k)}$

1: $G^{(k)} \leftarrow G^{(k-1)}$
2: $S_{\mathrm{NLP}}^{(k)} \leftarrow \emptyset$
3: $S_{\mathrm{LP}}^{(k)} \leftarrow \emptyset$
4: **for** $\mathrm{LP}_j \in S_{\mathrm{LP}}^{(k-1)}$ **do**
5: $\quad \mathrm{NLP}_j = \mathrm{initialize\_op\_node}(G, \mathrm{type}(\mathrm{LP}_j), \mathrm{OP}^{(k)})$
6: $\quad \mathrm{connect\_node}(G, \mathrm{NLP}_j, p_{\mathrm{conn}}^{(k)}, \mathrm{OP}^{(k)})$
7: $\quad S_{\mathrm{NLP}}^{(i)} \leftarrow S_{\mathrm{NLP}}^{(k)} \cup \{\mathrm{NLP}_j\}$
8: **end for**
9: **for** $i \leftarrow N_{\mathrm{LP}}^{(k-1)}$ to $N_{\mathrm{OP}}^{(k)}$ **do**
10: $\quad \mathrm{NLP} \leftarrow \mathrm{random}(S_{\mathrm{NLP}}^{(k)})$
11: $\quad v \leftarrow \mathrm{extend\_node}(G, \mathrm{NLP}, \mathrm{OP}^{(k)})$
12: $\quad \mathrm{connect\_node}(G, v, p_{\mathrm{conn}}^{(k)}, \mathrm{OP}^{(k)})$
13: **end for**
14: **for all** $\mathrm{NLP}_j \in S_{\mathrm{NLP}}^{(k)}$ **do**
15: $\quad$ **if** $\mathrm{LP}_j \notin D_G(\mathrm{NLP}_j)$ **then**
16: $\quad\quad \mathrm{connect\_specified\_node}(G, \mathrm{NLP}_j, \mathrm{LP}_j)$
17: $\quad$ **end if**
18: $\quad$ **if** $k = 1$ and $s1 \in\notin D_G(\mathrm{NLP}_j)$ **then**
19: $\quad\quad \mathrm{connect\_specified\_node}(G, \mathrm{NLP}_j, s1)$
20: $\quad$ **end if**
21: $\quad$ **if** $k = 1$ and $s0 \notin D_G(\mathrm{NLP}_j)$ **then**
22: $\quad\quad \mathrm{connect\_specified\_node}(G, \mathrm{NLP}_j, s0)$
23: $\quad$ **end if**
24: **end for**
25: **for** $i \leftarrow 1$ to $N_{\mathrm{LP}}^{(k)}$ **do**
26: $\quad \mathrm{NLP} \leftarrow \mathrm{random}(S_{\mathrm{NLP}}^{(k)})$
27: $\quad v \leftarrow \mathrm{add\_lp\_node}(G, \mathrm{NLP})$
28: $\quad S_{\mathrm{LP}}^{(k)} \leftarrow S_{\mathrm{LP}}^{(k)} \cup \{v\}$
29: **end for**
30: **for all** $\mathrm{NLP}_j \in S_{\mathrm{NLP}}^{(k)}$ **do**
31: $\quad \mathrm{connect\_terminal\_nodes}(G, \mathrm{NLP}_j, \hat{p}_{\mathrm{conn}}^{(k)})$
32: **end for**
33: **for all** $\mathrm{NLP}_j \in S_{\mathrm{NLP}}^{(k)}$ **do**
34: $\quad \mathrm{add\_const\_nodes}(G, \mathrm{NLP}_j)$
35: **end for**
36: **return** $G^{(k)}$, $S_{\mathrm{LP}}^{(k)}$

Algorithm 4 の詳細な説明を以下に示す.

1 行目 深さ $k-1$ まで構築されたグラフを引き継ぐ.
2 行目 深さ k の NLP ノードの集合 $S_{\mathrm{NLP}}^{(k)}$ を空集合で初期化する.
3 行目 深さ k の LP ノードの集合 $S_{\mathrm{LP}}^{(k)}$ を空集合で初期化する.
4-7 行目 深さ $k-1$ の各 LP ノードに基づいて NLP ノードを初期化操作で構築する. ただし, $\mathrm{type}(\mathrm{LP}_j)$ は LP_j の型を意味する.

9-13 行目　random 関数で深さ k の NLP ノードから一様ランダムに取得した上で, 命令ノード拡張操作と接続操作を繰り返す.

14 行目-24 行目　各 NLP ノードに対して必要な依存関係が存在しない場合にノード指定接続操作を用いて追加する. 妥当なアルゴリズム条件より, NLP ノードは原則自分自身を子孫ノードとして持つ必要があり, 深さ 1 の場合は, $s0$, $s1$ も子孫ノードに存在する必要がある.

25-29 行目　所与の学習パラメータの個数になるまで, random 関数で深さ k の NLP ノードから一様ランダムに取得した上で LP ノード構築操作を繰り返す.

30-32 行目　終端接続操作によって, 空きがある NLP ノードの子孫ノードを一定確率で既存の終端ノードと接続する.

33-35 行目　終端接続操作を適用後にも埋まらなかったノードを定数割当操作によって, 定数ノードで埋める.

36 行目　深さ $k+1$ の NLP ノードの構築ステップで利用するために, 深さ k まで構築したアルゴリズムグラフと LP ノードの集合を返却する.

3.4　突然変異による子個体生成

提案手法では, 既存手法の突然変異の問題点に対処するために, 本節で解説する VAG を用いた 7 つの突然変異操作を提案する. 突然変異を行う際には, 所与の選択確率に従って, これらの突然変異操作のいずれかを適用する. 提案手法は既存手法とは異なり, 非妥当な突然変異が起こらないように設計する. また, アルゴリズムグラフの部分グラフを用いた突然変異により, 特定の変数の計算に必要な部分命令列を局所的に変更することが可能となる. これにより, 良質なアルゴリズムの部分構造を維持しつつ, 有望な探索空間を効率的に探索することが期待される.

3.4.1　定数ノードの値の変更

定数ノードの変更の突然変異では, 定数ノードに割り当てられている値 x を, 以下の式に基づいて x' に更新する.

$$x' = \begin{cases} -x & \text{with probability } p_{\text{flip}} \\ x \exp(N(0,1)) & \text{with probability } 1 - p_{\text{flip}} \end{cases}$$

ここで, p_{flip} は符号反転を行う確率, $N(0,1)$ は標準正規分布に従う乱数を意味する. 定数ノードがベクトルや行列の場合は, 各要素に対して独立に更新を行う. また, 本更新式は既存手法の RE-AutoML-Zero で利用されている突然変異における定数値の更新と同じ計算式を利用している.

3.4.2　LP ノードの初期値の変更

LP ノードの初期値の変更の突然変異では, LP ノードに割り当てられている学習パラメータの初期値を, 定数ノードの値の変更と同様の更新式に基づいて更新する.

3.4.3　定数ノードを接続に置き換え

定数ノードを接続に置き換える突然変異では, 定数ノードと繋がっている辺を除去して, 別のノードと接続する変更を行う. 定数ノードを接続に置き換える突然変異の具体例を Fig.3.10 に示す. この例では, op1 ノードと 0.01 が設定されている定数ノードの間の辺を外して, op1 を op3 に接続する変化を示している.

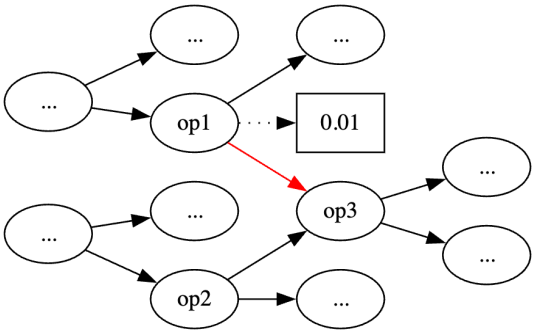

Fig.3.10　定数ノードを接続に置き換える突然変異の具体例．この例では, op1 ノードと 0.01 が設定されている定数ノードの間の辺を外して, op1 を op3 に接続する変化を示している．

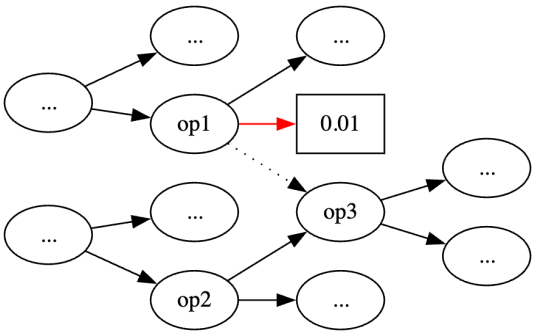

Fig.3.11　接続を定数ノードに置き換える突然変異の例．この例では, op1 ノードと複数の親ノードを持つ op3 ノードの間の辺を外して, op1 の子ノードに 0.01 が設定されている定数ノードを割り当てている．

3.4.4　接続を定数ノードに置き換え

接続を定数ノードに置き換える突然変異は, 複数の親ノードを持つノード v とその親ノード u を一様ランダムに選択した上で, u と v の間の辺を除去して, u の子ノードに定数ノードを割当てる操作である．ただし, 突然変異後にも妥当なアルゴリズムの条件を満たす u, v のみが選択対象であることに注意されたい．また, この突然変異操作は, 定数ノードを接続に置き換える操作の逆操作であると考えることができる．接続を定数ノードに置き換える突然変異の具体例を Fig.3.10 に示す．この例では, op1 ノードと親ノードを複数持つ op3 ノードの間の辺を外して, op1 の子ノードに 0.01 が設定されている定数ノードを割り当てている．

3.4.5　接続関係の変更

接続関係を変更する突然変異は, 複数の親ノードを持つノード v とその親ノード u を一様ランダムに選択した上で, u と v の間の辺を除去して, u を別のノードと接続する操作である．ただし, 突然変異後にも妥当なアルゴリズムの条件を満たす u, v のみが選択対象であることに注意されたい．接続関係を変更する突然変異の具体例を 3.12 に示す．この例では, op1 ノードと親ノードを複数持つ op4 ノードの間の辺を外して, op1 ノードと op3 ノードを接続している．

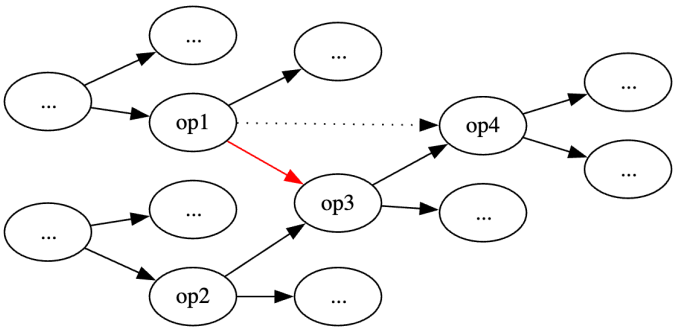

Fig.3.12　接続関係を変更する突然変異の具体例．この例では, op1 ノードと親ノードを複数持つ op4 ノードの間の辺を外して, op1 ノードと op3 ノードを接続している．

3.4.6　部分グラフの再構築

部分グラフの再構築は以下の手順で行われる．

1. 選択する部分グラフのサイズを所与の確率分布に従って決定する．
2. 以下の条件を満たす部分グラフを抽出する．
   - 部分グラフが連結である．
   - 部分グラフ内にルートノードが存在する．ここでルートノードとは, 部分グラフ内の全てのノードがルートノードの子孫ノードに属することを意味する．
   - 部分グラフ内の終端ノードを除くノード数が 1 で決定された数と同じである．ここで, 部分グラフ内の終端ノードとは, 部分グラフ内に子ノードを持たないノードを意味する．
   - 部分グラフのルートノードと終端ノード以外のノードが, NLP ノードでも, $s1$ ノードでもなく, 元のアルゴリズムグラフ内のノードを含めて, 親ノードが 1 つであること．
3. 選択された部分グラフをルートノードの型と終端ノードを保持した状態で再構成する．再構築はルートノードの命令を型を維持した状態で一様ランダムに変更した上で, 命令ノード拡張操作で 1 で決定したサイズになるまで命令ノードを追加し, 元の終端ノードを全てノード指定接続操作で追加することで実現される．

部分グラフの再構築を具体例を用いて説明する．サイズが 2 の部分グラフを, Fig.3.13 の灰色の領域に示した．$s6$ が部分グラフのルートノードであり, 他の部分グラフのノードは $s6$ の子孫に含まれる．$s0$, $s1$, $s3$ は, 部分グラフ内に子ノードを持たないので, 部分グラフの終端ノードである．また, $s4$ は NLP ノードでも, $s1$ ノードでもなく, 親ノードは元のグラフを含めて 1 つである．そのため, Fig.3.13 において灰色で示した部分グラフは, 再構成するための部分木の条件を満たしている．この部分グラフを突然変異させた結果を Fig.3.14 に示す．突然変異の前と後で, 部分グラフのサイズ, ルートノードの型, 葉ノードに違いがないことに注意されたい．

部分グラフの再構築による突然変異は、関数内の特定変数の計算に必要な部分命令列の変更に対応する。既存手法では関数全体を書き換える突然変異が高頻度で発生し、良質な部分命令列が破壊的に変更されていた。一方、提案手法における部分グラフの再構築は、関数内の特定変数の計算に必要な部分のみを変更するため、関数内の良質な部分命令列を維持した突然変異が可能となると考えられる。

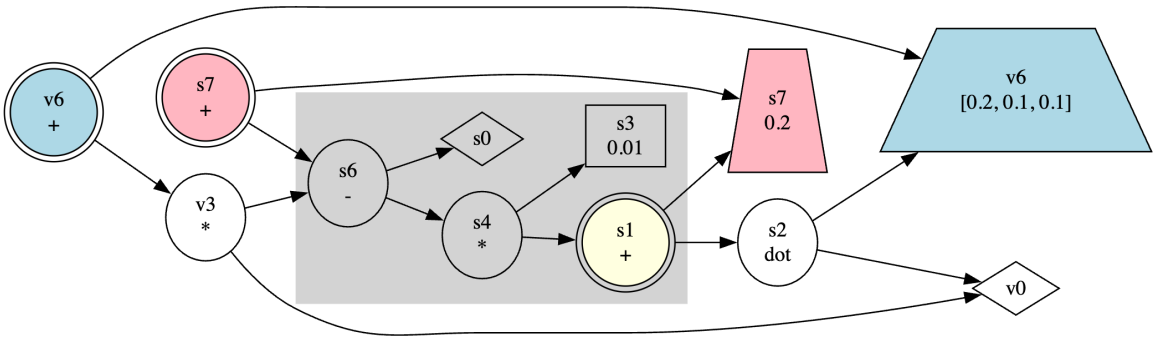

Fig.3.13　部分グラフの再構築の突然変異において選択される部分グラフの例．灰色の範囲が部分グラフの全体を意味する．本突然変異で選択する部分グラフでは，ルートノードと葉ノード以外のノードに，親を複数持つノード，NLP ノード，$s1$ ノードを含めることはできない．

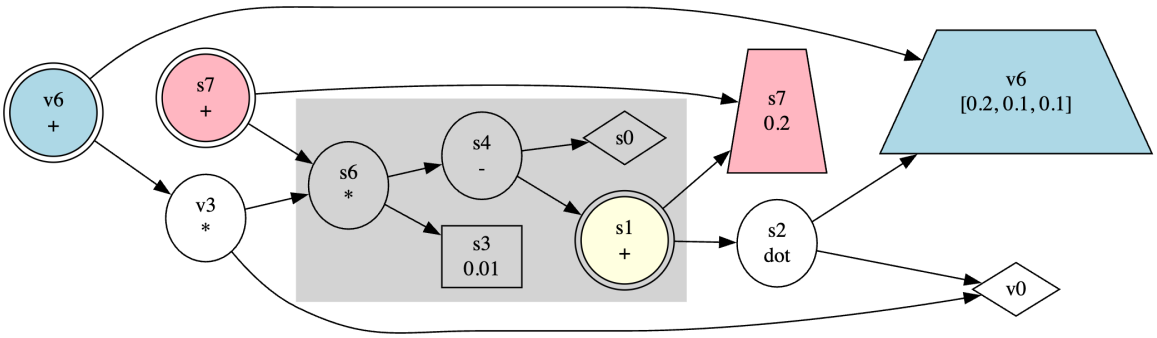

Fig.3.14　Fig.3.13 に部分グラフの再構築の突然変異を適用した例．ルートノードの型と葉ノードは突然変異前と完全に同じであることに注意されたい．

3.4.7　NLP ノードの再構築

NLP ノードを再構築する突然変異では，一様ランダムに NLP ノードを 1 つ選択し，その NLP ノードよりも深いノード，自分自身および自身の子孫ノードの一部を除去した上で，NLP ノードおよびその NLP ノードより深いノードの再構築を行う．自分自身および自身の子孫ノードの除去は，子ノードを再帰的に見ていき，$s1$ ノード，NLP ノード，親を複数持つノードが出現したら，それらのノードは除去せずに停止する．Fig.3.15 に NLP ノード $v6$ が選択された場合に，除去されるノードおよび辺を点線で示している．一連のノードを除去した後の再構築は，第 3.3.3 節で説明した NLP ノードと子孫ノードの構築を $k = \mathrm{depth(NLP)}$ から再度実行することで実現する．ただし，$k = \mathrm{depth(NLP)} + 1$ 以降は完全に同じだが，$k = \mathrm{depth(NLP)}$ では一部の処理が異なる．$k = \mathrm{depth(NLP)}$ においては，除去された NLP ノードを初期化操作により再度初期化した上で，Algorithm 4 の 9 行目から処理を開始する．ただし，random 関数では，常に再構築の対象として選択された NLP ノードが選択されるものとし，9 行目の for 文や 25 行目の for 文の最初の i の値は，既に構築済みの他のノードに基づいて設定する．

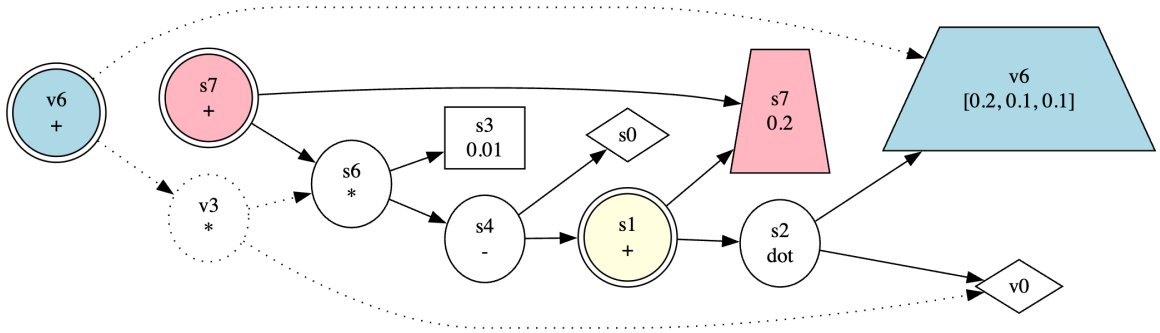

Fig.3.15　NLP ノード $v6$ の再構築をする際に除去されるノード．点線で囲まれたノードが除去されるノードであり, 点線で示された辺が除去される辺である．

3.4.8　s1 ノードの再構築

$s1$ ノードの再構築は初期個体生成と同等の操作である．一般に、機械学習アルゴリズムの予測式が変更される際には, 学習パラメータの構造が大きく変化するため、既存の NLP ノードを維持した突然変異は困難である。この再構築は破壊的な変異であるため、その選択確率を抑制し、代わりに初期集団を大きくすることが望ましい。ただし、$s1$ ノードの再構築による突然変異を完全に除外すると、深さ 0 の学習パラメータの構成変更が不可能となり、探索空間が不当に制限される可能性がある点に注意されたい。

3.5　多様性維持の工夫

3.5.1　世代交代モデル MGG の利用

提案手法では, 集団の多様性を維持しやすくするため, RE-AutoML-Zero と同様の方法で初期集団を生成した後に, Fig.3.16 に示した MGG による世代交代を繰り返し行う．各世代交代では, STEP1 で無作為に 2 つのアルゴリズム p_a, p_b を親個体として取り出し, STEP2 で p_a を突然変異させた個体を $N_{\text{children}}/2$ 個, p_b を突然変異させた個体を $N_{\text{children}}/2$ 個つくり, 合計で N_{children} 個の子個体を生成する．その後の STEP3,4 では, 生成した N_{children} 個の子個体と親個体 p_a, p_b を合わせた家族に対して生存選択で 2 つ個体を選択して集団に戻す．生存選択の詳細については, 第 3.5.3 節で説明する．集団サイズ N_{pop}, 子個体生成数 N_{children} はユーザパラメータである．

世代交代モデルを MGG に置き換えることで, 第 2.4.3 節で述べた 3 つの集団の多様性を低下させる要因に対処できると考えられる．MGG では, Fig.3.16 の STEP1 で淘汰される候補となった親個体も, Fig.3.16 の STEP3, 4 の生存選択で集団に戻される可能性があるため, 集団内の個体が無条件で淘汰されることはない．また, Fig.3.16 の STEP1 で無作為に複製選択するため, トーナメント選択のような強い選択圧がかかりにくいと考えられる．加えて, STEP3, 4 における生存選択で選ばれる個体が, 淘汰される候補である親個体の家族に限定されているため, 淘汰される個体と無関係な個体が次世代に引き継がれることが少ない．

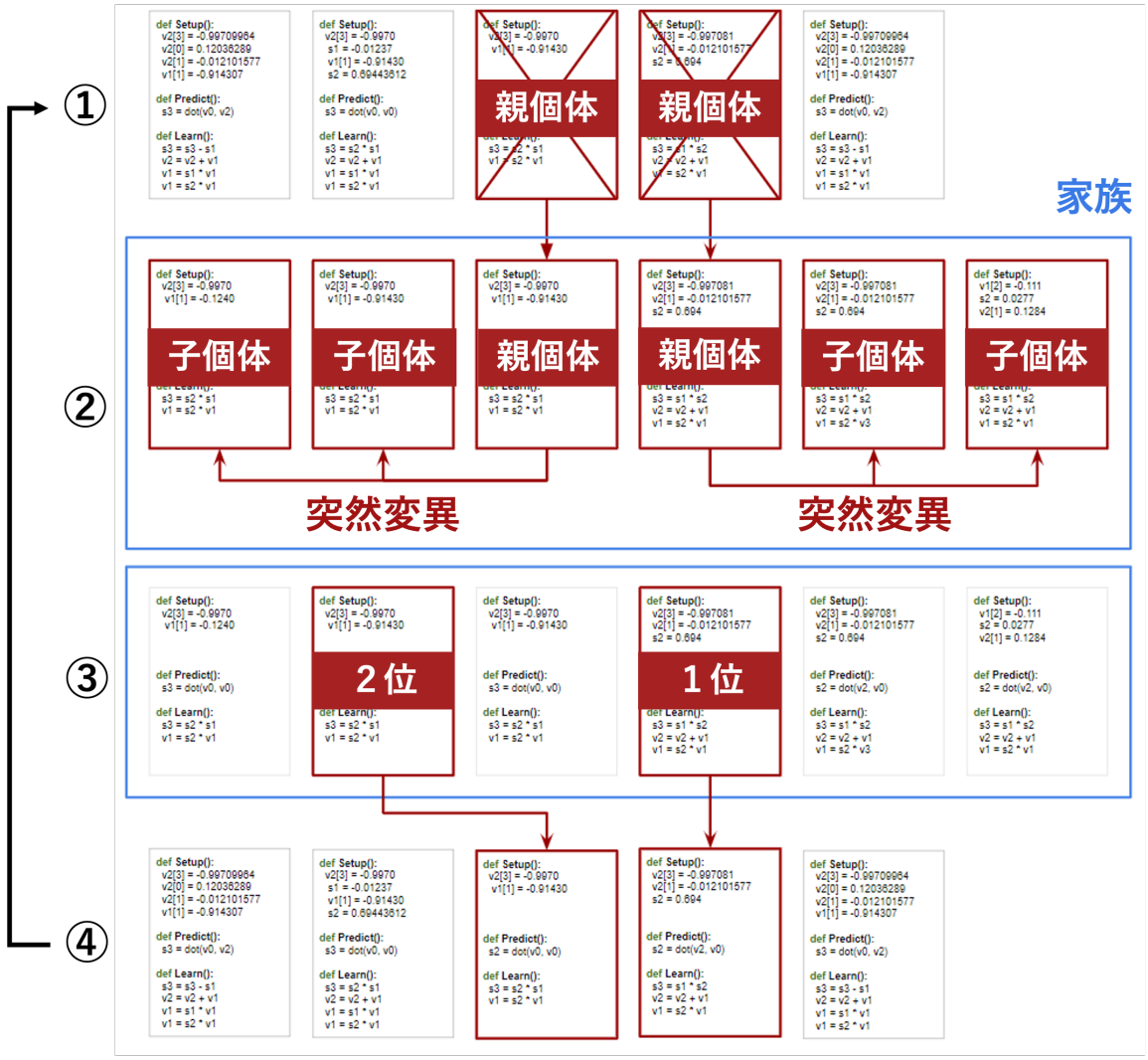

Fig.3.16　MGG による世代交代の流れ

3.5.2　集団内の同一個体の排除

本節では, 既存手法の同一のアルゴリズムが集団内に存在し, 集団の多様性が低下している問題に対する対処を説明する. 提案手法では, 同一なアルゴリズムを集団内から排除する工夫を導入する. 具体的には, 初期個体の生成時は初期集団内にグラフ構造が同一な個体が存在しないようにする. また, 子個体生成時には突然変異の特性によって, 排除するべきアルゴリズムが異なる. グラフ構造を変更する突然変異の場合は, 家族と集団に同一のグラフ構造を持つ個体が存在しないようにする. 定数ノードや LP ノードに割り当てられているパラメータを変更する突然変異の場合は, 家族と集団にグラフ構造とパラメータがどちらも同一となる個体が存在しないようにする.

提案手法で導入する同一個体の排除は, アルゴリズムの探索の特徴を踏まえると妥当であると考えられる. 一般に, グラフ構造に変化すると, 変化前のグラフ構造で最適化された定数や学習パラメータの初期値は, 無意味になることが多い. そのため, 集団ではグラフ構造を幅広く探索していきつつ, 有望なアルゴリズムの定数値や学習パラメータの初期値を最適化していくことが重要であると考えられる. 故に, 初期集団において定数値や学習パラメータの初期値のみが異なり, グ

ラフ構造が同一の個体が存在する意義は少ないと考えられる．また, 突然変異によって, グラフ構造が変わった結果, 集団内にグラフ構造が同一な個体が存在する場合は, 既に集団内に存在する個体の方が定数値や学習パラメータの初期値が最適化されていると考えられる．そのため, 集団内に存在する個体のみを残しておけば良いと考えられる．一方で, 定数値や学習パラメータの初期値が突然変異によって変わった場合は, グラフ構造, 定数値, 学習パラメータの初期値が完全に同一な個体が存在する場合以外は, 新たな有望な個体である可能性があるため, 排除することは避けるべきである．

3.5.3 希少度に基づく生存選択

多様性維持と探索効率のバランスを取る上で, MGG で利用する生存選択は非常に重要である．佐藤らの MGG の生存選択では, 家族の中で最良適合度を持つ個体とルート選択で選ばれた個体を集団に戻す方法が採用されている [10]．しかし, この生存戦略は子個体が改善する見込みが高い連続値の最適化において有力な手法であり, 子個体が改悪している可能性が高い離散最適化においては適切ではない可能性が高い．離散最適化における MGG の生存選択では, 家族の適合度の上位 2 つの個体を集団に戻す手法も一般的である．一方, この手法では初期段階で適合度の良い局所解に収束する可能性を高めてしまうという課題がある．

提案手法では, 多様性維持と探索効率のバランスを取るための生存選択手法として, 集団内の希少度を利用する方法を提案する．具体的には, 家族において最も適合度が高い個体と以下のスコア $S(a)$ が最大となる個体 $\arg\min_a S(a, \mathcal{T}_{\text{search}}, P)$ を集団に戻す．

$$S(a, \mathcal{T}_{\text{search}}, P) = (1 - w_{\text{r}})F(a, \mathcal{T}_{\text{search}}) + w_{\text{r}}R(a, P), \quad R(a, P) = \frac{1}{N_{\text{discrete}}(P, a) + 1}$$

ここで, a は個体, $F(a, \mathcal{T}_{\text{search}})$ は $\mathcal{T}_{\text{search}}$ に対するアルゴリズムの適合度, w_{r} はアルゴリズムの希少度をどの程度重要視するかの重み, $N_{\text{discrete}}(P, a)$ は集団 P において a とグラフ構造が同一の個体の数を表す．$R(a, P)$ は集団 P における個体 a の希少度を意味しており, 集団内の個体が少ないほど大きい値となる．スコア $S(a, \mathcal{T}_{\text{search}}, P)$ に基づいて, 生存選択を行うことで, 集団の多様性と探索効率のバランスを取れると期待できる．

3.5.4 詳細なアルゴリズム

Algorithm 5 に提案手法の世代交代アルゴリズムを示す．このアルゴリズムは, タスク集合 $\mathcal{T}_{\text{search}}$, 集団サイズ N_{pop}, 子個体生成数 N_{children}, 最大評価回数 N_{eval} を入力として受け取り, 探索の結果発見されたアルゴリズムを出力する．

Algorithm 5 提案手法の世代交代アルゴリズム

Input: タスク集合 $\mathcal{T}_{\text{search}}$, 集団サイズ N_{pop}, 子個体生成数 N_{children}, 最大評価回数 N_{eval}
Output: 発見されたアルゴリズム

```
eval_num ← 0
P ← ∅
for i = 1 to N_pop do
    while true do
        algorithm ← generate_algorithm_graph()
        if is_unique_structure(P, algorithm) then
            break
```

```
    end if
  end while
  fitness ← F(a, T_search)
  eval_num ← eval_num + 1
  individual ← (algorithm, fitness)
  P ← P ∪ {individual}
end for
while eval_num < N_eval do
  parent1 ← random(P)
  parent2 ← random(P)
  F = {parent1, parent2}
  P ← P \ F
  for i = 1 to N_children do
    if i ≡ 0 (mod 2) then
      parent ← parent1
    else
      parent ← parent2
    end if
    while true do
      algorithm ← copy(parent)
      (algorithm, mutation_type) ← mutate(algorithm)
      if mutation_type = STRUCTURE
          and is_unique_structure(P ∪ F, algorithm) then
        break
      else if mutation_type = PARAMETER
          and is_unique(P ∪ F, algorithm) then
        break
      end if
    end while
    fitness ← F(algorithm, T_search)
    eval_num ← eval_num + 1
    individual ← (algorithm, fitness)
    F ← F ∪ {individual}
  end for
  individual1 ← get_best_fitness(F)
  F ← F \ {individual1}
  individual2 ← get_best_score(F)
  P ← P ∪ {individual1, individual2}
end while
best_algorithm ← get_best_fitness(P)
return best_algorithm
```

Algorithm 5 の詳細な説明を以下に示す.

1 行目 評価回数のカウンターを 0 で初期化する.
2 行目 集団 P を初期化する.

3-14 **行目**　グラフ構造が同一の個体が集団内に含まれないように, 初期集団個体を生成して, 適合度を計算する.

4-9 **行目**　集団 P に存在しないグラフ構造が得られるまで, 初期個体を生成し続ける.

5 **行目**　第 3.3 説で述べた初期個体生成方法で個体を生成する関数 generate_algorithm_graph を用いてアルゴリズムグラフを生成する.

6-8 **行目**　集団 P に同一のグラフ構造をもつアルゴリズムグラフが存在する場合は個体の生成を行う. 同一のグラフ構造をもつアルゴリズムグラフの存在性は, is_unique_structure 関数によって判定される.

10 **行目**　アルゴリズムグラフを評価して適合度を計算する.

11 **行目**　評価回数のカウンターをインクリメントする.

12 **行目**　アルゴリズムグラフと適合度のペアを個体として代入する.

13 **行目**　集団に個体を追加する.

15 **行目**　評価回数の上限に到達するまで世代交代を繰り返す.

16 **行目**　1 つ目の親個体を一様ランダムに取得する.

17 **行目**　2 つ目の親個体を一様ランダムに取得する.

18 **行目**　家族の集合を 2 つの親個体で初期化する.

19 **行目**　選択された 2 つの親個体を集団から削除する.

20-39 **行目**　所与の個数に到達するまで子個体を生成する.

21-25 **行目**　各親個体から生成する子個体数を揃えられるように親個体を選択する.

26-34 **行目**　重複排除の条件が満たされるまで子個体生成を繰り返す.

27 **行目**　親個体をコピーする.

28 **行目**　第 3.4 節の突然変異による子個体生成を行う mutate 関数を実行する. mutate の返り値は, 突然変異後のアルゴリズムグラフと突然変異の種類である. 突然変異の種類は, 定数ノードの定数値の変更と LP ノードの学習パラメータの初期値の変更の場合は PARAMETER となり, それ以外の場合は STRUCTURE となる.

29-31 **行目**　グラフ構造の変更を伴う突然変異の場合は, 家族と集団に同一のグラフ構造を持つ個体が存在した場合は子個体の生成をやり直す. 同一のグラフ構造をもつアルゴリズムグラフの存在性は, is_unique_structure 関数によって判定される.

31-33 **行目**　パラメータの変更のみの突然変異の場合は, 家族と集団に完全に同一の個体が存在した場合は子個体の生成をやり直す. 完全同一のアルゴリズムグラフの存在性は, is_unique 関数によって判定される.

35 **行目**　アルゴリズムグラフを評価して適合度を計算する.

36 **行目**　評価回数のカウンターをインクリメントする.

37 **行目**　アルゴリズムグラフと適合度のペアで子個体として代入する.

38 **行目**　家族 F に作成した子個体を追加する.

40 **行目**　家族 F の中で最も適合度が高い個体を取得する.

41 **行目**　家族 F から最も適合度が高い個体を除去する.

42 **行目**　家族 F の中で最もスコア $S(a, \mathcal{T}_{\text{search}}, P)$ が高い個体を取得する関数 get_best_score を実行して, 希少度と適合度を考慮したスコアが最も高い個体を取得する.

43 **行目**　集団に最も適合度が高い個体と希少度と適合度を考慮したスコア $S(a, \mathcal{T}_{\text{search}}, P)$ が最も高い個体を戻す.

45 **行目**　集団の中で最も適合度の高い個体を取得する.

46 行目　集団の中で最も適合度の高い個体を返却する.

第 4 章

実験

4.1 目的

本実験の目的は, 回帰アルゴリズムや分類アルゴリズムの探索問題をローカル PC の少ない計算リソース上で実行し, 提案手法と既存手法の探索性能を比較をすることである. 既存手法である Esteban らの手法 RE-AutoML-Zero[9] に比べて, 提案手法の MGG-AutoML-Zero+VAG の方が, 所与の評価回数で高い適合度のアルゴリズムが得られることを確認する.

4.2 比較手法

既存手法である Esteban らの手法 RE-AutoML-Zero と提案手法の MGG-AutoML-Zero+VAG を比較する.

4.3 ベンチマーク問題

4.3.1 問題設定の概要

本実験では, Table.4.1 に示した合計 24 種類のベンチマーク問題を利用して実験を行う. 回帰問題と分類問題それぞれで, 線形, アフィン, 非線形の特性を持つ問題を用いる. また, それぞれの問題に対して, ノイズがない場合とノイズが正規分布 $N(0, 0.1)$ に従う場合の計 2 種類の問題を用意する.

Table.4.1 実験で利用する合計 24 種類のベンチマーク問題. 回帰問題と分類問題それぞれで, 線形, アフィン, 非線形の特性を持つ問題を利用. また, それぞれの問題に対して, ノイズがない場合とノイズが正規分布 $N(0, 0.1)$ に従う場合の計 2 種類の問題を用意.

問題設定		ノイズ
回帰問題	線形	なし　$N(0, 0.1)$
	アフィン	なし　$N(0, 0.1)$
	非線形	なし　$N(0, 0.1)$
分類問題	線形	なし　$N(0, 0.1)$
	アフィン	なし　$N(0, 0.1)$
	非線形	なし　$N(0, 0.1)$

Table.4.2　回帰問題のベンチマーク問題．問題設定における探索に利用するタスクの数 $|\mathcal{T}_{\text{search}}|$, 最終評価に利用するタスクの数 $|\mathcal{T}_{\text{eval}}|$, 各タスクの学習用データの数 $|D_{\text{train}}|$, 各タスクの検証データの数 $|D_{\text{valid}}|$

	線形回帰問題	アフィン回帰問題	非線形回帰問題
$\lvert\mathcal{T}_{\text{search}}\rvert$	10	10	100
$\lvert\mathcal{T}_{\text{eval}}\rvert$	10	10	100
$\lvert D_{\text{train}}\rvert$	1000	1000	1000
$\lvert D_{\text{valid}}\rvert$	100	100	100

4.3.2　タスク集合の構成

本節では, 各問題設定のタスク集合の構成方法について説明する．全ての問題設定で共通して, 探索用のタスク集合 $\mathcal{T}_{\text{search}}$ および最終評価用のタスク集合 $\mathcal{T}_{\text{eval}}$ に含まれるタスクは 10 個とした．また, タスク $T \in \mathcal{T}$ の学習データと検証用データの個数は, それぞれ 1000 個と 100 個として, エポック数は 1, 問題の次元は 4 に設定した．それぞれのタスクの入力ベクトル $\boldsymbol{x}_j$ と正解ラベル y_j の構成方法を問題設定ごとに以下に示す．以下の説明において, $\boldsymbol{v} \sim \boldsymbol{N}(N, 0)$ はベクトル $\boldsymbol{v}$ が各要素が平均 N, 分散 0 の正規分布に従うことを意味し, $U(x)$ は $x \leq 0$ の時に 0, $x > 0$ の時に 1 となるステップ関数, n_j は Table.4.1 に示したノイズの確率分布に従う変数である．

線型回帰問題

$$\boldsymbol{x}_j \sim \boldsymbol{N}(0,1), \quad y_j = \boldsymbol{w} \cdot \boldsymbol{x}_j + n_j \quad \boldsymbol{w} \sim \boldsymbol{N}(0,1)$$

アフィン回帰問題

$$\boldsymbol{x}_j \sim \boldsymbol{N}(0,1), \quad y_j = \boldsymbol{w}, \cdot \boldsymbol{x}_j + u + n_j, \quad \boldsymbol{w} \sim \boldsymbol{N}(0,1), \quad u \sim N(0,1)$$

非線形回帰問題

非線形回帰タスクは, 中間層が 1 層 4 ノードのランダムな重みのニューラルネットワークで, 中間層の出力を $\boldsymbol{v}_j = (v_{j,1}, v_{j,2}, v_{j,3}, v_{j,4})^{\mathrm{T}}$ とすると, 入力ベクトル $\boldsymbol{x}_j$ と正解ラベル y_j は以下のように構成される．

$$\boldsymbol{x}_j \sim \boldsymbol{N}(0,1), \quad v_{j,k} = \mathrm{ReLU}\left(\boldsymbol{x}_j \cdot \boldsymbol{w}_k\right), \quad y_j = \boldsymbol{v}_j \cdot \boldsymbol{w}_{\text{out}} + n_j$$
$$\boldsymbol{w}_k \sim \boldsymbol{N}(0,1), \quad \boldsymbol{w}_{\text{out}} \sim \boldsymbol{N}(0,1)$$

線型分類問題

$$\boldsymbol{x}_j \sim \boldsymbol{N}(0,1), \quad y_j = U(\boldsymbol{w} \cdot \boldsymbol{x}_j + n_j) \quad \boldsymbol{w} \sim \boldsymbol{N}(0,1)$$

アフィン分類問題

$$\boldsymbol{x}_j \sim \boldsymbol{N}(0,1), \quad y_j = U(\boldsymbol{w}\cdot\boldsymbol{x}_j + u + n_j), \quad \boldsymbol{w} \sim \boldsymbol{N}(0,1), \quad u \sim N(0,1)$$

非線形分類問題

非線形回帰タスクは, 中間層が 1 層 4 ノードのランダムな重みのニューラルネットワークで, 中間層の出力を $\boldsymbol{v}_j = (v_{j,1}, v_{j,2}, v_{j,3}, v_{j,4})^{\mathrm{T}}$ とすると, 入力ベクトル $\boldsymbol{x}_j$ と正解ラベル y_j は以下のように構成される.

$$\begin{gathered}\boldsymbol{x}_j \sim \boldsymbol{N}(0,1), \quad v_{j,k} = \mathrm{ReLU}\left(\boldsymbol{x}_j\cdot\boldsymbol{w}_k\right), \quad y_j = U(\boldsymbol{v}_j\cdot\boldsymbol{w}_{\mathrm{out}} + n_j) \\ \boldsymbol{w}_k \sim \boldsymbol{N}(0,1), \quad \boldsymbol{w}_{\mathrm{out}} \sim \boldsymbol{N}(0,1)\end{gathered}$$

4.3.3 Normalize / Loss / Rescale 関数の設定

本節では, アルゴリズムを評価する際に利用する Normalize 関数, Loss 関数, Rescale 関数の設定について説明する. 各関数の詳細は, 第 2.3.2 節を参照されたい.

回帰問題の Normalize 関数, Loss 関数, Rescale 関数の設定を以下に示す.

$$\begin{aligned}\mathrm{Normalize}(s1) &= s1 \\ \mathrm{Loss}(y, s1) &= |y - s1| \\ \mathrm{Rescale}(l) &= 1 - \frac{2}{\pi}\arctan\left(\sqrt{l}\right)\end{aligned}$$

回帰問題においては, 予測ラベルをそのまま利用し, 二乗誤差を計算するように設定している. 適合度が高いほど良いアルゴリズムであるため, Rescale 関数では, 損失関数の平均値を $[0,1]$ に変換した上で, 1 が高い適合度となるように逆転している.

分類問題の Normalize 関数, Loss 関数, Rescale 関数の設定を以下に示す.

$$\begin{aligned}\mathrm{Normalize}(s1) &= \mathrm{sigmoid}(s1) \\ \mathrm{Loss}(y, s1) &= \begin{cases}1 & |y - s1| \geq 0.5 \\ 0 & |y - s1| < 0.5\end{cases} \\ \mathrm{Rescale}(l) &= 1 - l\end{aligned}$$

分類問題においては, 予測ラベルを分類確率に変換するために, Normalize 関数に sigmoid 関数を適用している. また, Loss 関数は分類確率に基づいて分類した結果, 正しい分類であれば 0, 誤った分類であれば 1 を返すように設定している. 適合度が高いほど良いアルゴリズムであるため, Rescale 関数では 1 から誤分類率を引いた分類成功率を返すように設定している.

4.4 評価基準

本実験では, それぞれの問題設定で乱数を変えて 10 試行の実験を行い, 所与の評価回数までに発見されたアルゴリズムの適合度の平均値を評価基準とする. 第 4.3.3 説の定義より, 適合度は回帰問題の場合であれば全てのタスクの検証データに対して完全に誤差なしで回帰できた場合に 1, 分類問題であれば全てのタスクの検証データを正しく分類できた場合に 1 となる. しかし, 一般には学習データと検証データは独立であるため, 完全な適合度 1 を達成することは難しい. また, 各

Table.4.3　実験で利用する命令セット．これらの命令セットは Esteban らの Section 4.1 の実験と同様に設定している [9]．各命令の詳細は, 付録 B を参照されたい．

問題設定	関数	命令セット
線形	Setup	OP56, OP57
	Predict, $OP^{(0)}$	OP27
	Learn, $OP^{(1)}$	OP2, OP3, OP18, OP23
アフィン	Setup	OP56, OP57
	Predict, $OP^{(0)}$	OP1, OP27
	Learn, $OP^{(1)}$	OP1, OP2, OP3, OP18, OP23
非線形	Setup	OP56, OP63, OP64
	Predict, $OP^{(0)}$	OP27, OP31, OP48
	Learn, $OP^{(1)}$	OP2, OP3, OP16, OP18, OP23, OP25, OP28, OP40

実験における打ち切り適合度は 0.999, 打ち切り評価回数は線形回帰/分類の問題では 20,000 回, アフィン回帰/分類の問題では 200,000 回, 非線形回帰/分類の問題では 20,000,000 回とした．

4.5　実験設定

本節では, 既存手法と提案手法のユーザパラメータの設定について説明する．本実験は, 既存手法と提案手法の探索性能を比較することが目的なので, Esteban らの Section 4.1 の実験と同様, それぞれの問題設定に対して良質なアルゴリズムを再現する上で必要な命令に限定した [9]．具体的には, 各問題に対する命令セットについては, Table.4.3 に示した通りに限定する．各命令の詳細は, 付録 B を参照されたい．命令セット以外の既存手法と提案手法のユーザパラメータは, それぞれ Table.4.4 と Table.4.5 に示した値を用いた．既存手法における設定値は, Esteban らの Section 4.1 の実験と同じ値に設定した [9]．本実験では, 命令数が固定されているため, 命令を増減する突然変異に関しては, 既存手法および提案手法のいずれでも利用しないものとする．

4.6　実験環境

本実験は, 手元のローカル PC の環境で実行した．具体的には, メモリが 32GB で CPU は 12 コアの Apple M2 Max が搭載された MacBook Pro 2023 を利用した．また, OS は macOS Sequoia のバージョン 15.0.1, 実行環境は Java openjdk 21.0.1 を利用した．Esteban らの AutoML-Zero のように, 大規模な計算リソースを利用していない点に注意されたい．

4.7　実験結果

既存手法と提案手法で, 線型回帰アルゴリズム, アフィン回帰アルゴリズム, 非線形回帰アルゴリズムの探索を行った結果を Table.4.6 からに示す．実験の結果より, 所与の評価回数で発見できるアルゴリズムの適合度が提案手法の方が, 既存手法に比べて高いことがわかる．

Table.4.4　既存手法の回帰問題におけるハイパーパラメータの設定値．これらの設定値は Esteban らの Section 4.1 の実験と同じ値に設定している [9].

パラメータ			設定値		
名前		数式表記	線形	アフィン	非線形
集団サイズ		N_{pop}	1000	1000	1000
トーナメントサイズ		K	10	10	10
突然変異確率		p_{mutate}	0.9	0.9	0.9
アドレス数	スカラー	-	4	5	4
	ベクトル	-	3	3	8
	行列	-	1	1	2
命令数	Setup	-	5	6	21
	Predict	-	1	2	3
	Learn	-	4	6	9

Table.4.5　提案手法の回帰問題におけるハイパーパラメータの設定値.

パラメータ		設定値		
名前	数式表記	線形	アフィン	非線形
集団サイズ	N_{pop}	1000	1000	10000
子個体生成数	K	100	100	1000
命令数	$N_{\text{OP}}^{(0)}$	1	2	3
	$N_{\text{OP}}^{(1)}$	4	6	9
パラメータ数	$N_{\text{LP}}^{(0)}$	1	2	2
接続確率	$p_{\text{conn}}^{(0)}$	0.3	0.3	0.3
	$p_{\text{conn}}^{(1)}$	0.3	0.3	0.3
終端接続確率	$\hat{p}_{\text{conn}}^{(0)}$	0.3	0.3	0.3
	$\hat{p}_{\text{conn}}^{(1)}$	0.3	0.3	0.3
符号反転確率	p_{flip}	0.1	0.1	0.1
部分グラフの再構成の大きさの選択確率	$p_{\text{size}}(1)$	0.2	0.2	0.2
	$p_{\text{size}}(2)$	0.6	0.6	0.6
	$p_{\text{size}}(3)$	0.2	0.2	0.2
ルートノードの再構成の深さ選択確率	$p_{\text{depth}}(0)$	0.5	0.5	0.5
	$p_{\text{depth}}(1)$	0.5	0.5	0.5
多様性指標の重み	w_{rare}	0.4	0.4	0.4

Table.4.6　ノイズなしの回帰問題における既存手法と提案手法の性能の比較.

No.	線形回帰問題		アフィン回帰問題		非線形回帰問題	
	既存手法	提案手法	既存手法	提案手法	既存手法	提案手法
1			0.691	1.000		
2			1.000	1.000		
3			0.943	1.000		
4			0.369	1.000		
5			0.351	1.000		
6			0.448	1.000		
7			0.305	1.000		
8			0.488	1.000		
9			0.318	1.000		
10			0.948	1.000		
平均				1.00		
標準偏差				1.00		

Table.4.7　ノイズなしの分類問題における既存手法と提案手法の性能の比較.

No.	線形分類問題		アフィン分類問題		非線形分類問題	
	既存手法	提案手法	既存手法	提案手法	既存手法	提案手法
1			0.951	0.974		
2			0.744	0.975		
3			0.849	0.923		
4			0.836	0.942		
5			0.610	0.985		
6			0.833	0.982		
7			0.790	0.872		
8			0.880	0.974		
9			0.791	0.978		
10			0.815	0.961		
平均						
標準偏差						

Table.4.8　$N(0, 0.1)$ に従うノイズを付与した回帰問題における既存手法と提案手法の性能の比較.

No.	線形回帰問題		アフィン回帰問題		非線形回帰問題	
	既存手法	提案手法	既存手法	提案手法	既存手法	提案手法
1			0.366	0.939		0.936
2			0.443	0.939		0.912
3			0.428	0.935		0.884
4			0.368	0.934		0.884
5			0.358	0.935		0.930
6			0.422	0.937		0.934
7			0.306	0.936		0.936
8			0.915	0.936		0.906
9			0.315	0.936		0.898
10			0.387	0.934		
平均						
標準偏差						

Table.4.9　$N(0, 0.1)$ に従うノイズを付与した分類問題における既存手法と提案手法の性能の比較.

No.	線形分類問題		アフィン分類問題		非線形分類問題	
	既存手法	提案手法	既存手法	提案手法	既存手法	提案手法
1			0.961	0.966		
2			0.646	0.974		
3			0.865	0.966		
4			0.638	0.973		
5			0.859	0.860		
6			0.966	0.957		
7			0.962	0.969		
8			0.877	0.957		
9			0.956	0.965		
10			0.822	0.955		
平均						
標準偏差						

第 5 章

考察

5.1 実験結果について

各問題設定おける探索用のタスク集合 $\mathcal{T}_{\text{search}}$ に対する既存手法と提案手法の適合度の推移を Table.x から Table.x に示す．これらの結果より, 既存手法において探索が停滞している問題設定においても, 提案手法は段階的に適合度が改善していることがわかる．

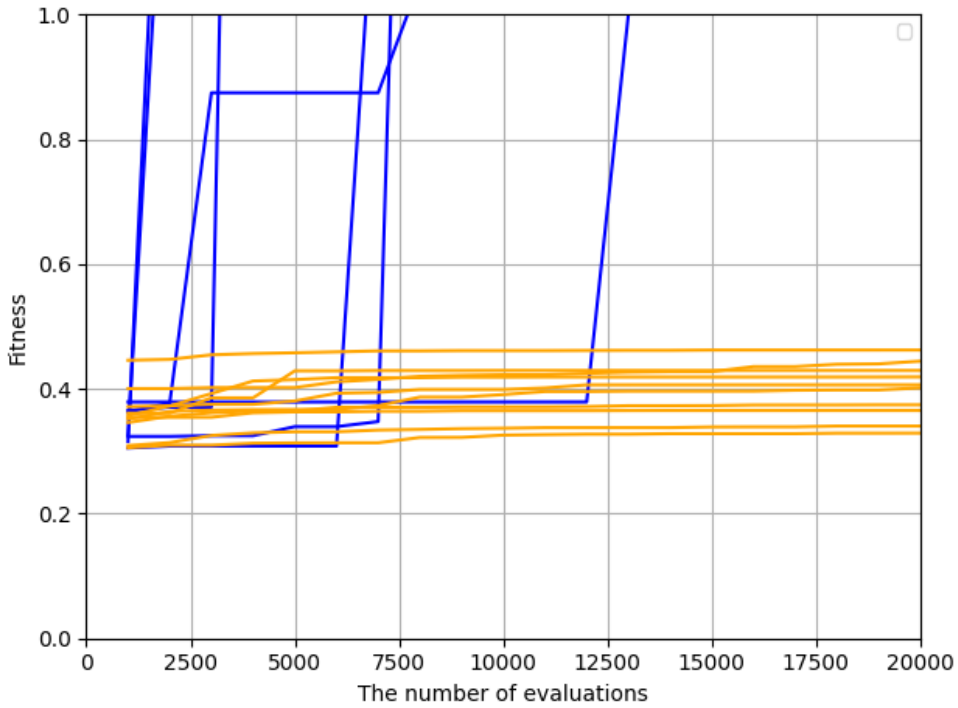

Fig.5.1　線形回帰問題における 10 試行分の既存手法（オレンジ色）と提案手法（青色）の適合度推移．横軸は評価回数, 縦軸は適合度を意味する．適合度は初期集団生成後から 100 回刻みで計測した．

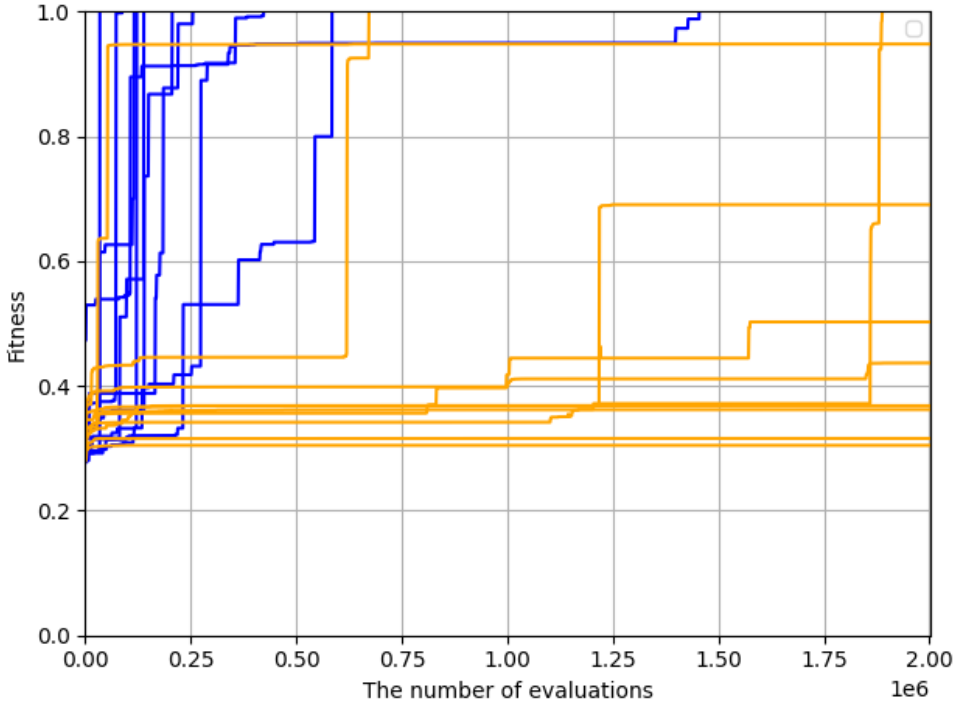

Fig.5.2　アフィン回帰問題における 10 試行分の既存手法（オレンジ色）と提案手法（青色）の適合度推移．横軸は評価回数, 縦軸は適合度を意味する．適合度は初期集団生成後から 100 回刻みで計測した．

5.2　探索効率の比較

提案手法と既存手法の探索効率を比較するために, 第 4 章の線形回帰の問題設定で, 最適解の発見までにかかる評価回数を比較する実験を行った．具体的には, 打ち切り評価回数は設定せず, 打ち切り適合度に 0.999 のみを設定して, 最適解が発見されるまでの評価回数を計測した．この実験の結果を Table.x, 適合度の推移を Fig. x に示す．この結果より, 提案手法の方が既存手法に比べて, およそ x 倍の探索効率をもつことがわかる．

5.3　発見されたアルゴリズムの解釈

本節では, 提案手法の MGG-AutoML-Zero+VAG によって発見されたアルゴリズムの解釈を行う．第 4 章の実験で発見された非線形回帰アルゴリズムを Code.xx に示す．

[アルゴリズムの解釈を追記する]

5.4　探索空間の冗長性

本節では, 既存手法において探索空間がどの程度冗長であるかを定量的に述べる．具体的には, 第 4 章の線形回帰問題と同じ探索空間で, 初期個体の生成と同様の方法で 100,000 個のアルゴリズムを生成した時の非妥当なアルゴリズムの割合を計測した．その結果, 既存手法の探索空間のうち 99.9% 以上が非妥当なアルゴリズムであることがわかった．

Table.5.1　提案手法の各工夫の有効性を検証するために用いる考察手法の一覧

提案手法の工夫要素	既存手法	考察手法			提案手法
		A	B	C	
アルゴリズムの表現	命令列	VAG	VAG	VAG	VAG
世代交代モデル	RE	RE	MGG	MGG	MGG
同一個体の排除	-	-	無効	有効	有効
生存選択	-	-	ベスト 2	ベスト 2	希少度利用

5.5　非妥当な突然変異

本節では, 既存手法における非妥当な突然変異の割合を調査する. この調査は, 第 4 章の実験のノイズなしの既存手法の線形回帰問題の設定と同様の条件で, 妥当なアルゴリズムを突然変異させた場合に, 非妥当になる割合を計測した. その結果, 既存手法による子個体生成は, x% の割合で, 非妥当な突然変異が発生していることがわかった.

5.6　提案手法の各工夫の有効性

本節では, 我々が導入したアルゴリズムグラフとその突然変異, MGG による世代交代, 集団内の同一個体の排除, 希少度を考慮した生存選択の有効性を調べる. 具体的には, Table.5.1 に示した考察手法を用いて, 第 4 章のノイズなしの線形回帰問題とアフィン回帰問題の実験を実施して結果を比較する. Table.x と Table.y にそれぞれ線形回帰とアフィン回帰の問題設定で, 所与の評価回数までに発見できたアルゴリズムの 10 試行分の適合度を示す. この結果より, 我々が導入した各工夫を導入するごとに, 10 試行分の平均の適合度が向上していることがわかる.

Table.5.2　線形回帰問題における各考察手法の結果

No.	既存手法	考察手法 A　B　C	提案手法
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
平均 標準偏差			

Table.5.3　アフィン回帰問題における各考察手法の結果

No.	既存手法	考察手法 A　B　C	提案手法
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
平均 標準偏差			

第 6 章

結論

6.1 本研究のまとめ

本研究は、Esteban らが提案した AutoML-Zero における探索効率の課題に着目し、AutoML-Zero の人間の事前知識や介入を最小限にできる利点は残しつつ、探索効率を改善する手法を提案した。Esteban らの AutoML-Zero は、機械学習アルゴリズムの自動探索において画期的な成果を示したものの、膨大な計算リソースを必要とするという重要な課題が残されていた。例えば、ReLU 関数の再発明に成功した実験では、1 秒間あたり 10,000 モデルの評価が可能な CPU を搭載したマシンを 10,000 台使用し、約 5 日間という大規模な計算を要した。

本研究では、この探索効率を低下させる要因として、(1) 探索空間の冗長性、(2) 突然変異操作の非効率性、(3) 集団の多様性維持の問題、という 3 つの課題に着目し、それぞれに対する解決策を提案した。

第一に、探索空間の冗長性に対しては、グラフ構造によるアルゴリズム表現を導入し、機械学習アルゴリズムとして満たすべき条件を明確に規定した。これにより、予測ラベルが代入されないアルゴリズムや、入力ベクトルが予測に利用されていないアルゴリズム、逐次更新される学習パラメータが存在しないアルゴリズムなど、機械学習アルゴリズムとして非妥当なものを探索空間から除外することに成功した。その結果、探索空間を xx% 以上削減することができた。また、グラフ構造による表現により、変数名の違いによる冗長性も排除することが可能となった。

第二に、突然変異操作の非効率性に対しては、グラフ構造に基づく新たな突然変異操作を提案した。既存手法では、非妥当なアルゴリズムへの突然変異や、関数全体を破壊的に変更する突然変異が高確率で発生していたが、提案手法ではグラフ構造の特性を活用することで、非妥当な突然変異を完全に防止することができた。さらに、部分グラフの再構築による突然変異を導入することで、関数内の特定の変数計算に必要な部分命令列のみを局所的に変更することが可能となった。これらの改善により、同一評価回数内で発見されるアルゴリズムの適合度を x% 以上向上させることに成功した。

第三に、集団の多様性維持の問題に対しては、Minimal Generation Gap (MGG) による世代交代モデルを採用し、さらに集団内の同一個体の排除および希少度を考慮した生存選択を導入した。既存手法では、世代交代モデルとして Regularized Evolution (RE) を採用していたことにより、集団の多様性維持が困難であるという問題が存在した。加えて、集団内における個体の重複や希少度が考慮されていなかったため、同一もしくは類似したアルゴリズムが集団内で過度に増加してしまう傾向があった。そのため、探索序盤で発見された局所最適なアルゴリズムに早期収束してしまい、探索効率の低下を招いていた。これに対して提案手法では、MGG による世代交代モデル, 集団内の同一個体の排除、希少度を考慮した生存選択を導入することで, 集団の多様性を維持しやすい探索を実現した。これにより、同一評価回数内で発見されるアルゴリズムの適

合度をさらに x% 向上させることに成功した。

提案手法の有効性を検証するため、ローカル PC という限られた計算リソース環境において、主要な回帰問題と分類問題を用いた比較実験を実施した。その結果、同一評価回数内で発見されるアルゴリズムの適合度が xx% 以上向上することを確認した。特に、線形回帰問題においては、適合度 0.999 以上のアルゴリズムを発見するまでの評価回数を既存手法の 1/x まで削減することに成功した。これらの結果は、提案手法が AutoML-Zero の利点を保持しつつ、探索効率を大幅に改善できることを示している。

以上の成果は、機械学習アルゴリズムの自動探索における計算コストの大幅な削減を実現し、より実用的な AutoML の実現に向けた重要な一歩となることが期待される。

6.2 今後の課題

本研究の今後の課題を以下に示す.

-

6.3 謝辞

本研究を遂行するにあたり, 多大なるご指導とご助言を賜りました小野功教授に深く感謝する. また, ゼミにおいて貴重なご意見や有益なアイデアを頂いた小野功研究室のメンバーにも心より感謝する.

参考文献

[1] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Advances in neural information processing systems, Vol. 2, pp. 524–532, 1989.

[2] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning, pp. 1126–1135. PMLR, 2017.

[3] Ritam Guha, Wei Ao, Stephen Kelly, Vishnu Boddeti, Erik Goodman, Wolfgang Banzhaf, and Kalyanmoy Deb. Moaz: A multi-objective automl-zero framework. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23, p. 485–492, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5, pp. 507–523. Springer, 2011.

[5] Hao Li, Tianwen Fu, Jifeng Dai, Hongsheng Li, Gao Huang, and Xizhou Zhu. Autoloss-zero: Searching loss functions from scratch for generic tasks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1009–1018, June 2022.

[6] Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. NAS-bench-suite: NAS evaluation is (now) surprisingly easy. In International Conference on Learning Representations, 2022.

[7] Renato Negrinho, Matthew Gormley, Geoffrey J Gordon, Darshan Patil, Nghia Le, and Daniel Ferreira. Towards modular and programmable architecture search. Advances in neural information processing systems, Vol. 32, pp. 524–532, 2019.

[8] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In Proceedings of the aaai conference on artificial intelligence, Vol. 33, pp. 4780–4789, 2019.

[9] Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In International Conference on Machine Learning, pp. 8007–8019. PMLR, 2020.

[10] Hiroshi Sato. A new generation alternation model of genetic algorithms and its assessment. Transactions of the Japanese Society for Artificial Intelligence, Vol. 12, No. 2, pp. 82–91, 1997.

[11] David So, Quoc Le, and Chen Liang. The evolved transformer. In International conference on machine learning, pp. 5877–5886. PMLR, 2019.

[12] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew

Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2820–2828, 2019.

[13] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. 2017.

[14] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697–8710, 2018.

付録 A

順序付き有向グラフ

A.1 はじめに

本付録では、アルゴリズムグラフの定式化で用いた順序付き有向グラフについて述べる。順序付き有向グラフとは、各ノードの子ノード間に順序関係を持つ有向グラフである。通常の有向グラフがエッジを 2 つのノードの組 (v_1, v_2) として表現するのに対し、順序付き有向グラフではエッジを始点ノード、自然数で表される順序値、終点ノードの組 (v_1, n, v_2) として表現する。この拡張により、非可換な命令（演算）の入力順序を扱うことが可能となる。

A.2 定義

順序付き有向グラフは, ノードの集合 U とエッジの集合 E からなるグラフ $G = (U, E)$ である. ここで, $E \subset U \times \mathbb{N} \times U$ であり, 各エッジ $e \in E$ は, ノード $u_1 \in U$, 順序値 $n \in \mathbb{N}$, ノード $u_2 \in U$ の順序付きペア (u_1, n, u_2) で表現される. また, 同じノードの同じ順序値に複数のエッジを対応させることはできない, 言い換えれば以下の条件を満たす.

$$\forall u_1, u_2, u_3 \in U,\ \forall n \in \mathbb{N}$$
$$(u_1, n, u_2) \in E \wedge (u_1, n, u_3) \in E \Rightarrow u_2 = u_3$$

一方で, 順序値が違えば, 同じノード間に複数のエッジを持つことも可能である.

A.3 経路と閉路

順序付き有向グラフにおける経路とは、以下の条件を満たすエッジの列 $(e_1, e_2, \ldots, e_m)$ である:

1. $e_i = (u_i, n_i, u_{i+1}) \in E$
2. $i \neq j \Rightarrow u_i \neq u_j$

u_1, u_{m+1} をそれぞれ経路の始点と終点と呼ぶ. また, $u_2, \cdots, u_m$ を経由点と呼ぶ. 閉路は, 始点と終点が同じ経路, 言い換えれば $u_1 = u_{m+1}$ となる経路である. 閉路を持たない順序付き有向グラフを非巡回順序付き有向グラフと呼ぶ. 非巡回順序付き有向グラフでは, エッジの終点ノードを始点ノードの子ノードとして考えることで, 各ノードに対して, 親ノードの集合, 子ノードの集合, 子孫ノードの集合, 祖先ノードの集合を一意に定義することができる.

AG では計算の依存関係を表現するため、グラフに閉路が存在してはならないため, AG は非巡回順序付き有向グラフとして定式化した。

A.4　トポロジカルソート

非巡回順序付き有向グラフ $G = (U, E)$ のトポロジカルソートとは、全てのノードを一列に並べた順列 $\sigma : \{1, 2, \ldots, |U|\} \to U$ であり、以下の条件を満たす：

$$\forall (u_1, n, u_2) \in E, \sigma^{-1}(u_1) < \sigma^{-1}(u_2)$$

ここで、$\sigma^{-1}(u)$ は順列 σ におけるノード u の位置を表す。この条件は、全てのエッジ $(u_1, n, u_2) \in E$ について、u_1 が u_2 より前に現れることを意味する。トポロジカルソートは、グラフが非巡回である場合に必ず存在し、一般に一意ではない。ノードをトポロジカルソートすることで、アルゴリズムグラフにおける命令の実行順序を決定することが可能となる。

具体例として、以下のグラフを考える：

- ノード集合: $U = \{A, B, C\}$
- エッジ集合: $E = \{(A, 1, B), (A, 2, C), (B, 1, C)\}$

このとき、$\sigma : (A, B, C)$ はトポロジカルソートとなる。

A.5　弱連結性

順序付き有向グラフが弱連結であるとは, 有向グラフの弱連結性と同様に, 無向グラフとして考えた時の連結性として定義される. 具体的には, 順序付き有向グラフ $G = (U, E)$ から, それに対応する無向グラフ $G' = (U, E')$ を生成し, G' が連結であるとき, G は弱連結であるという. ここで, $E' = \{(u_1, u_2) \mid (u_1, n, u_2) \in E\}$ である. 機械学習アルゴリズムにおいては, 予測ラベルの計算と独立な依存関係を持つ命令は不要であるため, アルゴリズムグラフは弱連結な順序付き有向グラフとして定式化した。

付録 B

命令セット

本研究のアルゴリズムを構成するための命令セットを Table.B.1 に示す．この命令セットは Esteban らの論文 [9] に使われている命令セットと同様である．

Table.B.1: アルゴリズムの各関数 Setup, Predict, Learn を構成する命令の一覧．Esteban らの論文 [9] に示されている命令セットと同様である．

命令	コード例	入力		出力		説明
		変数/型	定数	変数/型	添字	
OP0	no_op	-	-	-	-	-
OP1	$s2 = s3 + s0$	a, b/scalars	-	c/scalar	-	$s_c = s_a + s_b$
OP2	$s4 = s0 - s1$	a, b/scalars	-	c/scalar	-	$s_c = s_a - s_b$
OP3	$s8 = s5 * s5$	a, b/scalars	-	c/scalar	-	$s_c = s_a s_b$
OP4	$s7 = s5/s2$	a, b/scalars	-	c/scalar	-	$s_c = s_a / s_b$
OP5	$s8 = \mathrm{abs}(s0)$	a/scalar	-	b/scalar	-	$s_b = \lvert s_a \rvert$
OP6	$s4 = 1/s8$	a/scalar	-	b/scalar	-	$s_b = 1/s_a$
OP7	$s5 = \sin(s4)$	a/scalar	-	b/scalar	-	$s_b = \sin(s_a)$
OP8	$s1 = \cos(s4)$	a/scalar	-	b/scalar	-	$s_b = \cos(s_a)$
OP9	$s0 = \tan(s4)$	a/scalar	-	b/scalar	-	$s_b = \tan(s_a)$
OP10	$s0 = \arcsin(s4)$	a/scalar	-	b/scalar	-	$s_b = \arcsin(s_a)$
OP11	$s2 = \arccos(s0)$	a/scalar	-	b/scalar	-	$s_b = \arccos(s_a)$
OP12	$s4 = \arctan(s0)$	a/scalar	-	b/scalar	-	$s_b = \arctan(s_a)$
OP13	$s1 = \exp(s2)$	a/scalar	-	b/scalar	-	$s_b = e^{s_a}$
OP14	$s0 = \log(s3)$	a/scalar	-	b/scalar	-	$s_b = \log s_a$
OP15	$s3 = \mathrm{heaviside}(s0)$	a/scalar	-	b/scalar	-	$s_b = \mathbb{1}_{\mathbb{R}+}(s_a)$
OP16	$v2 = \mathrm{heaviside}(v2)$	a/vector	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \mathbb{1}_{\mathbb{R}+}(\boldsymbol{v}_a^{(i)})$
OP17	$m7 = \mathrm{heaviside}(m3)$	a/matrix	-	b/matrix	-	$\forall i, jM_b^{(i,j)} = \mathbb{1}_{\mathbb{R}+}(M_a^{(i,j)})$
OP18	$v1 = s7 * v1$	a, b/sc, vec	-	c/vector	-	$\boldsymbol{v}_c = s_a \boldsymbol{v}_b$
OP19	$v1 = \mathrm{bcast}(s3)$	a/scalar	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = s_a$
OP20	$v5 = 1/v7$	a/vector	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = 1/s_a^{(i)}$
OP21	$s0 = \mathrm{norm}(v3)$	a/scalar	-	b/vector	-	$s_b = \lVert \boldsymbol{v}_a \rVert$
OP22	$v3 = \mathrm{abs}(v3)$	a/vector	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\lvert \boldsymbol{v}_a^{(i)} \right\rvert$
OP23	$v5 = v0 + v9$	a, b/vectors	-	c/scalar	-	$\boldsymbol{v}_c = \boldsymbol{v}_a + \boldsymbol{v}_b$
OP24	$v1 = v0 - v9$	a, b/vectors	-	c/scalar	-	$\boldsymbol{v}_c = \boldsymbol{v}_a - \boldsymbol{v}_b$
OP25	$v8 = v0 * v9$	a, b/vectors	-	c/scalar	-	$\forall i, \boldsymbol{v}_c^{(i)} = \boldsymbol{v}_a^{(i)} \boldsymbol{v}_b^{(i)}$
OP26	$v9 = v8/v2$	a, b/vectors	-	c/scalar	-	$\forall i, \boldsymbol{v}_c^{(i)} = \boldsymbol{v}_a^{(i)} / \boldsymbol{v}_b^{(i)}$

命令 ID	コード例	入力		出力		説明
		変数/型	定数	変数/型	添字	
OP27	$s6 = \mathrm{dot}(v1, v5)$	a, b/vectors	-	c/scalar	-	$s_c = \boldsymbol{v}_a^T \boldsymbol{v}_b$
OP28	$m1 = \mathrm{outer}(v6, v5)$	a, b/vectors	-	c/matrix	-	$M_c = \boldsymbol{v}_a \boldsymbol{v}_b^T$
OP29	$m1 = a4 * m2$	a, b/sc/mat	-	c/matrix	-	$M_c = s_a M_b$
OP30	$m3 = 1/m0$	a/matrix	-	b/matrix	-	$\forall i, j, M_b^{(i,j)} = 1/M_a^{(i,j)}$
OP31	$v6 = \mathrm{dot}(m1, v0)$	a, b/mat/vec	-	c/vector	-	$\boldsymbol{v}_c = M_a \boldsymbol{v}_b$
OP32	$m2 = \mathrm{bcast}(v0, \mathrm{axis} = 0)$	a/vector	-	b/matrix	-	$\forall i, j, M_b^{(i,j)} = \boldsymbol{v}_a^{(i)}$
OP33	$m2 = \mathrm{bcast}(v0, \mathrm{axis} = 1)$	a/vector	-	b/matrix	-	$\forall i, j, M_b^{(j,i)} = \boldsymbol{v}_a^{(i)}$
OP34	$s2 = \mathrm{norm}(m1)$	a/matrix	-	b/scalar	-	$s_b = \lVert \boldsymbol{v}_a \rVert$
OP35	$v4 = \mathrm{norm}(m7, \mathrm{axis} = 0)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\lVert M_a^{(i,\cdot)} \right\rVert$
OP36	$v4 = \mathrm{norm}(m7, \mathrm{axis} = 1)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \left\lVert M_a^{(\cdot,i)} \right\rVert$
OP37	$m9 = \mathrm{transpose}(m3)$	a/matrix	-	b/matrix	-	$M_b = M_a^T$
OP38	$m1 = \mathrm{abs}(m8)$	a/matrix	-	b/matrix	-	$\forall i, j, M_b^{(i,j)} = \left\lvert M_a^{(i,j)} \right\rvert$
OP39	$m2 = m2 + m0$	a, b/matrixes	-	c/matrix	-	$M_c = M_a + M_b$
OP40	$m2 = m3 - m1$	a, b/matrixes	-	c/matrix	-	$M_c = M_a - M_b$
OP41	$m3 = m2 * m3$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = M_a^{(i,j)} M_b^{(i,j)}$
OP42	$m4 = m2/m4$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = M_a^{(i,j)} / M_b^{(i,j)}$
OP43	$m5 = \mathrm{matmul}(m5, m7)$	a, b/matrixes	-	c/matrix	-	$M_c = M_a M_b$
OP44	$s1 = \mathrm{minimun}(s2, s3)$	a, b/scalars	-	c/scalar	-	$s_c = \min(s_a, s_b)$
OP45	$v4 = \mathrm{minimun}(v3, v9)$	a, b/vectors	-	c/vector	-	$\forall i, \boldsymbol{v}_c^{(i)} = \min\left(\boldsymbol{v}_a^{(i)}, \boldsymbol{v}_b^{(i)}\right)$
OP46	$m5 = \mathrm{minimun}(m5, m7)$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = \min\left(M_a^{(i,j)}, M_b^{(i,j)}\right)$
OP47	$s8 = \mathrm{maximum}(s3, s0)$	a, b/scalars	-	c/scalar	-	$s_c = \max(s_a, s_b)$
OP48	$v7 = \mathrm{maximum}(v3, v6)$	a, b/vectors	-	c/vector	-	$\forall i, \boldsymbol{v}_c^{(i)} = \max\left(\boldsymbol{v}_a^{(i)}, \boldsymbol{v}_b^{(i)}\right)$
OP49	$m7 = \mathrm{maximum}(m1, m0)$	a, b/matrixes	-	c/matrix	-	$\forall i, j, M_c^{(i,j)} = \max\left(M_a^{(i,j)}, M_b^{(i,j)}\right)$
OP50	$s2 = \mathrm{mean}(v2)$	a/vector	-	b/scalar	-	$s_b = \mathrm{mean}(\boldsymbol{v}_a)$
OP51	$s2 = \mathrm{mean}(m8)$	a/matrix	-	b/scalar	-	$s_b = \mathrm{mean}(M_a)$
OP52	$v1 = \mathrm{mean}(m2, \mathrm{axis} = 0)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \mathrm{mean}\left(M_a^{(i,\cdot)}\right)$
OP53	$v3 = \mathrm{stdev}(m2, \mathrm{axis} = 0)$	a/matrix	-	b/vector	-	$\forall i, \boldsymbol{v}_b^{(i)} = \mathrm{stdev}\left(M_a^{(i,\cdot)}\right)$
OP54	$s3 = \mathrm{stdev}(v3)$	a/vector	-	b/scalar	-	$s_b = \mathrm{stdev}(\boldsymbol{v}_a)$
OP55	$s4 = \mathrm{stdev}(m0)$	a/matrix	-	b/scalar	-	$s_b = \mathrm{stdev}(M_a)$
OP56	s2 = 0.7	-	γ	a/scalar	-	$s_a = \gamma$
OP57	v3[5] = -2.4	-	γ	a/vector	i	$\boldsymbol{v}_a^{(i)} = \gamma$
OP58	m2[5,1] = -0.03	-	γ	a/matrix	i, j	$M^{(i,j)} = \gamma$
OP59	$s4 = \mathrm{uniform}(-1, 1)$	-	α, β	a/scalar	-	$s_a = \mathcal{U}(\alpha, \beta)$
OP60	$v1 = \mathrm{uniform}(0.4, 0.8)$	-	α, β	a/vector	-	$\forall i, \boldsymbol{v}_a^{(i)} = \mathcal{U}(\alpha, \beta)$
OP61	$m0 = \mathrm{uniform}(-0.5, 0.6)$	-	α, β	a/vector	-	$\forall i, j, M_a^{(i,j)} = \mathcal{U}(\alpha, \beta)$
OP62	$s4 = \mathrm{gaussian}(0.1, 0.7)$	-	μ, σ	a/scalar	-	$s_a = \mathcal{N}(\mu, \sigma)$
OP63	$v8 = \mathrm{gaussian}(0.4, 1)$	-	μ, σ	a/vector	-	$\forall i, \boldsymbol{v}_a^{(i)} = \mathcal{N}(\mu, \sigma)$
OP64	$m2 = \mathrm{gaussian}(-2, 1.3)$	-	μ, σ	a/vector	-	$\forall i, j, M_a^{(i,j)} = \mathcal{N}(\mu, \sigma)$