情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2021-CG-184 No.25
Vol.2021-DCC-29 No.25
Vol.2021-CVIM-227 No.25
2021/11/6

コンカレント・プログラミングでのアルゴリズムの可視化のためのグラフ・レイアウトの評価基準

佐藤 信[1]

概要：本稿では，コンカレント・プログラミングでのアルゴリズムの可視化のためのグラフ・レイアウトの評価基準を提案する．コンカレント・プログラミングは複数のエンティティ (プロセスまたはスレッド) を連携動作させるための手法であり，対話的なグラフィックス・ソフトウエアなどでもよく用いられる．そのアルゴリズムの各ステップの処理は集中することが多いことから，アルゴリズムを可視化すると表示が重なる部分が多いグラフが作成される．そのようなグラフを分かり易く作成するために，レイアウトの評価基準を基にグラフを選択するための手法について検討する．グラフィックスにより視覚的に表現することは，アルゴリズムのメンタル・モデルの構築を容易にするために有効である．

Graph Layout Criterion for Visualizing Concurrent Programming Algorithms

Makoto Satoh[1]

Abstract: This paper proposes a graph layout criterion for visualizing algorithms used in concurrent programming. Concurrent programming is a method for cooperative operation among multiple entities (processes and threads), used often in interactive graphics software. Graphs with many overlays are generated by visualizing the algorithms, because the processes of the each algorithm step often localize. A method for selecting graphs based on the layout criterion is considered so as to generate such a graphs in an easy-to-understand manner. Visual representations with graphics are effective to construct the mental models of the algorithms.

1. はじめに

本稿では，コンカレント・プログラミングで用いられるアルゴリズムの可視化のためのグラフ・レイアウトの評価基準を提案する[*1]．コンカレント・プログラミングは重要なプログラミング手法であることから情報工学コースなどにおいても学修されることがある [3], [6]．その手法では複数のエンティティ (プロセスまたはスレッド) を連携動作させることから，理解を容易にするために可視化がおこなわれる [1]．本稿では，コンカレント・プログラミングの入門手法 [9] でのアルゴリズムの可視化のためにグラフ・レイアウトの評価基準を提案する．[9] の手法および評価基準の特徴は，次のとおりである．

- Linux のシェルおよびコンカレント・プログラミングのためのモジュールを用いて，アルゴリズムの骨格に焦点をあてた学修をおこなえる．エンティティの連携動作の可視化が容易である．
- 可視化におけるグラフィックス要素のレイアウトのオーバレイに関する評価基準を用いて，アルゴリズムの表現に適したグラフを選択可能である．

これ以降の構成について説明する．2 節では，関連研究と提案手法の比較をおこなう．グラフ・レイアウトの評価

[1] 岩手大学
Iwate University, Ueda, Iwate 020–8551, Japan

[*1] 提案手法の実装を公開している．
https://blue0.an.cis.iwate-u.ac.jp/ConcurrentProgrammingToolkit/

基準について，3 節において述べる．4 節では提案手法により実験をおこない検討をおこなう．最後の 5 節において本稿のまとめと今後について述べる．

2. 関連研究

2.1 コンカレント・プログラミングにおける可視化

情報を視覚的に提示することを目的として，情報の可視化に関する研究が多数おこなわれている [7] ．数値により表現されたデータをグラフなどで視覚的に表現することは，データの特徴を分かりやすくし分析などを容易にする．

コンピュータ・アルゴリズムについても，手法の理解を容易にすることを目的として可視化に関する研究がおこなわれている [4], [11] ．アルゴリズムの概念を可視化することは，そのメンタル・モデルを構築するために有効である．コンカレント・プログラミングで使用されるアルゴリズムについても可視化のための研究がおこなわれている．コンカレント・プログラミングとは，複数のプログラムを連携動作させる場合などに必要となるプログラミング手法である [5] ．マルチプログラミングのオペレーティング・システム (OS) のために必要な技術として研究・開発が始められ手法であり，現在の OS(Linux, Windows, macOS など) のためにも必要不可欠な技術である．次に例を示すように，それ以外のプログラミングの場面においてもよく用いられる．

- 複数のプログラムによる共有データの更新
- 複数のプログラムの同期
- 対話的なグラフィックス・プログラミング
- ユーザ・インタフェース

入門レベルのプログラミング [8] において，コンカレント・プログラミングでのアルゴリズムを取り上げた研究もある [3] ．そこでは，コンカレント・プログラミングをプログラムのグラフィカルな動作により体験している．履修者へのアンケートでは，一般のアルゴリズムと比較してコンカレント・プログラミングに関する題材については，難しい (他の題材と比較して difficulty のスコアが高い) が面白くて (fun が高い) 学修する価値がある (educational value が特に高い) という回答が多く得られたことが報告されている．

Java, C++などを用いてコンカレント・プログラムの実験をおこなう場合には，それらの言語でのエンティティの生成およびプロセス間通信のための機能などについての知識が必要となるが，本稿でのコンカレント・プログラミングの入門手法 [9] では，シェル・スクリプトの入門レベルの知識があればコンカレント・プログラミングのためのアルゴリズムに関する実験が可能である．アルゴリズムの骨格に焦点をあてた学修が可能であり，アルゴリズムの可視化によりメンタル・モデルの構築が容易である．

2.2 可視化のためのコンピュータ・グラフィックス

コンピュータ・グラフィックスのための手法の多くは，何らかの目的のために視覚的情報を生成することを目的として研究・開発されている．例えば，形状のモデリング・データをレンダリングにより視覚情報として表現することなどがおこなわれる．そのため，可視化のための手法は，コンピュータ・グラフィックス分野の重要な研究テーマとなっている．また，一般的なデータの可視化においても，コンピュータ・グラフィックスのために研究された多数の既存手法が用いられる．その中でも特に，レンダリングに関する手法が可視化との関係が深いといえる．

可視化でのグラフィックス・レンダリングを対象として，色情報を有効に用いるための研究が多数おこなわれている [13], [14] ．色情報に関する研究では，透過性のない色 (不透明度が 1) を対象としたものが比較的多数であるが，[12] のように透過性のある色についても研究がおこなわれている．また，[2] では，グラフにオーバレイするグリッドのような補助的な情報を透過性のある色により適切に表現するための研究がおこなわれている．

本稿の可視化ではレンダリングでのグラフィックス要素のオーバレイを考慮する必要があることから (詳細は 3.1 節)，必要に応じて透過性のある色を用いてレンダリングをおこなう．また，アルゴリズムの説明のためには必要ではないオーバレイについては，オーバレイが少ないことが望ましい．そこで，グラフ・レイアウトの評価基準を用いることにより，アルゴリズムの説明に適したグラフを選択して生成できるような [9] での可視化のパラメータについて検討をおこなう．評価基準の計算では，レンダリングにおいて [10] での画像合成オペレータを用いてグラフィックス要素のオーバレイを測定する．

3. グラフ・レイアウトの評価基準

3.1 可視化対象の特徴

本稿で用いる手法 [9] では，乱数を用いてエンティティの動作にランダム性を与えている．また，アルゴリズムのある部分の動作は集中しておこなわれる可能性が大きい．そのため，次の方法により可視化でのグラフィックス要素のオーバレイに対応する．

- グラフィックス要素がオーバレイしても識別しやすいように，透過性のある色によりレンダリングする．
- オーバレイの発生を減少させるような，パラメータ設定により実験をおこなえるようにする．

Vol.2021-CG-184 No.25
Vol.2021-DCC-29 No.25
Vol.2021-CVIM-227 No.25
2021/11/6

Algorithm 1 Computing graph layout criterion.

Prepare an image buffer for current overlaid shapes: $\boldsymbol{C}$.
Prepare an image buffer for accumulated overlaid shapes: $\boldsymbol{A}$.
Prepare an image buffer for a visualization plot: $\boldsymbol{V}$.

for each graphics element $\boldsymbol{e}$ **in** a generating visualization plot **do**
  step 1: Synthesize $\boldsymbol{C}$ by composing $\boldsymbol{V}$ onto $\boldsymbol{e}$.
    composition operation: $\boldsymbol{V}$ **in** $\boldsymbol{e}$ (ref. [10])
  step 2: Synthesize $\boldsymbol{A}$ by composing $\boldsymbol{A}$ onto $\boldsymbol{C}$.
    composition operation:
      dissolve($\boldsymbol{A}$) **plus dissolve**($\boldsymbol{C}$) (ref. [10]),
      the factors of **dissolve** are computed
      so as to limit the maximum value of pixels.
  step 3: Render $\boldsymbol{e}$ onto $\boldsymbol{V}$.
end for

Compute graph layout criterion l using $\boldsymbol{V}$ and $\boldsymbol{A}$.

3.2 評価基準の計算

グラフィックス要素のオーバレイの検出手法，および，グラフ・レイアウト評価基準の計算手法をアルゴリズム 1 に示す．

手法では始めに，次の画像バッファを用意する．

- その繰り返しで検出したオーバレイの形状を記録するための画像バッファ: $\boldsymbol{C}$
- その繰り返しまでに検出したオーバレイの形状を積分するための画像バッファ: $\boldsymbol{A}$
- その繰り返しまでにレンダリングした部分グラフ形状を格納するための画像バッファ: $\boldsymbol{V}$

そして，グラフを構成する各グラフィックス要素 $\boldsymbol{e}$ について，次を繰り返しおこなう．

step 1 画像合成 $\boldsymbol{V}$ **in** $\boldsymbol{e}$ をおこなうことにより，$\boldsymbol{V}$ と $\boldsymbol{e}$ のオーバレイの形状を $\boldsymbol{C}$ に記録する．このとき $\boldsymbol{V}$ は変化させない．
step 2 画像合成 **dissolve**($\boldsymbol{A}$) **plus dissolve**($\boldsymbol{C}$) をおこなうことにより，$\boldsymbol{C}$ を $\boldsymbol{A}$ に積分する．
step 3 $\boldsymbol{e}$ を $\boldsymbol{V}$ にレンダリングする．

画像合成のオペレータ (**in**, **dissolve** および **plus**) は，[10] のものを用いる．

グラフ・レイアウト評価基準は，次式により計算する．

$$l = S_{(\boldsymbol{A})}/S_{(\boldsymbol{V})}$$

ここで，$S_{(\boldsymbol{A})}$ および $S_{(\boldsymbol{V})}$ は，それぞれ，$\boldsymbol{A}$ および $\boldsymbol{V}$ において形状を表現しているピクセルの数である．

図 1 は，3 個の内部を塗りつぶした円形をレンダリングする場合について，アルゴリズム 1 によりオーバレイを検出

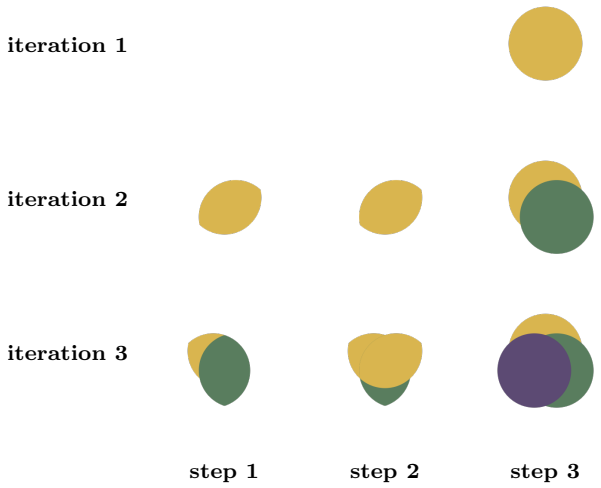

図 1: オーバレイ検出の各ステップ (アルゴリズム 1 参照)
Fig. 1 Each step of overlay detection (ref. algorithm 1).

する例である．各画像バッファ $\boldsymbol{C}$, $\boldsymbol{A}$ および $\boldsymbol{V}$ は，始めはクリアされた状態である．**iteration 1** の **step 1** では，$\boldsymbol{V}$ **in** $\boldsymbol{e}$ をおこなうが，$\boldsymbol{V}$ と $\boldsymbol{e}$ の共通部分はないので合成結果が格納された $\boldsymbol{C}$ はクリアされた状態のままである (第 1 行左)．**step 2** では，**dissolve**($\boldsymbol{A}$) **plus dissolve**($\boldsymbol{C}$) をおこなうが，$\boldsymbol{A}$ および $\boldsymbol{C}$ はクリアされたままなので，合成結果が格納された $\boldsymbol{A}$ はクリアされた状態のままである (第 1 行中央)．**step 3** では，$\boldsymbol{e}$ を $\boldsymbol{V}$ にレンダリングする (第 1 行右)．**iteration 2** の **step 1** では，$\boldsymbol{V}$ **in** $\boldsymbol{e}$ をおこない，$\boldsymbol{V}$(第 1 行右の状態) と $\boldsymbol{e}$(第 2 の円形) の共通部分が $\boldsymbol{C}$ に格納される (第 2 行左)．**step 2** では，**dissolve**($\boldsymbol{A}$) **plus dissolve**($\boldsymbol{C}$) をおこない，$\boldsymbol{A}$(第 1 行中央) および $\boldsymbol{C}$(第 2 行左) の積分結果が $\boldsymbol{A}$ に格納される (第 2 行中央)．**step 3** では，$\boldsymbol{e}$ を $\boldsymbol{V}$(第 1 行右の状態) にレンダリングする (第 2 行右)．**iteration 3** の **step 1** では，$\boldsymbol{V}$ **in** $\boldsymbol{e}$ をおこない，$\boldsymbol{V}$(第 2 行右の状態) と $\boldsymbol{e}$(第 3 の円形) の共通部分が $\boldsymbol{C}$ に格納される (第 3 行左)．**step 2** では，**dissolve**($\boldsymbol{A}$) **plus dissolve**($\boldsymbol{C}$) をおこない，$\boldsymbol{A}$(第 2 行中央) および $\boldsymbol{C}$(第 3 行左) の積分結果が $\boldsymbol{A}$ に格納される (第 3 行中央)．**step 3** では，$\boldsymbol{e}$ を $\boldsymbol{V}$(第 2 行右の状態) にレンダリングする (第 3 行右)．この例では，初期状態のアルファは 0，形状のアルファは 1 なので，**step 2** では $\boldsymbol{A}$ を $\boldsymbol{C}$ に上書きする．

3.3 オーバレイの発生の少ないパラメータの選択

[9] での手法を用いてアルゴリズムを可視化した場合について，グラフィックス要素のオーバレイが発生する頻度に影響する主な原因は次のとおりである．

- コンカレント・プログラミングのためのアルゴリズムによるもの
- エンティティの動作にランダム性を与えるための乱数

Vol.2021-CG-184 No.25
Vol.2021-DCC-29 No.25
Vol.2021-CVIM-227 No.25
2021/11/6

によるエンティティの動作のスリープ

そのため，セマフォの使用の有無，および，エンティティの動作をスリープさせる期間を計算するための基準値について，アルゴリズムの説明に適したオーバレイの少ない可視化をおこなうための設定を検討する．オーバレイの測定には，3.2 で述べたグラフ・レイアウトの評価基準を用いる．

4. 実験と結果の検討

4.1 実装

3 節で提案した手法の実装により動作確認をおこなった．アルゴリズム 1 のための C++のクラスを設計・実装し，[9] において実装をおこなった可視化のためのモジュールでそのクラスを使用した．動作確認は Linux でおこなった．シェルには bash を用いた．なお，コンカレント・プログラミングのためのアルゴリズムの実験には，[9] において実装をおこなったモジュールを用いた．

4.2 アトミシティについての実験の概要

コンカレント・プログラミングのためのアルゴリズムの実験として，複数のプロセスにより共有変数の値の更新をおこなう．図 2 は，共有変数とプロセスの関係を示している．プロセス **P1** および **P2** が，共有変数 **sum** の値を更新している．**P1** の動作は次のとおりである．

Step 1 プロセス **P1** は，共有変数 **sum** の値を **P1** 内の変数に読み込む ($\mathtt{get}_{(\mathbf{P1})}$)．

Step 2 **P1** は，読み込んだ値に 1 を加算してから加算結果を **sum** に書き込む ($\mathtt{put}_{(\mathbf{P1})}$)．

プロセス **P2** についても同様である．**sum** の初期値が 0 であるとすると，動作の順序により次のような結果となる．

順序パターン 1 (加算結果 1):

$$\mathtt{get}_{(\mathbf{P1})} \Rightarrow \mathtt{get}_{(\mathbf{P2})} \Rightarrow \mathtt{put}_{(\mathbf{P1})} \Rightarrow \mathtt{put}_{(\mathbf{P2})}$$

順序パターン 2 (加算結果 2):

$$\mathtt{get}_{(\mathbf{P1})} \Rightarrow \mathtt{put}_{(\mathbf{P1})} \Rightarrow \mathtt{get}_{(\mathbf{P2})} \Rightarrow \mathtt{put}_{(\mathbf{P2})}$$

可能な動作の順序には，他にも複数の組み合わせがある．各プロセスは OS により CPU を割り当てられて実行されるが，どのプロセスをどれだけの時間 CPU に割り当てるのかはシステムの状態などを基に OS が決定する．**P1** の動作のみに着目すると，

$$\mathtt{get}_{(\mathbf{P1})} \Rightarrow \mathtt{put}_{(\mathbf{P1})}$$

という順序は保証される．一方，$\mathtt{get}_{(\mathbf{P2})}$ の動作については，$\mathtt{get}_{(\mathbf{P1})}$ より前，$\mathtt{get}_{(\mathbf{P1})}$ と $\mathtt{put}_{(\mathbf{P1})}$ の間，あるいは，$\mathtt{put}_{(\mathbf{P1})}$ の後のいずれになるのかは，OS によるプロセスの割り当てがどのようにおこなわれるのかによる．$\mathtt{put}_{(\mathbf{P2})}$ についても同様である．そして，**P2** からみた **P1** の動作についても同様である．

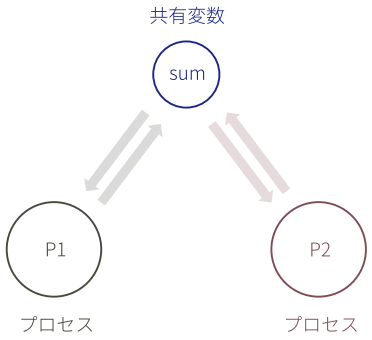

図 2: 複数プロセスによる共有変数の更新
Fig. 2 Updating a shared variable with multiple processes.

表 1: アトミシティの実験のためのモジュール
Table 1 Modules for the experiments of atomicity.

modules	description
StartCP	Preparing a shared variable.
Add1	Adding 1 to the shared variable.
Print	Verifying the shared variable.
EndCP	Removing the shared variable.
ProcessBehaviorGraph	Generating visualization.

以上のように，1 つのプロセスのみに着目した場合には動作の順序はプログラムのとおりに保障されるが，複数のプロセスの動作に着目した場合には複雑な順序関係が存在する．コンカレント・プログラミングためのアルゴリズムを用いると，プロセスを必ず**順序パターン 2** (または，**P1** と **P2** を交換した順序) で動作するようにプロセスの動作を制御できる．その場合には，プロセスによる共有変数の更新においてアトミシティが確保されているという．

表 1 に，Atomicity の実験用のモジュールを示す．

`StartCP` 共有メモリ領域を確保し，その領域に共有変数 **sum** を用意する．

`Add1` `Add1` プロセスを 1 つ生成する．`Add1` プロセスは，共有変数 **sum** に格納されている値を読み込み，その値に 1 を加算する．そして，加算結果で **sum** の値を更新する (オプションでアトミシティを指定)．

`Print` 共有変数 **sum** に格納されている値を表示する．

`EndCP` 共有メモリ領域を解放する (システムに返却する)．

`ProcessBehaviorGraph` プロセスの動作を可視化する．

4.3 グラフィックス要素のオーバレイの検出

コンカレント・プログラミングのアルゴリズムの基本である，アトミシティについての実験をおこなった．4.2 節の例のように，2 つのプロセスにより共有変数を更新した．そして，可視化をおこなった．

実験に用いたシェル・スクリプト Sum.sh を，リスト 1 に示す．シェル・スクリプトの引数により，加算をおこなうプロセスの総数，プロセスをスリープする時間を計算するための基準値 (スリープ・ベース)，および，セマフォの使用の有無を指定できる．次に，例を示す．

```
./Sum.sh 2 1000 true
```

リスト 1 の StartCP, Add1 および EndCP (茶色) は，アトミシティの実験のためのモジュールである (表 1)．プロセスの動作ログを記録するために，オプション-l を指定している．Add1 の引数に指定している$count は，複数の Add1 プロセスからの出力を区別するための ID である．

プロセスの動作を可視化したものを図 3 に示す．Sum.sh の実行が終了してから ProcessBehaviorGraph モジュールを起動するとグラフを作成できる．グラフには，プロセス **P1** および **P2**，共有変数 **sum**，および，セマフォ **add1** の動作の関連が表現されている．**P1** および **P2** について矩形で表現されているのが，プロセスが動作をスリープしている期間である．塗りつぶした矩形とそうでない矩形が表示されているように，スリープは 2 回おこなわれる．**sum** については共有変数の値，そして，**add1** についてはセマフォの初期値およびセマフォに対する操作が表示されている．プロセスの動作を，次に示す．

Step 1 プロセス **P1** および **P2** は動作を開始する．そして，乱数を基に決定した期間スリープする．

Step 2 **P2** は，スリープを終了する．そして，セマフォをデクリメントする．セマフォが負でない (0) ので **P2** の動作は継続される．**P2** は，**sum** の値をプロセスにコピーする．そして，**P2** はスリープする．

Step 3 **P1** は，スリープを終了する．そして，セマフォをデクリメントする．セマフォが負なので **P1** の動作は一時停止される．

Step 4 **P2** は，スリープを終了する．**P2** は，**Step 2** でコピーした値に 1 を加算した値で **sum** を更新する．そして，セマフォをインクリメントしてから動作を終了する．

Step 5 **P1** は，**Step 4** での **P2** によるセマフォのインクリメントにより動作が再開される．そして，**P2** と同様の動作をおこなう．

図 4 は，図 3 についてグラフィックス要素のオーバレイを検出した結果である．この場合のグラフ・レイアウトの評価基準の値は，0.221 である．

リスト 1: アトミシティの実験のコード
Listing 1: Code of the experiment of atomicity.

```
...
total_processes=$1
sleep_base=$2
semaphore=$3

StartCP -l
for (( count = 1; count <= ${total_processes}; \
       count ++ )) do
  Add1 --semaphore ${semaphore} \
       --base ${sleep_base} -l $count &
done
wait
Print
EndCP -l
```

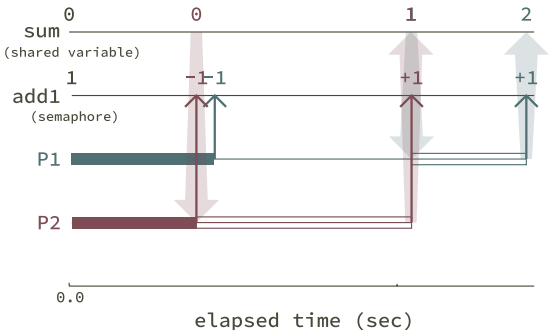

図 3: アトミシティの実験の可視化
Fig. 3 Visualization of the experiment of atomicity.

図 4: 図 3 の可視化でのオーバレイの検出
Fig. 4 Detecting the overlay of **Fig. 3**.

Vol.2021-CG-184 No.25
Vol.2021-DCC-29 No.25
Vol.2021-CVIM-227 No.25
2021/11/6

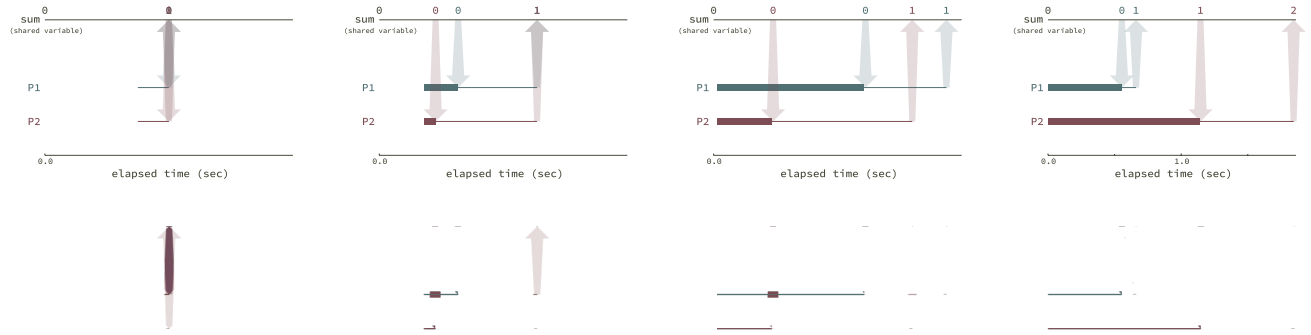

図 5: パラメータの選択 (セマフォ未使用): 可視化 (上段), オーバレイ (下段). スリープ・ベースは, 左から 1,10,100,1000msec. 見やすさのため下段の画像のアルファ値を調節.

Fig. 5 Selecting parameters (without a semaphore): visualization (top) and overlay (bottom). From left to right, the sleep base of each column is 1,10,100 and 1000 msec respectively. The alpha velues of the bottom images are adjusted for easiness to see.

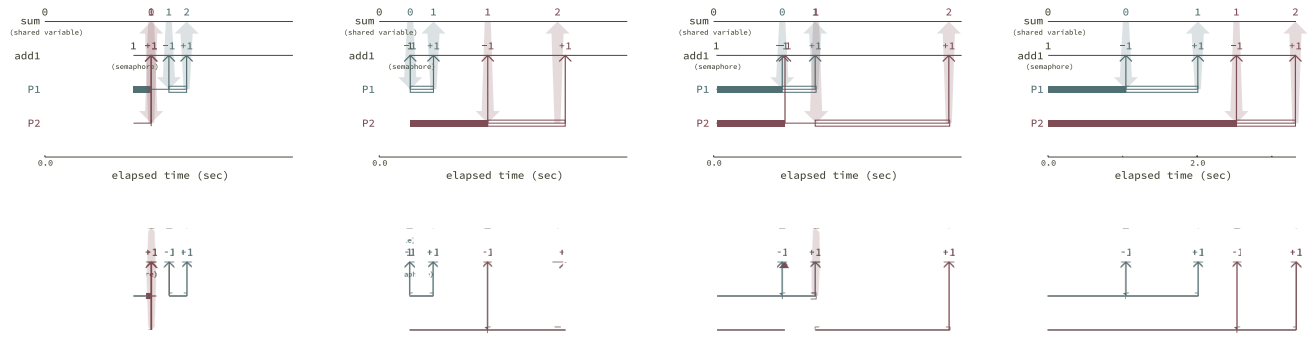

図 6: パラメータの選択 (セマフォ使用): 可視化 (上段), オーバレイ (下段). スリープ・ベースは, 左から 1,10,100,1000msec. 見やすさのため下段の画像のアルファ値を調節.

Fig. 6 Selecting parameters (with a semaphore): visualization (top) and overlay (bottom). From left to right, the sleep base of each column is 1,10,100 and 1000msec respectively. The alpha velues of the bottom images are adjusted for easiness to see.

表 2: グラフ・レイアウト評価基準の値: カッコ内はスリープ・ベースの値 (msec)

Table 2 Values of graph layout criterion: sleep bases (msec) are shown in the parentheses.

column	1 (1)	2 (10)	3 (100)	4 (1000)
Fig. 5	0.372	0.181	0.053	0.048
Fig. 6	0.245	0.122	0.200	0.142

4.4 評価基準に基づくパラメータの選択

コンカレント・プログラミングのためのアルゴリズムの可視化において，グラフィックス要素のオーバレイの少ないグラフを作成するためのパラメータについて実験をおこなった.

4.3 節での実験において，セマフォの使用の有無，および，複数の値のスリープ・ベースについて，可視化により作成されるグラフの比較をおこなった．スリープ・ベースの値には，1,10,100 および 1000msec を用いた．リスト 1 のシェル・スクリプトを起動するためのシェル・スクリプトを用いて実験をおこなった．パラメータの各組み合わせについて 1000 回，合計 8000 回の測定をおこなった．可視化により生成した各グラフについて，3 節でのグラフ・レイアウトの評価基準を計算した.

Vol.2021-CG-184 No.25
Vol.2021-DCC-29 No.25
Vol.2021-CVIM-227 No.25
2021/11/6

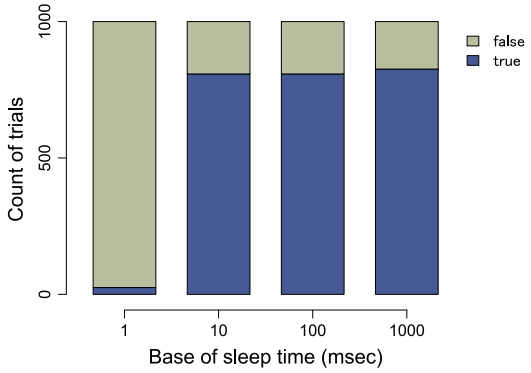

図 7: スリープ・ベースとオーバレイの関係 (セマフォ未使用)

Fig. 7 Relation between sleep base and overlay (without a semaphore).

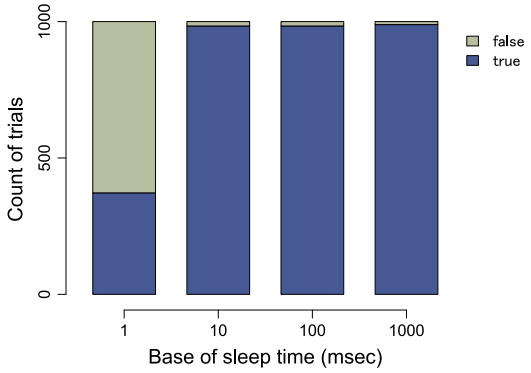

図 8: スリープ・ベースとオーバレイの関係 (セマフォ使用)

Fig. 8 Relation between sleep base and overlay (with a semaphore).

図 5 は，セマフォを使用しない場合の実験結果から選択したものである．上段はプロセスの動作の可視化であり，下段は上段の可視化でのオーバレイの検出結果である．スリープ・ベースの値は，左から右に 1,10,100 および 1000msec である．図 6 は，セマフォを使用した場合の実験結果から選択したものである．表 2 は，図 5, 6 でのレイアウト・オーバレイ評価基準の値である．

実験データについて，可視化のグラフ形状，オーバレイの形状，および，グラフ・レイアウト評価基準の値を比較することにより，アルゴリズムの動作原理の説明には必要のないオーバレイが発生していないグラフを選択するためのグラフ・レイアウト評価基準の最大値を次のように決定した．

- セマフォを使用しない場合 0.1
- セマフォを使用する場合 0.25

図 7, 8 は，この値を基準としてスリープ・ベースの値ごとに実験データを分類した結果である．上記の最大値以下のものが true である．

4.5 検討

4.3 節では，[9] で提案のコンカレント・プログラミングでのアルゴリズムの実験のためのモジュールにより，複数プロセスによる共有変数の更新の実験をおこなった．プロセスの動作ログからプロセスの動作の可視化をおこない，セマフォを用いたアルゴリズムを使用してアトミシティを確保しながら共有変数を更新するプロセスの動作を確認した．そして，3 節での提案手法を使用して，可視化でのグラフィックス要素のオーバレイを検出し，グラフ・レイアウトの評価基準の値を測定した．実験結果でのプロセスの動作は，プロセス **P2** がプロセス **P1** より先にクリティカル・セクションに入り，それにより，**P1** の実行が一時停止されるという順序パターンである．**P2** がクリティカル・セクションでの処理を終了してからセマフォをインクリメントすると **P1** の実行が再開されている．このような場合には，可視化でのグラフィックス要素のオーバレイが発生する．このオーバレイは，アルゴリズムの動作の説明のために必要なオーバレイである．

4.4 節では，グラフ・レイアウトの評価基準を用いて，オーバレイの少ない可視化をおこなうためのパラメータの選択をおこなった．始めに実験データから，オーバレイの少ない可視化をおこなうための評価基準の最大値を決定した．そして，その値を基準としてスリープ・ベースの値ごとに実験データを分類した．図 7, 8 に示す分類結果からは，スリープ・ベースが 1msec の場合には評価基準の最大値の条件を満たさない (false) オーバレイの多いグラフが多数を占めることが分かる．一方，スリープ・ベースが 10,100 および 1000msec の場合には，大部分のグラフが条件を満たす (true) ことが分かる．スリープ・ベースの値の増加に伴い，条件を満たすグラフの割合が増加することも分かる．これらにより，スリープ・ベースの値としては 10,100 および 1000msec が適当であるといえる．なお，本稿で用いたコンカレント・プログラミングのためのアルゴリズムの実験手法では，シェルのプロンプトから対話的にモジュールを起動することにより実験をおこなうことも可能である．その場合には，モジュールから出力される共有変数の操作についてのメッセージにより，対話的にアルゴリズムの動作を確認できる．そのことを考慮すると，乱数によるアルゴリズムの動作のばらつきを実感できることから，可視化でのオーバレイに条件を満たすスリープ・ベースの値のなかでは 1000msec が適当であると考えられる．

本稿で提案したグラフ・レイアウトの評価基準は，アルゴリズムが簡潔であること，および，確率的にレイアウトが決定されるグラフを対象に値を計算可能である点が特徴である．データが確率的に変化するような場合について，可視化のグラフの形式を決定したうえで，グラフィックス要素のオーバレイがどのように変動するのかを測定するような場合に適した手法であるといえる．一方，可視化でのグラフの表現形式が複雑であるような場合には，さらなる検討が必要である．例えば，アルゴリズム 1 では，グラフを構成する全グラフィックス要素を用いて評価基準を計算しているが，着目する一部分のグラフィックス要素を対象とするなどを検討するべき点としてあげることができる．

5. おわりに

コンカレント・プログラミングのためのアルゴリズムの可視化のためにグラフ・レイアウトの評価基準を提案した．グラフを構成するグラフィックス要素のオーバレイの測定により評価基準を計算するアルゴリズムを示し，実装により動作の確認をおこなった．オーバレイの測定にはグラフィックスのレンダリングでの画像合成のための手法を用いた．また，コンカレント・プログラミングで用いられるアルゴリズムの実験のための手法 [9] においてアルゴリズムの説明に適したオーバレイの少ないグラフを選択して生成するために，評価基準を用いてパラメータの選択をおこなった．提案手法は，確率的に変化するデータを対象として可視化をおこなう場合に，グラフのオーバレイがどのように変化するのかを確認するような場合に有効である．

今後の課題には，グラフの構成または意味を考慮することによる詳細なオーバレイの検出，および，レイアウトについての他の種類の評価基準に関する研究などがある．

参考文献

[1] Adams, J. C., Koning, E. R. and Hazlett, C. D.: Visualizing Classic Synchronization Problems: Dining Philosophers, Producers-Consumers, and Readers-Writers, SIGCSE '19, New York, NY, USA, ACM, pp. 934–940 (2019).
[2] Bartram, L., Cheung, B. and Stone, M.: The Effect of Colour and Transparency on the Perception of Overlaid Grids, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 17, No. 12, pp. 1942–1948 (2011).
[3] Bruce, K. B., Danyluk, A. and Murtagh, T.: Introducing Concurrency in CS 1, SIGCSE '10, ACM, pp. 224–228 (online), DOI: 10.1145/1734263.1734341 (2010).
[4] Grissom, S., McNally, M. F. and Naps, T.: Algorithm Visualization in CS Education: Comparing Levels of Student Engagement, *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, ACM, pp. 87–94 (2003).
[5] Hansen, P. B.: Concurrent Programming Concepts, *ACM Comput. Surv.*, Vol. 5, No. 4, pp. 223–245 (1973).
[6] Jackson, D.: A Mini-Course on Concurrency, *SIGCSE*, Vol. 23, No. 1, pp. 92–96 (1991).
[7] Liu, S., Cui, W., Wu, Y. and Liu, M.: A Survey on Information Visualization: Recent Advances and Challenges, *Vis. Comput.*, Vol. 30, No. 12, pp. 1373–1393 (2014).
[8] Luxton-Reilly et al.: Introductory Programming: A Systematic Literature Review, ITiCSE 2018 Companion, ACM, pp. 55–106 (online), DOI: 10.1145/3293881.3295779 (2018).
[9] 佐藤 信：Linux シェルを用いたコンカレント・プログラミング入門，2021 年度電気関係学会東北支部連合大会予稿集，p. 4C02 (2021).
[10] Porter, T. and Duff, T.: Compositing Digital Images, *SIGGRAPH Comput. Graph.*, Vol. 18, No. 3, pp. 253–259 (1984).
[11] Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S. and Edwards, S. H.: Algorithm Visualization: The State of the Field, *ACM Trans. Comput. Educ.*, Vol. 10, No. 3, pp. 1–22 (2010).
[12] Stone, M. and Bartram, L.: Alpha, Contrast and the Perception of Visual Metadata, *Proceedings of the 16th IS&T/SID Color Imaging Conference*, pp. 355–359 (2008).
[13] Tominski, C., Fuchs, G. and Schumann, H.: Task-Driven Color Coding, *2008 12th International Conference Information Visualisation*, pp. 373–380 (2008).
[14] Zhou, L. and Hansen, C. D.: A Survey of Colormaps in Visualization, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 22, No. 8, pp. 2051–2069 (2016).