アリコロニー最適化法を用いたモデル検査の効率化のための探索戦略

熊澤 努
株式会社 SRA
kumazawa@sra.co.jp

滝本 宗宏
東京理科大学
mune@rs.tus.ac.jp

神林 靖
日本工業大学
yasushi@nit.ac.jp

要旨

モデル検査は，ソフトウェアシステムの挙動を記述したモデルが，望ましい仕様を満たすかどうかを検査する形式的検証技術である．モデルや仕様には，有向グラフを用いた状態遷移モデルによる表現を採用することが多い．モデル検査は，モデルと仕様によって構成される状態空間を探索して仕様違反を発見し，反例として返す．モデル検査では，状態の指数関数的な増加のために検査が困難になる状態爆発問題が知られているが，加えて，反例の可読性を高めることも重要である．これらの問題を解決するために，群知能の一種であるアリコロニー最適化法を用いた検査技法が提案されている．これは，状態空間を非網羅的に探索することで，状態爆発問題を軽減し，同時に可読性の高い短い反例を求める方法である．本論文では，アリコロニー最適化法によるモデル検査技術の更なる効率化のための探索戦略を提案する．提案する戦略は，アリコロニー最適化法を用いた従来の技法よりも広範囲に渡る探索を実現し，大規模なモデルに対しても反例の発見を速めて状態爆発を低減する．従来法との性能評価実験を行い，提案する探索戦略が実行時間の高速化と消費メモリの低減を向上させ，効率的な探索を実現することを確認した．

1. はじめに

モデル検査は，ソフトウェアのモデルが所望の仕様を満たすかどうかを自動検証する技術である [1, 2]．モデルはソフトウェアの振る舞いを記述したもので，オートマトンに代表される状態遷移モデルが採用されることが多い．モデル検査は，状態遷移モデルと仕様が表す状態空間上を探索して，システムの挙動に関する仕様違反を発見する．発見した仕様違反は反例として出力される．従来のモデル検査では，反例の検出に，主に深さ優先探索法に基づく網羅探索技術が用いられてきた．

本論文では，モデル検査における次の 2 点の課題に取り組む．

- 状態爆発問題：実用的なソフトウェアシステムは状態数が多く，大規模であることが多い．それゆえ，モデル検査が扱う状態空間もまた状態数が非常に大きくなるので，大規模な空間上を効率的に探索するアルゴリズムが必要である．特に，複数のプロセスから成る並行システムを扱う場合は，取りうる状態数がプロセス数に対して指数関数的に増加し，従来の網羅探索技術で効率的に検査することが難しい．
- 反例短縮問題：反例は，ユーザがシステムの不具合を特定するための手掛かりとなる診断情報で，状態遷移モデル上の路（状態列）で与えられることが多い．長大な反例は可読性が悪く，その解析に時間を要する恐れがある．そのため，短い反例を出力することが望ましい．

これらの 2 つの問題を解決するために，群知能あるいはメタヒューリスティクスを用いた検証技術が研究されている [3]．群知能の一種であるアリコロニー最適化法 [4] を用いた探索アルゴリズム ACOhg [5] は，大規模グラフを確率的に探索して最短路を効率的に発見するアルゴリズムである．ACOhg は，ソフトウェア検証，特にモデル検査への応用が試みられてきた [5, 6, 7, 8, 9, 10]．文献 [8] では，ACOhg を 2 段階に分けて実行することで，グラフ上で閉路を構成する反例を発見するアルゴリズム ACOhg-live が提案された．ACOhg-live は，従来の網羅探索に基づく検証技法よりも，状態爆発問題と反例短縮問題に有効であることが報告されている．

本論文では，ACOhg-live の性能を向上させ，更なる効率化を実現する新たな探索戦略を 2 つ提案する．

- スキップ戦略：ACOhg-live は，探索の進行に応じて探索範囲を拡大していく方法を採る．しかし，モデル検査が扱う状態空間によっては，探索が効率的に進まず，その範囲を十分に拡大できない恐れがある．その場合に，各探索の終点となる状態で探索を停止せずに継続することで，探索範囲を拡大させる．
- 差替え戦略：ACOhg-live は，有向グラフ上の路を繰り返し探索して，反例の見込みのある路を絞り込んでいく．しかしながら，反例を発見するまで十分に探索が進んでいない段階で絞り込みが行われると，以降の探索の進行が遅れる恐れがある．そこで，探索が進まなくなった時点で，絞り込みから除外された路へと探索を切り替えることによって，以降の探索を進めやすくする．

2 種類の探索戦略を組込んだ試作プログラムを実装して，従来の ACOhg-live との性能評価実験を行った．その結果，提案する探索戦略を組合せることで，実行時間とメモリ消費量が低減し，効率的に探索が進むことを確認した．

本論文の構成は次の通りである．2 節で，技術的な背景として，モデル検査と ACOhg，ACOhg-live の概要を説明する．3 節で，提案する 2 つの戦略の詳細を説明する．4 節では，提案手法を実装したプロトタイプの概要と，プロトタイプを用いて行った実験の結果を議論する．5 節では，関連する先行研究について述べ，6 節で結論と今後の課題を述べる．

2. 背景

本節では，モデル検査の概略を述べる．さらに，本論文で着目するモデル検査技法 ACOhg 並びに ACOhg-live を，文献 [8] に基づいて説明する．

2.1 オートマトン理論を用いたモデル検査

モデル検査は，ソフトウェアの振る舞いが与えられた仕様を満たすかどうかを形式的に検証する技術である．モデル検査は，オートマトン理論を用いたモデル検査 [11, 2] と記号的モデル検査 [2] とに大きく分けることができる．オートマトン理論を用いたモデル検査は，状態遷移モデルと仕様を有限オートマトンで表し，その受理可能性を判定する検査法である．有限オートマトンは有向グラフとみなせるので，判定にはグラフ探索法を使用する．一方，記号的モデル検査は，モデルの振る舞いを論理式で表して検証する技法である．本論文は，オートマトン理論を用いたモデル検査を対象とする．

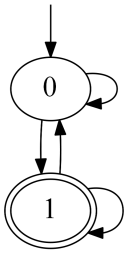


図 1: Büchi オートマトンの例

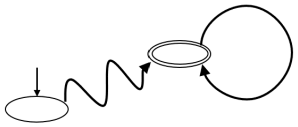


図 2: 閉路型反例の形状

オートマトン理論を用いたモデル検査では，無限語を受理する Büchi オートマトンを用いて検証を行う．有限語を受理する古典的な有限オートマトン（NFA）と同様に，Büchi オートマトンは初期状態と受理状態を持つ有向グラフで表現できる．ただし，NFA が受理状態に到達した語を受理するのに対して，Büchi オートマトンは受理状態を無限回通過する語を受理する．図 1 に Büchi オートマトンの例を示す．接続元となる状態のない矢印を付した状態 0 が初期状態であり，2 重丸の状態 1 が受理状態である．受理状態 1 自身が閉路を構成しているので，状態 0 を出発して状態 1 を無限回通る語を受理するが，状態 0 だけを無限回通る語は受理しない．このように，Büchi オートマトンの受理語は，受理状態を含む閉路とその閉路に達する路から成る（図 2）．

モデル検査の手順は次の通りである [2]．まず，仕様違反となるモデル上の路だけを受理する Büchi オートマトンを構成する．次に，構成したオートマトンが受理する語が存在するかどうかを，グラフ探索を網羅的に実行して調べる．そのために，有向グラフの強連結成分検出アルゴリズム [12] などを使用して，受理状態を含む閉路を検出する．検出した受理語は仕様に違反しているので，受理語が通過した路を反例として出力する．

2.2 ACOhg

ACOhg [5] は，大規模グラフを対象として最短路を確率的に探索するアルゴリズムである．ACOhg は，アリコロニー最適化法 (ACO) [4] に基づいている．ACO は，ユーザが事前に定めた始点（巣）から出発した多数の探索エージェント（アリ）が終点（エサ）を目指して探索空間上を探索する．エージェント同士は，路に配置したフェロモンによって相互に対話する．探索に際して，各エージェントは確率的に遷移を選択するが，先行するエージェントが配置した量の高い路ほど高確率で選択する．すなわち，フェロモンは選択する路を絞り込むために使用される．局所最適に陥ることを防ぐために，フェロモンは自然に低減（蒸発）するものとする．この手続きを繰り返し，最短路を発見する．

従来の ACO は，路の長さに関する制約がないので，エージェントは状態空間上を長時間遷移し続け，状態爆発を引き起こす恐れがある．そこで，ACOhg は，各エージェントが探索する路の長さに上限を設けることで，実行時間の超過とメモリ消費の増大を抑制して，大規模なグラフの探索を実現する．ただし，1 回の探索で終点に到達するとは限らないので，指定された回数探索を行う毎に，先行したエージェントが最後に到達した状態から後続のエージェントが遷移を開始するように始点を変更して，探索範囲を広げていく．

Algorithm 1 ACOhg [5]

1: フェロモンを初期化する;
2: **while** 探索回数が $mstep$s 回以下である **do**
3: **for all** $a \in A_{ant}$ **do**
4: a は路の長さが λ_{ant} になるか，終点に到達するまで探索する;
5: 上位の路を選択する;
6: **end for**
7: フェロモンを更新する;
8: σ_s 回に 1 回探索範囲を拡大する;
9: **end while**

アルゴリズム 1 に ACOhg の概要を示す．アルゴリズム 1 の 1 行目では，エージェント同士の相互対話に利用するフェロモンをランダムな値で初期化する．2 行目以降でエージェントによる探索を $mstep$ 回繰り返す．3 行目から 6 行目が個数 $colsize$ のエージェントの集合 A_{ant} による探索である．ここで，$mstep$ と $colsize$ はユーザが定めるパラメータである．4 行目では，エージェントは状態遷移を繰り返して探索する．フェロモンが多く配置された状態ほど遷移しやすくなるように，現在の状態からの遷移先を確率的に決定する．状態 i から状態 j へのエージェント k の遷移確率 p_{ij}^k は次式で計算する．

$$p_{ij}^k = \frac{[\tau_j]^\alpha [\eta_{ij}]^\beta}{\Sigma_{l \in N_i} [\tau_l]^\alpha [\eta_{il}]^\beta}, \quad \text{ただし，} j \in N_i. \tag{1}$$

ここで，τ_j は状態 j のフェロモン量，η_{ij} は適当なヒューリスティクス，N_i は i の後続状態の集合である．フェロモンとヒューリスティクスの寄与度をパラメータ α, β で定める．遷移の後には，現在の状態のフェロモン量を一定の割合 ξ だけ減少させる．つまり，状態 j のフェロモン量を $\tau_j = (1-\xi)\tau_j$ とする．他のエージェントが同じ路を選択する確率を低減して，局所最適に陥るのを防ぐことが目的である．

5 行目では，直前の探索と今回の探索でエージェントが通過した路の集合から上位 ι 個の路を選出することで，今後より深く探索するための始点となる状態を求める．ι はパラメータである．選出には，エージェントが通過した路の質を評価する目的関数を使用する．反例短縮問題の解決のために，短い路ほど良好な値をとるように，エージェント k の路 π^k に対する目的関数 f を次式で定める．

$$f(\pi^k) = \begin{cases} |\pi_{full}^k|, \quad (\pi_{-1}^k \text{が終点の場合}) \\ |\pi_{full}^k| + h(\pi_{-1}^k) + p_p + p_c \dfrac{\lambda_{ant} - |\pi^k|}{\lambda_{ant} - 1} \\ \qquad (\pi_{-1}^k \text{が終点ではない場合}). \end{cases} \tag{2}$$

ここで，π_{-1}^k は π^k の最後の状態，π_{full}^k は ACOhg の実行時点での始点から π_{-1}^k までの路，h は状態 π_{-1}^k に対するヒューリスティック関数である．パラメータ p_p は，終点に到達しなかった路の質を低く判定する懲罰の尺度，p_c は π^k に閉路が含まれる場合の懲罰の尺度である．

7 行目のフェロモンの更新処理では，各状態 i に配置済みのフェロモンを一定の割合 ρ だけ低減させ，$\tau_i = (1-\rho)\tau_i$ とする．エージェントの通る路の偏りを是正することが目的である．最良の路については，後続のエージェントを集中させて探索を深く進めるために，目的関数 f の値を使って，フェロモン量を高めるように更新する．8 行目では探索範囲を拡大する．終点に未到達のエージェントが最後に到達した状態に始点を変更し，フェロモン量を初期化する．

2.3 ACOhg-live

ACOhg-live [8] は，図 2 に示す閉路を成す反例を探索するモデル検査アルゴリズムである．ACOhg-live は ACOhg を 2 段階に分けて実行する．

アルゴリズム 2 に ACOhg-live の概要を示す．第 1 段の ACOhg はオートマトンの受理状態を探索する．次に，発見した各受理状態を始点として，その受理状態自身を探索する第 2 段の ACOhg を実行する．受理状態に到達した場合には探索を終了し，そうでない場合には，第 1 段に戻って探索を継続する．このとき，ACOhg-live は保持しているタブーリストに受理状態を格納する．1 つの受理状態に対して第 2 段を複数回実行することがないように，格納された受理状態を以降の探索で無視する．

Algorithm 2 ACOhg-live [8]

```
1: repeat
2:    ACOhg によって受理状態を探索する; { 第 1 段 }
3:    for all 受理状態 do
4:       ACOhg によって閉路を探索する;{ 第 2 段 }
5:    end for
6: until 第 1 段で受理状態を発見しない
```

図 3 は ACOhg-live の実行例である．図 3a は，第 1 段の ACOhg を実行した結果, 初期状態 0 を始点として受理状態 1 と 2 を発見した状況を示している．図では，エージェントが探索した受理状態までの路を灰色の状態で表す．図 3b と図 3c が第 2 段の ACOhg の実行の様子である．まず，受理状態 1 から ACOhg を実行するが，状態 1 自身に戻る路が存在せず，探索に失敗する（図 3b）．続けて，受理状態 2 を新たな始点として ACOhg を実行して，状態 2 を含む閉路の探索に成功した後，ACOhg-live の実行を終える（図 3c）．

3. 提案する探索戦略

本節では，ACOhg-live の性能向上のために，エージェントによる新たな探索戦略を提案する．

3.1 スキップ戦略

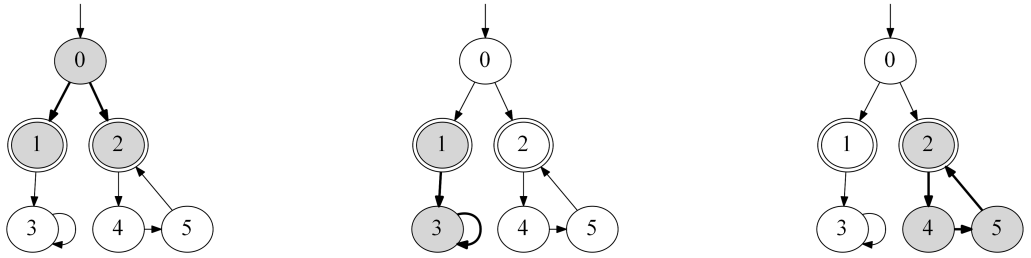
ACOhg-live の第 1 段の ACOhg は，受理状態に到達する最短路を計算する．一般に，Büchi オートマトンは複数の受理状態を含むが，図 4 の Büchi オートマトンのように，1 つの路に複数の受理状態が存在する場合には，第 1 段の ACOhg でのエージェントによる探索が十分に深く進まず，ACOhg-live の性能に大きな影響を与える可能性がある．図 4 の初期状態 0 に対して第 1 段の探索を実行すると，アルゴリズム 1 の 4 行目の探索終了条件に従って，受理状態 1 を発見して終了する．状態 1 を含む閉路は存在しないので，続く第 2 段の探索に失敗する．したがって，閉路を構成する受理状態 2 を発見するためには，状態 0 から第 1 段の探索を再度実行する必要がある．1 回目の第 1 段の探索を深く進めて状態 2 を発見することができれば，第 1 段を繰り返し実行する必要がなくなり，探索効率の向上が期待できる．

そこで, ACOhg における新たな探索戦略であるスキップ戦略を提案する．スキップ戦略は，各エージェントが路の長さが上限値 λ_{ant} に達するまで遷移を続け，探索中には終点に到達したかどうかの判定を行わない戦略である．エージェントが探索を終えた後に，探索した路が終点を通過したかを判定する．図 4 にこの考え方を適用すると，$\lambda_{ant} > 2$ ならば第 1 段の探索を 1 回実行するだけで受理状態 2 を発見できる．

Algorithm 3 スキップ戦略

```
1: フェロモンを初期化する;
2: while 探索回数が msteps 回以下である ∧ 終点に達していない do
3:    for all a ∈ A_ant do
4:       a は路の長さが λ_ant になるまで探索する;
5:       a が通過した路から，終点に到達する部分路をランダムに選択する;
6:       上位の路を選択する;
7:    end for
8:    フェロモンを更新する;
9:    σ_s 回に 1 回探索範囲を拡大する;
10: end while
```

アルゴリズム 3 に, ACOhg にスキップ戦略を組込んだアルゴリズムを示す．4 行目では，エージェントは λ_{ant} 回遷移を繰り返して探索を行う．ACOhg とは異なり, 終点である受理状態に到達してもエージェントは遷移を継続する．5 行目では，エージェントが通った路から，受理状態に到達した部分路を 1 つランダムに選択する．確率的に部分路を選択することで，エージェントの数 $colsize$ が十分大きければ，通過した受理状態を網羅的に抽出す



(a) 第 1 段: 状態 0 から受理状態の探索　**(b)** 第 2 段: 状態 1 を含む閉路の探索　**(c)** 第 2 段: 状態 2 を含む閉路の探索

図 3: ACOhg-live 実行例

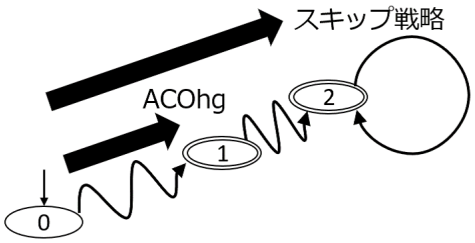


図 4: ACOhg とスキップ戦略の比較

ることが期待される．少なくとも 1 個体のエージェントが受理状態に到達した場合に，アルゴリズムは終了する．この終了条件は文献 [7] で採用している終了条件と同一であり，アルゴリズムの効率化のために必要である．5 行目で部分路を選択できない場合には，エージェントの通った路に終点が存在しないことを意味するので，従来の ACOhg と同様に，6 行目で質の高い路を選んでアルゴリズムを継続する．

3.2 差替え戦略

ACOhg において，先行エージェントが終点に達しなかった場合，後続エージェントは，より探索を深めて路を伸展させていく必要がある．しかしながら，先行エージェントが発見した路の目的関数値が好ましい場合，アルゴリズム 1 の 5 行目の選出手続きで後続エージェントが発見した路が選出されず，繰り返し探索を行ったとしても，既知の路の再選出を繰り返す恐れがある．この場合，路が終点に達しないにも関わらず，エージェントが状態空間の狭い領域に集中する．その結果，局所最適に陥ってしまい，新たな路を発見しながら効率的に探索を深めることが困難になる，

そこで，局所最適を回避して ACOhg の効率化を図る戦略として，差替え戦略を導入する．差替え戦略は，選出した路に変化がない場合に、目的関数値が好ましくない路を確率的に選択する戦略である．アルゴリズム 1 の 5 行目で選出した上位の路の集合が，選出前の集合と同一の場合には，4 行目で求めた路に差替えて路の選出をやり直す．路の差替えは次の手順で行う．4 行目から 6 行目のエージェントによる探索の終了後に，5 行目で選出したそれぞれの路について，確率 0.5 で刈り取るかどうか決定する．次に，4 行目で発見した路から上位の路を，刈り取らずに残した路に追加する．このとき，追加する路の数を刈り取った路の数と同数にし，路の数を最大 ι 個とすることで，ACOhg の 7 行目以降の手続きを変更なく実行することができる．

4. 評価実験

ACOhg-live について，提案する探索戦略を組込んだ方法と従来法とで性能評価実験を行った．本節ではその結果を報告する．

4.1 実験の概要

ACOhg-live と提案する探索戦略を実行する比較実験用プロトタイプを，Python 言語で実装した．使用した Python のバージョンは 3.6.8 である．

プロトタイプへの入力は，システムを記述した状態遷移モデル，検査する仕様を記述した Büchi オートマトン，ACOhg のパラメータの値などを記した設定情報である．提案する探索戦略を使用するかどうかも設定情報に含める．モデルと仕様はラベル付き遷移系 [13] で

与え，その記述フォーマットは Aldebaran フォーマット [14] とする．ラベル付き遷移系は状態遷移モデルの一種であり，イベント列で表されるシステムの挙動を記述するのに適している．入力はユーザが JSON 形式で記述する．プロトタイプは，モデルと仕様を読み込んだ後に，ACOhg-live を実行するために必要なパラメータと使用する探索戦略を設定する．続いて，プロトタイプは ACOhg-live を実行して反例を探索する．効率化のために，状態を探索中に必要に応じて生成する On-the-fly 検査方式 [2] を実装した．プロトタイプは，仕様違反を検出した場合には反例をイベント列で，検出しなかった場合には Python の組込み定数 None を出力する．

評価実験に使用するモデルと仕様には，分散データベースの事例 [13] を用いた．実験のためのモデルと仕様は文献 [13] のモデル記述を参考に作成した．分散データベースは，通信路を通してリング状に接続した複数のノードから成るシステムである．各ノードは，隣接するノードからの通信に応じて，自身の情報を更新する．そして，その情報をもう 1 つの隣接ノードに伝えることで，システムが保持する情報の整合性を保つ．全ノードが内部の情報の更新を終えることを要請する仕様をはじめとする 3 件の仕様 A，B，C を実験で扱うこととした．

ACOhg-live と提案する探索戦略の評価実験では，次の探索条件を用いた．

- OR：提案する探索戦略を使用せずに，ACOhg-live を実行する．
- ORT：スキップ戦略の終了条件だけを第 1 段の ACOhg に適用する．これは，文献 [8] の ACOhg とスキップ戦略とで，アルゴリズムの終了条件が異なることを考慮した探索条件である．
- SK：第 1 段にスキップ戦略を適用する．
- RE: 第 2 段に差替え戦略を適用する．
- RET：スキップ戦略の終了条件だけを第 1 段に適用し，第 2 段には差替え戦略を適用する．
- HY：第 1 段にスキップ戦略を，第 2 段に差替え戦略を適用する．

それぞれについてプロトタイプを 30 回実行し，ACOhg-live が出力した反例の長さ，実行に要した時間，消費したメモリを測定した．評価実験に使用するパラメータ設定は，先行研究である文献 [8] を元に決定した．パラメータの値を表 1 に示す． 2.2 節で述べたように，ACOhg は問題領域の経験的知識をヒューリスティクスとして活用する．本実験では，状態空間上での受理状態の位置を特定できないので，第 1 段ではヒューリスティクスを使用せずに定数値 1 とした．第 2 段では，状態をバイナリ表現に変換したときの現在の状態と探索の終点である受理状態の間のハミング距離を，終点までの距離を見積もったヒューリスティクスとした [8]．実験には HP ProDesk 600 G4 SFF を使用した．これは，OS に Windows 10 Pro を搭載し，Intel Core i7-8700, CPU 3.20GHz, RAM 16.0 GB を備えたマシンである．実行性能に影響を与える可能性を考慮して，Python に実装されているガベージ・コレクション機能を無効にして測定を行った．

4.2 実験の結果と考察

分散データベースの各仕様での測定結果を，仕様 A については表 2 と図 5 に，仕様 B については表 3 と図 6 に，そして，仕様 C については表 4 と図 7 に示す．

まず，スキップ戦略（SK）の結果を論じる．分散データベースのいずれの仕様の場合も，従来の ACOg-live（OR）より実行時間とメモリ消費量が減少し，性能の改善が見られた．終了条件をスキップ戦略と同一にした場合の結果（ORT，RET）と比較して，実行時間とメモリ消費量に関してはスキップ戦略の効率が高く，本戦略が探索性能の向上に有効であることが確認された．一方で，反例長については，スキップ戦略を使用すると長くなる傾向が現れた．スキップ戦略は探索を深める戦略なので，従来の ACOhg-live ほどには反例短縮の効果が得られなかったと考えられる．

次に，差替え戦略（RE）の結果を論じる．差替え戦略を適用すると，いずれの仕様の場合でも，従来の ACOhg-live（OR）と比べて反例長，実行時間，メモリ消費量に大きな違いは見られなかった．差替え戦略を単独で使用した場合には，性能の向上が得られるとは言えないものと考えられる．

最後に，スキップ戦略と差替え戦略を共に用いた結果（HY）を論じる．分散データベースの仕様 A を用いた実験では，実行時間の平均がすべての実験条件の中で最小となり、中央値はスキップ戦略に次いで 2 番目に小さい値となった（表 2）．仕様 B の実験の結果（表 3）によると，実行時間並びにメモリ消費量の平均，標準偏差，

表 1: 実験で使用したパラメータ設定

パラメータ	説明	第 1 段の ACOhg	第 2 段の ACOhg
$msteps$	探索回数	10	10
$colsize$	1 回の探索におけるエージェントの数	10	10
λ_{ant}	1 回の探索での路の長さの上限	20	10
σ_s	探索範囲拡大までの回数	2	2
ι	始点の保持数	10	10
ξ	探索中のフェロモンの蒸発率	0.7	0.5
a	フェロモンの下限に対する上限の比	5	5
ρ	各探索後のフェロモンの蒸発率	0.2	0.2
α	遷移時のフェロモンの寄与度	1.0	1.0
β	遷移時のヒューリスティクスの寄与度	2.0	2.0
p_p	終点に到達しなかった場合の懲罰度	1000	1000
p_c	閉路に対する懲罰度	1000	1000

表 2: 仕様 A を用いた実験結果

		OR	ORT	SK	RE	RET	HY
反例長	平均	15.23	21.67	17.50	**15.17**	17.87	16.90
	標準偏差	3.48	10.44	7.42	**3.33**	7.51	6.33
	中央値	**15**	16	16	**15**	**15**	16
実行時間 [sec]	平均	3.45	1.87	1.41	3.40	1.94	**1.40**
	標準偏差	0.70	0.24	0.39	0.68	**0.20**	0.47
	中央値	3.49	1.81	**1.25**	3.29	2.02	1.26
メモリ消費量 [KB]	平均	267.35	184.18	**154.59**	266.97	176.98	158.70
	標準偏差	35.38	15.72	23.89	41.48	**11.96**	19.72
	中央値	262.51	187.37	**148.71**	267.49	174.33	156.24

中央値が最小となった．仕様 C での実験結果（表 4）においても，実行時間とメモリ消費量のどちらの結果も最小となり，実行性能に改善が見られた．スキップ戦略だけを使用した場合と比較すると，仕様 A の実行時間以外の結果について，標準偏差を低減させる傾向を確認でき，実行性能のばらつきを抑える効果があることが期待できる．一方，仕様 B と C の実験において，反例長は最も大きいという結果だった．このことは，提案する探索戦略によって探索が効率よく進んだ一方，エージェントが状態空間の狭い領域を集中的に探索して得られる反例の短縮効果は弱まったことを示唆する．ただし，実行性能の改善と比べると，反例長の増加はわずかであると考えられる．

以上の実験結果から，提案する戦略を共に使用することで，従来法に比べて効率が向上しており，状態爆発問題を緩和する効果が期待できる．反例短縮問題については，提案する戦略で改善させることはできないものの，反例の可読性を大きく損なうほど悪化させてはいないと考えられる．ACOhg-live は古典的な探索法と比べて大きく反例長を短縮したという実験結果を Chicano と Alba は得ている [8] ことから，提案する探索戦略は，ACOhg-live に次ぐ反例短縮効果が期待できる．

5. 関連研究

群知能やメタヒューリスティクスを用いた形式検証技術は，近年活発に研究されている [3]．文献 [15] や [16] では，遺伝的アルゴリズム（GA）を用いた検査技法が提案されている．文献 [17] では，プロトコルの検証に，代

表 3: 仕様 B を用いた実験結果

		OR	ORT	SK	RE	RET	HY
反例長	平均	**27.47**	27.57	29.40	27.60	28.03	30.17
	標準偏差	**1.93**	1.94	2.74	1.96	1.97	2.31
	中央値	**26**	**26**	28	**26**	28.5	31
実行	平均	11.53	9.66	**1.16**	11.83	9.44	**1.16**
時間	標準偏差	0.68	1.18	0.37	0.73	1.07	**0.32**
[sec]	中央値	11.53	9.40	1.09	11.84	9.15	**1.04**
メモリ	平均	785.81	622.97	175.72	765.14	615.33	**164.95**
消費量	標準偏差	66.24	70.35	35.88	42.26	65.18	**28.88**
[KB]	中央値	773.68	628.91	174.52	774.07	603.93	**160.18**

表 4: 仕様 C を用いた実験結果

		OR	ORT	SK	RE	RET	HY
反例長	平均	28.93	**28.00**	29.70	29.07	29.77	31.30
	標準偏差	1.77	1.83	2.84	**1.69**	2.33	3.62
	中央値	30	**27.5**	29	30	30.5	31.5
実行	平均	12.04	9.56	1.49	11.78	4.83	**1.12**
時間	標準偏差	1.35	4.36	0.74	1.35	2.34	**0.39**
[sec]	中央値	11.71	9.16	1.41	11.42	4.55	**1.03**
メモリ	平均	770.61	483.03	176.92	726.17	349.29	**142.43**
消費量	標準偏差	129.75	193.30	44.22	52.75	124.68	**19.11**
[KB]	中央値	741.25	454.47	175.67	723.51	340.87	**139.15**

表的な群知能の 1 つである粒子群最適化法（PSO）を使った方法が提案されている．Chicano 等は，形式検証にシミュレーテッドアニーリング（SA）法を採用した [18]．加えて，従来の網羅探索によるモデル検査技法と GA，ACOhg，PSO，SA などの確率的探索技法を用いた検査技法の比較評価を行い，確率的探索の方が網羅探索よりも良好な結果が得られたという結果を報告している．

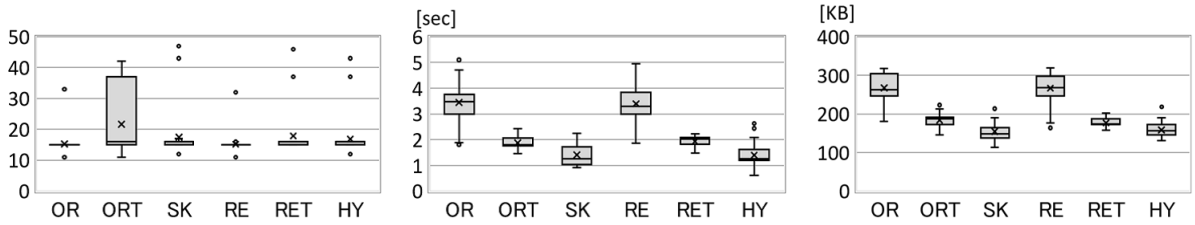
これまで述べた方法以外にも，分布推定アルゴリズム [19, 20]，モンテカルロ木探索法 [21]，PSO と重力探索法とのハイブリッド法 [22]，人口ハチコロニーアルゴリズムと SA とのハイブリッド法 [23] などを用いた検証技法が研究されている．

近年では，検証だけでなくソフトウェア工学の諸問題の解決に群知能を活用する，探索に基づいたソフトウェア工学と呼ばれるアプローチが注目されており，様々な研究成果が報告されている [24, 25]．本論文も，探索に基づいたソフトウェア工学の研究成果に位置づけられる．

6. おわりに

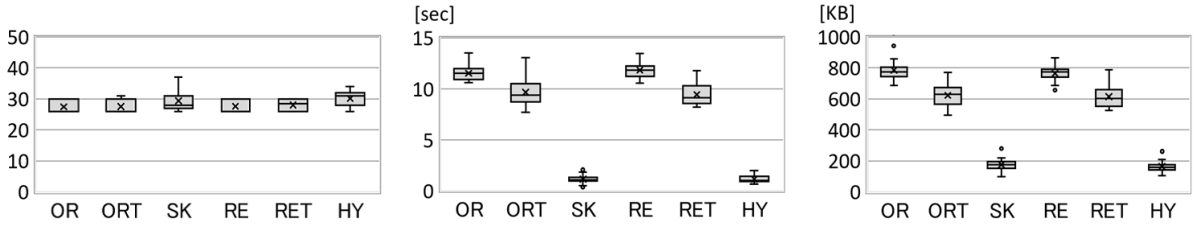
モデル検査における主要な研究課題に，状態爆発問題と反例短縮問題がある．これらの問題の解決のために，モデル検査に群知能を活用する方法が研究されている．本論文では，アリコロニー最適化法を用いたモデル検査アルゴリズム ACOhg-live の性能向上を目指し，新たな探索戦略であるスキップ戦略と差替え戦略を提案した．性能評価実験の結果，2 つの探索戦略を組合せることで，従来の ACOhg-live と比較して，実行速度とメモリ消費量については性能が向上することを確認した．反例の長さについては改善は見られないものの，可読性を大きく損なうほどには悪化しない，という結果であった．

今後の研究すべき課題について，次の点が重要である．第一に，本論文では，提案する探索戦略にヒューリスティクスを組込んだ場合について論じることができなかったので，適切なヒューリスティクスの下での探索戦



(a) 反例長 **(b)** 実行時間 [sec] **(c)** メモリ消費量 [KB]

図 5: 仕様 A を用いた実験結果の箱ひげ図



(a) 反例長 **(b)** 実行時間 [sec] **(c)** メモリ消費量 [KB]

図 6: 仕様 B を用いた実験結果の箱ひげ図

略の性能評価を行う必要がある．モデル検査で使われるヒューリスティクスは，有向モデル検査 [26] で研究されている．加えて，ACOhg-live 以外の群知能を用いる方法など，先行する検証技術との性能の比較評価も必要である．第二に，評価実験のために実装したプログラムはプロトタイプに留まるので，より実践的なプログラムに拡張することが挙げられる．従来のモデル検査で実装されている最適化法 [2] を組込むなど，実用化に向けた取り組みが必要である．

謝辞

有益なご指摘を頂いた査読者の方々に感謝する．

参考文献

[1] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, 1982.

[2] E. M. Clarke, Jr., O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model Checking*. MIT Press, 2nd edition, 2018.

[3] T. Kumazawa, M. Takimoto, and Y. Kambayashi. A survey on the applications of swarm intelligence to software verification. *Handbook of Research on Fireworks Algorithms and Swarm Intelligence*, pages 376–398, 2020.

[4] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Company, MIT Press, 2004.

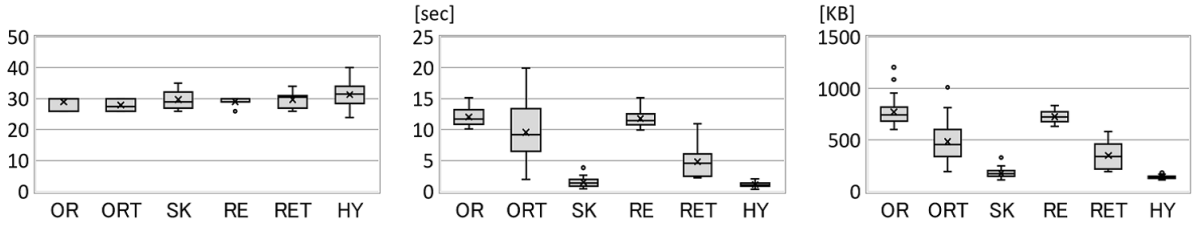
[5] E. Alba and F. Chicano. ACOhg: Dealing with huge graphs. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 10–17, 2007.

[6] E. Alba and F. Chicano. Finding safety errors with aco. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1066–1073, 2007.

[7] F. Chicano and E. Alba. Ant colony optimization with partial order reduction for discovering safety property violations in concurrent models. *Information Processing Letters*, 106(6):221–231, 2008.

[8] F. Chicano and E. Alba. Finding liveness errors with ACO. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2997–3004, 2008.

[9] T. Kumazawa, C. Yokoyama, M. Takimoto, and Y. Kambayashi. Ant colony optimization based model checking extended by smell-like pheromone. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 16(7), 4 2016.



(a) 反例長 **(b)** 実行時間 [sec] **(c)** メモリ消費量 [KB]

図 7: 仕様 C を用いた実験結果の箱ひげ図

[10] T. Kumazawa, K. Takada, M. Takimoto, and Y. Kambayashi. Ant colony optimization based model checking extended by smell-like pheromone with hop counts. *Swarm and Evolutionary Computation*, 44:511–521, 2019.

[11] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, 1986.

[12] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[13] J. Magee and J. Kramer. *Concurrency: State Models & Java Programming*. John Wiley & Sons, Inc., 2nd edition, 2006.

[14] Aldebaran Format: https://www.mcrl2.org/web/user_manual/language_reference/lts.htm.

[15] E. Alba and J. M. Troya. Genetic algorithms for protocol validation. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 870–879, 1996.

[16] R. Yousefian, V. Rafe, and M. Rahmani. A heuristic solution for model checking graph transformation systems. *Applied Soft Computing*, 24:169–180, 2014.

[17] M. Ferreira, F. Chicano, E. Alba, and J. A. Gómez-Pulido. Detecting protocol errors using particle swarm optimization with java pathfinder. In *Proceedings of the High Performance Computing & Simulation Conference*, pages 319–325, 2008.

[18] F. Chicano, M. Ferreira, and E. Alba. Comparing metaheuristic algorithms for error detection in java programs. In *Proceedings of the Third International Conference on Search Based Software Engineering*, pages 82–96, 2011.

[19] J. Staunton and J. A. Clark. Searching for safety violations using estimation of distribution algorithms. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 212–221, 2010.

[20] J. Staunton and J. A. Clark. Finding short counterexamples in promela models using estimation of distribution algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 1923–1930, 2011.

[21] S. Poulding and R. Feldt. Heuristic model checking using a monte-carlo tree search algorithm. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1359–1366, 2015.

[22] V. Rafe, M. Moradi, R. Yousefian, and A. Nikanjam. A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations. *Applied Soft Computing*, 33(C):136–149, August 2015.

[23] N. Rezaee and H. Momeni. A hybrid meta-heuristic approach to cope with state space explosion in model checking technique for deadlock freeness. *Journal of AI and Data Mining*, 8(2):189–199, 2020.

[24] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11:1–11:61, 2012.

[25] 半田久志. 探索に基づいたソフトウェア工学：SBSE − ソフトウェア工学におけるメタヒューリスティックスの援用 − . **知能と情報**, 24(6):224–229, 2012.

[26] S. Edelkamp, V. Schuppan, D. Bosnacki, A. Wijs, A. Fehnker, and H. Aljazzar. Survey on directed model checking. In *Proceedings of the 5th International Workshop on Model Checking and Artificial Intelligence*, volume 5348 of *Lecture Notes in Computer Science*, pages 65–89. Springer, 2008.