ACOに基づくモデル検査の匂いフェロモンを用いた拡張

高田 圭一郎[1,a)]　滝本 宗宏[1,b)]　熊澤 努[2,c)]　神林 靖[3,d)]

概要：本論文では，ACO(蟻コロニー最適化法) の改良アルゴリズムを用いた，効率的なモデル検査手法を提案する．モデル検査とは，ハードウェアやソフトウェアの設計が，目的の仕様を満たしているか，モデル検査器によって自動的に検証する技術である．設計は，状態遷移系のモデルとして表現し，仕様は時相論理によって指定する．一般的なモデル検査器は，決定的アルゴリズムに基づいて動いているので，検証に膨大な実行時間とメモリが必要になる場合がある．これは状態爆発問題と呼ばれる．状態爆発問題を低減する手法の一つに，蟻の振る舞いを真似たメタヒューリスティクスである ACO に基づく手法が提案されている．本論文では，ACO を，餌から発せられる匂いに相当する特殊なフェロモンによって拡張した，新しい ACO に基づくモデル検査法を提案する．匂いフェロモンは最終状態から開始状態に向けて，到達に必要なホップ数をモデル内の各辺に記録する．本手法では，蟻はホップ数が減少する方向へ最終状態を探索するので，効率的に検査を進めることができる．本手法に基づいた検査器を実現し，実験を行ったところ，解の品質を下げることなく，実行時間を短縮できることが分かった．

キーワード：モデル検査，蟻コロニー最適化法，群知能，状態爆発問題

1. 導入

モデル検査 (model checking) とは，状態遷移系で記述したソフトウェアの設計 (以降，モデル，model と呼ぶ) がユーザによって指定された性質，すなわち**仕様 (specification)** を満たすかどうか検査する技術である．モデル検査は通常，**モデル検査器 (model checker)** と呼ばれるツールによって自動的に行われる．一般的なモデル検査器は，決定的アルゴリズムを用いた手法によって検査を行う．この手法は，探索空間を網羅的に調べるので，非常に多くの計算資源を必要とする場合があるという**状態爆発問題 (state explosion problem)** が知られている．状態爆発問題を低減する手法の一つに，非決定的手法である**蟻コロニー最適化法 (ant colony optimization，以降 ACO と呼ぶ)** [1], [2], [4], [5] に基づいた手法が提案されている．ACO は**群知能 (swarm intelligence)** と呼ばれるアルゴリズムの一種で，自然界の蟻が餌を集める振る舞いからヒントを得たメタヒューリスティクスに分類される統計的アルゴリズムである [3]．ACO では，蟻に相当するエージェントが，人工的なフェロモンに誘導されながらモデル内を探索し

[1] 東京理科大学
[2] 株式会社 SRA
[3] 日本工業大学
[a)] k-takada@cs.is.noda.tus.ac.jp
[b)] mune@cs.is.noda.tus.ac.jp
[c)] tkumazawa@acm.org
[d)] yasushi@nit.ac.jp

て，組み合わせ最適化問題の解を求める．この性質から，探索を局所化できるので，計算資源を節約することができる．

モデル検査器の大きな特徴の一つは，モデルが仕様に違反している場合，**反例 (counter example)** としてエラートレースを表示する点である [6]．表示する反例は，長さが短ければ短いほどユーザにとって理解しやすい．ACO に基づく手法には，従来のモデル検査器はよりも短い反例を提示できるという利点もある．

ほとんどのモデル検査技術は，大きく分けて安全性・活性の二つの性質を取り扱う [13], [16]．安全性は望ましくない事象が絶対に起きないという性質であり，活性は最終的に望ましい事象が起こるという性質のことを指す．本研究では，安全性について取り扱う．安全性の違反を発見するということは，モデルを表す有向グラフの到達可能性の問題に帰着することができる．すなわち，安全性のモデル検査は，初期状態から最終状態 (仕様を満たさない状態) を探索するという問題に単純化することができる．その時発見された初期状態から最終状態までの経路はモデル検査における反例に相当する．

ACO に基づくモデル検査では，蟻は最終状態を目指して有向グラフ内を探索する．蟻は過去に蟻が通った経路に散布されたフェロモンの強さに基づいて，次に進む方向を確率的に選択する．大きなフェロモン値を持つ経路が多くの蟻によって選択される傾向があるので，他の蟻が最終状態への比較的短い経路を発見することに貢献する．フェロモンは時間とともに蒸発するので，短い経路には強いフェロモンが残りやすい．すなわち，蟻は，より質の高い短いを徐々に選択していくようになる．

著者らは，先行研究において，さらに蟻を最終状態へ導きやすくするために，餌場から放出する匂いを模して最終状態からもフェロモンを散布する手法を提案した **?**．この匂いフェロモンは蟻を効率的に誘引し，探索空間をさらに局所化するので，従来の ACO の手法と比較して反例の品質 (長さ) を犠牲にすることなく，メモリ使用量を減少させ，実行時間を短縮することができる．

本論文では，この匂いフェロモンにさらなる改良を加えた手法を提案する．先行研究では，最終状態から周囲に拡散するフェロモンは蒸発せず一定の強さを維持しているので，フェロモンが広範囲に拡散した際，蟻が最終状態に正確に誘導されない可能性があった．この問題に対処するために，匂いフェロモンに最終状態からのホップ数を持たせることを考える．ホップ数は匂いフェロモンが辺に散布されるたびに増加させ，その値は有向グラフにおける辺の始点の状態に記録する．匂いフェロモンが，すでに散布済みの状態に達すると，匂いフェロモンのホップ数と状態のホップ数を比較し，より小さい値で状態のホップ数を更新する．蟻は，一旦匂いフェロモンの散布済み状態に達すると，ホップ数が減少する方向に状態をたどることによって，さらに効率的に最終状態へ誘導されることができる．

本論文の構成は以下のとおりである．次の節ではモデル検査技術に関する知識の説明を行う．第 3 節では ACO をベースにしたモデル検査についてさらに詳しい説明を行う．第 4 節で本手法のアルゴリズムの説明を行い，第 5 節で本手法を用いた実験方法，及びその結果を示す．第 6 節で結論を述べる．

2. モデル検査の問題

本節ではモデル検査についての概要を与え，その問題点を説明する．モデル検査は，ソフトウェアの妥当性を検証する形式的で自動的な技術である [7]．モデル検査はソフトウェアエンジニアリングで最も成功した研究テーマの一つであり，通信プロトコルの検証にも応用されている [10]．

モデル検査の手順は次の通りである．

(1) モデルの設計を状態遷移系で表す
(2) 検査したい仕様を Linear Temporal Logic (以降，LTL と呼ぶ) [17] のような時相論理式で表現する
(3) 状態遷移系が仕様を満たしているか検査する

例えば，最もよく用いられているモデル検査器の一つである SPIN [12], [13] は，Promela という専用言語でで記述したモデルが LTL で表した仕様を

満たすかどうかを検査する．SPIN では，モデルと仕様の否定を Büchi オートマトンに変換し，受理状態までの経路を網羅的に探索する．幾つかの経路がもし発見された場合，そのモデルは仕様を満たさないことを意味している．また，その経路は仕様に対する反例であると見なすことができる．逆に，経路が発見されなかった場合，モデルは仕様を満たしていることを意味する．

モデル検査では，状態遷移系が非常に大きなものになる傾向がある．さらに，状態数はモデルの大きさに対して状態数は指数関数的に大きくなる．この状態数の指数関数的な増加を状態爆発と呼ぶ．このような理由で，モデル検査は非常に大きな計算資源を必要とし，検査が困難になる場合もある．状態爆発を低減する試みは，状態数を減少させる Partial Order Reduction [15] という手法や，各探索における使用メモリ領域を削減する Bitstate Hashing [13] といった手法が提案されている．また，この状態数の多さは，生じる反例を長大にする可能性を含んでいる．反例は，仕様を満たさない点に対処する指針としてユーザに提示されるので，ユーザにとって分かりやすいことが望ましい．先述したように，反例は．有向グラフ上の経路として示されるため，その長さが十分に短ければユーザにとって理解しやすいと考えることができる．本論文では，モデル検査が持つ，これらの状態爆発と長大な反例の二つの問題について，メタヒューリスティクスの一つである ACO を用いて対処することを試みる．

3. ACO に基づくモデル検査

この節では，ACO の基本原理と．その派生手法で状態数の大きなモデルの検証を目的とした Alba らによる ACOhg [1], [2], [4] について説明する．また ACOhg の拡張として著者らが提案した EACOhg [14] についても説明する．

3.1 ACO

ACO は，探索を行うメタヒューリスティクスの一つで，蟻が巣穴から餌を探る振る舞いをヒントにしている [9]．ACO は経路問題，割り当て問題，

```
procedure ACO
    Initialization
    while terminationCondition do
          ConstructAntSolutions
          UpdatePheromones
    od
end procedure
```

図 1　ACO の疑似コード

スケジューリング問題のような多くの最適化問題に効果的であることが知られている．ACO をモデル検査に応用する場合，取り扱う問題は有向グラフで表されたソフトウェア，もしくはハードウェアモデルであり，巣と餌場はそれぞれ初期状態と最終状態に相当する．ノード間の経路はモデル検査の候補解，すなわち反例を表す．

ACO では蟻に相当するエージェントがフェロモンを用いた間接的なコミュニケーションを取りながら短い経路を探索する．最適化するべき解は，蟻が探索する経路である．モデル検査の場合，最適化するべき対象は経路の長さである．フェロモンの影響力は蟻が通った経路内の各辺に持たせている重みの変化によって表される．フェロモンは，蟻が最終状態へのより短い経路を選択するよう案内する役割を持っている．

ACO は Dorigo らによって提唱された Ant System(以降，AS と呼ぶ) [8] というモデルが基礎となっている．しかし，ほとんどの場合において，AS は他のヒューリスティクスほど強力ではないので，実用化に利用するためには何らかの拡張が必要である．その拡張の中でも強力なものの一つに Max-Min Ant System (MMAS) [18] がある．MMAS が，反復アルゴリズムの一種であり，前の反復にける最適解に基づいて次の反復のフェロモン値を更新するだけでなく，局所解に陥ることを避けるためにフェロモン値に上限と下限を設定している．

図 1 に ACO のアルゴリズムを示す．基本的に，ACO は経路を探索する *ConstructAntSolutions* のステップと，フェロモン値を更新する *terminationCondition* ステップを繰り返す．また，条件 *terminationCondition* は解が幾つか見つかるか，

指定された反復回数に達した場合に満たされる．

Initialization のステップでは，蟻が探索する前に各辺のフェロモン値を初期化する処理を行っている．

ConstructAntSolutions のステップでは，ある状態に存在する蟻をそれぞれ確率的に次の状態へ遷移させている．全ての蟻は，探索開始時点では初期状態に位置していて，各ステップごとにモデル内を移動する．その際に次に遷移する状態は以下の式によって決定される．

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\Sigma_{l \in N_i}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \quad if\ j \in N_i \qquad (1)$$

式 (1) の N_i は蟻が状態 i から遷移可能な状態の集合である．j は蟻の 1 ステップ後の行き先の状態である．τ_{ij} は，辺 (i, j) に散布されているフェロモン値である．η は最終状態への遷移回数を表すヒューリスティクス値で，実際の数よりも小さい値に設定される．この値は最終状態の位置や，LTL に記述されている仕様や性質の長さに基づいて推測的に設定される．α と β は経験的に設定されるパラメータであり，ヒューリスティクス値やフェロモン値の及ぼす影響力を調整するために設定される．蟻が指定された回数遷移した，もしくは幾つかの候補解が発見された場合，*UpdatePheromones* ステップが実行される．

UpdatePheromones ステップは各辺のフェロモン値を更新する．各辺に設定されたフェロモンの強さは辺の解への適切性が高いと増加する．各辺のフェロモンは以下の式によって更新される．

$$\tau_{ij} \leftarrow (1 - x_i)\tau_{ij} \qquad (2)$$

式 (2) の x_i は，0 から 1 の間で設定される実数値であり，フェロモンの蒸発率を表す値である．この更新プロセスを通じて，以前選択された辺の優先度は，再び別の蟻に選択されるまで段階的に減少する．

3.2 ACOhg

従来の ACO では，非常に多くのノードを持つ状態遷移モデルを探索するのは困難である．なぜなら，モデルの持つ辺の数は最悪の場合ノード数の 2 乗となるので，そのフェロモン値の記録に非常に多くのメモリが必要となるからである．特に並行システムのモデルのサイズは巨大であることが知られている．例えば，Dining Philosophers のモデルのサイズは，Philosopher の数を n として，状態数が 3^n であり，指数関数的に状態数が増加するのがわかる．これを同一の状態へ再び到達することを禁止するような単純な手法で解決しようとすると，蟻が袋小路に陥ってしまったり，仮に最終状態に到達したとしても，それまでに必要以上の経路を彷徨ってしまうなどの問題を生じてしまう可能性がある．したがって，従来の ACO に基づくモデル検査では，解を発見することなく状態爆発に陥ってしまう可能性があった．さらに致命的な問題として，従来の ACO では初期化する際，全ての辺に対してフェロモン値を設定する必要があるので，非常に大きなメモリを必要とする．

この問題を緩和するために，Alba らは従来よりも少ないメモリ消費量で探索を行う手法として ACO for huge graphs(ACOhg) [1], [2] を提案した．ACOhg の基本的な考え方は，一段階で蟻の移動可能なステップ数に上限 λ_{ant} を与えることである．この探索方法によって実行時間とメモリ消費量を抑えることができる一方で，移動ステップ数を制限することによって蟻が最終状態まで到達できなくなる可能性がある．すなわち，λ_{ant} は最終状態を発見できるような適切な値に設定しなければならない．ACOhg で λ_{ant} を決定する手法には二つの手法，*expansion technique* と，*missionary technique* が提案されている．*expansion technique* は，現在の λ_{ant} の値で最終状態を発見できなかった場合，λ_{ant} を与えられたパラメータ分だけ増加する．そして，新たに設定した λ_{ant} に基づいてさらに大きな領域を探索するようにする．これらの振る舞いを **図 3** に示す．開始時は λ_{ant} を小さく設定し，最終状態が見つかるまでの間徐々にその値を増加させていく．*expansion technique* によって λ_{ant} は必要分だけ増加されるので，メモリ消費を抑えることができる．また，ACO の単純な拡張であることから，比較的簡単に実装を行うことができる．一方，λ_{ant} が増加すればするほど，蟻は

従来の ACO にに似た振る舞いに近づいていく．

missionary technique は，*expansion technique* と異なり，探索を初期状態でなく，前のプロセスで探索していた領域内の任意の状態から開始する．すなわち，古いフェロモンを無視して探索を行う．これらの振る舞いを **図 4** に示す．新たに開始する状態が，蟻が前のプロセスで到達した状態の中から選択されるので，ACO は，λ_{ant} を更新することなく徐々に探索範囲を広げていくことができる．その結果，各プロセスで要する実行時間やメモリ使用量を一定にすることができる．

双方の手法にとも，フェロモンの強さは適合度関数に基づいて決定される．適合度関数の返り値は各蟻の通った経路に対するペナルティに相当し，経路内に循環が含まれていた場合，最終状態に到達できなかった場合に特に大きくなる．適合度関数 f によって計算された値の中で最も小さな値を有する経路 a^{best} が選択される．これらの値を用いて，次のように散布するフェロモン値が決定される．

$$\tau_{ij} \leftarrow \tau_{ij} + \frac{1}{f(a^{best})}, \forall (i,j) \in a^{best} \qquad (3)$$

ACOhg は MMAS に基づいているので，フェロモンの上限，下限 τ_{max}，τ_{min} を，適合度関数を用いて以下の式の通り決定する．

$$\tau_{max} = \frac{1}{\rho f(a^{best})}, \qquad (4)$$

$$\tau_{min} = \frac{\tau_{max}}{a} \qquad (5)$$

式中 (4)，(5) の ρ と a は，フェロモン値の幅を決める定数である．また *missionary technique* ではこの関数を用いて，次のプロセスで新たに開始する状態を決定する．

3.3 EACOhg

ACOhg では，ある蟻が最初に最終状態まで到達するまではモデル内をランダムに探索しなければならない．一方，自然界の蟻はフェロモンがなくても餌の匂いを手掛かりに探すことができる．

そこで，ACOhg に匂いを模したフェロモンを新たに加えることで改良した手法 EACOhg [14] を

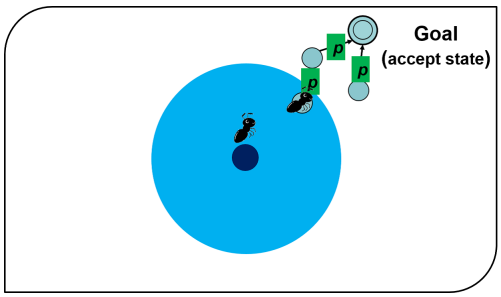


図 2 匂いフェロモンの散布

提案した．この匂いを模したフェロモンをゴールフェロモンと呼ぶことにすると，このゴールフェロモンの強さは通常のフェロモンよりも大きい値として設定する．つまり，ゴールフェロモンが辺に散布されると，**図 2** に示すように，蟻は他の辺よりもゴールフェロモンを持つ辺を優先的に選択するようになる．ゴールフェロモンを保持するためにメモリ使用量が若干増加することは懸念されるものの，探索領域をさらに局所化させることによってさらに早く最終状態を発見できたり，より短い反例を発見できたりすることが期待される．

ゴールフェロモンに蟻が遭遇したということは，現在の状態から最終状態へ進む経路が存在していることを意味している．そこで蟻を最終状態まで誘導させるために，ゴールフェロモンには以下の性質を持たせている．

(1) 他のフェロモンより強い
(2) 不揮発性である
(3) 最終状態からその周辺に拡散される

上記の特性によって蟻は最終状態への比較的短い経路を選択するよう誘導されるので，短い反例を発見しやすくなる．

通常のフェロモンは MMAS に従って振舞うので，その値は大きくても τ_{max} 以下となる．ゴールフェロモンは通常のフェロモンより強い誘引効果を示すように，τ_{max} より大きな値で設定している．また，通常のフェロモンは時間とともに蒸発していくが，ゴールフェロモンは常に一定の値を持っている．これは，現実世界では餌場からは絶えず匂いが発せられていることに基づいている．

ゴールフェロモンは最終状態から逆側に少しず

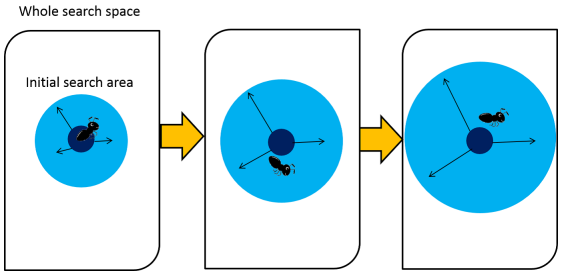


図 3　Expansion Technique

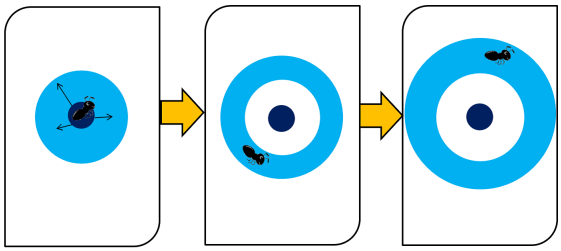


図 4　Missionary Technique

つ拡散する．このとき，フェロモンはそれまでに散布された辺に隣接する辺からランダムに選ばれた幾つかの辺にだけ散布するようにする．この散布方式は，実行時間とメモリ消費量を抑制することに貢献する．

4. 提案手法

EACOhg は，多くのモデルにおいて短い反例を発見する点では効果的ではあるが，大きなモデルに対しては実行時間と消費メモリ量をほとんど削減できない場合がある．このような大きなモデルでは，EACOhg が広範囲にゴールフェロモンを散布するので，ほとんどのゴールフェロモンが初期状態および最終状態から離れた領域に拡散してしまい，蟻がゴールフェロモンを見つけたあと，最終状態まで到達するのに時間を要してしまうのである．

この問題を緩和するために，本手法では，ゴールフェロモンに最終状態への到達までに必要なホップ数の情報を持たせるように改良した．ゴールフェロモンは最終状態からカウントを開始し，辺を逆に辿る度にそのカウントを増加させる．それと同時に，ゴールフェロモンは辺の終点の状態のホップ数より，始点の状態のホップ数が多い辺にだけその値を付与する．より詳細な説明を行うために，各エッジを順序付けしたペアの表現 (src, dst) として表現する．ここで，src および dst はそれぞれ始点の状態および終点の状態である．ゴールフェロモンが辺 (src, dst) を通る際のフェロモン散布過程は，以下の 2 つのステップ: ホップ数の記録とゴールフェロモンの散布からなる．

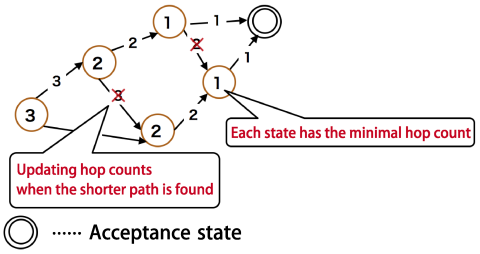


図 5　最小ホップ数の更新

ホップ数の記録

最初のステップでは，ゴールフェロモンが src におけるホップ数のカウントを行う．src でのホップ数は dst でのホップ数より 1 増加させることで決定する．これは，辺 (src, dst) を dst から src へ逆に辿るためには 1 ホップ必要であることに基づいている．現時点で，src がこれからカウントするホップ数よりも大きい値を既に持っている場合，あるいはホップ数を持っていない場合，ホップ数が新たに src に記録される．それ以外の場合は，src のホップ数を変更しない．この振る舞いを 図 5 に示す．

ゴールフェロモンの散布

次のステップでは， 図 6 に示すように，各辺において，始点の状態が終点の状態より 1 だけ大きいホップ数を持つ辺に対してゴールフェロモンを選択的に付与する．このゴールフェロモンの拡散方式によって，必要以上の辺にフェロモンを付与させることを抑制し，最終状態への最短経路の一部である状態列をフェロモン値によって表現することができる．

一旦，初期状態から最終状態までの経路を発見すれば，その経路が反例になるので，ユーザによ

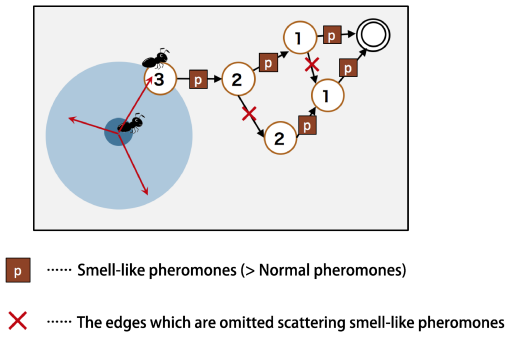


図 6　ゴールフェロモンの散布

り短い反例を提示することができる．

5. 実験

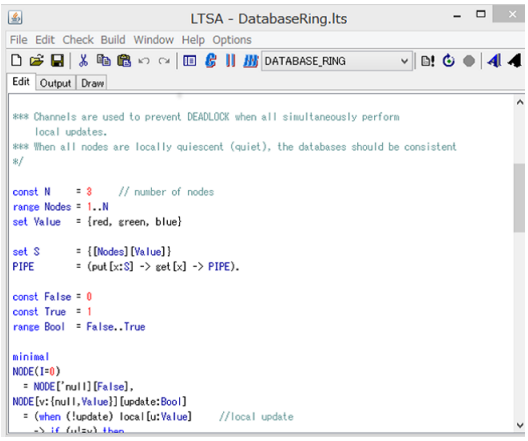
5.1 実験方法

本手法の効果を示すために，先行手法である EACOhg と本手法のアルゴリズムをそれぞれ **LTSA(Labeled Transition System Analyzer)** [11], [16] というモデル検査器上で実現した．LTSA は容易にカスタマイズ可能なモデル検査器の一種であり，**FLTL(Fluent Linear Temporal Logic)** [11] の検査をサポートしている．FLTL は Labeled Transition Systems として記述されたイベントに基づのシステムに特化した LTL の一種である．

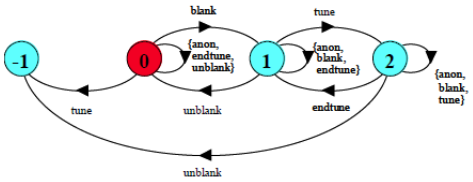
例えば，図 **7**(a) に示すように LTSA の入力として幾つかの最終状態を持つモデルを与えると，LTSA は図 7(b) に示すように，単一の最終状態を持つモデルを生成する．このモデルにおける最終状態は，安全性の検査における最終状態に対応している．最終状態が 1 つである性質によって，モデルの取り扱いを簡略化させることができる．

図 7(b) に示す通り，0 で番号付けした赤の状態は初期状態であり，-1 で番号付けした状態は最終状態である．LTSA は Aldebaran 形式で記述された有向グラフを生成する．

本研究では，提案手法の性能を示すために 2 つの実験を行った．最初の実験では提案手法と EACOhg のモデルの大きさにおけるスケーラビリティの比較を行った．それぞれの手法で用いるパラメータは表 1 の通りの設定とした．本手法は EACOhg の単純な拡張であるので，パラメータも 2 つの手法で同じ値を用いている．表内の P_p，P_c はそれぞれ最終状態に到達しなかった場合，経路内に循環が存在した場合に適合度関数に付加されるペナルティ項である．*phspeed* は 1 ステップでゴールフェロモンが拡散される速さである．このパラメータは本実験で新たに追加したものである．実験では，2 つの手法を，我々は 3 つの異なるサイズのモデル上で実行し，比較を行った．



(a) 入力モデル



(b) 生成モデル

図 7　Models of LTSA

2 つ目の実験では，提案手法の性能における *phspeed* の効果を調べた．本手法と EACOhg の 2 つの手法において，表 1 の *phspeed* 以外のパラメータを全て固定し，*phspeed* の値を 0 から 19 まで変化させて繰り返し実験を行った．

5.2 実験結果

最初の実験では状態数 5,884 のモデル A，状態数 39,274 のモデル B，状態数 266,218 のモデル C の 3 つのモデルを用意した．それぞれのモデルは Mutex fluent，DeadlockFreePhilosophers，

coefficients	values
mstep	100
colsize	10
α	1.0
β	2.0
η	1.0
ρ	0.2
a	5
P_p	70
P_c	70
$phspeed$	10

表 1 Settings of coefficients

DatabaseRing のパラメータを調整して生成したものである．それぞれの実験について，結果を実行時間，反例の長さ，消費メモリ量，ゴールフェロモンの散布量の 4 つの項目で評価を行った．各手法，各モデルについての実験では，それぞれ 100 回繰り返し実行し，各項目においてその平均を求めた．そして，その平均値の有意性を調べるために t 検定を行った．

各モデルおよび各項目に対して設定した帰無仮説は，提案手法の平均の結果が EACOhg の結果と等しいことである．t 検定の p 値が 0.05 より小さい場合，平均の差は統計的に有意であると考える．

モデル A に関しての結果は**図 8** を見る限り，全ての項目において EACOhg を上回っている．反例の長さ，実行時間，消費メモリ量，ゴールフェロモン散布量の各項目の p 値は 0.899，0.055，0.092，0.115 であった．この値は，本手法と EACOhg の結果にはあまり大きな有意差が無かったことを示している．

図 9 の通り，モデル B において，本手法の結果はメモリ使用量を除いて EACOhg の値を上回っている．各項目の p 値は 1.675×10^{-14}, 7.214×10^{-4}, 0.164, 6.794×10^{-4} であった．この結果から，本手法はメモリ使用量を除いて EACOhg よりも優れていると言える．一方，メモリ使用量に関しては，2 つの手法間でほぼ同等の結果であったと考えることができる．

モデル C では**図 10** に示す通り，全ての項目において EACOhg より結果が上回った．実行時間と

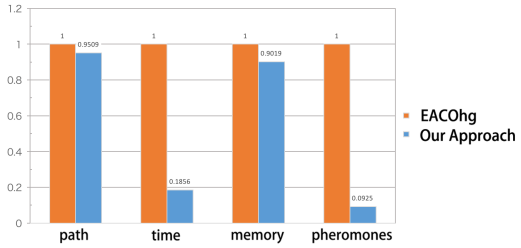


図 8 モデル A

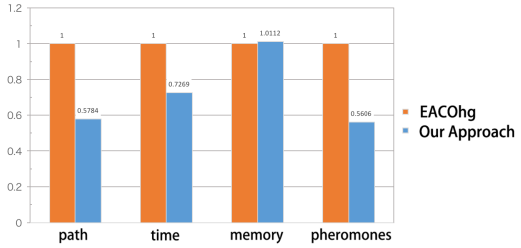


図 9 モデル B

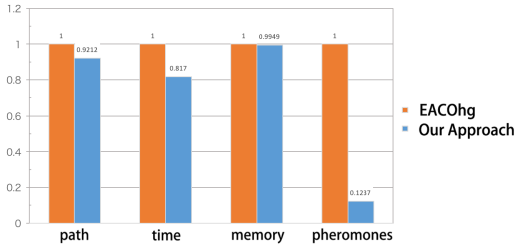


図 10 モデル C

散布フェロモン数の p 値はそれぞれ 6.313×10^{-5}, 9.122×10^{-17} と有意差が出ている．一方，反例の長さと消費メモリ量に関しては，p 値がそれぞれ 0.320，0.326 であることから，本手法が優れているとは言い難い．

2 つ目の実験では本手法の性能における $phspeed$ の影響の強さを調査した．この実験では状態数 39,274，遷移数 208,509 のモデルを用いた．前述した通り，この実験では $phspeed$ を 0 から 19 まで変化させ，他のパラメータを固定させて，全て同一のモデルを用いている．各手法のそれぞれの $phspeed$ において，100 回繰り返し実行し，最終状態への到達率，反例の長さ，メモリ消費量の 3 つの項目について比較を行った．**図 11** は $phspeed$ を変化させて反例の発見率がどう変化するかを示したグラフである．この値はモデル検査の成功率を

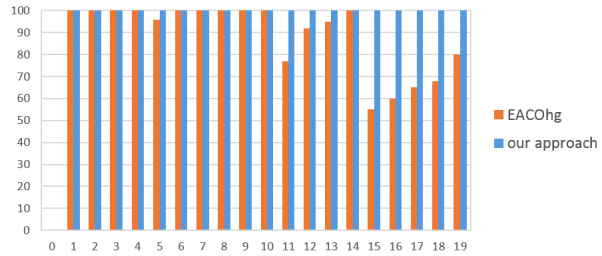


図 11 Values of *phspeed* vs. Rates of detecting counter examples

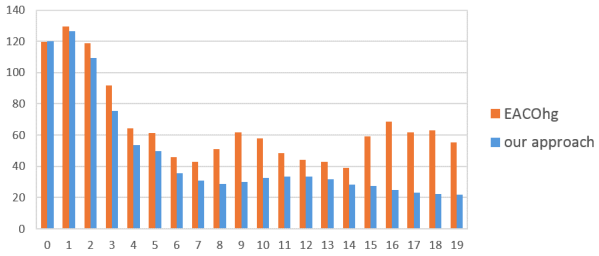


図 12 Values of *phspeed* vs. Lengths of paths

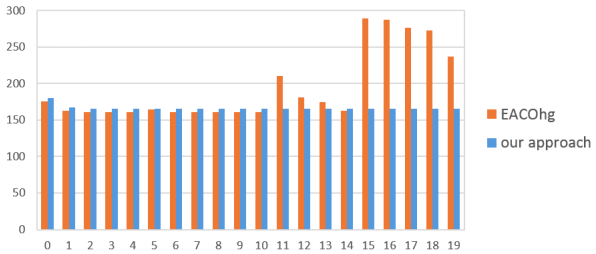


図 13 Values of *phspeed* vs. Memory consumptions (KB)

意味しているので，本手法では *phspeed* の値に関わらず，成功率は 100% を維持していた．しかしながら，EACOhg では *phspeed* を増加させるにつれ，成功率が低下する．すなわち，EACOhg を用いてモデル検査を行う場合，*phspeed* の値を注意して設定する必要がある．この結果はゴールフェロモンの飽和に起因している．EACOhg では，ゴールフェロモンの強さで蟻を最終状態まで導いているが，一旦ゴールフェロモンが飽和してしまうと，蟻は進む方向を決定付けることができなくなってしまう．一方，本手法では，フェロモンが飽和しても，ホップ数に基づいて蟻を最終状態まで導くことができる．したがって，この結果は，ホップ数の導入が，検証の成功率の向上に貢献していることを示している．

図 12 は *phspeed* を変化させた時の，本手法と EACOhg における反例の長さの変化を示している．この結果が示すように，本手法は *phspeed* が 1 以上の場合において，EACOhg よりも短い反例を見つけることができている．さらに，その効果は *phspeed* が大きくなるにつれ顕著になっている．この結果は，比較的大きな *phspeed* を設定しても，各状態が持つホップ数によって蟻を最終状態まで導くことができることを意味しており，速さが増加するにつれて成功率が増加するのと同じ理由から生じると考えられる．

最後に，メモリ消費量に関する結果を **図 13** に示す．*phspeed* が 0 から 14 の場合において，提案手法と EACOhg で必要なメモリ量はほぼ同等である．その理由は，本手法がホップ数を格納するための追加の領域を必要とする為である．しかし，本手法は EACOhg に比べて短い経路を効率的に見つけることができ，長い経路を見つけるより必要となるメモリ量が少なくて済むので，14 より高い *phspeed* の値において，優れた結果を出している．この結果は，ゴールフェロモンの拡散方法の違いによって生じている．EACOhg では，ゴールフェロモンはランダムに拡散されているので，ゴールフェロモンが広く拡散されるにつれて，蟻は最短経路で最終状態に到達する可能性が低くなっていく．したがって，EACOhg では，蟻が歩いた道のりを長く記録するためにより多くのメモリを消費する必要が生じる．一方，ホップ数の導入によってゴールフェロモンは経路が最短になるように拡散されるので，蟻もそのような経路を優先的に選択するようになる．したがって，提案手法は，大きな *phspeed* においても EACOhg と比較して，短い経路を格納することが十分にできるので，メモリ消費量を削減できる可能性が高い．2 つ目の実験を通して，*phspeed* が上がるにつれて提案手法は EACOhg よりも，成功率，反例の長さ，メモリ消費量に関して優れた結果が出るということを結論づけることができる．

5.3 考察

前節で述べた通り，最初の実験において，2 つの手法の間で有意な差が出ないモデルおよび項目があった．その考えられる理由を以下に示す．

モデル A ではモデルのサイズが結果に影響したと推測される．事実，モデル A は実験に用いたモデルの中で最も小さく，そのためにゴールフェロモンが広範囲に広がる前に蟻がゴールフェロモンに到達することができる．以上から，ゴールフェロモンの散布が最終状態の付近周辺に限定されている状態であれば，EACOhg も十分に機能する．したがってモデル A では EACOhg も十分に機能していたと考えられる．

モデル B において，提案手法は EACOhg に比べてメモリ消費量を除く全ての項目において優れていた．特に，本手法では反例の長さに関して顕著に優れた結果が得られた．実際，提案手法では生成された反例の長さは概ね 20 程度であった一方，EACOhg では 30 ないし 50 程度の長さであった．これは，本手法におけるゴールフェロモンの拡散方式によって不必要な領域へのフェロモンの散布を抑制し，ゴールフェロモンをより効果的に蟻を最終状態へ導くことができたことが理由である．また，蟻がグラフ上を辿る距離が短くなることによって実行時間の短縮にも繋がった．

モデル C では反例の長さにおいて顕著な効果が得られなかった．これは，モデルが非常に大きい関係で，蟻が受理状態からのフェロモンを発見する頃には ACO によって経路の殆どが確立されていることが多く，経路の決定には受理状態からのフェロモンよりも ACO そのものの影響の方が強いからであると考えられる．しかし，このことは，散布するフェロモンの数を少なくしても先行研究の手法とほぼ同等の長さの経路を発見できたという点では優れた結果であると言える．

6. 結論

モデル検査のような，形式的検証の手法は，信頼性の高いソフトウェアの設計を構築することを目的とした，ソフトウェア工学の主な研究テーマの一つである．本論文では，自然界の振る舞いからヒントを得た最適化技術を，モデル検査における状態爆発問題の解決のために応用した手法を提案した．この問題に対処するために，ACO に基づいた先行研究である EACOhg を改良し，最終状態からのホップ数に基づいて不要なフェロモンの散布を抑制する働きを持った匂いフェロモンの仕組みを新たに導入した．匂いフェロモンは最終状態から最小の経路に向かって拡散する傾向があるので，反例の質を犠牲にすることなく，フェロモンの散布回数を削減することができる．本手法を実際のモデル検査器上で実行し幾つかの実験を行ったところ，本手法は従来の ACO に基づくの手法と比較して，反例の長さ，消費メモリ量やフェロモン拡散の性能，効率の面で改善されることが分かった．さらに，ホップ数の導入によって，フェロモンの拡散速度が遅い場合でも検証結果に良い効果を示すことがわかった．

今後の方針として，以下の問題解決に対する研究を検討している．

フェロモン拡散速度の調整

モデルの大きさに応じて匂いフェロモンの拡散速度を調整する技術を検討する必要がある．この研究によって，ACO に基づくのモデル検査の性能をさらに向上させることが期待できる．

モデル検査の最適化

Partial Order Reduction [15] のような，検証性能の向上のためのモデル検査の最適化手法が幾つか提案されている．そのため，本手法のような ACO に基づくの手法と最適化手法を組み合わせることによって，より効率的な反例の発見手法が得られる可能性がある．

本手法の活性への応用

先述したように，ソフトウェア設計の検証には，一般的に，安全性と活性の二つの性質に分類される．本論文で提案した手法は安全性だけに適用できる．一方，安全性とは異なり，活性では無限ループ (即ち，有向グラフ上の強連結成分) を有する反例が生じる．我々は，蟻にそのようなループを探索させることによって，活性にも対応できるよう手法の改良を行う予定である．

参考文献

[1] Alba, E. and Chicano, F.: ACOhg: Dealing with Huge Graphs, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pp. 10–17 (online), DOI: 10.1145/1276958.1276961 (2007).

[2] Alba, E. and Chicano, F.: Finding Safety Errors with ACO, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pp. 1066–1073 (online), DOI: 10.1145/1276958.1277171 (2007).

[3] Boussaïd, I., Lepagnot, J. and Siarry, P.: A Survey on Optimization Metaheuristics, *Inf. Sci.*, Vol. 237, pp. 82–117 (online), DOI: 10.1016/j.ins.2013.02.041 (2013).

[4] Chicano, F. and Alba, E.: Ant colony optimization with partial order reduction for discovering safety property violations in concurrent models, *Information Processing Letters*, Vol. 106, No. 6, pp. 221–231 (online), DOI: 10.1016/j.ipl.2007.11.015 (2008).

[5] Chicano, F. and Alba, E.: Finding liveness errors with ACO, *Proceedings of the IEEE Congress on Evolutionary Computation, CEC*, pp. 2997–3004 (online), DOI: 10.1109/CEC.2008.4631202 (2008).

[6] Clarke, E. M.: The Birth of Model Checking, *25 Years of Model Checking - History, Achievements, Perspectives* (Grumberg, O. and Veith, H., eds.), pp. 1–26 (online), DOI: 10.1007/978-3-540-69850-0_1 (2008).

[7] Clarke, E. M. and Emerson, E. A.: Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic, *Logic of Programs, Workshop*, pp. 52–71 (online), DOI: 10.1007/BFb0025774 (1982).

[8] Dorigo, M., Maniezzo, V. and Colorni, A.: The Ant System: Optimization by a colony of cooperating agents, *IEEE Transactions on SYSTEMS,Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 26, No. 1, pp. 29–41 (online), DOI: 10.1109/3477.484436 (1996).

[9] Dorigo, M. and Stützle, T.: *Ant Colony Optimization*, Bradford Company, MIT Press (2004).

[10] Edelkamp, S., Leue, S. and Lluch-Lafuente, A.: Directed explicit-state model checking in the validation of communication protocols, *International Journal on Software Tools for Technology Transfer*, Vol. 5, No. 2-3, pp. 247–267 (online), DOI: 10.1007/s10009-002-0104-3 (2004).

[11] Giannakopoulou, D. and Magee, J.: Fluent Model Checking for Event-based Systems, *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-11, pp. 257–266 (online), DOI: 10.1145/940071.940106 (2003).

[12] Holzmann, G. J.: The Model Checker SPIN, *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, pp. 279–295 (online), DOI: 10.1109/32.588521 (1997).

[13] Holzmann, G.: *The Spin Model Checker: Primer and Reference Manual*, Addison-Wesley (2004).

[14] Kumazawa, T., Yokoyama, C., Takimoto, M. and Kambayashi, Y.: Ant Colony Optimization Based Model Checking Extended by Smell-like Pheromone, *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, Vol. 16, No. 7 (online), DOI: 10.4108/eai.21-4-2016.151156 (2016).

[15] Lluch-Lafuente, A., Edelkamp, S. and Leue, S.: Partial Order Reduction in Directed Model Checking, *Proceedings of the 9th International SPIN Workshop on Model Checking of Software*, Springer-Verlag, pp. 112–127 (online), DOI: 10.1007/3-540-46017-9_10 (2002).

[16] Magee, J. and Kramer, J.: *Concurrency: State Models & Java Programs*, John Wiley & Sons, Inc. (1999).

[17] Manna, Z. and Pnueli, A.: *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag New York, Inc. (1992).

[18] Stützle, T. and Hoos, H. H.: MAX-MIN Ant System, *Future Generation Computer System*, Vol. 16, No. 9, pp. 889–914 (2000).