群知能でソフトウェアの不具合を発見する

株式会社 SRA 技術本部 先端技術研究室 研究員
東京理科大学 研究推進機構 総合研究院
デジタルトランスフォーメーション研究部門 客員研究員　熊澤（くまざわ）　努（つとむ）

はじめに

現在，ソフトウェアは私たちにとって最も身近な技術の一つとなりました．ソフトウェアは生活のいろいろな場面で使われています．自動車をはじめとする機器の制御から，スマートフォンの様々なアプリケーション，インターネットでのオンライン会議サービスまで，実現のためにはソフトウェア技術が欠かせません．その一方で，多様化の進んだソフトウェアは技術の複合体となり，その中身を理解するのが困難なくらい複雑化することも珍しくなくなりました．結果として，注意深く作られたソフトウェアであっても，人間が全てを把握することはできず，重大な不具合が見過ごされる恐れが出ています．大きな不具合が顕在化した場合には，社会に深刻な影響を与えることにもなりかねません．このような特徴を持つソフトウェアを開発するための主要な課題の一つは，ソフトウェアから不具合が取り除かれており，正しいものを作ったと自信を持って言うための技術の確立です．このことは，品質保証をどのように実現するか，と言い換えることができます．品質保証の代表的な方法は，作成したソフトウェアを計算機で実際に動かして正しさを確認するテスティングやシミュレーションです．本稿では，ソフトウェアを実行することなく品質を確認する方法を考え，こうした品質保証技術の一つであるモデル検査[1]を取り上げます．モデル検査は，ソフトウェアが仕様に従って正しく作られていることを自動的に検査する形式的検証と呼ばれる技術です．ここでいう検査や検証は，数学的にあるいは機械的な手続きで正しさを判定することを意味します．モデル検査は，それ自体がソフトウェアとして実現されることを目指し，検査アルゴリズムの開発が進められてきました．現在では，モデル検査を機能として持つソフトウェア・アプリケーションが開発されており，特に，高い信頼性が要求される産業で活用されています．ソフトウェアは，工業製品と同様に，顧客の要望や何を作るか決める上流工程から，開発工程を経て完成品の運用，保守に至る下流行程まで，複雑な工程を通じて開発が進められます．仕様の確認であるモデル検査は，工夫次第でプログラミングを始める前の上流工程でも使うことができます．開発したプログラムを動かすテスティングやシミュレーションは，下流行程で特に強みを発揮します．これらの技術を適切に組み合わせることで，開発工程のいろいろな場面で，ソフトウェアの品質を高めていくことができます．

モデル検査の研究では，自動化と結果の分かりやすさが鍵になります．そのため，これまでの研究で力を入れて取り組まれてきた課題は，主に実行性能の向上と，検査の結果得られる不具合診断情報の簡潔さを高める検査アルゴリズムの開発です．本稿では，筆者らが取り組んできた群知能あるいはメタヒューリスティクスといわれる問題解決アプローチを用いた検査技法を紹介します．

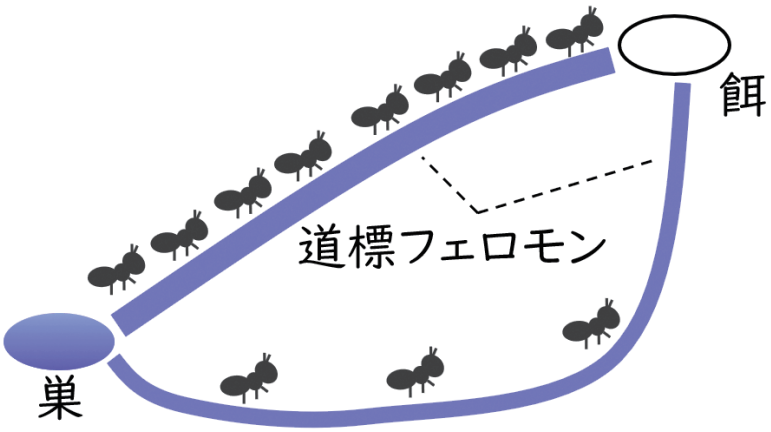
群知能とアリコロニー最適化法

群知能は，生物の集団的な行動や物理現象が生み出す創発的な性質を活用して，現実世界の多様な問題を解決する技法やアルゴリズムのことです．生物種の進化に着目した遺伝的アルゴリズム，鳥類や魚類が群れで移動する集団行動にヒントを得た粒子群最適化法，音楽の演奏過程を元にしたハーモニー探索など，非常にたくさんの問題解決法が研究されています．

ここでは，代表的な群知能であるアリコロニー最適化法[1]を例に説明します．アリコロニー最適化法は，真社会性生物であるアリ，特に働きアリの群れによる採餌行動の際に生じる行列の発生メカニズムに着想を得た問題解決法です．問題空間を大地に見立てて，働きアリの出発点である巣と，目標到達点である餌を結ぶ最短の道のりを探索することが目標です．各アリは順に巣を出発して，勝手に餌を探します．アリ同士がうまく行列を作るために，道標フェロモンを使った個体間のコミュニケーションを利用します．アリは，自身の通った道に道標フェロモンを出します．後続のアリはフェロモンの濃い道を優先して選択します．そのため，この道のりのフェロモンはますます濃くなり，

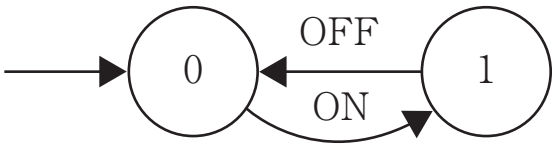
より多くのアリを集めるようになります．フェロモンの濃さやアリの移動の仕方などをうまく設定したアルゴリズムを設計すると，最短の道にアリを集めることができるようになります【図1】．面白いことに，注意深く設計したアルゴリズムは短い道をアリが発見するまでに多くの時間を必要としないことが，これまでの研究で知られています．

応用の際には，解きたい問題をうまく上の枠組みに当てはめて，計算機で実行するための具体的なアルゴリズムを構成します．例えば，より質の高い解を求める最適化問題を効率的に解くためには，アリの移動する道のりを解とし，道の長さを解の質の高さの指標となるようにアルゴリズムを作ることになります．



【図1】アリコロニー最適化法の考え方

モデル検査



【図2】モデルの例

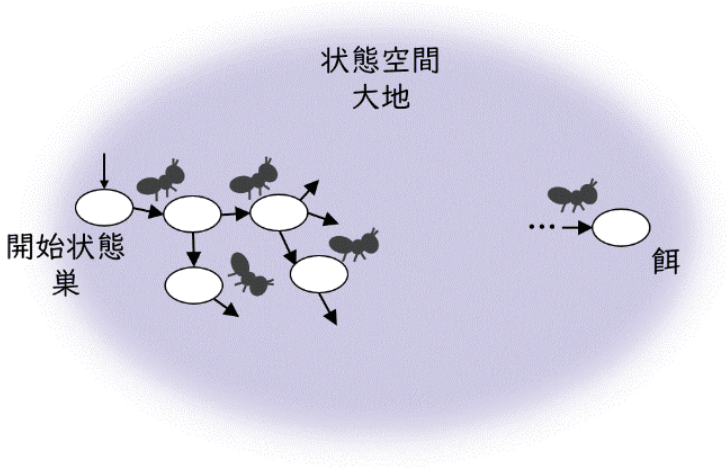
モデル検査[2]は，ソフトウェアの振る舞いが事前に定められた仕様を満たしているかどうかに注目した形式的検証技術です．ソフトウェアの振る舞いはモデルで記述します．モデルには【図2】のような状態遷移グラフがよく使われます．【図2】はソフトウェアで実現したスイッチのモデルで，0，1と付された丸は実行中の状態です．ONとOFFのラベルが付けられた矢印は，このラベルの動作により状態が矢印の向きに従って移り変わることを表します．例えば，スイッチをONにすることで，状態が0から1に変わり，OFFにすることによって状態は0に戻ります．状態0はスイッチを切った状態，状態1はスイッチが入った状態を表すと解釈できます．また，状態0はスイッチの実行開始を表す開始状態です．本稿では，開始状態を根元に状態がない矢印で示すことにします．状態遷移グラフは，その簡潔さと分かりやすさのおかげで実用的なことに加えて，理論的にも扱いやすいため，ソフトウェアに限らず，様々な分野で広く使われています．このようにモデル検査は，ソフトウェアそのものではなくモデルを扱う点が強みです．モデルを対象とすることで，完成したプログラムの品質確認だけでなく，ソフトウェアの機能を決定する段階，これからソフトウェアをどのように作るかを構想，設計している段階でも活用することができます．

検査する仕様のよく知られた例には，安全性と活性があります．安全性は「機能不全を起こす状態に決して到達しない」といったソフトウェアの継続的な実行を保証する性質です．一方，活性は「ユーザからの入力などに対していずれ適切に応答する」というソフトウェアが実現する機能の正常な実行を保証します．どちらの仕様にもソフトウェアが従っていなければ，不具合を引き起こす可能性があります．

本稿で考えるモデル検査は，状態空間と呼ばれる，モデルに仕様の情報を加えた特別な状態遷移グラフ上で実行します．その方法は，仕様の情報を参照しながら開始状態から状態空間を辿って，仕様違反のため不具合が発生するエラーを探すだけです．開始状態からエラー状態までの道のりを発見した場合には，ソフトウェアは仕様に従っていないと結論付けられます．発見した道のりは，ソフトウェアが仕様違反に至った実行系列を表す診断情報としてユーザに提示されます．この道のりが長いと，診断情報をユーザが読むのが大変なので，短いことが望ましいとされています．なお，一般にソフトウェアが取りうる状態の数は非常に多く，人間が目視でエラーを見つけるのは容易ではないので，エラーを探す処理は計算機で自動化します．また，自動化をする際には，検査が終わるまでの時間をできる限り短くする技術的な工夫が必要です．ユーザであるソフトウェア技術者は多くの仕事を遂行していて忙しいので，長時間待たせることは避けなければなりません．

アリコロニー最適化法のモデル検査への適用

アリコロニー最適化法をモデル検査に適用する基本的な考え方は，状態空間をアリが歩く大地，開始状態を巣，エラーを判定できたときの状態を餌とすることです【図3】．このように設定すると，可能な限り短い診断情報をアリが探索する問題を解くことで，本来の検査問題を解くことができます．



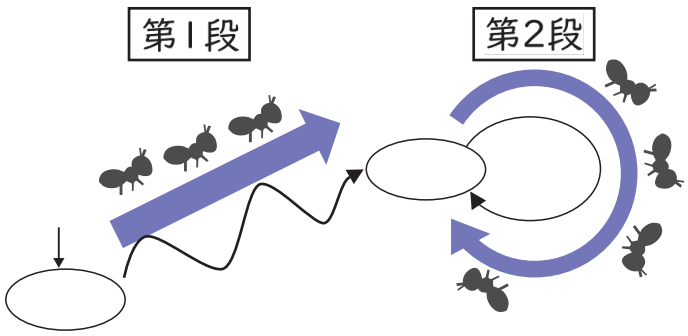
【図3】アリコロニー最適化法のモデル検査への適用

アリコロニー最適化法を用いた検証の難しさに，どこにエラーがあるか事前にはわからないことがあります．ここでは先に挙げた活性を考えてみましょう．活性のエラーが判定できるのは，アリが「入力に対する応答が将来に渡って返らない」ことを確認できたときです．

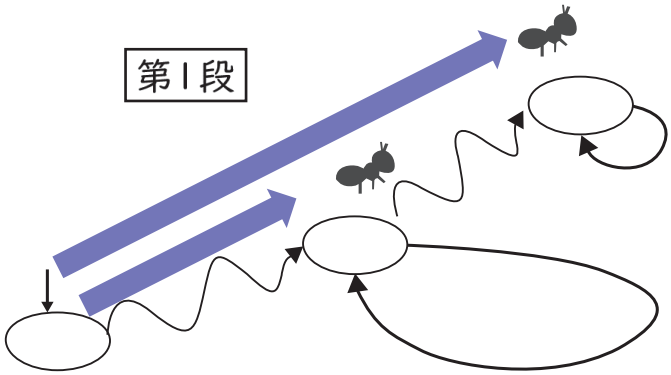
このようなエラーを発見するには，長い実行時間に渡ったソフトウェアの動きを分析する必要があります．ソフトウェアの実行を複雑にして分析を難しくする代表的な要因には，ソフトウェアの分岐処理と繰り返し処理があります．加えて，繰り返しの場合，実際に実行してみないとその終了条件が決まらず，動き続けてしまうこともあります．例えば，【図2】のスイッチの例を再び考えると，ONとOFFを何度でも繰り返す実行系列を表していることが分かります．モデルはソフトウェアを簡潔に記述することを目的としているため，その動きの全てを記述したものではなく，いつ停止するかの情報は記載していません．そのため，エラーを発見するには，ONとOFFを繰り返す道のりを網羅的に分析します．この図から分かるように，このような道のりは状態0と1を交互に移る「輪」を形成しています（正確には輪とその輪の付け根までの直線状の道のりです）．したがって，アリがエラーを含む輪をうまく検出すれば検査は完了です．残念ながら，古典的なアリコロニー最適化法は，アリは巣からエサまでの一直線の道のりしか移動しないため，輪を発見するのに向いていません．そこで文献[3]では，【図4】のように，アリコロニー最適化法を二段階実行して，一段階目に輪の付け根になる状態を，二段階目に輪を見つける方法を提案しています．この方法は，二段階目は輪の付け根に戻って来る道のりを発見するだけでよいので輪を直接扱う必要がなく，アリコロニー最適化法をそのまま使うことができます．

アリの探索戦略

文献[3]の方法は画期的ですが，課題もあります．エラーとなる輪とその輪への道のりは，【図5】のように複数あるかもしれません．アリは餌を見つけると探索をやめてしまうので，この図の場合には，一段階目のアリの探索は，左の開始状態から始

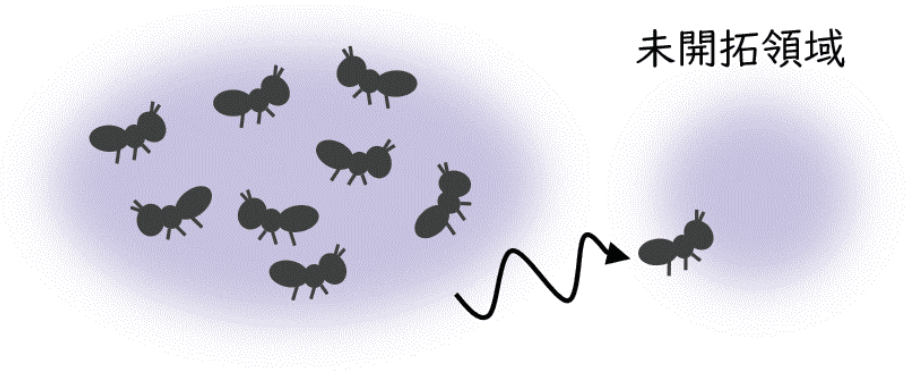


【図4】アリによる輪の探索



【図5】輪が複数ある場合

まって，中央の状態で一度停止します．そして，二段階目の輪の探索が行われることになります．しかし，右の状態にも輪があることから，輪を含めた短い道を発見するには，中央の状態も右の状態もどちらも分析しなくてはいけません．文献[3]によると，右の状態に到着するには，一段階目のアリコロニー最適化法をもう一度実行する必要があります．もし，右の状態まで一回目のアリコロニー最適化法で到達していれば，やり直しの必要がなくなり，時間の節約になるのではないでしょうか．



【図6】新方向の探索

以上の考えをもとに，筆者らはアリの新しい探索戦略として，「スキップ戦略」を提案しました[4]．スキップ戦略は，一段階目の探索において，各アリが探索中に輪の付け根にあたる状態を発見しても無視して探索を先に進める，という戦略です．アリの立場では，せっかく見つけた餌を我慢して，もっと良い餌が奥にあることを期待しながら餌の探索を継続する，という索餌方法です．餌を無視するかどうかは，無作為（ランダム）に決めることにします．目の前の餌を無視した方がよいかどうかは探索中にはわからないためです．無作為に決めることの利点は，探索するアリが十分に多ければ，従来の方法よりも多くの餌を見つけることが期待できることです．多くの餌に早く到達すれば，それだけ探索の効率が高まるでしょう．

以上のスキップ戦略を実際にソフトウェアとして実現するアルゴリズムで書き表す際には，少し工夫が必要です．筆者らは，輪の付け根に達するたびに先に進むかアリが決定するのではなく，予め決めておいた回数だけ状態を移動したところでアリは停止して，途中で遭遇した複数の付け根から一つ選ぶ，という方法を採用しています．他にもやり方は考えられると思いますが，この方法は，アルゴリズムを無暗に複雑にすることがなく計算の負荷がかからない，という利点があると考えています．

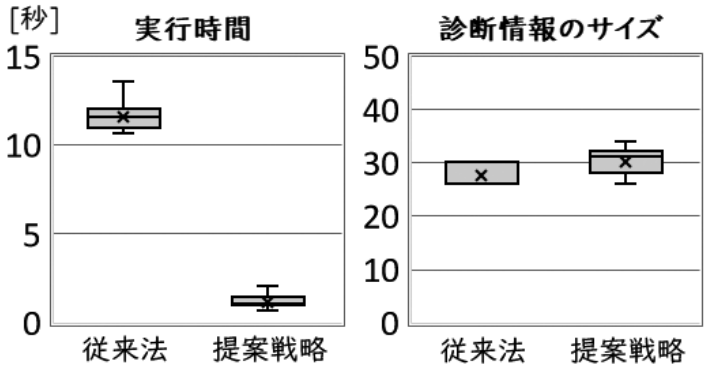
スキップ戦略に加えて，筆者らは，アリが発見する道のりが中途半端な長さにならないようにする探索戦略も新たに開発しました．短い道を発見できない停滞した状況では，周辺をある程度探索し尽くしたといえます．そこで，この戦略は，これまで探索されなかった状態に探索範囲を広げるようにアリの移動方策を決定します．現在発見されている最短の道より短い道が存在するかもしれないので，探索範囲を広げる価値があるというわけです．しかし，探索の進んでいない道には道標フェロモンはまかれていないか，まかれていたとしてもとても薄いでしょう．そのため，道標フェロモンを使ったアリ同士の情報交換は活用できません．そこで，アリが短い道のりを発見できなくなった場合には，後続のアリはあまり探索の進んでいない方向を敢えて選択することにします．餌を探し尽くした土地を捨てて，新しい餌を求めて未開の土地を開拓していくイメージです**【図6】**．この戦略は二段階目の探索で使用します．二段階目では広い空間にあるたった一つの状態を見つける必要があり，アリはできる限り広い範囲を歩き回る必要があると考えられるからです．また，一段階目で使用するスキップ戦略と組合せて使うことができるという利点もあります．

評価実験

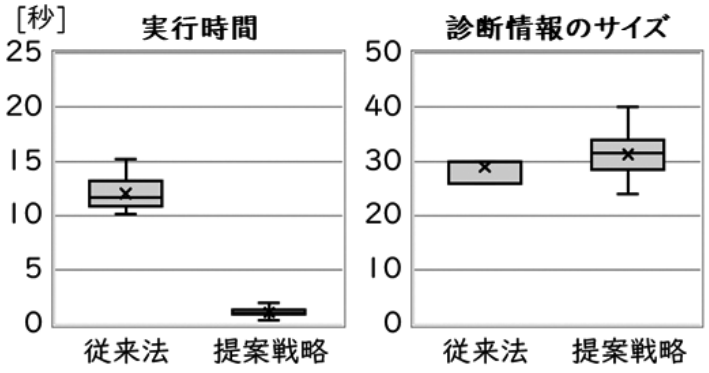
開発した二種類の探索戦略とアリコロニー最適化法を，ソフトウェアとして実装し，従来法と性能を比較する評価実験を行いました．並行システムについての検査問題を対象に行った評価実験の結果を**【図7】**に示します．並行システムは複数の処理が，相互にやり取りしながら同時並行に実行されるソフトウェアシステムで，挙動の分析や検査が難しい問題として知られています．図中の左の箱ひげ図は，計算の速さの評価のために，実装したツールを30回実行して，その実行時間を記録したものです，右の箱ひげ図は，検査結果の短さを評価することを目的として，30回の実行の各回で発見した最短の道のりの長さを示しています．どちらの評価指標も小さい値ほど良好な結果です．比較対象とした従来法は文献[3]の技法です．一方，提案戦略は，従来法に今回筆者らが開発した二つの探

索戦略をどちらも組み込んだ新技術です．なお，アリコロニー最適化法はアリの動きに無作為性を与えた乱択アルゴリズムの一種なので，実行結果にばらつきが発生します．実験で30回の実行結果を使用したのは，平均的な実行性能を測定するためです．

【図7】から分かるように提案戦略を組み込むと，従来法の実行速度に著しい改善が見られました．検査結果の短さは従来法と大きな差はありませんでしたが，実行速度と合わせて考えると，提案戦略の効果は非常に大きいと考えてよいと思います．同様の結果は，異なる検査問題を対象にした実験でも確認されました**【図8】**．また，本稿では詳細を省略しますが，実行性能の指標の一つである計算機のメモリ消費量を測定した結果においても，筆者らが提案した探索戦略によって改善が見られました．提案した探索戦略は，検査結果の長さを従来法と同程度に保ちながら，特に実行時性能の向上に高い効果を発揮すると考えられます．



【図7】 実験結果（1）



【図8】 実験結果（2）

おわりに

本稿では，ソフトウェアの品質保証のための技術であるモデル検査技術を述べ，群知能を使った自動検査アプローチを，筆者らの研究成果を紹介しつつ解説しました．本稿で述べたアリの探索戦略は，（1）適度に餌を無視して先に進む，（2）他のアリがあまり向かっていない領域に探索範囲を広げる，の二種類です．筆者らの実験結果から，本稿の戦略は，検査結果の簡潔さ悪化させることなく高速化を実現しており，有効な戦略であることが確認できました．これらの戦略は，自然界のアリを模倣することを意図している訳ではなく，モデル検査という特定の問題に特化した解法となっています．このように，群知能を実際の問題解決に活用するには，自然の振る舞いに拘り過ぎず，解きたい問題に適合した解決戦略をバランス良く組み合わせることが重要です．本稿で述べた二種類の戦略は，モデル検査におけるエラー探索の特徴に着目してうまく利用した方法であると考えています．

最後に，今後の展望を簡単に述べたいと思います．群知能に含まれるアルゴリズムは非常に多い中，本稿ではアリコロニー最適化法を使った方法を紹介しました．

しかしながら，モデル検査技術に最も効果的な群知能は何か，という問いについて，今のところ研究者間で納得の得られた結論はありません．現在，筆者らは粒子群最適化法を使った異なるアプローチを考案し，その特徴を明らかにすべく研究を行っています．実行性能などの点で，問題に応じた適切なアルゴリズムを使用できるようになれば，より多くの種類のソフトウェアへの適用が期待できます．群知能の特徴を生かしたアルゴリズムの確立を目指して研究を進めています．

[参考文献]

1) M. Dorigo and T. Stützle, Ant Colony Optimization, Bradford Company, MIT Press, 2004.
2) E. M. Clarke Jr., O. Grumberg, D. Kroening, D. Peled, and H. Veith, Model Checking, MIT Press, 2018.
3) F. Chicano and E. Alba, "Finding Liveness Errors with ACO," In IEEE Congress on Evolutionary Computation, pp. 2997–3004, 2008.
4) T. Kumazawa, M. Takimoto, and Y. Kambayashi, “Exploration Strategies for Model Checking with Ant Colony Optimization,” In 13th International Conference on Computational Collective Intelligence, LNAI 12876, pp. 264–276, 2021.