解説

探索に基づいたソフトウェア工学：SBSE[†] ー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ー

半田 久志*

1. はじめに

ソフトウェア工学の諸分野では、メモリ使用量、実行速度、ソースの可読性、工数、品質などの競合する目的バランスを取って、解を選択することが多い。この競合する環境下では、解の選択は非常に困難なものである。例えば、このような選択が必要な状況として以下のような項目を列挙できる：

1. あるプログラムに対して、すべての分岐を網羅できる最小のテスト集合、
2. システムの最適なアーキテクチャ、
3. 顧客満足度とソフトウェア開発コストのバランスをとった仕様、
4. ソフトウェア開発における資源配置、
5. ソフトウェアへの最適ナリファクタリング手順。

これらは、テスト、要求分析、ソフトウェア設計、ソフトウェア保守などの分野で研究されてきた。これまで、ソフトウェア工学における探索手法は深さ優先探索に基づくものが多く、探索手法に力点を置くよりも、近傍構造や評価指標の設計に力点を置いた研究が多かった。しかしながら、深さ優先探索は、いわゆる厳密解法であるため、内在する問題がNP困難な問題クラスであると現実的な時間で解を見出すことができない問題点がある。

ソフトウェア工学における諸問題に対して、メタヒューリスティックスを用いることにより、現実的な時間でより良好なバランスを取るための研究が、ヨーロッパを中心に展開されている。この研究領域に対して、「探索に基づいたソフトウェア工学(Search Based Software Engineering)」という用語が用いられている。

[†] Search Based Software Engineering: SBSE
ー Meta-Heuristics for Software Engineering ー
Hisashi HANDA

* 近畿大学理工学部情報学科
School of Science and Technology, Kinki University

本稿では、この探索に基づいたソフトウェア工学について概説する。次章では、探索に基づいたソフトウェア工学の概要を述べ、3章以降で、典型的な研究事例を紹介する。

2. 探索に基づいたソフトウェア工学

探索に基づいたソフトウェア工学は10年ほど前からEPSRCやEUで採択されたSEMINAL(Software Engineering using Metaheuristic INnovative ALgorithms), SEBASE(Software Engineering by Automated Search), EvoTest(Evolutionary Testing)といった大規模なプロジェクトを中心として進展しており、Symposium on Search-Based Software Engineeringという独自のシンポジウムが開催されている[1]。また、この分野の実用化も進められつつあることが報告されている。テストデータ選択で、例えば、Googleでのソフトウェア間の依存関係を考慮したテストデータ選択やIBMのミドルウェアに対して、進化型多目的最適化を用いた事例[2, 3]が報告されている。また、MicrosoftのPEXというホワイトボックスのテストデータ生成ツールに対して、(1+1)ESを使ったテストデータ生成のプラグインが提供されている[4]。

SBSE repositoryでは、SBSEに関する研究者やいくつかの統計データがまとめられている[5]。ここでは、そのいくつかを紹介する：図1は、年ごとの論文数を示している。SEMINALというプロジェクトが2001年に開始されているが、そのあたりから論文数が増加していることが分かる。2012年の途中でのデータであるので、2012年の論文数が少ないことはもちろんのこと、2011年分も未登録のデータがあることも考えられる。

図2はSBSEのソフトウェア工学における適用分野の内訳を示す[5]。ソフトウェアのテストに関する研究が半分を占めていることが分かる。また、ソフトウェア工学が対象としているほぼすべての分野について適用が試みられていることが分かる。ソフトウェア

図1 年ごとの論文数 (SBSE repository [5] より)

図3 国別の論文発表数 (SBSE repository[5] より)

図2 ソフトウェア工学における適用分野の内訳 (SBSE repository [5] より)

のテストが良く研究されている理由として、現在のソフトウェアの開発において、テストに係るコストが膨大であり、テストの省略化が希求されていること、そして、コード生成などに比べて、テストの場合、比較的問題空間が小さいことが考えられる(テストデータの選択などは組合せ最適化問題に帰着できる)。同様の図が、文献[1]にも掲載されている。2009年の段階では、ソフトウェアテストに関する研究はおおよそ60%を占めており、おおよそ10ポイントほど、ソフトウェアテストの占める割合が減少している。たった3年ほどの違いであるが、ソフトウェア工学の他分野への応用が近年進んでいることが分かる。図3は論文を国別にまとめたグラフである。25%を占める英国が一番論文を出版した国となっている。これは先述のSEBASEの巨額な研究資金によるプロジェクトが走っていたことによると考えられる。占有率1%以上の国がグラフ中に国名が記されているが、残念ながら日本はグラフ中に名前がない。

文献[1]では、SBSEの文献でメタヒューリスティックスの中で用いられていた手法も分類されている。2009年までの集計では、遺伝的アルゴリズム (Genetic

Algorithms)が全体の半分以上を占めており、それに続いて遺伝的プログラミング (Genetic Programming)、シミュレーテッドアニーリング (Simulated Annealing)、山登り法 (Hill Climbing) が続いている。これらは、探索空間の広さ・利用可能な計算資源・評価時間を勘案して、選ばれるものと考えられる。これらのメタヒューリスティックスを適用する際、解の表現方法・評価方法が判っている必要がある。つまり、手法によりアルゴリズムの設定や近傍の定義が異なり、具体的に適用する場合にはさらなる考察が必要となるが、解表現・評価が判明すればメタヒューリスティックスに総称される手法の使用に目処が立つ。

3. ソフトウェアのテスト

ソフトウェアのテストはプログラムのバグの洗い出しに用いられる。あるテストでバグが見つからなかったといっても、そのプログラムにバグが全く存在しないということを証明できるものではない。しかしながら、そのプログラムに対する信頼性が、幾分、向上する。本章で紹介する手法は、回帰テストを行うため、既存コード (ライブラリ) に対して、適切なテストデータセットを獲得・選択するためのものである。既存コードはすでにバグ取りが終わっていることが多いため、そのようなコードから適切なテストデータセットを獲得することは意味がないように思われるかも知れない。しかしながら、ソフトウェアシステムというのは常に機能追加・更新が起きるものである。つまり、何か機能追加するたびに、仮に既存コードには変更がなかったとしても、既存コードに影響を及ぼしていないかチェックする必要がある。テストはリリース前に不可欠であり、新規に追加するコードだけでなく、既存コードにもリリース前に行う必要があり、そのコストは膨大なものになっている。ソフトウェア開発費の半分以上がテストに費やされているという報告もある[9]。以下では、3種類のテストに関する研究を紹介する。

いずれも網羅率(カバレッジ)を指標とした評価関数を用いている。網羅率とは、文やブロック、分岐といったソースコードがテスト時に実行された割合を指す。単体テストのように、特定のクラスの動作についてのテストコードであれば、人手でコーディングした方が効率的である。しかしながら、システム全体の動作確認を行うためには、システムが大きくなればなるほど、テストを効率的に行うことができるテストデータ集合の発見という、最適化問題として帰着されるような問題クラスとなり、効果的なテストが難しくなる。

3.1 テストデータの優先順位付け

Liらは、多数のテストデータに優先順位をつけることにより、効率的にテストを行う方法を提案した[6]。すなわち、各テストデータの順列を最適化するものである。評価関数として、文網羅率、ブロック網羅率、分岐網羅率を用いており、それぞれについて、以下の関数を用いて評価を行った。これは、出来るだけ少ないテスト回数で網羅率が100となれば高い評価が得られるようになっている。

$$f = 1 - \frac{1}{nm} \sum_i^m B_i,$$

ここで、 n はテストデータの個数で、 m は網羅率を調べる文・ブロック・分岐の個数とする。 B_i は、 i 番目の文・ブロック・分岐が初めてテストされたときのテスト回数を表している。テストデータのテストを行う順番を順列で表現し、より少ない回数で幅広いブロックをテストできれば、評価が高くなることが判る。

表1にある6つのベンチマーク問題を導入し、アルゴリズムの比較を行った。表中のテストデータ数は前パラグラフの n に相当する。print_tokensの場合、長さが4130の順列を最適化することになる。研究用/実用の分類は、この研究用に構成された問題と、sedのように実用的に用いられているプログラムのソースを対象にしたものとを分けている。Spaceはヨーロッ

表1 Liらの論文で対象とした問題 [6]

問題の名前	テストデータ数	研究用/実用
print_tokens	4130	研究用
print_tokens2	4115	研究用
Schedule	2650	研究用
Schedule2	2710	研究用
Space	13585	実用
Sed	1293	実用

パ宇宙機関(European Space Agency)で用いられているプログラムを転用したものである。

アルゴリズムは深さ優先探索2種類、山登り法2種類、局所探索法を併用していない遺伝的アルゴリズムの5種類が用いられた。テストデータ数が少なくブロック数も少ないときに、遺伝的アルゴリズムがわずかながらであるが、一番良い性能を示し、規模が大きくなると2-optを用いた山登り法が最も良い性能を示した。テストデータ数が少ないといっても1000程度あり、TSPでいうと大きめな問題と考えられ、その中で遺伝的アルゴリズムは健闘したと言える結果であると考えられる。その一方で、問題規模がそれ以上になると試行回数が足らないようで、2-optを使って局所的に改善した方が良好な解が得られたと考えられる。

3.2 数値データを入力とするプログラムのテスト

プログラムの特定の分岐をテストするために、その分岐まで到達させるようなプログラムの初期値を獲得させることを考える[7, 8]。ここでは数値をプログラムに与えることを考えているので、実数値が入力となる。目標となる分岐で、 A_x という値をとれば良いと判っているときに、値が B_x であったとする。このとき $d(A_x, B_x) = |A_x - B_x|$ に基づいてプログラムの初期値を評価する。ここで、値 B_x は、分岐に至る計算の結果で、初期値として与えた値とは何らかの関係を持つはずではあるが、その関係は明白でないものとする。このような設定のとき、評価関数 $h(x)$ を次式で定める：

$$h(x) = \begin{cases} d_c(v_c, v_n) & \text{if COND not reached} \\ \frac{d(A_x, B_x) + K}{L + (d(A_x, B_x) + K)} & \text{if COND reached and} \\ & B_x \text{ not attained} \\ 0 & \text{Otherwise.} \end{cases}$$

ここで、 v_c と v_n は、プログラムを制御フローグラフで表現したときの、分岐があるノードと、分岐まで到達しなかったときに最もその分岐に近づいたノードを表している。 L, K は正の定数である。真ん中のif文は条件文まで処理が到達したものの、条件を満足しなかった時の評価となる。 $d_c(v_c, v_n)$ はそれらのノードの制御フローグラフ上の距離を表す。文献[8]では、このような問題について、実数値型分布推定アルゴリズムを用いた研究がなされている。

3.3 コンテナクラスにおけるテストデータ生成

文献[9]では、既存コード(コンテナクラス)のホワイトボックステストを例として、網羅率測定に基づいたテストデータ生成法が提案されている：コンテナク

ラスについて、 $\langle Object, Method, Input_list \rangle$ の三つ組みでテストデータを与えることを考える。入力を与えられ、実行すると実行された分岐の集合が得られる。このとき、分岐網羅率を $cov(S_i)$ 、列 S_i の長さを $len(S_i)$ とすると、分岐網羅率が高く、かつ、効率という観点から呼び出し列が短い方が良いテストであるものとする：

$$\begin{cases} cov(S_i) > cov(S_j) \text{ or} \\ cov(S_i) = cov(S_j) \text{ and } len(S_i) < len(S_j). \end{cases}$$

このような問題に対して、次のような適応度関数が用いられている。

$$f(S_i) = cov(S_i) + \frac{1}{1 + len(S_i)}$$

ここで、問題はどのようなテストデータ群を与えれば、高い網羅率で効率的なテストが行えるか、すなわち、高い適応度が得られるかという問題となる。問題空間、すなわち、取りうるテストデータ群の集合は、とてつもなく巨大である。しかしながら、従来はカバレッジの値を眺めながら手作業で入力列を考えたり、非効率なことを承知で列挙的にテストを行うしかなかった。

文献[9]では、この分岐網羅率を対象として、ランダム探索、山登り法、シミュレーテッドアニーリング、遺伝的アルゴリズム、メメティックアルゴリズムの比較を行っている。山登り法やメメティックアルゴリズムが網羅率で高い性能を示し、加えて、後者の方が、短い長さを持つテストデータ群が獲得されたことが報告されている。

4. ソフトウェア管理

ソフトウェア管理では、主にモジュール化やリファクタリングについて探索手法の利用が検討されている。本稿ではリファクタリング[10]を紹介する：この文献では、筆者らが実装したCode-Imp[11]上の探索エンジンとして、遺伝的アルゴリズム、シミュレーテッドアニーリング、局所探索法の比較を行っている。Code-Impでは、ソースコードを抽象構文木に変換して処理を行う。リファクタリングを行う操作として、以下の操作が定義されている：

- ・ Push Down/Pull Up Field
- ・ Push Down/Pull Up Method
- ・ Extract/Collapse Hierarchy
- ・ Increase/Decrease Field Security
- ・ Replace Inheritance with Delegation, and vice versa
- ・ Make Super class Abstract/Concrete

この列挙はまとめられたものであり、[/]を挟んだ相

反する語は、それぞれ別の操作となっている。たとえば、「Push Down Field」では、特定の箇所のソースを、サブクラスに移すものである。Extract Hierarchy」では、すべてのメソッドを継承した新たなサブクラスを生成するものである。適応度評価は、Quality Model for Object-Oriented Design (QMOOD) [10, 12]に基づいて行われた。

遺伝的アルゴリズムは、上記の操作の操作列を個体としていた。シミュレーテッドアニーリングや局所探索法は、近傍定義に上記の操作を用いていた。シミュレーテッドアニーリングや局所探索法の適用方法の方がより適切であったため、文献[10]では、シミュレーテッドアニーリングが最良の結果を示していた。

5. モデル検査

モデル検査とは、ハードウェアやソフトウェアの設計から導出されたモデルが仕様を満足するかを検査することである。主にデッドロックなどを形式的に検出することを目的としている。文献[13]では並行で稼働するコンポーネント間の通信システムのモデルについて検査を行うAnt Colony Optimizationを提案している。この文献では、活性(liveness)、すなわち、ある状態から設計者が定義した状態に到達可能であることを検証するためにACOを適用している。彼らのACOでは、巡回セールスマン問題(Travelling Salesman Problems: TSP)のためのACOと同様に、フェロモン τ に基づいて、確率的に次の遷移ノード(この場合ではノードは状態を示す)を決定している。このフェロモン τ は次式で更新される：

$$\tau \leftarrow + \frac{1}{f(a^{best})}, \quad \forall j \in a^{best}$$

これは、最良個体に含まれる状態のフェロモンを強化することになる。そして、適応度関数は次式で表現されている。

$$f(a^k) = \begin{cases} |\pi + a^k| & \text{if } a_*^k \in O \\ |\pi + a^k| + H(a_*^k) + p_p + p_c \frac{\lambda_{ant} - |a^k|}{\lambda_{ant} - 1} & \text{Otherwise.} \end{cases}$$

O は目的状態を表しており、上側の式は目標状態への到達を意味している。下側の式は未到達で、 H は既存のヒューリスティックで現在のパスがどれくらい望ましいかを表現するものである。 p_p と p_c はペナルティ項で、それぞれ、目的状態に達しているのに終了状態ではない時、または、ループを形成している時に、ペナルティを与えるものである。 π はこれまでに探索したパスを表している。 λ_{ant} は探索時の深さの最大値を表しており、一回の探索では λ_{ant} 個以上先にある

ノードを探索しないものとする。

換言すれば、彼らの手法は、網羅的な探索をする代わりに、評価手法は従来のままでACOの方法で確率的に探索をしていく手法であると言える。実験では、従来手法(深さ探索手法の改良)では見つけることが出来なかった解を見出したり、よりパスの短い解(長さが半分から10分の1)を見出していた。解けた問題では、メモリ使用量は1割程度増加した程度であったと報告があった。

6. ソフトウェア品質

ソフトウェア品質に分類される研究の中で、N-バージョンプログラムへの適用事例を紹介する。N-バージョンプログラムとはソフトウェアシステムのフォールトトレラント設計技術で、同一機能を有するN個のプログラムを構成し、N個の出力結果から最も多い出力を採用する方法論である。

Feldt は、遺伝的プログラミングにより、航空機の着陸の操作をシミュレータ上で獲得させた[14]。進化によって得られた80個のプログラムを解析した結果、80-バージョンプログラムを構成すれば、概ね、改善が見込まれるという結果になったものの、いくつかのシミュレーション環境では、ほとんどのプログラムが着陸に失敗してしまい、その結果、80-バージョンプログラムを構成しても改善が見込めない結果となった。

山地らは、N-バージョンプログラムにおいて、各プログラムのコストとサブタスクに対する性能評価が与えられたとした上で、与えられた予算内で、システムの信頼度が最大となるようなプログラムの選択法を提案した[15]。論文では、この選択問題を0-1非線形整数計画問題として定式化し、遺伝的アルゴリズムによる解法が試みられた。バイナリ表現とランダムキー表現の遺伝的アルゴリズムが試され、両者とも与えられた予算内でのシステムの高信頼化が実現できたものの、バイナリ表現ではコスト上限付近の探索が行われにくく、ランダムキー表現では、コスト上限付近の解が重点的に探索される結果となった。

7. むすび

本解説では、探索に基づいたソフトウェア工学として、テストデータ生成、モデル検査、リファクタリング、N-バージョンプログラムの例を紹介した。これ以外にも、ソフトウェア工学の他の分野、例えば、リリース時におけるバランスのとれた仕様の構成や、プログラム設計にも幅広く用いられている。いずれにしても、これまで網羅的に探索するしかない中で、効率的に探索しうる指標や部分解評価方法の研究が主で

あったソフトウェア工学に近似探索手法を導入することにより、従来法では解けなかった問題規模でも良好な結果を示している。最近、組合せ最適化問題で、メタヒューリスティックスが多用されているように、ソフトウェア工学分野においても、進化計算などの近似解法が重用される時代が来るかも知れない。また、組合せ最適化問題では、問題はgivenであったが、ソフトウェア工学で導入されている指標や部分解評価は、ヒューリスティックスに基づくものである。したがって、現状では、従来のソフトウェア工学研究に熟知した研究者と組むことが重要であると思われる。しかしながら、Bonnazaの将棋での例で示されているように、そのようなヒューリスティックさえも、機械学習手法や進化的方法により置き換えられる時が来るかも知れない。

謝辞

本解説の一部は若手研究(B)(23700267)の助成を受けたものである。

参考文献

- [1] M. Harman, S.A. Mansouri, and Y. Zhang: Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications, Technical Report of King's College London, TR-09-03 (2009)
- [2] Y. Shin, R. Nilsson, and M. Harman: Faster Fault Finding at Google Using Multi Objective Regression Test Optimisation, Proc. 8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (2011)
- [3] Y. Shin and U. Shmuel: Highly Scalable Multi-Objective Test Suite Minimisation Using Graphics Cards, Proc. 3rd International Symposium on Search Based Software Engineering, pp.219-236 (2011)
- [4] K. Lakhotia, N. Tillmann, M. Harman, and J. de Halleux: FloPSy - Search - Based Floating Point Constraint Solving for Symbolic Execution Proc. 22nd IFIP International Conference on Testing Software and Systems, pp.142-157 (2010)
- [5] http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/
- [6] Z. Li, M. Harman, and R.M. Hierons: Search Algorithms for Regression Test Case Prioritization, IEEE trans. on Software Engineering, 33 (4), pp.225-237 (2007)
- [7] J. Wegener, A. Baresel, and H. Sthamer: Evolutionary test environment for automatic structural testing, Information and Software Technology, 43 (14), pp.841-85 (2001)
- [8] R. Sagarna and J.A. Lozano: Dynamic Search Space Transformations for Software Test Data Generation, Computational Intelligence, 24 (1), pp.23-61 (2008)

- [9] A Arcuri, and X. Yao: Search based software testing of object-oriented containers, Information Sciences, 178 (15), pp.3075-3095 (2008)
- [10] M. O’Keeffe and M. Cinnide: Getting the most from search-based refactoring, Proc. 9th Annual Conference on Genetic and Evolutionary Computation GECCO’07, pp.1114-1120 (2007)
- [11] I.H. Moghadam and M. Cinnide: Code-Imp: a tool for automated search-based refactoring, Proc. 4th workshop on Refactoring tools, pp.41-44 (2011)
- [12] J. Bansiya and C. G. Davis: A hierarchical model for object-oriented design quality assessment, IEEE Trans. Software Eng., 28 (1), pp.4-17 (2002)
- [13] E. Alba and J.F. Chicano: Searching for liveness property violations in concurrent systems with ACO, Proc. 10th Annual Conference on Genetic and Evolutionary Computation GECCO’08, pp.172-1734 (2008)
- [14] R. Feldt: Generating diverse software versions with genetic programming: an experimental study, IEE Proc. Software, 145 (6), pp.228-236 (1998)
- [15] 山地秀美, 辻村泰寛, 山本久志: NVPシステム設計に対する遺伝的アルゴリズム適用の有効性の評価, 日本経営工学会論文誌, 57 (2), pp.112-119 (2006)
(201 年 10 月 31 日 受付)
- [問い合わせ先]
〒577-8502 大阪府東大阪市小若江3-4-1
近畿大学理工学部情報学科
半田 久志
TEL: 06-6721-2332
FAX: 06-6727-2024
E-mail: handa@info.kindai.ac.jp

 著 者 紹 介

はん だ ひさし
半田 久志 [非会員]

1998 年京都大学大学院工学研究科博士後期課程退学, 同年より岡山大学工学部助手, 2012 年より近畿大学理工学部准教授。その間, 英国パーミンガム大学訪問研究員(2004-2005), 同大学Honorary Research Fellow(2006-2011), 2009 GECCO Best Paper Award, 2009 年度人工知能学会研究会優秀賞を受賞。2010-2011 年 IEEE CIS 進化計算技術委員会委員, WCCI 2014 CEC Technical Co-Chair, GECCO 1999-2012, CEC 2005-2010, PPSN IX-XII 等のプログラム委員等を務める。進化計算・強化学習の研究に従事。博士(情報学)