マルチモデル推測に基づいたソフトウェア信頼性予測精度の向上に向けて

呉 敬馳, 土肥 正, 岡村博之
広島大学先進理工系科学
{ d220580, dohi, okamu}@hiroshima-u.ac.jp

要旨

本稿ではマルチモデル推測に基づいて，組合せソフトウェア信頼性モデルにおける候補モデルの重みを決定する問題について考察する．この問題は約 *30* 年間ソフトウェア信頼性研究において積み残されてきた課題であり，これまでに理論的に妥当な方法が確立されていなかった．ここでは，赤池ウェイトと呼ばれる統計量を用いて，情報理論的かつ頻度論的観点から，複数の候補モデルを平均化することでソフトウェア信頼性の予測精度の向上を試みる．実際のバグ検出データに基づいた数値例において，通常の赤池情報量基準 *(AIC)* を最小にする単一モデルを用いて予測するよりも，組合せソフトウェア信頼性モデルを適用する方が安定した予測性能を与えることが示される．

1. 緒言

ソフトウェアの定量的な品質評価の中でも, ソフトウェア信頼性は最も基本的かつ重要な属性であることはよく知られており, 定量的なソフトウェア信頼性評価指標としてソフトウェア信頼度を予測する問題がある．ここで, ソフトウェア信頼度とは, 既定の条件下である定められた期間中に, ソフトウェアバグに起因するシステム障害が発生しない確率として定義される．通常, ソフトウェア信頼性モデルと呼ばれる数理モデルに基づいて, ソフトウェアバグ検出データのようなテスト工程で観測される統計データからソフトウェア信頼度を予測する [10]．中でも, 非同次ポアソン過程 (Non-homogeneous Poisson Process; NHPP) によるソフトウェア信頼性モデルの記述は, その数学的取り扱いの簡便さや統計推論の容易さから有効であるとされており, 過去 50 年間に渡るソフトウェア信頼性研究の歴史の中でも際立って多くのモデルが提案されてきた [1, 2, 6, 7, 16, 17, 19, 28, 29]．十分な量のソフトウェアバグ検出データが得られているという条件の下で, これらのデータに適合するモデルを求める問題はさほど困難である訳ではないが, 過去に観測された少数標本データから予測性能を含む汎化能力の高いモデルを獲得することは必ずしも容易ではない．このような傾向は, NHPP のような数理モデルだけでなく, 深層学習やアンサンブル学習のような機械学習モデルを用いて予測を行ったとしても変わることはない．

ソフトウェアバグ検出数の予測精度を向上させる取組みのひとつに, 組合せソフトウェア信頼性モデルがある．NHPP モデルのような候補モデルを複数用いることで過去のバグ検出データへの適合性の情報を活用し, 将来検出されるであろうソフトウェアバグ数を推測するために用いられる．初期において, Lyu and Nikora [11–13] と Nikora et al. [15] は, 複数の候補モデルの線形結合によって予測モデルを構築する方法を提案しているが, 各候補モデルのランク（相対順位）を合理的に決定することが出来ないという課題があった．後に, Su and Huang [24] やその後の一連の研究 [21–23] において, 多層パーセプトロン型 3 層ニューラルネットワークにおける活性関数としてシグモイド関数を用いる代わりに, 異なる NHPP モデルの平均値関数を代入し, ニューロンの結合荷重を重み係数とみなすことで各候補モデルのランクを決める方法を提案している．残念ながら, 上記のようなヒューリスティックな方法で候補モデルのランクを矛盾なく決定することは出来ないばかりか, 得られたソフトウェア検出バグ数の予測値にも統計的な説得力は見受けられない．よって, 組合せソフトウェア信頼性モデルの解法は, 約 30 年の長きに渡り積み残されてきた課題であったことに注意すべきである．

本稿では, 近年, 統計的推測の分野で中心的な役割を演じるマルチモデル推測 [5] の考え方に基づいて，組合せソフトウェア信頼性モデルにおける候補モデルの重みを理論的に決定する問題について考察する．ここでは，赤池ウェイトと呼ばれる統計量（例えば, [8] を参照）を用いて，情報理論的かつ頻度論的観点から，複数の候補モデルを平均化する Model Averaging の考え方を用いて, ソフトウェア信頼性の予測精度の向上を試みる．実際のバグ検出データに基づいた数値例において，通常の赤池情報量基準 (AIC) を最小にする単一モデルを用いて予測するよりも，組合せソフトウェア信頼性モデルを適用する方が安定した予測性能を与えることが示される．

2. ソフトウェア信頼性モデル

システムテストにおいて, 時刻 $t\ (\geq 0)$ までに検出されるソフトウェアバグの累積件数を $N(t)$ とし, 平均値関数 $\Lambda(t;\boldsymbol{\theta}) = \int_0^t \lambda(x;\boldsymbol{\theta})dx$ の非同次ポアソン過程 (NHPP) に従うものと仮定する．すなわち, $N(t)$ の確率関数が以下のように与えられるものとする．

$$\Pr\{N(t) = n \mid N(0) = 0\} = \frac{\{\Lambda(t;\boldsymbol{\theta})\}^n e^{-\Lambda(t;\boldsymbol{\theta})}}{n!}. \quad (1)$$

式 (1) の $\Lambda(t;\boldsymbol{\theta}) = \mathrm{E}[N(t)]$ は NHPP の平均値関数, $\lambda(t;\boldsymbol{\theta})$ はソフトウェア強度関数と呼ばれ, 通常は $\Lambda(t;\boldsymbol{\theta}) = \omega F(t;\boldsymbol{\alpha})$, $\lambda(t;\boldsymbol{\theta}) = af(t;\boldsymbol{\alpha})$ のように仮定される． ここで, $\boldsymbol{\theta} \in (a,\boldsymbol{\alpha})$ は強度関数に含まれる非負値実数パラメータ, $a\ (>0)$ はテスト前にソフトウェア内に残存する総期待バグ数を表す実数パラメータ, $F(t;\boldsymbol{\alpha})$ と $f(t;\boldsymbol{\alpha}) = dF(t;\boldsymbol{\alpha})/dt$ は各ソフトウェアバグが検出されるまでの時間の累積分布関数 $(F(0;\boldsymbol{\alpha}) = 0, F(\infty;\boldsymbol{\alpha}) = 1)$ と確率密度関数である．

定量的なソフトウェア信頼性評価尺度として, ソフトウェア信頼度がある．ソフトウェア信頼度は, 時刻 t でシステムテストを終えた後, リリース後の任意の時間間隔 u でバグに起因するソフトウェア障害が発生しない確率として定義され, NHPP の仮定の下で以下のように求められる．

$$R(u \mid t) = e^{-\{\Lambda(t+u;\boldsymbol{\theta})-\Lambda(t;\boldsymbol{\theta})\}}. \quad (2)$$

ソフトウェア信頼性理論において, これまでに数多くの NHPP に基づいたソフトウェア信頼性モデルが提案されており、その主な違いはソフトウェアバグ検出時間の累積分布関数 $F(t;\boldsymbol{\alpha})$ の違いに他ならない．表 1 は従前までに知られている代表的な NHPP モデルを表しており, 典型的な 11 種類の連続形累積分布関数に対応していることが分かる．これらの NHPP モデルのパラメータ推定機能は, SRATS (Software Reliability Assessment Tool on Spreadsheet) [20] に実装されている．

表 1: 代表的な NHPP に基づいたソフトウェア信頼性モデル.

累積分布関数	$\Lambda(t;\boldsymbol{\theta}),\ \boldsymbol{\theta} \in (a,\boldsymbol{\alpha}) = (a,b,c)$
Exp [6]	$\Lambda(t;\boldsymbol{\theta}) = a(1-e^{-bt})$
Gamma [28, 29]	$\Lambda(t;\boldsymbol{\theta}) = a\int_0^t \frac{c^b s^{b-1}e^{-cs}}{\Gamma(b)}ds$
Pareto [1]	$\Lambda(t;\boldsymbol{\theta}) = a\left(1-\left(\frac{c}{t+c}\right)^b\right)$
TruncNormal [19]	$\Lambda(t;\boldsymbol{\theta}) = a\frac{F(t)-F(0)}{1-F(0)}$, $F(t;\boldsymbol{\alpha}) = \frac{1}{\sqrt{2\pi}b}\int_t^{-\infty} e^{-\frac{(s-c)^2}{2b^2}}ds$
LogNormal [2, 19]	$\Lambda(t;\boldsymbol{\theta}) = a\frac{1}{\sqrt{2\pi}b}\int_{\log(t)}^{-\infty} e^{-\frac{(s-c)^2}{2b^2}}ds$
TruncLogist [16]	$\Lambda(t;\boldsymbol{\theta}) = a\frac{F(t;\boldsymbol{\alpha})-F(0;\boldsymbol{\alpha})}{1-F(0;\boldsymbol{\alpha})}$, $F(t;\boldsymbol{\alpha}) = \frac{1}{1+e^{-\frac{t-c}{b}}}$
LogLogist [7]	$\Lambda(t;\boldsymbol{\theta}) = a\frac{1}{1+e^{-\frac{\log(t)-c}{b}}}$
TruncEVMax [17]	$\Lambda(t;\boldsymbol{\theta}) = a\frac{F(t;\boldsymbol{\alpha})-F(0;\boldsymbol{\alpha})}{1-F(0;\boldsymbol{\alpha})}$, $F(t;\boldsymbol{\alpha}) = e^{-e^{-\frac{t-c}{b}}}$
LogEVMax [17]	$\Lambda(t;\boldsymbol{\theta}) = ae^{-e^{-\frac{\log(t)-c}{b}}}$
TruncEVMin [17]	$\Lambda(t;\boldsymbol{\theta}) = a\frac{F(t;\boldsymbol{\alpha})-F(0;\boldsymbol{\alpha})}{1-F(0;\boldsymbol{\alpha})}$, $F(t;\boldsymbol{\alpha}) = e^{-e^{-\frac{t-c}{b}}}$
LogEVMin [17]	$\Lambda(t;\boldsymbol{\theta}) = ae^{-e^{-\frac{\log(t)-c}{b}}}$

一旦, 平均値関数が仮定されると, ソフトウェアバグ検出データから未知パラメータ $\boldsymbol{\theta} \in (a,\boldsymbol{\alpha})$ を統計的に推定する必要がある． 各テスト時刻 $t_i\ (i = 1,2,\ldots,n)$ においてそれぞれ x_i 個の累積バグ数が観測されているとき, 最終観測時刻 $t = t_n$ までの累積検出バグ数を表すグループデータ（不完全データ）は $\boldsymbol{x} = \{(t_1,x_1),(t_2,x_2),\ldots,(t_n,x_n)\}$ のように与えられる．このソフトウェアバグ検出データに対する対数尤度

関数は, 式 (1) より

$$\ln\mathcal{L}(\boldsymbol{\theta};\boldsymbol{x}) = \sum_{i=1}^{n}\{(x_i - x_{i-1})\ln\left[\Lambda\left(t_i;\boldsymbol{\theta}\right) - \Lambda\left(t_{i-1};\boldsymbol{\theta}\right)\right] - \ln\left[(x_i - x_{i-1})!\right]\} - \Lambda(t_n;\boldsymbol{\theta}) \quad (3)$$

のように求めることが出来る．最尤推定法とは, 式 (3) の対数尤度関数を最大にするパラメータ（最尤推定値）

$$\tilde{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \ln\mathcal{L}(\boldsymbol{\theta};\boldsymbol{x}) \quad (4)$$

を求める問題に帰着され, 何らかの最適化アルゴリズムを用いてテスト期間中に観測されたバグ検出データからモデルパラメータを推定する．

しかしながら, 仮定された平均値関数がソフトウェアバグ検出過程を支配する真のモデルであるとは限らないため, 複数の候補モデルから観測データに最も適合したモデル（平均値関数もしくはバク検出時間累積分布関数）を選択する必要がある．最も基本的なモデル選択法として赤池情報量基準 (AIC) [3] がある．AIC は

$$\mathrm{AIC} = -2\ln\mathcal{L}(\tilde{\boldsymbol{\theta}};\boldsymbol{x}) + 2\phi \quad (5)$$

のように定義される．ここで, $\ln\mathcal{L}(\tilde{\boldsymbol{\theta}};\boldsymbol{x})$ は最大対数尤度であり, ϕ は任意パラメータ数を表すモデル次元である．AIC が小さいモデルが最も観測データに適合したモデルとみなされる．しかしながら, 観測時間経過後の将来における時間間隔 u において, AIC を最小にするソフトウェア信頼性モデルが常に予測性能が高いとは限らない．

ソフトウェアバグ検出データ $\boldsymbol{x}$ を観測した後に, 将来の時刻 t_{n+k} $(k = 1, 2, \ldots)$ において累積バグ数 $x_{n+1}, x_{n+2}, \ldots, x_{n+k}$ を予測することを考える． 予測時刻を挟んだ過去と将来における $N(t)$ の確率法則が同一である保証はないが，通常，頻度論的推論では簡便のため, 最尤推定値 $\tilde{\boldsymbol{\theta}}$ を平均値関数 $\Lambda(t;\boldsymbol{\theta})$ に代入し, 時刻 t_{n+k} $(k = 1, 2, \ldots)$ における値を予測するプラグイン予測が用いられる．また，予測性能を測る評価尺度のひとつとして, 予測絶対誤差 (PMAE) がある．ソフトウェアバグ検出データ $\boldsymbol{x}$ を観測した後, 将来時刻 t_{n+k} $(k = 1, 2, \ldots)$ での累積バグ数 $x_{n+1}, x_{n+2}, \ldots, x_{n+k}$ が事後的に与えられたとき, PMAE は

$$\mathrm{PMAE} = \frac{\sum_{i=n+1}^{n+k}|\Lambda(t_i;\tilde{\boldsymbol{\theta}}) - x_i|}{k} \quad (6)$$

で定義され, k は予測期間長と呼ばれる．むろん, いくつかの候補モデルの中で PMAE を最小にするソフトウェア信頼性モデルを予測時点で知ることは不可能である．

3. 組合せソフトウェア信頼性モデル

先にも述べたように, AIC を最小にするソフトウェア信頼性モデルが常に予測期間中の PMAE を最小にする保障はないため, 予測性能に関するモデル選択基準の効用は限定的となる．また, システムテストを終えた別のソフトウェア製品のバグ検出データを用いて PMAE を最小にするようなソフトウェア信頼性モデルを同定したとしても, その候補モデルが他のソフトウェア製品の信頼性評価に適用出来る保障はなく, モデルの（適合性ではなく）予測性能を如何に高めるかがソフトウェア信頼性理論における未解決の難題である．表 1 で列挙した典型的なソフトウェア信頼性モデルにおいても, 指数形モデル [6], 遅延 S 字形モデル [28], 習熟 S 字形モデル [16] など典型的な（期待値の意味での）信頼度成長曲線のパターンが存在することは経験的に知られているが, 予測期間内にどの成長曲線のパターンが生じるかを事前に知ることは出来ない．そこで, 複数のソフトウェア信頼性モデルの候補モデルの加重平均を用いて予測を行う, 組合せソフトウェア信頼性モデルが提案されている．

Lyu and Nikora [11] は 3 つの代表的なソフトウェア信頼性モデルの加重平均を予測モデルに用いる組合せソフトウェア信頼性モデルを提案し, いくつかの典型的な信頼度成長曲線のパターンを考慮することで予測性能が向上する場合があることを示している．今, M $(= 1, 2, \ldots)$ 種類のモデルがソフトウェア信頼性モデルの候補として与えられ, j $(= 1, 2, \ldots, M)$ 番目の NHPP モデルの平均値関数とモデルパラメータをそれぞれ $\Lambda_j(t;\boldsymbol{\theta}_j)$ とおく．組合せソフトウェア信頼性モデルの発想は, 各候補モデル j に対する重み ω_j $(\sum_{j=1}^{M}\omega_j = 1, j = 1, 2, \ldots, M)$ が与えられたとき, 総期待累積バグ数を表わす平均値関数を

$$\Lambda(t;\boldsymbol{\theta}) = \sum_{j=1}^{M}\omega_j\Lambda_j(t;\boldsymbol{\theta}_j), \quad \boldsymbol{\theta} = \{\boldsymbol{\theta}_j; j = 1, 2, \ldots M\} \quad (7)$$

のように仮定することである．これにより, 異なる成長曲線の出現パターンの割合を重み係数によって表現することが可能となり, ω_j の値が大きいモデルは予測において高いランクが与えられることを意味する．

Abdel-Ghaly et al. [1] は予測精度を向上させるために, 複数の候補モデルに対する予測器を用意し, これらのモデルの線形結合をメタ予測器として用いること提唱しているが, 重み係数を適切に推定する方法については論じ

ていない．Lyu and Nikora [11] では, NHPP とは異なる確率過程（同次マルコフ連鎖）や異なる推論原理（ベイズ推定）が混在した $M = 7$ 種類のソフトウェア信頼性モデルを仮定し, 等価な重み係数 $\omega_j = 1/7$ $(j = 1, 2, \ldots, 7)$ を持つ組合せソフトウェア信頼性モデルを考えている．各モデルパラメータの推定法について論文中では具体的に言及されておらず，最小二乗法に基づいて各モデルパラメータ $\boldsymbol{\theta}_j$ $(j = 1, 2, \ldots, 7)$ を推定したものと推察される．さらに文献 [12, 15] では, 重み係数を経験的に決める方法，算術平均の代わりに平均値関数の中央値になるよう重み係数を調整する方法, prequential likelihood に基づいて動的に重み係数を決定する方法を提案し, これらの予測器を CASE ツールとして実装している．以上の一連の研究の概要は, 文献 [13] でまとめられている．

上述の既存研究の問題点は, ソフトウェア信頼性モデルの基本的な前提となる確率法則を全く無視して累積バグ数の平均値だけに着目している点である．ソフトウェア信頼度を推定する際にベースとなる検出バグ数のポアソン性を考慮していないため，合理的な統計的推論とみなすことは困難である．一方, 既存研究とは異なり, すべての候補モデルが NHPP に従う仮定の下で, ポアソン過程の加法性から, 式 (7) を平均値関数にもつ NHPP の最尤推定値を求めることが考えられるかもしれない．しかしながら, 式 (7) を式（3）に代入した対数尤度関数は高次元で, 非線形かつ多峰性をもつ複雑な関数となることから, この制約付き最適化問題を安定的に解くことは極めて困難である．さらに, 最尤推定値として求められた重み係数がモデルのランクを正しく表現できる正当な理由が見当たらない点にも注意すべきである．

このような組合せソフトウェア信頼性モデルの重み係数を決定する方法として, Su and Huang [24] は多層パーセプトロン型 3 層ニューラルネットワークの活性関数に シグモイド関数の代わりに平均値関数 $\Lambda_j(t; \boldsymbol{\theta}_j)$ を代入し, 各ニューロンの結合荷重を重み係数とみなすことで ω_j を動的に決定する方法を提案している．このアイデアはいくつかの研究者によって議論され, リカレント型ニューラルネットワークへの拡張 [23], 不完全デバッグモデルへの適用 [21], 粒子群最適化 (Partial Swarm Optimization) を利用した探索アルゴリズムの適用 [22] などが考えられてきた．しかしながら，これらの方法は, 一見して組合せソフトウェア信頼性モデルの重み係数を誤差逆伝搬アルゴリズムで推定しているように見えるが, 各候補モデルの重要度を示す重み係数の統計的推論をしている訳ではなく, しかも NHPP の確率法則とは全く無関係な方法であることに変わりはない．上術のような理由から, 組合せソフトウェア信頼性モデルにおける重み係数の決定法は, 未だに解決されてはおらず，かつそれが通常の単一モデルのプラグイン予測よりも有効であることすら検証されていない．次節では，モデル平均化アプローチ (Model Averaging Approach) と呼ばれるマルチモデル推測の中でも, モデル選択基準 AIC に特化した頻度論的方法に着目し，組合せソフトウェア信頼性モデルの重み係数を合理的に推定する方法を提案する．

4. モデル平均化アプローチ

モデル選択基準である AIC 自体がデータの関数であるため, AIC 自身も確率変数であることは明らかである．よって, モデル選択に起因する不確定性は常につきまとい, 最小となる AIC の値に近い他の候補モデルとの差を如何に解釈するかは重要な問題である．Akaike [4] は $-\mathrm{AIC}/2$ が期待対数尤度の漸近的不偏推定量であることから, これを指数関数に乗じた $\exp(-\mathrm{AIC}/2)$ は最尤推定法によって推定したモデルの尤度とみなすことが出来ることを指摘している．さらに, 複数のモデルの存在を前提にしたマルチモデル推測において, 最小 AIC 値と各候補モデルの AIC 値との差に基づくモデルの相対的な確からしさを尤度と関連付けて評価する方法を提案している．情報量基準 AIC に関するサーベイは文献 [8] を参照されたい．以下では, Burnham and Anderson [5] によるモデル平均化アプローチをソフトウェア信頼性モデルに適用することを考える．

M $(= 1, 2, \ldots)$ 種類の NHPP モデル $\Lambda_j(t; \boldsymbol{\theta}_j)$ $(j = 1, 2, \ldots, M)$ に対する AIC 値を AIC_j によって表し, 候補モデルの中で最小 AIC 値を $\mathrm{AIC}_{\min}$ とする．各候補モデルと最小 AIC モデルにおける相対的な比較は

$$\Delta\mathrm{AIC}_j = \mathrm{AIC}_j - \mathrm{AIC}_{\min} \tag{8}$$

によって表現されるため, $\exp(-\Delta\mathrm{AIC}_j/2)$ が候補モデル j の近似尤度を示すことから, AIC 値の差を標準化した

$$\omega_j = \frac{\exp(-\Delta\mathrm{AIC}_j/2)}{\sum_{l=1}^{M} \exp(-\Delta\mathrm{AIC}_l/2)} \tag{9}$$

は候補モデルの相対的な確からしさを表す指標となり, 赤池ウェイトと呼ばれる．ここで, $\sum_{j=1}^{M} \omega_j = 1$ である．

赤池ウェイトは候補モデルのランクを表す標準化された重要度としてみなすことが可能であり, 式 (7) にこれ

表 2: データセット.

Data Set	Total number of bugs	Testing length (day/week/month)	Source	Nature of software system
DS1	54	17	SYS2 [14]	Real time command and control system
DS2	38	14	SYS3 [14]	Real time command and control system
DS3	120	19	Release2 [27]	Tandem software system
DS4	61	12	Release3 [27]	Tandem software system
DS5	66	20	DS1 [18]	Embedded application for printer
DS6	58	33	DS2 [18]	Embedded application for printer
DS7	368	100	Github [30]	Retro video game emulation for macOS
DS8	80	59	Github [31]	A simple web-based tool for Spriting and Pixel art

表 3: テスト段階初期 (20%) における赤池ウェイトの推定値.

DS1	**Exp(0.19)** TruncEVMin(0.09) TruncLogist(0.09) TruncNormal(0.09) TruncEVMax(0.09) LogEVMin(0.08) LogLogist(0.08) Gamma(0.08) LogNormal(0.08) LogEVMax(0.08) Pareto(0.07)
DS2	**Gamma(0.11) TruncNormal(0.11) LogNormal(0.11) TruncLogist(0.11) LogLogist(0.11) TruncEVMax(0.11) LogEVMax(0.11) TruncEVMin(0.11) LogEVMin(0.11)** Exp(0.00) Pareto(0.00)
DS3	**Exp(0.18) Gamma(0.11) LogEVMin(0.11) LogLogist(0.11)** LogNormal(0.09) LogEVMax(0.07) Pareto(0.07) TruncEVMax(0.07) TruncLogist(0.06) TruncNormal(0.06) TruncEVMin(0.06)
DS4	**Exp(0.21)** Gamma(0.08) TruncNormal(0.08) LogNormal(0.08) TruncLogist(0.08) LogLogist(0.08) LogEVMax(0.08) TruncEVMin(0.08) LogEVMin(0.08) Pareto(0.08) TruncEVMax(0.08)
DS5	**LogNormal(0.15) LogLogist(0.15) TruncEVMax(0.15) Gamma(0.14) LogEVMax(0.14)** LogEVMin(0.09) TruncLogist(0.09) TruncNormal(0.08) TruncEVMin(0.02) Exp(0.00) Pareto(0.00)
DS6	**LogEVMax(0.13) LogNormal(0.13) LogLogist(0.12) Gamma(0.12) TruncEVMax(0.12)** LogEVMin(0.09) Exp(0.08) TruncNormal(0.07) TruncLogist(0.07) TruncEVMin(0.05) Pareto(0.03)
DS7	**TruncNormal(0.53) TruncLogist(0.34) LogEVMin(0.13)** TruncEVMax(0.00) LogLogist(0.00) TruncEVMin(0.00) Gamma(0.00) LogNormal(0.00) LogEVMax(0.00) Exp(0.00) Pareto(0.00)
DS8	**LogEVMax(0.17) Pareto(0.16) LogNormal(0.16) LogLogist(0.15) LogEVMin(0.14) Gamma(0.13)** Exp(0.04) TruncEVMax(0.01) TruncLogist(0.01) TruncEVMin(0.01) TruncNormal(0.01)

を代入した平均値関数をもつ NHPP は, ランク付けされた複数の候補モデルを融合して, 将来における推測を行う予測器として機能することが期待出来る. また, 観測されているバグ検出データから, ω_j の値をチェックするだけでどのバグ検出時間分布の形状が寄与しているかを把握することが可能となり, 指数形モデル [6], 遅延 S 字形モデル [28], 習熟 S 字形モデル [16] のような関数の形状だけに着目した曖昧な候補モデルの選択をする必要がない（表 1 の 11 種類のモデルはすべて指数形もしくは S 字形曲線を示すことは, 累積分布関数の形状から自明である）.

5. 数値実験

ここでは, 表 2 に示すような 8 種類のグループデータを用いて, 各データセットの 20%, 50%, 80%が観測された時点で残りのテスト期間（80%, 50%, 20%）で検出されるソフトウェアバグ数を予測することを考える. 候補モデルとして表 1 で与えられる $M = 11$ 種類の代表的な NHPP モデルを仮定する. まず, 各予測時点において AIC を最小にする最も過去データへの適合性が高いモデルをひとつだけ選択 (miminum AIC model) し, それを将来の予測に用いる. 次に, 事後的にすべてのデータが観測された状況で PMAE を最小にするモデル

表 4: テスト段階中期 (50%) における赤池ウェイトの推定値.

DS1	**TruncLogist(0.22) TruncEVMin(0.20) TruncNormal(0.16) TruncEVMax(0.12)** LogEVMin(0.07) LogLogist(0.07) Gamma(0.06) LogNormal(0.05) LogEVMax(0.03) Exp(0.00) Pareto(0.00)
DS2	**LogEVMax(0.33) LogNormal(0.19) LogLogist(0.13) Gamma(0.10)** LogEVMin(0.07) TruncEVMax(0.06) TruncLogist(0.03) TruncNormal(0.03) Exp(0.03) TruncEVMin(0.02) Pareto(0.01)
DS3	**Exp(0.23) TruncEVMin(0.10)** LogEVMin(0.09) TruncNormal(0.09) Gamma(0.09) TruncLogist(0.09) LogLogist(0.09) TruncEVMax(0.09) Pareto(0.08) LogNormal(0.03) LogEVMax(0.01)
DS4	**TruncEVMin(0.20) TruncLogist(0.20) TruncNormal(0.18) TruncEVMax(0.17)** LogEVMin(0.06) LogLogist(0.05) Gamma(0.05) Exp(0.04) LogNormal(0.02) LogEVMax(0.01) Pareto(0.01)
DS5	**LogEVMax(0.80)** LogNormal(0.09) LogLogist(0.09) Gamma(0.01) LogEVMin(0.00) TruncEVMax(0.00) Exp(0.00) TruncLogist(0.00) Pareto(0.00) TruncNormal(0.00) TruncEVMin(0.00)
DS6	**LogNormal(0.12) Gamma(0.12) LogEVMin(0.12) LogLogist(0.12) LogEVMax(0.12)** TruncEVMax(0.09) TruncNormal(0.09) TruncEVMin(0.09) TruncLogist(0.09) Exp(0.04) Pareto(0.01)
DS7	**LogLogist(1.00)** LogNormal(0.00) LogEVMax(0.00) Gamma(0.00) TruncEVMax(0.00) LogEVMin(0.00) TruncLogist(0.00) TruncNormal(0.00) TruncEVMin(0.00) Exp(0.00) Pareto(0.00)
DS8	**Gamma(0.31) LogEVMin(0.30) LogLogist(0.29)** LogNormal(0.07) LogEVMax(0.03) Exp(0.00) Pareto(0.00) TruncEVMax(0.00) TruncLogist(0.00) TruncEVMin(0.00) TruncNormal(0.00)

(minimum PMAE model) との比較を行う．通常, どの候補モデルが PMAE を最小にするかは予測時点では分からないので, minimum PMAE model の PMAE の値は常に予測性能の最良値 (PMAE の下限値）を与えることに注意されたい．これに対して, 11 種類のすべての候補モデルを仮定した上で赤池ウェイトを求め，組合せソフトウェア信頼性モデルによって予測値を求めたモデルを AIC weight と表記する．

表 3〜5 では，データセットの各予測段階において, 組合せソフトウェア信頼性モデルを適用した際の赤池ウェイトの推定値を求めた結果を示す．10% 以上の重み係数を示したモデルを太字で記載している．これらの結果より，組合せソフトウェア信頼性モデルを構成する代表的な候補モデルの種類と重み係数の値は各データセットにおいて大きく異なっており，必ずしも全ての候補モデルを予測に用いる必要はないことが分かる．すなわち，赤池ウェイトを算出することで候補モデルの相対順位を知ることが出来るだけでなく，赤池ウェイトが極めて小さい（ほぼゼロの値をとるような）予測に不要なモデルを自動的に排除することが可能となる．

次に，図 1 において，DS1 のテスト初期段階，中期段階，後期段階において累積バグ数を予測した際の振舞いを表わす．初期テスト段階では，事後的に最良の予測モデルである minimum PMAE model と比べて, minimum AIC model と Akaike weight の予測精度は極端に低く，特に AIC を最小にするモデルを選択しても将来検出されるであろうバグ数を精度よく予測することは容易ではないことがわかる．テスト中期段階では，minimum AIC model は累積バグ数を過小評価し，Akaike weight は逆に過大評価する傾向にあった．テスト後期段階においては，minimum AIC model と Akaike weight の予測結果はほぼ同様であることが読み取れる．

表 6 では, 各データセットにおいて, minimum PMAE model, minimum AIC model, Akaike weight を PMAE の観点から比較した結果を表わす．ここで括弧内は, 表 1 の候補モデルの中での最良モデルを表わしている．黄色で色付けした箇所は PMAE の観点から全く等価な予測性能を表わしている．まず，minimum PMAE model と minimum AIC model を比較すると，黄色で色付けされた箇所を除き minimum AIC model の PMAE の値は常に大きいが，PMAE の値で 1 程度の大きさしか違わないケースが 24 件中 9 件あった．これより，約半数程

表 5: テスト段階後期 (80%) における赤池ウェイトの推定値.

DS1	**TruncLogist(0.35) TruncNormal(0.24) LogEVMin(0.14) TruncEVMax(0.12)** LogLogist(0.07) Gamma(0.04) TruncEVMin(0.03) LogNormal(0.00) LogEVMax(0.00) Exp(0.00) Pareto(0.00)
DS2	**LogEVMax(0.27) LogNormal(0.15) Exp(0.14)** LogLogist(0.08) Pareto(0.06) Gamma(0.06) LogEVMin(0.05) TruncEVMax(0.05) TruncLogist(0.05) TruncNormal(0.05) TruncEVMin(0.05)
DS3	**TruncEVMin(0.26) TruncNormal(0.20) TruncLogist(0.18) TruncEVMax(0.11)** Exp(0.09) LogEVMin(0.04) Gamma(0.04) Pareto(0.03) LogLogist(0.03) LogNormal(0.01) LogEVMax(0.01)
DS4	**TruncEVMin(0.46) TruncLogist(0.23) TruncNormal(0.20)** TruncEVMax(0.05) LogEVMin(0.02) Gamma(0.01) Exp(0.01) LogLogist(0.01) LogNormal(0.00) Pareto(0.00) LogEVMax(0.00)
DS5	**LogEVMax(0.40) LogNormal(0.19)** Pareto(0.08) Exp(0.08) LogLogist(0.06) LogEVMin(0.05) Gamma(0.04) TruncEVMax(0.03) TruncLogist(0.03) TruncNormal(0.02) TruncEVMin(0.02)
DS6	**TruncEVMin(0.14) TruncLogist(0.12) LogEVMin(0.11) TruncNormal(0.11) LogLogist(0.11) Gamma(0.11) LogNormal(0.10) TruncEVMax(0.10)** LogEVMax(0.08) Exp(0.00) Pareto(0.00)
DS7	**LogEVMax(1.00)** LogNormal(0.00) LogLogist(0.00) Gamma(0.00) LogEVMin(0.00) Pareto(0.00) Exp(0.00) TruncLogist(0.00) TruncEVMax(0.00) TruncNormal(0.00) TruncEVMin(0.00)
DS8	**Gamma(0.33) LogEVMin(0.32) LogLogist(0.30)** LogNormal(0.03) LogEVMax(0.01) Exp(0.01) Pareto(0.00) TruncEVMax(0.00) TruncLogist(0.00) TruncEVMin(0.00) TruncNormal(0.00)

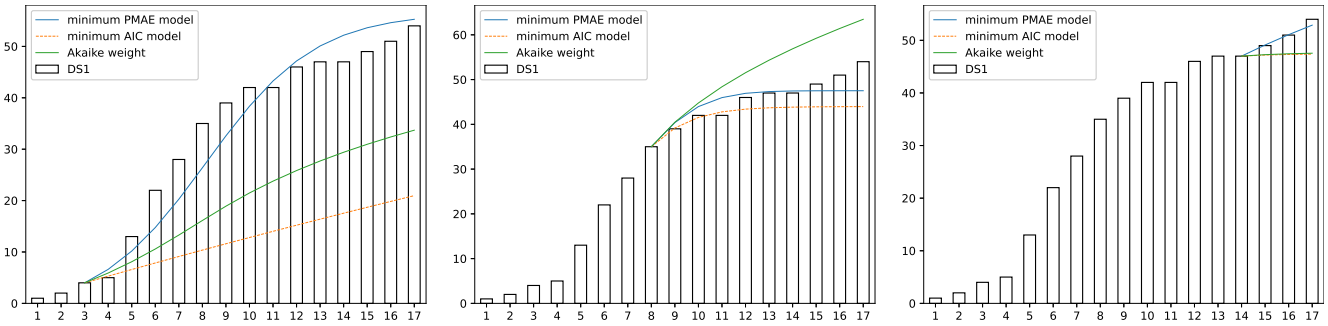
度の割合で minimum AIC model と minimum PMAE model の予測性能は大きく違わないと言える．一方で，minimum PMAE model と minimum AIC model 共に，最良の候補モデルはデータによって大きく異なっていることから，すべての候補モデルのモデルパラメータを推定した後に AIC に基づいたモデル選択を行う必要がある．一方，本稿で提案する組合せソフトウェア信頼性モデルを適用した場合，表 6 中の赤字で記載した 24 ケース中, 14 ケースにおいて minimum AIC model よりも高い予測性能を示すことがわかった．これにより，赤池ウェイトを用いて候補モデルの重みを決定する組合せソフトウェア信頼性モデルの予測性能の有効性が示される．

6. 結論

本稿ではマルチモデル推測に基づいて，組合せソフトウェア信頼性モデルにおける候補モデルの重みを決定する問題について考察した．特に, 組合せソフトウェア信頼性モデルにおける重み係数の決定に赤池ウェイトと呼ばれる統計量を活用し，複数の候補モデルを平均化することでソフトウェア信頼性を予測する方法を提案した．8 つの実プロジェクトで観測されたバグ検出データに基づいて，予測精度の比較を行い, 通常の赤池情報量基準 (AIC) を最小にする単一モデルを用いて予測するよりも，組合せソフトウェア信頼性モデルを適用する方が安定した予測性能を与えることを実証した．

ここで述べた赤池ウェイトに基づいた方法は, マルチモデル推測におけるモデル平均化アプローチのひとつであり, 修正情報量基準 AIC_c [25] やベイズ型情報量基準 BIC (Bayesian Information Criterion) [26] など他の情報量基準に基づいたアプローチを考えることも可能である．マルチモデル推測の汎用的な有効性を示すためには, モデルランクの決定法についてさらに検証する必要があると考えられる．

また, 候補モデルとして一体どれだけの数と種類のモデルを準備すれば推定性能が向上するかについて, 詳細に検討する必要がある．SRATS [20] で実装されているように, 最適化アルゴリズムを工夫することで複数の候補モデルに対する最尤推定を行うコストはかなり低い．よって, 多くの候補モデルに基づいた組合せソフトウェア信頼性モデルを列挙すればよいと考えるかもしれないが, 一方で, 過去データへの適合性が高い同質的なモデルだけを集めたとしても, 将来における不確実性を安定的



(a) Early testing phase. (b) Middle testing phase. (c) Late testing phase.

図 1: 期待累積バグ数の予測値の振舞い (DS1).

表 6: PMAE に基づいた予測性能の比較.

Early Software Testing Phase (20%)			
Data Set	minimum PMAE model	minimum AIC model	AIC weight
DS1	4.17 (TruncLogist)	23.88 (Exp)	15.97
DS2	5.40 (Pareto)	13.85 (Gamma)	13.69
DS3	17.98 (LogEVMin)	31.23 (Exp)	26.63
DS4	31.02 (Pareto)	32.30 (Exp)	33.45
DS5	14.25 (Pareto)	17.90 (LogNormal)	17.86
DS6	15.43 (Exp)	25.43 (LogEVMax)	26.87
DS7	54.07 (LogNormal)	82.76 (TruncNormal)	82.01
DS8	29.49 (Pareto)	29.70 (LogEVMax)	29.99
Middle Software Testing Phase (50%)			
Data Set	minimum PMAE model	minimum AIC model	AIC weight
DS1	2.28 (TruncNormal)	3.63 (TruncLogist)	7.06
DS2	5.60 (Pareto)	7.14 (LogEVMax)	7.68
DS3	1.50 (TruncEVMin)	20.39 (Exp)	16.83
DS4	5.70 (Pareto)	46.87 (TruncEVMin)	34.74
DS5	18.02 (Pareto)	18.55 (LogEVMax)	18.66
DS6	4.00 (Gamma)	4.53 (LogNormal)	4.38
DS7	22.53 (LogEVMax)	37.05 (LogLogist)	37.05
DS8	2.56 (Pareto)	5.58 (Gamma)	5.81
Last Software Testing Phase (80%)			
Data Set	minimum PMAE model	minimum AIC model	AIC weight
DS1	0.44 (LogEVMax)	4.03 (TruncLogist)	3.91
DS2	0.57 (LogEVMin)	0.79 (LogEVMax)	0.58
DS3	0.48 (TruncNormal)	1.87 (TruncEVMin)	1.33
DS4	0.21 (TruncNormal)	1.06 (TruncEVMin)	0.14
DS5	1.97 (LogEVMax)	1.97 (LogEVMax)	2.37
DS6	2.40 (TruncEVMin)	2.40 (TruncEVMin)	7.01
DS7	4.78 (LogEVMax)	4.78 (LogEVMax)	4.78
DS8	1.50 (Pareto)	1.83 (Gamma)	1.84

に捉えることは困難であるように思われる．そこで, 出来る限り多様な候補モデルを列挙した上でモデルクラスのクラスタリングを行い, 真に予測に必要な候補モデルを選定することが重要であると考えられる．例えば, 最近 Li et al. [9] は, 表 1 で示す寿命分布を表現する代表的なバグ検出時間累積分布関数よりも, Burr 分布とその周辺クラスに属する分布関数を用いてソフトウェアバグ予測を行う方が予測性能が高いことを実証している．

参考文献

[1] A. A. Abdel-Ghaly, P. Y. Chan and B. Littlewood (1986), Evaluation of competing software reliability predictions, *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 950–967.

[2] J. A. Achcar, D. K. Dey, M. Niverthi (1998), A Bayesian approach using nonhomogeneous Poisson processes for software reliability models, *Frontiers in Reliability*, A. P. Basu, S. K. Basu and S. Mukhopadyyay (eds.), pp. 1–18, World Scientific, Singapore.

[3] H. Akaike (1973), Information theory and an extension of the maximum likelihood principle, *Proceedings of The 2nd International Symposium on Information Theory*, B. N. Petrov and F. Caski (eds.), Akadimiai Kiado, pp. 267–281, Budapest.

[4] H. Akaike (1978), On the likelihood of a time series model, *The Statistician*, vol. 27, pp. 217–235.

[5] K. P. Burnham and D. R. Anderson (2002), *Model Selection and Multimodel Inference: A Practical Information-theoretical Approach*, Springer-Verlag, New York.

[6] A. L. Goel and K. Okumoto (1979), Time-dependent error-detection rate model for software reliability and other performance measures, *IEEE Transactions on Reliability*, vol. R-28, pp. 206–211.

[7] S. S. Gokhale and K. S. Trivedi (1998), Log-logistic software reliability growth model, *Proceedings of The 3rd IEEE International High-Assurance Systems Engineering Symposium (HASE-1998)*, pp. 34–41, IEEE CPS.

[8] 小西貞則 (2019), 情報量基準 AIC の統計科学に果たしてきた役割, 統計数理, vol. 67, pp. 193–214.

[9] S. Li, T. Dohi and H. Okamura (2022), Burr-type NHPP-based software reliability models and their applications with two type of fault count data, *Journal of Systems and Software*, vol. 191, p. 111367.

[10] M. R. Lyu (ed.) (1996), *Handbook of Software Reliability Engineering*, McGraw-Hill, New York.

[11] M. R. Lyu and A. P. Nikora (1991), A heuristic approach for software reliability prediction: The equally weighted linear combination model, *Proceedings of The 2nd IEEE International Symposium on Software Reliability Engineering (ISSRE-1991)*, pp. 172–181, IEEE CPS.

[12] M. R. Lyu and A. P. Nikora (1991), Software reliability measurements through combination models: approaches, results, and a CASE tool, *Proceedings of The 15th IEEE Annual International Computer Software and Applications Conference (COMPSAC-1991)*, pp. 577–584, IEEE CPS.

[13] M. R. Lyu and A. P. Nikora (1992), Applying reliability models more effectively, *IEEE Software*, vol. 9, pp. 43–42.

[14] J. D. Musa (1979), *Software Reliability Data*, Technical Report in Rome Air Development Center, New Jersey.

[15] A. P. Nikora, M. R. Lyu and T. M. Antczak (1992), A linear combination software reliability modeling tool with a graphically-oriented user interface, *Proceedings of The 2nd Symposium on Assessment of Quality Software Development Tools*, pp. 21–31, IEEE CPS.

[16] M. Ohba (1984), Inflection S-shaped software reliability growth model, *Stochastic Models in Reliability Theory*, S. Osaki and Y. Hatoyama (eds.), pp. 144–162, Springer, Berlin/Heidelberg.

[17] K. Ohishi, H. Okamura and T. Dohi (2009), Gompertz software reliability model: Estimation algorithm and empirical validation, *Journal of Systems and Software*, vol. 82, pp. 535–543.

[18] H. Okamura, Y. Etani, and T. Dohi (2011) Quantifying the effectiveness of testing efforts on software fault detection with a logit software reliability growth model, *Proceedings of Joint Conference of The 21st International Workshop on Software Measurement and The 6th International Conference on Software Process and Product Measurement*, pp. 62–68, IEEE CPS.

[19] H. Okamura, T. Dohi and S. Osaki (2013), Software reliability growth models with normal failure time distributions, *Reliability Engineering and System Safety*, vol. 116, pp. 135–141.

[20] H. Okamura and T. Dohi (2013), SRATS: software reliability assessment tool on spreadsheet (Experience Report), *Proceedings of The 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE 2013)*, pp. 100–107, IEEE CPS.

[21] P. Rani and G. S. Mahapatra (2018), Neural network for software reliability analysis of dynamically weighted NHPP growth models with imperfect debugging, *Software: Testing, Verification and Reliability*, vol. 28, p. e1663.

[22] P. Rani and G. S. Mahapatra (2019), A novel approach of NPSO on dynamic weighted NHPP model for software reliability analysis with additional fault introduction parameter, *Heliyon*, vol. 5, p. e02082.

[23] P. Roy, G. S. Mahapatra, P. Rani, S. K. Pandey and K. N. Dey (2014), Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction, *Applied Soft Computing*, vol. 22, pp. 629–637.

[24] Y.-S. Su and C.-Y. Huang (2007), Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models, *Journal of Systems and Software*, vol. 80, pp. 606–615.

[25] N. Sugiura (1978), Further analysis of the data by Akaike's information criterion and the finite corrections, *Communications in Statistics*, vol. A7, pp. 13–26.

[26] G. Schwarz (1978), Estimating the dimension of a model, *Annals of Statistics*, vol. 6, pp. 461–464.

[27] A. Wood (1996), Predicting software reliability, *IEEE Computer*, vol. 29, pp. 69–77.

[28] S. Yamada, M. Ohba and S. Osaki (1983), S-shaped reliability growth modeling for software error detection, *IEEE Transactions on Reliability*, vol. R-32, pp. 475–484.

[29] M. Zhao and M. Xie (1996), On maximum likelihood estimation for a general non-homogeneous Poisson process, *Scandinavian Journal of Statistics*, vol. 23, pp. 597–607.

[30] https://github.com/OpenEmu/OpenEmu, from Feb. 2012 to Sept. 2020.

[31] https://github.com/piskelapp/piskel/, from Sept. 2012 to Jul. 2018.