社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

選択的交叉を用いた遺伝的プログラミングに基づく自動修正手法

高　良多朗†　　肥後　芳樹†　　楠本　真二†

† 大阪大学大学院情報科学研究科
〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †{r-kou,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　ソフトウェア開発においては，信頼性を保証するためにデバッグ工程が必要不可欠である．しかし，ソフトウェア開発工程の大規模化および複雑化に伴ってデバッグ工程に要する労力もまた深刻化している．そのような背景からデバッグ工程の効率化手法，ひいては自動化手法が注目されており，中でも遺伝的プログラミングを用いて修正プログラムを自動的に生成する手法が高い評価を得ている．この手法は大きく分けて選択，変異，交叉の 3 つの処理から成り立つ．そこで，本研究では一度の処理で大きな変化をもたらせる交叉に着目した上で，交叉対象を無作為に決定するのではなく，定量的な基準を用いて決定することで効果的に交叉を行えると考えた．本稿では，個体間の比較に基づいた選択的交叉手法について提案する．

キーワード　ソフトウェア保守, ソースコード解析, 遺伝的プログラミング, 自動プログラム修正

An Automatic Repair Method Based on Genetic Programming Using Selective Crossover

Ryotaro KOU†, Yoshiki HIGO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1–5, Suita–Shi, Osaka, 565–0871 Japan
E-mail: †{r-kou,higo,kusumoto}@ist.osaka-u.ac.jp

1. まえがき

ソフトウェア開発および保守で必須とされる作業の 1 つに，信頼性や安全性を保障するためのデバッグ工程が存在する．デバッグ工程はソフトウェア中に生じたバグの発見から原因の特定，修正に加えてその成否の確認までを指す．しかし，ソフトウェアは開発を通して大規模化および複雑化する傾向にあり，同様にしてデバッグ工程に要する労力も増大化する．Baker の調査では，開発工程の半数以上がデバッグ工程に費やされたケースが確認されている [1]．上述の背景から，デバッグ工程の支援を目的とした様々な手法が提案されており，中でもバグの特定から修正，成否の確認までを自動で行う自動修正手法が注目を集めている．

近年では GenProg [2] を始めとした，遺伝的プログラミングに基づいて修正プログラムを生成する自動修正手法が高く評価されている．遺伝的プログラミングは生物の進化過程を模した探索アルゴリズムであり，個体の適合値を評価して優れたものを一定数取り出す “選択”，プログラムを変形して個体を生成する “変異”，複数の個体を組合わせて新しい個体を生成する “交叉” で構成される．GenProg およびその関連手法では修正プログラム候補を生物の個体と見立てて，テストケースの実行結果を用いて修正プログラム候補の評価や変形を行う．

GenProg の関連手法としては，ランダム探索によって修正プログラムを生成する RSRepair [3] や, あらかじめ用意した修正パターンを用いてプログラムを変形する PAR [4]，与えられた制約式を満たすようにプログラムを変形する SemFix [5] および DirectFix [6] などが存在するが，いずれも選択処理または変異処理に着目した手法である．

一方で本研究では交叉処理の特色である，一度の処理で大きな変化を引起こせる能力に着目した．本稿では，以下に示す 2 つの手法を合わせた選択的交叉を提案する．

（1） 交叉対象となる修正プログラム候補を類似度や適合値といった基準を用いて選択する手法

（2） 新たな交叉方式，すなわち交叉で入換える要素を選択する手法

また，手法評価のために GenProg の交叉処理に上述のアルゴリ

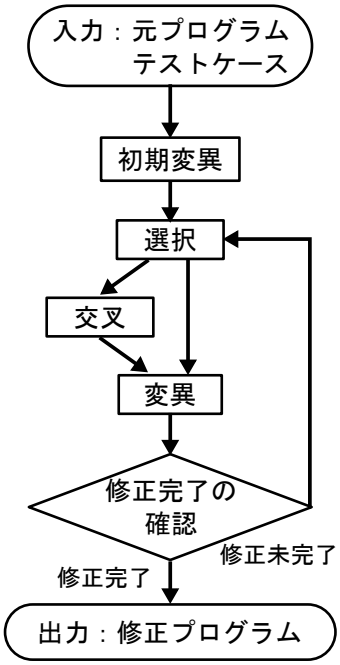


図 1　GenProg の流れ

ズムを実装したツール “Marriagent” を作成して GenProg との比較実験を行った．実験結果より，Marriagent は GenProg に比べて修正の成功率が高いことを示した．

2. 研 究 背 景

2.1 遺伝的プログラミング

遺伝的プログラミングとは生物の進化過程を模した探索アルゴリズムである [7]．このアルゴリズムはプログラムを生物の個体に見立てたもので，個体の変形や生存選択を繰り返して世代を進め，徐々に解に収束させることを目的としている．遺伝的プログラミングの主要部分は以下 3 つの処理から成り立つ．

選択　各個体の適合値，すなわちどれだけ優れた個体であるかを評価し，一定数の個体を取り出す．

変異　各個体ごとに変形を加え，次世代の個体を生成する．

交叉　2 つの個体を選び，各個体の要素を半分ずつ併せ持つ次世代の個体を生成する．

2.2 関 連 研 究

遺伝的プログラミングを用いた自動修正手法に GenProg [2] が存在する．GenProg は修正対象のプログラムとテストケースを入力とし，図 1 に示す流れで修正プログラム候補を生成する．生成した修正プログラム候補のいずれかが全テストケースを通過する場合，そのプログラムを修正プログラムとして出力する．GenProg における各処理を以下に示す．

選択　生成された各修正プログラム候補に対してテストケースを実行し，多くのテストケースを通過するプログラムから順に一定数を取り出す．

変異　各修正プログラム候補にさらなる変形を加え，新しい修正プログラム候補を生成する．プログラムの変形は追加，削除，置換からなる．

交叉　2 つの修正プログラム候補を選び，各々に加えられた変形を半分ずつ入換える．

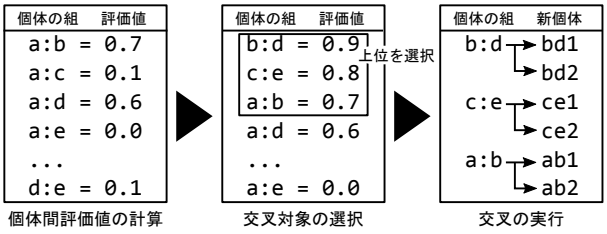


図 2　提案手法における交叉処理の流れ

ここで，GenProg における個体の構成要素はプログラムの文や字句ではなく，“x 行目を y 行目に追加” の様に，変異処理によって加えられた変形内容であることに留意されたい．つまり，元のプログラムは要素を持たない個体であり，交叉処理で交換する対象は変形内容の集合である．これらの変形内容は順不同であり，順番を入換えても実行結果には影響しない．

GenProg に関連する手法として，RSRepair [3] は遺伝的プログラミングの代わりにランダム探索を採用しており，世代更新を行わずに 1 箇所のみの変異を行った修正プログラム候補の生成を繰り返す．このため，1 箇所の変異で修正可能なバグに対しては非常に優れた性能を持つ．

PAR [4] はあらかじめ修正パターンを用意しておき，それらを用いてプログラムの変形を行う．定型的な修正を用いることで多くのソフトウェアに生じるバグの修正が容易であり，また修正プログラムの理解性の高さも評価されている．

SemFix [5] はテストケースから満たすべき制約式を計算し，これらを全て満たすようにプログラムを変形する．SemFix の最大の特徴として既存のプログラム文を再利用しないため，過去に記述されていない文を使用できる点が挙げられる．

DirectFix [6] は SemFix の発展手法であり，より簡潔で理解し易い修正が可能な点が評価されている．

2.3 交叉処理の特色

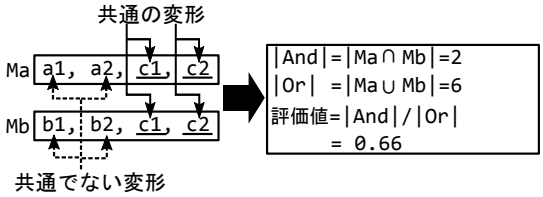
節 2.2 で挙げた既存手法はいずれも修正プログラム候補の評価またはプログラムの変形，言い換えれば選択処理や変異処理を改良した手法である．一方で，複数のプログラムを組合わせる交叉処理も他処理と同様に十分な改良の余地を残していると著者らは考えた．そこで，本研究では交叉処理の持つ特色に着目した．プログラムの変形を行う交叉処理と変異処理を比較すると，交叉処理は以下に示す特色を持つ．

- 一度の処理で複数の要素を入換えるため，大きな変化を起こしやすい
- 過去に加えられた不要な変形を取消すことが可能
- 同じプログラムの組を用いた場合でも，交叉パターンを変えれば異なるプログラムを生成できる

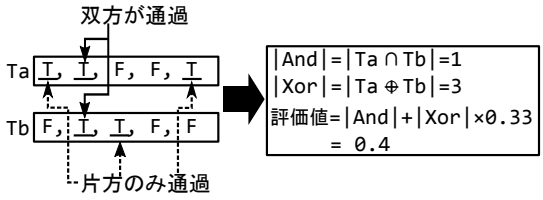
上述の特色を活かすように修正プログラム候補の組合わせを選んで要素を入換えることで，必要な変形のみを加えた修正プログラムが生成できると考えた．以上から，本研究では交叉対象の選択方法および交叉方式を改良した手法を提案する．

3. 提 案 手 法

提案手法は GenProg と同様に図 1 に示す流れで修正プログラム候補の生成を行う．ここで，選択処理および変異処理は



(a) 類似度での評価



(b) テストケースでの評価

図 3　個体間評価方法

GenProg と同一のアルゴリズムを使用しているため，本稿では説明を省略する．提案手法における交叉処理は図 2 に示す 3 つのステップで構成される．以下に各ステップの概要を示す．

個体間評価値の計算　最新世代の個体の全組合せについて，指定の評価方法を用いて交叉処理の優先度を計算する．

交叉対象の選択　交叉処理の優先度を用いて交叉対象を選択する．

交叉の実行　各交叉対象について指定の方法で交叉処理を実行する．

次に，各ステップについてその内容を詳細に述べる．

3.1　個体間評価値の計算

提案手法で使用する個体間評価方法を図 3 に示す．一度の処理で大きな変化が狙えるという交叉の特色を活かすためには，できるだけ異なる要素を持つ個体を選択することが望ましい．したがって，図 3(a) に示す方法では個体間での類似度を評価する．類似度の計算方法は式 1 に示す Jaccard 係数を用いる．提案手法では類似度が低いほど良い組合わせとみなすため，式 2 の値で評価する．この評価値は [0..1] の範囲をとる．

$$Jaccard\ 係数 = \frac{|M_a \cap M_b|}{|M_a \cup M_b|} \tag{1}$$

$$類似度評価値 = 1 - Jaccard\ 係数 = 1 - \frac{|M_a \cap M_b|}{|M_a \cup M_b|} \tag{2}$$

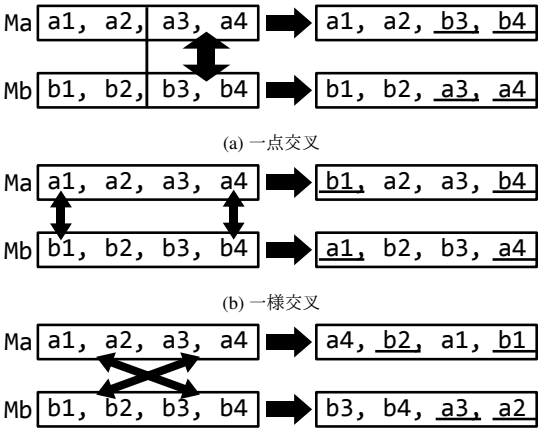
- M_a, M_b: プログラム a, b に加えられた変形の集合
- $A \cap B$: A と B に共通して加えられた変形の集合
- $A \cup B$: A と B の片方のみに加えられた変形の集合
- $|A|$: A の要素数

次に，交叉処理によって優れ個体を生成するためにはより優れた個体同士を選ぶことも重要である．そこで，図 3(b) に示す方法では各個体に対するテストケースの実行結果を比較し，より多くのテストケースを通過する個体同士の組合わせほど良いとする．この評価方法ではテストケース毎に双方通過 (And)，片方通過 (Xor)，双方失敗のいずれか判定を行い，式 (3) の値を評価する．この式の値は類似度と同様に [0..1] を取る．

$$テストケース評価値 = \frac{|T_a \cap T_b| + |T_a \bigoplus T_b| \times 1/3}{テストケース数} \tag{3}$$

- T_a, T_b: プログラム a, b に各テストケースを実行した結果の集合
- $A \cap B$: A と B の双方が通過したテストケースの集合
- $A \bigoplus B$: A と B の片方のみが通過したテストケースの集合
- $|A|$: A の要素数

類似度での評価法とテストケースでの評価法は互いに独立しているため，合わせて使用可能である．評価実験の章では，各評価法を単一で用いたパターンに加えて，2 つの評価方法を掛け合わせたパターンも実施している．



(a) 一点交叉

(b) 一様交叉

(c) ランダム交叉

図 4　交 叉 方 式

3.2　交叉対象の選択

本ステップでは，評価値が高い組合わせから順に交叉を行う．交叉回数は式 (4) で得られる．この値は GenProg における交叉回数の期待値である．また，図 2 における個体 b が該当するように，同じ個体が複数回選択される可能性もある．

$$交叉回数 = \frac{修正プログラム候補の数 \times 交叉確率}{2} \tag{4}$$

3.3　交叉の実行

提案手法で使用する交叉パターンを図 4 に示す．GenProg では図 4(a) に示す一点交叉を使用しているが，提案手法ではこれに加えて図 4(b) に示す一様交叉 および図 4(c) に示すランダム交叉を使用する．各交叉方式の定義を以下に示す．

一点交叉　要素の集合を中間で区切って後半の要素を入換える．

一様交叉　個体間で向かい合う要素をそれぞれ半分の確率で入換える．個体間で要素数が異なる場合，余った要素は半分の確率で交叉相手側に移すのみとする．

ランダム交叉　全要素を一度纏めて無作為に並び替え，要素数が均等になるように再分布する．

上述のパターンの内，一点交叉は一般的に性能が低いといわれ，特に隣接する要素が最適解の生成を妨げるヒッチハイキングに弱いとされる [8]．一方で一様交叉とランダム交叉はヒッチハイキングに耐性があるため優れていると考えられる．また，ランダム交叉を実行する際は，多くの要素の順番が変更されるが，要素の位置は実行結果に影響しないため手法の性能には関係しない．

4.　評 価 実 験

提案手法の有効性を示すため，GenProg を改変して提案手法

を実装したツール Marriagent を作成し，GenProg との比較実験を行った．評価項目は修正の成功率と，修正に要する時間である．

4.1 実験対象

実験対象には航空機制御システムである tcas [9] を選択し，3つのバージョンを用意した．各バージョンの詳細を表 1 に示す．バージョン 9 は代入文に欠陥を含んでおり，容易に修正可能である．バージョン 43 は制御文に欠陥を含んでおり，バージョン 9 に比べて修正は困難である．バージョン 44 は他バージョンと異なり二箇所に欠陥を持っており，一箇所の変形では修正不可能なため修正が非常に困難である．

4.2 実験方法

前節で示した実験対象に対して，GenProg と Marriagent を 50 回ずつ実行し，修正が 15 分以内に成功した回数を記録する．同時に，修正成功時の実行時間も記録する．

また，実行によっては交叉を行う前，修正プログラム候補の初期生成の時点で修正に成功する場合がある．この場合は既存手法と提案手法の比較が無意味なため，これらを除外した結果も合わせて記録する．

4.3 実験設定

GenProg および Marriagent を実行した際の設定を表 2 に示す．乱数は 1 から 50 までの範囲で実行毎に変更し，世代更新は 15 分の制限時間内ならば無制限に進行させる．また，表 2 に記載していない設定はデフォルトのものを指定した．加えて，Marriagent 固有の実行設定を表 3 に示す．各手法は 3.1 と 3.3 で示したものである．

4.4 実験結果

実験の結果を表 4 に示す．バージョン 44 の実験結果について，一度も修正に成功しなかった実行パターンが存在したため，該当する実行パターンの平均実行時間はハイフンを記載している．

実験の結果より，成功回数については，バージョン 9, バージョン 43 は Marriagent の全パターンが GenProg よりも良い結果を出しており，特にランダム交叉を用いた実行パターンは優れた結果を残している．一方，バージョン 44 では GenProg は一度も修正に成功していないが，Marriagent では修正に成功したパターンが存在する．

平均修正時間について，バージョン 9 では GenProg と Marriagent の間に大きな差は見られなかった．バージョン 43 では実行パターンによって大きく差が開いており，GenProg の半分程度のものもあれば 2 倍近いものも存在する．バージョン 44 は GenProg が修正に成功していないため比較不可能である．

また，どのバージョンの実験結果でも，最も成功数が多い実行パターンはテストケースのみを用いて交叉対象を選択し，ランダムに要素を入換えるパターンであった．

表 1　実験対象 tcas の詳細

バージョン	行数	欠陥の数	欠陥箇所
バージョン 9	173	1	代入文
バージョン 43	175	1	制御文
バージョン 44	175	2	代入文と制御文

表 2　実験の実行設定

項目	設定値	項目	設定値
成功テスト数	100	失敗テスト数	9
1 世代の個体数	10	交叉確率	0.5
乱数	実行毎に変更	世代更新	無制限

5. 考　　察

5.1 個体間評価方法

バージョン 9，バージョン 43 において，類似度のみを評価したパターンはどの交叉方式を用いても優れた結果を出していた．その理由として，当初の大きな変化を狙うという考えが効果を発揮したと考えられる．

テストケースのみを評価した場合，ランダム交叉のパターンは特に優れた結果を出しているが，他の交叉方式を用いたパターンではやや劣る結果を出している．これについては，テストケースを用いて交叉対象を選択した場合は優れた個体から順番に選ぶため，類似したものが選ばれやすい．したがって，要素の順番が変更されない一点交叉と一様交叉を用いると交叉の結果，元の個体とあまり変わらない個体が生成され易いためだと考えられる．一方でランダム交叉を用いた場合，要素の順番に依存しないため類似した組合わせでも十分に変化した個体を生成しやすいといえる．

また，個体類似度とテストケースを共に評価した方法は若干劣る結果となった．この原因としては，双方の評価法の相性が悪い可能性が考えられる．遺伝的プログラミングでは世代が進むにつれて優れた個体の割合が増加することが多いが，特定の個体を基にした個体が多く生成された場合，すなわち類似した個体が多数存在する場合は類似度による評価法が機能しづらくなる．特にテストケースによる評価法を用いると優秀な個体が複数回にわたって交叉対象として選ばれることが多い．したがって，優秀な個体を基にした類似個体が大量に生成されるために類似度による評価法が効果を発揮できなくなると考えられる．一方で，個体類似度とテストケースを共に評価したパターンの平均実行時間は短い傾向にあるため，類似個体が少ない初期状態では優れた性能を持っていると推測できる．

以上より，類似度による評価法とテストケースによる評価法はどちらも優れた点を持っているが，これらを組合わせて用いる場合，単純に組合わせるのではなく世代数や適合値に応じて

表 3　Marriagent の実行設定

パターン名	個体間評価方法	交叉方式
MarriagentSO	類似度	一点交叉
MarriagentSU		一様交叉
MarriagentSR		ランダム交叉
MarriagentTO	テストケース	一点交叉
MarriagentTU		一様交叉
MarriagentTR		ランダム交叉
MarriagentBO	双方使用	一点交叉
MarriagentBU		一様交叉
MarriagentBR		ランダム交叉

切り替える，もしくは重み付けを行うなどの工夫が必要になると考えられる．

5.2 交叉方式

どのバージョンまたは評価法を用いた場合でも，ランダム交叉を用いたパターンが優れた結果を出した．この理由としては，前節で触れたように，ランダム交叉は類似個体の組合わせでも大きな変化を起こしやすいことが大きな要因だと考えられる．

一方で，一点交叉と一様交叉はランダム交叉に比べて劣る結果となった．特に一様交叉はランダム交叉と同様にヒッチハイキングに強くランダム性が高いとされるが修正成功率に大きな差が生じている．一様交叉が有効でない理由としては，ヒッチハイキングの有無は手法の有効性にあまり影響しないためだと考えられる．言い換えれば，隣接した要素が修正を阻害するケースは少数であり，切り離す必要が無いといえる．ヒッチハイキングへの耐性はランダム交叉も同様だが，上述のようにランダム交叉は大きな変化を起こしやすいため，変化量という点で一様交叉に差をつけたと考えられる．

6. 妥当性の脅威

本稿では評価実験として GenProg と Marriagent の比較を行ったが，GenProg の実行設定は表 2 で設定した以外に変異の確率やテストケースの重み付けなど，多岐にわたる項目が存在する．このため，その他項目の設定を変更することで本実験とは大きく異なった結果が得られる可能性がある．特に提案手法は個体の組合わせに強く依存するため，1 世代辺りの個体数や世代数の上限など個体の生成に関する項目を変更した場合，大きく結果が変わる恐れがある．

また，今回実験対象としたソフトウェアである tcas の規模は高々百数十行であるため，数千行もしくは数万行の大規模なソフトウェアを実験対象に用いた場合は手法の有効性，特に類似度を用いた評価法の有効性が変化する可能性がある．また，テストケースの数も合計 109 個と多数使用しているためテストケースを用いた評価が有効であったが，高々数個のテストケースを用いた場合はプログラム毎のテストケース通過数に差が開きにくくなるため，ソフトウェア規模と同様に手法の有効性に影響すると考えられる．このような，ソフトウェアを変更した際に生じる提案手法の有効性の変化の調査は今後の課題となる．

7. あとがき

本研究では，GenProg の交叉処理を改良し，個体すなわち修正プログラム候補の中からから効果的な組合わせを選択して交叉を行う手法を提案した．実験の結果より，テストケース通過の是非をもって交叉対象を選択し，入換える要素をランダムに選択する方法を用いることで，高い確率で修正プログラムを生成できることが示された．今後の課題としては，個体間評価方法および交叉方式について新たなアルゴリズムの考案および既存のアルゴリズムとの組合わせの他，選択や変異に関する他の手法に提案手法を組合わせた場合の修正性能の評価，大規模ソフトウェアをはじめとした実験対象の拡大が挙げられる．

謝辞 本論文は，日本学術振興会科学研究費補助金基盤研究 (S)(課題番号：25220003) の支援を受けて行われた．

文　献

[1] J. Baker, “Experts battle £ 192bn loss to computer bugs,” Feb. 2012. Accessed:2015-12-07, http://www.cambridge-news.co.uk/Experts-battle192bn-loss-bugs/story-22514741-detail/story.html.

[2] C.L. Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, “Systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each,” In 34th International Conference on Software Engineering (ICSE) 2012., pp.3–13, June 2012.

[3] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang, “The strength of random search on automated program repair,” In 36th International Conference on Software Engineering (ICSE) 2014., pp.254–265, June 2014.

[4] D.Kim, J. Nam, J. Song, and S. Kim, “Automatic patch generation learned from human-written patches,” In 35th International Conference on Software Engineering (ICSE) 2013., pp.802–811, May 2013.

[5] H.D.T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, “Semfix: program repair via semantic analysis,” In 35th International Conference on Software Engineering (ICSE) 2013., pp.772–781, May 2013.

[6] S. Mechtaev, J. Yi, and A. Roychoudhury, “Directfix: Looking for simple program repairs,” In 37th International Conference on Software Engineering (ICSE) 2015., pp.448–458, May 2015.

[7] J.R. Koza, Genetic programming: on the programming of computers by means of natural selection, MIT Press, Dec. 1992.

[8] S. Forrest and M. Mitchell, “Relative building block fitness and the building block hypothesis,” Foundations of Genetic Algorithms 2, ed. by D. Whitley, pp.109–126, Morgan Kaufmann, Feb. 1993.

[9] H. Do, S. Elbaum, and G. Rothermel, “Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact,” Empirical Softw. Engg., vol.10, no.4, pp.405–435, oct. 2005.

表 4　実験結果

ツール名	成功数	交叉有	平均時間 (s)	交叉有	成功数	交叉有	平均時間	交叉有	成功数	交叉有	平均時間	交叉有
	バージョン **9**				バージョン **43**				バージョン **44**			
GenProg	18	13	190.17	251.54	7	4	147.22	234.92	0	0	-	
MarriagentSO	27	22	167.98	199.26	12	9	145.06	183.24	0	0	-	
MarriagentSU	27	22	207.21	248.06	8	5	90.69	128.60	0	0	-	
MarriagentSR	27	22	179.31	213.17	12	9	215.69	278.54	0	0	-	
MarriagentTO	25	20	180.67	218.07	8	5	183.28	276.81	1	1	844.01	844.01
MarriagentTU	22	17	169.98	210.99	8	5	181.82	274.30	0	0	-	
MarriagentTR	34	29	216.29	248.36	18	15	275.89	325.39	3	3	377.42	377.42
MarriagentBO	20	15	159.93	203.18	8	5	266.80	409.93	0	0	-	
MarriagentBU	22	17	172.54	214.43	8	5	99.34	141.16	0	0	-	
MarriagentBR	28	23	165.34	194.69	11	8	113.95	145.43	1	1	371.64	371.64