

大規模OSS開発における不具合修正時間の短縮を目的とした バグトライアージ手法

柏 祐太郎

大平 雅雄

阿萬 裕久

亀井 靖高

目次

- バグトリージとは
- ケーススタディ
- 考察
- まとめ

バグトリージとは

バグトリージ

- 報告された不具合に対して、開発者に適切に修正タスクを割り当てる技術
- 不具合の修正作業を滞らせる担当者の再割り当てを防ぐことができる

提案手法

- 不具合と開発者の組み合わせ問題をナップサック問題とみなす手法
- 不具合修正時間の効率化を目的としている
- 一部の開発者に集中して不具合を割り当てない

ナップサック問題

- 最大重量が決まっているナップサックにアイテムを入れる際、総価値が最大になるにはどのようにすればよいかという問題

ナップサック	→	修正能力の上限
アイテム	→	不具合
重み	→	修正に要する時間
利得	→	開発者の適正

用語定義

表 1 本論文で用いる用語一覧

Table 1 Terms used in this study.

用語	記号	意味
カテゴリ	k	不具合票の「コンポーネント」と「優先度」で分類したもの。
プリファレンス	P_{ij}	修正タスクをどの開発者に優先的に割り当てるべきかを示す尺度。 P_{ij} とは開発者 D_i がカテゴリ k の不具合修正タスク B_j に対するプリファレンスを示している。 $P_{ij} = \frac{\text{カテゴリ}k \text{ における開発者}D_i \text{ の修正数}}{\text{カテゴリ}k \text{ における修正タスクの総数}}$
コスト	C_{ij}	開発者 D_i が不具合修正タスク B_j に要する時間。過去に開発者 D_i がカテゴリ k の不具合修正タスクを完了するのに要した修正時間の中央値とした。
上限	L	タスクの集中を防ぐために設定する値
割当て可能時間	T_i	一定期間内に修正可能なタスク量（時間）。 T_i は開発者 D_i の担当可能時間を示す。 $T_i = \text{上限 } L - \sum_{j=1}^n C_{ij} * x_{ij}$

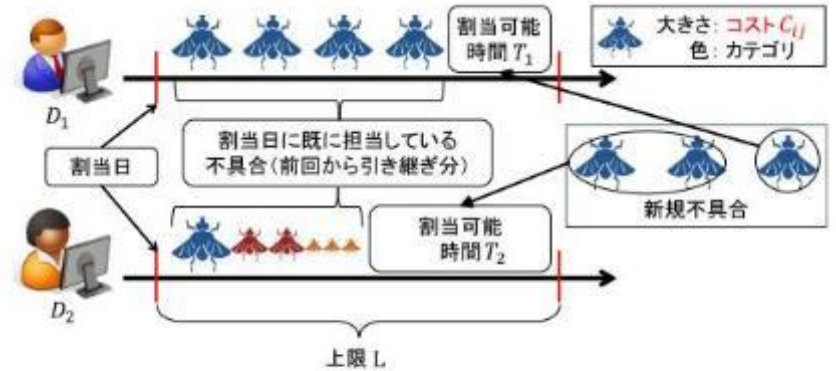


図 1 修正可能なタスク量（時間）の求め方

Fig. 1 The amount of available time for bug fixes.

0-1整数計画法の適用

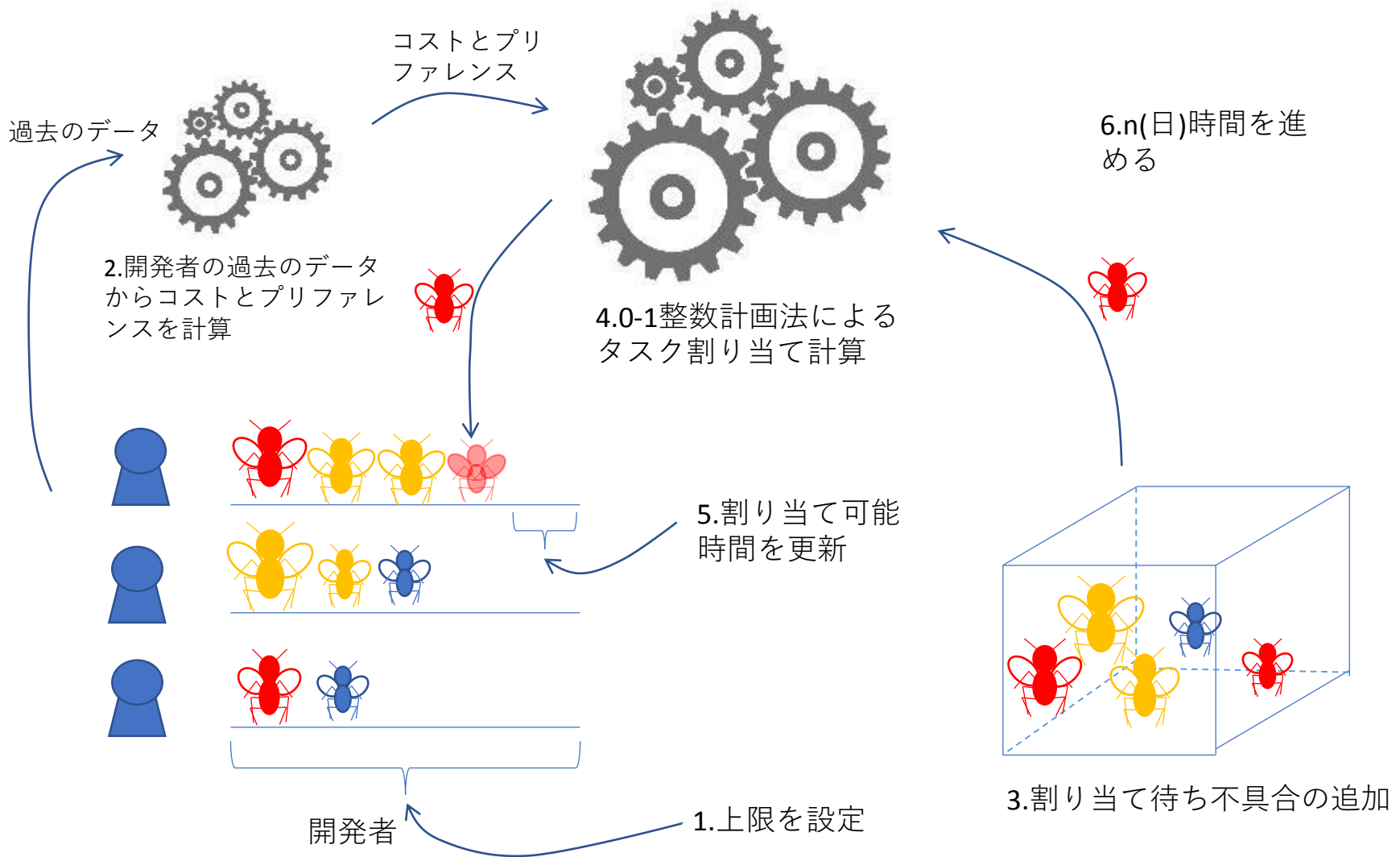
- **0-1整数計画法**：ナップサック問題の解を，選択する（1），選択しない（0）のどちらかにする方法

目的変数： $x_{ij} = \{0,1\}$

目的関数： $Maximize: \sum_{i=1}^m \sum_{j=1}^n P_{ij} x_{ij}$

制約条件： $\sum_{j=1}^n C_{ij} x_{ij} \leq T_i \quad (i = 1, 2, \dots, m)$
 $\sum_{j=1}^n x_{ij} \leq 1 \quad (j = 1, 2, \dots, n)$

提案手法の適用手順



ケーススタディ

実験

- 実際の現場で手法を用いるのではなく、実験上の日付でおこなう

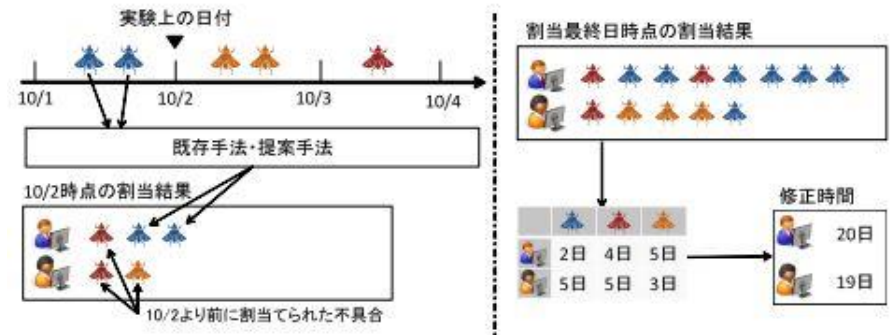


図 3 実験の概要

Fig. 3 An overview of the experiment.

- データセット

表 2 データセット

Table 2 Data sets.

プロジェクト	データセット	期間	解決済み 不具合 (件)	対象不具 合数 (件)	コンポー ネント数	カテゴリ リー数
Firefox	学習データ	2002/10/1~2011/9/30	10,165	2,536	28	60
	評価データ	2011/10/1~2012/9/30	1,648	659	31	39
Platform	学習データ	2002/10/1~2011/9/30	21,802	1,910	19	28
	評価データ	2011/10/1~2012/9/30	932	578	17	23

リサーチクエスチョン

- **RQ1**：既存手法は、特定の開発者へ負荷を集中させる傾向があるか？ 提案手法ではその問題を緩和できるか？
- **RQ2**：提案手法はプロジェクト全体の不具合修正時間の短縮化に寄与するか？
- **RQ3**：提案手法で用いる設定（プリファレンスと上限）が、タスク割り当ての最適化にどのように寄与するか？

リサーチクエスチョン

- **RQ1**：既存手法は、特定の開発者へ負荷を集中させる傾向があるか？ 提案手法ではその問題を緩和できるか？
- **RQ2**：提案手法はプロジェクト全体の不具合修正時間の短縮化に寄与するか？
- **RQ3**：提案手法で用いる設定（プリファレンスと上限）が、タスク割り当ての最適化にどのように寄与するか？

RQ1

- アプローチ：365日間と繁忙期の31日間の2つの期間で既存手法と提案手法によるタスク割り当ての結果を比較
- 結果

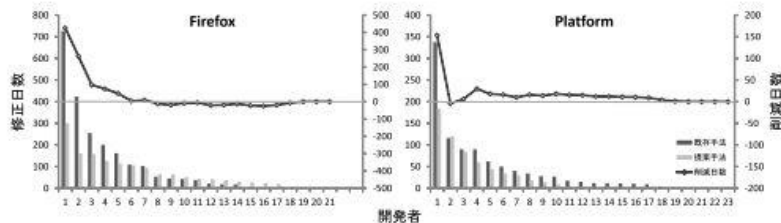


図 4 全期間における各開発者の修正日数（既存手法 vs. 提案手法）
Fig. 4 Distribution of the total cost of assignments by developer in the testing sets.

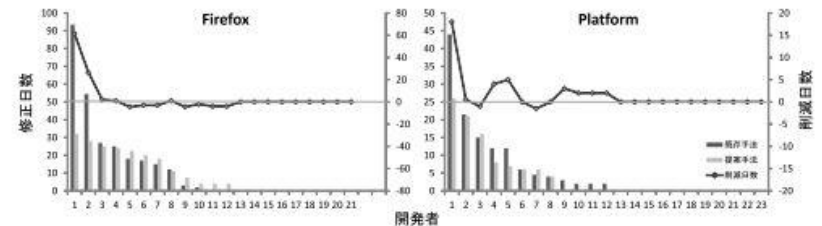


図 5 繁忙期における各開発者の修正日数（既存手法 vs. 提案手法）
Fig. 5 Distribution of the total cost of assignments by developer in a busy time of year.

リサーチクエスチョン

- **RQ1**：既存手法は、特定の開発者へ負荷を集中させる傾向があるか？ 提案手法ではその問題を緩和できるか？
- **RQ2**：提案手法はプロジェクト全体の不具合修正時間の短縮化に寄与するか？
- **RQ3**：提案手法で用いる設定（プリファレンスと上限）が、タスク割り当ての最適化にどのように寄与するか？

RQ2

- アプローチ：現状のタスク割り当て，既存手法，提案手法のそれぞれで得られた修正時間を比較
- 結果

表 5 現状の割当て方法，既存手法および提案手法によるタスク割当て結果の比較

Table 5 Comparison of task assignments between the three methods.

プロジェクト		Firefox			Platform		
手法		現状の割当て方法	既存手法	提案手法	現状の割当て方法	既存手法	提案手法
割り当てた不具合（件）	共通化前	659	439	440	578	386	572
	共通化後	377			385		
割り当てた開発者（人）	共通化前	16	17	18	20	19	12
	共通化後	16	15	18	20	19	11
合計修正日数（日）	共通化前	5,816	3,091	1,685	4,893	995	901
	共通化後	2,920	2,220	1,471	3,433	967	599

リサーチクエスチョン

- **RQ1**：既存手法は、特定の開発者へ負荷を集中させる傾向があるか？ 提案手法ではその問題を緩和できるか？
- **RQ2**：提案手法はプロジェクト全体の不具合修正時間の短縮化に寄与するか？
- **RQ3**：提案手法で用いる設定（プリファレンスと上限）が、タスク割り当ての最適化にどのように寄与するか？

RQ3

- アプローチ：Pの有無，Lの有無で4種類のモデルを構築しそれぞれを比較

表 6 条件の有無による 4 つのモデル

Table 6 Four models with and without *P* and *L*.

	<i>L</i> :なし	<i>L</i> :あり
<i>P</i> :なし	モデル A: プリファレンスと上限を設定せず、修正時間の短い開発者に不具合を優先的に割り当てる	モデル B: 上限を設定した条件下で、修正時間の短い開発者に不具合を優先的に割り当てる
<i>P</i> :あり	モデル C: プリファレンスのみ設定し、カテゴリごとのプリファレンスが高い開発者にタスクを割り当てる	モデル D: 上限を設定した条件下で、プリファレンスの合計が最大となるように不具合を割り当てる (提案手法)

- 結果

表 7 モデルごとのタスク割当ての結果

Table 7 Comparison of task assignments between the four models.

プロジェクト		Firefox				Platform			
モデル		A	B	C	D	A	B	C	D
適性		無	無	有	有	無	無	有	有
上限		無	有	無	有	無	有	無	有
割り当てた不具合 (件)	共通化前	659	659	559	440	578	578	575	572
	共通化後	440	440	440	440	572	572	572	572
割り当てた開発者 (人)	共通化前	1	15	10	18	1	14	12	12
	共通化後	1	15	9	18	1	14	12	12
合計修正日数 (日)	共通化前	4,284	3,066	4,897	1,685	5,780	1,186	997	901
	共通化後	2,860	2,022	2,115	1,685	5,720	1,174	910	901

考察

- 小規模なプロジェクトでは既存手法の利用が適する
- 修正時間算出方法の結果は現状の割り当てに比べて大きく制度を損なう場合がある
- 学習データ量は少なくても多すぎてもよくない

まとめ

- バグトリージにより不具合を適切に開発者に割り当てることができる
- 提案手法は修正タスク割り当てを分散して行い、全体の修正時間を削減する