

Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code

Denys Poshyvanyk, Andrian Marcus

ソースコード内の実装位置特定のための 形式概念分析と情報検索の合わせ技

Denys Poshyvanyk, Andrian Marcus

Feature Location (実装位置特定)

- 特定の機能に対応するソースコード片を見つけること
 - ここでいう機能は、ユーザに見える機能に関連付けられている概念
- 既存プログラムの保守を行う開発者が、プログラムを理解するために行う重要な作業
- 効率よく行うために、古くから様々な方法が研究されている

提案手法

- 潜在的意味インデキシング (Latent Semantic Indexing, LSI) と形式概念分析 (Formal Concept Analysis, FCA) を組み合わせた手法
- 開発者の手間を軽減する目的で組み合わせる

潜在的意味インデキシング (LSI)

- 文書群とそれらに含まれる単語群から、関連した概念の集合を生成して、その関係进行分析する技術
- ベクトル空間モデルを利用した自然言語処理の技法の1つ
- 各文書における単語の出現を表した**文書-単語行列** (Bag of Wordsモデル) から、**文書-概念行列**、**単語-概念行列**を生成する

文書-単語行列

	d1	d2	d3	d4
会場	1	1	1	0
店	0	0	0	1
自動車	0	1	0	0
自転車	0	0	1	1
行く	1	1	1	1
車	1	0	0	0

このままでは言い換え表現や表記ゆれなどに対応できない

単語の持つ潜在的意味を軸にした行列を作成して対応する

特異値分解

文書-単語行列 (X) を、直交行列 U、V と対角行列 Σ に分解する

$$X = U\Sigma V^T$$

ここで、直交行列 U、V がそれぞれ文書、単語の潜在的意味を軸にした行列 (基底行列) になっている

特異値分解の例

d1と**d3**は共通の潜在的意味を持つ



$$\begin{array}{l} \text{医者} \\ \text{病院} \end{array} \begin{array}{cc} \mathbf{d1} & \mathbf{d3} \end{array} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} -0.707 & 0 \\ 0 & 1 \\ -0.707 & 0 \end{bmatrix} \begin{bmatrix} 3.16 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -0.447 & 0 \\ 0 & 1 \\ -0.894 & 0 \end{bmatrix}^T$$



医者と**病院**は共通の潜在的意味を持つ

これら文書-概念行列、単語-概念行列を、検索用のインデックス(索引)とする

形式概念分析(FCA)

- 概念構造の明確化や事象の分析、データ可視化、データ依存関係などを明らかにする技法
- オブジェクトと属性の集合を、**コンテキスト表**や**コンセプトラティス**として表現する
- ソフトウェアを構成するプログラムの構造を分析する場合にも役立つ

表1.4 スポーツのコンテキスト

	a. 球技	b. 道具	c. ネット	d. 陸上	e. 屋内	f. 団体	g. 点数
1. 野球	×	×		×		×	×
2. サッカー	×			×		×	×
3. 相撲				×	×		
4. テニス	×	×	×	×			×
5. バレー	×		×	×	×	×	×
6. ゴルフ	×	×		×			×
7. マラソン				×			
8. 水泳					×		
9. ラグビー	×			×		×	×
10. レスリング				×	×		

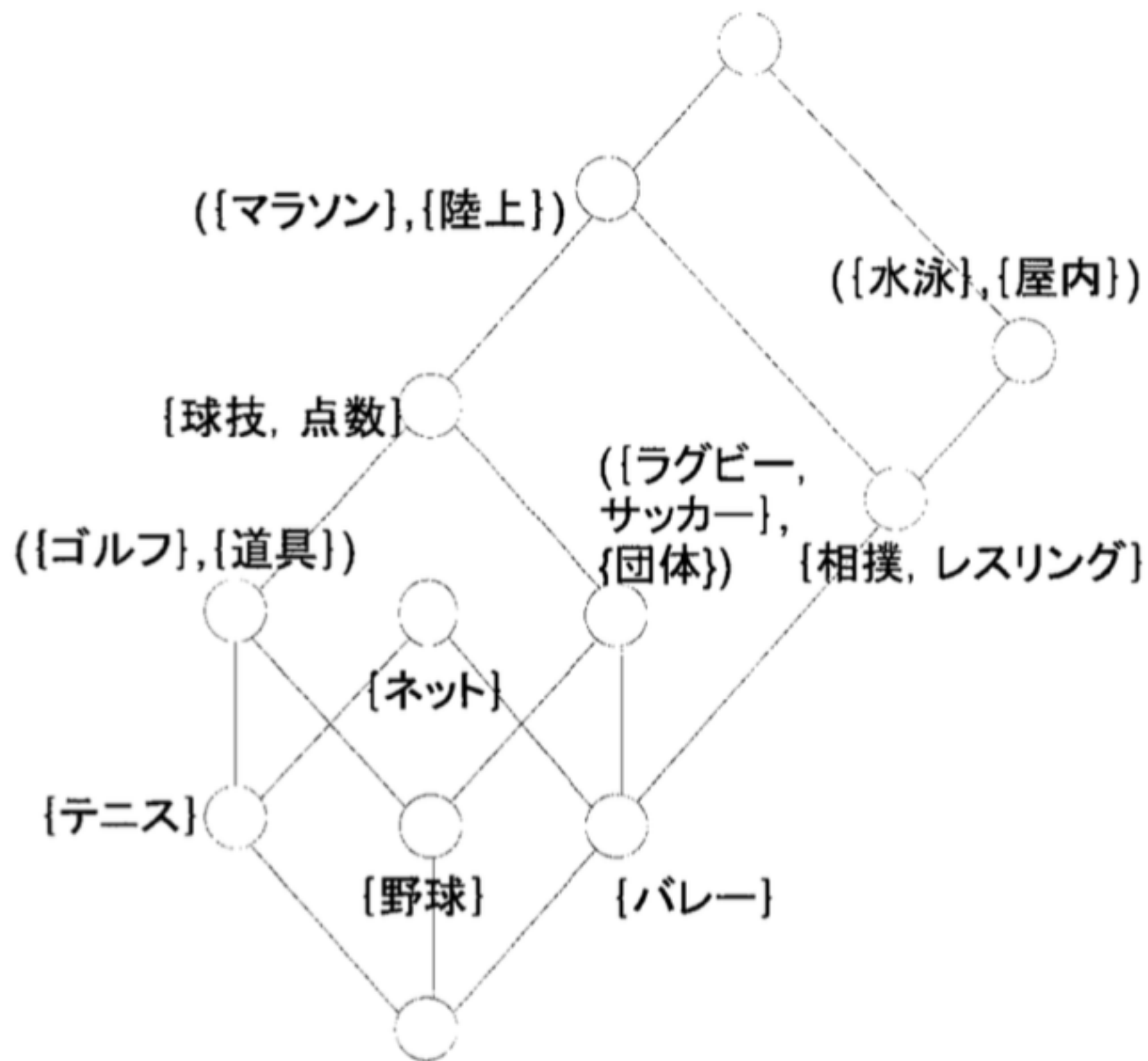


図 1.6 スポーツの概念ラティス

提案手法

- 潜在的意味インデキシング (LSI) と形式概念分析 (FCA) を組み合わせた手法 (再掲)
- **LSI** を用いて、開発者が入力するクエリ文字列をもとにランク付けされた検索結果を出力する
- **FCA** を用いて、LSI の結果から関連する結果をまとめて整理して提示する
- FCA によって生成されるコンセプトラティスで **開発者の手間を軽減する**

提案手法の流れ

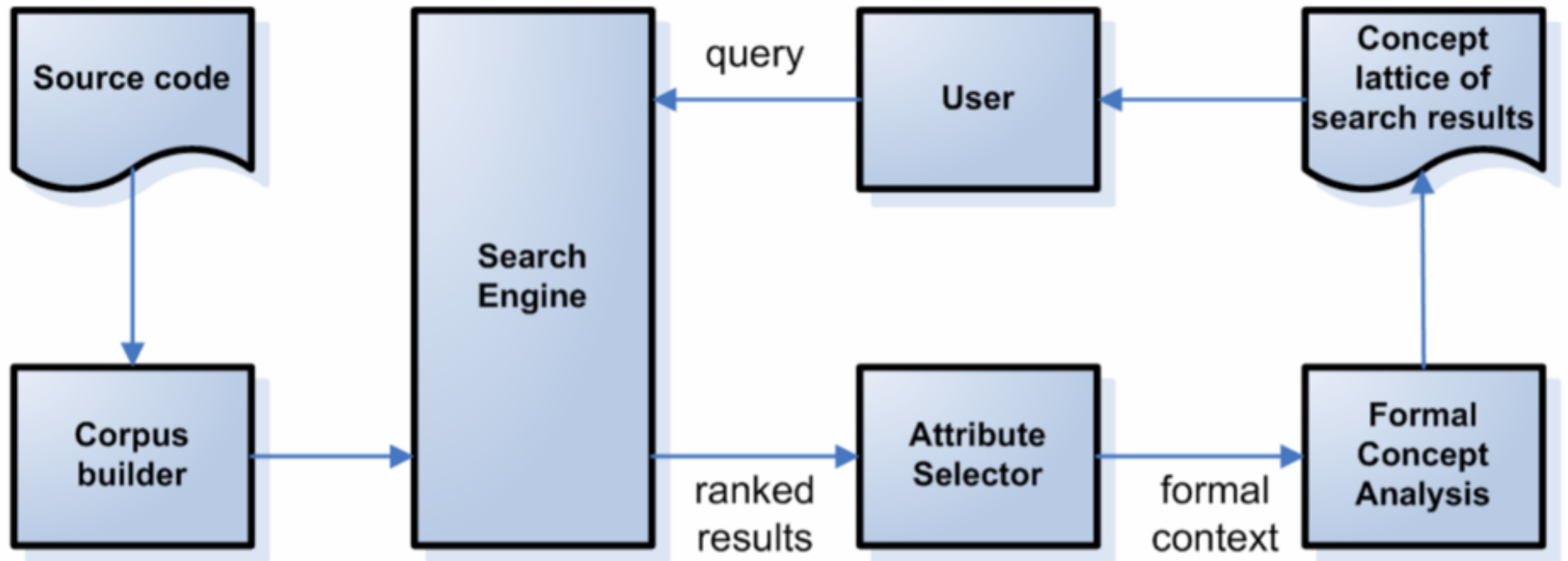


Figure 1. Concept location process using LSI and FCA

FCAに使う属性の選択

- FCAで使うために、コード片内の特徴的な単語を選択する
- Kuhnらの重み付けアルゴリズムを使う(ソースコードに対して適用し、関連語を選択することを目的に定義された手法)

Kuhnらの手法をもとにした手順

1. LSIの検索結果の上位n件のコード片集合**D_n**に含まれている単語を取り出す
2. このうち、検索結果のn位以下のコード片集合**D¹**にも含まれる単語は除外する
3. 以下を計算して重みとし、上位k個の単語を選択する(ここで|D¹|は集合D¹の要素数)

$$sim(t_i, D_n) = sim(t_i, D_n) - \frac{1}{|D^1|} \times \sum sim(t_i, D^1)$$

コンテキスト表の例

Table 1. Formal context: objects (six methods from source code of Eclipse) and attributes (shared in identifiers and comments of those methods)

	printer	print	page	job	device	paper	rendering
startJob		X		X			
endJob		X		X			
cancelJob		X		X			
startPage			X			X	X
endPage			X			X	X
getBounds	X				X	X	

抽出されたコンセプト

$C_1 = (\{\}, \{\text{paper}\})$

$C_2 = (\{\text{getBounds}\}, \{\text{printer}, \text{device}\})$

$C_3 = (\{\text{startPage}, \text{endPage}\}, \{\text{page}, \text{rendering}\})$

$C_4 = (\{\text{startJob}, \text{endJob}, \text{cancelJob}\}, \{\text{print}, \text{job}\})$

コンセプトラティス

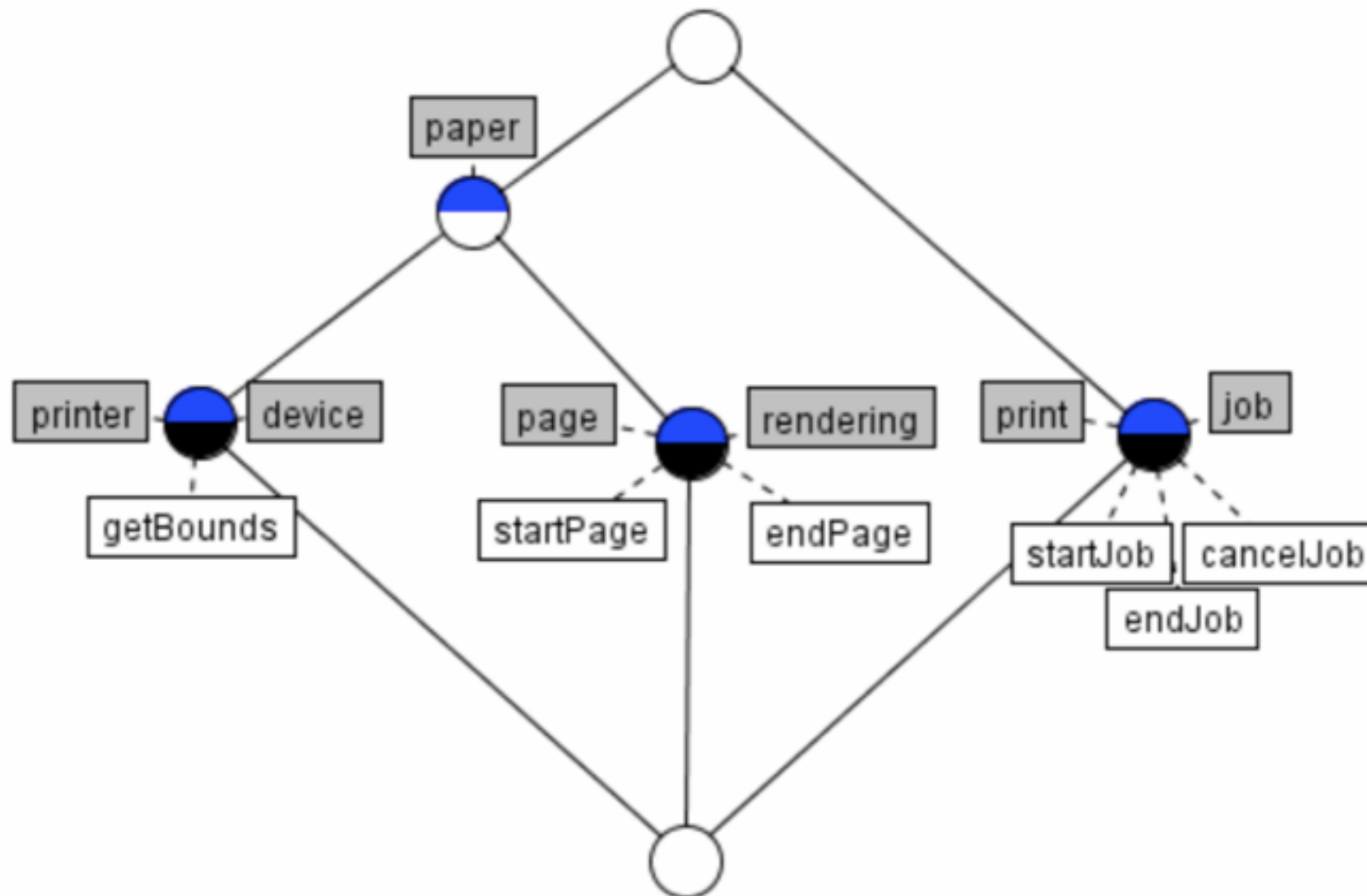


Figure 2. Concept lattice for the ‘*print page*’ feature. Grey boxes are attributes (words) and white boxes are objects (methods).

評価実験

- あえて変更を加えたプログラムを対象に手法を適用し、それによる変化を調べる
- 今回は報告済みのバグが含まれているプログラムを用いて、そのバグと関連のある実装位置を特定する

実験対象

Table 2. Eclipse source code and corpus vitals

Item	Count
MLOC	2.9
Vocabulary	56,863
Number of parsed documents	86,208

実験に使うバグ

Table 3. Descriptions, user queries, method ranking, and modified methods for Eclipse bugs

Bug #	Description	Query	Rank	Methods
34160 ²	The task list, which uses the native table widget, cannot be <i>sorted</i> by clicking on the <i>table headers</i>	“table header sort”	71	org.eclipse.swt.widgets.Table.createHandle org.eclipse.swt.widgets.Table.createWidget org.eclipse.swt.widgets.Table.kEventMouseDown org.eclipse.swt.widgets.Table.itemNotificationProc
25457 ³	<i>Renaming project</i> to the same name but with different case causes <i>source files</i> to be deleted if project's folder is locked by other application.	“rename project source”	89	org.eclipse.core.internal.localstore.FileSystemStore.move

結果

Table 4. Experimental results for locating two features in Eclipse with 24 concept lattices of various configurations

Bug	Docs	Terms	P _{MBA}	LDF	C	C _{VIEW}	LBC
34160	100	10	0.02	42.8%	17	8	47%
34160	100	15	0.025	78.5%	25	11	44%
34160	100	20	0.026	85%	33	11	33%
34160	100	25	0.033	96.6%	39	10	26%
34160	90	10	0.023	62.3%	17	8	47%
34160	90	15	0.027	98.4%	24	11	46%
34160	90	20	0.029	104%	31	11	36%
34160	90	25	0.039	174%	36	11	31%
34160	80	10	0.026	87.9%	15	7	47%
34160	80	15	0.033	138%	22	10	45%
34160	80	20	0.034	146%	28	10	36%
34160	80	25	0.043	207%	33	10	30%
25457	100	10	0.013	15%	13	8	62%
25457	100	15	0.013	15%	17	9	53%
25457	100	20	0.019	72%	25	10	40%
25457	100	25	0.021	91%	33	10	30%
25457	90	10	0.013	15%	13	8	62%
25457	90	15	0.013	15%	17	9	53%
25457	90	20	0.017	55%	25	11	44%
25457	90	25	0.046	318%	32	13	40%

- LDF: コンセプトラティスを用いた場合の精度向上度
- LBC: コンセプトラティスの読みやすさ
- P_{MBA}: ラティスの容量の上限
- C: ラティスのノード数
- C_{VIEW}: 各ノードの下位コンセプトのノード数

結果

- 単語数を25にするとコード片数にかかわらずLDFは最高になったが、ラティスサイズが増えてLBC(読みやすさ)が下がった
- 開発者が調査するのに必要なラティスのノード数は7~13
- 関連情報の分類の観点からコンセプトラティスは効果的で、単語数を増やすほど効果的だった
- 90コード片(n)と20~25単語(k)をFCAに適用すると最も効果的だった

まとめ

- 潜在的意味インデキシング (LSI) と形式概念分析 (FCA) を組み合わせた Feature Location 手法を提案した
- 提案手法は開発者の手間の軽減に寄与した
- 90コード片 (n) と20~25単語 (k) をFCAに適用すると最も効果的だった