Software Architecture Recovery based on Pattern Matching

PhD Dissertation Synopsis
Kamran Sartipi
School of Computer Science, University of Waterloo
Waterloo, ON. N2L 3G1, Canada
ksartipi@math.uwaterloo.ca

Abstract

This paper is a summary of the author's thesis that presents a model and an environment for recovering the high level design of legacy software systems based on user defined architectural patterns and graph matching techniques. In the proposed model, a high-level view of a software system in terms of the system components and their interactions is represented as a query, using a description language. A query is mapped onto a pattern-graph, where a component and its interactions with other components are represented as a group of graph-nodes and a group of graph-edges, respectively. Interaction constraints can be modeled by the description language as a part of the query. Such a pattern-graph is applied against an entity-relation graph that represents the information extracted from the source code of the software system. An approximate graph matching process performs a series of graph transformation operations (i.e., node/edge insertion/deletion) on the pattern-graph and uses a ranking mechanism based on data mining association to obtain a sub-optimal solution. The obtained solution corresponds to an extracted architecture that complies with the given query.

1 Introduction

In a nutshell, the existing approaches to software architectural recovery can be classified as clustering-based techniques and pattern-based techniques. The clustering-based techniques generate architectural components by gradually grouping the related system entities using a similarity measure [6]. On the other hand, the pattern-based techniques first compose a high-level mental model of the system architecture (also known as conceptual architecture or architectural pattern) using a modeling means such as a query language [5] or a block diagram [4], and then a pattern matching engine searches to identify an instance of the architectural pattern in the software system.

The clustering-based techniques can only implicitly guide the result of the recovery toward a constrained objective. In contrast, the pattern-based techniques can use a formalism to model structural constrains that are derived from the application domain and system documentation and can link the analysis results with intended objectives, hence provide a user/tool cooperative environment for architectural recovery. However, the pattern-based architecture recovery approaches suffer from the lack of an expressive representation of the architectural patterns, and in most cases an expert user is required to formulate the architectural pattern.

This thesis [7] argues that a pattern-based environment for software architecture recovery with an expressive architectural pattern language that incorporates knowledge from the system's domain and documentation, and a process that ensures a repeatable recovery result would best suit to the requirements of the architectural recovery problem. Moreover, the software systems are intuitively represented as graphs and the reverse engineering community is on the verge of adopting a graph standard for information exchange among the existing reverse engineering tools [3].

Specifically, this thesis proposes a pattern-based recovery approach whose objectives can be specified in terms of the structural properties that are defined through an architectural pattern. The proposed architectural pattern is based on the expressive features of the architecture description languages (ADLs) and is incrementally generated via an interactive procedure that allows to incorporate the knowledge from the application domain and system documentation. The proposed approach considers the high-level design of a system as a pattern-graph, and models the recovery process as a graph pattern matching problem that matches such a high-level pattern-graph of the system with an entity-relationship graph representation of the source-code system entities. The result of the recovery can be directly tested against the recovery objectives through: i) conformance checking with the available documentation that ensures the decomposition of the core system functionality into components; ii) measuring the

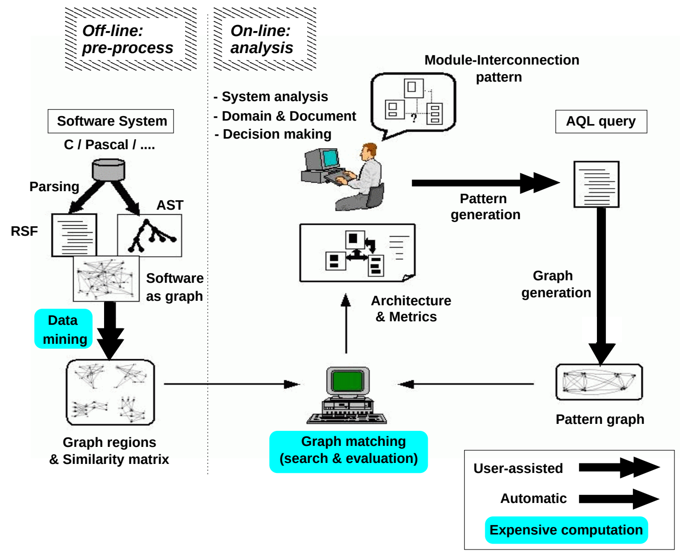


Figure 1. The proposed interactive environment for pattern-based software architecture recovery and evaluation.

modularity quality of the recovered architecture to ensure the recovery of a maintainable system; and iii) conformance with the component and connector size and type constraints imposed by the pattern.

2 Proposed recovery environment

Figure 1 illustrates the different parts of the proposed environment where the thick arrows signify the automatic or user-assisted processes in the environment; boxes represent the different forms of information in the environment; the thin arrows indicate the inputs and output of the graph matching engine; and the user is the high-level decision maker that produces a mental model of the architecture and verifies the result of recovery. The proposed environment employs techniques from approximate graph pattern matching, data mining, clustering, and architecture description languages. The environment consists of an *off-line pre-process* part and an *on-line analysis* part, that are performed in the following steps.

Step 1: in the off-line part, the software system (written in a procedural language such as C) is parsed and represented as an *attributed relational graph*; the system graph is pre-processed using data mining techniques to extract groups of highly intra-related entities; and finally the system graph is represented as a group of graph regions in a database to be used by the graph matching engine. Each graph region represents a reduced search space for recovery of a component.

Step 2: in the on-line part, using an iterative recovery process the user incrementally generates an architectural pattern for the software system based on domain knowledge, system documents, or tool-provided system analysis information. The architectural pattern (can be referred to as conceptual architecture) is directly defined for the tool, using a proprietary language that we call *Architecture Query Language* (AQL). Therefore, a query in AQL represents a macroscopic graph-form pattern for a part or the whole of the system architecture to be recovered.

Step 3: the graph matching engine then represents the architectural pattern in the AQL query as a graph-pattern; sub-optimally matches the graph-pattern against the system graph regions in the database; and finally presents the recovered architecture along with the evaluation metrics to the user.

Step4: the user investigates the result of recovery using the tool-provided metrics and if the result is satisfactory, stops the recovery process; otherwise, the user augments the architectural pattern by adding another component and/or by adding/adjusting the constrained connectors, and then resumes a new recovery iteration.

The proposed environment has been implemented in a toolkit called *Alborz* which is described in Chapter 8 of the thesis. Alborz provides the result of the architectural recovery into two forms: i) HTML pages for the recovered components, tool generated metrics, and source code, to be visualized by a Web browser such as Netscape; and ii) graphs of boxes and arrows to be visualized by the Rigi tool [1], where the boxes are the system files or the analyzed components and the arrows are either the resource interaction (i.e., import/export) between the components or their association values.

Contributions: the specific contributions of this thesis with respect to the solid circles in the environment of Figure 1, and with refer to the Chapters of the thesis, are discussed in the items C1 to C6 below:

C1: a new domain model that allows to represent the software system as an attributed relational graph at a higher-level of abstraction than the source-code which is suitable for architectural recovery (in Chapter 3). **C2:** two new similarity metrics that are based on the structural properties of the groups of entities with *maximal association* (defined later) generated by data mining techniques (in Chapter 3). **C3:** a novel technique to limit the computational complexity of the graph matching process for architectural recovery using graph regions (in Chapter 3). **C4:** an architecture query language (AQL) to model an architectural pattern of constrained components and their interactions (in Chapter 4). **C5 and C6:** a new multi-phase approximate graph matching algorithm to match a pattern-graph with the regions of the software system graph (in Chapters 5 and 6).

3 Overview of the graph matching process

In modeling the proposed incremental graph matching approach for architecture recovery, a number of intermediate graphs and connector edges are defined. Such intermediate graphs allow to represent the architectural pattern and an input graph at each iteration of the matching process in terms of their constituents (i.e., a number of recovered components and their import/export links) and consequently formulate these graphs using recursive graph summation equations. This formulation provides a valuable means for modeling and implementing the whole incremental pattern matching process.

3.1 Software system representation

In this approach, the software system and the architectural pattern are presented using the *attributed relational graph* notation.The software system is represented by a *source-graph* $G^s = (N^s, R^s)$, where the nodes (n_j) represent files, functions, datatypes, and variables and the edges (r_y) represent *call* and *use* relationships. The nodes and edges comply with the specific domain model defined for architectural analysis [7].

The source-graph G^s provides a search-space for the matching process. However, since even in a medium-size software system the number of entities and relationships that are generated are prohibitively high, any matching algorithm will be intractable.

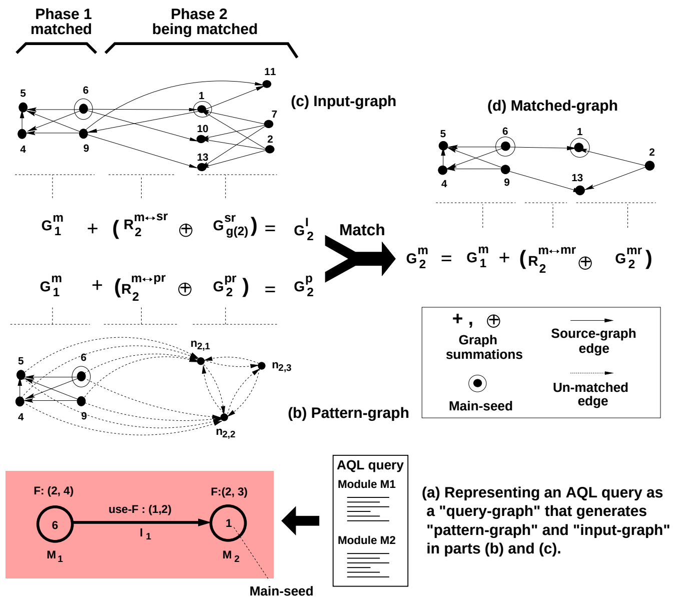
To address this problem, using data mining association relation, the search space is divided in order to generate a collection of sub-spaces, where each sub-space is a sub-graph of the source-graph G^s, namely a *source-region* G^{sr}_j. Each source-region G^{sr}_j is distinguished by the main-seed node n_j in that region.

Data mining: in this context, the data mining technique Apriori [2] is used to discover all groups of system entities that are related by maximal association. Maximal association is defined in a group of system entities in the form of a maximal set of entities that all share the same relation to every member of another maximal set of entities.

Every node in a source-region is labeled with an association-based similarity value to the main-seed of the source-region as a means for the matching process to operate on groups of highly associated entities. The similarity value between two entities and between two groups of entities are defined using the groups of entities with maximal association extracted by the data mining technique.

3.2 Architectural pattern representation

The architectural pattern is represented by an *AQL query*. Each component (module or subsystem) of the query uses one or more entities as fixed entities to appear in the result of the recovery, namely *main-seed*(s) which determine the corresponding search-space to be searched for the component, and *seeds* which just appear in the result without search. In the following a part of an AQL query, consisting of a subsystem S1 of files and its interconnection links to other subsystems is shown:



Phase 1 matched
Phase 2 being matched
(c) Input-graph
(d) Matched-graph
$G^m_1 + (R^{m\leftrightarrow sr}_2 \oplus G^{sr}_{g(2)}) = G^I_2$
Match
$G^m_2 = G^m_1 + (R^{m\leftrightarrow mr}_2 \oplus G^{mr}_2)$
$G^m_1 + (R^{m\leftrightarrow pr}_2 \oplus G^{pr}_2) = G^p_2$
+ , ⊕ Graph summations
Source-graph edge
Main-seed
Un-matched edge
(b) Pattern-graph
AQL query
Module M1
Module M2
(a) Representing an AQL query as a "query-graph" that generates "pattern-graph" and "input-graph" in parts (b) and (c).
F: (2, 4)
F:(2, 3)
use-F : (1,2)
l_1
M_1
M_2
Main-seed

Figure 2. An example of matching a pattern-graph with an input-graph to yield a matched-graph at phase 2 of the matching process.

```
BEGIN-AQL
SUBSYSTEM: S1
    MAIN-SEEDS:        files  e_edit,  e_update
    IMPORTS:
      RESOURCES:       rsrc  ?IR,
                       rsrc  ?R1(6 .. 10) S2,
                       rsrc  ?R2(12 .. 20) S4
    EXPORTS:
      RESOURCES:       rsrc  ?ER,
                       rsrc  ?R3(10 .. 15) S2,
                       rsrc  ?R4(1 .. 5) S3
    CONTAINS:
      FILES:           file  $CFI(7 .. 10),  files  e_edit,  e_update
END-AQL
```

The above AQL fragment is interpreted as: the subsystem S1 which will be instantiated with seven to ten files, and definitely contains the files *e_edit* and *e_update* (main seeds), imports minimum six and maximum ten resources (?R1) from subsystem S2. A similar interpretation holds for the *EXPORTS* and *CONTAINS* sections. The notations *?IR* and *?ER* in the import and export parts denote unidentified quantities of links between the current subsystem and any other subsystems in the query that have not been specified by the architectural pattern, therefore, are not matched by the matching algorithm.

An AQL query can be further represented as a *query-graph* consisting of composite nodes that are linked through composite edges. The i^{th} composite node is expanded into a *pattern-region* G_i^{pr}, and each composite edge is expanded into a group of *edge-bundles* $\mathcal{R}_i^{m \leftrightarrow pr}$, as illustrated in the example of Figures 2(a) and (b). Each edge-bundle connects every node from a corresponding recovered component to one node in the pattern-region G_i^{pr} with respect to the direction of the composite edge. The expansion of the query-graph generates a *pattern-graph* G_i^p. The rationale for expanding the composite-edges is to allow every subset of the nodes in a source component to be connected to every subset of the nodes in the destination component, according to the constraints modeled in the AQL query.

3.3 Graph pattern matching

In order to address the tractability of the recovery process, the whole recovery process is divided into k incremental phases (as k partial-matchings) where k is the number of components to be recovered and the current matching phase is denoted by "i" ($i \in$ [1.. No. of components]). Therefore, the recovery process performs a *multi-phase* matching. This allows to back-tracking to the previous phase of recovery if the recovery of the current component fails. Figure 2 illustrates an example of the matching process at phase 2, where the process computes a sub-optimal match between a pattern-graph G_2^p that originates from an AQL query and an *input-graph* G_2^I that originates from the system source-graph G^s.

We use the A^* search algorithm that has been modified by a "*Bounded-Queue*" heuristic (namely *BQ-*A^*) to compute a sub-optimal match between the pattern-graph G_i^p and input-graph G_i^I while the AQL query constraints are not violated. The *BQ-*A^* algorithm generates a search-tree and uses a cost function based on graph transformation operations (i.e., node or edge deletion/insertion) that guides the search algorithm to expand a tree-node with minimum cost.

In Figures 2(b) to (d), the graph summation notations "plus $+$" for connecting two graphs, and "oplus $\oplus$" for connecting a graph to a group of connector-edges are used to model the whole incremental pattern matching process in terms of the recursive graph algebraic equations.

3.4 Evaluation

A comprehensive set of experimentations related to the time/space complexity, accuracy, stability, and quality of the proposed architecture recovery technique has been presented in the Chapter 8 of the thesis [7].

The experimentations are performed on six middle-size industrial systems, namely: i) *Xfig.3.2.3* drawing editor, ii) *Clips.4.20* expert system builder, iii) *Apache.1.2.4* http server; iv) *Bash.2.03* Unix shell; v) *Elm.2.5.6* Unix mail system; and vi) *Ghostview.3.5.8* postscript file viewer. Figure 3(a) presents the source-code related characteristics of the experimentation suite. Figure 3(b) illustrates the architectural pattern of the Clips system with five subsystems, where the constrained components and connectors are shown. Figure 3(c) illustrates the result of recovery, where the constraints of the architectural pattern have been satisfied.

Figures 3(d) to (g) illustrate the result of recovery for the Clips system using the Netscape browser and the Rigi graph visualizer. In Figure 3(h) the accuracy of the recovery result is presented using the Precision and Recall metrics.

4 Conclusion

The thesis of this work argues that the pattern-based approaches to software architecture recovery with an expressive pattern language, and with an interactive and incremental recovery process that incorporates the knowledge from the application domain and system documentation, best suites the objectives of an architecture recovery task. The paper, then briefly discussed the characteristics of such an environment with focus on the graph matching model, tractability, and accuracy of the proposed technique.

Acknowledgment: I would like to thank Professor Kostas Kontogiannis for supervising this thesis.

References

[1] Rigi, URL = http://www.rigi.csc.uvic.ca/rigi/rigiindex.html.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499, 1994.

[3] Bell, IBM. *Workgroup on Standard Exchange Format (WoSEF)*, Limerick, Ireland, June 06 2000.

[4] P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, et al. The software bookshelf. *IBM Systems Journal*, 36(4):564–593, November 1997.

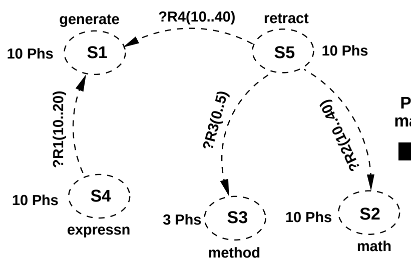
[5] R. Kazman and M. Burth. Assessing architectural complexity. In *Proceedings of the CSMR*, pages 104–112, 1998.

[6] A. Lakhotia. A unified framework for expressing software subsystem classification techniques. *Journal of Systems and Software*, 36(3):211–231, 1997.

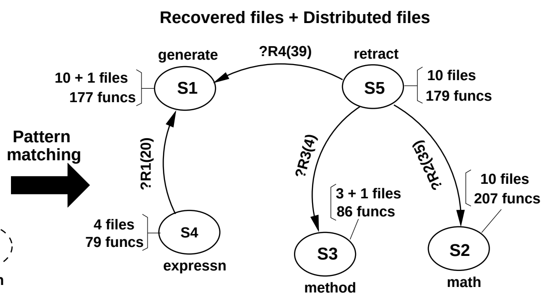
[7] K. Sartipi. *Software Architecture Recovery based on Pattern Matching*. PhD thesis, School of Computer Science, University of Waterloo, Waterloo, ON, Canada, 2003.

Systems	*KLOC*	*Files*	*Funcs*
Xfig	74	98	1662
Clips	40	44	736
Apache	38	42	709
Bash	44	47	1017
Elm	35	62	420
GSview	39	47	469

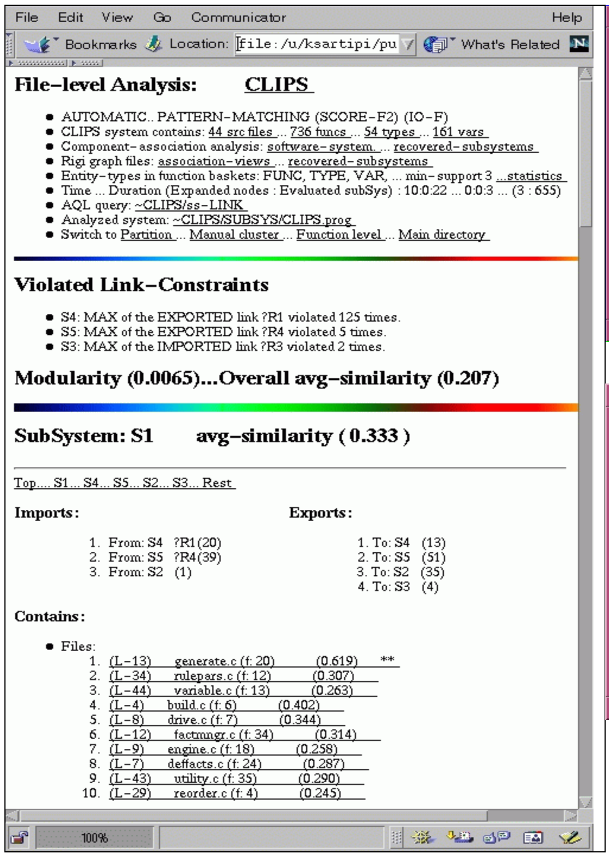
(a) Source code information of the six analyzed systems



(b) Architectural pattern of Clips using AQL query



(c) Recovered architecture of Clips


File Edit View Go Communicator Help
Bookmarks Location: file:/u/ksartipi/pu What's Related
100%

File–level Analysis: CLIPS

- AUTOMATIC.. PATTERN–MATCHING (SCORE–F2) (IO–F)
- CLIPS system contains: 44 src files ... 736 funcs ... 54 types ... 161 vars
- Component–association analysis: software–system. ... recovered–subsystems
- Rigi graph files: association–views ... recovered–subsystems
- Entity–types in function baskets: FUNC, TYPE, VAR, ... min–support 3 ...statistics
- Time ... Duration (Expanded nodes : Evaluated subSys) : 10:0:22 ... 0:0:3 ... (3 : 655)
- AQL query: ~CLIPS/ss–LINK
- Analyzed system: ~CLIPS/SUBSYS/CLIPS.prog
- Switch to Partition ... Manual cluster ... Function level ... Main directory

Violated Link–Constraints

- S4: MAX of the EXPORTED link ?R1 violated 125 times.
- S5: MAX of the EXPORTED link ?R4 violated 5 times.
- S3: MAX of the IMPORTED link ?R3 violated 2 times.

Modularity (0.0065)...Overall avg–similarity (0.207)

SubSystem: S1 avg–similarity (0.333)

Top.... S1... S4... S5... S2... S3... Rest

Imports:

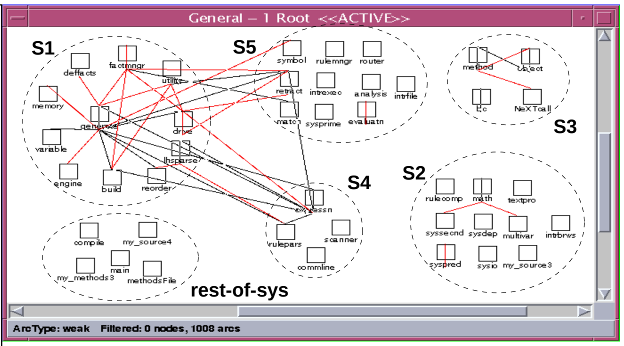
1. From: S4 ?R1(20)
2. From: S5 ?R4(39)
3. From: S2 (1)

Exports:

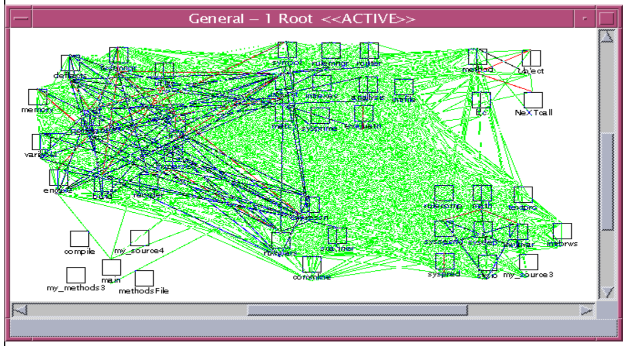
1. To: S4 (13)
2. To: S5 (51)
3. To: S2 (35)
4. To: S3 (4)

Contains:

- Files:
  1. (L–13) generate.c (f: 20) (0.619) **
  2. (L–34) rulepars.c (f: 12) (0.307)
  3. (L–44) variable.c (f: 13) (0.263)
  4. (L–4) build.c (f: 6) (0.402)
  5. (L–8) drive.c (f: 7) (0.344)
  6. (L–12) factmngr.c (f: 34) (0.314)
  7. (L–9) engine.c (f: 18) (0.258)
  8. (L–7) deffacts.c (f: 24) (0.287)
  9. (L–43) utility.c (f: 35) (0.290)
  10. (L–29) reorder.c (f: 4) (0.245)

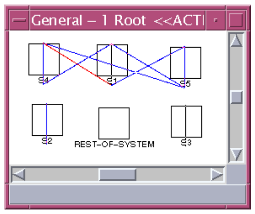


(d) Final recovered of Clips system with Strong and Medium links



(e) Adding "loose" and "weak" association links to part (d).

(f) Recovered subsystem S1 with various statistics and links to other HTML pages.



(g) Assocition links among the recovered subsystems.

Recovered subsystems	No. of files	Clips subsystems (documented)	No. of files	Pre-cision	Recall
S1	11	- Defrule structures - Inference engine	13	82%	70%
S2	10	- Rule manipulation	6	50%	83%
S3	4	- Object	3	75%	100%
S4	4	- Expression eval	4	75%	75%
S5	10	- System function - User interface	7	40%	57%
rest-of-sys	5	————————	———	———	———

(h) Architectural recovery evalution of the Clips system.

Figure 3. (a) The architectural pattern of Clips with five subsystems, where the constrained components and connectors are shown. (b) In the result of recovery, the constraints of the architectural pattern have been satisfied. (d) The accuracy of the recovery result using the Precision and Recall metrics.