From Informal Process Diagrams To Formal Process Models

Debdoot Mukherjee[1], Pankaj Dhoolia[1], Saurabh Sinha[1], Aubrey J. Rembert[2], and Mangala Gowri Nanda[1]

[1] IBM Research – India, {debdomuk, pdhoolia, saurabhsinha, mgowri}@in.ibm.com
[2] IBM T.J. Watson Research Center, ajrember@us.ibm.com

Abstract. Process modeling is an important activity in a business-transformation project. Free-form diagramming tools, such as PowerPoint and Visio, are the preferred tools for creating process models. However, the designs created using such tools are informal sketches, which are not amenable to automated analysis. Formal models, although desirable, are rarely created (during early design) because of the usability factors associated with formal-modeling tools. In this paper, we present an approach for automatically inferring formal process models from informal business process diagrams, so that the strengths of both types of tools can be leveraged. We discuss different sources of structural and semantic ambiguities, commonly present in informal diagrams, which pose challenges for automated inference. Our approach consists of two phases. First, it performs structural inference to identify the set of nodes and edges that constitute a process model. Then, it performs semantic interpretation, using a classifier that mimics human reasoning to associate modeling semantics with the nodes and edges. We discuss both supervised and unsupervised techniques for training such a classifier. Finally, we report results of empirical studies, conducted using flow diagrams from real projects, which illustrate the effectiveness of our approach.

1 Introduction

Business Process Models are key artifacts that are created during the early stages of a business-transformation project. A *business process model* depicts how various tasks are coordinated to achieve specific organizational goals. Such models are used to build a consensus among the stakeholders during the requirements-elicitation phase and then drive the subsequent transformation phases. Free-form diagramming tools, such as Powerpoint and Visio, are widely used for creating informal sketches of process models.

On the one hand, these tools are easy-to-use, ubiquitous, offer creative expression, and have a low barrier to adoption. On the other hand, the diagrams created using such tools have no formal underpinnings; therefore, they are not amenable to automated analysis—*e.g.,* for model checking, process improvements, process reuse, and bootstrapping process realization. Unlike the free-form diagramming tools, formal process-modeling softwares offer many such benefits, but suffer from a high barrier to adoption; this occurs for different reasons, such as complexity,

costs, and the requirement of some level of formal training. Empirical studies reveal that the authoring constraints imposed by formal-modeling tools have generated mixed reactions from designers and have resulted in limited adoption of the tools [10].

To take advantage of the merits of free-form diagramming and yet leverage the benefits of formal modeling, automated techniques for converting informal sketches to formal process models are essential. A manual approach can be tedious and error-prone; especially when enterprises want to harvest formal models from a large corpus of legacy flow diagrams—a scenario that is attracting increasing interest.

Many existing formal-modeling tools (*e.g.,* Websphere Business Modeler[3] (WBM), ARIS,[4] System Architect,[5] and Lombardi[6]) offer, to various degrees, capabilities to import informal diagrams, such as Visio diagrams. However, accurately interpreting these diagrams, which are primarily intended for human consumption is challenging. Diagramming tools offer a rich collection of shapes (known as *stencils*) from which designers freely choose depictions for process flow entities, such as activities, events, gateways, etc. Existing tools perform mainly a shape-based transformation, using fixed or pluggable mappings from drawing shapes to process modeling entities. This is inadequate because often the same shape is used to represent different semantics. Further, these tools are not able to interpret diagrammatic cues (*e.g.,* dangling connectors) commonly used in describing the flow connections; therefore, they often identify imprecise flow structures.

To address the limitations of existing tools, we present an automated approach for extracting formal process models, that conform to a given target metamodel,[7] from informal process-flow diagrams. It consists of two phases: a structure-inference phase and a semantic-interpretation phase. In the first phase, the approach precisely infers the flow-graph structure of a diagram, in terms of the nodes and edges (*i.e.,* flow elements) that comprise the diagram. It identifies each shape, or set of shapes, that could correspond to a flow node. Next, we propose a novel edge-inference algorithm to trace the lines between the identified nodes and infer the set of directed edges in the diagram. In this phase, the approach also infers the association of unlinked texts to appropriate flow elements. In the second phase, our approach annotates each node and edge with a process-modeling semantics, defined in the target metamodel. To perform the annotation, we use pattern classification. Specifically, the approach uses a classifier trained on relational, geometric, and textual features of flow elements to perform semantic disambiguation. We present both supervised and unsupervised approaches for training such a classifier.

---

[3] `http://www.ibm.com/software/integration/wbimodeler/advanced/features/`

[4] `http://www.ids-scheer.com/en/ARIS\_ARIS\_Platform/3730.html`

[5] `http://www.ibm.com/software/awdtools/systemarchitect/`

[6] `http://www.lombardisoftware.com/enterprise-bpm-software.php`

[7] A target metamodel lists the set of elements allowed in process modeling. For example, the Business Process Modeling Notation (BPMN) defines a metamodel comprising of activities, gateways, events, swimlanes, artifacts and connecting objects.

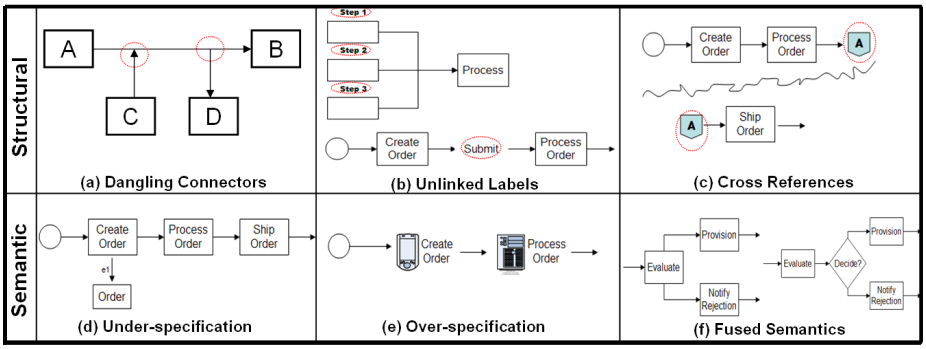

Fig. 1. Common structural and semantic ambiguities in process flow diagrams.

To evaluate the effectiveness of our approach, we implemented it in a tool called *i*DISCOVER that infers formal models from Visio diagrams. We conducted empirical studies using flow diagrams taken from real business-transformation projects. Our results illustrate that, for the diagrams considered, *i*DISCOVER infers formal models with high precision and recall, and outperforms existing commercial tools. Our results also indicate that most standard classifiers are applicable for interpreting process semantics with our feature space modeling. Interestingly, an unsupervised clustering approach, which may be used in practical settings where training data is unavailable, also proves nearly as effective as supervised ones.

The main benefit of our approach is that it automates, with a high degree of accuracy, a transformation task, that is tedious to perform manually. In doing so, the approach enables process engineering to leverage the strengths of both free-form diagramming tools and formal-modeling tools. More importantly, such a facility can help greater industrial adoption of formal methods developed in BPM research – currently the unavailability of formally specified process models in enterprises proves to be an impediment in applying such research.

The main contributions of this work are

- The development of a novel end-to-end approach for converting informal flow diagrams to formal process models, addressing both structural and semantic ambiguities that are commonly present in the informal diagrams.
- The implementation of the approach in a tool called *i*DISCOVER that converts Visio process flow diagrams to BPMN process models.
- An empirical evaluation that demonstrates the effectiveness of the approach

2 Diagram Interpretation Challenges

Figure 1 presents some ambiguities, which present challenges in interpreting the structure and semantics of flow models. Our empirical study (Section 4) shows that such ambiguities are indeed common in real process diagrams.

2.1 Structural Ambiguities

Identifying the correct set of drawing shapes corresponding to a node or an edge is hard when, for example, the edges are not properly connected to nodes or multiple lines are connected to form a single flow. The top part of Figure 1 illustrates three common structural ambiguities.

Dangling Connectors Existing tools can recognize a line to be an edge only if the line is properly glued[8] at both ends of two 2D shapes. However, users often join multiple lines to represent a single edge. Moreover, the endpoints of a line may be left dangling (*i.e.,* not be properly glued). In Figure 1(a), four edges exist: (A, B), (A, D), (C, B), and (C, D). But, existing tools can recognize only A-B because it is the only properly glued edge.

Unlinked Labels People often use separate drawing shapes to specify a flow element and its text label. In Figure 1(b), `Submit` is intended to be a label on the edge from `Create Order` to `Process Order`. Label association becomes a challenge when nearness alone does not suffice to tie unlinked texts with shapes identified as flow elements. Tracking patterns of text label usage may help—for example, if text labels are consistently placed on the top of shapes (*e.g.,* `Step x`), we can apply that pattern to resolve ambiguous cases.

Cross-references Cross-reference linkages across diagrams are often required to split a large diagram across pages for convenience, as shown in Figure 1(c). Use of cross references can occur within a single page as well.

2.2 Semantic Ambiguities

Inferring the semantics for flow elements is straightforward if each drawing shape is used consistently to convey a single modeling semantic. However, in practice, the following scenarios are extremely common and pose challenges for semantic interpretation.

Under-specification This occurs when different instances of the same shape are used to convey different semantics. For example, in Figure 1(d), a rectangle is used to denote both the output data artifact `Order` and the step `Create Order`. In general, under-specification lowers the effectiveness of a simple shape-based mapping of diagram elements to process model entities.

Over-specification This occurs when the same semantic is being conveyed by different shapes. In Figure 1(e), both `Create Order` and `Process Order` are activities, but are represented using different graphics. Over-specification too tends to reduce the usefulness of shape-based mapping: the number of shapes to be enumerated by such approaches can become prohibitively large.

Fused Semantics In Figure 1(f), the two flow fragments are semantically equivalent. The left fragment has an `Evaluate` block that represents a fusion of a task and a decision. In the fragment on the right, `Evaluate` and `Decide` are separate entities. Automatic interpretation of such fused semantics is difficult.

3 Automated Process Model Discovery

Our approach consists of two phases: *structural inference* and *semantic interpretation*. The structural-inference phase takes as input a *flow diagram*, and extracts a *flow graph*, which consists of nodes and edges. Additionally, the first phase computes information, such as structure, geometry, and text, for each flow element

[8] Most diagramming formats support a notion of proper connection or glue. On clicking a connector, the endpoints turn red if they are properly connected to shapes and are green if they are dangling.

(*i.e.,* node and edge). The second phase of the algorithm constructs the *process model* from the flow graph by associating modeling semantics with each flow element using pattern classification. Specifically, this phase applies a classifier that, based on the relational, geometric, and textual features of the flow elements, performs semantic disambiguation. The resulting process model is well-defined in terms of both structure and semantics, and thus can be formally analyzed.[9]

3.1 Structure Inference

The goal of the first phase is to infer the flow graph nodes and edges. It does this in three steps. First, it parses the input flow diagram to identify the basic diagram elements, which consists of shapes, lines, and text. Second, it constructs the set of nodes, selects candidate edges from diagram elements and determines associations of text with nodes and candidate edges. Finally, this phase applies an edge-inference algorithm to compute the flow.

Diagram Element Extraction An informal flow diagram is a collection of diagram elements such as (1) *shapes*, which are candidates for nodes, (2) *lines*, which are candidate edges, and (3) *text*, which may be used to label either nodes or edges. We identify the diagram elements by parsing XML representations of diagrams or using COM APIs. For each diagram element, we extract information about its coordinates, dimensions, text, geometry, and group (if any) in the diagram; this information is used for node and edge discovery. All coordinates are expressed in some standard unit with respect to the origin of the page, which we consider to be the bottom, left corner.

Some properties, such as text labels and arrowheads for lines, may be readily available if they are linked to drawing shapes. If not, they need to be inferred later as part of flow element discovery. For a drawing shape that is taken from a stencil, the geometry is identified by the shape name as per the stencil. For a manually created shape, the geometry is encoded by the different types of lines (*e.g.,* straight lines, elliptical arcs, bezier arcs) used to form the shape. The *group* feature in diagramming tools, which lets a user tie related diagram elements together, often conveys important structural cues. Therefore, for each diagram element, we identify elements, if any, that are grouped alongwith it.

Flow Element Discovery The second step of structure inference discovers flow elements: nodes, candidate edges, and associations of text with nodes and candidate edges. It uses the following heuristics to perform the element discovery.

1. Ignore elements, such as bounding boxes, title bars, legends, etc., that are close to page boundaries and do not have possible connections.
2. Trace undirected connected lines that form closed paths to form new nodes.
3. Recognize arrowheads that are depicted as separate shapes and associate them with nearby lines.
4. Identify shapes that are near possible connectors as nodes.

[9] An XML serialization of the annotated flow graph can be easily transformed to conform to specific XML process-modeling schemas, such as BPMN 2.0 and the WBM XML schema.

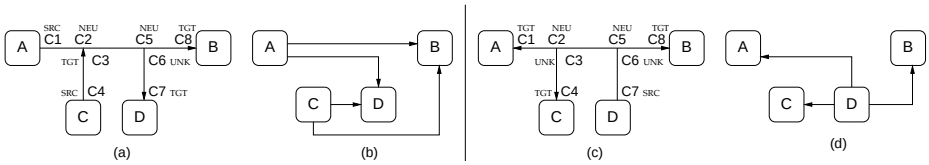

Fig. 2. Examples to illustrate flow-edge inference.

5. Lines whose endpoints lie near nodes or near other lines are recursively identified to be part of candidate edges.
6. Identify as a node, shape groups that have connectors emanating from the group boundaries but that do not have the possibility of a flow within them.
7. *Label association:* Text in unbordered shapes are taken as labels of the nodes or edges with which they are grouped, or of their nearest node or edge.

Flow-Edge Inference The key step of structural inference is the identification of directed connections between the nodes: that is, we need to list possible source and target nodes for each candidate edge discovered from the diagram elements.

The edge-inference algorithm uses the concept of a connection point. A *connection point* is the notional point of connection at which the endpoint of a line connects to a node or an intermediate point on another line. The intersection of the endpoint of a line l with an intermediate point on another line l_x generates a pair of connection points—one at the end of l and the other at the intersection point on l_x. To illustrate, consider the flow graph shown in Figure 2(a), which has eight connection points (labeled $C1$–$C8$). The line (A, B) has connection points $C1$ and $C8$ at its endpoints; it also has two connection points, labeled $C2$ and $C5$, at its intersections with the lines from C and D. Each intersection generates two connection points: one on line (A, B) and the other at the endpoint of the line meeting (A, B). Thus, each line and each node has a set of connection points.

A connection point has five attributes:

1. An end point C_{ep} to mark the coordinates
2. An associated line C_{line} on which it is either an endpoint or an intermediate intersection point
3. A boolean value C_{isEnd} indicating whether it is an endpoint or an intermediate intersection point
4. An associated node or connection point C_{conn} on which it is an endpoint
5. A direction C_{dir}, which can be SRC (source), TGT (target), NEU (neutral), or UNK (unknown)

The direction of a connection point C is found using the following rules

1. If C lies at the junction of a node and a line:
   C_{dir} = TGT, if there is an arrowhead at C_{ep}
   C_{dir} = SRC, otherwise
2. If C is at the junction of the endpoint of a line l and an intermediate point on line l_x:
   C_{dir} = NEU, if C lies on l_x (*i.e.*, C_{isEnd} = `false`)
   C_{dir} = TGT, if C lies on l and there is an arrowhead at C_{ep}
   C_{dir} = UNK, if C lies on l and there is no arrowhead at C_{ep}

```
algorithm InferEdges
 input   W_n, W_e   nodes and candidate edges
 output  edgeList   inferred flow edges
begin
     // create CP for line-node junctions
  1. foreach l ∈ W_e and endpoint ep do
  2.    foreach node n ∈ W_n do
  3.       if n is nearby ep then
  4.          create connection point C:
                   C_ep = ep, C_line = l, C_conn = n
                   C_isEnd = true, C_dir = SRC | TGT
  5.          add C to CP sets for l and n
     // create CP for intersection of lines
  6. foreach l ∈ W_e and endpoint ep do
  7.    foreach l_x ∈ W_e do
  8.       if l is nearby ep then
  9.          create connection point C_x:
                   C_x,ep = ep, C_x,line = l_x, C_x,conn = C
                   C_x,isEnd = false, C_x,dir = NEU
 10.          add C_x to CP set for l_x
 11.          create connection point C:
                   C_ep = ep, C_line = l, C_conn = C_x
                   C_isEnd = true, C_dir = TGT | UNK
 12.          add C to CP set for l
     // connect nodes
 13. foreach node n ∈ W_n do
 14.    foreach conn. point C of n do
 15.       nodeSet = ∅; visit(C) = true
 16.       propagateDirection(C_dir, C_line, nodeSet)
 17.       foreach n' ∈ nodeSet do
 18.          if C_dir = SRC then
 19.             add (n, n') to edgeList
 20.          else add (n', n) to edgeList
end

procedure propagateDirection(dir, l, ns)
begin
 21. foreach conn. point C of l do
 22.    if visit(C) then continue
 23.    visit(C) = true
 24.    if C_isEnd ∧ match(C_dir, dir) then
 25.       continue
 26.    if ¬C_isEnd ∧ ¬match(C_dir, dir) then
 27.       continue
 28.    if C_conn instanceof Node then
 29.       add C_conn to ns
 30.    else if C_conn instanceof ConnPt then
 31.       C_x = C_conn; l_x = C_x,line
 32.       if ¬match(C_x,dir, dir) then continue
 33.       propagateDirection(dir, l_x, ns)
end

function match(interDir, destDir)
begin
 34. if interDir = destDir then
 35.    return true
 36. else if interDir = NEU ∨ UNK then
 37.    return true
 38. return false
end
```

Fig. 3. The edge-inference algorithm.

Figure 3 presents the edge-inference algorithm: `InferEdges`. The algorithm takes as inputs the set of nodes and candidate edges in the flow graph, and returns as output the inferred edges. Intuitively, the algorithm traverses the flow graph, starting at a connection point on a node, and identifies reachable nodes such that the directions encountered during the traversal are consistent.

Lines 1–5 of the algorithm create connection points for the junctions of lines and nodes. For each line l and each endpoint ep of l, the algorithm iterates over the nodes (lines 1–2). It uses thresholds for *nearness*[10] to determine whether a node n could be connected to ep (line 3). It then creates a connection point C, with the appropriate attributes, and adds C to the set of connection points for l and n. The direction of C is set to SRC or TGT depending on whether an arrowhead occurs at ep.

Similarly, lines 6–12 of `InferEdges` create connection points for the intersection of two lines l and l_x. In this case, two connection points (C_x and C) are created. C_x is created for the intermediate intersecting point on l_x, whereas C is created for the endpoint of l.

After creating the connection points, `InferEdges` connects the nodes by traversing the flow graph starting at each node n and following each connection point on n (lines 13–16). After the traversal for a connection point is complete, appro-

[10] Such nearness bounds are set relative to the width and height over which diagram elements span in a page and are thus scale invariant.

priate edges are created between n and the nodes reached during the traversal (lines 17–20).

Procedure `propagateDirection` traverses the flow graph, along paths in which the directions are consistent, and identifies the reached nodes. Given a line l and direction dir, the procedure processes each connection point C of l that has not been previously visited (lines 21–22). If C is an endpoint and C_{dir} and dir match—*i.e.,* either both are SRC or both are TGT—the procedure abandons the traversal as the path does not represent a valid edge (lines 24–25). If C is an intermediate point, the traversal terminates if both of the following conditions hold: (1) C_{dir} does not match dir and (2) C_{dir} is neither NEU nor UNK (lines 26–27). If C occurs at a node, the procedure has found an edge; therefore, it adds the node to the set of nodes (lines 28–29). Alternatively, if C occurs at an intermediate intersection, the algorithm continues traversing if the directions are consistent (lines 30–33).

To illustrate the steps of the algorithm, consider the traversal performed by `InferEdges`, starting at connection point $C1$ in Figure 2(a). `InferEdges` processes each connection point on line (A, B) (line 21 of `propagateDirection`). We consider the processing of $C2$ and $C5$. In both cases: the condition at line 24 evaluates `false` because $C2$ and $C5$ do not occur at endpoints; the condition at line 26 evaluates false because their dir = NEU (which causes line 36 to evaluate `true`); the condition on line 28 evaluates `false` as well; but line 30 evaluates `true`. In the case of $C5$, the condition on line 32 evaluates to `false` and `propagateDirection` is invoked recursively (line 33) to continue the traversal toward node D. In this invocation, the algorithm reaches $C7$, whose direction is TGT (*i.e.,* does not match $C1_{dir}$); thus, a valid edge is detected. It therefore, adds D to the node set (at line 29), and later adds (A, D) to the edge list (at line 19). However, the line towards C is not traversed since $C3_{dir}$ = TGT causes the condition on line 32 to evaluate to `true` during the processing of $C2$. Note that for the traversal starting at $C7$, `InferEdges` reaches $C6$, where $isEnd$ = `true` and direction is UNK which causes line 24 to evaluate true, and the traversal stops. This ensures that the edge (A,D) does not get added twice in the edge set. The final set of edges for the flow graph is shown in part(b) of Fig. 2. Parts (c) and (d) of the figure illustrate a different example in which the line from D is split into three edges, one each to nodes A, B and C.

In this manner, lines get converted to edges that have sources and targets. For each edge, we also track the text label, color, and line type. For an edge E, these attributes are obtained by aggregating the attributes of all lines that constitute E.

Cross Reference Resolution: We identify nodes that have the same text label and exist in different pages of a drawing to be *cross references* if either their indegree or their outdegree is zero. We reduce the flow graph by fusing such nodes and merging the incoming and outgoing edges on the fused nodes.

3.2 Semantic Interpretation of Flow Elements

Phase 1 of our approach infers a well-formed graph, which has none of the ambiguities present in the flow diagram from which it was inferred. Next, Phase 2

Table 1. Features used for disambiguating node semantics.

Category	Features	Comments
Relational	No. of incoming edges (*indegree*), No. of outgoing edges (*outdegree*), No. of nodes contained within (*numContains*), Whether it is contained in another node (*isContained*)	Can discriminate amongst many entities irrespective of local styles in diagrams. For example, *Indegree* and *outdegree* can easily distinguish between start, end and intermediate events; Non-zero *numContains* may strongly indicate presence of a *swimlane* or a *group*.
Geometric	Shape name in stencil, No. of vertical lines, No. of horizontal lines, No. of arcs, line style, width, height	Can provide highly accurate insight, if data sets follow templates very rigorously. Such features can work well with small sets of process diagrams with uniform styles per entity.
Textual	No. of cue words for every entity in label for the node and labels for incident edges	Humans comprehend text to determine semantics in highly ambiguous scenarios. For example, text in outgoing edges from *gateways* is often 'yes'/'no'/'y'/'n', text in *activities* typically starts with strong verbs, 'report' and 'e-mail' are common in *data objects*.

associates semantics with the nodes the edges present in the graph, based on similarity of the nodes and edges. Semantic similarity of nodes and edges quite often follows from similarity in their geometry, relational attributes, and textual content. We formulate semantic disambiguation as a pattern-classification problem [5]. Using a representative corpus of business process diagrams, we train a classifier to learn patterns in features of flow elements that indicate the class of process semantic of an element. For semantic interpretation of new diagrams, we extract pertinent features for each flow element and feed them into the trained classifier, which detects learnt patterns to decide process semantics. We discuss both supervised and unsupervised schemes for learning that may be used depending upon whether a corpus of diagrams is available for training or not.

Feature Extraction We attempt to mimic human reasoning used in recognizing process semantics from a diagram. Humans usually analyze a range of visual and textual cues to understand diagram semantics. We abstract such cues as symbolic or numeric features that can be acted upon by standard classifiers.

Table 1 lists a set of features for nodes, grouped into three categories: relational, geometric, and textual. For each group, the table lists examples of features (column 2), and discusses how the features are indicative of process semantics in nodes (column 3). (A similar list can be formulated for edges; for space constraints, we do not present it.) Relational features such as indegree and outdegree can be obtained directly from the extracted flow graphs, whereas geometric and textual features are aggregated from attributes of the diagram elements involved in the flow. For each process entity, a set of *cue words* that characterize expressions in the labels for the entity is taken to be a textual feature. For example, interrogative words (*e.g.,* "Whether," "Is," "Does") in the text associated with a node label indicate the possibility of a *gateway*; similarly, text starting with strong verbs (*e.g.,* "Create," "Process") indicate an *activity*. If training data is available, we can perform text classification on labels to identify such representative words for each target entity; otherwise, these word lists have to be created with inputs from human experts.

Supervised Learning If we have a set of diagrams for which the corresponding correct process models are known, we can train a classifier, in a supervised

manner, to learn classification rules from the labeled instances. The learned rules can be used to infer the semantics of new diagrams. A decision tree learner [14] can formulate a decision task as a sequence of logical or binary operations from a series of examples. It is a set of if-then-else like classification rules over the feature set, which can be easily interpreted (also edited if required) by data mining practitioners. A Naïve Bayes classifier [11], after training on a labeled dataset, can assign probabilities that an observation (flow element) belongs in each class (process entity). Neural networks [13] consist of layers of interconnected nodes where each layer produces a non-linear function of its input, thus enabling the modeling of very general functions. Our empirical study evaluates different classifiers for their efficacy in choosing process semantics for flow elements.

Unsupervised Learning Clustering is a popular and effective technique used in data mining for discovering, without any human supervision, patterns from large amounts of data [12]. We cluster flow elements based on their geometrical, relational, and textual features, and hypothesize that elements with identical process semantics get grouped into the same cluster. Next, we consider the cluster assignments as class labels for the flow elements and train a classifier. The classifier trained in this manner can then perform semantic disambiguation—eliminating the need for performing clustering on each new diagram.

We define a measure of similarity (or distance) such that flow elements in the same cluster exhibit greater similarity in semantics amongst them than with items in any other cluster. We compute similarity for each feature category: relational (sim_r), geometric (sim_g), and textual (sim_t). We use the euclidean distance to compute similarity between numeric attributes, a boolean measure (1 for match, 0 for mismatch) for attributes that can be enumerated (*e.g.,* shape, name, color), and string edit distances (*e.g.,* Levenshtein, Monge Elkan, Jaro) [4] for text. The aggregate feature-based similarity of two flow elements, $f_i and f_j$, is obtained using a linear combination of the three similarity components:

$$sim(f_i, f_j) = w_r \times sim_r(f_i, f_j) + w_g \times sim_g(f_i, f_j) + w_t \times sim_t(f_i, f_j)$$

The weights for the different components can be set either using domain knowledge about the importance of different aspects of the similarity measure, or, alternatively, by validation over a set of labeled training instances (if available). Given the aggregated similarity measure, there are various clustering approaches, such as agglomerative, divisive, and k-means, for iteratively improving the clustering goodness. The choice of the number of clusters may be governed by a knowledge of the number of entities in the target meta-model. After clustering is run, the user can examine a few exemplars flow elements in each cluster to decide a process semantic for the cluster. Then, the semantic classification (thus obtained via clustering) of flow elements from the training corpus is used to train a classifier, and semantic interpretation proceeds as in the supervised case. Our empirical studies show that clustering on features similar to those listed in Table 1 indeed groups together elements with common process semantics, and that an unsupervised approach is almost as effective as supervised learning for recognizing certain semantics. In practice, an unsupervised approach is always more welcome because sound training data is hard to get.

Table 2. Occurrence of structural ambiguities in the dataset.

Ambiguity	Instances per File		% Files with Ambiguity
	High	Average	
Dangling Conn.	47 (100%)	3 (14%)	56
Unlinked Labels	46 (39%)	2 (3.7%)	38
Cross References	10	1	35

Table 3. Accuracy of structural inference by *i*DISCOVER and PMT.

Element	*i*DISCOVER		PMT	
	Precision	Recall	Precision	Recall
Node	96.93	95.91	70.44	86.29
Edge	93.26	90.86	63.43	59.87

4 Empirical Evaluation

We implemented our approach in a tool called *i*DISCOVER, and conducted empirical studies to evaluate its accuracy and compare it with the accuracy of a commercial tool. We organized the evaluation to report results on both aspects of model discovery: structural inference and semantic interpretation.

Experimental Setup As experimental data, we used a set of 185 Visio process diagrams created as part of real business-transformation projects. We randomly selected the diagrams from a repository of archived projects within IBM's Service Delivery practice. For comparison with *i*DISCOVER, we selected a top commercial process-modeling tool. The Visio-import capabilities of all such tools recognize diagram shapes as process elements only if the shapes come from specific Visio stencil(s) prescribed by the tools. Moreover, they cannot resolve structural ambiguities, such as dangling connectors and unlinked labels. The representative tool that we selected offers support for the largest number of Visio stencils. For proprietary reasons, we refer to the tool as PMT(Process Modeling Tool). We employed human experts to identify the true process models captured in each diagram and used these to compute the accuracies of *i*DISCOVER and PMT.

We use *precision* and *recall* to measure accuracy. In the following equations, *Actual* is the set of (manually identified) correct interpretations and *Retrieved* is the set of (automatically inferred) interpretations by a tool.

$$Precision(P) = \frac{|Actual \cap Retrieved|}{|Retrieved|}, Recall(R) = \frac{|Actual \cap Retrieved|}{|Actual|}$$

4.1 Study 1: Structure Inference

Goals and Method The goal of the first study was to compare the accuracies of the structural inference performed by *i*DISCOVER and PMT. We also evaluated how the structural ambiguities present in the dataset impact structure inference in both cases. We ran *i*DISCOVER's structure extractor to infer XML flow graphs from the Visio diagrams and compared them with actual process models, manually identified by human experts, to measure accuracy. First, we matched the text labels for nodes in *Retrieved* with those in *Actual*. Next, we traced the recovered edges between matched nodes and checked if equivalent edges were present in *Actual* models. Finally, we computed precision and recall for both node and edge detection. Similarly, we validated the flow graphs produced by PMT against the actual process models. *i*DISCOVER also reports the number of dangling connections, unlinked labels, and cross references found in the input.

Results and Analysis Table 2 lists, for each type of structural ambiguity, the highest and average number of instances found in the files in the dataset. It also reports the percentage of edges that are dangling, percentage of nodes and edges that have unlinked text labels and percentage of files that have at least one instance of the ambiguity. Over half of the files contain dangling connectors, whereas unlinked labels and cross references occur in over a third of the files in the dataset. The fact that 100% of connectors in some files were left dangling suggests that certain users may be completely unaware of notions such as proper gluing of connectors. The data indicate that structural ambiguities can occur frequently in practice; therefore, to be useful, an automated inference technique must handle them effectively.

Table 3 shows the average precision and recall of node and edge detection across the dataset. The data illustrate that *i*DISCOVER performs much better than PMT. For node inference, *i*DISCOVER had $\approx 27\%$ higher precision and $\approx 9\%$ higher recall. For edge inference, the performance of *i*DISCOVER was even better: it achieved $\approx 30\%$ improvement in both precision and recall. We observed far greater correlation of the number of dangling connectors with the edge recall of PMT (Pearson's coefficient, $\rho = -0.48$) than that with the edge recall of *i*DISCOVER ($\rho = -0.08$). This clearly suggests that while *i*DISCOVER successfully resolves dangling connectors, PMT's edge-inferencing capability is adversely affected by their presence.

Discussion Overall, the study shows that *i*DISCOVER consistently performs better than PMT in terms of both precision and recall. The data also indicate that structural ambiguities, which complicate automated structure inference, can occur frequently in practice; therefore, an approach, such as ours, that effectively deals with such ambiguities can be valuable.

We investigated the reasons for errors in node detection in both tools, and observed the following reasons: (1) imprecise resolution of unlinked labels in ambiguous scenarios where nearness does not suffice; (2) failure to recognize some shapes when a group of diagram shapes represent a single node. An edge is taken to be accurate only if its source and target nodes are correctly inferred. Thus, although our edge-inference algorithm is highly accurate and identifies edges precisely, the overall precision of edge inference suffers from inaccuracies in node detection.

4.2 Study 2: Semantic Interpretation

Goals and Method The goals of the second study were to (1) evaluate the effectiveness of different pattern-classification techniques in assigning semantics to flow elements, and (2) compare the effectiveness of these techniques with that of PMT. Specifically, we evaluated three supervised classification techniques—C4.5 decision tree [14], Naïve Bayes, and Multi-layer perceptron (MLP) neural network [13]—and an unsupervised clustering technique.

We asked human experts to create BPMN[11] models for the 185 diagrams in our dataset. To compute accuracy, we compared the semantic interpretations

[11] `http://www.omg.org/spec/BPMN`

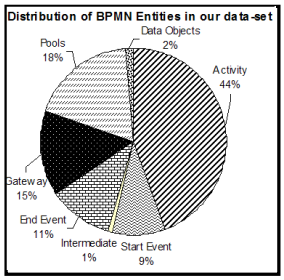

Fig. 4. Distribution of BPMN entities.

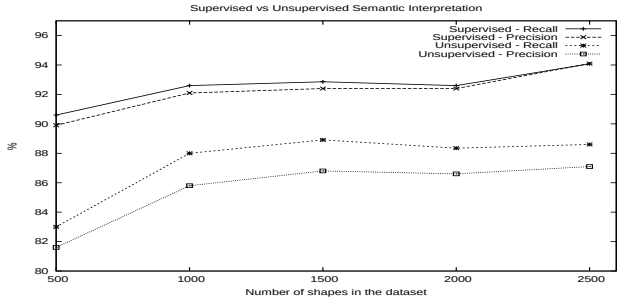

Fig. 5. Effects of varying the dataset size.

of the pattern classifiers and PMT against those made by the experts.[12] We chose to evaluate interpretation of node semantics only because BPMN flow-edge semantics can be resolved unambiguously by applying simple rules.[13]

To construct the training dataset for the supervised classifiers, we compared the nodes in the flow graphs extracted by *i*DISCOVER with the nodes in the expert-created BPMN models: we obtained 2943 matches, which formed the training set. For each node, we aggregated different features listed in Table 1. The training set contained seven classes of BPMN entities labeled by the experts; Figure 4 shows the distribution of these entities. Therefore, for the three supervised classifiers, we set up a 7-class classification problem using the *Weka* toolkit.[14] Further, to study the effects of the training-set size on the classification accuracy, we experimented with different sizes of the training set.

In the unsupervised case, we performed classification via clustering. We ran k-means clustering using the similarity measures discussed in Section 3.2. We computed pairwise similarity between the nodes and fed the similarity matrix to a graph clusterer, which produced clusters representative of process semantics. Then, a user decided the class of BPMN semantics (out of the seven classes) for each cluster by studying a few exemplars in each cluster. Next, for each node, we compared the process semantic assigned to its cluster and that assigned by human experts to compute precision and recall. We also studied the classes of semantics that come up in new clusters as we increase k in different runs of the k-means clusterer. Finally, as in the case for supervised classifiers, we investigated the effects of varying the dataset size on clustering results.

Results and Analysis To quantify the degree of semantic ambiguities present in our dataset, we measured under-specification and over-specification. We found that 79.8% of the nodes were touched by under-specification in the sense that they were represented by shape types whose instances referred to at least two forms of semantics. We also found each class of BPMN entity to be over-specified: *i.e.,* it was represented by more than one shape in the dataset.

[12] Note that an expert's interpretation may differ from the actual intent of the designer in highly ambiguous scenarios. Nevertheless, we consider the expert's judgment to indicate true semantics.

[13] An edge that cuts across two *pools* is a *message flow*; an edge that exists between two nodes in the same *pool* is a *sequence flow*; an edge whose source or the target is an *artifact* is an *association*

[14] `http://www.cs.waikato.ac.nz/ml/weka/`

Table 4. 10-fold cross-validation results of semantic interpretation by *i*DISCOVER and PMT.

Class	Supervised		Unsuper.		PMT	
	P	R	P	R	P	R
Activity	92.6	91.0	89.8	88.5	66.1	84.4
Start Event	82.1	91.0	77.4	84.9	18.4	24.2
Intermediate	14.3	6.7	0	0	0	0
End Event	83.6	84.7	71.7	87.9	26.5	35.1
Gateway	96.7	97.0	90.0	97.9	93.3	92.0
Pool	100	100	99.1	92.4	76.6	87.7
Data Object	56.5	57.8	0	0	0	0
Overall	91.9	92.1	87.1	88.4	60.2	73.7

Table 5. 10-fold cross-validation results for the three supervised classifiers.

Class	C4.5		Naïve Bayes		MLP	
	P	R	P	R	P	R
Activity	92.6	91.0	90.5	81.5	90.9	91.5
Start Event	82.1	91.0	81	78.6	84.4	83.6
Intermediate	14.3	6.7	14.3	53.3	75	20
End Event	83.6	84.7	75.9	83.8	78.5	84.3
Gateway	96.7	97.0	91.2	96.2	96.2	97
Pool	100	100	100	92.7	99.6	99.8
Data Object	56.5	57.8	28.4	73.3	60	40
Overall	91.9	92.1	88.9	85.6	91.2	91.4

Table 4 reports 10-fold cross-validation results (of precision and recall) over the training dataset for the best-performing supervised classifier and the unsupervised classifier; it also presents the precision and recall results for PMT. We find that overall precision and recall of both pattern-classification approaches are $\approx 90\%$ (within 5% of each other); and they are over $\approx 20\%$ higher than that of PMT. We observe that the classification approaches fare well in recognizing most BPMN entities present in the dataset except for intermediate events and data objects, which together constituted only 3% of the dataset. Top discriminating features were noted to be: *isContained*, *indegree*, *outdegree*, and *shape name*. PMT could detect only gateways and pools with high precision.

Table 5 reports the accuracy results for the three supervised classifiers. We find that C4.5 decision tree performs the best, MLP neural network is almost as effective, and Naïve Bayes is less effective by a few percentage points. Figure 5 evaluates both supervised and unsupervised approaches on a constant test dataset of 443 nodes (not part of the training data), when the training dataset size is varied from 500 through 2500. We find that the variation in results between the least and the greatest sizes of the training set is as low as $\approx 4\%$. Moreover, the curves show less than 2% deviation as we increase the size beyond 1000 nodes. For a training set with a balanced distribution of entities, the results for the fringe entities improved considerably but the overall results dropped by a few points.[15] In the unsupervised case, for $k = 5$, we obtained clusters that represented activity, pool, gateway, start event, and end event. However, on increasing k beyond 5, we observed that the new clusters represented new process semantics, such as join/merge, fused merge, branch points, etc., but they did not isolate data objects or intermediate events, which are part of our target set.

Discussion Figure 5 indicates that the classification approach could work just as well with only a third of our current dataset size. However, for such approaches to succeed, the input set should have a balanced distribution of target model entities. We missed two BPMN classes because they existed only as fringe entities, but we were able to recover them from a balanced dataset.

The results show that clustering is indeed effective in grouping together related process semantics. The relational features and shape name were found to be the most discriminative features. The set of cues words for gateways emerged

[15] Intermediate event (P:64%, R:44%); Data Objects (P:80%, R:90%); Overall (P:88%, R:88%). Reference [17] provides the details of this experiment and illustration of *i*DISCOVER and PMT outputs.

as the top textual feature. We realize that textual features need to be modeled more effectively, *e.g.,* by capturing the relative position of words within a sentence, performing parts-of-speech tagging, and using WordNets.

5 Related Work

Although informal expressions of business process designs are extremely common, (to the best of our knowledge) there is no existing research that addresses the problem of automatically understanding process diagrams. Moreover, there have been no studies of the challenges that arise in interpreting diagrams created using stencil-based tools, such as Visio, Powerpoint, and Dia.

However, there exists a large body of work in the area of understanding line drawings and hand sketches (*e.g.,* [1, 3, 9, 15]). The primary problem addressed by sketch recognition is that of identifying various shapes present in a diagram; semantic interpretation follows directly from a fixed mapping between the geometry of source shapes and the semantics in the target metamodel [3, 9]. In contrast, the primary challenge in inferring formal process models from informal flow diagrams is the detection of higher-level semantics—the basic shapes are readily parsable from the diagram format. Thus, unlike the research in sketch recognition, our work focuses on semantic interpretation of informal diagrams.

Gross [8] presents an approach for sketch interpretation, which consists of: low-level glyph recognition, detection of spatial relations among glyphs, and assembly of glyphs into high-level configurations. Although the approach is organized in a similar manner as ours, the high-level structural patterns (*e.g.,* tree and polyline) that they discover do not have any semantic significance. Moreover, polyline recognizers in these sketching tools can detect only pre-specified patterns of inter-linked lines and are not as general as our edge-inference algorithm. Barbu et al. [2] apply frequent graph discovery to symbol recognition from line drawings: their approach extracts a feature vector to represent nodes and applies an unsupervised hierarchical clustering algorithm. Unlike our approach, they focus on inferring graphic symbols only and not semantic classes. Also, we consider a richer set of features that includes higher-level relational attributes, such as indegree and outdegree.

Approaches in visual language theory (*e.g.,* [6, 7, 16]) rely on upfront codification of the production rules for interpretation. Such codification becomes hard in the presence of local styles of diagramming (individual, group, or organization-based). Our work can be viewed as the first step toward learning such grammars. Wittenburg [16] shows that relational grammars are required to parse process modeling constructs. Our current work attempts semantic classification at the level of a single node. Future work can attempt learning more complex relational patterns involving a sequence of nodes (*e.g.,* loop, fork, and merge).

6 Conclusions and Future Work

We presented a comprehensive approach for discovering formal process models from informal process diagrams that contain structural and semantic ambiguities. We presented techniques for resolving the structural ambiguities to extract

precisely a flow graph underlying a process diagram. We also showed that standard pattern-classification techniques can be successfully employed in interpreting process semantics if the features space is carefully modelled. Our approach mimics human reasoning used in recognizing diagram semantics: it models relational, geometric, and textual attributes of flow elements as features during pattern classification, instead of simply relying on shape geometry, as existing tools do. Our empirical results showed that unsupervised clustering can almost match supervised techniques in performance; thus, such an approach can work in practical scenarios where sound training data may not be available. Our tool, *i*DISCOVER, has better precision and recall, in both structural inference as well as semantic interpretation, than the state-of-the-art Visio import capabilities.

Future work can strive to improve the precision and recall of semantic interpretation through more efficient modeling of textual cues because text is the only reliable feature in highly ambiguous scenarios. Label association can also be perfected by tracking spatial patterns of label assignments that emerge due to local styles followed by designers. Finally, future research can investigate the identification of higher-level relations (block structures) between model entities (*e.g.,* sub-process, loop, and fork-merge) and extend the approach to other varieties of flow diagrams.

Acknowledgements We would like to thank Indrajit Bhattacharya, David Marston and Juhnyoung Lee for their valuable inputs and many useful discussions on the topic. We thank Vanitha Nachimuthu and Mary Joshua for their efforts in preparing the training data.

References

1. A. Apte and T. Kimura. Recognizing multistroke geometric shapes: an experimental evaluation. In *Proc. of ACM UIST*, pages 121–128, 1993.
2. E. Barbu et al. Frequent graph discovery: Application to line drawing document images. *Electronic Letters on Computer Vision and Image Analysis*, 2005.
3. Q. Chen, J. Grundy, and J. Hosking. SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. *Software Focus*, 38(9):961–994, 2007.
4. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *IIWeb*, pages 73–78, 2003.
5. R. Duda, P. Hart, and D. Stork. Pattern Classification.
6. R. P. Futrelle et al. Understanding diagrams in technical documents. *Computer*, 1992.
7. E. Golin and S. Reiss. The specification of visual language syntax. In *IEEE Workshop on Visual Languages, 1989.*, pages 105–110, 1989.
8. M. Gross. Recognizing and interpreting diagrams in design. In *Workshop on Advanced visual interfaces*, pages 88–94, 1994.
9. T. Hammond. Tahuti: a geometrical sketch recognition system for uml class diagrams. In *Intl. Conf. on Computer Graphics and Interactive Techniques*, 2006.
10. J. Iivari. Why are CASE tools not used? 1996.
11. A. Jain, R. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, pages 4–37, 2000.
12. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
13. S. Pal and S. Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, 1992.
14. J. Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann, 2003.
15. D. Rubine. Specifying gestures by example. In *Proc. of the Conf. on Computer graphics and interactive techniques*, pages 329–337, 1991.
16. K. Wittenburg and L. Weitzman. Relational grammars: Theory and practice in a visual language interface for process modeling. *Visual language* , 1998.
17. D. Mukherjee, P. Dhoolia, S. Sinha, A. J. Rembd M. G. Nanda. From Informal Process Diagrams To Formal Process Models. *IBM Technical Report No. RI09014*, 2010, `http://domino.research.ibm.com/library/cyberdig.nsf/index.html`