JMLR: Workshop and Conference Proceedings 25:443–458, 2012 Asian Conference on Machine Learning

Two-way Parallel Class Expression Learning

An C. Tran A.C.TRAN@MASSEY.AC.NZ
Jens Dietrich J.B.DIETRICH@MASSEY.AC.NZ
Hans W. Guesgen H.W.GUESGEN@MASSEY.AC.NZ
Stephen Marsland S.R.MARSLAND@MASSEY.AC.NZ
School of Engineering and Advanced Technology
Massey University
Palmerston North 4442, New Zealand

Editor: Steven C.H. Hoi and Wray Buntine

Abstract

In machine learning, we often encounter datasets that can be described using simple rules and regular exception patterns describing situations where those rules do not apply. In this paper, we propose a two-way parallel class expression learning algorithm that is suitable for this kind of problem. This is a top-down refinement-based class expression learning algorithm for Description Logic (DL). It is distinguished from similar DL learning algorithms in the way it uses the concepts generated by the refinement operator. In our approach, we unify the computation of concepts describing positive and negative examples, but we maintain them separately, and combine them at the end. By doing so, we can avoid the use of negation in the refinement without any loss of generality. Evaluation shows that our approach can reduce the search space significantly, and therefore the learning time is reduced. Our implementation is based on the DL-Learner framework and we inherit the Parallel Class Expression Learning (ParCEL) algorithm design for parallelisation.

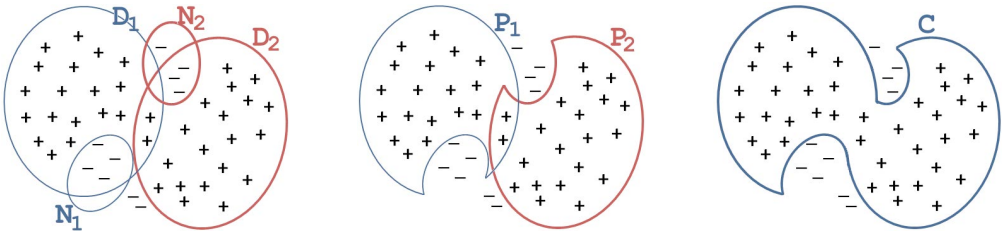
Keywords: description logic, class expression learning, parallel learning, exception

1. Introduction

Description logic (Baader, 2003) is a popular knowledge representation language. A family of description-logic-based languages, the Ontology Web Language (OWL), has been proposed by W3C as the standard knowledge representation languages for the Semantic Web (McGuinness et al., 2004).

The original problem that motivates our research in DL learning is the detection of abnormal behaviours in smart homes, where the elderly are monitored by automated systems that can send alerts to a carer if an abnormal behaviour is detected (Tran et al., 2010). In our research, we are interested in using description logic learning to learn the normal activity patterns and then to detect abnormal behaviours as variations to normality.

In this scenario, it is often the case that a normal behaviour can consist of a general rule and then a set of exceptions. For example, an elderly person may usually go to the library at 9am on Mondays. However, this pattern might be broken by particular circumstances such as rain, or public holidays. Thus, rather than 'The person goes to the library on Mondays at 9am', the rule becomes 'The person goes to the library on Mondays at 9am so long as it is not raining and it is not a public holiday'. A similar well-known example



(*a*) The target concept can be constituted from simple descriptions D_1, D_2, N_1 and N_2: $(D_1 \sqcap \neg N_1) \sqcup (D_2 \sqcap \neg N_2)$

(*b*) ParCEL learnt concepts: more complex descriptions $P_1 = D_1 \sqcap \neg N_1$ and $P_2 = D_2 \sqcap \neg N_2$ describe the dataset concisely

(*c*) CELOE learnt concept: a single complex description $C = (D_1 \sqcap \neg N_1) \sqcup (D_2 \sqcap \neg N_2)$ is required

Figure 1: Exceptions in learning pattern

is the relation between *birds* and *fly*: all birds can fly except penguins. We will use the term exception in this sense: a set of examples not covered by simple concepts that describe the vast majority of examples correctly. Negative examples in smart home scenario are the behaviours that are not expected to happen, such as cooking at midnight. There may be several ways to generate the negative examples: explicitly defined, using negation as failure, or using classification feedback.

Most current DL learning algorithms (e.g., Lisi and Malerba, 2003; Fanizzi et al., 2008; Lehmann and Hitzler, 2010; Tran et al., 2012) focus only on the definitions of positive examples. Learning starts from a very general concept, usually the `TOP` concept in DL. Then, it uses subclasses (specialisation), conjunction (intersection) and the combination of conjunction and negation (subtraction) to remove the negative examples from the potential concepts. Negative examples are used as constraints to qualify whether a concept is acceptable.

This approach suits many learning problems and has been used successfully in experiments (Lehmann and Hitzler, 2010; Tran et al., 2012). However, this strategy does not deal very well with learning problems that have exceptions in the normal patterns. For example, given a learning problem with a set of positive (+) and negative (–) examples, the concepts D_1, D_2, N_1, N_2 and their coverage (Figure 1(*a*)), CELOE tries to find the single concept $C = (D_1 \sqcap \neg N_1) \sqcup (D_2 \sqcap \neg N_2)$. On the other hand, ParCEL (Tran et al., 2012) will generate the following simpler concepts: $P_1 = D_1 \sqcap \neg N_1$ and $P_2 = D_2 \sqcap \neg N_2$. Those concepts are visualised in Figures 1(*c*) and 1(*b*), respectively. If all of the concepts D_1, D_2, N_1 and N_2 have the same length of 3, the length of the longest concepts generated by CELOE and ParCEL are 16 (4 concepts of length 3 plus 4 operators) and 8 (2 concepts of length 3 plus 2 operators). If a concept is of length 16 then it must occur at depth at least 16 in the search tree. Thus, CELOE will only be able to identify this concept after searching the previous 15 levels of the search tree. However, our algorithm can combine (potentially very short) concepts and so define relatively complex concepts based on much smaller depth traversals.

This is particularly useful where removing negative examples leads to shorter definitions of positive examples.

In this paper, we describe a *Two-way Parallel Class Expression Learning* (or *PARallel Class Expression Learning with Exception* – ParCEL-Ex) algorithm that employs the definitions of both positive and negative examples to find the target concept. The negative example definitions are used to remove the negative examples from the concepts in the search tree to create simple definitions for positive examples.

Consequently, concepts in the search tree are used more effectively. It helps to reduce the search space of the learning problem and thus the learning time is reduced. This approach is best suited to learning problems with exceptions as described previously, especially when the exceptions have complex characteristics.

2. Related Work

Learning in description logic is basically a search problem in which the search tree is often dynamically generated by a refinement operator. Generally, there are two basic approaches for DL learning: top-down and bottom-up. In the top-down approach, the learning algorithm starts from the most general concept of the concept hierarchy and uses a downward refinement operator to specialise the concepts until the target concept is found (Lehmann, 2010; Rouveirol and Ventos, 2000). The search tree expansion is directed by a search heuristic that mainly replies upon the accuracy (a combination of correctness and completeness) of the concepts.

The bottom-up approach instead uses an upward refinement operator. This approach is used widely in inductive logic programming for first order logic (Muggleton and Buntine, 1992; Ade et al., 1995), but it is rarely used for description logic because while the downward refinement operator can use the concept hierarchy as an effective method for the specialisation, it cannot be used in the upward refinement. An exception is the upward refinement operator proposed by Baader et al., 1999. The refinement is to compute the least subsumer of a given number of concepts, i.e. the generalisation of the given concepts. In this approach, concepts are represented in the form of description trees. The generalisation of the given concepts is essentially the product of description trees under particular rules called *homomorphisms*. An early ILP system that used the bottom-up approach is LCSLearn (Cohen and Hirsh, 1994) which is a very simplistic approach that creates large concept definitions that are not truly intentional in the sense that definitions are only enumerations of the sets of individuals they define.

Our research is similar to other approaches in description logic learning that combine both bottom-up and top-down strategies such as YinYang (Iannone et al., 2007) and DL-FOIL (Fanizzi et al., 2008). Often, the top-down step uses the specialisation to find the solutions for a subset of the given examples (sub-solutions) and the bottom-up step uses the generalisation to aggregate the sub-solutions to form an overall solution.

However, our approach is different from the others in the following ways: Firstly, we use a parallel computation model. Secondly, we use both concepts that define positive and negative examples. Although negative examples are also used in Iannone et al. (2007), we use them in very different ways. In Iannone et al. (2007), negative examples are used as counterfactuals to form constraints on the generalisation. Their definitions are generated

on-demand and the generation of these definitions is separated from the generation of the positive example definitions. In our approach, the definitions of positive and negative examples are computed in the same refinement, but they are maintained independently and only combined when the combination condition is satisfied. Finally, we propose an extra combination step to make use of the negative example definitions and a reduction step to compute an optimal set of sub-solutions to be used to construct the overall solution. The above differences bring some benefits: i) the search space is reduced significantly due to the removal of negation and disjunction from the refinement, ii) as the result, in most of the datasets used in our evaluation, the learning time is improved, iii) our approach is well-suited to learning problems that have exceptions (as discussed in the introduction), particularly when the definition of exceptions is long or complicated.

3. Algorithm

Before giving a formal description of our algorithm, we introduce briefly some basic concepts. A more thorough description can be found in Baader (2003). Given a concept C and a set of positive ($\mathcal{E}^+$) and negative ($\mathcal{E}^-$) examples, we define:

$R(C)$: the set of instances covered by C.

$cp(C, \mathcal{E}^+)$: the subset of positive examples covered by C, $cp(C, \mathcal{E}^+) = R(C) \cap \mathcal{E}^+$

$cn(C, \mathcal{E}^-)$: the subset of negative examples covered by C, $cn(C, \mathcal{E}^-) = R(C) \cap \mathcal{E}^-$

Also, we will use the terms *concept*, *class expression* and *description* interchangeably.

Definition 1 (Knowledge base) *A knowledge base K in description logic is a structure $K = \{N_C, N_R, A\}$ that consists of a set of concepts N_C, a set of roles (properties) N_R and a set of assertions A including concept assertions (individuals) and role assertions.*

Definition 2 (Coverage) *Given a knowledge base K, a concept C is said to* cover *an example e if from K we can entail that e is an instance of C.*

Definition 3 (Concept learning problem) *Given a structure $(K, \mathcal{E}^+, \mathcal{E}^-)$ that consists of a knowledge base K, a set of positive examples $\mathcal{E}^+$ and a set of negative examples $\mathcal{E}^-$,* the concept learning problem *is to find a set of concept $\mathcal{C}$ that cover all positive examples and no negative examples.*

Definition 4 (Correct, complete and irrelevant concept) *A concept C is called* correct *if it covers none of the negative examples,* complete *if it covers all positive examples, and* irrelevant *if it covers neither positive examples nor negative examples.*

Definition 5 (Definition, partial definition, and counter-partial definition) *A concept is referred to as a* partial definition *if it is correct and not irrelevant, a* counter-partial definition *if it covers no positive examples and is not irrelevant, and a* definition *if it is correct and complete.*

Figure 1 demonstrates some of the above concepts: P_1, P_2 are partial definitions (correct and not irrelevant), N_1, N_2 are counter-partial definitions (not irrelevant and cover some negative examples), and C is a (complete) definition (correct and complete).

There are also some metrics to measure the amounts of correctness, completeness and accuracy of a concept C:

Definition 6 (Correctness, completeness, and accuracy)

$$correctness(C) = \frac{|\mathcal{E}^-| - |cn(C, \mathcal{E}^-)|}{|\mathcal{E}^-|}$$

$$completeness(C) = \frac{|cp(C, \mathcal{E}^+)|}{|\mathcal{E}^+|}$$

$$accuracy(C) = \frac{|cp(C, \mathcal{E}^+)| + (|\mathcal{E}^-| - |cn(C, \mathcal{E}^-)|)}{|\mathcal{E}^+ \cup \mathcal{E}^-|}$$

Basically, description logic learning is a search for a *complete and correct concept*: A concept that covers all positive examples and none of the negative examples. The search tree is constructed dynamically using a *refinement operator*. Here, we use a downward refinement operator: Given a concept C, the refinement operator ρ returns a set of concepts that are more specific than C. This is formally denoted as $\forall D \in \rho(C), D \sqsubseteq C$. The refinement operator in our algorithm does not use negation and disjunction due to the features discussed in Section 2. It can now be defined as follows:

Definition 7 (Downward refinement operator $\rho_\sqcap$) *Given a concept C and a set of atomic concepts N_C:*

- *if $C \in N_C$:*
  $\rho_\sqcap(C) = \{C' \in N_C \mid C' \sqsubseteq C \text{ and } \nexists C'' \mid C' \sqsubseteq C'' \sqsubseteq C\} \cup \{C \sqcap C' \mid C' \in \rho_\sqcap(C)\}$
- *if C is a conjunctive description $C = C_1 \sqcap ... \sqcap C_n$:*
  $\rho_\sqcap(C) = \{C' \mid C' \in \rho_\sqcap(C_i), 1 \leq i \leq n\}$
- *if $C = \forall r.D$:* $\rho_\sqcap(C) = \{\forall r.D' \mid D' \in \rho_\sqcap(D)\}$
- *if $C = \exists r.D$:* $\rho_\sqcap(C) = \{\exists r.D' \mid D' \in \rho_\sqcap(D)\}$

The refinement operator uses the subclass hierarchy and conjunction to perform the specialisation. Some other DL semantics can be used to optimise the refinement such as disjoint, range, etc. We define a refinement operator for classes and object properties only. The refinement for the data properties is not defined in Definition 7 because it depends upon the particular datatypes of the properties (e.g. integer, double) and the semantics of the datatype (e.g. date, time). In practice, implementations of refinement for datatype properties are often tailored to particular datatypes or properties.

3.1. Two-way class expression learning algorithm

Our learning algorithm combines top-down and bottom-up learning approaches. The top-down step is used to solve the sub-problems of the given learning problem and the bottom-up one is used to combine the sub-solutions into an overall solution. The top-down step is performed by the downward refinement operator, while the bottom-up step currently uses

a set coverage algorithm to choose the best partial definitions and disjunction to form the overall solution.

Algorithm 1 describes our learning algorithm. It chooses the best concepts (highest score) from the search tree and uses the Specialise algorithm (see Algorithm 2) for refinement and evaluation until the completeness of the partial definitions is sufficient. Concepts are scored using an expansion heuristic that is mainly based on the correctness of the concepts. In addition, a penalty is applied for complexity of the concepts (short expressions are preferred), and bonuses for accuracy and accuracy gained.

Algorithm 1: Two-way parallel class expression learning algorithm (ParCEL-Ex)

Input: background knowledge K, a set of positive $\mathcal{E}^+$ and negative $\mathcal{E}^-$ examples, and a noise value $\varepsilon \in [0, 1]$ (0 means no noise)

Output: a target concept C such that $|cp(C, \mathcal{E}^+)| \geq (|\mathcal{E}^+| \times \varepsilon)$ and $cn(C, \mathcal{E}^-) = \emptyset$

```
1  begin
2      initialise the search tree ST = {⊤}                    /* ⊤: TOP concept in DL */
3      cum_pdefs = ∅ (empty set)                              /* cumulative partial definitions */
4      cum_cpdefs = ∅                                         /* cumulative counter-partial definitions */
5      cum_cp = ∅                                             /* cumulative covered positive examples */
6      cum_cn = ∅                                             /* cumulative covered negative examples */
7      while |cum_cp| < (|E+| × ε) do
8          get the best concept C and remove it from ST       /* see text */
9          (pdefs, cpdefs, descriptions) = Specialise(C, E+, E-)   /* cf.algorithm 2 */
10         cum_pdefs = cum_pdefs ∪ pdefs
11         cum_cpdefs = cum_cpdefs ∪ cpdefs
12         cum_cp = cum_cp ∪ {e | e ∈ cp(P, E+), P ∈ pdefs}
13         cum_cn = cum_cn ∪ {e | e ∈ cn(P, E-), P ∈ cpdefs}
14         foreach D ∈ descriptions do
15             if (cn(D, E-) \ cum_cn) = ∅ then
                   /* combine D with counter-partial definitions if possible */
16                 candidates = Combine(D, cum_cpdefs, E-)    /* cf.algorithm 3 */
17                 new_pdef = D ⊓ ¬(⊔_{C∈candidates}(C))      /* create new partial def. */
18                 cum_pdefs = cum_pdefs ∪ new_pdef
19                 cum_cp = cum_cp ∪ cp(D, E+)
20             end
21             else
22                 ST = ST ∪ {D}
23             end
24         end
25     end
26     return Reduce(cum_pdefs)                               /* for description, see text */
27 end
```

The set of new descriptions, partial definitions and counter-partial definitions returned from the *specialisation algorithm* are used to update the corresponding data structures and the set of covered positive examples and covered negative examples in the learning algorithm. In addition, the new descriptions are combined with the counter-partial definitions to create

new partial definitions if possible. Note that the concepts that have been refined can be scheduled for further refinements.

The refinement operator in Definition 7 is infinite, but in practice each refinement step is finite, since it is only allowed to generate descriptions with a given length. For example, a concept of length N will first be refined to concepts of length $(N+1)$, and later, when it is revisited, to concepts of length $(N+2)$, etc. For the sake of simplicity, we use $\rho_\sqcap$ in the algorithms to refer to one refinement step rather than the entire refinement. This technique is used in DL-Learner and discussed in detail in Lehmann et al., 2011.

When the algorithm reaches a sufficient degree of completeness, it stops and tries to reduce the partial definitions to remove the redundancies using the Reduce function, which is essentially a set coverage algorithm: given a set of partial definitions $\mathcal{C}$ and a set of positive examples $\mathcal{E}^+$, it finds a subset $\mathcal{C}' \subseteq \mathcal{C}$ such that $\mathcal{E}^+ \subseteq \bigcup_{D \in \mathcal{C}'}(cp(D, \mathcal{E}^+))$. The solution returned by the algorithm is a disjunction of the reduced partial definitions. However, returning the result as a set of partial definitions instead may be useful in some contexts, e.g. to make the result more readable. The reduction algorithm may be tailored to meet particular requirements such as the shortest definition or the least number of partial definitions. Note that the combination of descriptions and counter-partial definitions in the above learning algorithm is one of the combination strategies implemented in our evaluation. This strategy is called an *on-the-fly* combination strategy; it gave the best performance in our evaluation. There is a brief discussion of the combination strategies in Section 3.3.

The *specialisation* and *combination algorithms* are described in Algorithms 2 and 3.

Algorithm 2: Specialisation algorithm (Specialise)

Input: a concept C, a set of positive $\mathcal{E}^+$ and negative $\mathcal{E}^-$ examples

Output: a triple of a set of partial definitions $pdefs \subseteq \rho_\sqcap(C)$; a set of counter-partial definitions $cpdefs \subseteq \rho_\sqcap(C)$; and a set of $descriptions \subseteq \rho_\sqcap(C)$ such that $\forall D \in descriptions : D$ is not irrelevant and $D \notin (pdefs \cup cpdefs)$, in which $\rho_\sqcap$ is the refinement operator defined in Definition 7.

```
1 begin
2     pdefs = {D ∈ ρ⊓(C) | cp(D, E+) ≠ ∅ ∧ cn(D, E−) = ∅}
3     cpdefs = {D ∈ ρ⊓(C) | cp(D, E+) = ∅ ∧ cn(D, E−) ≠ ∅}
4     descriptions = {D ∈ ρ⊓(C) | cp(D, E+) ≠ ∅ ∧ cn(D, E−) ≠ ∅}
5     return (pdefs, cpdefs, descriptions)
6 end
```

The specialisation performs the refinement and evaluation of the concepts given by the learning algorithm. Firstly, it refines the given concept $(\rho_\sqcap(C))$ and evaluates the result $(cp(C, \mathcal{E}^+)$ and $cn(C, \mathcal{E}^-))$. Irrelevant concepts are removed from the result as no partial definition or counter-partial definition can be computed though the irrelevant concept specification. Then, the specialisation finds new partial definitions, counter-partial definitions and descriptions from the refinements. Practically, redundancies are often checked before evaluating descriptions to avoid redundant evaluations and duplicated descriptions in the search tree, as a description can be generated from different branches.

The *combination algorithm* is used to combine the descriptions and counter-partial definitions to find new partial definitions. This is basically a set coverage algorithm. Counter-partial definitions that cover at least one negative example covered by the given concept

are added gradually into the set of candidates until all negative examples covered by the given concept are also covered by the candidates. Note that when a candidate is chosen, the set of remaining counter-partial definitions is often re-ordered. For simplicity, this is not shown in the algorithm.

Algorithm 3: Combination algorithm (Combine)

Input: a concept C, a set of counter-partial definitions *cpdefs* and negative examples $\mathcal{E}^-$
Output: a set $candidates \subseteq cpdefs$ such that $cn(C, \mathcal{E}^-) \subseteq \bigcup_{P \in candidates}(cn(P, \mathcal{E}^-))$

```
1  begin
2      candidates = ∅                          /* candidate counter-partial definitions */
3      cn_c = cn(C, E⁻)                        /* negative examples covered by C */
4      sort cpdefs by descending coverage of negative examples
5      while cpdefs ≠ ∅ and cn_c ≠ ∅ do
6          get and remove the top counter-partial definition D from cpdefs
7          if (cn(D, E⁻) ∩ cn_c) ≠ ∅ then
8              candidates = candidates ∪ D
9              cn_c = cn_c \ cn(D, E⁻))
10         end
11     end
12     if cn_c ≠ ∅ then
13         return ∅                                        /* return empty set */
14     else
15         return candidates
16
17 end
```

3.2. Algorithm implementation architecture

To inherit the advantages of the parallelisation approach in class expression learning, we employ the ParCEL learning algorithm design (Tran et al., 2012) which uses the map-reduce architecture (Ghemawat and Dean, 2004) to implement the idea of *parallel divide and conquer*. Here, our algorithm architecture is divided into two parts, as shown in Figure 2. The computationally heavy part, including refinement and evaluation of concepts, is done by the *multiple* workers. Moreover, the combination of descriptions and counter-partial definitions is also performed on the worker side to avoid overload on the learning algorithm.

The main learning algorithm is implemented on the reducer side. In our design, the reducer performs not only the reduction, as its name suggests, but also most of the tasks in Algorithm 1. Therefore, the terms *reducer* and *learner* are used interchangeably.

3.3. Counter-partial definition combination

Counter-partial definitions are combined with descriptions to create new partial definitions. In our implementation, we tried three combination strategies and the evaluation of these strategies is discussed in Section 4.2.

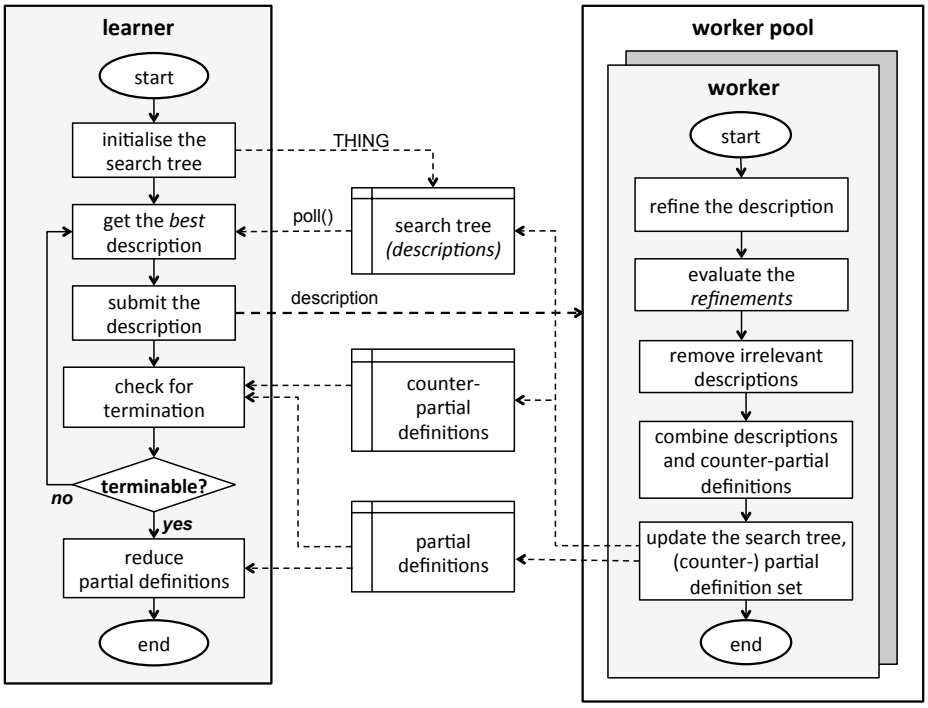


Figure 2: Algorithm implementation architecture

Lazy combination: In this strategy, the learner maintains the set of partial definitions and counter-partial definitions separately. When all positive or negative examples are covered, the combination is performed on the descriptions in the search tree and the set of counter-partial definitions. Consequentially, this is done by the learner.

Since the combination is performed after the learning stops, this strategy may provide a better combination, i.e. it may make better choices for counter-partial definitions. This advantage may result in shorter length or higher coverage partial definitions. However, for the learning problems in which both positive and negative examples need negation to be completely defined, the algorithm may not be able to find the definition because the refinement operator designed for this algorithm does not use negation. If this is the case and the timeout is set, the combination will be made when the timeout is reached to find the definition. Otherwise, it will not terminate until the system runs out of memory.

On-the-fly combination: This strategy is used in Algorithm 1. In this strategy, when a new description is generated from the refinement, it is combined with the existing counter-partial definitions if possible. Practically, the combination is performed on the worker side as new descriptions are always generated by workers. This strategy can avoid the termination problem discussed in the *lazy combination* strategy because negation is used in the combination which is performed for every new description. Our evaluation suggests that this strategy, overall, gives the best performance and the smallest search tree.

Delayed combination: This is an intermediate solution between the above strategies that checks for the possibility of combinations when a new description is generated. However, even if the combination is possible, only the set of cumulative covered positive examples is

updated (by removing from this set the elements that are covered by the new description), while the new description is put into a *potential partial definitions* set. The combination is executed when the termination condition is reached, i.e. either all positive examples or all negative examples are covered, or the timeout is approached.

This strategy may help to prevent the problem of the lazy combination strategy. In addition, it may return better combinations in comparison with the *on-the-fly* strategy because it inherits the advantage of the *lazy combination* strategy. However, the search tree is likely to be bigger and the learning time is longer than the *on-the-fly* strategy.

4. Empirical Evaluation

This section presents the empirical evaluation result of our algorithm. Our algorithm is implemented in Java and based on the DL-Leaner framework. The algorithm package is available at `https://parcelex.google.com` and it is also integrated into the DL-Learner repository at `http://dl-learner.svn.sourceforge.net`.

4.1. Methodology

In this evaluation, we are interested in measuring the learning time, correctness, completeness, predictive accuracy, F-measure (Rijsbergen, 1979), definition length, number of partial definition, average partial definition length (counted by the number of axioms in the definition) and the number of descriptions generated (search tree size). These dimensions can help to: i) quantify the scalability of the algorithm, ii) validate our approach, and iii) quantify the accuracy and readability of the learning result.

We benchmark our learner against the CELOE and ParCEL algorithms since studies in Hellmann (2008) and Tran et al. (2012) showed that these algorithms outperform many other description logic learning algorithms.

We use *10-fold cross validation* on a number of datasets that had been used by other authors in similar experiments (Železny et al., 2002; Hellmann, 2008; Tran et al., 2012). In addition, we use another real scenario dataset called MU-Bus that was generated from the bus operation timetable of a bus service. The data were sampled every 5 minutes and each observation has the form of `observation_id(day, month, year, hour, minute, bus/no_bus)`. The bus operation time depends upon the following conditions: semester (semester 1, 2 or summer semester), break (summer break, mid-year break 1, mid-year break 2, Christmas break), holiday (may be divided into two groups in which the holidays in the same group have the similar affect on the bus schedule), weekday/weekend. These parameters can be derived from the observation's date and time. We divided the dataset into subsets based on the sampling time (e.g. [07:10, 09:10], [07:10, 12:00], etc.) and time range (e.g. a week in each month plus special days or the whole year, etc.). This dataset can be found in our repository.

For the experiments, we used a Linux server with 8 Intel Xeon E5440 @2.83GHz CPU-cores, 32GB memory and the Redhat 4.1.2 (Linux version 2.6.18) operating system with a JRE 1.6.0 (64-bit) Java Virtual Machine (JVM). The heap size of the JVM in our experiments is 8GB.

4.2. Combination strategy comparison

Before comparing with other learning algorithms, we compare the combination strategies to choose the best strategy for further comparisons. Table 1 shows results using the three combination strategies (*lazy*, *delayed* and *on-the-fly* strategies) that are named ParCEL-Ex1, ParCEL-Ex12 and ParCEL-Ex2 respectively.

Table 1: Combination strategies evaluation result – MU-Bus Dataset *(averages ± standard deviations)*

Dimension	ParCEL-Ex1		ParCEL-Ex12		ParCEL-Ex2	
13 weeks[1], [07:10, 09:10][2]: 383 positive examples (p), 2292 negative examples (n)						
Learning time (s)	47.59	± 17.66	23.02	± 28.33	7.52	± 2.37
Accuracy (%)	100.00	± 0	99.81	± 0.36	99.74	± 0.36
F-measure (%)	100.00	± 0	99.36	± 1.23	99.10	± 1.21
Definition length	297.30	± 22.67	383.00	± 170.47	393.30	± 73.51
No of descriptions	39,096.80	± 12,210.86	14,463.90	± 20,232.28	2,130.10	± 1,179.48
No of pdef.[3]	1.00	± 0.00	4.20	± 1.69	6.30	± 1.25
Avg. pdef. length	297.30	± 22.67	145.32	± 155.66	63.40	± 10.54
13 weeks, [07:10, 12:00]: 670p, 5643n						
Learning time (s)	int. @600s		90.97	± 73.43	56.25	± 23.34
Accuracy (%)	99.89	± 0.15	99.84	± 0.20	99.83	± 0.20
F-measure (%)	99.48	± 0.70	99.26	± 0.92	99.18	± 0.96
Definition length	1,419.20	± 328.20	1,372.30	± 539.13	1,179.50	± 209.80
No of descriptions	187,279.60	± 2,584.59	23,843.40	± 21,430.22	8,575.10	± 4,184.27
No of pdef.	8.00	± 0	8.60	± 0.10	10.20	± 2.10
Avg. pdef. length	177.40	± 41.03	161.52	± 67.40	120.70	± 36.62

The result shows that these strategies have similar accuracy, but the learning time and the search space size (number of descriptions) is very different. As discussed in Section 3.3, the *lazy combination strategy* does not terminate in some cases. Result of the dataset *13 weeks, [07:10, 12:00]* shows that the learner was interrupted (by timeout) at 600 seconds, that means no complete definition was found after 600 seconds of learning. However, applying the combination algorithm on descriptions in the search tree and counter-partial definitions after the algorithm was interrupted, we obtained a definition with 100% accuracy on the training dataset. This means the solution implicitly existed, but the learner was not able to compute it. To make sure that we did not terminate the learner too early, we repeated the experiment and allowed it to run for 1,800 seconds. However, the learner was still not able to find the solution on the training dataset. This demonstrates the disadvantage of this strategy. We also ran experiments on some other MU-Bus datasets and the results are similar. For space reason, we do not show all experiment results here.

1. Around one week is chosen in each month. If there exists special events in the month such as holidays, break, etc., the selection will be expanded to include the events. This leads to about 13 weeks worth of data.
2. Time for sampling.
3. Partial definition

The *on-the-fly combination strategy* gives very promising results. It dominates other strategies on most of the dimensions, especially the learning time and the number of descriptions (search space). There is only one exception on the definition length in the dataset *13 weeks, [07:10, 09:10)* in which this strategy produces a longer definition than others. The number of partial definitions can help to explain the difference: there exists common parts (counter-partial definitions) amongst the partial definitions.

Finally, the *delayed combination strategy* is better than the *lazy evaluation strategy* but worse than the *on-the-fly strategy* on the learning time and the number of descriptions. In this strategy, we expected to get the advantages of both the *on-the-fly* and *delayed* combination strategies to get shorter definitions than ParCEL-Ex2 and smaller search spaces than ParCEL-Ex1. However, the experimental results show that this idea does not help much: the definitions are not always shorter than ParCEL-Ex2 while the search spaces are always bigger. Therefore, we chose *on-the-fly combination* as the major combination strategy for our learning algorithm to compare with other learners in our evaluation.

4.3. Evaluation result

Tables 2 and 3 show a summary of the evaluation results on popular datasets. In general, the learning time of our algorithm is better than CELOE and ParCEL. In particular, the learning time is significantly different in the UCA1 and MU-Bus datasets. Looking at the definition length, we can see that the longer the target definition is, the better ParCEL-Ex is. In particularly, for learning problems with exceptions that require complex definitions, e.g. the MU-Bus datasets, the learning time of our approach is much shorter than the others.

In our evaluation, we also want to focus on the size of the search trees generated by the algorithms to verify our objective of reducing the search space. The result matches with our expectation: the search trees generated by our algorithm have the smallest size in comparison with the search trees of other algorithms except for the Moral dataset. The possible reasons for that are: i) this learning problem has a very simple definition (length is 3), therefore it requires a small number of refinements, ii) in our algorithm, since there are several workers that explore different branches of the search tree at the same time, when a worker found a solution, the other workers may still be working and they need to finish their work. Since ParCEL-Ex2 mostly has smaller search trees, it always has better learning times than CELOE and ParCEL except in the case of Moral dataset discussed above.

For the definition length, ParCEL-Ex2 often produces the longest definitions. However, if we look at the definition, in some cases they can be shortened using optimisation. For example, the definitions returned by CELOE and ParCEL-Ex2 for the dataset Forte Uncle are:

- **CELOE:** `(male AND (EXISTS married.EXISTS sibling.Thing OR EXISTS sibling.EXISTS parent.male)`
- **ParCEL-Ex2:**
  - partial def. 1: `(sibling.EXISTS married.EXISTS Thing AND (NOT female))`
  - partial def. 2: `(married.EXISTS sibling.EXISTS Thing AND (NOT female))`

In the above examples, the length of the definition suggested by CELOE is 13 and ParCEL-Ex2 is 17 (length of two partial definitions plus 1 for disjunction). However, we can reduce it by at least 2 by reducing the common part among partial definitions, i.e. `(NOT female)` or even more if we replace `NOT female` by `male` using the disjoint property. Currently, this idea has not been implemented in our algorithm. However, receiving the result in the form of a set of partial definitions may help to provide a more readable representation.

In the most important scenarios that we want to focus on, the UCA1 and MU-Bus datasets, our algorithm outperformed the others. The definition for UCA1 illustrates the benefit of counter-partial definitions:

- **CELOE**: `activityHasDuration (hasDurationValue` $\geq$ `4.5 AND hasDurationValue` $\leq$ `21.5)`
- **ParCEL** :
  1. `EXISTS activityHasDuration.(hasDurationValue` $\geq$ `4.5 AND hasDurationValue` $\leq$ `15.5)`
  2. `EXISTS activityHasDuration.(hasDurationValue` $\geq$ `15.5 AND hasDurationValue` $\leq$ `19.5) AND EXISTS activityHasStarttime.Spring`
  3. `EXISTS activityHasDuration.(hasDurationValue` $\geq$ `15.5 AND` $\leq$ `19.5) AND EXISTS activityHasStarttime.Summer`
  4. `EXISTS activityHasStarttime.Autumn AND ALL activityHasDuration. (hasDurationValue >= 4.5 AND hasDurationValue <= 19.5`
- **ParCEL-Ex2**: `(activityHasDuration SOME (hasDurationValue` $\geq$ `4.5 AND hasDurationValue` $\leq$ `19.5) AND (NOT (activityHasDuration SOME hasDurationValue` $\geq$ `15.5 AND activityHasStarttime SOME Winter)))`

The definition of ParCEL-Ex2 is a combination of one description and a counter-partial definition (negated description). Note that the CELOE definition is shorter, but it does not fully describe the scenario.

The dataset MU-Bus is a harder learning problem in which the target definition is very complicated, since the bus operation time depends upon many conditions. For this dataset, ParCEL-Ex2 outperforms both ParCEL and CELOE in all dimensions. Our algorithm can always find the complete definition on training set and the accuracy on the test set is always over 99.7%, while CELOE could not find the correct definition on the training set and the accuracy on the test set is very low, from 13.17% to 48.88%. ParCEL is in the middle: the accuracy on the training set is between 94.16% and 100% and on the test set from 94.12% to 99.63% .

However, predictive accuracy is not an appropriate measurement for the accuracy in this experiment, as the number of negative examples is much bigger than the number of positive examples, around 8 to 10 times. Consequently, the algorithms that prefer correctness to completeness such as ParCEL and our algorithm are overly optimistic. Therefore, the F-measure is preferred in this circumstance to have a fair comparisons between algorithms. The result shows that our algorithm outperformed ParCEL and CELOE.

Although the definitions for MU-Bus datasets are rather long, their readability is still acceptable as they are broken into partial definitions. Moreover, the definition length may

be reduced significantly by using an optimisation, as discussed above. For example, one of the partial definitions for the dataset *MU-Bus, [07:10-12:00]* has a length of 120 axioms, but it can be reduced to 67 axioms by eliminating the common axioms.

Table 2: Experiment results summary *(averages ± standard dev.)*

Dimension	ParCEL-Ex2		ParCEL		CELOE	
Forte Uncle: 23p, 163n						
Learning time (s)	0.06	± 0.06	0.23	± 0.17	6.69	± 3.50
Accuracy (%)	100.00	± 0	100.00	± 0	98.00	± 4.22
F-measure (%)	100.00	± 0	100.00	± 0	96.67	± 10.54
Definition length	16.70	± 0.68	15.5	± 0.71	12.00	± 1.05
No of descriptions	174.10	± 108.23	859.50	± 251.02	64,707.90	± 33,641.92
Moral: 102p, 100n						
Learning time (s)	0.08	± 0.08	0.02	± 0.04	0.15	± 0.07
Accuracy (%)	100.00	± 0	100.00	± 0	100.00	± 0
F-measure (%)	100.00	± 0	100.00	± 0	100.00	± 0
Definition length	5.60	± 6.65	3.00	± 0	3.00	± 0
No of descriptions	223.40	± 364.23	33.30	± 10.95	540.50	± 16.85
Aunt: 41p, 41n						
Learning time (s)	0.30	± 0.15	0.45	± 0.22	29.15	± 28.10
Accuracy (%)	100.00	± 0	100.00	± 0	98.75	± 3.95
F-measure (%)	100.00	± 0	100.00	± 0	98.89	± 3.51
Definition length	22.10	± 5.30	17.60	± 0.84	18.60	± 1.27
No of descriptions	2,127.80	± 1,160.91	7,023.40	± 2,912.07	85,883.80	± 67,328.78
Uncle: 38p, 38n						
Learning time (s)	0.28	± 0.08	0.39	± 0.22	48.16	± 53.03
Accuracy (%)	100.00	± 0	98.33	± 5.27	100.00	± 0
F-measure (%)	100.00	± 0	98.57	± 4.52	100.00	± 0
Definition length	19.10	± 1.52	17.30	± 0.68	18.00	± 0
No of descriptions	2,400.00	± 551.22	6,332.50	± 3,247.89	541,081.80	± 559,103.75
UCA1: 73p, 77n						
Learning time (s)	1.08	± 0.65	57.96	± 7.82	int. @300	
Accuracy (%)	100.00	± 0	100.00	± 0	91.24	± 6.41
F-measure (%)	100.00	± 0	100.00	± 0	91.98	± 5.72
Definition length	27.30	± 15.03	51.40	± 1.27	9.00	± 0
No of descriptions	14,053	± 9,409	991,828	± 132,524	1,465,263.20	± 11,515.54

Table 3: Experiment result summary – MU-Bus Dataset *(averages ± standard dev.)*

Dimension	ParCEL-Ex2		ParCEL		CELOE	
13 weeks, [07:10, 09:10]: 670p, 5643n						
Learning time (s)	7.52	± 2.37	240.34	± 46.9	int. @600s	
Accuracy (%)	99.74	± 0.36	99.63	± 0.31	48.88	± 0.26
F-measure (%)	99.10	± 1.21	98.72	± 1.05	35.98	± 0.19
Definition length	393.30	± 73.51	254.40	± 21.58	12.00	± 0
No of descriptions	2,130.10	± 1,179.48	643,401.10	± 139,030.85	256,919.50	± 907.22

Continued on next page

Table 3 – continued

Dimension	ParCEL-Ex2		ParCEL		CELOE	
No of pdef.	6.30	± 1.25	15.00	± 1.56	N/A	
Avg. pdef. length	63.41	± 10.54	16.99	± 0.42	N/A	
13 weeks, [07:10, 12:00]: 670p, 5643n						
Learning time (s)	56.25	± 23.34	int. @600s		int. @600s	
Accuracy (%)	99.83	± 0.20	97.91	± 0.50	14.35	± 1.10
F-measure (%)	99.18	± 1.96	66.48	± 6.78	19.86	± 0.02
Definition length	1,179.50	± 209.80	398.30	± 54.89	2	± 0
No of descriptions	8575.10	± 4184.27	879,866.30	± 34,000.30	79,959.50	± 118.47
No of pdef.	10.20	± 2.10	24.80	± 3.52	N/A	
Avg. pdef. length	120.70	± 36.62	16.07	± 0.19	N/A	

A paired t-test was used to test the statistical significance of the results between our algorithm and CELOE and between our algorithm and ParCEL. Since all three algorithms achieved 100% accuracy for most of the datasets in Table 2, we only used the t-test for the learning time and number of descriptions. All results between our algorithm and CELOE, and our algorithm and ParCEL were significant at the 5% significance level.

For the experimental results in Table 3, we tested the running time, accuracy, F-measure, number of descriptions and definition length. The results show that the difference between our algorithm and CELOE is statistically significant at the 1% significance level for all tested dimensions. There were also significant differences between our algorithm and ParCEL except for the accuracy, F-measure and the number of descriptions in the dataset *MU-Bus 13 weeks, [07:10, 09:10]*.

5. Conclusion

We have proposed a new approach to class expression learning: we learn from both positive and negative examples, motivated by learning scenarios that have exceptions in the patterns of the positive examples. The exception is common in the practice and our empirical experiments suggest that our approach works well.

Some current learning algorithms, e.g. CELOE and ParCEL, which are used in our evaluations, can also solve this category of problem by specialising the concepts or using negation and conjunction to remove negative examples from the potential concepts. However, for some datasets with regular exception patterns such as MU-Bus and UCA1, these algorithms have difficulties in finding the right concept: The learning time is very long in comparison with our algorithm, which may cause the system to run out of memory before the definition is found.

Our algorithm works well not only on the expected scenarios, but also on other scenarios that do not use negation, as shown in Table 2. The most impressive improvements are the search tree size and learning time. Although our algorithm often generates longer definitions, there is no over-fitting for the datasets used. This does not mean that our algorithm can avoid over-fitting for all learning problems, but currently with the datasets used, our algorithm shows promises.

The definitions generated by our algorithm are not optimised. The optimisation may help to get the better definitions, i.e. shorter length and more readable. This, together with investigations on more datasets will be the future works for our research.

References

H. Ade, L. Raedt, and M. Bruynooghe. Declarative bias for specific-to-general ILP systems. *Machine Learning*, 20(1):119–154, 1995.

F. Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.

F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. IJCAI*, volume 16, 1999.

W.W. Cohen and H. Hirsh. Learning the classic description logic: Theoretical and experimental results. In *Proc. International Conference on Principles of Knowledge Representation and Reasoning*, pages 121–133, 1994.

N. Fanizzi, C. d'Amato, and F. Esposito. DL-FOIL concept learning in description logics. *Inductive Logic Programming*, pages 107–121, 2008.

S. Ghemawat and J. Dean. MapReduce: Simplified data processing on large clusters. In *Symposium on Operating System Design and Implementation (OSDI04), San Francisco, CA, USA*, 2004.

S. Hellmann. Comparison of concept learning algorithms. *Master's Thesis, Leipzig University*, 2008.

L. Iannone, I. Palmisano, and N. Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.

J. Lehmann. *Learning OWL Class Expressions*. AKA Akademische Verlaggesellschaft, 2010.

J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78(1):203–250, 2010.

J. Lehmann, S. Auer, S. Tramp, et al. Class expression learning for ontology engineering. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2011.

F. Lisi and D. Malerba. Ideal refinement of descriptions in AL-Log. *Inductive Logic Programming*, pages 215–232, 2003.

D.L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C Recommendation*, 10:2004–03, 2004.

S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. *Inductive Logic Programming*, pages 261–280, 1992.

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. ISBN 0408709294.

C. Rouveirol and V. Ventos. Towards learning in CARIN-ALN. *Inductive Logic Programming*, pages 191–208, 2000.

A.C. Tran, S. Marsland, J. Dietrich, H. Guesgen, and P. Lyons. Use cases for abnormal behaviour detection in smart homes. *Aging Friendly Technology for Health and Independence*, pages 144–151, 2010.

A.C. Tran, J. Dietrich, H. Guesgen, and S. Marsland. An approach to parallel class expression learning. *Rules on the Web: Research and Applications*, pages 302–316, 2012.

F. Železny, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th International Conference on ILP*, pages 333–345. Springer-Verlag, 2002.