Journal of Machine Learning Research 18 (2017) 1-34 Submitted 8/14; Revised 6/17; Published 8/17

Parallel Symmetric Class Expression Learning

An C. Tran TCAN@CIT.CTU.EDU.VN
College of Information and Communication Technology
Can Tho University, Vietnam

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, New Zealand.

Editor: Luc De Raedt

Abstract

In machine learning, one often encounters data sets where a general pattern is violated by a relatively small number of exceptions (for example, a rule that says that all birds can fly is violated by examples such as penguins). This complicates the concept learning process and may lead to the rejection of some simple and expressive rules that cover many cases. In this paper we present an approach to this problem in description logic learning by computing partial descriptions (which are not necessarily entirely complete) of both positive and negative examples and combining them. Our *Symmetric Parallel Class Expression Learning* approach enables the generation of general rules with exception patterns included. We demonstrate that this algorithm provides significantly better results (in terms of metrics such as accuracy, search space covered, and learning time) than standard approaches on some typical data sets. Further, the approach has the added benefit that it can be parallelised relatively simply, leading to much faster exploration of the search tree on modern computers.

Keywords: description logic learning, parallel, symmetric, exception

1. Introduction

In machine learning we seek general rules that describe data. However, it is not uncommon for there to be exceptions to general rules, i.e., particular situations where those rules do not apply. In this situation, non-monotonic reasoning can be used, and a default rule that covers most of the cases is retracted if further evidence is provided that overrides the general case. Non-monotonic reason will be discussed more in Section 4.1.

For example, in the classic *Tweety* demonstration of logical proof, the first assertion is that *'Birds fly'*. This is *generally* true, and so the follow-up statement that *'Tweety is a bird'* has the implication that *'Tweety can fly'*. However, there are several species of bird that cannot fly, such as penguins. This type of example can be covered by a set of extra statements: *'Penguins are birds. Penguins do not fly. Tweety is a penguin.'* Non-monotonic reasoning is able to retract the earlier inference that *'Tweety can fly'* given the extra information that *'Tweety is a penguin.'*

An alternative approach to this is to make the general rules correct by listing the exceptions: *'Birds fly **except** penguins'*. Providing that the complete rule is still 'simpler' (e.g., shorter) than other rules that describe the same situation (i.e., enumerations, or disjunctions of types of birds that can fly), this can provide a net gain in learning. Thus, a short general rule can be augmented by a set of exception rules that make the general rule correct. In the Tweety case this could be a list of counter-examples, possibly by species and genus, such as *'All birds can fly except ratites and penguins'*.

In this paper, we enable such use of exception for description logics by explicitly considering negative examples and building rules for them. As we will show, this leads to a parallel algorithm that treats positive and negative examples in exactly the same way (thus meaning that the learning would be invariant to a switch between positive and negative in the data). Experiments on a variety of standard learning problems demonstrate that this algorithm is statistically significantly faster and more accurate than standard approaches from the literature.

Most existing description logic (DL) learning algorithms, e.g. $\mathcal{AL}$ − QuIn (Lisi and Malerba, 2003), DL-FOIL (Fanizzi et al., 2008), CELOE (Lehmann and Hitzler, 2010) and ParCEL (Tran et al., 2012), only take the definitions of negative examples into account implicitly, by incorporating them into the definitions of positive examples. Learning starts from a very general concept, usually 'top' in DL, and subclasses and sub-properties, conjunction (intersection) and the combination of conjunction and negation (subtraction) are used to remove the negative examples from the candidate concepts.

While this approach is suitable for many concept learning problems and has been used successfully by Hellmann (2008); Lehmann and Hitzler (2010); Tran et al. (2012), it does not use descriptions in the search space efficiently. Consider a top-down approach for learning such as CELOE, which constructs the search tree using a downward refinement operator until it finds a node (expression) that correctly and completely defines the positive examples. Given a concept learning problem with a set of positive (+) and negative (–) examples, the concepts D_1, D_2, N_1, N_2 and their coverage[1] as described in Figure 1(a), CELOE will find the single concept $C = (D_1 \sqcap \neg N_1) \sqcup (D_2 \sqcap \neg N_2)$ (shown in Figure 1(b)).

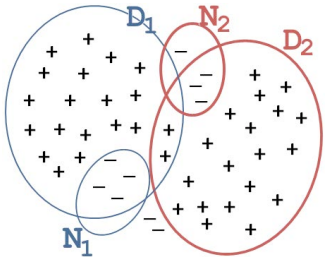
Algorithms that combine both top-down and bottom-up approaches (such as ParCEL and DL-FOIL) find two simpler concepts: $P_1 = D_1 \sqcap \neg N_1$ and $P_2 = D_2 \sqcap \neg N_2$. These concepts are visualised in Figure 1(c). If all of the concepts D_1, D_2, N_1 and N_2 have the same length of 3, then the length of the longest concepts generated by these two approaches are 16 (4 concepts of length 3 plus 4 operators) and 8 (2 concepts of length 3 plus 2 operators) respectively. If a concept is of length 16 then it must occur at a depth of at least 16 in the search tree, and so CELOE will only be able to identify this concept after searching the previous 15 levels of the search tree. As the expansion of the search tree in these approaches is directed by a heuristic that is based upon accuracy, not all branches of the search tree are fully expanded; this prevents 'dummy' breadth-first expansion.

The difference between these two methods is the way that the search tree is constructed: CELOE refines a general concept, while ParCEL searches for partial solutions that provide a correct (true) description of a subset of the data, and combines them to generate a complete solution. This idea of partial solutions can be extended to define exceptions as

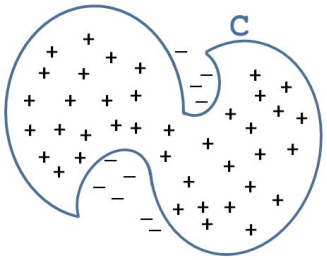
1. Coverage of a concept is the set of its instances, see Definition 1

Figure 1: Exception patterns in learning. Pluses (+) denote positive examples, minuses (–) denote negative examples. The ellipses represent coverage of the corresponding class expressions.

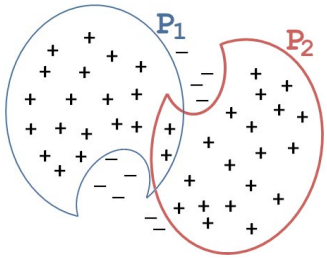
(a) The target concept can be constituted from simple descriptions D_1, D_2, N_1 and N_2: $(D_1 \sqcap \neg N_1) \sqcup (D_2 \sqcap \neg N_2)$. The description would be given by the SPaCEL learning algorithm described in this paper.



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



(c) ParCEL learnt concepts: constructed from more complex descriptions $P_1 = D_1 \sqcap \neg N_1$ and $P_2 = D_2 \sqcap \neg N_2$ describe the data set concisely



partial descriptions of negative examples, and this is what we consider in this paper. It enables a learning algorithm to find the solution suggested in Figure 1(a), which is at depth three in the search tree that combines descriptions of positive and negative examples separately. This is the main idea of the algorithm that we propose in this paper, which we call SPaCEL for reasons that will be made clear later, and which treats positive and negative examples equally and can search for their definition in parallel. Definitions of negative examples can then be combined with other concepts in the search tree to construct new partial solutions. In the case of the example in Figure 1, definitions N_1 and N_2 of negative examples, which are ignored by CELOE and ParCEL, are employed by SPaCEL and are combined with concepts D_1 and D_2 to to constitute partial solutions $D_1 \sqcap \neg N_1$ (equivalent to P_1 in CELOE) and $D_2 \sqcap \neg N_2$ (equivalent to P_2 in CELOE). In some cases, definitions of negative examples can be very simple and thus they can be found very early in the search process. In addition, each negative example can be combined with a variety of concepts to constitute several partial solutions. This will result in smaller search trees and faster learning time. Our evaluation shows that this approach works well not only in cases where exceptions exist, but also in normal cases.

Figure 2, which extends the *Tweety* example using the knowledge base shown in Figure 2(a), provides another simple example of the benefits of exceptions. For the instances given in Figure 2(b) (where b_i are instances of Bird, p_i instances of Penguin, c_k instances of Mammal and t_m instances of Bat), the p_i can be considered as exceptions to a rule that says that birds fly, and likewise t_m for a rule that says that mammals do not. Figure 2(c) represents the search tree for the three learning strategies described above. Finding a complete definition (triple-line node, e.g., CELOE) requires a deeper search tree than finding several shorter partial definitions (double-line nodes, e.g. ParCEL, DL-FOIL). However, using definitions of exceptions (dotted-line nodes) can produce even shallower search trees.

Further, since some 'useful' definitions (such as `Bat` in this example) are based on negative examples, they may be ignored by the first two approaches.

Another, more complex example that demonstrates differences between the three approaches is in the uncle relationship for the Forte Uncle data set. The knowledge base of this data set contains concepts and relations that are related to family relationships, such as *Person, Male, Female, married, has parent, has sibling*, etc. It also includes some instances of people and their properties. For example, *Alfred* is a male and is the father of David and Elisa. *Alice* is a female, she is married to Art and is the mother of F14, M13, M15. *Art* is a male and he is the father of F14, M13 and M15. He also has two siblings, Umo and Wendy. *Wendy* is a female and is married to Walt. She has two children, F12 and M11. Her siblings are Art and Umo. In this set, Art is a positive example of an uncle, and Alfred, Alice and Wendy are negative examples (Alfred does not have siblings, Alice and Wendy are female). There are a total of 186 instances in the data set. A detailed description of this data set is provided in Section 3.1.

Written in Manchester syntax (Horridge and Patel-Schneider, 2009) and also in English, definitions of the uncle relationship learnt by the three different approaches are :

- **CELOE:** generates a single complex solution of length 15:

  male AND ((married SOME hasSibling SOME Person) OR
  (married SOME hasSibling SOME hasChild SOME Thing))

  (An uncle is a male who has married someone with a sibling; or a man who married someone's sibling and the sibling has some children)

- **ParCEL:** generates two partial solutions of length 7 and 9:

  1. male AND hasSibling SOME hasChild SOME Thing

     (An uncle is a man who has at least sibling and a sibling has children)

  2. male AND married SOME hasSibling SOME hasChild SOME Thing

     (An uncle is a man who married someone with at least one sibling, and that sibling has a child)

- **SPaCEL:** generates two partial solutions that are created by combining two concepts "hasSibling SOME hasChild SOME Thing" of length 5 and "married SOME hasSibling SOME hasChild SOME Thing" of length 7 with a partial solution "female" of negative examples:

  1. hasSibling SOME hasChild SOME Thing AND (NOT female)

     (An uncle is a person who has a sibling with children, and who is not female)

  2. married SOME hasSibling SOME hasChild SOME Thing AND (NOT female)

     (An uncle is a person who married someone with a sibling who has children and who is not female)

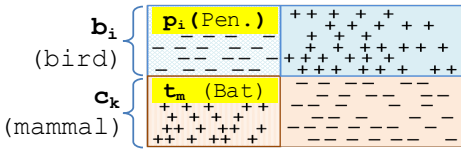
Figure 2: Different approaches to learning the definition for the *extended Tweety* concept learning problem.

(a) The knowledge base of the learning problem.

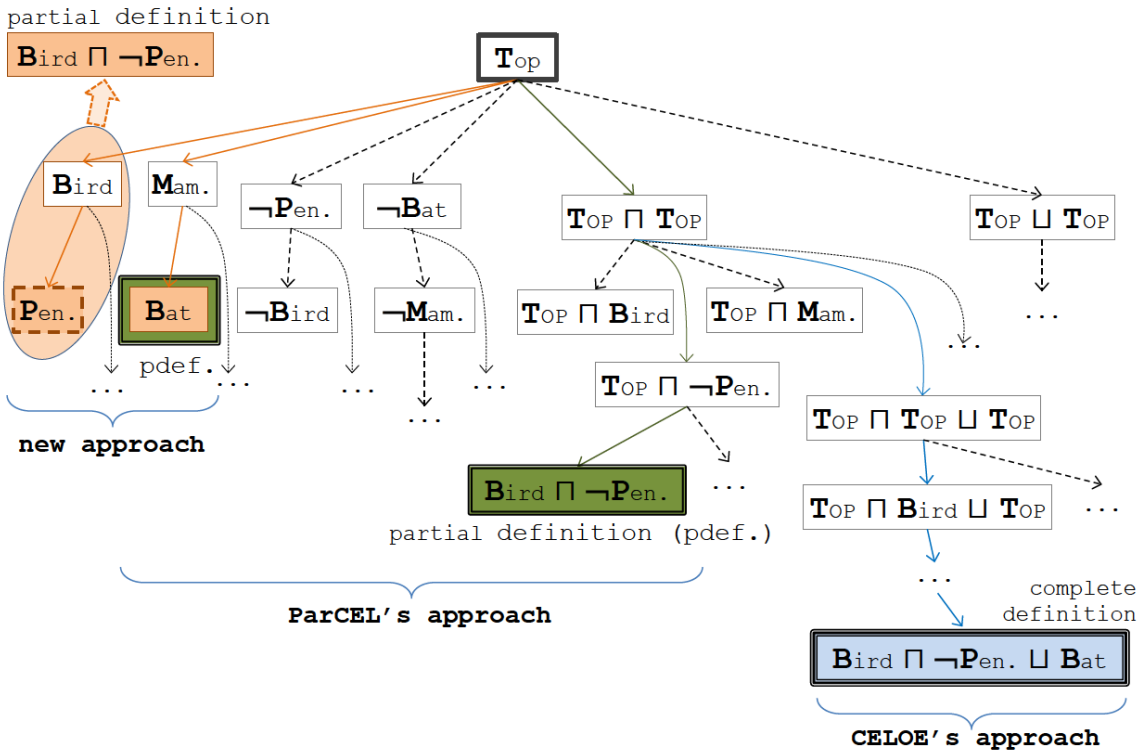
Concepts	Instances
Top	
Bird	$\{b_i\}$
Penguin	$\{p_j\}$
Mammal	$\{c_k\}$
Bat	$\{t_m\}$
Fly	$\{b_i, t_m\}$
¬Fly	$\{p_j, c_k\}$

observations

(b) Visualisation of the knowledge base.



(c) The search tree for learning the *extended Tweety* example using three different approaches. *Triple-line nodes* represent complete definitions (e.g., CELOE). *Double-line nodes* represent partial definitions (e.g., ParCEL). *Dashed-line nodes* represent definitions of some negative examples. The *ellipses* represent the combinations of an (incorrect and incomplete) expression with a definition of negative examples that constitutes a partial definition. *Solid arrows* represent the paths to the definitions for positive or negative examples.



The partial solution of negative examples can be found in an early stage of the learning progress. However, it is ignored by CELOE and ParCEL, whereas SPaCEL employs it for later combinations with concepts in the search tree to generate partial solutions if possible. All three solutions are valid, but the SPaCEL one can be found more quickly.

Using exceptions could increase the risk of selective over-fitting, since sub-cases are given high weight, but the emphasis on short descriptions for the general rules, and the fact that the exception cases are also based on short descriptions, ensures that this is unlikely to happen. We present experimental evidence that this method often gives higher predictive accuracy on test data. It is true that logical formulae based on exceptions can quickly become torturous, as a short general description gets progressively more complex caveats added to it, and possibly even exceptions to the exceptions. However, the constraints on search that are implicit in logic programming (such as search tree size, definition length, and learning time) means that these complex cases are very unlikely to ever be found.

One key part of our approach is that positive and negative examples are treated in the same way; this 'symmetry' gives us the name of our algorithm: *Symmetric Parallel Class Expression Learning* (SPaCEL). A secondary point (referred to in the 'parallel' part of the name) is that the algorithm is inherently parallelisable in the first phase of learning, where separate definitions of positive and negative examples are sought. These definitions are combined in the second phase to construct a sufficiently accurate definition. By using rules about negative examples to remove them from the concepts in the search tree it is possible to combine (potentially very short) concepts and so define relatively complex concepts based on much smaller depth traversals. Consequently, concepts in the search tree are used more effectively and thus the learning time can be reduced.

We present a formal definition of our learning algorithm in Section 2, together with a discussion of various strategies for combining partial definitions of positive and negative examples. In addition, we demonstrate the parallelisation strategy that is one part of what makes our algorithm so effective computationally. We then present the data sets that we use for evaluation in Section 3.1, followed by a comparison of the three strategies for combining definitions that we have considered. Following this, in Section 3.3 we report search tree size, predictive accuracy, learning time, and definition length for 15 data sets of varying complexity for CELOE, ParCEL and SPaCEL. In Section 4.1, we summarise some related work before concluding with a summary and discussion of future work in Section 5.

2. Symmetric Parallel Class Expression Learning

Our learning approach constructs definitions of both positive and negative examples and gives them equal weight in order to find the final definition. This is performed in three main steps:

1. Find partial definitions for positive and negative examples separately such that the set of definitions, together, cover all positive and negative examples.
2. Consolidate the partial definitions to remove redundancies (i.e., the partial definitions whose coverage is a subset of the union of other partial definitions) and select the best candidates for constructing a complete definition.
3. Aggregate the best candidates to form the complete definition using disjunction.

In the first step, partial definitions can be produced directly by specialisation using a downward refinement operator, as in algorithms such as DL-FOIL and ParCEL. Definitions for negative examples are constructed side-by-side with definitions for positive examples and are used in combination with existing expressions in the search tree to create further partial definitions.

In the next section we introduce the concepts necessary for our algorithms. The algorithms themselves will be presented in Section 2.2. We follow the definitions used by Lehmann and Hitzler (2010) closely, although some of the concepts have to be extended in order to include negative examples.

2.1 Symmetric class expression learning

A *concept learning problem* in description logics can be described as a structure $\langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$ where $\mathcal{K}$ is a knowledge base (ontology), $\mathcal{E}^+$ is a set of positive examples and $\mathcal{E}^-$ is a set of negative examples such that $\mathcal{E}^+ \cap \mathcal{E}^- = \emptyset$. A learnt concept is also called a *hypothesis* or *definition*. $\mathcal{K} \models C(a)$ denotes that a is an instance of class C with respect to the knowledge base $\mathcal{K}$. If $\mathcal{K} \models C(a)$ is satisfied, C is said to *cover* a with respect to $\mathcal{K}$. The aim of *description logic learning* is to find a concept C such that $\mathcal{K} \models C(e)$ for all $e \in \mathcal{E}^+$ (*completeness*) and $\mathcal{K} \not\models C(e)$ for all $e \in \mathcal{E}^-$ (*correctness*). A learnt concept is called *complete* if it covers all positive examples, *correct* if it does not cover any negative examples, *accurate* if it is both complete and correct, *overly general* if it is complete but incorrect, and *overly specific* if it is correct but incomplete.

In order to define negative examples, we extend the concept *cover* for a set of instances in the form of a function, as follows:

Definition 1 (Cover) *Let $\mathcal{K}$ be a logic knowledge base, $\mathcal{X}$ be a set of instances and* C *be a concept. Then,* $\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{X})$ *is a function that computes a set of examples in $\mathcal{X}$ covered by* C *with respect to $\mathcal{K}$:*

$$\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{X}) = \{\mathrm{e} \in \mathcal{X} \mid \mathcal{K} \models C(e) (\text{that is:e } \textit{is covered by } \mathrm{C} \textit{ with respect to } \mathcal{K})\}$$

This can be generalised to a set of definitions as:

$$cover(\mathcal{K}, \mathcal{Q}, \mathcal{X}) = \bigcup_{C \in \mathcal{Q}} cover(\mathcal{K}, \mathrm{C}, \mathcal{X})$$

where $\mathcal{K}$ be a knowledge base, $\mathcal{X}$ be a set of instances and $\mathcal{Q}$ be a set of definitions.

Definition 1 enables us to give definitions of positive examples (which are called *partial definitions*) and negative examples (which are called *counter-partial definitions*):

Definition 2 (Partial definition) *For a concept learning problem $\langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$, a concept* C *is called a* partial definition *if* $\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{E}^+) \neq \emptyset$ *and* $\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{E}^-) = \emptyset$.

Definition 3 (Counter-partial definition) *For a concept learning problem $\langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$, a concept* C *is called a* counter-partial definition *if* $\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{E}^+) = \emptyset$ *and* $\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{E}^-) \neq \emptyset$.

Definitions 2 and 3 are 'ideal' in that they insist on the cover of the opposite set being the empty set. In practice, it is possible to allow a few negative examples to be covered by a partial definition, and a few positive examples by a counter-partial definition; this could potentially help with noisy data.

The first (top-down) step of our algorithm consists of finding partial and counter-partial definitions separately using a refinement operator and a combination strategy to compute further partial definitions from the counter-partial definitions and existing expressions in the search tree. As these searches are performed separately and simultaneously, our approach avoids being over-specific for concept learning problems that contain exceptions.

These sets are then combined, which enables the algorithm to cope with exceptions to general rules through the opposing definitions (so rules made of partial definitions can have counter-partial definitions added, and vice versa). This is simply the use of conjunction and negation, The combination acts as an extra step to deal with the exceptions with the support of counter-partial definitions. In our approach, only one downward refinement operator is used in the specialisation step to generate both partial and counter-partial definitions. The combination strategy essentially consists of checking for the possibility of creating new partial definitions from an expression and counter-partial definitions, using conjunction and negation: a combination of an expression C with a set of counter-partial definitions $\mathcal{X}$ has the form $C \sqcap \neg(\sqcup_{D \in \mathcal{X}} D)$.

Definition 4 (Combinability) *Given a concept learning problem* $\mathrm{LP} = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$*, a set of counter-partial definitions* $\mathcal{Q}$*, and an expression* C *such that* $\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{E}^+) \neq \emptyset$ *and* $\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{E}^-) \neq \emptyset$*,* C *is said to be* combinable *with* $\mathcal{Q}$ *iff:*

$$\mathrm{cover}(\mathcal{K}, \mathrm{C}, \mathcal{E}^-) \subseteq \mathrm{cover}(\mathcal{K}, \mathcal{Q}, \mathcal{E}^-),$$

which means that C *can be "corrected" by* $\mathcal{Q}$*.*

The main objective of the combination step is to check for the combinability of descriptions in the search tree. Combining definitions can be performed at several stages of the learning algorithm; this is discussed further in Section 2.3.

In this paper, we employ the refinement operator used by Lehmann and Hitzler (2010). However, the combination step helps us avoid the usage of negation in the refinement without any loss of generality, and the use of *partial models* and disjunction in the reduction steps means there is no need for disjunction in the refinement. Therefore, we use only 5 of the 7 basic rules in the original refinement operator described by Lehmann and Hitzler (2010). The two rules which refine the negation and disjunction are not used, thus our refinement operator is defined as:

Definition 5 (SPaCEL refinement operator $\rho_\sqcap$**)** *Given an expression* C*, a set of concept names* $\mathrm{N_C}$ *and a set of property (role) names* $\mathrm{N_R}$*, then* $\rho_\sqcap$ *is defined as follows:*

1. *if* $C \in N_C$ *(C is an atomic concept):*
   $\rho_\sqcap(C) = \{C' \mid C' \sqsubset C \text{ and } \nexists C'' \in N_C : C' \sqsubset C'' \sqsubset C\} \cup \{C \sqcap C' \mid C' \in \rho_\sqcap(\top)\}$
   (if C is an atomic concept, it is specialised by using its proper sub-concepts or creating a conjunction with the refinements of the `TOP` *concept* $\rho_\sqcap(\top)$*).*

2. *if* $C = \top$ *(C is the* TOP *concept):*
   $\rho_{\sqcap}(C) = \{C' \mid C' \in N_C, \nexists C'' \in N_C : C' \sqsubset C'' \sqsubset C\}$
   $\cup\ \{\exists r.\top \mid r \in N_R\}\ \cup\ \{\forall r.\top \mid r \in N_R\}$
   (if C is the TOP *concept, specialise it by using its proper sub-concepts or property restrictions)*

3. *if* $C = C_1 \sqcap ... \sqcap C_n$ *(C is a conjunctive of descriptions):*
   $\rho_{\sqcap}(C) = \{C_1 \sqcap ... \sqcap C' \sqcap ... \sqcap C_n \mid C' \in \rho_{\sqcap}(C_i), 1 \leq i \leq n\}$
   (if C is a conjunction, it is specialised by refinements of its conjuncts)

4. *if* $C = \forall r.D, r \in N_R$*:* $\rho_{\sqcap}(C) = \{\forall r.D' \mid D' \in \rho_{\sqcap}(D)\}$
   $\cup\ \{\forall r.\bot \mid D = A \in N_C$ and $\nexists A' \in N_C$ *s.t.* $A' \sqsubset A\}$
   (if C is a 'for all' property restriction, it is specialised by refinements of its range)

5. *if* $C = \exists r.D, r \in N_R$*:* $\rho_{\sqcap}(C) = \{\exists r.D' \mid D' \in \rho_{\sqcap}(D)\}$
   (this rule is for the 'there exists' property restriction, and is similar to the 4^{th} *rule.)*

Note that in the expression for $\rho_{\sqcap}$ in the second rule, negation and disjunction are not used, since they can be generated by *combination* and *aggregation* (steps 2 and 3 of our algorithm) respectively.

The top-down step finishes when either the partial definitions cover all positive examples or the counter-partial definitions cover all negative examples (up to allowable noise), which produces this formal definition (where the first line of this describes as the union of correct descriptions of the positive data and correctable descriptions of the same data):

Definition 6 (SPaCEL top-down learning) *Given a concept learning problem* $LP = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-)\rangle$*, the top-down learning step in this approach aims to find a set of partial definitions* $\mathcal{P}$*, a set of counter-partial definitions* $\mathcal{Q}$ *and a set of expressions* $\mathcal{X}$ *such that:*

$$cover(\mathcal{K}, \mathcal{P}, \mathcal{E}^+) \cup cover(\mathcal{K}, \mathcal{X}, \mathcal{E}^+) = \mathcal{E}^+,$$
$$\textit{such that } \forall C \in \mathcal{X} \mid C \text{ is combinable with } \mathcal{Q}, \textit{ or}$$
$$cover(\mathcal{K}, \mathcal{Q}, \mathcal{E}^-) = \mathcal{E}^-.$$

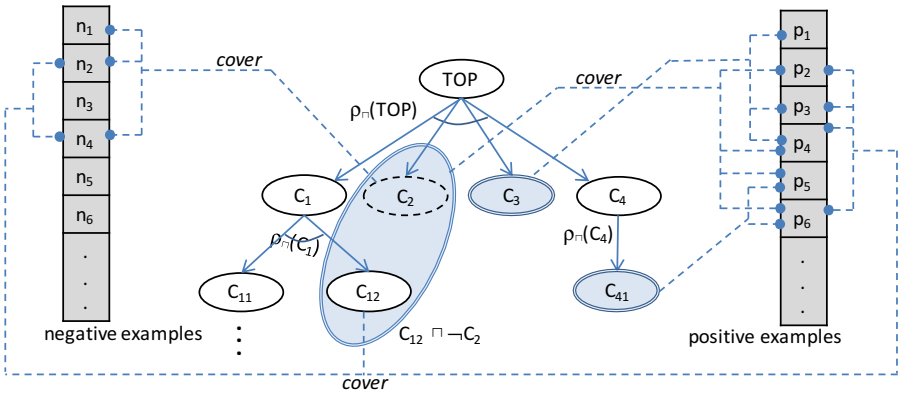
Figure 3 demonstrates the top-down step in our approach that uses the refinement operator $\rho_{\sqcap}$ to produce both partial and counter-partial definitions. In this example, the combination of the counter-partial definition C_2 and the expression C_{12} creates a partial definition $C_{12} \sqcap \neg C_2$.

As in the ParCEL algorithm, the selection of expressions in the search tree for refinement is controlled by node scores that are computed by a learning heuristic. The particular learning heuristic that we use is adapted from Lehmann and Hitzler (2010). However, the factors involving in the scoring function and their weights are redefined accordingly to our learning strategy:

Definition 7 (SPaCEL score) *Let* $LP = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-)\rangle$ *be a concept learning problem, C be a class expression and* C' *the parent expression of C. The score of C is computed as:*

$$score(C) = correctness(C, LP)$$
$$+ \alpha \times gain(C, LP) + \beta \times completeness(C, LP) - \gamma \times length(C, LP)$$
$$(\alpha \geq 0, \beta \geq 0, \gamma \geq 0)$$

Figure 3: The top-down learning step aims to find both partial definitions and counter-partial definitions. *Double-line* nodes are *partial definitions*, *dashed-line nodes* are *counter-partial definitions*. $\rho_{\sqcap}$ is the refinement operator defined in Definition 5. *Connections* from nodes to examples represent the *coverage* of the expressions.



where $gain(C, LP) = accuracy(C, LP) - accuracy(C', LP)$.

In this definition, correctness, completeness and accuracy have their standard meanings, i.e., (Lehmann and Hitzler, 2010):

$$\begin{aligned} correctness(C, LP) &= \frac{|\mathcal{E}^- \backslash cover(|\mathcal{K}, C, \mathcal{E}^-)|}{|\mathcal{E}^-|}, \\ completeness(C, LP) &= \frac{|cover(\mathcal{K}, C, \mathcal{E}^+)|}{|\mathcal{E}^+|}, \\ accuracy(C, LP) &= \frac{|cover(\mathcal{K}, C, \mathcal{E}^+)| + |\mathcal{E}^- \backslash cover(\mathcal{K}, C, \mathcal{E}^-)|}{|\mathcal{E}^+ \cup \mathcal{E}^-|}. \end{aligned}$$

The length of an expression is taken to be the sum of the number of concept names, role names, quantifiers, and connective symbols occurring in the expression, which for expressions in the $\mathcal{ALC}$ language can be defined as:

Definition 8 (Length of an $\mathcal{ALC}$ expression) *The length of an expression D or E (denoted by $|D|$) is defined inductively as follow (where A denotes an atomic concept):*

$$\begin{aligned} |A| = |\top| = |\bot| &= 1 \\ |\neg D| &= |D| + 1 \\ |D \cup E| = |D \cap E| &= 1 + |D| + |E| \\ |\exists r.D| = |\forall r.D| &= 2 + |D| \end{aligned}$$

The learning heuristic is mainly based on the correctness of the class expression (in contrast to that of Lehmann and Hitzler (2010), which is based on accuracy). A penalty is applied for long definitions to avoid infinitely deep search (necessary because the refinement operator used in our learning algorithm is infinite). A bonus for accuracy gained by an

expression is also applied, as accurate expressions are more likely to be close to the solution. A bonus is also given for the relatively complete expressions.

Default values for the weightings of the above factors were chosen based on experimental investigations and the work by Lehmann and Hitzler (2010). We chose $\alpha = 0.2, \beta = 0.01, \gamma = 0.05$. These values can be adjusted based on the characteristics of the learning problem, such as decreasing the penalty for learning problems that need long definitions, thus biasing the search to look for deeper solutions.

The last step of our algorithm, the aggregation of the best candidates, can be performed using disjunction.

One optimisation step that can be applied to our learning is to notice *irrelevant concepts* and remove them from the search tree. Irrelevant concepts are those from which no partial or counter-partial definition can be produced, hence they cannot be expanded. An irrelevant concept in SPaCEL is defined as follows:

Definition 9 (Irrelevant concept) *Given a concept learning problem* $LP = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$*, a concept* C *is* irrelevant *if* $cover(\mathcal{K}, C, \mathcal{E}^+) = \emptyset$ *and* $cover(\mathcal{K}, C, \mathcal{E}^-) = \emptyset$*, i.e., it covers no positive and no negative examples.*

2.2 The algorithms

The SPaCEL learning algorithm is essentially top-down learning combined with a reduction task. The top-down step is used to solve the sub-problems of the given concept learning problem and the reduction is used to reduce and combine the sub-solutions into an overall solution. The former is performed by the downward refinement operator and a combination strategy, while the latter uses a set coverage algorithm to choose the best partial definitions and disjunction to form the overall solution.

Algorithm 1 describes the main loop of our learning algorithm, the *reducer*. It chooses the best concepts (i.e., those with highest score, based on the search heuristic described in Definition 7) 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. Our reducer is the analogue of the cover removal step in ILP in that it tries to remove redundancies.

The *specialisation*, which performs the refinement and evaluation of the concepts assigned by the learning algorithm, is described in Algorithm 2. It refines the given concept ($\rho_\sqcap(C)$) and evaluates the result ($cover(\mathcal{K}, C, \mathcal{E}^+)$ and $cover(\mathcal{K}, C, \mathcal{E}^-)$). Irrelevant concepts are removed, as no partial definition or counter-partial definition can be computed from them. Then, the specialisation finds new partial definitions, counter-partial definitions and descriptions from the refinements. In practice, new descriptions are checked for redundancy before being evaluated and added to the search tree, as the same description can be generated from different branches of the search tree.

The sets of new descriptions, partial definitions and counter-partial definitions produced by the Specialise algorithm are used to update the corresponding data structures and the set of covered positive and negative examples in the learning algorithm, and the new descriptions are combined with the counter-partial definitions to create new partial definitions

Algorithm 1: Symmetric Class Expression Learning Algorithm– SPACEL($\mathcal{K}, \mathcal{E}^+, \mathcal{E}^-, \varepsilon$)

Input: background knowledge $\mathcal{K}$, 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 definition C such that $|cover(\mathcal{K}, C, \mathcal{E}^+)| \geq (|\mathcal{E}^+| \times (1 - \varepsilon))$ and $cover(\mathcal{K}, C, \mathcal{E}^-) = \emptyset$ (*empty set*)

begin
- initialise the search tree $ST = \{\top\}$ `/* ⊤: TOP concept in DL */`
- $cum\_pdefs = \emptyset$ `/* set of cumulative partial definitions */`
- $cum\_cpdefs = \emptyset$ `/* set of cumulative counter-partial definitions */`
- $cum\_cp = \emptyset$ `/* set of cumulative covered positive examples */`
- $cum\_cn = \emptyset$ `/* set of cumulative covered negative examples */`
- `/* cf. Def. 6 for the meaning of the OR operator */`
- **while** $|cum\_cp| < (|\mathcal{E}^+| \times (1 - \varepsilon))$ **or** $|cum\_cn| < |\mathcal{E}^-|$ **do**
  - get and remove the 'best' concept B from ST `/* see text */`
  - $(pdefs, cpdefs, descriptions) = \text{SPECIALISE}(B, \mathcal{E}^+, \mathcal{E}^-)$ `/* cf.Alg.2 */`
  - $cum\_pdefs = cum\_pdefs \cup pdefs$
  - $cum\_cpdefs = cum\_cpdefs \cup cpdefs$
  - $cum\_cp = cum\_cp \cup \{e \mid e \in cover(\mathcal{K}, P, \mathcal{E}^+), P \in pdefs\}$
  - $cum\_cn = cum\_cn \cup \{e \mid e \in cover(\mathcal{K}, P, \mathcal{E}^-), P \in cpdefs\}$
  - `/* online combination, see Sec.2.3 */`
  - **foreach** $D \in descriptions$ **do**
    - `/* if D can be ''corrected'' by existing cpdefs */`
    - **if** $(cover(\mathcal{K}, D, \mathcal{E}^-) \setminus cum\_cn) = \emptyset$ **then**
      - `/* combine D with cpdefs if possible */`
      - $candidates = \text{COMBINE}(D, cum\_cpdefs, \mathcal{E}^-)$ `/* cf.Alg.3 */`
      - $new\_pdef = D \sqcap \neg(\bigsqcup_{A \in candidates}(A))$ `/* new partial def. */`
      - $cum\_pdefs = cum\_pdefs \cup new\_pdef$
      - $cum\_cp = cum\_cp \cup cover(\mathcal{K}, new\_pdef, \mathcal{E}^+)$
    - **else**
      - $ST = ST \cup \{D\}$
- `/* explore more partial definitions to satisfy completeness */`
- **if** $|cum\_cp| < (|\mathcal{E}^+| \times (1 - \varepsilon))$ **then**
  - **foreach** $D \in ST$ **do**
    - $candidates = \text{COMBINE}(D, cum\_cpdefs, \mathcal{E}^-)$
    - $new\_pdef = D \sqcap \neg(\bigsqcup_{A \in candidates}(A))$
    - $cum\_pdefs = cum\_pdefs \cup new\_pdef$
- **return** REDUCE*(cum_pdefs)* `/* for description, see text */`

where possible, a form of theory learning. Concepts that have been refined can be scheduled for further refinements.

The refinement operator given in Definition 5 is infinite (i.e., it can continue to expand the tree without limit), but in practice each refinement step is monotonic and bounded, as it is only allowed to generate descriptions of at most a pre-defined length (see Definition 8). 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 by Lehmann and Hitzler (2010).

When the learning algorithm reaches a sufficient degree of completeness, it stops and reduces the partial definitions to remove the redundancies using the REDUCE function, which is essentially a set coverage algorithm: given a set of partial definitions $\mathcal{X}$ and a set of positive examples $\mathcal{E}^+$, it finds a subset $\mathcal{X}' \subseteq \mathcal{X}$ such that $\mathcal{E}^+ \subseteq \bigcup_{D \in \mathcal{X}'} (\mathit{cover}(\mathcal{K}, D, \mathcal{E}^+))$. The solution returned by the algorithm is a disjunction of the reduced partial definitions. As an alternative, a set of partial definitions could be returned instead. This may be useful in some contexts, such as making the result more readable. The reduction algorithm may be tailored to meet particular requirements, such as the shortest definition or the smallest number of partial definitions. The combination of descriptions and counter-partial definitions that is given in Algorithm 1 is one of the combination strategies implemented in our evaluation. This strategy is called the *on-the-fly* combination strategy; it gave the best performance in our evaluation (see Section 3.2). A description of the various combination strategies we tried is given in Section 2.3.

Algorithm 2: Specialisation algorithm – SPECIALISE$(C, \mathcal{K}, \mathcal{E}^+, \mathcal{E}^-)$

Input: a description C and a concept learning problem $LP = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$.

Output: a triple consisting of a set of partial definitions $\mathit{pdefs} \subseteq \rho_\sqcap(C)$; a set of counter-partial definitions $\mathit{cpdefs} \subseteq \rho_\sqcap(C)$; and a set of new descriptions $\mathit{descriptions} \subseteq \rho_\sqcap(C)$ such that $\forall D \in \mathit{descriptions} : D$ is not irrelevant and $D \notin (\mathit{pdefs} \cup \mathit{cpdefs})$, in which $\rho_\sqcap$ is the refinement operator defined in Definition 5.

1 **begin**
2 $\quad \mathit{pdefs} = \{D \in \rho_\sqcap(C) \mid \mathit{cover}(\mathcal{K}, D, \mathcal{E}^+) \neq \emptyset \ \wedge \ \mathit{cover}(\mathcal{K}, D, \mathcal{E}^-) = \emptyset\}$
3 $\quad \mathit{cpdefs} = \{D \in \rho_\sqcap(C) \mid \mathit{cover}(\mathcal{K}, D, \mathcal{E}^+) = \emptyset \ \wedge \ \mathit{cover}(\mathcal{K}, D, \mathcal{E}^-) \neq \emptyset\}$
4 $\quad \mathit{descriptions} = \{D \in \rho_\sqcap(C) \mid \mathit{cover}(\mathcal{K}, D, \mathcal{E}^+) \neq \emptyset \ \wedge \ \mathit{cover}(\mathcal{K}, D, \mathcal{E}^-) \neq \emptyset\}$
5 $\quad$ **return** *(pdefs, cpdefs, descriptions)*

Algorithm 3 describes the *combination algorithm* that is used to combine the descriptions and counter-partial definitions to construct new partial definitions. This is basically a set coverage algorithm. One of the smallest sets of counter-partial definitions that together cover all negative examples covered by the given expression will be returned. This set of counter-partial definitions is then used to correct the given expression.

Algorithm 3: Combination algorithm – Combine(C, *cpdefs*, $\mathcal{E}^-$)

Input: a description C, a set of counter-partial definitions *cpdefs* and a set of negative examples $\mathcal{E}^-$

Output: a set *candidates* $\subseteq$ *cpdefs* such that *cover*$(\mathcal{K}, C, \mathcal{E}^-) \subseteq \bigcup_{P \in candidates}$(*cover*$(\mathcal{K}, P, \mathcal{E}^-)$)

```
1  begin
2      candidates = ∅                    /* candidate counter-partial definitions */
3      cn_c = cover(K, 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 (cover(K, D, E⁻) ∩ cn_c) ≠ ∅ then
8              candidates = candidates ∪ D
9              cn_c = cn_c \ cover(K, D, E⁻))
10     if cn_c ≠ ∅ then
11         return ∅                                  /* return the empty set */
12     else
13         return candidates
```

Completeness of the algorithm

The refinement operator used in our algorithm is based on the refinement operator described by Lehmann (2010). In that paper a proof of completeness over the $\mathcal{ALC}$ language is given, i.e., any concepts in the $\mathcal{ALC}$ language can be produced by this operator after a finite number of refinement steps.

Due to our learning strategy, our refinement operator does not generate negation and disjunction. As a result, our refinement operator itself is not complete. However, negation and disjunction can be created by the reduction and combination steps of the algorithm. The combination step uses negation and conjunction to correct an expression with counter-partial definitions (see Algorithm 3 and Definition 4) whereas the reduction step can generate disjunctions. Clearly, the completeness of our algorithm depends upon the combination strategy that produces the negations. Three combination strategies are proposed in Section 2.3, and our learning algorithm is complete when the *on-the-fly* and *delayed* combination strategies are used; completeness is not guaranteed for the *late* combination strategy.

2.3 Counter-partial definitions combination strategies

Counter-partial definitions are combined with expressions in the search tree to create new partial definitions where possible, producing expressions of the form $C \sqcap \neg(\sqcup_{D \in \mathcal{X}} D)$ (for expression C and a set of counter-partial definitions $\mathcal{X}$). The combinability of an expression with respect to a set of counter-partial definitions is given in Definition 4. The combination step can be performed at several stages of the learning algorithm, and different choices have different effects on the learning result. We describe three combination strategies. An evaluation of these strategies is given in Section 3.2.

2.3.1 Late combination

In this strategy, the algorithm maintains sets of partial definitions and counter-partial definitions separately. When all positive or negative examples are covered (or another termination condition is reached, such as maximum time allowed, or a sufficient percentage of examples covered), it will combine descriptions from the search tree and the set of counter-partial definitions.

Since the combination is performed after the learning stops, this strategy may provide a better combination, i.e., it may have better choices of counter-partial definitions to use for the combination. However, for concept learning problems in which both positive and negative examples use negation, 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 a maximum time timeout is set, then the combination will be made when the timeout is reached in order to find the definition. Otherwise, the algorithm will not terminate.

For example, the concept learning problem shown in Figure 2 may cause this strategy to run out of memory without finding an accurate concept if there is no timeout or no noise is allowed. To cover all positive examples, we need a class expression with negation as follows:

$$\text{Bat} \sqcup (\text{Bird} \sqcap \neg\text{Penguin})$$

and to completely cover all negative examples, the following class expression is needed:

$$\text{Penguin} \sqcup (\text{Mammal} \sqcap \neg\text{Bat})$$

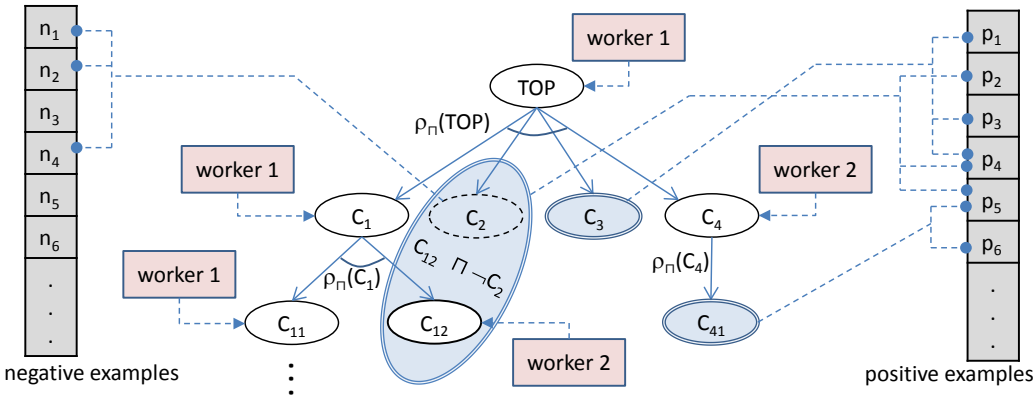
To produce the definition for positive examples, the counter-partial definition `Penguin` needs to be combined with the expression `Bird`, which cannot be performed directly due to the removal of negation from the refinement operator. However, in this strategy, the combination is only called when all positive examples or negative examples are covered, and in the example this condition is never met, as the expression $\neg$`Bat` cannot be produced.

2.3.2 On-the-fly combination

This strategy is used in Algorithm 1. When a new description is generated from the refinement or an existing description is revisited, it is combined with the existing counter-partial definitions if possible. This strategy can avoid the termination problem discussed in the *late combination* strategy because negation is used in the combination. The evaluation suggests that this strategy, overall, gives the best performance and the smallest search tree. However, the final results can be optimised since a counter-partial definition may be combined with many expressions and thus the final definition is unnecessarily long. For example, in the concept learning problem described in Figure 2, when the description `Bird` is generated, there is no counter-partial definition to combine it with. However, when it is revisited for refinement, it will be combined with the new counter-partial definition `Penguin` to produce a partial definition `Bird`$\sqcap$ $\neg$`Penguin`.

This strategy seems to produce similar results to the original refinement by Lehmann and Hitzler (2010), which can generate negation. However, they are essentially different. In our approach, the negation is only used for counter-partial definitions to remove negative examples from the description in the search tree, while negation may be applied for any description in the original refinement operator. In addition, in our approach, the counter-

Figure 4: Top-down learning in SPaCEL with multiple workers.



partial-definitions are stored to be reused to "correct" the new descriptions in the search tree, while the original refinement may regenerate the same negated description.

2.3.3 Delayed combination

This strategy lies inbetween the previous two, and checks 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 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 algorithms times out.

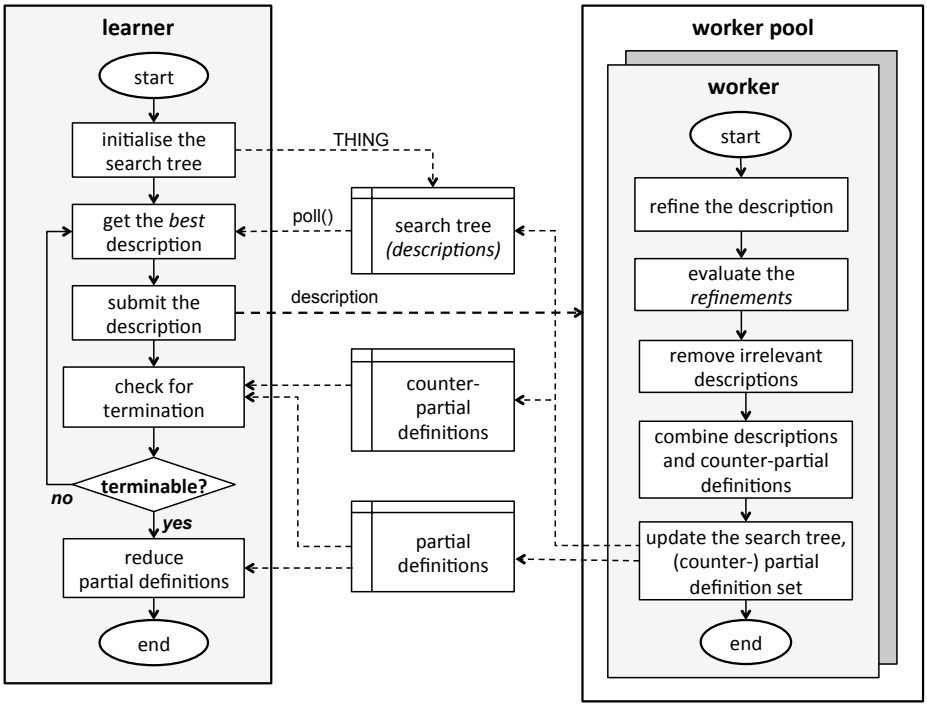
This strategy may help to prevent the problem of the late combination strategy, and it may return better combinations in comparison with the *on-the-fly* strategy because it inherits the advantage of the *late combination* strategy. However, as the combinable descriptions in this strategy are not combined until the learning terminates, the search tree is likely to be larger than that of the on-the-fly strategy.

2.4 Parallelisation of the learning algorithm

One benefit of the symmetric approach is that it is inherently parallel: several expressions (branches) in the search tree can be processed (refined and evaluated) in parallel using multiple *workers* to find the partial definitions. A central *reducer*, which is part of the learner, controls the management of partial and counter-partial definitions produced by the workers until the termination condition described in Definition 6 is met. Then it performs the reduction and aggregation steps to create a final definition. The parallel exploration of the search tree by multiple workers is illustrated in Figure 4.

Our organisation of learning tasks described above follows the idea of the map-reduce architecture (Dean and Ghemawat, 2008), which enables the *parallel divide and conquer* strategy. Here, we organise our algorithm architecture into two parts as shown in Figure 5. The computationally heavy parts, including refinement and evaluation of concepts, are

Figure 5: Implementation architecture. Note that the combination step may be performed by the workers or the reducer, depending upon the combination strategy (see Section 2.3).



done by the multiple workers. The combining of descriptions and counter-partial definitions is also performed on the worker side.

3. Evaluation

In order to evaluate the Symmetric Class Expression Learning approach of SPaCEL, we compared it to CELOE and ParCEL using 10-fold cross validation with 15 data sets. These are summarised in Section 3.1, using the search tree size, predictive accuracy and learning time, and the length of the definitions learnt as the metrics for comparison. CELOE is one of the description logic learning algorithms in the DL-Learner framework. It is well-evaluated and has been compared with many other learning algorithms. Therefore, using it for comparison enables an implicit comparison with other learning algorithms; the evaluations by Hellmann (2008); Lehmann (2010) suggest that, overall, CELOE produced better results than the comparators. The implementation of CELOE that we used is publicly available at `https://github.com/tcanvn/SParCEL/`.

Before performing the comparison, we used the same evaluation method to compare the three evaluation strategies described in Section 2.3, meaning that only one of them (the on-the-fly strategy) was used in the comparison experiments.

3.1 Experimental data sets

To ensure the evaluation covers as much as possible of the space of cases, which helps to archive a thorough assessment of the learning algorithms, we chose data sets that vary in: i) size of knowledge base and examples, ii) the length of definitions, and iii) the amount of noise. In some cases, (i) was achieved by using subsets of a particular data set.

The data sets are divided into three groups. The first group includes 7 learning problems that have low to medium complexity and on which all algorithms in the experiment can find solutions on the training set without timing out in the 10 minutes we allowed. The second group contains high to very high complexity learning problems; all algorithms in the experiment can also find accurate solutions on the training set without timing out. The third contains leaning problems for which one of the learning algorithms could not find accurate solution on the training set (i.e. timeout occurred). The complexity of the learning problems is approximately estimated based on the definition length produced by CELOE: i) low: (0, 8]; ii) medium: (8, 15]; iii) high: (15, 20]; and very high: longer than 20. The reason for treating problems learnt with and without timeout separately is that some metrics cannot be compared if the learning algorithm cannot find the solution (e.g. learning time, search space size).

The list of data sets used and their properties are summarised in Table 1, and an overview of them is given next. We use Manchester OWL Syntax (Horridge and Patel-Schneider, 2009) to describe the learning problems' expected definitions.

Moral This data set was first introduced by Wogulis (1994) and is available at the University of California Irvine (UCI) machine learning repository[2] (Frank and Asuncion, 2010). It is intended to be used to learn definitions of harm-doing activities and contains concepts and observations related to the classification of activities as 'guilty' and 'not guilty' through a set of sub-categories such as 'blameworthy' and 'justified'.

Poker A description of 10 classes of poker hands (5 cards drawn from a standard deck) based on 10 predictive attributes (suit and rank of each card). This data set can be customised into binary-class learning and is available at the UCI repository[3].

Family A set of data sets containing observations about family relations such as aunt, brother, cousin, daughter, father, grandson and uncle (Richards and Mooney, 1995). This data set can be found in many repositories. In our evaluation, we use the data set distributed with the DL-Learner package (which can be found in the DL-Learner repository[4]). Each family relation can be treated as a separate learning problem, and we use several of them. We also use the `Forte Uncle` problem used by Lehmann (2007) and available in the DL-Leaner repository.

CarcinoGenesis This data set is used to learn the structure of chemical compounds and bioassays that may cause cancer based on data from the US National Toxicology Program. It was transformed into logic programming and used for evaluation of ILP algorithms by many authors such as Bahler (1993); Srinivasan et al. (1997) and was

2. http://archive.ics.uci.edu/ml/datasets/Moral+Reasoner
3. http://archive.ics.uci.edu/ml/datasets/Poker+Hand
4. https://github.com/AKSW/DL-Learner

transformed into the OWL ontology format and used to evaluate the description logic learning algorithms in the DL-Learner framework (Lehmann, 2010)[5]. It is particularly noisy, which makes it a challenging problem in machine learning for both symbolic and sub-symbolic learning approaches.

MUSE Dataset We employ use case 1 of a set of context-aware smart home simulations by Lyons et al. (2010) for normal and abnormal living behaviours. The data set is available in the SPaCEL repository[6].

ILPD Dataset A data set based on the liver function test records of patients collected from the North East of Andhra Pradesh, India (Indian Liver Patient Dataset). Each record comprises patient information such as age and gender and results of liver function tests including total Bilirubin, Albumin, and total protein, and the target is an expert-assigned label of cancerous or non-cancerous. This data set is also available in the UCI repository[7].

MUBus Dataset A data set developed to demonstrate the SPaCEL algorithm, it is based on a bus timetable that varies according to conditions such as weekday/weekend, university holidays, and school holidays. Data was generated by sampling at 5 minute intervals. Sub-datasets can be created by restricting the sampling times, such as "MUBus12 [07:00 - 09:20]" (shortened as MUBus-1), "MUBus12 [07:00 - 12:00]" (shortened as MUBus-2) and "MUBus12 [07:00 - 21:30]" (shortened as MUBus-3). This data set is available in the SPaCEL repository; for more information about it, see (Tran, 2013).

The timetable makes surprising complicated rules, such as the following description for a weekend bus:

```
(NOT Holiday1) AND ((hasMinute VALUE 0 and hasHour VALUE 9) OR
  (hasMinute VALUE 20 AND
    (hasHour VALUE 10 OR hasHour VALUE 14 OR hasHour VALUE 16)) OR
  (hasMinute VALUE 40 AND
    (hasHour VALUE 11 OR hasHour VALUE 13 OR hasHour VALUE 15)))
```

3.2 Comparison of Combination Strategies

As was described in Section 2.3, three combination strategies were implemented: *late*, *delayed* and *on-the-fly*. We chose to compare these strategies in order to choose one as the most useful for SPaCEL. Table 2 shows the experimental results. It can be seen that the accuracies achieved are similar across all three, but the learning times and search space sizes differ. For example, the late combination strategy did not terminate in some cases: for the MUBus-2 data set the learner was interrupted by timeout after 10 minutes. However, by applying the algorithm after the learner was interrupted, a definition with 100% accuracy on the training data set was produced. This means the solution existed implicitly, but the

5. http://svn.code.sf.net/p/dl-learner/code/trunk/examples/carcinogenesis/
6. https://github.com/tcanvn/SParCEL/
7. http://archive.ics.uci.edu/ml/datasets/ILPD+(Indian+Liver+Patient+Dataset)

Table 1: Properties of the evaluation data sets. OPs and DPs are short for Object Properties and Data Properties respectively. Number of examples is given in the form positive/negative examples.

No	Dataset	Classes	OPs	DPs	Assertions			Examples
					Class	OP	DP	
1.	Moral	43	0	0	4646	0	0	102/100
2.	Forte uncle	3	3	0	86	251	0	23/163
3.	Poker	4	6	0	374	1080	0	4/151
4.	Carcino-Genesis	142	4	15	22,372	40,666	11,185	182/155
5.	ILPD	4	0	10	976	0	1952	323/165
6.	UCA1 (MUSE)	30	4	11	300	200	200	73/77
	Family data set	4	4	0	606	728	0	-
7.	Aunt	-	-	-	-	-	-	41/41
8.	Brother	-	-	-	-	-	-	43/30
9.	Cousin	-	-	-	-	-	-	71/71
10.	Daughter	-	-	-	-	-	-	52/52
11.	Father	-	-	-	-	-	-	60/60
12.	Grandson	-	-	-	-	-	-	30/30
13.	Uncle	-	-	-	-	-	-	38/38
	MUBus12 Dataset							
14.	[07:00-09:10]	25	0	12	2675	0	32110	383/2292
15.	[07:00-12:00]	25	0	12	6314	0	75766	670/5643
16.	[07:00-21:30]	25	0	12	17128	0	205534	1250/15877

learner was not able to compute it. To make sure that the learner was not terminated too early, the experiment was rerun for 3 hours, but the learner was still not able to find the solution on the training data set. This demonstrates the disadvantage of this strategy.

Table 2: Combination strategies experimental result (*means ± standard deviations of 10 folds*).

Metric	**Late**		**Delayed**		**On-the-fly**	
UCA1						
Learning time (s)	1.01	± 0.33	0.54	± 0.23	0.72	± 0.17
Accuracy (%)	100.00	± 0	100.00	± 0	100.00	± 0
Definition length	20.20	± 0.63	58.60	± 21.31	66.60	± 1.90
No of descriptions	11,137.30	± 2,711.99	5,065.30	± 1,867.64	6,998.70	± 859.74
No of pdef.[8]	1.00	± 0.00	3.40	± 1.27	4.00	± 0.00
Avg.[9] pdef. length	20.20	± 0.63	17.65	± 2.02	16.65	± 0.47
MUBus1						
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
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.	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
MUBus2						
Learning time (s)	int.[10] @600s		90.97	± 73.43	48.46	± 16.56
Accuracy (%)	99.89	± 0.15	99.84	± 0.20	99.78	± 0.23
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	12.00	± 2.79
Avg. pdef. length	177.40	± 41.03	161.52	± 67.40	101.29	± 45.37
MUBus3						
Learning time (s)	int. @900s		566.89	± 224.52	495.94	± 267.67
Accuracy (%)	99.83	± 0.10	97.96	± 1.90	99.72	± 0.40
Definition length	4,120.10	± 1,045.59	3,740.50	± 1,228.56	3,728.00	± 1,598.00
No of descriptions	141,243.00	± 2,185	49,980.00	± 21,778	23,477.00	± 13,897
No of pdef.	18.20	± 0.79	15.70	± 2.83	18.10	± 1.85
Avg. pdef. length	225.38	± 51.33	249.16	± 100.58	205.90	± 85.49

In addition, as the late combination strategy performed the combination at the end, only after all positive or negative examples are covered, its learning times were always longer than other strategies. In theory this might produce more concise solutions than the on-the-fly strategy, as can be seen in the following partial definitions produced by the on-the-fly strategy with the UCA1 data set:

1. activityHasDuration SOME hasDurationValue ≤ 15.5 AND
   (NOT (Activity AND activityHasDuration SOME hasDurationValue ≤ 4.5))

8. Partial definitions
9. Average
10. Interrupted

2. activityHasStarttime SOME Autumn AND
   (NOT (Activity AND activityHasDuration SOME hasDurationValue $\leq$ 4.5)) AND
   (NOT (Activity AND activityHasDuration SOME hasDurationValue $\geq$ 19.5))

3. activityHasStarttime SOME Summer AND
   (NOT (Activity AND activityHasDuration SOME hasDurationValue $\leq$ 4.5)) AND
   (NOT (Activity AND activityHasDuration SOME hasDurationValue $\geq$ 19.5))

4. activityHasDuration SOME hasDurationValue $\geq$ 4.5 AND
   activityHasStarttime SOME Spring AND
   (NOT (Activity AND activityHasDuration SOME hasDurationValue $\geq$ 19.5))

In contrast, the delayed strategy found:

activityHasDuration SOME (hasDurationValue $\geq$ 4.5 AND
hasDurationValue $\leq$ 19.5) AND
(NOT (activityHasStarttime SOME Winter AND
activityHasDuration SOME hasDurationValue $\geq$ 15.5))

The difference is between a large number of short expressions and a smaller number of long ones; usually the delayed strategy produced shorter descriptions overall. However, the on-the-fly strategy dominated in most aspects, particularly the learning time and search space covered. Our hope that the delayed strategy would combine the advantages of both of the others was not bourne out, and we therefore chose to use the on-the-fly strategy for the experimental comparison of SPaCEL with other algorithms.

3.3 Comparison Experiment

We measured search tree size (Table 3), predictive accuracy (Table 4), learning time (also Table 4), and definition length (Table 6) for all 15 data sets using CELOE, PaRCEL and SPaCEL. This section reports and discusses the results of this experiment.

The search tree size reported in this experiment is the total number of all descriptions that are inserted into the search tree, including the irrelevant descriptions (the partial and counter-partial definitions which are removed later by the algorithm). If an algorithm can find the solution, the result reported is the search tree size after the learning algorithm has terminated. Otherwise, it is the search tree size at the time when the timeout has occurred. In this case, the comparison should be treated with caution as it depends upon the timeout assigned for the learning algorithm. Only learning problems on which at least one of the algorithms found an accurate definition on the training set were considered. Therefore, the results for the CarcinoGenesis and ILDP learning problems are not reported, since all three algorithms timed out.

In the first group of learning problems, SPaCEL produced smaller search trees than both CELOE and ParCEL for 4 out of 7 learning problems. However, the definitions of these learning problems were short and therefore the search tree sizes were usually small. Consequently, they were sensitive to the parallelisation: as the search tree in our algorithm is expanded by multiple workers, the solution may be found by one of the workers while the other workers are still processing the search tree. The search tree may be therefore

Table 3: Search tree size (*means* ± *standard deviations of 10 folds*). The underlined values are the search tree size after the *timeout* has occurred. The statistical significance was tested using t-tests: bold values are statistically significantly higher (at the 95% confidence level) than other values; unformatted values are statistically significantly lower than other values; bold italic values are statistically significantly lower than the bold values, and statistically significantly higher than the unformatted values.

Problem	**CELOE**		**ParCEL**		**SPaCEL**	
Low to medium complexity learning problems without timeout						
Moral	540.5	± 16.9	**33.3**	± **11.0**	**223.4**	± **364.2**
Forte	64,707.9	± 33,641.9	***859.5***	± ***251.2***	**174.1**	± **108.2**
PokerStraight	**1,090.8**	± **12.5**	14,204.8	± 2,847	***2,105.0***	± ***1,217.7***
Brother	***37.0***	± ***0***	104.4	± 92.6	**18.4**	± **9.1**
Daughter	***21.0***	± ***0***	111.3	± 110.2	**18.1**	± **5.9**
Father	***29.0***	± ***0***	82.9	± 63.8	**23.0**	± **7.0**
Grandson	**80.5**	± **3.0**	1,867.5	± 1,519.3	***125.3***	± ***25.3***
High to very high complexity learning problems without timeout						
Aunt	85,883.8	± 67,328.8	***7,023.4***	± ***2,912.1***	**2,127.8**	± **1,160.9**
Cousin	***20,331.6***	± ***233.6***	35,484.4	± 39,055.4	**6,761.1**	± **722.9**
Uncle	541,081.8	± 0	***6,332.5***	± ***3,247.9***	**2,400.0**	± **551.2**
Learning problems with at least 1 timeout						
UCA1	1,465,263.2	± 11,515.5	28,676.0	± 14,935.2	**6,998.70**	± **859.7**
MUBus-1	161,832.2	± 3,054.5	643,401.5	± 139,303	**2,130.1**	± **1,179.5**
MUBus-2	79.959.2	± 118.5	879,866.1	± 34,000.4	**8,575.1**	± **4,184.3**
MUBus-3	55,130.4	± 73.7	453,605.1	± 7,565.3	**23,477.0**	± **13,897**

expanded redundantly and thus the reported search tree sizes might be unnecessarily larger than the minimal search tree size required to learn the problem.

In the second and the third groups, SPaCEL always produced the smallest search trees for all learning problems. The search tree sizes generated by the three algorithms for the same learning problem in this group were extremely different. For example, SPaCEL only needed to explore about 2,400 expressions to find the solution for the Uncle learning problem while CELOE had to explore more than 541,081 expressions. Similarly, SPaCEL found a solution after exploring about 2,130 expressions, while ParCEL needed to explore 643,401 expressions on the MUBus-1 learning problem. CELOE could not find an accurate solution for this learning problem and timed out after 10 minutes. The average search tree size at the time of timeout was about 161,832 expressions.

A t-test rejected the null hypothesis for similarity of search tree size of the algorithms on all learning problems at the 99% confidence level, meaning that the search trees generated by SPaCEL were statistically significantly smaller than CELOE in 12 out of 14 learning problems and than ParCEL for 13 out of 14 learning problems.

With regard to predictive accuracy and learning time, which are reported in Table 4, in general, SPaCEL achieved better predictive accuracy in most learning problems. In the first two groups (10 learning problems) all three learning algorithms achieved very high accuracy, including 100% accuracy in 5 of them. In the remaining 5 learning problems, SPaCEL was statistically significantly more accurate than CELOE for 3 out of 5 and ParCEL for 1 out of 5 problems. There were no learning problems in this group where SPaCEL was statistically significantly less accurate than either of the other algorithms.

Table 4: Learning time and predictive accuracy experimental results summary (*means ± standard deviations of 10 folds*). Result of the statistical significance t-test (at the 95% confidence level) is also included: the bold and highlighted values are statistically significantly *better* than other values; the unformatted values are statistically significantly *worse* than other values; the bold and italic and highlighted values are statistically significantly *worse* than the bold and highlighted values, and statistically significantly *better* than the unformatted values; the underlined values represent the values that are not statistically significantly different from the other values.

Problem	**Predictive accuracy (%)**			**Learning time (s)**		
	CELOE	**ParCEL**	**SPaCEL**	**CELOE**	**ParCEL**	**SPaCEL**
Low to medium complexity learning problems without timeout.						
Moral	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	0.15 ±0.03	**0.02 ±0.01**	**0.03 ±0.02**
Forte	98.86 ±2.27	100.00 ±0.00	100.00 ±0.00	2.60 ±1.64	***0.23 ±0.17***	**0.05 ±0.02**
Poker-Straight	100.00 ±0.00	96.43 ±4.12	98.21 ±3.57	0.36 ±0.71	0.59 ±0.08	**0.32 ±0.18**
Brother	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	0.19 ±0.16	***0.03 ±0.02***	**0.02 ±0.01**
Daughter	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	0.2 ±0.02	**0.03 ±0.03**	**0.02 ±0.01**

Continued on next page

Table 4 – continued

Problem	Predictive accuracy (%)			Learning time (s)		
	CELOE	**ParCEL**	**SPaCEL**	**CELOE**	**ParCEL**	**SPaCEL**
Father	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	0.02 ±0.10	0.03 ±0.03	0.02 ±0.01
Grandson	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	0.08 ±0.07	0.19 ±0.79	0.05 ±0.02
High to very high complexity learning problems without timeout.						
Aunt	96.5 ±0.00	**100.00 ±0.00**	**100.00 ±0.00**	30.01 ±0.02	**0.26 ±0.15**	**0.22 ±0.15**
Cousin	99.29 ±0.00	99.29 ±2.26	**100.00 ±0.00**	3.79 ±0.54	**0.54 ±0.20**	***0.80*** ±***0.28***
Uncle	95.83 ±6.80	98.75 ±3.95	95.42 ±10.84	34.13 ±14.94	***0.29*** ±***0.18***	**0.16 ±0.11**
Learning problems with timeout.						
CarcinoGenesis	53.73 ±4.79	56.05 ±4.30	**60.52 ±6.06**	int.* @2000s	int.* @2000s	int.* @2000s
UCA1	91.42 ±7.01	**100.00 ±0.00**	**100.00 ±0.00**	int.* @2000s	***29.75*** ±***5.77***	**0.72 ±0.17**
MUBus-1	53.61 ±2.45	**99.63 ±0.31**	**99.74 ± 0.36**	int.* @600s	***395.33*** ±***11.83***	**7.52 ±2.37**
MUBus-2	14.35 ±1.10	***97.91*** ±***0.50***	**99.78 ±0.23**	int.* @600s	int.* @600s	**48.46 ±16.56**
MUBus-3	11.34 ±0.06	***95.85*** ±***0.31***	**99.72 ±0.40**	int.* @900s	int.* @900s	**495.94 ±267.67**
ILPD	**76.02 ±2.61**	71.12 ±5.36	72.67 ±8.12	int.* @120s	int.* @120s	int.* @120s
Note: *: *Interrupted by timeout*						

In the last group of problems SPaCEL outperformed CELOE on 5 out of 6 and ParCEL on 3 out of 6 learning problems. The data set MU-Bus is a complex learning problem in which the target definition is very long, as the bus operation time depends upon many conditions. For this data set, SPaCEL outperformed both ParCEL and CELOE. It always found the complete definition on the training set and the accuracy on the test set was always over 99.7%, while CELOE could not find accurate definitions on the training set and the accuracy on the test set was very low, from 11.34% to 53.61%. ParCEL performed better than CELOE and the accuracy was also very high but it was still statistically significantly less accurate than SPaCEL.

For the ILPD data set it appears that CELOE had higher predictive accuracy that SPaCEL, although the difference was not statistically significant. However, as the ILPD and MuBus data sets are unbalanced, it is more appropriate to use balanced accuracy [11] for these learning problems. Table 5 shows the balanced predictive accuracy for ILPD and some other unbalanced learning problems. The balanced accuracy of SPaCEL and ParCEL on the ILDP learning problem were statistically significantly higher than CELOE. The outcome of the statistical significance test on the balanced accuracy of other learning problems did not change.

11. $\textit{balanced accuracy} = \frac{1}{2} \cdot \left(\frac{|\textit{true positive}|}{|\textit{positive examples}|} + \frac{|\textit{true negative}|}{|\textit{negative examples}|} \right)$

Table 5: Balanced predictive accuracy of unbalanced data sets in Table 4. Conventions of the results' representation are similar to that of in Table 4.

Problem	Balanced predictive accuracy (%)					
	CELOE		ParCEL		SPaCEL	
MUBus-1	71.12	± 0.65	**99.68**	**± 0.60**	**99.74**	**± 0.62**
MUBus-2	52.09	± 0.06	***91.86***	***± 3.02***	**99.78**	**± 0.59**
MUBus-3	52.18	± 0.03	***73.07***	***± 1.91***	**99.02**	**± 2.67**
ILDP	64.62	± 4.83	**70.94**	**± 10.87**	**71.73**	**± 12.29**

Our symmetric approach to class expression learning not only increased the predictive accuracy, but also decreased the learning time, although this effect can only be observed for the medium and high complexity data sets. The t-test result on learning time shows that SPaCEL was statistically significantly faster than CELOE in 13 out of 14 problems and than ParCEL for 10 out of 14 learning problems. It was statistically significantly slower than ParCEL for 2 out of 14 problems, both in the first and second groups. The definition lengths in this group were short and the learning times were very small.

The last metric that we evaluated was the length of the definitions learnt; the results can be seen in Table 6. The definition lengths of ParCEL and SPaCEL are reported by the number of partial definitions and the average length of the partial definitions, the total average length of their definitions is the product of those numbers.

Table 6: Definition length of the learning problems (*means ± standard deviations of 10 folds*). The bold values are the definition lengths after the timeout occurred.

Problem	(Partial) definition length			No of partial definitions	
	CELOE	ParCEL	SPaCEL	ParCEL	SPaCEL
Low to medium complexity learning problems without timeout					
Moral	3.00 ±0.00	1.52 ±0.05	1.50 ±0.00	2.10 ±0.32	2.00 ±0.00
Forte	13.50 ±1.00	7.75 ±0.50	8.50 ±0.00	2.00 ±0.00	2.00 ±0.00
PokerStraight	11.70 ±0.68	10.90 ±1.31	19.75 ±2.50	1.70 ±0.68	1.00 ±0.00
Brother	6.00 ±0.00	6.00 ±1.83	6.40 ±0.84	1.00 ±0.00	1.00 ±0.00
Daughter	5.00 ±0.00	5.25 ±1.09	6.20 ±0.63	1.10 ±0.32	1.00 ±0.00
Father	5.00 ±0.00	5.50 ±0.53	5.90 ±0.98	1.00 ±0.00	1.00 ±1.00
Grandson	7.25 ±0.50	7.25 ±0.50	8.00 ±0.00	1.00 ±0.00	1.00 ±0.00
High to very high complexity learning problems without timeout					
Aunt	19.00 ±0.00	8.80 ±0.48	10.10 ±0.97	2.00 ±0.00	2.00 ±0.00
Cousin	23.40 ±2.59	8.50 ±0.00	8.50 ±0.00	2.00 ±0.00	2.00 ±0.00

Continued on next page

Table 6 – continued

Problem	(Partial) definition length			No of partial definitions	
	CELOE	**ParCEL**	**SPaCEL**	**ParCEL**	**SPaCEL**
Uncle	19.00 ±14.94	8.40 ±0.38	10.15 ±1.67	2.00 ±0.00	2.00 ±0.00
Learning problems with timeout					
CarcinoGenesis	4.80 ±0.42	55.87 ±9.52	138.00 ±51.46	72.70 ±3.43	17.00 ±5.74
UCA1	**9.00** ±0.00	12.75 ±0.00	16.65 ±0.47	4.00 ±0.00	4.00 ±0.00
MUBus-1	**12.70** ±0.48	16.99 ±0.42	63.40 ±10.54	15.00 ±1.56	6.30 ±1.25
MUBus-2	**2.00** ±0.00	**16.07** ±0.19	101.29 ±45.37	24.80 ±3.52	12.00 ±2.79
MUBus-3	**2.00** ±0.00	**14.64** ±0.13	**205.90** ±85.49	25.40 ±1.58	18.10 ±1.85
ILPD	**5.80** ±1.69	**8.34** ±0.12	**13.30** ±1.40	42.80 ±1.69	37.00 ±2.06

The experimental results show that ParCEL and SPaCEL produced longer definitions than CELOE for most learning problems. By manually inspecting the results we identified that this was because partial definitions tend to overlap, and that the creation of sub-solutions leads to more specific definitions. It may also be that since CELOE timed out more often, it had not yet managed to produce completely accurate (and necessarily longer) solutions before the learning was terminated.

For the learning problems in the first two groups, where all three algorithms can find the definition for the training sets without timeout, the differences in definition length between the three algorithms were small, and any difference was caused by the overlap between partial definitions. In some cases, this can be shortened using an optimisation strategy. For example, the definitions produced by CELOE and SPaCEL for the Forte Uncle data set are:

- **CELOE:**

  male AND ((married SOME sibling SOME Person) OR
  (married SOME hasSibling SOME hasChild SOME Thing))

- **SPaCEL** (two partial definitions):

  1. hasSibling SOME hasChild SOME Thing AND (NOT female)

  2. married SOME hasSibling SOME hasChild SOME Thing AND (NOT female)

The length of the definition produced by CELOE is 15 and SPaCEL is 19 (length of two partial definitions plus 1 for disjunction). However, at least 3 axioms in the SPaCEL final definition can be reduced by removing the common part among partial definitions, i.e. `AND NOT female`. That means that if the same normal form (Lloyd, 1984; Gabbay

et al., 1998) is applied for both CELOE and SPaCEL, the difference between their lengths can be reduced. Moreover, `NOT female` can be replaced by `Male` if `male` and `female` are declared as disjoint and jointly exhaustive properties. This is the idea of *optimisation* and *simplification* in description logic (Horrocks and Patel-Schneider, 1999; Baader et al., 2010). Currently, this idea has not yet been implemented in our algorithm. However, rather than applying optimisation and simplication to the definition constructed by combining multiple partial definitions using conjunction, breaking down a long definition into several smaller partial definitions (as in SPaCEL) may help to make the definition more readable, particularly when the definition is long.

For the learning problems on which at least one of the learning algorithms cannot find an accurate definition on the training set, i.e. timeout occurred in the experiments, definitions produced by SPaCEL are significant longer than those of CELOE. An interesting comparison is seen in the definitions produced for the UCA1 data set, which is complex, but noiseless:

- **CELOE**:

  activityHasDuration SOME (hasDurationValue $\geq$ 4.5 AND
  hasDurationValue $\leq$ 21.5)

- **ParCEL**:

  1. activityHasDuration SOME
     (hasDurationValue $\geq$ 4.5 AND hasDurationValue $\leq$ 15.5)

  2. activityHasDuration SOME
     (hasDurationValue $\geq$ 15.5 AND hasDurationValue $\leq$ 19.5) AND
     activityHasStarttime SOME Spring

  3. activityHasDuration SOME
     (hasDurationValue $\geq$ 15.5 AND hasDurationValue $\leq$ 19.5) AND
     activityHasStarttime SOME Summer

  4. activityHasStarttime SOME Autumn AND activityHasDuration SOME
     (hasDurationValue >= 4.5 AND hasDurationValue <= 19.5)

- **SPaCEL** (late combination):

  activityHasDuration SOME (hasDurationValue $\geq$ 4.5 AND
  hasDurationValue $\leq$ 19.5) AND (NOT (activityHasDuration SOME
  hasDurationValue $\geq$ 15.5 AND activityHasStarttime SOME Winter))

Obviously, the short definition (length 9) produced by CELOE does not fully define the positive examples (both training accuracy and predictive accuracy were below 100%). Meanwhile, ParCEL produced a far longer definition (length 51) that describes the positive examples accurately (both training and predictive accuracy were 100%). The definition produced by SPaCEL for this data set better demonstrates the idea of using the symmetric learning approach and exceptions in learning, since it is a combination of an expression:

activityHasDuration SOME
(hasDurationValue $\geq$ 4.5 AND hasDurationValue $\leq$ 19.5)

and a counter-partial definition:

activityHasDuration SOME
(hasDurationValue $\geq$ 15.5 AND activityHasStarttime SOME Winter)

This definition is shorter than the definition produced by ParCEL and it still describes the positive examples accurately.

4. Related Work

4.1 Description logic learning

Learning in description logic is basically a search problem in which the search tree is often dynamically generated by a refinement operator. In contrast to other algorithms, SPaCEL uses a top-down approach to learning both positive and negative patterns. Only YinYang (Iannone et al., 2007) has particularly considered negative patterns previously, and they took a bottom-up approach to them, despite learning top-down for the positive patterns. The bottom-up approach creates concepts by joining most specific concepts created for positive examples using disjunction, which can create large concept definitions that are not truly intentional.

Top-down learning has been used by several algorithms reported in the literature, such as Badea and Nienhuys-Cheng (2000), which uses a refinement operator designed for the $\mathcal{ALER}$ description logic. However, as discussed by Lehmann and Hitzler (2010), this refinement operator cannot be extended to handle more expressive description logic language such as $\mathcal{SRIOQ}(\mathcal{D})$, which is the language on which OWL2 is based.

Lisi and Malerba (2003) proposed an alternative based on the hybrid $\mathcal{AL}$-log language, which combines the $\mathcal{ALC}$ description logic language and Datalog. In Nienhuys-Cheng and De Wolf (1997) an ideal downward refinement operator for the $\mathcal{AL}$-log language was proposed that is based on the notation of $\mathcal{B}$-subsumption. This is a generalisation of the generalised subsumption in the $\mathcal{AL}$-log language and is suitable for the hybrid knowledge representation systems that are constituted by the relational and structural subsystems. It was implemented in the $\mathcal{AL}$-QuIn ($\mathcal{AL}$-log Query Induction) system.

In Esposito et al. (2004); Iannone et al. (2007), a top-down refinement operator for the $\mathcal{ALC}$ language is used in combination with a bottom-up most specific concept (MSC) operator. The essential idea of these studies lies in the concept of *counterfactuals*, that is *errors* within candidate hypotheses. Therefore, for each candidate hypothesis, the learning algorithm uses an MSC operator to find the concept(s) representing errors and remove them from the candidate hypothesis. However, this approach has two disadvantages: i) finding of concept(s) for counterfactuals may be repeated for the same set of errors, and ii) as a result, it tends to generate unnecessarily long concepts.

Some description logic learning algorithms also have been proposed by Lehmann (2010), where two top-down refinement operators have been proposed. One is for the $\mathcal{EL}$ language and the other is for $\mathcal{ALC}$ (although it can be extended to more expressive languages). The latter is the most expressive refinement operator proposed so far. In addition, several learning approaches based on the proposed refinement operators were also developed. Two

interesting algorithms in this framework are Class Expression Learner for Ontology Engineering (CELOE) and OWL Class Expression Learner (OCEL). These algorithms perform very well when compared with other learning approaches (Hellmann, 2008). However, they are sequential algorithms and thus they cannot take advantages of parallel systems such as multi-core processors or cloud computing platforms. In addition, these algorithms focus on generating short descriptions and thus they have another disadvantage: they cannot handle complex learning problems (see Hellmann (2008) and Tran et al. (2012)). Recently, an extension of CELOE has been developed and integrated into the DLLearner framework. This algorithm is the parallelisation of CELOE in which the search tree is expanded by multiple threads. Basically, the learning approach is the same as CELOE, i.e., it looks for a complete definition. Therefore, its performance (learning speed) is improved, but not the capability for learning problems requiring complex (long) concepts.

DL-FOIL (Fanizzi et al., 2008) combines both top-down and bottom-up learning. The top-down step finds a set of *correct concepts* such that each of them correctly defines a subset of the positive examples. Then, the bottom-up step computes a complete concept that defines all positive examples. This approach handles complex concepts better than the top-down or bottom-up approach alone. However, it produces longer concepts in comparison with the approaches proposed by Lehmann (2010). The unnecessarily long learnt concepts are caused by the lack of optimisation in the aggregation step. In addition, like other existing description logic learning algorithms, this approach is serial by nature and thus it cannot take advantage of concurrency.

During the early 90s, a related idea to the overly general descriptions that we use is this paper was proposed by Bain and Muggleton (1990) for first-order theories. Called non-monotonic learning, this approach adopted a specialisation schema based on an existing non-monotonic logic formalism for incremental specialising of over-general beliefs. Although this learning method also employs the negative examples to *correct* the learned beliefs, the use of negative examples in this approach is different from our approach. In Bain and Muggleton's approach, each negative example is intentionally and incrementally used to specialise the learned beliefs. On the other hand, negative examples in our algorithm are mainly used to verify the correctness of the learned beliefs. Their definitions, which are occasionally found in the process of finding the definition for positive examples, are used to correct (specialise) the over-general concepts.

Recent research in description logic learning has focused on the induction instance retrieval cast as a classification problem (Rizzo et al., 2015). They extend the decision tree to describe logic representations (terminological decision trees) by using the nodes to represent conjunctive description logics and left and right branches of the binary tree corresponding to the result of checking some instance against the parent node. In essence, this is a top-down learning approach in which the child nodes are generated by a downward refinement operator.

4.2 Parallelisation in description logic learning

Parallel computing has a history of development since the late 1950s, from multi-processor computer systems to multi-core processors. Parallelisation helps to use computer resources more effectively in order to create fast, efficient scalable algorithms. Currently, parallelisa-

tion can be found in every area of computing, with many parallel frameworks developed, such as Parallel Virtual Machine (PVM) (Sunderam, 2006) and Apache Hadoop (Bhandarkar, 2010).

In logics, parallelisation has been used to develop parallel logic programming languages such as PARLOG (Gregory, 1987) and PEPSys (Ratcliffe and Syre, 1987), allowing parallelised logic deduction. A parallel inductive logic programming approach was proposed by Dehaspe and De Raedt (1995), based on partitioning positive examples into several sets depending on the parallel level of the system and combining the concepts for the partitioned sets at the end. In Matsui et al. (1998), several parallelisation strategies were implemented in FOIL, an inductive logic learning algorithm (Quinlan, 1990). These strategies differ mainly in the dividing strategies, either dividing space among processors (i.e. multiple refinements work in parallel) or partitioning the data and learning the partitions independently, similarly to Dehaspe and De Raedt (1995), although using background knowledge to perform the partition. Our parallel learning approach is similar to the first strategy: we explore the search space in parallel. However, we use a different learning strategy, employing concepts for a subset of positive examples and use a reduction strategy to construct the final concept.

Parallelisation is also used in description logics, but mainly not for reasoning, with only the Deslog reasoner (Wu and Haarslev, 2012), and parallel inferencing algorithms for OWL (Liebig and Müller, 2007; Soma and Prasanna, 2008) being reported to date.

5. Conclusion and Future Work

A symmetric approach to class expression learning has been proposed, where we learn from both positive and negative examples simultaneously. This is motivated by learning scenarios where negative examples can be classified using simple patterns. This is common in practice and our empirical experiments suggest that our approach deals well with this kind of scenario. This approach to class expression learning can deal with other kinds of concept learning problems, as shown in Table 4. For example, the Forte learning problem can be solved by a top-down approach (e.g., CELOE and ParCEL) without using negation. However, this learning problem is solved faster by SPaCEL, which uses the definitions of negative examples, without decreasing the predictive accuracy.

Some current learning algorithms, e.g. CELOE and ParCEL, which were 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 candidate concepts. However, for some data sets with regular exception patterns such as MUBus and UCA1, these algorithms had difficulties in finding the right concept: their learning times were very long in comparison with SPaCEL, which sometimes caused the system to run out of memory before the definition could be found. The most impressive improvements were in the search tree size and learning time. Although SPaCEL often generated longer definitions than other algorithms, there was no over-fitting for the data sets used.

However, the definitions generated by SPaCEL are not optimised. Normalisation and simplification can be used to produce better definitions, i.e. shorter and more readable. This, together with investigations on more data sets, will require further research.

Acknowledgments

Part of this article was written while the first author worked as a researcher at the Vietnam Institute for Advanced Study in Mathematics (VIASM). We gratefully acknowledge the financial support of VIASM for the first author during that time. We also appreciate the constructive comments and suggestions for improving this article made by the anonymous reviewers.

References

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The description logic handbook: Theory, implementation and applications*. Cambridge University Press, 2010.

Liviu Badea and Shan-Hwei Nienhuys-Cheng. A refinement operator for description logics. *Inductive Logic Programming*, pages 40–59, 2000.

Dennis Bahler. The induction of rules for predicting chemical carcinogenesis in rodents. In *Intelligent Systems for Molecular Biology*, pages 29–37. AAAI/MIT Press, 1993.

Michael Bain and Stephen Muggleton. *Non-monotonic learning*. Turing Institute, 1990.

Milind Bhandarkar. MapReduce programming with apache Hadoop. In *Parallel & Distributed Processing (IPDPS) Symposium*, pages 1–1. IEEE, 2010.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Luc Dehaspe and Luc De Raedt. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, page 5, 1995.

Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. *The Semantic Web–ISWC 2004*, pages 441–455, 2004.

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

Andrew Frank and Arthur Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

Dov M. Gabbay, Christopher John Hogger, John Alan Robinson, J. Siekmann, Donald Nute, and Anthony Galton. *Handbook of Logic in Artificial Intelligence and Logic Programming*. Clarendon Press, 1998.

Steve Gregory. Parallel logic programming in PARLOG: The language and its implementation. 1987.

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

Matthew Horridge and Peter F. Patel-Schneider. OWL 2 web ontology language Manchester syntax. *W3C Working Group Note*, 2009.

Ian Horrocks and Peter F. Patel-Schneider. Optimizing description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

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 Verlagesellschaft, 2010.

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

Jens Lehmann. Hybrid learning of ontology classes. *Machine Learning and Data Mining in Pattern Recognition*, pages 883–898, 2007.

Thorsten Liebig and Felix Müller. Parallelizing tableaux-based description logic reasoning. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, pages 1135–1144. Springer, 2007.

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

John Wylie Lloyd. *Foundations of logic programming*. Springer-verlag Berlin, 1984.

Paul Lyons, An C. Tran, H. Joe Steinhauer, Stephen Marsland, Jens Dietrich, and Hans W. Guesgen. Exploring the responsibilities of single-inhabitant smart homes with use cases. *Journal of Ambient Intelligence and Smart Environments*, 2(3):211–232, 2010.

Tohgoroh Matsui, Nobuhiro Inuzuka, Hirohisa Seki, and Hidenori Itoh. Comparison of three parallel implementations of an induction algorithm. In *8th Int. Parallel Computing Workshop*, pages 181–188, 1998.

Shan-Hwei Nienhuys-Cheng and Ronald De Wolf. *Foundations of Inductive logic programming*. Springer, 1997.

J. Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

Michael Ratcliffe and Jean-Claude Syre. A parallel logic programming language for PEPSys. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, volume 1, pages 48–55. Morgan Kaufmann Publishers Inc., 1987.

Bradley L. Richards and Raymond J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

Giuseppe Rizzo, Claudia dAmato, Nicola Fanizzi, and Floriana Esposito. Terminological tree-based models for inductive classification in description logics. *Semantic Web*, 2015.

Ramakrishna Soma and Viktor K. Prasanna. Parallel inferencing for OWL knowledge bases. In *37th International Conference on Parallel Processing (ICPP'08)*, pages 75–82. IEEE, 2008.

Ashwin Srinivasan, Ross D. King, Stephen Muggleton, and Michael JE. Sternberg. Carcinogenesis predictions using ILP. *Inductive Logic Programming*, pages 273–287, 1997.

Vaidy S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: practice and experience*, 2(4):315–339, 2006.

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.

An C. Tran. *Symmetric Parallel Class Expression Learning*. PhD thesis, Massey University, 2013.

James Lee Wogulis. *An approach to repairing and evaluating first-order theories containing multiple concepts and negation*. PhD thesis, University of California at Irvine, Irvine, CA, USA, 1994. UMI Order No. GAX94-12189.

Kejia Wu and Volker Haarslev. A parallel reasoner for the Description Logic ALC. In *Proceedings of the 2012 International Workshop on Description Logics (DL 2012)*, 2012.