Semantic Web 0 (2020) 1–0
IOS Press

Towards a Distributed Refinement Operator for TCT induction

Giuseppe Rizzo *, Claudia d'Amato **, Nicola Fanizzi ***
LACAM – Dipartimento di Informatica
Università degli Studi di Bari "Aldo Moro"
Via Orabona 4, 70125 Bari, Italy

Abstract. The document illustrates how the refinement operator adopted in TCT learning algorithm can be reworked to support parallelism /distributed computation by means proper frameworks. We will consider the case of APACHE SPARK.

1. Basic Notions

In this section we will recall the definition of terminological cluster tree (TCT) and shortly describe how it can be induced from a KB, assuming familiarity with the basics notions on Description Logics [1].

1.1. Terminological Cluster Tree: definition and induction

The model is formally defined as follows:

Definition 1 (terminological cluster tree) *Given a knowledge base $\mathcal{K}$, a* terminological cluster tree *(TCT) is a binary logical tree [2] where:*

- *each node, which stands for a cluster* $\mathbf{C}$ *of individuals, contains a concept description D (defined over the signature of $\mathcal{K}$)*
- *each edge departing from an internal node corresponds to a partition of* $\mathbf{C}$ *in (two) sub-clusters*[1]

The induction of a TCT relies on divide-and-conquer strategy where the concept installed into a node is generated by means an incomplete *refinement operator*[3].

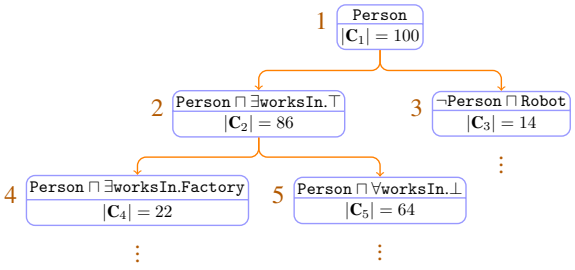

Fig. 1. A fragment of TCT whose nodes are also decorated with the size of the respective cluster of individuals $\mathbf{C}_i$. Intuitively, the concept descriptions D_i in the various nodes could be roughly mapped ($CN_i \equiv D_i$) to the following names, respectively: CN_2 : `Employee`, CN_3 : `Robot`, CN_4 : `Worker`, CN_5 : `Freelance`.

1.2. Refinement Operators

The solution of the concept learning problem can be cast [**?**] as the task of traversing a quasi-ordered space, i.e. one endowed with a reflexive and transitive relationship, searching for a concept definition that satisfies the problem constraints. To this purpose, we will consider a space of concept definitions $\mathcal{L}$ ordered by the subsumption relationship $\sqsubseteq$. Naturally only descriptions that can be built upon the signature of the given KB are considered as well as subsumption must hold w.r.t. its models (i.e. $\mathcal{K} \models D \sqsubseteq D'$). Suitable op-

*Corresponding author. E-mail: giuseppe.rizzo1@uniba.it.
**Corresponding author. E-mail: claudia.damato@uniba.it.
*** Corresponding author. E-mail: nicola.fanizzi@uniba.it.

[1] Noticeable difference with concept hierarchies: for each node in the TCT, its cluster, composed by instances of the concept in the parent node (ideally $\top$ for the root), is bi-partitioned according to the membership w.r.t. the concept in the current node.

erators to traverse $\mathcal{L}$ can be formally defined as follows:

Definition 2 (refinement operators in DL) *Given a knowledge base $\mathcal{K}$ and a quasi-ordered space of concept descriptions $(\mathcal{L}, \sqsubseteq)$*

- *a* downward (specializing) refinement operator *is a mapping $\rho : \mathcal{L} \rightarrow 2^{\mathcal{L}}$ such that*

$$\forall D \in \mathcal{L} \quad \rho(D) \subseteq \{D' \in \mathcal{L} \mid \mathcal{K} \models D' \sqsubseteq D\}$$

- *an* upward (generalizing) refinement operator *is a mapping $\delta : \mathcal{L} \rightarrow 2^{\mathcal{L}}$ such that*

$$\forall D \in \mathcal{L} \quad \delta(D) \subseteq \{D' \in \mathcal{L} \mid \mathcal{K} \models D \sqsubseteq D'\}$$

In the sequel, we will focus on downward refinement. A downward refinement operator ρ may fulfill important properties that are related to its effectiveness, such as [**?**]:

local finitness for each C, $\rho(C)$ is finite and can be computed in a finite time;

properness for all C, D, $D \in \rho(C)$ implies $\mathcal{K} \models C \not\equiv D$;

completeness for all C, D such that $C \sqsubset D$, there is an $E \in \rho^*(D)$ such that $\mathcal{K} \models E \equiv C$ (ρ^* denotes the reflexive transitive *closure* of ρ [**?**]).

Analogous properties are defined for upward operators. A refinement operator that is endowed with all of these properties is defined as **ideal** [**?**]. In particular *completeness* may be important because it implies that any possible refinement can be reached from a general concept.

2. Distributed Implementation of the Specialization Procedure

In the case of TCT induction, the mentioned incompleteness of the refinement operator adopted and limitations on the size of beam of candidates imply that the search algorithm may miss some important features (concepts) that would describe the clusters optimally. Conversely, tuning the algorithm with a large beam sizes makes the approach inefficient. To tackle these issues, we illustrate an approach to reworking the refinement operator benefiting from frameworks for supporting distributed architectures, i.e. *Spark*.

Spark is a distributed processing framework intended for large amounts of heterogeneous data. It provides an API for devising applications that are able to process such kind of data by means a transparent approach with respect to specific file systems and architectures. *Spark* relies on the notion of *resilient distributed datasets* (RDDs), distributed memory abstractions that let programmers perform in-memory computations on large computer clusters. Essentially, an RDD is a read-only, partitioned collection of records. RDDs are created through operations called *transformations* that take an RDD and return a new RDD as their output, and *actions* that return a single value of a given type rather than an RDD. Some examples of such operations are the well known *map & reduce* functions for parallel processing: MAP provides a one-to-one processing of each element contained in the input RDD) and REDUCE processes a RDD containing values of a given type to get a single new value (of the same type).

Algo. 1 reports a new version of the specialization procedure described in Algo. **??** reworked for allowing an implementation on a distributed processing frame-

Algorithm 1 Distributed specialization procedure for *Spark*

```
1  const n: number of candidates {from the configuration}
2  function SPECIALIZE(C, I, CS)
3  input C: concept description
4        I: set of individuals
5        CS: set of concept descriptions
6  output S: set of concept descriptions
7  begin
8     S ← RDD[n] {an RDD of n elements}
9     S ← MAP(S, INITIALIZE(S, C)) {RDD init.}
10    PARALLELIZE(S)
11    S ← MAP(S, GENERATEAREFINEMENT(S, I))
12    return S
13 end
14
15 function GENERATEAREFINEMENT(S,I)
16 input S: RDD
17       I: set of individuals
18 output S′: RDD
19 begin
20    S′ ← RDD[n]
21    for each C ∈ S do {RDD transformation}
22       repeat
23          D ← C ⊓ ADDCONJUNCT()
24          D ← SIMPLIFY(C ⊓ E) {reduce complexity}
25       until r_K(D) ∩ I ≠ ∅ and r_K(¬D) ∩ I ≠ ∅ and
         ¬OVERLAP(C, CS)
26       S′ ← S′ ∪ D
27    return S′
28 end
```

work. Similarly to the original version, the procedure SPECIALIZE takes as its arguments the concept description C to be refined, the set of individuals **I**, and the control set **CS** of concept descriptions. The procedure creates an RDD **S**, which is initialized by the transformation MAP. The high-order function requires a function INITIALIZE[2] as an argument. This function returns a new RDD containing the concept description C that will be refined. After the initialization, the RDD **S** is physically distributed among the available processing units (of a cluster or, if running on a single machine, of the various cores decomposed in multiple threads) through the auxiliary procedure PARALLELIZE[3]. Once the initialization is parallelized, the refinement process starts invoking again the function MAP. In this case, the algorithm passes the function GENERATEAREFINEMENT, which generates for each concept in **S** the (properly simplified) specializations (similarly to Algo. **??**).

3. Experiments

One of the extensions proposed in this work concerns the use of a distributed version of the refinement operator in order to speed up the specialization generation task which represents one of prominent bottlenecks of the approach.

We carried out various tests aiming at determining how the solution implemented on the *Spark* framework could improve the efficiency of this task. To this purpose, we considered both the procedure implementing the new version of the refinement operator (henceforth *Distributed Refinement Operator* - DRO) and the original version (*Single-core Refinement Operator* - SRO) [4] also increasing the size of the beam of candidates: 100, 300, 400, 500, 600, 1000. Also, we ran these procedures using the entire set of individuals in each KB to test the stop condition in the procedures (see Algo. **??** and Algo. 1). We repeated the experiments considering both the original ontologies and the versions obtained applying the SDA.

Fig. 2 illustrates the outcomes (*execution time*) using the SDO and the DRO under SDA; similar trends were observed in the experiments on the original KBS.Throughout the experiments, we noted the DRO was significantly faster than the SRO, with differences spanning from less than 500 ms to more than 2,000,000 ms. We noted that using the SRO in most of the cases the time grew linearly w.r.t. the number of specializations, e.g. see the case of the experiments with MONETARY. This depended on two factors: the complexity, in terms of syntactic length, of the generated concept descriptions and the threshold on the number of individuals used to stop the condition. In particular, the generation of the concepts was biased towards the introduction of new concept names as conjuncts rather than the existential and universal restrictions. This means that there was a limited number of recursive calls of the refinement operator and shorter concepts. As a consequence the stop condition was satisfied earlier w.r.t. the case of concepts involving existential and universal restrictions. Indeed, instances of concept descriptions obtained as a conjunction of concept names are generally easier to find than for concepts involving universal and existential restrictions, due to the sparseness of assertional knowledge concerning roles observed in the KBs. As previously mentioned, the DRO was considerably more efficient: the time required for generating refinements increased less than linearly. In particular, with high dimensional beams, the benefits in terms of efficiency obtained with the distributed solution are noticeable whereas the use of low dimensional beams limited this improvement, due to the inherent overhead (e.g. for the transparent management of the RDD distribution).

A noteworthy case was the one related to the experiments with MUTAGENESIS: the time required by the SRO grew less than linearly also in the experiments with SRO. This was likely due to the effect of the SDA used for the limited number of concepts and their organization into a shallow hierarchy. As a consequence, due to the large number of axioms introduced in the KB, the refinement operator could rapidly check which individuals were either positive or negative instances w.r.t. a specialization, satisfying the stop condition early.

A final remark concerns the line of experiments in which the SDA was not made: they showed that the two operators behave similarly to the cases in which the SDA was made. Even if DRO (resp. SRO) showed a less than linear (resp. linear) increase of the time as larger beams were considered, the lack of disjointness axioms of the original KBs makes hard to find individuals with a definite membership for each specialization, thus delaying the satisfaction of the stop condition. However, it should be clarified that this problem

[2] This is implemented in Java 8 as a *lambda expression*

[3] It can be transparently managed by the underlying framework infrastructure.

Table 1
Ontologies employed in the experiments

Ontology	*DL Language*	*#Concepts*	*#Roles*	*#Individuals*	*#Disj.Axioms*
BIOPAX	$\mathcal{ALCIF}(D)$	74	70	323	85
NTN	$\mathcal{SHIF}(D)$	47	27	676	40
FINANCIAL	$\mathcal{ALCIF}(D)$	60	16	1000	113
GEOSKILLS	$\mathcal{ALCHOIN}(D)$	596	23	2567	378
MONETARY	$\mathcal{ALCHIF}(D)$	323	247	2466	236
DBPEDIA	$\mathcal{ALCHI}(D)$	251	132	16606	11
MUTAGENESIS	$\mathcal{AL}(D)$	86	5	14145	0
VICODI	$\mathcal{ALHI}(D)$	196	10	16942	0

depends on the specific reasoner adopted to make the inferences required by the algorithms.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider (eds), *The Description Logic Handbook*, 2nd edn, Cambridge University Press, 2007.

[2] L. De Raedt and H. Blockeel, Using logical decision trees for clustering, in: *Proceedings of ILP 1997*, N. Lavrač and S. Džeroski, eds, LNAI, Vol. 1297, Springer, 1997, pp. 133–140.

[3] J. Lehmann and P. Hitzler, Concept learning in description logics using refinement operators, *Machine Learning* **78**(1) (2009), 203.

[4] G. Rizzo, C. d'Amato, N. Fanizzi and F. Esposito, Terminological Cluster Trees for Disjointness Axiom Discovery, in: *Proceedings of ESWC 2017, Part I*, E. Blomqvist et al., eds, LNCS, Vol. 10249, 2017, pp. 184–201.

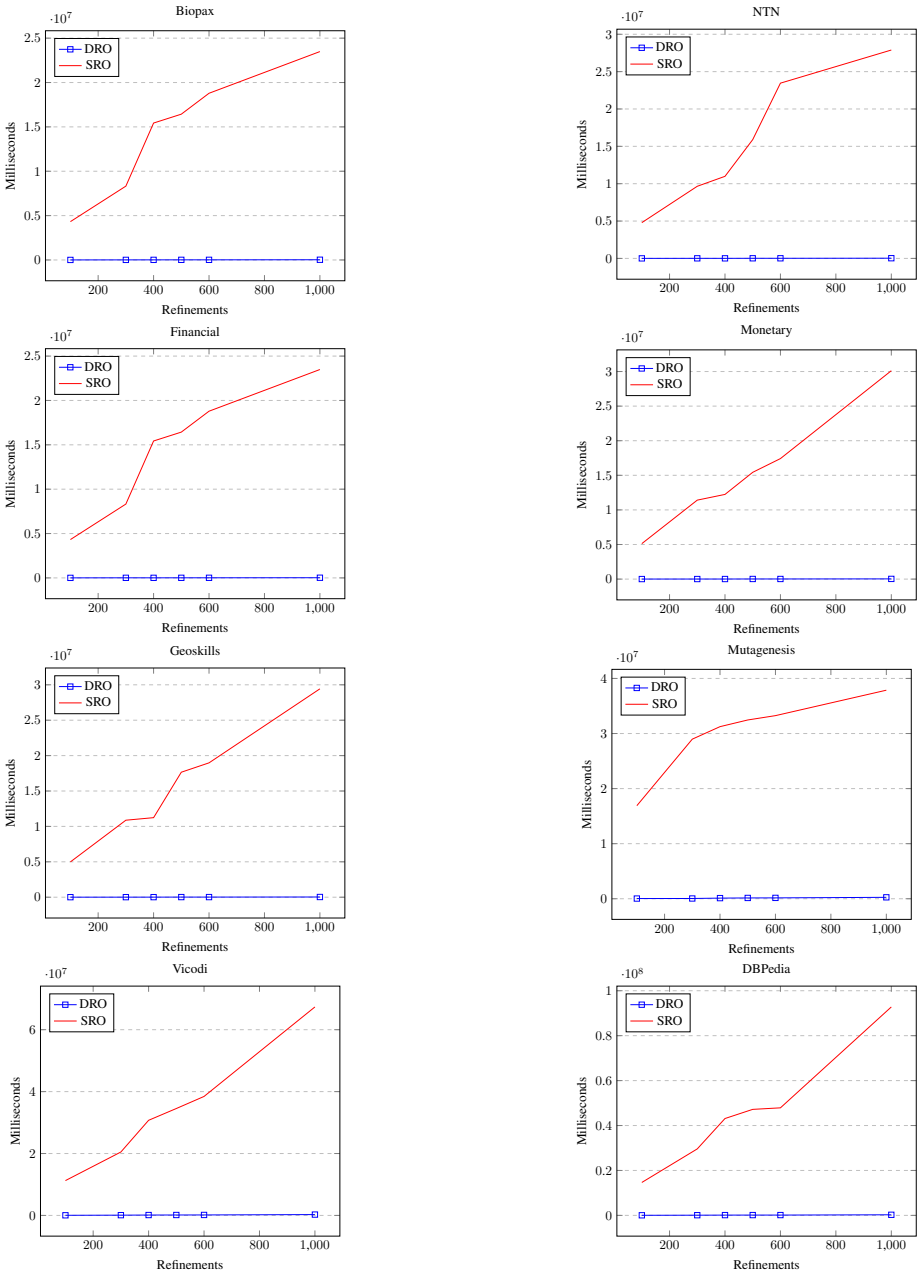

Fig. 2. Efficiency (ms) of the distributed refinement operator compared to the single-core implementation