Learning relations without closing the world (extended abstract)*

Edgar Sommer

GMD (German National Research Center for Computer Science)
Schloss Birlinghoven, 53757 St. Augustin, Germany
email sommer@gmd.de

Abstract. This paper describes LINK, a heuristically guided learner that combines aspects of three major approaches to ILP — LGG, search heuristic and (declarative) structural bias. In the manner of LGG algorithms, LINK generates sets of candidate premise literals by collecting facts about the terms that appear in goal examples. It uses a *linked-enough* heuristic to select amongst these candidates to form hypothesis clauses (conjunctions of literals), and uses structural criteria to select among possible hypotheses in the manner of declarative bias-based systems. This combination — together with a parametrized hypothesis evaluation function — allows LINK to learn in realistic situations where many FOL learners have problems because they are forced to make assumptions about the data: when there are no negative examples, when information is sparse, and when the closed-world assumption cannot or should not be made on examples and/or background.

1 LINK

The premise of this paper is that one cannot always determine a priori whether a given data set is complete or sparse². LINK is an ILP learner based on the idea of *describing* known examples in terms of what is known about their arguments, rather than discriminating between positive and negative examples and making assumptions about what is not known.

A number of facts in a knowledge base can be seen as spanning a graph that represents links between objects (facts' arguments) in the following manner. A graph is initialized with an example fact of the learning goal. It is constructed by collecting all the facts (in the KB) in which one or more of the example's arguments appear. Facts' arguments are nodes, and the predicates in which they appear are arcs. The graph initially shows all the facts known about the example's arguments, but may contain additional nodes for which this is not the case. This graph represents the set of candidates that can be used to form a hypothesis. An LGG algorithm would construct a second graph (fact chain) — initialized by a second example — at this point and generalize the two. LINK, on the other hand, uses the linked-enough heuristic to select some connected part of it (Sec. 1.1) and treat this as a hypothesis (Sec. 1.2).

If LINK finds no good hypothesis using candidates from the current graph, it can be augmented by adding facts about nodes not yet fully described (i.e. expanded).

* Longer version including comparison and empirical results is available from the author.

² Missing positive and/or negative examples of the goal clause; missing descriptors in the background theory; incompletely described examples.

```

LINK top level while there are uncovered examples of the goal concept
1 select an uncovered example
2 build conjunction with linked-enough seeded by example
3 form hypo from conjunction & test
4 case:
  > accepted  $\Rightarrow$  true, remember covered examples
  > too_specific  $\Rightarrow$  backtrack to linked-enough
  > too_general  $\Rightarrow$  specialize conjunction with linked-enough and go to 3.
end{top level loop}

linked-enough(Conjunction)
  > collect Conjunction arguments (args)
  > while Conjunction not linked-enougha and shorter than c*arity(goal):
    > problem-args = args - linked-args (in specialization: problem-args = args)
    > in current graph, look for a fact (in the following order of preference) that
      a. links problem-args to head-args
      b. links problem-args to problem-args
      c. links problem-args to other args in Conjunction, or
      d. links problem-args to new args
    > if successful, add fact to Conjunction else mark problem-args as constants
      or (in specialization) expand graph around problem-args and try again.
  end{link-up loop}

```

^a This condition does not apply in specialization, since it is already linked-enough.

Fig. 1. The LINK algorithm

1.1 The linked-enough heuristic

The linked-enough heuristic builds a conjunction of facts from the pool defined by the current graph until all arguments are linked enough. An argument is *linked-enough* if it is not free, i.e. it appears in at least two literals of the conjunction. It is linked to the other arguments of the literals it occurs in. LINK collects the 'problem' arguments that do not pass the linked-enough test and for each tries first to find a new conjunct linking it to the current example's arguments, to other arguments otherwise (here giving preference to the other problem arguments). As a last resort, an argument may be marked as a constant if no suitable conjunct is found.

The linked-enough heuristic (Fig. 1) is used both to generate initial hypotheses and to specialize hypotheses that were found to be too general. In the first case, the conjunction consists only of the current example fact, and facts are added until the result is linked enough. In the second case (specialize conjunction in Fig. 1), the linked-enough test is not necessary, but a new conjunct is added from the current graph according to the same preference as above. It is here that LINK may expand the current graph: rather than turn an argument into a constant, it may expand the current graph to include all known facts about this argument, and select among these to link up the conjunction.

An additional structural condition, expressed as a ratio between premise length and goal arity, is also placed on the conjunction before it passes the linked-enough test: it should be no longer than $c \cdot \text{arity}(\text{goal})$, where c 's value is a parameter of

the system³. While this is not strictly necessary to guarantee termination (in a finite knowledge base), it can be used to change what kinds of rules LINK accepts, as well as the direction of heuristic search through the hypothesis space.

1.2 Hypothesis formation and test

The conjunction of facts produced by the linked-enough heuristic is turned into a hypothesis by treating the current example as the conclusion, the other facts as premise, and by systematically turning terms into variables, retaining those marked as constants.

The test procedure then generates all instantiations of the premise that are possible in the current knowledge base⁴. This results in a set H_{total} of goal instances that is used to compute the following values characterizing the hypothesis (KB_{pos} and KB_{neg} are the known positive and negative examples of the goal):

- ▷ $pos = |(KB_{pos} \cap H_{total})|$, i.e. true positives.
- ▷ $neg = |(KB_{neg} \cap H_{total})|$, i.e. false positives.
- ▷ $pred = |(H_{total} - KB_{pos})|$, predicted positive instances.
- ▷ $unc = |(KB_{pos} - H_{total})|$, uncovered positive instances.

Two criteria placing conditions on these values are then used to judge the acceptability of the hypothesis: the **confirmation** criterion and the **pruning** criterion. A hypothesis is **too_general** if it fulfills neither confirmation nor pruning criterion; it is **too_specific** if it fulfills the pruning criterion, and **accepted** if it fulfills the confirmation criterion. A strict confirmation criterion might be $pred=0 \ \& \ neg=0$ stating that a hypothesis is acceptable if it covers no negative examples and predicts no new examples⁵. A pruning criterion might be $pos < 10$ stating that a hypothesis is too specific if it covers less than 10 known examples. These two criteria are given to the system as parameters, so that LINK is very flexible in what kinds of hypotheses it accepts as rules and is able to produce results in domains where other algorithms fail or show strange behavior. This method of testing and evaluating hypotheses is due in large part to [Kietz/Wrobel 92].

1.3 Brief discussion

The hypothesis language defined procedurally by LINK can be described as:

$$\begin{aligned} \mathcal{L}_{Link} = \{h = L_{prems} \rightarrow l_{concl} \mid & \exists \sigma : \{\{l_{concl}\} \cup L_{prems}\} \sigma \subseteq \mathbf{Bg} \\ & \wedge \text{linked-enough}(l_{concl}, L_{prems}) \\ & \wedge |L_{prems}| \leq \text{depth-bound}(l_{concl})\} \end{aligned}$$

where l_{concl} is the conclusion literal (the head), L_{prems} is the set of premise literals, and $\mathbf{Bg}$ is the set of facts that make up the knowledge base⁶. σ is a substitution of terms for variables such that two different variables are replaced by different terms:

$$\sigma \in \{v_i/t_i \mid \forall 1 \leq i \leq j \leq n : (v_i \neq v_j \Rightarrow t_i \neq t_j) \wedge t_i, t_j \text{ are ground}\}$$

³ A brief motivation: as LINK takes a *descriptive* approach, the number of premise literals should bear some relationship to the number of objects involved in the goal relation.

⁴ Note that e.g. GOLEM's test works the other way around: it uses the given examples to partially instantiate the current hypothesis. This is much more efficient, but does not allow analysis of a rule's predictiveness.

⁵ Which implements the framework proposed in [deRaedt/Bruynooghe 93].

⁶ Instances of the learning goal are also in $\mathbf{Bg}$, i.e. LINK may learn recursive rules.

The next restriction imposed on the base language (function-free Horn clauses with negation) is that the premise together with the goal of a hypothesis must be linked-enough (Section 1.1): **linked-enough**(l_{concl}, L_{prems}) $\iff$

$$\begin{aligned} & V_{concl} \subseteq V_{L_{prems}} & (a) \\ & \wedge \forall v \in V_{L_{prems}} \exists v_{concl} \in V_{concl} : v = v_{concl} \vee \text{linked}(v, v_{concl}) & (b) \\ & \wedge \forall v \in V_{L_{prems}} \exists l_1, l_2 \in L_{prems} : l_1 \neq l_2 \wedge v \in V_{l_1} \wedge v \in V_{l_2} & (c) \end{aligned}$$

where V_{concl} are the head variables, and $V_{L_{prems}}$ are the variables occurring in the premise. (a) makes the hypothesis generative, (b) produces a linked horn clause and (c) makes the result strongly generative. The linked relation between variables of a hypothesis is best defined recursively: **linked**(v_1, v_2) $\iff$

$$[\exists l \in \{l_{concl}\} \cup L_{prems} : v_1, v_2 \in V_l] \vee [\exists v_3 : \text{linked}(v_1, v_3) \wedge \text{linked}(v_2, v_3)]$$

The over-all effect of this is that hypotheses with free variables, especially non-generative hypotheses, are not put forth by LINK. Variables for which this condition cannot be fulfilled are turned into constants. On the whole, this is a weaker restriction than that of *ij-determinacy* used in GOLEM, since determinacy is not supposed, and there is no explicit depth limit (i) for the variables occurring in a hypothesis, so that LINK is able to learn in domains where GOLEM isn't. FOIL's information gain heuristic ensures that only linked conjuncts are added to the premise during specialization, but accepts hypotheses with free variables, and adds strictly one literal at a time ("myopy")⁷, whereas *linked-enough* may add several before testing a hypothesis. The restriction formalized with the σ substitution above also has the effect that LINK is not affected by the "theta-redundant specialization literal" pitfall identified by Semeraro [Semeraro et al. 93].

In combination, the linked-enough heuristic and hypothesis test method map a snaking path through the hypothesis search space, consisting of one or more general-to-specific passes for each uncovered example, since there are usually several ways of fulfilling the linked-enough condition. The direction of search within one pass (that is, the order in which **linked**(v_1, v_2) is fulfilled) is dependent on the connectivity of the data at hand and governed by the preference defined in Fig.1. Note that LINK may also be used 'incrementally' to augment an existing set of rules, since only uncovered examples are used in the construction of linked-enough hypotheses.

References

- [deRaedt/Bruynooghe 93] Luc deRaedt and Maurice Bruynooghe. A Theory of Clausal Discovery. In S. Muggleton (ed.), *Proc. 3rd Intl. Workshop on Inductive Logic Programming (ILP'93)*, pp. 25-40, 1993.
- [Kietz/Wrobel 92] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the Complexity of Learning in Logic through Syntactic and Task-Oriented Models. In Stephen Muggleton (ed.), *Inductive Logic Programming*, chapter 16. Academic Press, London, New York, 1992.
- [Semeraro et al. 93] G. Semeraro, C. Brunk, and M. Pazzani. Traps and Pitfalls when Learning Logical Theories: A Case Study with FOIL and FOCL. Technical Report 93-33, University of California at Irvine, 1993.

⁷ Or *all* determinate literals if there is no information gain.