Perfect Refinement Operators can be Flexible

Liviu Badea[1]

Abstract. A (weakly) perfect ILP refinement operator was described in [1]. It's main disadvantage however is that it is *static* and inflexible: for ensuring non-redundancy, some refinements of a hypothesis are disallowed in advance, regardless of the search heuristic which may recommend their immediate exploration. (Similar problems are faced by Progol and other complete and non-redundant systems). On the other hand, there are systems, like FOIL, which give up completeness for maximum flexibility. But if the heuristic fails to guide the search to a solution, such a system cannot rely on a complete refinement operator to explore alternative paths.

In this paper we construct a *dynamically* perfect refinement operator which combines the advantages of completeness, non-redundancy and flexibility, and which represents one of the best tractable ILP operators one can hope for.

1 Introduction and motivation

Inductive Logic Programming (ILP) systems are computationally expensive due to the large spaces of hypotheses they search. Often, the difficulties they face are attributed solely to the weakness (or maybe myopia) of the search heuristic employed. We argue that among the responsible factors, one should also count the lack of *flexibility* of the refinement operator, its *redundancy*, as well as its *incompleteness*.

While completeness and non-redundancy are desiderata that have been achieved in state-of-the art systems like Progol [3], *flexibility* has hardly been studied or even defined in a precise manner. Flexibility becomes an issue especially in the case of (weakly) complete and non-redundant refinement operators, because redundancy is usually avoided by imposing a strict discipline on refinement operations, which usually relies on a predetermined (static) ordering of the literals and variables from the Most Specific Clause. The resulting lack of flexibility can unfortunately disallow certain refinements, even in cases in which the search heuristic recommends their immediate exploration. These hypotheses will be explored eventually, but maybe with an exponential time delay.

Example 1 *Consider the most specific clause $\bot = L_1 L_2 \ldots L_n$ and a static ordering: $L_1 < L_2 < \ldots < L_n$ of its literals.*

Let us assume that from the one-step refinements of the empty clause $\square$, adding L_n produces the largest decrease in negative examples covered[2]*, so that after adding L_n just one negative example (neg_i) remains covered (but which can be removed by the subsequent addition of L_1). On the other hand, let us further assume that the addition of L_1 by itself (to $\square$) eliminates just neg_i, so its refinement will be delayed by the coverage heuristic.*

A static *refinement operator (like the one in [1]) using the literal ordering $L_1 < \ldots < L_n$ will disallow the addition of L_1 to L_n (because this would violate the ordering $L_1 < L_n$), whereas the heuristic will discourage the refinement of L_1 and thereby the addition of L_n to L_1. Thus, blocking the refinement of L_n with L_1 just for reasons of a static discipline enforced to prevent redundancies, will postpone obtaining the solution $L_1 L_n$ very much. This delay can be exponential if an exponential number of combinations of $\{L_2, \ldots, L_{n-1}\}$ eliminate at least 2 negative examples and are therefore preferred to L_1 for refinement.*

The solution to this problem is to enhance the *flexibility* of the refinement operator by using a dynamic literal ordering, constructed at search time. Starting with an empty order relation, literals are ordered as they are added to the hypothesis selected for refinement by the search heuristic. In our case, L_n will be preferred for refinement by the heuristic and L_1 will be ordered w.r.t. L_n ($L_n < L_1$) only after refining L_n with L_1.

Combining completeness and non-redundancy with flexibility hasn't been tackled up to now. In this paper, we show that although maximal flexibility can only be achieved at the expense of intractability and exponential storage space, a limited form of flexibility can be achieved without significant additional costs, while preserving the completeness and non-redundancy (even perfectness) of the refinement operator from [1]. This hints at a very general trade-off between: (weak) completeness, non-redundancy and (maximal) flexibility.

Note that if we insist on the tractability of the refinement operator, we can have any two of the above three properties, but not all three taken together. For example, FOIL [4] gives up completeness for maximum flexibility. But if the heuristic fails to guide the search to a solution, the system cannot rely on a complete refinement operator to explore alternative paths. On the other hand, Progol [3] insists on completeness and non-redundancy at the expense of flexibility: some refinement steps are never considered because of the static discipline used for eliminating redundancies. Finally, systems based on *ideal* refinement operators are complete and can be maximally flexible, but they are highly redundant.

For achieving non-redundancy, the traversed space of hypotheses has to be encoded somehow in order to be avoided in the future, or a certain traversal discipline has to be imposed.

Whenever a given hypothesis C can be reached from both C_1 and C_2, a non-redundant refinement operator would have to choose between considering C as a refinement of C_1 or one of C_2. In order to avoid the encoding of the traversed search space, [1] achieves non-redundancy by adopting a discipline in traversing the hypotheses space. This discipline is induced by an order relation on the literals and on the variables, which amounts to predetermining[3] which one of C_1 or C_2 will be refined to C. For example, if $C \in \rho(C_1)$, then C will not be explored as a refinement of C_2, even if the heuristic considers C_2 to be more promising than C_1. The exploration of C will have to wait until C_1 will be selected for refinement by the heuristic, although C could have been reached from C_2 if only the refinement operator had been flexible enough. (The lack of flexibility is due to the simple scheme – based on predetermined orderings – for avoiding redundancy without significant space or time overheads.)

Maximal flexibility would amount to deciding *at search time* whether $C \in \rho(C_1)$ or $C \in \rho(C_2)$, depending on which one of C_1

[1] AI Lab, National Institute for Research and Development in Informatics, 8-10 Averescu Blvd., Bucharest, Romania. e-mail: badea@ici.ro

[2] while covering the same positive examples.

[3] independent of the heuristic or the order of refinements at search time.

or C_2 is selected first for refinement. Unfortunately, maximal flexibility can only be achieved at the expense of large (at least exponential) time and space overheads (even if our best current approach keeps the traversed hypotheses space in compressed form). However, a limited form of flexibility can be achieved without introducing such significant overheads.

Before going into technical details, let us illustrate the general intuition behind our idea on an example. The *static* refinement operator from [1] uses an order relation on the literals of the Most Specific Clause, for example $L_1 < L_2 < L_3 < L_4 < L_5$ for $\bot = L_1L_2L_3L_4L_5$. When trying to add a new literal, say L_3, to a hypothesis, say L_2L_5, we first check whether the order relation is preserved. In the present case, L_3 cannot be added to L_2L_5 because $L_3 < L_5$.

For obtaining a more flexible (dynamic) refinement operator, we need to construct the order relation at search time.

Assume we first expand the empty clause $\Box$ with L_2, then L_2 with L_5, and finally L_2L_5 with L_3. This introduces an ordering between the successively added literals: $L_2 < L_5 < L_3$. This order will be globally visible, i.e. valid for all branches of the search space (and not just the current one). Therefore, when L_3 is considered for refinement by trying to add L_2, the global "constraint store" will be (temporarily) augmented to $L_2 < L_5 < L_3 < L_2$, which is cyclic and thus inconsistent. Therefore, L_3L_2 is *not* allowed as a refinement of L_3. (If we would allow it, then we would regenerate a node identical to L_2L_3, the latter being already a valid refinement of L_3.)

The refinement operator is dynamic (flexible) because if L_3 would have been expanded (refined) *before* L_2, then L_3L_2 would have been a refinement of L_3, while $L_2L_3 \notin \rho(L_2)$ and $L_2L_5L_3 \notin \rho(L_2L_5)$ because, in this scenario, $L_3 < L_2$. In fact, the first FOIL-like sequence of refinements is fully unconstrained, while preserving weak completeness and non-redundancy.

The refinement operator is not maximally flexible because when trying to expand L_3, L_3L_5 is not allowed as a refinement of L_3, although the node L_5L_3 has not been explored (the relation $L_5 < L_3$ comes from adding L_3 to L_2L_5 rather than L_5 alone; however, since for reasons of tractability and space, we do not want to keep n-tuples with $n > 2$, we cannot avoid such situations).

2 Refinement operators for hypotheses spaces bounded below by a MSC

Refinement operators decouple the search heuristic from the search algorithm. For a top-down search, we deal with a *downward* refinement operator, i.e. one that constructs clause *specialisations*. In the following, we will consider refinement operators w.r.t. the subsumption ordering between clauses. We briefly review the relevant notions [2, 1]. (Due to space limitations, proofs had to be omitted.)

Definition 1 *A* static *refinement operator ρ is called:*

- (locally) finite *iff $\rho(C)$ is finite and computable for all C.*
- proper *iff for all C, $\rho(C)$ contains no $D \sim C$.*
- complete *iff for all C and D, $C \succ D \Rightarrow \exists E \in \rho^*(C)$ such that $E \sim D$.*
- weakly complete *iff $\rho^*(\Box) = S$ (the entire set of clauses).*
- non-redundant *iff for all C_1, C_2 and D, $D \in \rho^*(C_1) \cap \rho^*(C_2) \Rightarrow C_1 \in \rho^*(C_2)$ or $C_2 \in \rho^*(C_1)$.*
- ideal *iff it is locally finite, proper and complete.*
- optimal *iff it is locally finite, non-redundant and weakly complete.*
- minimal *iff for all C, $\rho(C)$ contains only downward covers*[4] *and all its elements are incomparable ($D_1, D_2 \in \rho(C) \Rightarrow D_1 \not\succeq D_2$ and $D_2 \not\succeq D_1$).*
- perfect *iff it is minimal and optimal.*

[4] D is a downward cover of C iff $C \succ D$ and no E satisfies $C \succ E \succ D$.

Limiting the hypotheses space below by a most specific (bottom) clause $\bot$ leads to a more efficient search. This strategy has proven successful in state-of-the-art systems like Progol, which search the space of hypotheses C between the most general clause (for example the empty clause $\Box$) and the most specific clause $\bot$: $\Box \succeq C \succeq \bot$ (for reasons of efficiency, the generality ordering employed is subsumption rather than full logical implication).

Formalizing Progol's behavior amounts to considering hypotheses spaces consisting of clause-substitution pairs $C = (cl(C), \theta_\bot(C))$ such that $cl(C)\theta_\bot(C) \subseteq \bot$. (For simplicity, we shall identify in the following $cl(C)$ with C.) [5]

In the following, we restrict ourselves for simplicity to refinement operators for *flattened definite Horn clauses*.

3 Flexibility: static versus dynamic refinement operators

The subsumption lattice of hypotheses (like almost every other generality order used in machine learning) is far from being tree-like: a given clause D can be reachable from several incomparable hypotheses $C_1, C_2, \ldots$.

Proposition 1 *A refinement operator (in a non-tree-like lattice of hypotheses) cannot be both* complete *(a feature of* ideal *operators) and* non- redundant *(a feature of* optimal *operators).*

Proposition 2 *For each ideal refinement operator ρ we can construct an optimal refinement operator $\rho^{(o)}$.*

$\rho^{(o)}$ is obtained from ρ such that for $D \in \rho(C_1) \cap \ldots \cap \rho(C_n)$ we have $\exists i$ such that $D \in \rho^{(o)}(C_i)$ and $\forall j \neq i$, $D \notin \rho^{(o)}(C_j)$.

If the choice of C_i from $\{C_1, \ldots, C_n\}$ does not depend on the history of previous refinements, we shall call the optimal refinement operator $\rho^{(o)}$ *static*.

Such a static operator has a major drawback. If the heuristic happens to guide the search so that some C_j ($j \neq i$) is visited before C_i, a static refinement operator will not reach the hypothesis D (which can be a solution!) until it will explore C_i, which is the only node from which D is reachable. And it may be that C_i is explored much later than C_j, maybe even after an exponential delay.

It seems that this is the price we have to pay for maintaining non-redundancy. In other words, it seems we have to make a trade-off between flexibility and non-redundancy (while maintaining weak completeness): we can reach D from each C_j (maximal flexibility), but apparently only at the expense of redundancy. Fortunately, this is true only for *static* operators. In fact, we can construct a *dynamic* refinement operator which will choose C_i from $\{C_1, \ldots, C_n\}$ at "search-time", i.e. depending on the history of previous refinements, or – more generally – depending on some contextual information.

More precisely, whereas a *static* refinement operator maps a hypothesis to the set of its refinements:

$$\begin{array}{rccl} \rho_s: & HYP & \longrightarrow & 2^{HYP} \\ & C & \longmapsto & \rho_s(C) \subseteq HYP, \end{array}$$

a dynamic operator maps a hypothesis C *and a given context* Ctx to the set of refinements of C and the updated context Ctx':

$$\begin{array}{rccl} \rho_d: & HYP \times CTX & \longrightarrow & 2^{HYP} \times CTX \\ & (C, Ctx) & \longmapsto & \rho_d(C, Ctx) = (Refs, Ctx'), \\ & & & Refs \subseteq HYP. \end{array}$$

[5] In general, for a given clause C there can be several distinct substitutions θ_i such that $C\theta_i \subseteq \bot$. Viewing the various clause-substitution pairs (C, θ_i) as distinct hypotheses amounts to distinguishing the $\bot$-literals associated to each of the literals of C.

The role of the context Ctx is to encode somehow the history of previous refinements in order to avoid redundancies in a dynamic, rather than static way.

The dynamic operator ρ_d is used as a subroutine by the search algorithm of an ILP system. Such an algorithm is depicted below.

search$_\rho$(Open, Closed, Ctx)
$C = select(Open)$
$(Refs, Ctx') = \rho_d(C, Ctx)$
$Refs' = Refs \setminus (Open \cup Closed)$ $\quad (*)$
if C_i is a solution for some $C_i \in Refs'$ **then** return C_i
$search_\rho((Open \setminus \{C\}) \cup Refs', Closed \cup \{C\}, Ctx')$

Checking in $(*)$ whether some refinement C_i is not already in $Open \cup Closed$ may take time proportional to the size of $Open \cup Closed$ (which is typically at least exponential). We could get rid of $(*)$ (and use $Refs'$ instead of $Refs$), but then we may redundantly visit certain parts of the search space, unless ρ_d itself is non-redundant.

The role of the context Ctx is to allow the dynamic operator ρ_d to avoid redundancies. The simplest such dynamic operator would use the set of visited nodes as context ($Ctx = Open \cup Closed$): $\rho_d(C, Ctx) = \rho(C) \setminus Ctx$, where $\rho(C)$ is a complete refinement operator. Unfortunately, computing the set difference above for such an exponentially large context Ctx is too expensive, computationally. We shall adopt a more sophisticated approach involving a polynomially-sized context.

The search algorithm consists of a sequence of refinement steps $refine_{\rho,\sigma}(Open, Closed, Ctx) = (Open', Closed', Ctx')$ which map a state $s = (Open, Closed, Ctx)$ to a new state $s' = (Open', Closed', Ctx')$

$$\begin{array}{rccl} refine_{\rho,\sigma}: & State & \longrightarrow & State \\ & s & \longmapsto & refine_{\rho,\sigma}(s) = s', \end{array}$$

where $State = 2^{HYP} \times 2^{HYP} \times CTX$.

Here, $Closed$ is the set of hypotheses that have been refined (expanded), while $Open$ are the hypotheses visited, but not yet refined.

$refine_{\rho,\sigma}(Open, Closed, Ctx) = (Open', Closed', Ctx')$
$C = \sigma(Open)$
$(Open', Closed', Ctx') = refine_\rho(C)(Open, Closed, Ctx)$

$refine_\rho(C)(Open, Closed, Ctx) = (Open', Closed', Ctx')$
$(Refs', Ctx') = \rho(C, Ctx)$
$Open' = (Open \setminus \{C\}) \cup Refs'$
$Closed' = Closed \cup \{C\}$

The *selection function* $\sigma : 2^{HYP} \to HYP$ chooses for refinement the best unrefined hypotheses (according to a given heuristic).

Definition 2 *A* history $Hist \in HYP^\star$ *induced by the selection function σ is the sequence of hypotheses $Hist = C_0 C_1 \ldots C_n$ as they are selected for refinement by σ: $C_i = \sigma(Open_i)$, where*
$s_0 = (\{\Box\}, \emptyset, \emptyset)$ is the initial state,
$s_0, s_1, \ldots, s_n$ is the sequence of states induced by ρ and the selection function σ: $s_{i+1} = refine_{\rho,\sigma}(s_i)$, and
$s_i = (Open_i, Closed_i, Ctx_i)$.
For a history $Hist$, we define

$$refine_\rho(Hist)(s_0) \stackrel{def}{=} refine_\rho(C_n)(\ldots refine_\rho(C_1)(s_0))$$

to be the state reached from s_0 by ρ if the hypotheses C_i selected for refinement are exactly those from $Hist = C_0 C_1 \ldots C_n$.

Definition 3 *D is a refinement of C (w.r.t. ρ and σ) iff the history $Hist = C_0 C_1 \ldots C_n$ contains a sequence of hypotheses $C_{i_1} = C, C_{i_2}, \ldots, C_{i_p} = D$ ($0 \le i_1 \le \ldots \le i_p \le n$) such that $C_{i_{j+1}} \in Refs$, where $(Refs, Ctx_{i_j+1}) = \rho(C_{i_j}, Ctx_{i_j})$, for $1 \le j < p$.*

(In other words, D can be reached from C by a sequence of direct refinements).

Definition 4 *A* dynamic *operator ρ is called (w.r.t. a selection function σ):*

- weakly complete *iff $refine^\star_{\rho,\sigma}(s_0) = (Open, Close, Ctx)$ with $Open \cup Close = HYP$ (the entire set of hypotheses)*
- redundant *iff some D is a refinement of both C_1 and C_2, while neither C_1 nor C_2 is a refinement of the other.*

The above notions allow us to define in a precise manner the *flexibility* of a dynamically optimal operator.

Definition 5 *A dynamically optimal refinement operator ρ_d is* maximally flexible *iff $D \in \rho(C)$ for an ideal operator ρ entails that for all selection functions σ and their induced histories $Hist = C_0 C_1 \ldots C_n$ with $C_n = C$ and $refine_\rho(Hist)(s_0) = (Open, Closed, Ctx)$, we have*
$D \in \rho_d(C, Ctx)$ iff D doesn't occur in $Hist$ ($D \neq C_i, i = 1, \ldots, n$).

In other words, any D reachable from C using an ideal (complete) operator should also be reachable from C by ρ_d iff D wasn't explored before (in $Hist$).

Although desirable, maximal flexibility can only be achieved at the expense of an at least exponential complexity of each refinement step. Even our best compressed representation[6] of the visited nodes may become exponentially large. Therefore, in order to preserve the tractability of the individual refinement steps, we will have to settle for a form of flexibility weaker than maximal flexibility. Such a weaker flexibility would nevertheless allow a completely unconstrained first sequence of refinements, like in FOIL (which represents a significant improvement over a static refinement operator). Unlike FOIL however, which is incomplete[7], our flexible operator preserves (weak) completeness and non-redundancy.

Definition 6 *A dynamically optimal refinement operator ρ_d is* flexible *iff $D \in \rho(C)$ for an ideal operator ρ entails the existence of a selection function σ and its induced history $Hist = C_0 C_1 \ldots C_n$ with $C_n = C$ and $refine_\rho(Hist)(s_0) = (Open, Closed, Ctx)$ such that $D \in \rho_d(C, Ctx)$ and D doesn't occur in $Hist$ ($D \neq C_i$ for $i = 1, \ldots, n$).*

4 Tractable non-redundancy by destroying the commutativity of refinement operations

For avoiding the exponentially-sized context representing the histories of already visited hypotheses, we settle for a less faithful representation of the histories using *binary* order relations between literals and variable occurrences (instead of a faithful but very large encoding of the visited hypotheses as *n-ary tuples*). The main advantage of such binary relations consists in their worst-case polynomial[8] size.

The intuitive justification behind the introduction of such binary order relations as contexts is the following. Since redundancies arise due to the *commutativity* of the operations of the refinement operator[9], we can achieve non-redundancy by destroying the commutativity of these operations by imposing an ordering on these operations. *Static* (predefined) orderings lead to the static refinement operator from [1].

[6] whose precise details are outside the scope of this paper.
[7] if the sequence of refinements recommended by the search heuristic fails to reach a solution, the whole search is compromised.
[8] in our case quadratic, due to the pairs of variables from equality literals.
[9] such as literal additions or variable unifications. For example, $D \cup \{L_1, L_2\}$ can be reached both from $D \cup \{L_2\}$ by adding L_1 and from $D \cup \{L_1\}$ by adding L_2. This redundancy is due to the commutativity of the operations of adding literal L_1 and literal L_2 respectively.

For obtaining *dynamic* operators, we need to construct the orderings dynamically, i.e. at search time.

In fact, we can view this in a more abstract setting in which we construct hypotheses using a set of commutative operations $o_1, o_2, \ldots$. If we aim to achieve non-redundancy while avoiding the use of exponential space, we shall only store tuples $o_1 < o_2$ of some *binary* relation '$<$'. Since the operations are commutative ($o_1 o_2 = o_2 o_1$) and we want to avoid 2-redundancies, the relation '$<$' should be *asymmetric*: if on some branch of the search tree we apply o_1 followed by o_2 ($o_1 < o_2$), we should disallow o_2 to be followed by o_1 ($o_2 < o_1$) even on other branches: $(o_1 < o_2) \rightarrow \neg(o_2 < o_1)$, i.e. $\neg(o_1 < o_2 \wedge o_2 < o_1)$. Asymmetry implies *irreflexivity*: $\neg(o < o)$.

Avoiding the different permutations of $n > 2$ operations is more difficult since we do not want to keep n-tuples, but just 2-tuples. For example, for avoiding the redundancy $o_1 o_2 \ldots o_n = o_2 \ldots o_n o_1$, we have to impose the *acyclicity condition* $\neg(o_1 < o_2 \wedge \ldots \wedge o_{n-1} < o_n \wedge o_n < o_1)$. (Note that irreflexivity and asymmetry are special cases of acyclicity (for $n = 1$ and $n = 2$ respectively).) Although acyclicity does not imply *transitivity*: $o_1 < o_2 \wedge o_2 \wedge o_3 \rightarrow o_1 < o_3$, transitivity plus irreflexivity implies acyclicity. In fact, since *a relation is acyclic iff its transitive closure is irreflexive*, we can consider our '$<$' to be a *strict order relation* (irreflexive and transitive).

5 From ideal to dynamically optimal refinement operators

We have already seen (proposition 1) that, due to completeness, ideal refinement operators cannot be non-redundant and therefore optimal. As already argued, non-redundancy is extremely important for efficiency. We shall therefore start with an *ideal* refinement operator (which is easier to construct) and transform it to an *optimal* operator by replacing the stronger requirement of completeness with the weaker one of weak completeness.

The following refinement operator is *ideal w.r.t. weak subsumption* (as defined in [1]).

$D \in \rho_\perp(C)$ iff either

(1) $D = C \cup \{L'\}$ with $L' \in \perp' \setminus C$, or
(2) $D = C \cup \{X_i = X_j\}$ with $\{X_i/A, X_j/A\} \subseteq \theta_\perp(C)$.

The clauses C and D are considered to be ordered sets of (ordinary) literals (such as $p(X_i, X_j, \ldots)$) and equality literals $X_i = X_j$.

For each literal $L \in \perp$ of the Most Specific Clause $\perp$, we denote by L' the literal L with new and distinct variables and $\perp' = \{L' | L \in \perp\}$. For example, if $\perp = \ldots \leftarrow p(A, A, B), q(A, B)$, then $\perp' = \ldots \leftarrow p(X_1, X_2, X_3), q(X_4, X_5)$.

An example of a clause in this representation is $C = \ldots \leftarrow p(X_1, X_2, X_3), X_1 = X_2, q(X_4, X_5), X_1 = X_4$. Note that ordinary literals do not share variables – the unification of variables is explicitly represented by means of equality literals.

The refinement operator $\rho_\perp$ above constructs clauses with at most one occurrence of each $\perp$-literal. This allows us to avoid the problems due to the non-existence of ideal refinement operators w.r.t. (ordinary) subsumption [2] by introducing a weaker form of subsumption, which exactly captures the behaviour of implemented systems (like Progol) by disallowing substitutions that identify literals.

Definition 7 *Clause C* weakly-subsumes *clause D* relative to $\perp$, *$C \succeq_w D$ iff $C\theta \subseteq D$ for some substitution θ that does not identify literals (i.e. for which there are no literals $L_1, L_2 \in C$ such that $L_1\theta = L_2\theta$) and such that $\theta_\perp(D) \circ \theta = \theta_\perp(C)$.*

We now construct a *dynamically optimal* operator $\rho_\perp^{(o)}$ from $\rho_\perp$ by destroying the commutativity between the operations of $\rho_\perp^{(o)}$.

A dynamic *order* relation between literals (ordinary literals and equalities $X_i = X_j$) is however insufficient to avoid all redundancies. Due to the transitivity of equality, a given variable cluster $X_1 = X_2 = \ldots = X_n$ can be obtained not just with different permutations of a given set of equality literals $X_i = X_j$, but also using *different* sets of equality literals. For instance, the cluster $X_1 = X_2 = X_3$ can be obtained either with $\{X_1 = X_2, X_1 = X_3\}$ or with $\{X_1 = X_2, X_2 = X_3\}$.

To avoid such redundancies, we shall also introduce an order relation '$\prec$' on the set of variable occurrences from $\perp'$ (whose role is to impose a discipline in the construction of variable clusters) and disallow adding $X_i = X_j$ after $X_k = X_l$ if $X_i \prec max(X_k, X_l)$ and $X_j \prec max(X_k, X_l)$. This can be shown to be equivalent with

$$X_i = X_j < X_k = X_l \text{ iff } max(X_i, X_j) \prec max(X_k, X_l). \quad (1)$$

Note that the order relations '$<$' and '$\prec$' are total but can be dynamic and thus need not be fully specified at all times – all we need to do is to ensure the *consistency* of the global constraint store (containing the tuples of '$<$' and '$\prec$').

In the case of a shared variable X_i, (1) becomes

$$X_i = X_j < X_i = X_k \text{ iff } X_i \prec X_k \text{ and } X_j \prec X_k. \quad (2)$$

Note that each of the variables of a given literal L' from $\perp'$ is the same throughout the whole search space (in all hypotheses to which L' is added), so that the constraints involving them are globally visible.

An optimal (even perfect[10]) refinement operator obtained from $\rho_\perp$ can be constructed as follows:

$D \in \rho_\perp^{(o2)}(C)$ iff either

(1) $D = C \cup \{L'\}$ with $L' \in \perp' \setminus C$ such that adding the global constraints $L' > C$ preserves the consistency of the global constraint store, $\theta_\perp(D) = \theta_\perp(C) \cup \theta_\perp(L')$, or
(2) $D = C \cup \{X_i = X_j\}$ with $\{X_i/A, X_j/A\} \subseteq \theta_\perp(C)$, such that $cluster_C(X_j) = \{X_j\}$ and adding the global constraints
   (a) $(X_i = X_j) > C$ and
   (b) $X_k \prec X_j$ for each $X_k \in cluster_C(X_i)$, but only if $|cluster_C(X_i)| \geq 2$

   preserves the consistency of the global constraint store. $\theta_\perp(D) = (\theta_\perp(C) \setminus \{X_i/A, X_j/A\}) \cup \{select_D(cluster_D(X_i))/A\}$

where

$cluster_C(X_i) = \{X_i\} \cup \{X_k \mid \{(X_i = X_{j_1}), (X_{j_1} = X_{j_2}), \ldots, (X_{j_p} = X_k)\} \subseteq C\}$ is the cluster of variables unified with X_i in C, and

$select_C(K_i) = X_i$ is a function that selects a variable $X_i \in K_i$ from the variable cluster K_i.

Initially, $\theta_\perp(\Box) = \emptyset$.

Adding $L > C$ to the global constraint store (L being a literal and C a clause) amounts to adding $L > L_i$ for all literals (either ordinary or equalities) $L_i \in C$ (or, more practically, $L > L_i$ for all *maximal* literals L_i from C).

In order not to clutter the presentation, we have preferred a more procedural writing of $\rho_\perp^{(o2)}$ in which the context (represented here by the set of order constraints on literals and variables) is a "global variable", implicitly modified by the call to $\rho_\perp^{(o2)}(C)$. The associated dynamic operator is, formally speaking, $\rho_d(C, Constrs) = (\rho_\perp^{(o2)}(C), Constrs')$, where $Constrs$ and $Constrs'$ represent the global constraint set before and respectively after the call to $\rho_\perp^{(o2)}(C)$.

$\rho_\perp^{(o2)}$ adds either a new ordinary literal, or a new equality. The order relation on literals is constructed dynamically, as literals are added during successive refinements. Of course, the consistency of the global constraint store needs to be preserved.

[10] w.r.t. weak subsumption

Adding equalities is trickier to a certain extent due to the transitivity of equality. First, we have to avoid trivial redundancies arising if we would allow adding $X_i = X_j$ for X_i and X_j already belonging to the same cluster. We do this by keeping in the set $\theta_\perp(C)$ of variables candidates for unification just one representative of each cluster of variables $cluster_C(X_i)$. (We use a selection function to pick the representative of the new cluster.)

We have required the variable cluster $cluster_C(X_j)$ to be a singleton because otherwise $X_i = X_j$ would represent a *"bridge"* between two non-trivial clusters[11], and such "bridges" are disallowed by our convention (2) as we show below.

Indeed, for two non-trivial clusters $K_i = \{X_i, X_k, \ldots\}$ and $K_j = \{X_j, X_l, \ldots\}$, $X_i = X_j$ must have been preceded by some $X_i = X_k$ and by some $X_j = X_l$. But $X_i = X_k < X_i = X_j$ entails, by (2), $X_i \prec X_j$, while $X_j = X_l < X_i = X_j$ entails $X_j \prec X_i$ leading to an inconsistency.

The constraints introduced in step *(2b)* ensure that a variable cluster $X_1 = X_2 = \ldots = X_n$ can be generated with only one sequence of refinements of type *(2)*, for example $X_1 = X_2$, followed by successively adding $X_3, X_4, \ldots, X_n$ to the growing cluster.

Note that the addition of X_i to the subcluster $X_1 = X_2 = \ldots = X_{i-1}$ can, in principle, be done in several ways, such as $X_j = X_i$, for $j = 1, \ldots, i-1$. The precise X_j (determined by the selection function $select_C(\{X_1, \ldots, X_{i-1}\})$) is not important – all that matters is the sequence of variables that are added to the growing cluster.

If the selection function $select_C(K)$ depends on C, then checking consistency may prove more complicated than simply testing the acyclicity of the literal and variable orderings.

Consistency checking can be simplified if a given variable cluster $X_1 = X_2 = \ldots = X_n$ can be obtained only with a given sequence of equalities, for example $X_1 = X_2 < X_1 = X_3 < \ldots < X_1 = X_n$. (In this case, the ordering between literals and the one between variables do not interact [12], so it suffices to test the consistency of the literal ordering and that of the variable ordering separately.)

Since no "bridges" between clusters are allowed, variables are added to a cluster one by one, for example $X_1 = X_2$, followed by $X_3, X_4, \ldots, X_n$. The constraints introduced in step *(2b)* lead to the variable ordering $\begin{smallmatrix} X_1 \prec \\ X_2 \prec \end{smallmatrix} X_3 \prec X_4 \prec \ldots \prec X_n$.

If, on a given path, variables have been added in this order, it will not be possible to add them on a different path in a different order to form the same cluster without violating the variable constraints above. Now, if the selection function $select_C(K)$ does not depend on C, then the addition of X_j, which will be done by the equality $(X_i = X_j)$ with $X_i = select_C(\{X_1, \ldots, X_{j-1}\})$, will be done by the equality $(X_i = X_j)$ everywhere (on all search paths).

Proposition 3 *If the selection function $select_C(K)$ does not depend on C, then checking the consistency of the constraint store can be reduced to separately testing the acyclicity of '$<$' and '$\prec$' respectively.*

Note that this does not affect the flexibility of the resulting refinement operator – only checking consistency is easier to do.

Example 2 *The sequence* $X_i = X_j < \boxed{X_i} = X_k < \boxed{\boxed{X_j}} = X_l$ *is induced by the selection function* $select(\{X_i, X_j\}) = \boxed{X_i}$, $select(\{X_i, X_j, X_k\}) = \boxed{\boxed{X_j}}$.

If we additionally require that the variable selected from a non-trivial variable cluster doesn't change as we go from a clause to its refinements:

$$select(cluster_D(X_k)) = select(cluster_C(X_k))$$

for $C \subseteq D$ and $|cluster_C(X_k)| \geq 2$, then using the fact that clusters can only grow as literals are added to C:

$$cluster_C(X_k) \subseteq cluster_D(X_k)$$

for $C \subseteq D$ and $|cluster_C(X_i)| \geq 2$, we can show that the only sequences of unifications leading to the $clusterX_1 = X_2 = \ldots = X_n$ will be of the form $X_1 = X_2 < X_1 = X_3 < \ldots < X_1 = X_n$, inducing the variable ordering $\begin{smallmatrix} X_1 \prec \\ X_2 \prec \end{smallmatrix} X_3 \prec X_4 \prec \ldots \prec X_n$. Thus, the cluster of X_i from step *(2)* of $\rho_\perp^{(o2)}$ will be given by:

$$cluster_C(X_i) = \{X_k \mid (X_i = X_k) \in C\} \cup \{X_i\},$$

while $select(cluster_C(X_i)) = X_i$ for all C. Note that from the variables of $cluster_C(X_i)$, only X_i appears in $\theta_\perp(C)$.

We thereby obtain the following simplified *dynamically optimal* (even perfect) refinement operator.

$D \in \rho_\perp^{(o3)}(C)$ iff either

(1) $D = C \cup \{L'\}$ with $L' \in \perp' \setminus C$ such that adding the global constraints $L' > C$ preserves the consistency of the global constraint store, $\theta_\perp(D) = \theta_\perp(C) \cup \theta_\perp(L')$, or

(2) $D = C \cup \{X_i = X_j\}$ with $\{X_i/A, X_j/A\} \subseteq \theta_\perp(C)$, such that adding the global constraints

(a) $(X_i = X_j) > C$ and

(b1) $X_i \prec X_j$, $X_k \prec X_j$ for each X_k such that $(X_i = X_k) \in C$ or $(X_k = X_i) \in C$[13],

(b2) $X_j \prec X_i$, $X_k \prec X_i$ for each X_k such that $(X_j = X_k) \in C$ or $(X_k = X_j) \in C$,

preserves the consistency of the global constraint store.
$\theta_\perp(D) = \theta_\perp(C) \setminus \{X_i/A\}$ if (b2) was applied,
else $\theta_\perp(D) = \theta_\perp(C) \setminus \{X_j/A\}$.

Note that if in step *(2)* there exist in C both $X_i = X_k$ and $X_j = X_l$, then we obtain a "bridge" which induces the conflicting constraints $X_i \prec X_j$ and $X_j \prec X_i$. So (b1) and (b2) cannot be both applicable.

$\rho_\perp^{(o3)}$ can be shown to be *flexible* (although not *maximally flexible*).

6 Conclusions and further work

Implemented ILP systems are either incomplete (like FOIL), redundant, or inflexible (i.e. cannot take advantage of a good search heuristic due to the fixed discipline they use to ensure both completeness and non-redundancy).

This hints at a very general trade-off between completeness, non-redundancy and flexibility, which hasn't been explored before. We give a precise definition of flexibility and construct the first flexible refinement operator that combines the advantages of completeness, non-redundancy and flexibility, while preserving tractability.

We have implemented the refinement operator $\rho_\perp^{(o3)}$ and plan to use it in a more sophisticated refinement operator for complete theories. We also plan to incorporate mode declarations and extend our approach to knowledge refinement rather than learning from scratch.

REFERENCES

[1] Badea Liviu, Stanciu M. *Refinement Operators can be (weakly) perfect.* Proceedings ILP-99, pp. 21-32, LNAI 1634, Springer Verlag, 1999.

[2] Nienhuys-Cheng S.H., de Wolf R. *Foundations of Inductive Logic Programming.* LNAI 1228, Springer Verlag 1997.

[3] Muggleton S. *Inverse entailment and Progol.* New Generation Computing Journal, 13:245-286, 1995.

[4] Quinlan J.R. *Learning Logical Definitions from Relations.* Machine Learning 5:239-266, 1990.

[11] i.e. clusters with more than one variable.

[12] except the condition (2) which needs to be applied only on the current path and which is directly enforced by the algorithm in step *(2b)*.

[13] We add $X_i \prec X_j$ only if such an $X_i = X_k$ or $X_k = X_i$ exists in C.