An Efficient Hypothesis Search Algorithm based on Best-Bound Strategy

Kouzou Ohara, Noboru Babaguchi, and Tadahiro Kitahashi

I.S.I.R., Osaka University,
8-1 Mihogaoka, Ibaraki, Osaka 567-0047, JAPAN
{ohara,babaguchi,kitahashi}@am.sanken.osaka-u.ac.jp

Abstract. Learning with Inductive Logic Programming (ILP) can be regarded as searching the hypothesis space for rules called the hypotheses. The efficiency of an ILP system is mainly dependent on the size of the space. Since the size of the space may be huge, it is required to reduce the number of candidate hypotheses to be actually evaluated to improve the efficiency. For this purpose, we improve A^*-like algorithm in Progol by introducing best-bound strategy and a novel criterion for pruning. Best-bound strategy allows us to avoid the redundant evaluations which can arise in best-first search such as A^*-like algorithm. And the proposed criterion is effective for pruning a lot of candidate hypotheses which are not pruned by existing criteria. We will also show the results of preliminary experiments.

1 Introduction

Inductive Logic Programming (ILP)[2, 4] aims at generating the *hypotheses*, or logical formulae describing the target concept, when given are the positive and negative examples and background knowledge. The framework based on mathematical logic brings some advantages such as the high expressive power. Moreover from the ability to use background knowledge and to reuse hypotheses learned, ILP is expected as an effective approach in the domain of knowledge discovery in database (KDD)[3, 6]. However the efficiency of learning with ILP is not always adequate to deal with a large amount of data in KDD.

Learning with ILP is equivalent to finding out a *best hypothesis*, which is correct and has a maximal evaluation value, from the search space called the *hypothesis space*, which is a subsumption lattice consisting of possible *candidate hypotheses*. Then the computational cost to learn hypotheses is mainly dependent on both the number of candidates to be evaluated and the cost of coverage check for each candidate. Thus to improve the efficiency of the hypothesis search, it is required to reduce them.

For the improvement of coverage check, a method based on Model Generation Theorem Prover has been proposed[8]. On the other hand, Progol[5] is successful in reducing the size of the hypothesis space by adopting, as the bottom clause of the space, the *most specific hypothesis* (MSH) which is derived from a positive example and background knowledge according to *inverse entailment*. In addition, Progol also uses the *mode declaration* which specifies Input/Output relation of arguments of the literals in MSH as a bias to reduce the hypothesis space.

In the contrast with these *static reduction* before searching, Progol reduces the space dynamically in the search with A^*-like algorithm, which is a kind of exhaustive search with pruning and explores the space general to specific. A^*-like algorithm can find the best hypothesis if exists, while the search strategy based on information gain in FOIL[9] may lead to the local solution. In spite of these static and dynamic reduction of the space, however, Progol may require large amount of executing time to deal with the complex and large data set.

From this background, this paper aims at improving A^*-like algorithm so as to further reduce the hypothesis space. First of all, we will point out the redundant evaluations in A^*-like algorithm, and introduce the concept of *best-bound search* in brunch-bound (B&B) method into the search algorithm to avoid the redundancy. In addition, we propose a novel pruning criterion based on the relation between the number of examples two candidate hypotheses cover, which is available if the evaluation function for the hypothesis satisfies the *restricted monotonicity* defined in this paper. We also show the results of the preliminary experiments, which arise our expectation that the proposed algorithm could greatly reduce the hypothesis space dynamically.

2 Outline and Redundancy of A^*-like algorithm

2.1 Outline of A^*-like algorithm

In this section, we briefly explain the search by A^*-like algorithm[5] in Progol as a typical one based on best-first search. The hypothesis space to be explored in ILP is regarded as a subsumption lattice, whose top is an empty clause and whose bottom is the most specific clause to explain an example e. In the following, we assume that the bottom clause is the MSH given with inverse entailment.

A^*-like algorithm searches the hypothesis space for the best hypothesis, enumerating candidate hypotheses. The enumeration is realized by applying the *refinement operator* ρ to a candidate hypothesis H. In fact ρ generates the *children* of H either by adding a new literal in MSH to H or by instantiating a variable in H. For a child H_i of H, H_i is more specific than H and there exists no candidate H' such that H' subsumes H_i and H subsumes H'. We say H *is expanded* if ρ is applied to H. The *descendants* of H are the candidates resulting from applying ρ to H recursively.

In A^*-like algorithm, selection of the candidate hypothesis to be expanded follows best-first strategy: it selects the candidate H whose evaluation value $f(H)$ is maximal in $OPEN$, which is a set of known *open candidates* which have not been expanded yet. As the evaluation function for a candidate hypothesis H, Progol employs the following one:

$$f(H) = p - (n + c + t), \tag{1}$$

where p, n are the number of positive and negative examples H covers, respectively, and c is the number of literals in the body of H. The parameter t is the

1. $OPEN := \{H_0\}$, $CLOSED := \emptyset$
   $H_{max} := e$, $f_{max} := 0$
2. **if** $OPEN = \emptyset$ **then** $Return(H_{max})$
3. $H := Best(OPEN)$
   $OPEN := OPEN - \{H\}$
4. $CLOSED := CLOSED \cup \{H\}$
5. **if** $f_{max} < f(H)$ **and** $n_H = 0$ **then** $f_{max} := f(H)$, $H_{max} := H$
6. **if** $prune(H)$ **then goto** Step2
7. $OPEN := (OPEN \cup \rho(H)) - CLOSED$
8. **if** $g(H') < f_{max}$ for any $H' \in OPEN$
   then $Return(H_{max})$
9. **goto** Step2

Fig. 1. A^*-like algorithm of Progol

number of literals expected to be necessary to satisfy the *mode declarations* given in advance. The value of t is given by another heuristic function.

The selected candidate H is appended to the list $CLOSED$ after being expanded unless it satisfies any of pruning criteria. If H satisfies any of them, it is *terminated*, or appended to $CLOSED$ without being expanded. In Progol4.1, following conditions are adopted as the pruning criteria:

1. $g(H) < f_{max}$,
2. $t = 0, n = 0$,
3. $g(H) < 0$,
4. $c > c_{max}$.

f_{max} is the maximum of known evaluation values, and the function $g(H)$ gives the upper-bound of evaluation values the descendants of H can achieve, which is $p - (c + t)$. In the following, we call the value of $g(H)$ the *upper-bound value of* H. The parameter c_{max} is defined by user in advance, and stands for the maximal number of literals in the body of a candidate. The first condition means that any of descendants of H can not be the best hypothesis if H satisfies it, because their evaluation values are at most $g(H)$ and $g(H)$ is less than the current maximum f_{max}. The second condition means H is correct as the concept description since H satisfies mode declarations ($t = 0$) and it does not cover any negative example ($n = 0$). Thus it is not necessary to specialize, or expand H with ρ any more. The last two criteria mean that any descendant of H can never achieve a positive evaluation value, and that H is too long as the best hypothesis intended by use, respectively: the candidate H satisfying them will be terminated.

A^*-like algorithm will terminate if either $g(H) < f_{max}$ for any $H \in OPEN$ or $OPEN = \phi$ holds. Then the algorithm returns the best hypothesis H_{max} such that $f(H_{max}) = f_{max}$ and H_{max} covers no negative example if exists; otherwise it returns e since there does not exist the best one in the search space.

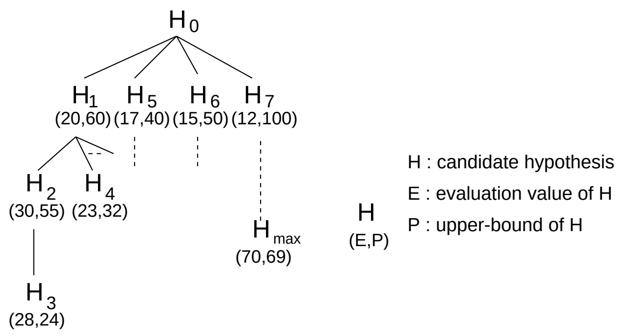

Fig. 2. a partial hypothesis space

Fig.1 illustrates A^*-like algorithm, in which *Best* is the function which selects the candidate H from $OPEN$ such that $f(H') \leq f(H)$ for $\forall H' \in OPEN$, and the function $prune(H)$ returns *true* if H satisfies at least one of the given criteria; otherwise it returns *false*. And n_H is the number of negative examples H covers.

2.2 Redundant Evaluations in A^*-like algorithm

Let us consider a part of hypothesis space shown in Fig.2 in order to discuss the redundancy of A^*-like algorithm. In Fig.2, each H_i stands for a candidate hypothesis, and the 2-tuple (E, F) below H_i represents the pair of the evaluation value E and the upper-bound F of H_i. H_{max} is the best hypothesis in the space.

According to best-first search, the six candidates, $H_1, \ldots, H_6$, will be expanded before the best hypothesis H_{max} is found out, and all of the candidates resulting from the expansion will have to be evaluated for the next selection. In fact, however, these evaluations are redundant because the six candidates do not have to be expanded if H_{max} is found out before they are selected: their upper-bound value is not more than $f(H_{max})$.

Generally the candidates near to the top of a hypothesis space intend to cover many negative examples as well as positive ones because of their generality. This means that their evaluation values intend to be low. If the ancestor of the best hypothesis has a low evaluation value as shown in Fig.2, the redundant evaluations may arise in best-first search, since the search strategy selects the candidate with the maximal evaluation value.

In case of pruning based on the upper-bound value of a candidate hypothesis, the minimal number of candidates to be evaluated is uniquely decided according to the function g. This is because the candidate hypothesis H' such that $f(H_{max}) < g(H')$ is never terminated, even if the search progresses in any order. Now in the whole of search, let N be the number of candidates to be actually evaluated, and N_g be the total number of those which are generated by expanding the candidates like H'. Then N_g is the minimal number of candidates to be evaluated and ideally N should be N_g. However in the actual search with A^*-like algorithm, N may be extremely larger than N_g as mentioned above. This redundancy is shared by the best-first search algorithms with pruning based on the value of g.

3 Searching Hypothesis based on Best-Bound Search

3.1 Applying Best-Bound Search to Hypothesis Search

To avoid the redundant evaluations mentioned above, we focus on *best-bound search* in brunch and bound (B&B) method. B&B method is well known as a solution of the optimization problem and consists of two operations: *branching operation* and *bounding operation*. The former operation is dividing an open problem into some subproblems. The problem to be divided is selected according to the search strategy, and examined by bounding operation whether it is necessary to be divided or not. Actually if its upper-bound value is not more than the *incumbent*, the maximum of known evaluation value, it is terminated without being divided.

Note that the A^*-like algorithm is a kind of B&B method with the best-first search strategy, as well as A^* algorithm is: branching operation in A^*-like algorithm is expanding a candidate hypothesis, and bounding operation is testing it with pruning criteria. On the other hand, In B&B method, *best-bound* search strategy allows us to minimize the number of subproblems to be expanded without being pruned, or without being terminated based on the incumbent. Best-bound strategy selects the subproblem whose upper-bound is maximal in open subproblems as one to be expanded, rather than one whose evaluation value is maximal.

Then we can expect that it is possible to avoid the redundant evaluations mentioned above by selecting the candidate hypothesis H whose upper-bound is maximal in $OPEN$ according to best-bound strategy. For example, in Fig.2, all of the six candidates, $H_1, \cdots, H_6$, are not selected until the best one, H_{max}, is selected, since their upper-bound is less than $g(H_{max})$. In fact, N, the number of hypotheses to be evaluated, can be nearly equal to N_g, because $f(H_{max}) \leq g(H)$ always holds for the selected hypothesis H. The reason why $N \neq N_g$ is that the hypothesis H such that $f(H_{max}) = g(H)$ may be expanded if exists.

It is generally required the upper-bound $g(H)$ must be computable in order to employ B&B method. This requirement is satisfied if the evaluation function f is monotonic. Thus we define the *restricted monotonicity* of the function in ILP with respect to its arguments as follows:

Definition 1. Let H and H' be candidate hypotheses. Suppose p, p', n, n', b, and b' are the number of positive examples, negative examples covered by each of H and H', and literals in the body of each of them, respectively. Then it is said that the evaluation function f satisfies the restricted monotonicity if $f(H)$ and $f(H')$ satisfy the following conditions:

if $p' \leq p$, $n \leq n'$, and $b \leq b'$, then $f(H') \leq f(H)$;

$\square$

Moreover if the evaluation function satisfies the restricted monotonicity defined above, the following theorem holds.

Theorem 2. *Let p be the number of positive examples covered by a candidate hypothesis H, let b_{min} the minimal number of literals in the body of the children of H. Suppose an evaluation function f satisfies the restricted monotonicity defined by definition 1, and it takes the three values as its arguments: the number of positive and negative examples the hypothesis covers and the number of literals in its body. Then the upper bound of the evaluation values the descendants of H can achieve is given by $f(p, 0, b_{min})$.* $\square$

Proof. Suppose there is a candidate hypothesis H' which is a descendant of H such that $f(p, 0, b_{min}) < f(H')$. Let p' and n' be the number of positive and negative examples H' covers, respectively, and b' be the number of literals in H'. Since H' is a descendant, $p' \leq p$ and $b_{min} \leq b'$. Furthermore $0 \leq n'$. Then from definition 1, $f(H') \leq f(p, 0, b_{min})$. This is contradiction. Thus there does not exist a candidate H' such that $f(p, 0, b_{min}) < f(H')$. $\square$

3.2 Updating Incumbent and Pruning

Selecting a candidate hypothesis based on best-bound strategy increases the number of candidates in $OPEN$. This is because the search based on best-bound strategy intends to explore the search space like breadth-first search. The upper-bound of the candidate near to the top of the hypothesis space is larger than that of the candidate at a deeper layer. In addition, any candidate selected is never terminated by pruning test based on the incumbent until the best one is selected.

In fact, there are some open nodes in $OPEN$ such that their upper-bound values are not more than the incumbent f_{max} at that time. Such the candidates do not have to be stored $OPEN$, because they are immediately terminated even if selected. The problem is that any of such the candidates is never given any chance to be examined on the pruning criterion until all of candidates whose upper-bound values are larger than its upper-bound have been expanded.

To solve this problem, we modify the basic B&B algorithm so as to update the incumbent and prune open candidates *immediately* after expanding a candidate H. However we do not always examine all of open candidates for this purpose. To update the incumbent, we have only to compare the current incumbent f_{max} with the maximum $f(H_i)$ of evaluation values of the children $H_1, \ldots, H_n$. If $f_{max} < f(H_i)$, then $f(H_i)$ be the new incumbent. By setting the initial value of the incumbent f_{max} to $-\infty$, f_{max} has the maximum of known evaluation values at all times.

Then we can minimize the number of candidates in $OPEN$, only if we examine open candidates after this update of the incumbent. Again we do not always have to examine all of open candidate hypotheses in pruning test. The candidate hypotheses to be examined are determined according to whether or not the incumbent is updated as follows:

1. if the current incumbent was updated,
   then all open candidate hypotheses including those which are newly generated by ρ are tested;
2. if the current incumbent was not updated,
   then only the new candidate hypotheses generated by ρ are tested.

In the second case, it is obvious that any candidate H which exists in $OPEN$ before applying the refinement operator is never terminated by this test, because it is guaranteed that its upper-bound value $g(H)$ is not less than the current incumbent due to the first condition.

3.3 Pruning Criterion based on Relation of Coverage between Candidate Hypotheses

Here we consider whether we can terminate another candidates other than those which are terminated by the pruning criterion based on the upper-bound value. Intuitively, it seems to be meaningless to expand a candidate H_2 if the number of negative examples H_2 covers is as same as does its parent H_1. For example, let us assume H_1 and H_2 as follows:

$$H_1 : p(A,B) \leftarrow q_1(A,C), q_2(B,D), \tag{2}$$
$$H_2 : p(A,B) \leftarrow q_1(A,C), q_2(B,D), L, \tag{3}$$

where L is a positive literal, and p, q_1 and q_2 are predicates and A, B, C, D are variables. In the following, let p_i and n_i be the number of positive and negative examples a candidate H_i covers, respectively, and the evaluation function f satisfies the restricted monotonicity. We also assume $n_1 = n_2$.

Then we could terminate H_2 if there exists a candidate H_4 in descendants of H_1 such that $f(H_3) \leq f(H_4)$ for H_3, an arbitrary descendant of H_2. In fact, if $n_3 = n_4$ holds, we can guarantee the existence of such the candidate as H_4 because of the restricted monotonicity. To satisfy $n_3 = n_4$, H_1 and H_2 have to satisfy the two requirements. One of them is that any variable in L should appear in the body of H_1. If L contains a variable which does not appear in H_1, then $f(H_3) \leq f(H_4)$ is not guaranteed since $n_3 < n_4$ may hold. Let us consider the following example, in which $L = q_3(A,E)$, and E^- and BK are sets of negative examples and ground atoms in background knowledge, respectively:

$$\begin{aligned}
H_3 &: p(A,B) \leftarrow q_1(A,C), q_2(B,D), q_3(A,E), q_4(B,E),\\
H_4 &: p(A,B) \leftarrow q_1(A,C), q_2(B,D), q_4(B,E),\\
E^- &= \{p(1,2), p(5,3)\},\\
BK &= \{q_1(1,c), q_1(1,d), q_1(5,c), q_2(2,4), q_2(3,4), q_2(3,7),\\
&\quad q_3(1,10), q_3(5,11), q_4(2,10), q_4(3,9)\} .
\end{aligned}$$

Then H_4 covers both $p(1,2)$ and $p(5,3)$, while H_3 covers only $p(1,2)$: in this case, $n_3 = n_4$ does not hold, even if $n_1 = n_2$.

The other requirement is that any negative example H_1 covers should be derived with a unique substitution satisfying the body of H_1. Let us consider the following example in which $L = q_3(A, D)$:

$$\begin{aligned}
H_3 &: p(A,B) \leftarrow q_1(A,C), q_2(B,D), q_3(A,D), q_4(C,D),\\
H_4 &: p(A,B) \leftarrow q_1(A,C), q_2(B,D), q_4(C,D),\\
E^- &= \{p(1,2), p(5,3)\},\\
BK &= \{q_1(1,c), q_1(1,d), q_1(5,c), q_2(2,4), q_2(3,4), q_2(3,7),\\
&\qquad q_3(1,4), q_3(5,7), q_4(c,4)\} \ .
\end{aligned}$$

Then H_4 covers both $p(1,2)$ and $p(5,3)$, while H_3 covers only $p(1,2)$: also in this case, $n_3 = n_4$ does not hold, even if $n_1 = n_2$.

On the other hand, the following theorem shows that if H_1 and H_2 satisfy the above two requirements and $n_1 = n_2$, then there exists a candidate H_4 in descendants of H_1 such that $f(H_3) \leq f(H_4)$ for H_3, an arbitrary descendant of H_2.

Theorem 3. *Let H be a candidate hypothesis and let H' be a candidate hypothesis generated by expanding H. Suppose n_H and $n_{H'}$ are the number of negative examples covered by H and H', respectively, and the evaluation function f satisfies the restricted monotonicity. Then for a descendant of H', H_i, there exists a candidate hypothesis H'_i such that H'_i is a descendant of H and $f(H_i) \leq f(H'_i)$, if H and H' satisfy the following conditions:*

1. *$n_H = n_{H'}$,*
2. *any variable in the body of H' appears in that of H, and*
3. *any of negative examples H covers is derived with a unique substitution satisfying the body of H.*

□

Proof. First of all, suppose H' is a candidate hypothesis resulting from adding a literal L to H. Then there is a candidate H'_i resulting from deleting L from H_i in the descendants of H. Let b_{H_i} and $b_{H'_i}$ be the number of literals in the bodies of H_i and H'_i, respectively. Then $b_{H'_i} < b_{H_i}$. Let p_{H_i} and $p_{H'_i}$ be the number of positive examples H_i and H'_i cover, respectively, and n_{H_i} and $n_{H'_i}$ be the number of negative ones H_i and H'_i cover, respectively. Then $p_{H_i} \leq p_{H'_i}$ since H_i is more specific than H'_i. Now the second condition in the theorem means that the body of H' has to be satisfied with at least one of substitutions satisfying the body of H. In addition, from the first and third conditions, a negative example derived from H with a substitution has to be also derived from H' with only the same substitution. This means $n_{H_i} = n_{H'_i}$ holds since a negative example derived from H'_i with a substitution is also derived from H_i with only the same substitution. Consequently $f(H_i) \leq f(H'_i)$ holds from definition 1.

1. $OPEN := \rho(H_0)$
   $CLOSED := \emptyset$, $H_{max} := e$, $f_{max} := -\infty$
2. **if** $OPEN = \emptyset$ **then** return(H_{max})
3. $Change\_flag :=$ **false**
4. $H := Select(OPEN)$
5. $OPEN := OPEN - \{H\}$
   $CLOSED := CLOSED \cup \{H\}$
6. $SUB := \rho(H) - (OPEN \cup CLOSED)$
7. $f(H_k) := \max_{H_i \in SUB} f(H_i)$
8. **if** $f_{max} < f(H_k)$ **and** $check(H_k)$
   then $f_{max} := f(H_k)$, $H_{max} := H_k$
   $Change\_flag :=$ **true**
9. **each** $H' \in SUB$
   if $g(H') \leq f_{max}$ **or** $prune1(H, H')$ **or** $prune2(H')$
   then $SUB := SUB - \{H'\}$
   $CLOSED := CLOSED \cup \{H'\}$
10. **if** $Change\_flag =$ **true**
    then each $H' \in OPEN$
    if $g(H') \leq f_{max}$
    then $OPEN := OPEN - \{H'\}$
    $CLOSED := CLOSED \cup \{H'\}$
11. $OPEN := OPEN \cup SUB$
12. **goto** Step2

Fig. 3. The Best Bound Hypothesis Search Algorithm

On the other hand, in case that H' is generated by instantiating a valuable V in H with a constant v, there is a candidate H_i' resulting from replacing v in H_i with V in the descendants of H. Then obviously $b_{H_i'} = b_{H_i}$ and $p_{H_i} \leq p_{H_i'}$. And it is provable as well as the above case that a negative example derived from H with a substitution is also derived from H' with only the same one. Thus $n_{H_i} = n_{H_i'}$ if H and H' satisfy the three conditions in theorem 3. Again from definition 1, $f(H_i) \leq f(H_i')$ holds. □

From this theorem we can terminate such the candidate as H'. This pruning criterion is available on any algorithm for hypothesis search whose evaluation function satisfies the restricted monotonicity.

We summarize the discussion in this section as the *best-bound hypothesis search algorithm* in Fig.3, in which *Select* is the function which selects the open candidate with the maximal upper-bound value from $OPEN$, and the function *check* returns *true* if H_k satisfies the condition to be a correct hypothesis such as it should cover no negative example; otherwise it returns *false*. And the function *prune1* returns *true* if H and H' satisfy the three conditions in theorem 3, while *prune2* returns *true* if H' satisfies any other pruning criteria user defined.

As a result, this algorithm has the features of both best-bound and best-first strategies in the sense that it updates the incombent based on the *maximal* *evaluation value* as well as best-first one, while it selects the candidate hypothesis to be expanded based on the *maximal upper-bound value* according to best-bound strategy.

4 Preliminary Experiment

4.1 Experimental Design

We implemented each algorithm, the best-first hypothesis search algorithm and best-bound hypothesis search algorithm, on SUN Ultra1 (CPU: UltraSPAR-C1/144MHz, Memory:192MB) by using C language and SICStus Prolog3.0 as a module in the ILP system G-REX[7], which is a Progol-based system we developed and capable of learning hierarchical *default rules*, or hypotheses with exceptions like LELP[1]. In the implementation, we employed inverse entailment and the mode declaration to restrict the hypothesis space in advance as well as Progol. As the evaluation function, we adopted the following one:

$$f(p,n,c) = (p-n-c) \times (1 - E(\frac{p}{p+n})), \tag{4}$$

where p, n, c are the same as the equation(1), and $E(x) = -x\log(x) - (1-x)\log(1-x)$. $E(\frac{p}{p+n})$ means the entropy of the hypothesis, or its uncertainty, which reflects the risk of covering negative examples to the evaluation value. For example, if the hypothesis h_1 with 5 literals in its body covers 100 positive examples and 10 negative ones, and the other hypothesis h_2 with the same length as h_1 covers 1,000 positive ones and 910 negative ones, then f_1, the evaluation value of h_1, is higher than f_2, that of h_2, while $f_1 = f_2 = 85$ if we use only the first term in the equation(4).

To examine the performance of the proposed algorithm, we used two databases in the Machine Learning Repository of University of California, Irvine: House vote database and Mushroom database. In the following, we summarize their features.

House Votes database: This database contains 435 examples, or records about votes for the U.S. House of Representatives Congressmen, 267 democrats and 168 republicans, on 16 key votes. When regarding these votes as attributes, each of them has either *yes*, *no*, or *?* as its value. We set the target concept to the republicans represented by the atom $republican(A)$ and use as background knowledge the ground atoms in the form of $p(s,t)$, where p is an attribute name, s is an identifier of each example, and t is the value of p in the record of s. Consequently the background knowledge consists of about 7,000 atoms.

Mushroom database: This database contains 8,124 records about North American mushrooms, each of which is classified into either *edible* or *poisonous* and involves 22 attributes which can take at least 2 and at most 12 kinds of values. We set the target concept to the poisonous mushroom which is represented by the atom $poison(X)$ and use as background knowledge the ground atoms in the form of $p(s,t)$ as well as House votes database. Then the number of positive and negative examples are 3,916 and 4,208, respectively and the number of ground atoms in the background knowledge is about 170,000.

Since these databases are propositional, and each example exactly corresponds to one record in each database, we have only to check the first condition in theorem 3 in order to prune a candidate based on the theorem.

We use the House votes database to compare the number of hypotheses the two algorithms evaluated in the search. On the other hand, to evaluate the two algorithms from the statistical view point, we use the Mushroom database in the following way:

1. randomly extract 1,000 examples from each of positive and negative ones;
2. using the 2,000 examples, generate best hypotheses;
3. repeat step1-2 10 times and average the total number of evaluated candidates;
4. repeat step1-3, changing the number of extracted examples to 2,000 and 3,000.

Since the purpose of these experiments is to examine the number of candidates these algorithms evaluate and to compare the results, we will omit the description of the hypotheses they acquired in this paper.

4.2 Experimental Results

In the experiment on House vote database, the two algorithms learned the same descriptions, 13 hypotheses (Horn clauses) and 3 examples which fail to be generalized. First of all, we show, in Fig.4, the comparison of the number of candidates evaluated to acquire each hypothesis by the best-first search and by the best-bound search *without pruning based on theorem 3*, in order to examine the effect of best-bound strategy. The number of candidates the best-bound hypothesis search evaluated is never more than did the best-first search. In fact the total number of candidates the best-bound search evaluated was 69,351, while the best-first search evaluated 78,826 candidates. Thus the 9,475 candidates, the difference between them, are just those which are redundantly evaluated by the best-first search. This result means that introducing best-bound strategy into the hypothesis search is effective for avoiding the redundancy in the best-first search like A^*-like algorithm.

On the other hand, Fig. 5 shows the comparison of the number of candidates evaluated to acquire each hypothesis by the best-first hypothesis search and by the best-bound hypothesis search *with pruning based on theorem 3*. The number of hypotheses the best-bound search evaluated is less than the best-first hypothesis search in any hypothesis. Especially for the 9th hypothesis, it is less than the half of the number in the best-first search. The total number of candidates

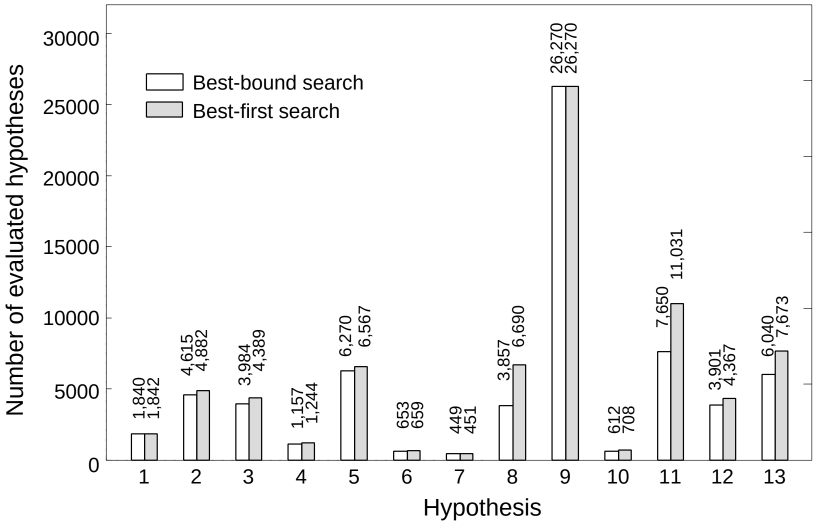

Fig. 4. Comparison of the number of hypotheses evaluated by each hypothesis (1)

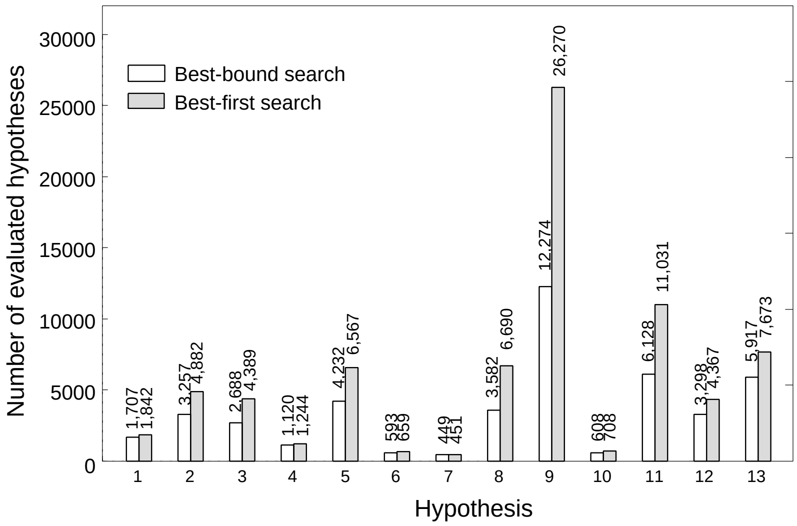

Fig. 5. Comparison of the number of hypotheses evaluated by each hypothesis (2)

Table 1. Number of hypotheses evaluated by applying each algorithm to Mushroom database

Number of examples	2,000	4,000	6,000
Best-first search	39,177	42,414	50,492
Best-bound hypothesis search	12,535	13,872	16,866
Reduction rate	68.00%	67.29%	66.60%

the best-bound search evaluated was only 47,924. Consequently the proposed algorithm cut by about 40% the number of candidates to be evaluated. This result shows the proposed criterion can prune a large number of candidates through the search.

Next Table 1 shows the averages of the number of candidate hypotheses each algorithm evaluated for each amount of data set in the experiments on Mushroom database. We see from Table 1 that the best-bound search cut by about 70% the number of evaluated candidates compared to the best-first search in any amount. For the whole of the original data set, the total number of candidates the best-bound hypothesis search evaluated was 19,437, while the best-first search evaluated 63,153 candidates. Namely the best-bound search is successful in reducing the number of candidates to be evaluated by about 70% also in this case. These results mean that the pruning criterion based on theorem 3 is highly effective at least for the propositional data set, and show potential for being effective also for the data set including multiple relations.

5 Conclusion

In this paper, we proposed an algorithm for the hypothesis search, which adopts both best-bound search strategy and a novel pruning criterion based on the coverage of a candidate and its child in the subsumption lattice. Best-bound search strategy allows us to avoid the redundant evaluations which may arise in the search based on best-first search like A^*-like algorithm. And the results of the preliminary experiments showed the proposed criterion is highly effective at least for the propositional data set. We believe that these results show potential of proposed algorithm for being effective also for the data set including multiple relations.

As the future work, we are planning to apply the proposed algorithm to many kinds of data sets including multiple relations in order to show its advantage. In fact, it would be expected that checking the third condition in theorem 3 entails additional costs. However we could avoid introducing the additional operations to check it by means of relational calculus in deductive database[10] for the coverage check: we could know the number of substitutions satisfying the body of a candidate hypothesis as that of tuples in a relation corresponding to the body, which would be obtained in the process of computing the coverage.

Acknowledgment

This research was partly supported by a Grant-in-Aid for Encouragement of Young Scientists of Japan Society for the Promotion of Science.

References

1. Inoue, K., & Kudoh, Y. (1997). Learning extended logic programs. *Proc. IJCAI'97* (pp. 176 – 181).
2. Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming: Techniques and applications*. Ellis Horwood.
3. Lavrač, N., & Džeroski, S., & Numao, M. (1999). Inductive Logic Programming for Relational Knowledge Discovery, *New Generation Computing*, *17*, (pp. 3 – 23).
4. Muggleton, S. (1992). *Inductive logic programming.* Academic Press.
5. Muggleton, S. (1995). Inverse entailment and progol. *New Generation Computing*, *13*, 245–286.
6. Muggleton, S. (1999). Scientific knowledge discovery using inductive logic programming. *Communications of the ACM*, *42*, *11*, 245–286.
7. Ohara, K., Taka, H., Babaguchi, N., & Kitahashi, T. (2000). Determination of General Concept in Learning Default Rules. *Proc. PRICAI2000*(to be published).
8. Ozaki, T., Furukawa, K., Murakami, T., & Ueno, K. (1997). Realizing Progol by Forward Reasoning. *ILP-97* (pp. 227 – 234).
9. Quinlan, J. R. & Camereon-jones, R. M. (1995). Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing*, *13*, 287–312.
10. Ullman, J. D. (1988). *Principles of Database and Knowledge-base Systems*. Computer Science Press.