Incremental Data-driven Refinement of Knowledge

Robert A. Morris **Ibrahim El-far**
Computer Science Program
Florida Institute of Technology
150 W. University Blvd.
Melbourne, FL 32901
{*morris*, *ielfar*}@cs.fit.edu

Abstract

This paper studies reasoning for the purpose of uncovering hidden assignments of values to a finite set of variables based on data acquired incrementally over time. This reasoning task, referred to here as *data-driven refinement*, is found in tasks such as diagnosis and learning. An empirical study is undertaken to evaluate the utility of knowledge gained from observations in effectively solving refinement problems.

Introduction

Refinement has been defined as *the search that starts with the set of all potential solutions for a problem, and repeatedly narrows and splits the set until a solution for the problem can be extracted* (Kambhampati 1998). *Incremental data-driven refinement*, the focus here, can be viewed as involving three *phases*: interpreting observations, updating beliefs, and generating *queries*. Refinement thus involves a store of beliefs, which is incrementally updated as the result of a sequence of queries. To each query, there is assigned a "context" which provides the reasoner with additional information which guides it in future queries; the query and context together form what is called here an *observation*. There is also one or more *solutions*, initially hidden from the reasoner, whose discovery terminates the reasoning process. Informally, by *effective refinement* is meant the ability to terminate the search for a solution in a timely manner, with a minimal number of queries. In this paper, refinement problems have only one solution. One focus here is on tradeoffs that arise when a reasoner *randomly forgets* some of the past knowledge gained from observations.

In what follows, data-driven refinement is defined, examples provided, and the dual notions of *soundness* and *completeness* with respect to knowledge updates defined. Binary refinement is introduced to serve as the environment for the experimental study, as is a constraint-based reasoner to solve binary refinement problems. The search space of binary refinement is analyzed. Experiments are described on large refinement problems which test the role of applying observations in effective refinement.

Problem Formulation

By a *representation* is meant a pair $R = (V, D)$, where V is a finite set of variables and $D = \{D_1, D_2, \ldots, D_n\}$ is a set of domains for each variable in V. Given $R = (V, D)$ $R' = (V', D')$ is a *sub-representation* of R if $V' \subseteq V$ and $D' \subseteq D$. We write $R' = R\lceil(V', D')$ to express the sub-representation relationship. Two representations are said to be disjoint if they have no variables in common. The union of two (disjoint) representations, $R = (V, D) \cup R' = (V', D')$ is the representation $R'' = (V \cup V', D \cup D')$.

A language, or formula in a language is *based on* a representation R if its atoms consist of variables and values from R. Given a set of formulas F based on a representation $R = (V, D)$, the models of F are the set of assignments of values from $D_j \in D$ to each $V_j \in V$ that satisfies each sentence in F, i.e., makes all the sentences true. Thus a model is a subset of $D_1 \times \ldots \times D_n$. The set of models of F will be denoted by M_F.

The general refinement problem

A refinement problem is based on a triple of potentially distinct, but related representations:

Definition 1 *(Refinement Problem)* The *basis* for a refinement problem is a triple $(\mathcal{Q}, \mathcal{C}, \mathcal{I})$ where:

- $\mathcal{Q} = (Q, D_Q)$ is a representation for expressing *queries*;
- $\mathcal{C} = (C, D_C)$ is a representation for expressing *contexts*; and
- $\mathcal{I} = (I, D_I)$ is a representation for interpreting *observations*. We assume the relationship $\mathcal{I} = \mathcal{Q}\lceil(I, D_I)$.

Given the basis $(\mathcal{Q}, \mathcal{C}, \mathcal{I})$ for a refinement problem, the following notions are defined:

- a *query* q is a conjunction of assignments of domain elements in D_Q to the variables in Q;
- a *context* c is a set of formulas based on the representation $\mathcal{C}$;
- an *observation* is a pair (q, c), where q is a query and c is a context;
- an *interpretation* $i = Int(q, c)$ of an observation is a sentence in Disjunctive normal form (DNF) based on $\mathcal{I}$.
- Where f is a sentence in DNF, let $|f|$ denote the number of terms in f. A *complete solution* to a refinement problem is a query q such that $|Int(q, c)| = 1$.

This definition of a solution to a data-driven refinement problem captures the intuitive essence of refinement, viz., as the process of deriving a single truth about a domain.

A *refinement problem* is a 4-tuple $\langle \mathcal{Q}, \mathcal{C}, \mathcal{I}, s \rangle$, consisting of a basis and an initially "hidden" solution s.

Solving a data-driven refinement problem involves discovering the solution. It is accomplished through the incremental accumulation of information by generating queries and interpreting observations. The reasoner consults a knowledge base KB to generate queries, and KB is revised based on observations. KB is assumed here to be based on a representation R_{KB} which is a sub-representation of $\mathcal{Q} \cup \mathcal{C} \cup \mathcal{I}$. $\mathcal{I}$ is also a sub-representation of R_{KB}, i.e., the interpretation of an observation is based on a representation that the reasoner consulting a KB can use.

Examples

An example of refinement is the game of Mastermind.

Example 1 *(4-peg, 6-color Mastermind)* The standard game of Mastermind is the refinement problem $\langle \mathcal{Q}, \mathcal{C}, \mathcal{I}, s \rangle$, where:

- $\mathcal{Q} = (H = \{H_1, H_2, H_3, H_4\}, D = \{col_1, \ldots, col_6\})$, representing 4 holes and 6 colors. A query language based on this representation is the set of formulas of the form:

$$H_1 = col_{j_1} \wedge H_2 = col_{j_2} \wedge H_3 = col_{j_3} \wedge H_4 = col_{j_4}.$$

- $\mathcal{C} = (\{CP, CC\}, D_C = \{1, 2, 3, 4\})$ is a representation of contexts. A context is a formula of the form $(CP = i \wedge CC = k)$, $i, k \in D_C$, where this means "There are j correct colors in the correct position, and k correct colors in the wrong position". An observation is a tuple of the form (q, c).
- Informally, the interpretation of an observation $(q, CP = m \wedge CC = k)$ is "m of the pegs in q are the right color in the right position, and k of the pegs in q are the right color but in the wrong position". Formally, each interpretation can be viewed as a sentence in DNF, where each term is a set of assignments to variables in H which are consistent with the context.

A more realistic example of refinement is the problem of testing in order to resolve ambiguity in diagnosis (Struss 1995). A model for diagnosis consists of a *device model* CM describing the behavior of the components of a device, and a set M of diagnostic modes, where each mode classifies each component in the system as either working normally, or being in some abnormal state. At each state of the process, a diagnostic reasoner is working with a mode M_c in M, a component model CM, and a set OBS of observations of the system. When a state is reached in which $CM \cup OBS \cup M_c$ is inconsistent, the diagnostic reasoning process commences. The reasoning terminates when M_c is replaced by an alternative mode M'_c which is consistent with OBS and CM. Ambiguity arises when $CM \cup OBS \models M_1 \vee \ldots M_n$, i.e., the current knowledge is consistent with a set of alternative modes. Ambiguity resolution is reasoning for the purpose of deriving a single correct mode, if one exists. Presumably, to converge on a single mode, further observations will be taken. The process of generating a set of observations OBS_c which, together with CM, imply a single component mode, is a data-driven refinement problem.

A model of refinement

By a *refinement strategy* is meant a method for selecting queries, updating beliefs and interpreting observations for the purpose of discovering a solution. We specify two properties of updates, soundness and completeness.

Definition 2 *(Sound and Complete Updates to Knowledge).* Let KB be a set of formulas and $O = (q, c)$ be an observation based on $\mathcal{I}$. Let $M_{KB|\mathcal{I}}$ and M_O be models of KB restricted to the variables and domains of $\mathcal{I}$, and models of O, respectively. An update KB^+ to KB based on O is said to be *complete* if $M_{KB^+} \subseteq M_{KB} \cap M_O$. The update is *sound* if $M_{KB} \cap M(O) \subseteq M_{KB^+}$.

Thus, sound updates remove only assignments not in M_O, and complete updates remove all of them.

Informally, a strategy for refinement will be said to be *effective* for a class of refinement problems if it solves problems in that class in a reasonable time using a small number of queries. The time it takes to solve a refinement problem is determined by the number of queries to solution and the time it takes between queries to update a knowledge base and select the next query. A sound and complete strategy is expected to find a solution with a number of queries that is close to optimal for the problem. On the other hand, for refinement problems in which the number of possible solutions is large, the time spent between queries using a sound and complete update strategy might be prohibitive, due to the overhead incurred maintaining soundness and completeness of the knowledge. A question is raised therefore whether sound and complete strategies are generally necessary for effective refinement. This question is examined empirically later in this paper by comparing a sound and complete refinement strategy with a set of incomplete strategies. The class of refinement problem which forms the task, environment, and protocol for this study is called *binary refinement*. First, this variety of problem is defined, as is a sound and complete strategy for solving it.

Binary refinement

Binary refinement is a reasoning problem in propositional logic. It can be viewed as the problem of deriving, from an initial knowledge base comprised of the formula $p_1 \vee p_2 \vee \ldots \vee p_n$, a unique solution $p_1^* \wedge p_2^* \wedge \ldots p_n^*$, where p^* is either p or $\neg p$. Contexts indicate how many of the literals in a query are correctly assigned.

Formulation of problem

Definition 3 *(Binary refinement).*A *Binary Refinement Problem* is a 4-tuple $(\mathcal{Q}, \mathcal{C}, \mathcal{I}, s)$ where

- $\mathcal{Q} = (P = \{p_1, \ldots, p_n\}, \{0, 1\})$. An assignment of a variable p to 1 or 0 will be designated using the literal notation, respectively, p and $\neg p$. A query is a conjunction of literals for each member of P.
- $\mathcal{C} = (\{N\}, \{1, 2, \ldots, n\})$, where N is a variable standing for the number of "correct" literals in a query. A context is thus simply a number between 1 and n.
- Given an observation $ob = (q, m)$, the interpretation $Int(ob)$ of ob is a DNF proposition with $\binom{n}{m}$ terms. Each term is a conjunction of literals (taken from P) m of which are identical to those found in q.
- s is the unique query out of 2^n possible queries associated with the observation (s, n).

For example, let q be $p_1 \wedge p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge p_5 \wedge \neg p_6$. The observation $(q, 2)$ is interpreted as the disjunction of $\binom{6}{2} = 30$ terms each of which has exactly two literals in common with q. For example, one of the terms in $Int(q, 2)$ is $p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge \neg p_5 \wedge p_6$. Therefores, an observation (q, m) is interpreted to mean "m of the literals in q are correct".

thus, for any instance of binary refinement with n variables, the termination condition is a query q associated with the observation (q, n), i.e., such that $|Int(q, n)| = 1$. Binary refinement thus resembles a version of Mastermind in which there are only two colors of pegs, and clues specify how many of the assignments are correct and in the right hole.

Observation-based binary refinement

We introduce a sound and complete binary refinement strategy called *observation-based refinement* (OBR). OBR has the following features:

- At the state at which an observation O_i has been assigned, the knowledge base $KB_i = \{O_1, \ldots, O_{i-1}\}$ consists of the previous $i - 1$ observations;
- Queries q_i are generated by a constraint solver treating the knowledge base KB_i as a set of constraints.

Hence, an initially empty knowledge base $KB = \emptyset$ can be viewed as a Constraint Satisfaction Problem (CSP) with no constraints specified for solving the problem of assigning values from a binary domain to a finite set of variables.

Let q be a query in a Binary Refinement Problem and let $q[i]$ be the literal corresponding to the ith propositional variable in q; the condition $q[i] = q'[i]$ is true if these literals are the same in q and q'. Let $common(q, q')$ be the number of assignments in common between q and q', i.e., the number of times the condition $q[i] = q'[i]$ is true. The following defines a legal query in OBR:

Definition 4 *(Legal query in observation-based binary refinement)* A legal query for an observation based strategy on a binary refinement problem $(\mathcal{Q}, \mathcal{C}, \mathcal{I}, s)$, where $KB = \{O_1, \ldots, O_{i-1}\}$ is the current set of beliefs is a query q_i such that

$$common(q_i, q_j) = k, \ \forall O_j \in KB, \ O_j = (q_j, k).$$

Thus, a legal query q_i is any assignment in which the number of common assignments between q_i and every query $q_j, j < i$ such that $(q_j, k) \in KB_i$ is k.

An OBR strategy generates only legal queries, using a backtracking constraint solver. The idea is to construct a query incrementally, testing each extension to

a partial assignment by computing the number of common assignments with previous queries. For example, the number of common assignments between the partial query $p_1 \wedge \neg p_2$ and $p_1 \wedge p_2 \wedge \neg p_3$ is 1. One backtrack point for the generation of a query is the point at which $common(q_i, q_j) = k + 1$, where q_i is a query being generated and $(q_j, k) \in KB$ is a previous observation. Another backtrack point is where a complete query q has been generated, and there exists a previous observation $O = (q', c)$ such that q has fewer than c assignments in common with q'.

Theorem 1 OBR is a sound and complete refinement strategy.

This theorem follows from the following lemmas:

Lemma 1 *$common(q, q') = k$ if and only if $q \in Int(q', k)$.*

This guarantees that legal queries comprise all and only the possible solutions:

Lemma 2
Given $KB = \{(q_1, k_1), (q_2, k_2), \ldots, (q_p, k_p)\}$, a query q in OBR is legal if and only if

$$q \in Int(q_1, k_1) \cap Int(q_2, k_2) \cap \ldots \cap Int(q_p, k_p).$$

Analysis of search space of OBR

OBR is based on a "literal" representation of knowledge gained from observations, i.e., in which the observations are simply added verbatim to the knowledge. This leads to a virtually cost-free mechanism for belief update and interpretation. On the other hand, there is significant cost incurred in using this knowledge to generate queries.

We measure this cost in terms the number of times an atomic operation, called a *compatibility check*, is performed. A compatibility check is simply a test of the condition $q[i] = q'[i]$ between the query q being constructed and some previous query q' in the knowledge base; i.e., such that $(q', m) \in KB$. Let n be the number of variables and j be the size of the current knowledge base, i.e., the number of previous observations. This knowledge base can be viewed as a $j \times n+1$ matrix of literals, where a row constitutes a previous observation, and there is a column for each variable, as well as a column for the context. Thus, each extension to a partial solution requires a column's worth of (i.e., j) compatibility checks, and a complete query requires a matrix worth of compatibility checks.

The search space explored by OBR during binary refinement to generate a query is a complete binary tree of depth n. We divide the number of compatibility checks required for a single query into two parts: the $n \times j$ checks that were needed for the query itself, and the $j \times f_j$ checks that were conducted off the path that comprised the query. The variable f_j stands for the number of nodes examined that are not on the solution path during the search for the j + 1st query. The complexity of refinement search can be expressed as follows:

Theorem 2 *The number of compatibility checks required to solve an n-variable Binary Refinement problem in p moves using a sound and complete OBR is*

$$\Sigma_{i=1}^{p}(i-1) \times (n + f_i) = n \times \frac{f_i(p-1)p}{2} = O(p^2 n f_i).$$

A visualization of the search space for a simple 4 variable binary refinement problem is found in Figure 1. Each node of the binary search tree is labeled by a *search vector* of context values from the previous observations, drawn from the knowledge base, represented in the figure as a matrix. The root is initialized by the values from the column marked "c" (for context) in the knowledge base, viz. "s=322". Thus, $s[k]$ is the value of the context variable of the *kth* previous observation. Each arc of the search tree is labeled by 1 or 0 depending on the assignment selected for the variable p_i, where i is the depth of the tree after traversing the arc.

In this example, the search is systematic, where each left direction is explored before each right. Assume the solution to this refinement problem is $\neg p_1 \wedge p_2 \wedge p_3 \wedge p_4$. When an arc to depth i is traversed, $s[k]$ is decremented by 1 if the value assigned to the variable p_i as the result of the traversal agrees with the value assigned to p_i in previous observation k. It follows that a *legal query* is the sequence of arc labels on a path of length n whose search vector on the leaf node of the path is a vector of all zeros.

The search space of the problem can be pruned based on a number of conditions being true of the search vector. As discussed earlier, one backtracking point occurs when a search vector contains "−1" as a value, indicating that the partial query has more than m assignments in common with a previous query q such that $(q, m) \in KB$. Second, if there exists a search vector value $s_m[k]$ at depth m which is larger than the remaining depth of the tree, i.e., $s_m[k] > (n - m)$, then no path below it will result in a zero search vector, and hence the sub-tree rooted at such a node need not be further explored.

In the example, $f_4 = 9$, i.e., there are nine compatibility checks for nodes not on the legal query path. Clearly, f_j will change as the knowledge base of observations changes. At the early stages of refinement, there will be few constraints in the knowledge base, hence more legal queries consistent with the knowledge; f_j should thus be small. Conversely, the last

vbles/board size	*avg queries*	*avg checks*	*hi/lo query*	*hi/lo checks*	*failures*	*avg forgotten*
25/9	19.7	4484696	43/8	16582045/14929	0	10.8
25/13	11.8	4086470	17/8	9381376/14929	0	0.4
25/25	11.5	3954191	14/8	9381376/14929	0	0
30/10	19.3	5.9×10^8	32/9	$13.7 \times 10^8/1.7 \times 10^7$	0	8.5
30/11	17.1	6.3×10^8	34/9	$13.7 \times 10^8/1.7 \times 10^7$	0	5.4
30/12	13.3	4.8×10^8	17/9	$13.5 \times 10^8/1.7 \times 10^7$	0	0.8
30/15	13.1	4.8×10^8	15/9	$13.5 \times 10^8/1.7 \times 10^7$	0	0
35/11	72.9	7.8×10^8	501/15	$20 \times 10^8/28 \times 10^7$	1	60.9
35/13	16.6	8.1×10^8	21/13	$30 \times 10^8/7 \times 10^7$	0	2.7
35/14	15.3	6.7×10^8	23/14	$36 \times 10^8/6 \times 10^7$	0	0.9
35/18	14.9	8.3×10^8	17/14	$20 \times 10^8/16 \times 10^7$	0	0
36/11	29.3	13×10^8	58/16	$28.6 \times 10^8/12 \times 10^7$	0	17.3
36/12	19.3	10.4×10^8	25/13	$26.8 \times 10^8/35 \times 10^7$	0	6.3
36/18	15	10.5×10^8	16/13	$27 \times 10^8/29 \times 10^7$	0	0
38/13	74.8	16×10^8	501/18	$41 \times 10^8/31 \times 10^8$	1	61.8
38/15	17.4	18×10^8	22/14	$38 \times 10^8/36 \times 10^8$	0	1.7
38/16	16.3	19×10^8	18/14	$38 \times 10^8/36 \times 10^8$	0	0.2

Table 1: Sample of results of experiments comparing different Data-driven Refinement strategies

query generated, i.e., the solution, will, on average, tend to be against a tightly constrained knowledge base. This should mean that failure nodes will tend to be generated at more shallow levels of the tree, which again will tend to reduce f_j. It is expected that the "middle game" will tend to produce the most search; deeper levels of the tree will be searched before failure points are discovered, and backtracking beyond one level will be required.

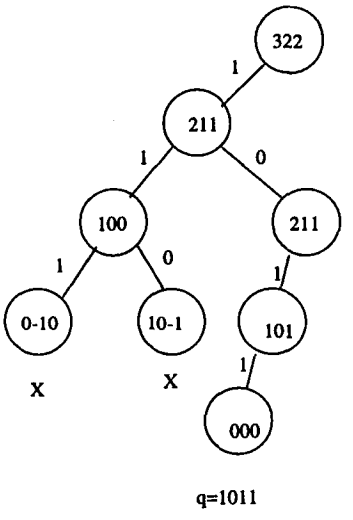
The controllable aspects of the cost of refinement are the number p of queries to solution, and f_j. If p is kept small, it will be a negligible contributor to the cost, especially as n gets large. Alternatively, another way of reducing the cost is to keep f_j small. Sound and complete strategies will tend to focus on keeping p small, possibly at the expense of a larger value for f_j. At the opposite extreme, a random guesser will reduce f_j to 0 (every node examined will be part of the path to the query), but p will be unbounded.

An approach for controlling f_j might be to place a bound in the constrained-ness of the problem examined during search by considering only a fixed number of previous observations, and "forgetting" the rest. The motivation for this idea is the already noted fact that the fewer the constraints, the more queries that are consistent with the knowledge, and the higher the probability of reaching one with little or no backtracking. Of course, bounding the number of constraints means a lesser probability of generating a solution on a given trial, hence it is expected that p will grow. Notice that this approach amounts to relaxing the completeness requirement, since forgetting previous observations means that a reasoner might fail to reject a set of non-solutions based on previous observations. The question to be examined empirically in the remainder of this paper is whether there exists a "degree of forgetfulness" that can be effectively applied for refinement.

Experiments and Discussion

In the tests here, limited checking was imposed by a bounded board size. After the board was filled, a row was randomly selected for deletion. A systematic search was performed for each query based on the method outlined above. To avoid cycles of the same queries in the bounded case, the first assignment for each variable (1 or 0) was chosen randomly. Tests were conducted on varying game sizes (between 25 and 40 variables) and different board sizes, typically in the range between 1/2 to 1/3 of the number of variables. For each game size and board size, 10 games were played on randomly generated solutions[1]. Table 1 contains a representative sample of the results of the experiments. Each row contains the game and board size (number of variables/number of rows), the average

[1]The small number of samples was imposed for practical reasons, dealing with the inordinate amount of time it took to play each game. The reader would be justified in some degree of skepticism with the results, until a larger sample size is taken; on the other hand, the results obtained were repeated for virtually all configurations.



c	queries
3	1111
2	1110
2	1101

Figure 1: Search space for Observation-based Binary Refinement

number of queries to solution, the average number of compatibility checks (an abstract measure of the time to solve the problem), the highest and lowest numbers of queries and checks, the number of "failures" (games which exceeded the limit of 500 guesses), and the average number of times a query was deleted from the board.

Here is a brief summary of the patterns that emerged from the data. On relatively small problem sizes (under 25 variables), it was not clear that any amount of forgetting makes a significant difference in the outcome. This is confirmed in the table, where restricting the board size only results in inferior performance, both in average queries and average checks. It was only for games of more than 25 variables that patterns emerged that could be interpreted as favoring some degree of forgetting. The samples in the table for sizes 30, 35, 36 and 38 are typical for larger games. Although for game size 30 there was no absolute improvement by restricting the board size, there seemed to be a relative improvement in the number of checking by reducing the board size from 11 to 10. For sizes 35, 36 and 38, the significance of forgetting is more dramatic, where a gain in time does not incur, on average, a significant gain in the number of queries to solution, although the range between high and low queries is larger. However, we have not seen, nor do we expect to see with simple random forgetting, an *order of magnitude* improvement in time, in any results obtained thus far.

We conclude from these preliminary data that for large search spaces a small, controlled amount of random forgetting of past knowledge may improve performance of a refinement reasoner, where performance is expressed in terms of time to solution. Future experiments will focus on the effects, if any, of more intelligent ways of forgetting knowledge. For example, we are experimenting with criteria for deleting knowledge based on the "information content" of observations, where this is measured in terms of the size of $int(q, c)$, i.e., the number of terms in the DNF formula that interprets the observation.

Refinement is the subject of a recent study (Kambhampati 1998), which demonstrates that it is applied in a wide range of reasoning tasks, including planning and constraint reasoning. The work here bears a resemblance to the use of dynamic CSPs to reasoning with changing knowledge about the world (Dechter & Dechter 1988). The work on organizing relational data into a form that is effective for reasoning (Dechter & Pearl 1988) is also related. In future research, we will consider such organization as an alternative to forgetting in improving the effectiveness of refinement search.

References

Kambhampati, S. On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. *Artificial Intelligence*, 105(1998), 161-208.

Struss, P. Testing Physical Systems. In *Proceedings of AAAI-94*, Seattle, WA , 251-256, 1994.

Dechter, R. and Dechter, A. Belief Maintenance in Dynamic Constraint Networks. In *Proceedings of AAAI-88*, St. Paul, MN, 37-42, 1988.

Dechter, R. and Pearl, J. Structure Identification in Relational Data. *Artificial Intelligence*, 58(1992), 237-270.