Automatic Generation of Dynamic Investigation Problems

Ramin Ramezani and Simon Colton
*Computational Creativity Group, Department of Computing, Imperial College, London, UK
raminr,sgc@doc.ic.ac.uk

1 Introduction and Motivation

One of the ultimate goals of AI computer programs is to solve real world problems as efficiently, or even better than people; sometimes to even solve problems that cannot be solved by people. Imagine a crime case with many suspects involved where each of the suspects has various motivations for the murder which makes the case fairly complicated and the amount of information would be large for a detective to process. Considering that the knowledge about the crime may not even be sufficient for the detective to deduce the murderer, he/she may refer to previously solved cases which bear resemblance to the current one, hoping to find information that can be generalized to the present problem. Employing this new information may lead to identifying the murderer or to at least making it easier by excluding some of the suspects. We call such problems *investigation problems*, (IPs). These may exhibit ambiguity and complexity but AI problem solving techniques such as machine learning, constraint solving and automated theorem proving are considered as powerful tools for solving such problems. Having focused on investigation problems inspired us to initially come up with a formalized way of defining IPs where we present here. Furthermore, we will discuss an experiment in which different scenarios of a certain IP is generated and is solved by a Constraint Satisfaction Problem (CSP) solving approach.

2 Definition of Dynamic Investigation Problems (DIP)

We consider investigation problems similar to CSPs with a finite set of variables, each associated with a finite domain and a set of constraints; the difference is that the background knowledge of an IP contains information about past cases in addition to the current case description; such information may contain the same set of constraints and variables as in the current case, partially or completely. Past cases contain facts that can be interpreted as embedded constraints; these constraints can become explicit using machine learning techniques.

Let $X \in \{y_1, y_2, \cdots, y_n\} = D_x$, where D_x is the domain of X and $y_1, \cdots, y_n$ are domain values. In a general CSP problem with one variable, the task is to assign a value from domain D_x to X such that all the constraints (see below) are satisfied simultaneously. However, for an IP, the task is slightly different and is explained in the following definition

The problem is to identify an ordered list of domain values for X. Let $\{g'_1, g'_2, \cdots, g'_k\} \subseteq D_x$, where Each g'_i carries a likelihood degree; The likelihood degree of g'_i = $\frac{\text{Number of constraints satisfied by } g'_i}{\text{Total number of constraints}}$. The constraints being satisfied by g'_i are either from problem constraints or from the past cases' embedded constraints. The more constraints each domain value satisfies, the higher the likelihood it has for being the correct answer, thus the answers are prioritized.

Let $C = \{C_1, C_2, \cdots, C_m\}$ be a set of constraints declared in the problem definition where each constraint C_i contains predicates and variables. Define $Pred\_C_i$ to be a set of all predicate names appearing in the constraint C_i and every C_i is in the form:
$C_i(X) :- \quad P_{i1}(Y_{11}, \ldots, Y_{1n}), \ldots, P_{ia}(Y_{a1}, \ldots, Y_{al})$ where:

- a, l, n are arbitrary finite values
- The arguments in P_{ij} may each be a variable or a ground term.
- $\forall \quad P_{ij} \quad s.t \quad 1 \leq j \leq a$, P_{ij} should have a variable V which is 1-connected (Colton and Muggleton, 2006). Therefore:
  - $V = X$ or
  - $\exists\, p, q, r \quad s.t \quad q \neq r$, $Y_{pq} = V$ and $Y_{pr} = X$ or
  - $\exists\, p, q, r \quad s.t \quad q \neq r$, $Y_{pq} = V$ and is a 1- connected variable

Let $Const\_C_i$ to be the set of all constants appearing in C_i

Let $E = \{E_1, E_2, \cdots, E_p\}$ be a set of past cases; We define every past case E_i as a set of $\{p_1, p_2, \cdots, p_n, p_q\}$ where each p_i is a ground fact, i.e., a ground predicate. There are embedded constraints in every E_i which will become explicit. Let $Pred\_E_i$ to be the set of all predicate names appearing in the past case E_i. Let $Const\_E_i$ to be the set of all constants

appearing in the past case E_i. Let $Pred\_C_T$ be the union of all the predicates present in each constraint, $Const\_C_T$ be the union of all the constants present in each constraint and $Pred\_E_T$ be the union of all the predicates present in each of the past cases. It is possible that constraints and past cases may have common predicates.

2.1 Conditions for being an IP

(i) $\forall j\,(\exists\, S \subseteq Pred\_E_j \quad s.t \quad S \subset Pred\_C_T)$
There should be an overlap between the predicates present in every past case and the overall predicates appearing in the constraints. The commonality confirms the relevance between a past case and constraints.

(ii) $\forall j\,\exists\, S' \subseteq Const\_E_j \quad \wedge \quad (\exists\, M : S' \rightarrow (D_x \cup Const\_C_T)) \quad s.t$
$\Big((\exists \delta \in S',\, \exists \lambda \in (D_x \cup Const\_C_T)) \quad s.t \quad M(\delta) = (\lambda)\Big)$
M is a mapping function; This condition implies that at least one constant in a past case can be mapped to a value in the problem domain. By definition: an IP is to hold $(i) \wedge (ii)$

A dynamic investigation problem is similar to an IP. The only difference is the additional time aspect in the problem. The background knowledge keeps changing over time due to addition or extraction of constraints and past cases at different instances. With change in time, constraints, past cases and the domain of the variable can be modified. Let T be a finite set $s.t \;\; T = \{t_1, t_2, \cdots, t_n\}$ where $n \geq 2$ represent time instances. At each time the conditions of an IP should be satisfied. Over time, constraints and past cases can be completely altered, however, assuming D_{1x} to be the domain of X at time $T = t1$ and D_{2x} to be the domain of X at time $T = t2$, the following condition should be always satisfied: $\exists\, y$ in D_{1x} that is in D_{2x}.

3 DIP Constraints Generation

To show that an investigation problem can be defined in a DIP form and is ultimately amenable to a constraint solving approach, we considered a board game known as Cluedo. In this game, the player moves around a mansion with nine rooms where the murder can take place and collects clues to infer which suspect has murdered the victim. In the classic single player cluedo game, the player tries to determine the mystery of the murder by searching every single room. The information gathered by the player at each step provides the constraints and the information about a DIP at time T. We wrote a program using Prolog to randomly generate different scenarios for a Cluedo game. The program outputs a limited set of constraints and predicates at each time slot, making it similar to the real game in which the player can only achieve few clues at every room. However, it is worth pointing out that we only focused on the current case of a DIP, therefore, random generation of the previous cases has not been taken into consideration.

To make the game more interesting, the amount of information being generated at each step in also erratic. Hence the player may find more evidence in a room and less in another. In addition, we increased the complexity of the problem by adding constraints and predicates about the suspects and the murder case in general. For instance *"the murderer should be tall and quick"* or *"the murderer should be angry with something"* are the added constraints. The added predicates could be complementary to the constraints, like *"Professor Plum is tall"* or *"Scarlett is short"*. Below is the partial output of the DIP Cluedo generator:
$at\,Time\,T = 4: \quad is\_angry(scarlett). \quad was\_found(revolver, patio). \quad murderer(X) : -is\_tall(X).$
To show that the DIP is amenable to a constraint solving approach, we wrote a CSP in the syntax of Sicstus CLPFD (Carlsson et al., 1997). At each step we fed the output of the DIP to the CSP and showed that the solver can not come up with a single solution only until the last stage. This means that all the constraints and background information are needed for the program to infer the murderer, murder scene and murder weapon; the output of the final stage depending on the scenario being generated might be: *"Scarlett committed the murder in the Kitchen using a Candlestick"*. However, the program at each stage, depending on the available information, can assign a likelihood degree to each suspect; in addition, if the generated data is not sufficient to solve the mystery even at the final stage, using the previously solved cases could help us to learn a set of rules that can replace the missing information (Ramezani and Colton, 2009).

Acknowledgments

This work is funded by EPSRC grant EP/F036647.

References

Mats Carlsson, Greger Ottosson, and Bjrn Carlson. An open-ended finite domain constraint solver. 1997.

Simon Colton and Stephen Muggleton. Mathematical applications of inductive logic programming. *Machine Learning*, 64(1-3):25–64, 2006.

Ramin Ramezani and Simon Colton. Solving mutilated problems. *Automated Reasoning Workshop*, 2009.