KDD PROCESS PLANNING

Ning Zhong* Dept. of C. S. & Sys. Eng. Yamaguchi University

Chunnian Liu Dept. of C. S. Beijing Polytechnic Univ.

Yoshitsugu Kakemoto RCAST The University of Tokyo

Setsuo Ohsuga Dept. of Infor. & C. S. Waseda University

Abstract

KDD (Knowledge Discovery in Databases) process has become a new and important research area. Within the framework of KDD process and the GLS (Global Learning Scheme) system recently proposed by us, this paper concentrates on the issue of KDD process planning. In our method, the KDD process is modeled as an organized society of intelligent agents (called KDD agents), and planning is a meta-agent. We propose a formalism to describe KDD agents, in the style of OOER (Object Oriented Entity Relationship data model). Based on this representation of KDD agents as operators, we apply AI planning techniques to organize dynamically the KDD process so that the GLS system increases both autonomy and versatility.

Introduction

Recently, it has been recognized in the KDD (Knowledge Discovery in Databases) community that the KDD process for real-world applications is extremely complicated (Brachman 1996; Fayyad 1996; Zhong & Ohsuga 1995; Zhong et al 1997). There are several levels, phases and large number of steps and alternative KDD techniques in the process, iteration can be seen in anywhere and at any time, and the process may repeat at different intervals when new/updated data comes. In many aspects, a KDD process resembles a Software Development Process (Liu 1991; Liu & Conradi 1993). In (Zhong et al 1997), we propose a KDD process framework as an organized society of intelligent agents. There are two meta-levels (for planning and controlling), and the meta-agents are called planner and controller respectively in this paper. There is also an object level with three learning phases (preprocessing, knowledge elicitation, and knowledge refinement) of actual KDD activities with each modeled as an intelligent agent (called KDD agent). We also design a general-purpose KDD system – GLS (Global Learning Scheme) based on the framework, which increases both autonomy and versatility. In this paper, we focus on issues of KDD process planning, that is, the meta-agent planner.

*Email: zhong@ai.csse.yamaguchi-u.ac.jp. Copyright@1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Though research on KDD process in general has got some results, KDD process planning is a brand-new area. So far we have seen in the KDD literature only one paper (Engels 1996) devoted to this topic, emphasizing user-guided task-decomposition. In contrast, we apply various AI planning techniques to the area of KDD, taking much broader view, including formal specification of KDD process, its planning, controlling, and evolution.

To be able to apply AI planning techniques, each KDD agent should be regarded as an operator, and formally described. We introduce a formalism for this purpose in the style of OOER (Object-Oriented Entity Relationship data model). For each type of KDD agents, the types of its input/output, the precondition and effect of its execution, and its functionality are explicitly specified in the data model. The most difficult problem in a general-purpose KDD system is that how to choose appropriate KDD techniques to achieve a particular discovery goal in a particular domain. In our method, the combination of the formal description of KDD agents and the planning mechanism gives an automatic solution to this problem (to some extent, at least). In such a KDD system, both autonomy and versatility are increased.

An Architecture of KDD Process

KDD process is a multi-step process centered on data mining algorithms to identify what is deemed knowledge from database. In (Zhong et al 1997), we model the KDD process as an organized society of KDD agents. Based on this model, we have been developing a multi-strategy and cooperative KDD system called GLS (Global Learning Scheme). Here we give a brief summary of the architecture of the GLS system.

The system is divided into three levels: two meta-levels and one object level. On the first meta-level, the *planning meta-agent* (planner, for short) sets the discovery process plan that will achieve the discovery goals when executed. On the second meta-level, the KDD agents are dynamically generated, executed, and controlled by the *controlling meta-agent* (controller, for short). Planning and controlling dynamically the discovery process is a key component to increase both autonomy and versatility of our system. On the object level, the KDD agents are grouped into three learning phases:

Pre-processing agents include: agents to collect information from global information sources to generate a central large database; agents to clean the data; and agents to decompose the large database into several local information sources (subdatabases), such as *CBK* (attribute oriented clustering using background knowledge), *QDR* (quantization by the division of ranges), *FSN* (forming scopes/clusters by nominal or symbolic attributes), and *SCT* (stepwise Chow test to discover structure changes in time-series data).

Knowledge-elicitation agents include: agents such as *KOSI* (knowledge oriented statistic inference for discovering structural characteristics – regression models), *DBI* (decomposition based induction for discovering concept clusters), and *GGDT* (generalization-distribution-table based generalization for discovering if-then rules).

Knowledge-refinement agents acquire more accurate knowledge (hypothesis) from coarse knowledge (hypothesis) according to data change and/or the domain knowledge. KDD agents such as *IIBR* (inheritance-inference based refinement) and *HML* (hierarchical model learning) are commonly used for this purpose.

In terms of AI planning, no matter how many KDD agents we have, each of them is an *operator*. Each operator by itself can only do some simple thing, only when they are organized into a society, we can accomplish more complex discovery tasks. The KDD planner reasons on these operators to build KDD process plans – networks of KDD agents that will achieve the overall discovery goals when executed. But to apply AI planning techniques, we must be able to formally describe the KDD agents as operators. This is the subject of the next section.

Formal Description of KDD Agents

The KDD planner, as any AI planner, needs a World State Description (WSD) and a pool of Operators (Ops). We use the OOER (Object-Oriented Entity Relationship) data model to describe them. The traditional ER model has concepts of entity/relation, type/instance, instance-level attributes, and so on. The OOER model further incorporates object-oriented concepts such as subtyping, multiple inheritance, procedures, and type-level attributes/procedures, and so on. There are two kinds of types, *D&K* types and *Agent* types, for passive and active objects respectively. Figure 1 shows the (simplified) type hierarchy used in the GLS system:

The *D&K* types describe various data and knowledge presented in a KDD system. On the data part, we have *RawData* from the global information source,

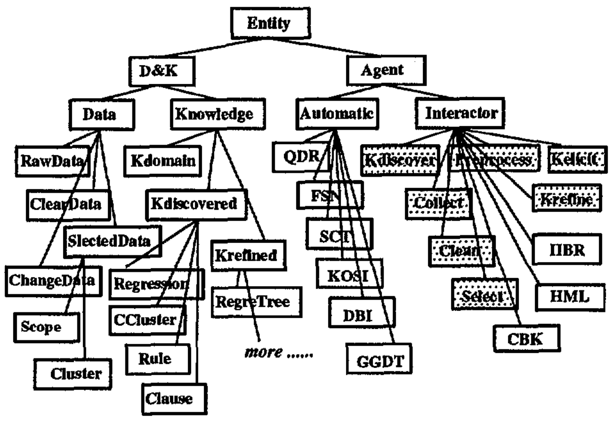

Figure 1: The type hierarchy of the GLS system

CleanData from the central large database, *SelectedData* (*Scope* or *Cluster*) from the subdatabases, and so on. On the knowledge part, we first distinguish among *Kdomain* (the background knowledge), *Kdiscovered* (the discovered knowledge) and *Krefined* (the refined knowledge). The type *Kdiscovered* has subtypes: *Regression* (structural characteristics), *CCluster* (conceptual clusters), *Rule* (if-then rules), *Clause* (predicate definitions), and so on. *Krefined* has subtypes *RegreTree* (family of regression models) and so on.

The *Agent* types describe various KDD techniques used in the GLS system. We distinguish Automatic (KDD algorithms) from Interactor (KDD techniques that need human assistance). *Kdiscover* means the overall KDD task, while *Preprocess, Kelicit* and *Krefine* stand for the three learning phases: pre-processing, knowledge-elicitation, and knowledge-refinement, respectively. *Collect, Clean* and *Select* are activities in *Preprocess.* Most agent types take the same technical names as mentioned above, such as *CBK, QDR, FSN, SCT, KOSI, DBI, GGDT, IIBR, HML.*

Note that in Figure 1, we show only the subtype relations among KDD objects (a subtype *is-a* special case of the supertype). For example, all of *Kdiscover, Preprocess, Kelicit, Krefine* are subtypes of Interactor. We will see below how to express the subagent relation, for example, *Preprocess, Kelicit, Krefine* are three subagents of *Kdiscover.*

Types have the ordinary instance-level attributes. For example, *D&K* has the attribute status describing the current processing status of the data/knowledge (created, cleaned, reviewed, stored, etc.), and this attribute is inherited by all subtypes of *D&K. Kdiscovered* has the attribute timestamps recording the time when the knowledge is discovered, and this attribute is inherited by all subtypes of *Kdiscover (Regression, CCluster, Rule,* and *Clause*).

As for Agent types, there are additional properties defined. For example, we may have type/instance-level procedures expressing operations on the types or instances (creation, deletion, modification, etc.). However, the most interesting properties of Agent types are the following type-level attributes with information that is used by the planning meta-agent:

- In/Out: specifying the types of the input/output of an agent type. The specified types are some subtypes of *D&K*, and the types of the actual input/output of any instance of the agent type must be subtypes of the specified types.
- Precond/Effect: specifying the preconditions for an agent (an instance of the agent type) to execute, and the effects when executed. Precond/Effect are logic formulas with the restrictions as in the classical STRIPS (see (Russell 1995), for example). A large part of the Precond/Effect, concerning constraints on input/output of the agent type, has been specified implicitly by the In/Out attribute. At planning time, the In/Out specification will be transformed into conjunctions of literals, then added to the Precond/Effect on which the planner reasons.
- Action: a sequential program performing real KDD actions upon agent execution, e.g. to call the underlying KDD algorithms. It is empty for high-level agents.
- Decomp: describing possible subtasking. High-level agents are allowed and should be decomposed into a network of subagents. Decomp specifies the candidate agent types for the subagents. For example, the Decomp for agent type *Kdiscover* is: *Preprocess, Kelicit, Krefine.* Next section will explain how the hierarchical planning utilizes this property to realize the task decomposition.

KDD Process Planning

The planning meta-agent (the planner) has three layers as shown in Figure 2. The inner layer is a domain-independent non-linear planner, the middle layer deals with KDD-specific issues, and the out layer interacts with another meta-agent, the controller, to realize hierarchical planning. In the following subsections we will discuss these three layers separately, and give an example of KDD process planning.

Non-Linear Planning

The inner layer as shown in Figure 2 is a domain-independent non-linear planner. Given the initial world state, the discovery goal and the pool of operators (KDD agents), it starts with a dummy partial plan (Russell 1995), then expands the partial plan until finds a complete and consistent plan that solves the problem. The process can be implemented by a production system (Liu 1991), or a nondeterministic algorithm (Russell 1995).

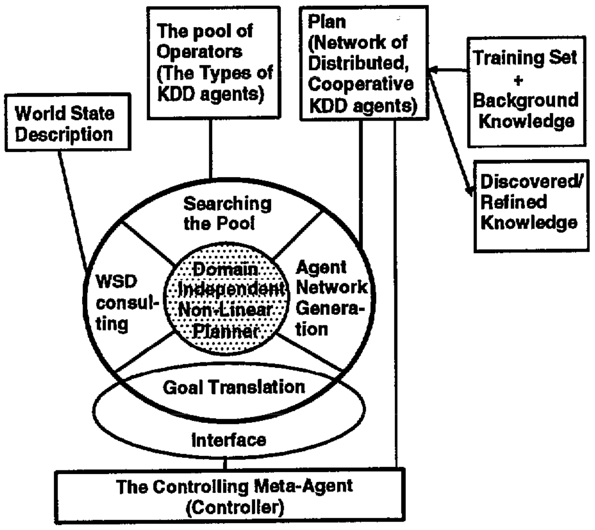
World State Description
The pool of Operators (The Types of KDD agents)
Plan (Network of Distributed, Cooperative KDD agents)
Training Set + Background Knowledge
Discovered/ Refined Knowledge
Searching the Pool
WSD consulting
Domain Independent Non-Linear Planner
Agent Network Generation
Goal Translation
Interface
The Controlling Meta-Agent (Controller)

Figure 2: Coupling of the planning and controlling meta-agents

Hierarchical Planning

As we are dealing with real world KDD applications, a hierarchy of abstractions is essential. The process of alternatively adding detailed steps to the plan and actually executing some steps should continue until the goal is achieved. In GLS, this hierarchical planning is accomplished by the cooperation of the two meta-agents – planner and controller. The interface between them is the outer layer of Figure 2. The two meta-agents interact as follows.

At the beginning, the controller generates a high-level KDD agent HA (*Kdiscover,* for example) with the discovery goal as its effect. This single agent HA can be regarded as the first coarse plan. When the controller tries to execute HA, it calls the planner to decompose it into a more detailed subplan. The planner works as described in the above subsection, taking the current world state as its initial-state, the effect of HA as its goal, and searching the types of subagents specified in the DECOMP attribute of HA, instead of the whole pool of operators, to achieve the goal. The produced subplan is added to the original plan, with each node linked to HA by a subagent relationship. Then the controller resumes its work (generating, executing, and controlling KDD agents according to the subplan).

Obviously this mechanism can work in multi-levels: if the controller meets another high-level agent when it executes the subplan, it will call the planner again.

KDD Specific Issues

Because the core of the planner is domain-independent, we provide a middle layer as shown in Figure 2 to deal with all KDD specific issues:

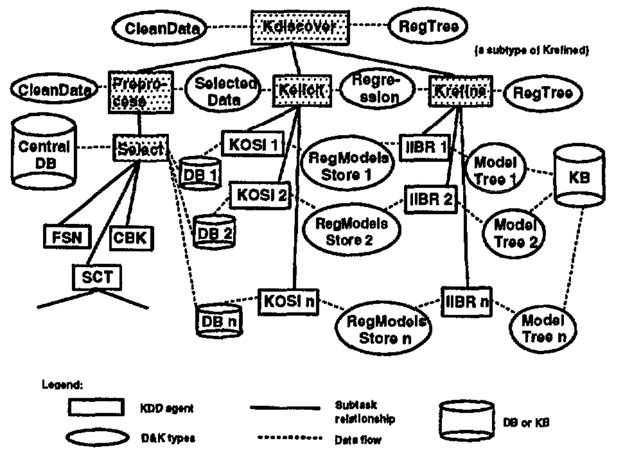

Figure 3: A sample KDD process plan

1. To transform the KDD goals into STRIPS goals (logic formulas in the style of STRIPS, that is, conjunctions of literals), especially to translate the input/output constraints specified in the In/Out attribute into Precond/Effect.
2. To search the pool of operators (or more exactly, to search the types of subagents specified in the DECOMP attribute of a high level agent HA in the decomposition process) to introduce suitable KDD agents into the plan.
3. To consult the world state description to see if a precondition is already satisfied by the WSD, and/or help to transform the In/Out specification into conjunction of literals as part of Precond/Effect.
4. To represent the resulting plan as a network of KDD agents, so the controller can dynamically generate and execute the KDD agents according to the network. The network can be also used by the user of the GLS system as a visualization tool.

An Example

Assume that we have a central, large space science database, each record (tuple) describing a star. The interesting attributes include CD (cluster designation), ET (effective temperature), LU (luminosity), B-V and U-B (color indexes). The facts such as we have already had a central, large database with *CleanData,* and the nominal attribute CD can be used for forming *Scopes,* etc. are explicitly stated in the initial-state (WSD). The discovery goal is to find structural characteristics hidden in the database and to refine them upon data change. Based on the specifications of WSD, goal, and KDD agent types, the planner and the controller cooperate in the manner as described above, and come up with a full KDD process plan as shown in Figure 3.

Conclusions

As the KDD process for real-world applications is extremely complicated, it has become a new and important research area, in addition to the traditional study on individual KDD techniques. In this paper, following our framework of KDD process and within the GLS system, we discussed in detail one of the meta-agents in GLS, the KDD process planning. We proposed a formalism to describe KDD agents, in the style of OOER (Object Oriented Entity Relationship data model). Based on this representation of KDD agents as operators, we apply several AI planning techniques: non-linear planning (partial-order planning) as the basic reasoning mechanism; hierarchical planning (task-decomposition) to tackle the complexity of KDD process; and replanning (or integration of planning and execution) to support data/knowledge/process evolution. The GLS system, as a general-purpose KDD system based on the framework and using the planning mechanism, increases both autonomy and versatility.

References

Brachman, R.J. and Anand, T. 1996. "The Process of Knowledge Discovery in Databases: A Human-Centered Approach", in *Advances in Knowledge Discovery and Data Mining,* MIT Press, 37-58.

Fayyad, U.M., Piatetsky-Shapiro, G, and Smyth, P. 1996. "From Data Mining to Knowledge Discovery: an Overview", in *Advances in Knowledge Discovery and Data Mining,* MIT Press, 1-36.

Liu, C. 1991. "Software Process Planning and Execution: Coupling vs. Integration", *Proc. the 3rd International Conference on Advanced Information Systems (CAiSE91),* LNCS 498, Springer, 356-374.

Liu, C. and Conradi, R. 1993. "Automatic Replanning of Task Networks for Process Evolution in EPOS", *Proc. the 4th European Software Engineering Conference (ESEC'93),* LNCS 717, Springer, 437-450.

Minsky, M. 1986. *The Society of Mind,* Simon and Schuster.

Russell, S.J. and Norvig, P. 1995. *Artificial Intelligence - A Modern Approach* Prentice Hall, Inc.

Engels, R. 1996. "Planning Tasks for Knowledge Discovery in Databases - Performing Task-Oriented User-Guidance", *Proc. Second Inter. Conf. on Knowledge Discovery and Data Mining (KDD-96),* AAAI Press, 170-175.

Zhong, N. and Ohsuga, S. 1995. "Toward A Multi-Strategy and Cooperative Discovery System", *Proc. First Inter. Conf. on Knowledge Discovery and Data Mining (KDD-95),* AAAI Press, 337-342.

Zhong, N., Kakemoto, Y., and Ohsuga, S. 1997. "An Organized Society of Autonomous Knowledge Discovery Agents", *Proc. First Inter. Workshop on Cooperative Information Agents - DAI meets Database Systems (CIA'97),* LNAI 1202, Springer, 183-194.

Zytkow, J.M. 1993. "Introduction: Cognitive Autonomy in Machine Discovery", *Machine Learning,* KAP, 12(1-3) 7-16.