Evolutionary Agent Societies Applied to Knowledge Discovery & Predictive Data Mining

Steve G. Romaniuk
ANSER
Fairmont, WV 26554, USA
Email: romanius@anser.org
http://nexos.anser.org:8080/java/app/

Abstract

This paper reports ongoing efforts in developing a Knowledge Discovery & Predictive Data Mining (KD&PDM) system at ANSER as part of an National Institute of Justice (NIJ) grant to locate missing and exploited children. The here in described system represents a first pilot project for exploring the general use of Evolutionary Agent Societies (EAS). EAS seeks to expand current efforts in IT software development, deployment, and maintenance by addressing such important factors as reliability, scalability, and adaptability of intelligent software.

Keywords: Evolutionary Agent Societies, Learning, Autonomous, Knowledge Discovery, Data Mining.

Introduction

The process of locating missing and exploited children by both State and Federal investigative offices is often marred by inadequate manpower support. Modern computer technology – especially the Internet and agent-based software – can help enhance ongoing investigative efforts. At the same time, it can be expected that a number of technologies developed for locating missing and exploited children, can also be applied to other areas of law enforcement. ANSER, as part of this overall effort, is developing technology for the automatic analysis of case-related information via expert/planning systems, autonomous intranet/Internet search & data harvesting agents and automated tools for Knowledge Discovery & Data Mining.

Introduction to EAS

Overview

Evolutionary Agent Societies (EAS) encompass software systems, in which individual processes exhibit agent-like behavior (Nwana and Azarmi 1997) (i.e., communication, mobility, etc.) and agents are capable of forming social structures (i.e., cooperate, form relations of differing intensities and frequencies, etc.). Formation of these structures is facilitated via individual agents *need* and *capabilities*. The basic building block of an EAS system consists of a collection of agents, which share a common Agent Framework and can be placed within groups of similar acting/performing agents (I.e., Data Miners, Data Prospectors, etc.). The EAS Agent Framework is composed of four components:

1. Communication
2. Mobilit
3. Reproduction
4. Persona

Agent Framework Components

A brief overview of the four agent framework components is provided.

Communication

The communications component is based on a generic message passing system. Messages can arrive asynchronously or be processed synchronously (I.e., during a protocol handshake). A message datum consists of a simple string. Every message center supports message in/out boxes. Messages are processed based on an agent's assessment/ranking of the originating agent (i.e., how reliable an information source it is). controller is responsible for adding, removing, or prioritizing messages. The controller is also responsible for acting upon timing events generated by an internal timer (I.e., initiate periodic communications protocols like *AgentMeet*). The number of messages stored/acted upon and/or the frequency of protocol invocation is determined by an agent' *Persona*. The agent's communications component is abstracted into three separate layers:

1. Communication Protocols/Services

Is responsible for identifying agents on the current system host, registering/un-registering of agents, processing subscriptions (based on their basic type), soliciting mating agents, synchronizing agent list (*known*, *collaborator*, *trusted*, etc.), handle sending/receiving of messages, and binding agent capabilities/needs. Furthermore this component is responsible for launching and managing threads to execute protocols for the identification of agents, solicit mating procedures, message receiving, and agent registration. Protocols can either be classified as

periodic (I.e., *AgentFind*, *AgentMeet*, etc.), or event driven (I.e., subscription processing).

2. Conversational Protocols/Services

Manages conversational protocol threads (I.e., Receive & Send). Typical protocols include Hello-Ask, Say-Hello-Ask, Hello-Reply, Subscription, Access-Host, Launch-Agent, Procreate, Agent-Suspend, Agent-Resume, Agent-Shutdown, etc.

3. Transport Layer

Determines how messages are physically transported from one agent to another. Currently supports 4 mechanisms:

- Via ODBC Database (using SQL-92 standard for DB query language).
- Via Java/RMI.
- Via connectionless sockets (datagrams).
- Via a shared file system.

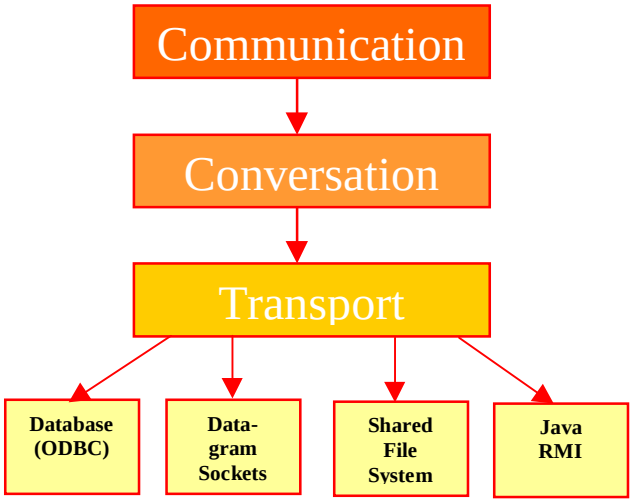

Figure 1. Agent Communications

Figure 1. displays a high level outline of the agent communications components.

Mobility
Is responsible for transporting an EAS agent from one computer host to another. Two schemes for mobility are considered:

- Body/Persona: The agent is restarted on the new host and terminated on the old. The agent's persona is saved to persistent storage and reloaded during the agents' restart on the new host (only agent's persona is actually transported, not the body).
- Full : All agent services are stopped (I.e., threads) and the agent is moved in its entirety to the new host (using Java/RMI). The agent is terminated on the old host and its services are restarted on the new host.

Both approaches provide a tradeoff between computational resources and agent design considerations.

Reproduction
To generate cyber-diversity in the EAS system, it i necessary that individual agents of the same type procreate. Agent Persona traits are encoded as individual genes or gene groups. Continuous parameter genetic algorithms are utilized. Value extrapolating gen blending (within the inherited species and outside the parental value bounds) is invoked during crossover. Agents procreate on the basis of their explicit and implicit fitness functions. Population control is administered via a local host Monitor Agent. Agents can only be created/executed if resources are available (currently only supported by TDL agents (Romaniuk 1993, Romaniuk 1995)).

Persona
Defines an agent's general and specific capabilities. This feature is partially inherited from an agent's parents and modified during an agent's existence through it personal experiences. It embodies all pertinent behavioral characteristics an agent will display during it lifetime. Example persona traits:

- Inherited: Reproduction, Longevity, Curiosity, Approachability, Cooperation, etc.
- Adapted: Frequency of Reproduction, Problem Solving Strategies, Degree of cooperation, Degree of curiosity, Modes of Communication, etc.

Inherited traits may subsequently be modified based on environmental conditions. For example, reproduction is largely inherited, but its frequency can change dependent on other agent persona traits (I.e., agent's ability to attract others for procreation). All agents inherit some common traits, whereas others are unique to each agent type.

EAS Explained

EAS Structural Components
Figure 2. displays a specific instance of an EAS structure. The structure consists of an arbitrary number of layers. Individual layers are composed of one or more groups. Agents within an individual group are generally of similar type and may either decide to cooperate

amongst each other, or not. Default EAS structure consists of a single layer with a single group of agents. There is no limitation with regard to the number of groups per layer, or the total number of layers, except for those imposed by design considerations and resource limitations. Agents can cooperate amongst one another either across layers or within groups. Individual agent have the capability to reproduce and create new offspring in a variety of ways (asexual, bi-sexual, etc.). Groups of agents located on the same EAS layer can engage in co-evolution. Even though agents in different groups do not cooperate with one another, they may still be competing for shared resources (I.e., different types of data mining agents competing amongst each other).

As Figure 2. points out, groups of agents can collaborate with other groups (communicate and exchange information), as long as these groups do not reside on the same layer. The connections emanating from individual groups indicate cooperative efforts between these groups. Just because two or more groups are connected in this manner, does not imply that either

- All agents within a given group collaborate with all agents in another group.
- The collaborative links are permanent.

As a matter of fact, collaborative links only act as suggestions, which are conveyed via the EAS systems *Capabilities/Needs* communications mechanism. At any point in time, dependent on environmental conditions (i.e., availability of computing resources) and the current state of the EAS system (i.e., number and types of agents available), collaborative links may be formed or dismissed. Furthermore, cooperation of agents across groups is influenced by individual agents' personas. For example, agent characteristics favoring a high degree of collaboration will result in strong and manifold links. In other words, it is quite possible that even though collaborative links (i.e., connections) exist between groups, only few agents within such groups actuall engage in this cooperative exchange. Other group agents may simply choose to ignore these collaborative efforts.

By extending current terminology of EAS structures and components, a connection with evolutionary systems can be made. Groups of agents can be viewed as individual populations of a species. Collaborative links default to symbiotic and/or parasitic relationships between populations. Finally, populations of similar agents ma co-evolve (groups of agents on same EAS layer).

As a consequence of the above extension, it may very well follow that a group of agents that enjoyed a high degree of cooperation with another group in the past will loose this distinction. This loss will be the direct result of natural selection amongst group individuals whose genes favor not to cooperate.

Creating structure in EAS

The EAS example depicted in Figure 2. has no structure beyond the initial layout of collaborative links between agent groups. For example, no detail exemplifying information flow between individual agent groups is supplied (i.e., there is no concept of up and down). Structure is added to EAS systems by adding end-consumer processes. These may be agents themselves or applications in the traditional sense. The example of Figure 3. shows two such processes attaching themselves to the earlier EAS structure. Agent end-consumers can provide feedback to those agent groups of the EAS structure they attach to. This feedback will most likel consist of subjective values they assign to th information obtained from the agent groups. Agent groups receiving external feedback as to the information they generate will propagate this information further down the information food chain. Consequently, collaborative links will either be strengthened, weakened, or may all together disappear. This dynamic interplay eventually assigns structure, in terms of information flow, to an EAS system. Naturally, as processes attach or detach themselves from an EAS this structure will change.

Embedding EAS

It was pointed out that the individual nodes within a group correspond to one particular instance of an agent. But what constitutes an agent in the first place?

At the beginning of this section, 4 components wer listed as important requirements for an EAS agent: communication, mobility, reproduction, and persona. Unfortunately, this still does not answer what constitutes an agent in terms of its capabilities. Is it simply an object, a component, a full-blown application, or a complete suite? The answer to this question is simply all of the above. The EAS architecture does not place restrictions on agent *size*. Of course, this does not mean that any size is a valid solution for all problems. Rather environment, intention (by the original designers), and performance matters. If the agent granularity is too high, massive resource consumption in the form of communications and other agent framework overloads will rear their ugly head. On the other hand, if the granularity is too low, lack of agent diversity (due to larger program sizes and execution requirements) can also be a major pitfall. To use nature as an analogy, organism biometries are very diversified, ranging from simple one-cell organisms to complex multi-cell organisms.

Returning to Figure 3., we can state that an individual agent within a group, can be as simple or as complex a desired. For example, a single agent could consist of an application that performs some form of pattern recognition (i.e., facial, speech, handwriting, ATR, etc.).

At the same time, the EAS system depicted in Figure 3. can also represent a complete pattern recognition application. In this particular case, agents within groups may implement various pattern transforming, scaling, or mapping operations on top of implementing diverse recognition strategies. Using the latter case as a starting point it follows, that EAS systems can be embedded within one another. Each node in Figure 3. could be an atomic agent or a complete, self-contained EAS system.

Knowledge Discovery & Predictive Data Mining

Overview

Data mining systems work best on known information sources. In other words, they rely on human managers to turn unstructured or semi-structured information sources into *agent sources*. Typical approaches utilize ontologies to provide a common interface for knowledge representation and discovery (Bigus 1996, Fayyad et al. 1996, Weiss and Indurkhya 1998]. There is nothing wrong with this approach, as long as one seeks to *agentify* information from known and analyzed data sources; and there is enough manpower and time for creating the required ontologies. Unfortunately, thi approach fairs less well for all those cases that rely on harvesting unknown data sources. Knowledge discovery seeks the exploration of new and untapped data sources and elucidating hidden as of yet unknown relationships. These relationships can be restricted to within a data source, or – more interestingly – branch across man different data sources. In any event, automating th process of knowledge discovery (KD) and subsequently performing predictive data mining (PDM) is at the heart of the EAS-DM system.

KD & PDM Agent Types

The EAS Data Mining system (EAS-DM) consists of several different types of agents (to be discussed in more detail latter on). Two types of agent populations are embedded in the current configuration:

1. Prospector agents
2. Mining agents

Figure 4. outlines a high-level description of the overall system architecture in terms of process flow, wherea Figure 5. depicts the same information by focusing on data flow.

Knowledge Discovery

Initially, an autonomous population of Prospector agents is engaged in data mining of existent ODBC-compliant databases (these databases have either been identified by humans or via a Surveyor agent). The knowledge discovery strategies exploited by the Prospector agent are discussed in (Romaniuk 1998). Similar to conventional data mining tools, the Prospector agent attempt to find interesting relationships between groups of database table fields (both intra- and inter-table relations) and format these in a standard symbolic data definition format.

Predictive Data Mining

Data-mining agents on the other hand can subscribe and receive data definition files, which are generated by the Prospector agents. The first step that is performed by Mining agent, is to convert the generic data definition file into a specific pattern file suitable for supervised learning from examples (Figure 5.). The Mining agent employed in the EAS-DM system is based on an incremental Instance-Based Learning (IBL) algorithm. The system learns by automatically forming instance neighborhoods in the form of n-dimensional hyper boxes. See (Romaniuk 1994) for a detailed discussion of this type of IBL system. The IBL system used in this work is augmented to support extraction and refinement of symbolic rules and is characterized by a fast – provably deterministic - learning approach (important to the data-mining task undertaken in this project). Figure 5. lists an example pattern file generated by the Data Mining agents. Notice that the original data is directl encoded into numeric data. Depending on the type of pattern recognition algorithm employed by the Data Mining agents, it may be necessary to encode non-numeric features. Encoding of features is based on an agent's internal biases and can support *independent feature* and *dependent feature* coding. The final output of the Miner is a file of rules deemed relevant by the agent. These rules can either be used by SADIE agents (support their decision making process), or be viewed/enhanced by the Knowledge Base (KB) browser and/or search engine. Screen shots of the latter two are provided in Figures 6 & 7. The KB Browser provides a knowledge engineer with several ways for viewing and searching the KB. The browser is passive, since it does not directl interact with the EAS system. The KB Search Engine on the other hand (Figure 7) allows the knowledge engineer to supply *hints* to the EAS-DM system via the concept and relation search engines. Results found by the EAS agents will be posted to a discovery window, where the user can inspect the discovered rules. Included are information such as rule types and their relative frequencies.

Agent Performance Feedback

For EAS-DM to function properly it is necessary to supply feedback to individual agents. This performance assessment can come in 2 flavors:

1. Explicit

Explicit feedback is directly propagated from one agent to another (different layer group). It captures the overall validity of the information obtained from the data generated by the source agent and utilized by the destination agent.

2. Implicit

Implicit feedback is generated within agents themselves and represents a form of self-worth that the agent has created of itself. For example, a rule-generating agent may be biased (directly due to its persona characteristics and indirectly through its gene traits) to favor the generation of simple rules (limited complexity of rule premise). Based on the types of rules this agent generates over time, it can derive an internal measure of their quality (at least in terms of the rule premise complexity). If over time, an agent's performance fails to meet internally set goals, its self-worth (implicit fitness function) will drop. Eventually, an agent's lack of tolerance to a continual drop in self-worth can result in its elimination from the population.

Explicit Feedback Sources

Explicit feedback can be provided either by expert system–like agents, which are direct consumers of EAS-DM generated rules, or via one or more knowledge engineers using the KB search engine to score rules, or initiate (guide) the search for specific rule knowledge.

Hints in EAS-DM

Overview

Users can also provide feedback in the form of *hints* to a Prospector population. *Hints* can contain suggestions with regard to:

- What table fields have the highest potential to act either as dependent or independent variables during prediction?
- Which tables are most suitable for table join operations?
- Which databases may contain relevant information that can be gleaned by merging individual tables?

As the name indicates, *Hints* are only suggestions and individual agents (based on their persona) may decide to reject them altogether, whereas other agents may follo them religiously.

Current State

Every *Hint* is preceded by a *Hint Marking Tuple*, which contains a time stamp, a unique creator ID, a status flag, a process count, and a max. process count.

Two general types of hints are provided:

- *Skip* hints announce databases, tables, or fields, which should be ignored.
- *Seek* hints either references individual concepts o relations between concepts that should be pursued.

The following snippet of code provides an overview of a hint file for a database source **SDP** and tables **Employees** and **Orders**.

```
[905442204080,0,0,0,1] SKIP
        SDP.Employees.LastName;
[905442206610,0,0,0,1]  SKI
        SDP.Employees.FirstName;
[905442210180,0,0,0,1]  SKI
        SDP.Employees.EmployeeID
                AS PREDICTEE;
[904684794700,0,0,1,1]  SEE
        SDP.Orders.RequiredDate;


[905088362000,0,0,0,1]  SEEK RELATION
                OR (SDP.Employees.BirthDate,
                SDP.Employees.HireDate  )
                IMPLIES SDP.Employees.Title;
```

The last hint attempts to seek out rules which either relate an employees birth date to his/her title, or relate an employees hire date to the very same title.

Future Extensions

Currently, the EAS-DM system can not perform discoveries, which attempt to relate concepts that ma differ due to naming conventions. Many times abbreviations, context specific terminology, transcription errors, etc. give rise to variations in concept labeling. Naturally, a simple string comparison will fail promptly. To address these concerns, it is proposed to reference individual concepts during a search (i.e., discovery) via 5 direct relations:

1. synonyms
2. antonyms
3. hyponyms (subordination)
4. hypernyms (superordination)
5. meronyms (HASA/part-of relation)

By using the above knowledge to discover new concept/relation descriptions from existing data (DB table & field names, and textual field content), one should expect improved performance and robustness of the underlying discovery system. For example, knowledge of a concepts hyponyms and hypernyms can help establish concept clusters for clustering text,

whereas knowledge of a concepts synonyms and antonyms can help label text as either positive o negative examples of these clusters. Additional knowledge – based on psycholinguistic analysis of word sense frequencies of English language words – can help derive non-crisp membership values for individual text segments during cluster assignment.

Complete EAS-DM Agent Overview

EAS-DM defines 6 different agents. Each is briefly discussed with respect to its behavioral patterns.

Java Agents:

Surveyor Agent: an agent capable of traversing intranets and the Internet. Its purpose is to identify data sources (i.e., databases, knowledge bases, etc.) Uncovered sources are forwarded to the Prospector agent.

Prospector Agent: The agents' task is to perfor knowledge discovery and pre-format encountered data so it can be further processed by Mining agents.

IBLMiner Agent: Its goal is to identify possibly interesting relations contained in data and output it in a human or machine-readable format (i.e., rules for expert system shells).

SADIE Agent: A cyclic, hierarchical, adversial planning agent. These agents have knowledge bases of concepts, goals, and events and a rule base that may perform both crisp and fuzzy inferences. SADIE can function as a rule-based expert system. The format of the rules generated by the IBLMiner mirror the one used by SADIE.

Monitor Agent: Responsible for monitoring system resources and granting access to a network host's resources to other EAS agents. Monitors are installed on each host of a network. Collaboration among agents is restricted to those agents running on the same virtual network. Virtual networks are created via a common shared access list. In other words, the hosts that make up a physical LAN can be grouped to form any number of virtual networks. Creating separate access lists forms different -disjoint - virtual networks. Utilizing disjoint virtual networks is a natural way to provide support for evolutionary island theory, since no agent contact occurs across virtual networks. Future extensions will provide virtual network overlaps (via gateway hosts) and host migration (dynamic re-assignment of hosts between virtual networks). Finally, Monitor agents can suspend/resume/shutdown individual agents residing on the monitored host, or if necessary perform the sam operations on other Monitors (and their agents). Figure 8. previews a Monitor agent. The virtual networ consists of three hosts, each executing a Monitor, Prospector, and IBLMiner agent. One of the computers also hosts a Surveyor agent. Surveyor agents are mobile and can hop from one host to another. This feature is necessary in order to discover data sources suitable fo mining.

Scavenger Agent: cleans up intermittently generated data files/cookies etc., which where not properly removed during the generating agents existence.

Non-Java Agents:

TDLMiner Agent: TDL mining agents are based on a self-growing neural network architecture. Genetic algorithms are employed for 3 distinct tasks: problem decomposition, transfer function parameter discovery, and evolving agent persona.

Problem Decomposition: Relies on a standard canonical genetic algorithm to decompose any given training set into more manageable chunks (which are easier to learn).

Transfer Function Parameter Discovery: N-Level threshold functions (discreet and continuous flavors) have their parameter settings evolved.

Agent Persona: Evolves individual agents via a continuous-valued genetic algorithm. Parameters are grouped into general and specific groups. The general group contains parameters such as number of direct collaborator agents, frequency of reproduction, etc. The second group consists of parameters such as: type of learning rule, types of transfer functions, type of crossover operator, etc.

TDL agents cooperate amongst one another when learning new data. Whenever an agent discovers a new data set to be mined, it consults other agents, which are in its collaborator/friend group to determine if any of these agents might be more suitable for mining said data. The collaborator group is constantly updated depending on the number of non-responses, or the quality of a response. If the agent perceives a collaborator agent to be a better choice for mining the data, it hands over the data. TDL mining agents can either be controlled by the Monitor agents, or perform their own load balancing. Agents reproduce sexually, requiring two parents. Mates are chosen from the collaborator agent list. TDL agents continuously save their learned knowledge to persistent storage, where it can be readily accessed by other agents, nearly eliminating information loss due to unanticipated system events. Finally, these agents also support training of weightless and semi-weighted neural networks, co-

evolution of transfer functions, and forgetting of knowledge.

TDLCollector Agent: Collector agent's pickup networks created by mining agents and combine them into a single network. These agents also act as Expert Information Network (EIN) oracles. In other words, agents can supply pattern data and a context and have the oracle return the classified results.

Figure 9. shows three cooperating TDL Miner agents and one collector agent.

Implementation

The EAS agent framework (AF) is written entirely in Java. In order to support legacy systems (such as TDL (Romaniuk 1994, Romaniuk 1995)), that is to allow limited communication between Java agent implementing the AF and those that do not, two of the communication transport layers are implemented via a shareable database and file system. Furthermore, TD mining agents consume the very same numerical data files, which are created by the IBL mining agents. TDL agents have their own persona (currently consisting of 15 evolvable parameters). Also, TDL agents cooperate amongst one another when solving pattern recognition problems. TDL collector agents act as sweepers, picking up TDLMiner generated neural networks and fusing them into a single expert information network (EIN). EINs can act as oracles, which in turn can be queried b tools like SADIE. Tighter integration of TDL agents is still being pursued.

Finally, four additional pattern recognition strategies (i.e., Version Spaces, Bayesian Networks, etc.) will be added to the EAS system, to further increase the strategy pool.

Experiments

Though the EAS KD & PDM system is still in development, preliminary tests were conducted to both evaluate the systems correctness and ability to solve KD & PDM tasks via parallel processing.

Objective
Determine how well Prospector and Mining agents can cooperate on a set of distributed databases, by varying the number of agents and hosts. The goal is to measure the total time it takes to solve a specific KD & PDM task for different configurations.

Data Description
A total of 5 databases were used in this experiment. On database consisted of data dealing with missing children, whereas the remaining 4 included standard pattern recognition tasks (data was obtained from the UCI Repository of Machine Learning Examples). The 5 databases contained 14, 1, 8, 1, and 6 tables, respectively. Databases were accessed using the JDBC/ODBC bridge and stored as Microsoft Access DBs.

Task Description
Hints were supplied to the system for each of the databases and tables. All hints were of the form: *Seek specific table fields as predictees using all other fields as potential predictors*. Using these hints, the Prospector agents generated a total of 56 symbolic definition files (see Figure 5 for an example). Correspondingly, a total of 56 numerical data files and 56 rule files were generated by the IBLMiner agents (again refer to Figur 5 for examples of these files).

Hardware Setup
A virtual network consisting of 3 LAN connected Windows-based computers were used in this experiment. Two of the computers executed Windows NT 4.0 operating system (P6-333Mhz, 64Meg RAM, P6-233Mhz, 32Meg RAM), whereas the remaining machine ran Windows 95 (P6-233Mhz, 64Meg RAM). Communication occurred via datagram sockets.

Results
For a single machine running Monitor, Surveyor, Prospector, and IBLMiner agents (one of each), the time to complete the specified task was 4 hrs and 32 min. For two machines (second machine hosted one Monitor, Prospector, and IBLMiner agent) processing tim dropped to 1 hrs and 42 minutes. Finally, by adding a 3rd machine (also hosting one Monitor, Prospector, and IBLMiner agent) processing time further dropped to just 1 hrs. The results are averaged over three trials.

Interpretation
Though the results indicate impressive speedups, caution has to be taken when generalizing from these results. Firstly, the learning task was only moderate in size. Secondly, it is a well known fact that Java performance is strongly dependent on the type of operating system used and the amount of multi-threading performed. The EAS Agent Framework makes extensive use of multi-threading, which can lead to significant performance degradation on Windows 95 machines. Finally, differences in CPU speed and minimum RAM size need to be taken into account.

Extensions

Experiments on larger databases and a dedicated LAN of about a dozen machines is currently being prepared. The mix of computers will be more uniform to ensure less potential fluctuations in results.

Related Work

The EAS-DM system expands on earlier work by the author on utilizing EAS for automating the predictive data mining process (Romaniuk 1995). A prototypical system for the Windows 95 & NT platform was constructed around a population of TDL (Romaniuk 1993) (Trans-Dimensional Learning) agents. These agents are capable of learning, cooperating, and reproducing. Furthermore, TDL agents are adept at learning pattern associations across databases. In other words, data pattern sources of different dimensional specifications can be learned and represented within a single homogenous neural network. Finally, the various network structures, which have been learned by individual TDL agents, can be fused by Collector agents into a single network.

The Amalthaea system (Moukas and Zacharia 1997) is an example of an evolving, multi-agent ecosystem fo personalized filtering, discovering, and monitoring of information sites. Amalthaea's ecosystem provides an environment endowed with limited resources and supports evolving and cooperative agents. Thi particular architecture is based on two types of agents: (a) Information Filtering and (2) Information Discovery. Filtering agents keep track of user interests, whereas discovery agents handle actual information resources including adapting to them.

Moriwaki et al. (Moriwaki et al. 1998) explore the use of co-evolution via their nBDD gene expression. Their study favors nBDD for quicker adaptation of agents compared to finite state automatons.

Finally, Devine et al. (Devine, Paton, and Amos 1997) present a system which uses evolutionary agents to model ecological simulations via classifier-based animats.

Conclusions

EAS focuses on the development of an agent framework for evolving societies of adaptive software agents, which display competitive behaviors akin to those observed in natural organisms. This paper presented further insight into the application of EAS to knowledge discovery and predictive data mining. The EAS-DM system consists of several types of agents, which are either engaged in data source identification, knowledge discovery, predictive data mining and knowledge extraction, expert system/planning, and monitoring of system resources. Even though this article concentrated on a particular application of EAS, the framework attempts to cover far wider range by applying it to general softwar process distribution, control, and adaptation.

Acknowledgments

Supported under Award number 98-LB-VX-K021 from the Office of Justice Programs, National Institute of Justice, Department of Justice. Points of view in this document are those of the author and do not necessarily represent the official position of the U.S. Department of Justice.

References

Bigus, J.P. (1996) Data Mining with Neural Networks, McGraw Hill Press.

Devine, P., Paton, R., Amos, M. (1997) Adaptation of Evolutionary Agents in Computational Ecologies, BCEC-97, Sweden.

Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (1996) Advances in Knowledge Discovery and Data Mining, MIT Press.

Moriwaki, K., Inuzuka, N., Yamada, M., Seki, H., Itoh, H. (1998) A Genetic Method for Evolutionary Agents in a Competitive Environment, I. Chawdry, P.K., Roy, R., Pant, R.K (eds.) Soft Computing in Engineering Design and Manufacturing, Springer-Verlag.

Moukas, A., Zacharia, G. (1997) Evolving a Multi-agent Information Filtering Solution in Amalthaea, in Agents'97 Conference Proceedings.

Nwana, H. S., Azarmi, N. (1997) Software Agents and Soft Computing: Towards Enhancing Machine Intelligence, Springer Publishing.

Romaniuk, S.G. (1994) Efficient Storage of Instances: The MultiPass Approach, in Procee. 7th Inter. Conf. O Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Austin, Texas, June, 1994.

Romaniuk, S.G. (1995) TDL Agents applied to Autonomous Data Mining, White Paper: Universal Problem Solvers, Inc. (Also see *Through the Looking Glass…* in TDL's Windows Help System, http://pages.prodigy.com/upso).

Romaniuk, S.G. (1993) Trans-Dimensional Learning, International Journal of Neural Systems, Vol. 4, No. 2, pp. 171-185.

Romaniuk, S. G. (1998) Evolutionary Agent Societies. In: P.A. Flach & R.A. Meersman (eds.) 4th International Conference on Information Systems Analysis and Synthesis, July 12-16. 427-438, Orlando, Florida. (http://nexos.anser.org:8080/pubs/).

Weiss, S. M., Indurkhya, N. (1998) Predictive Data Mining, Morgan Kaufmann Publishers, Inc.

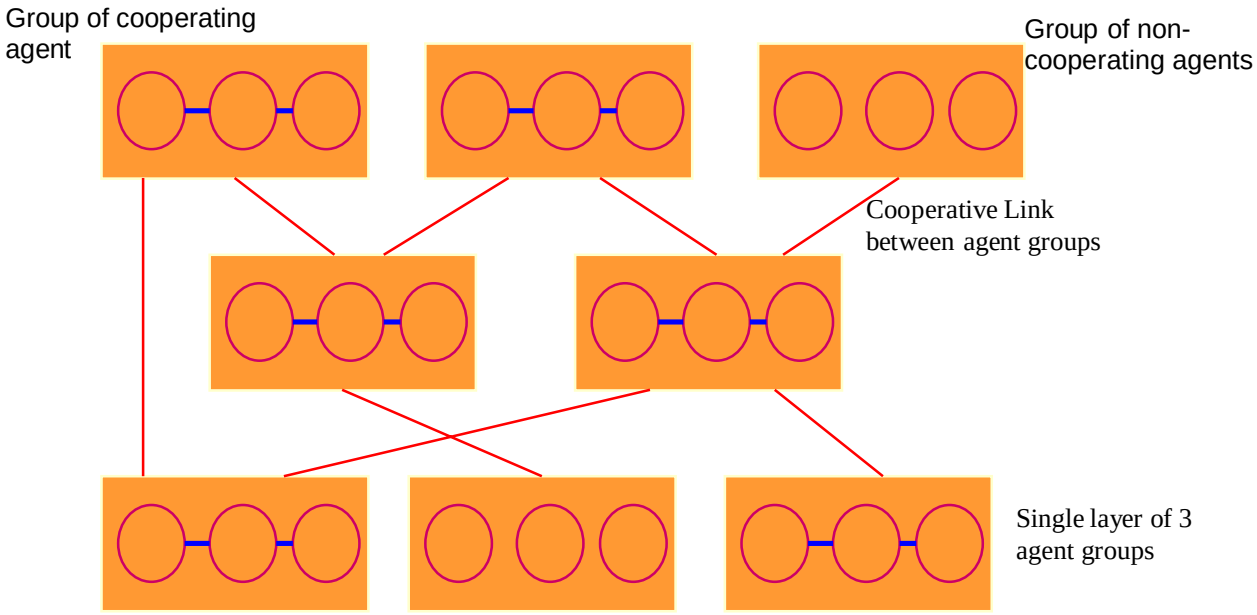

Figure 2. Example EAS system without process attachment

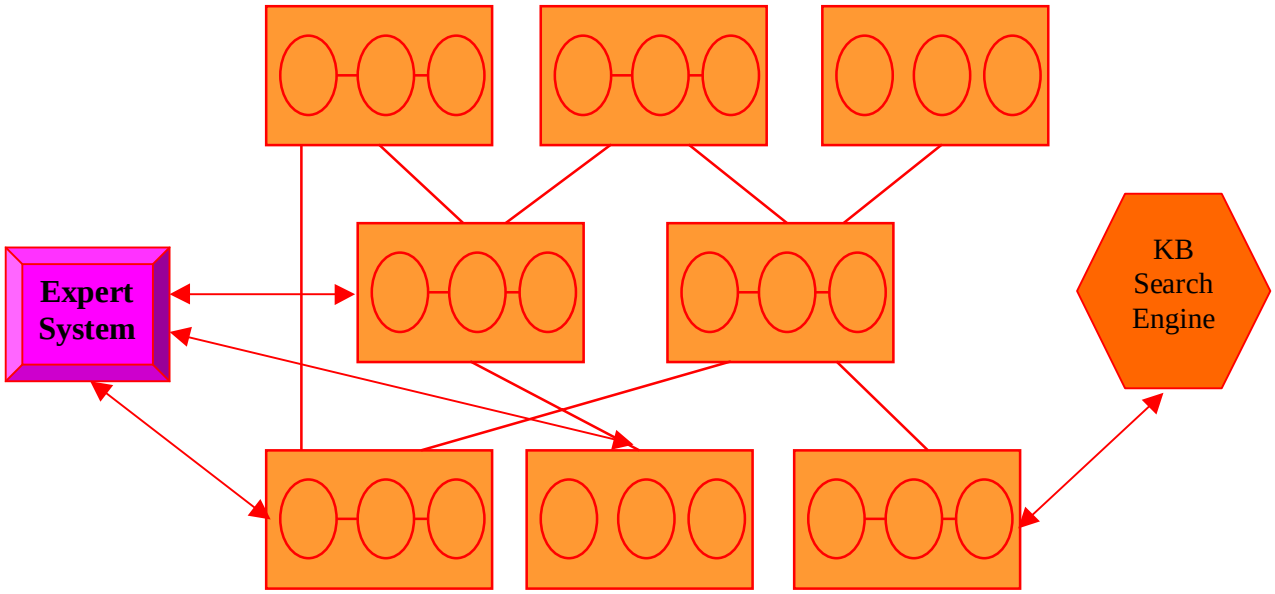

Figure 3. Example EAS system with process attachments

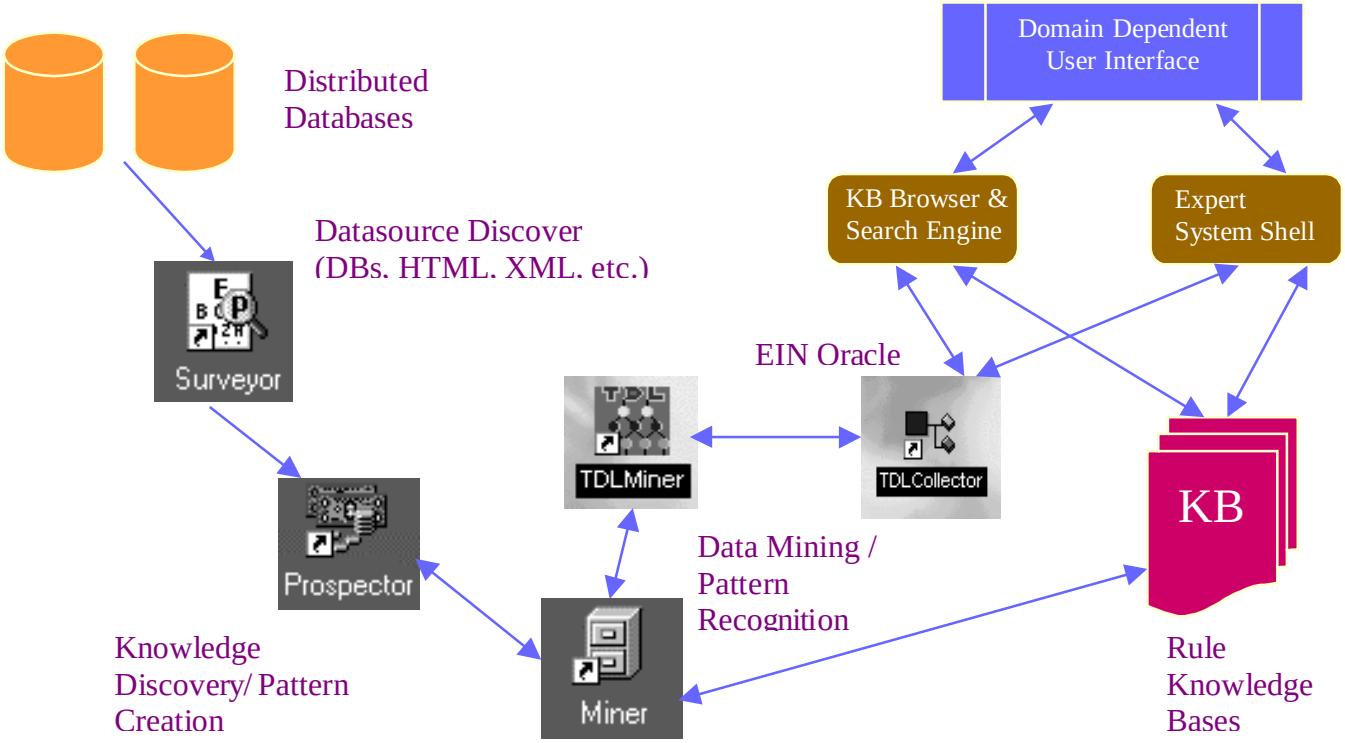

Figure 4. Example of KD & PDM process flo

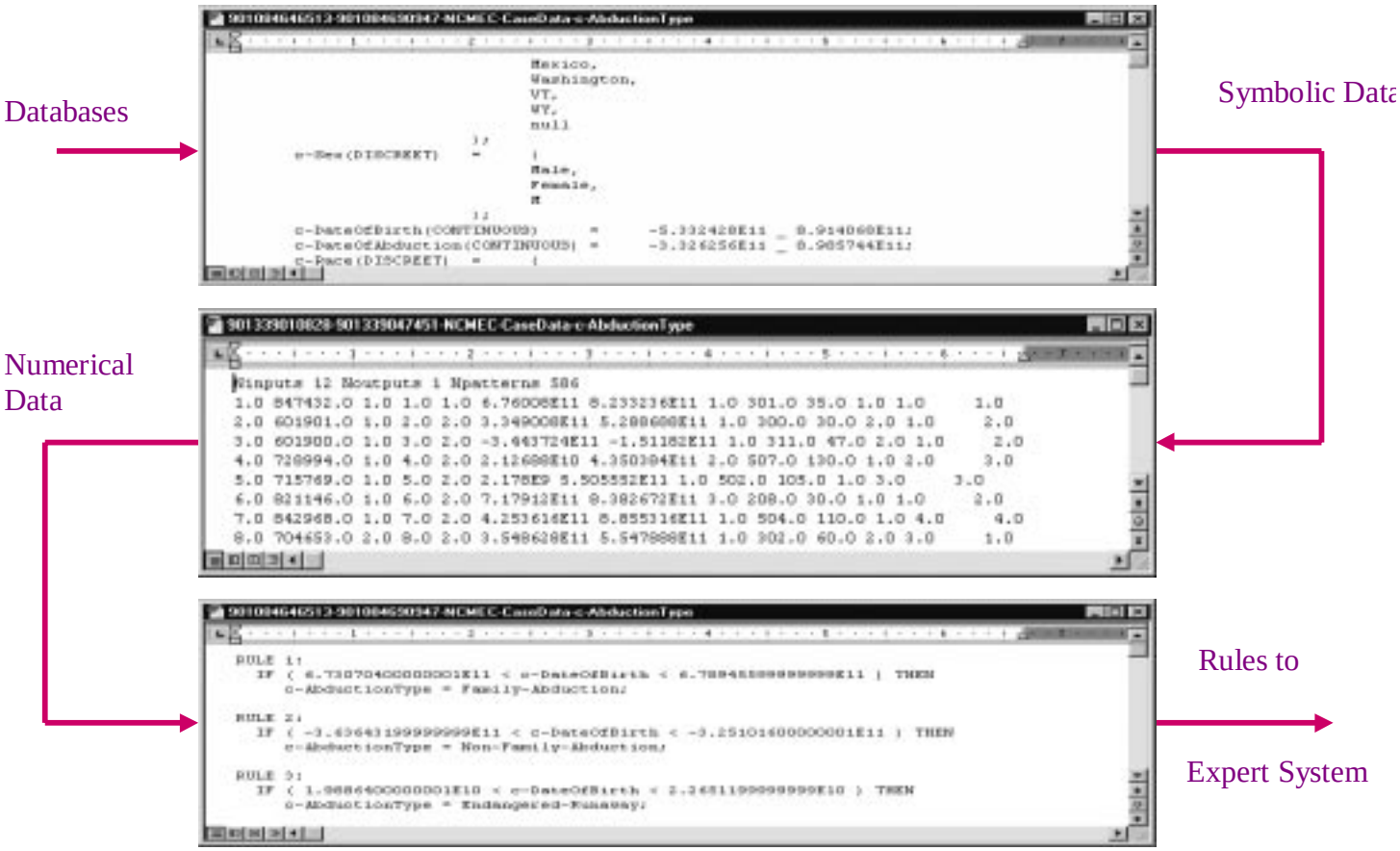

Figure 5. Example of KD & PDM data flo

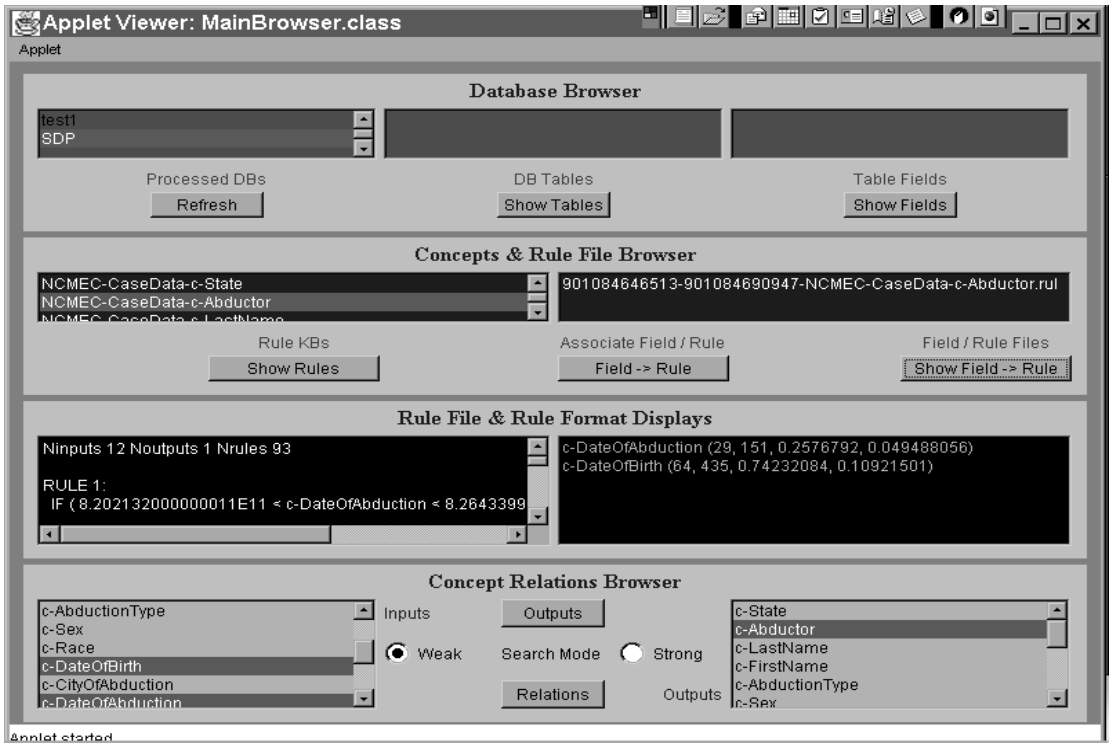

Figure 6. Example snapshot of Knowledge Base Browser

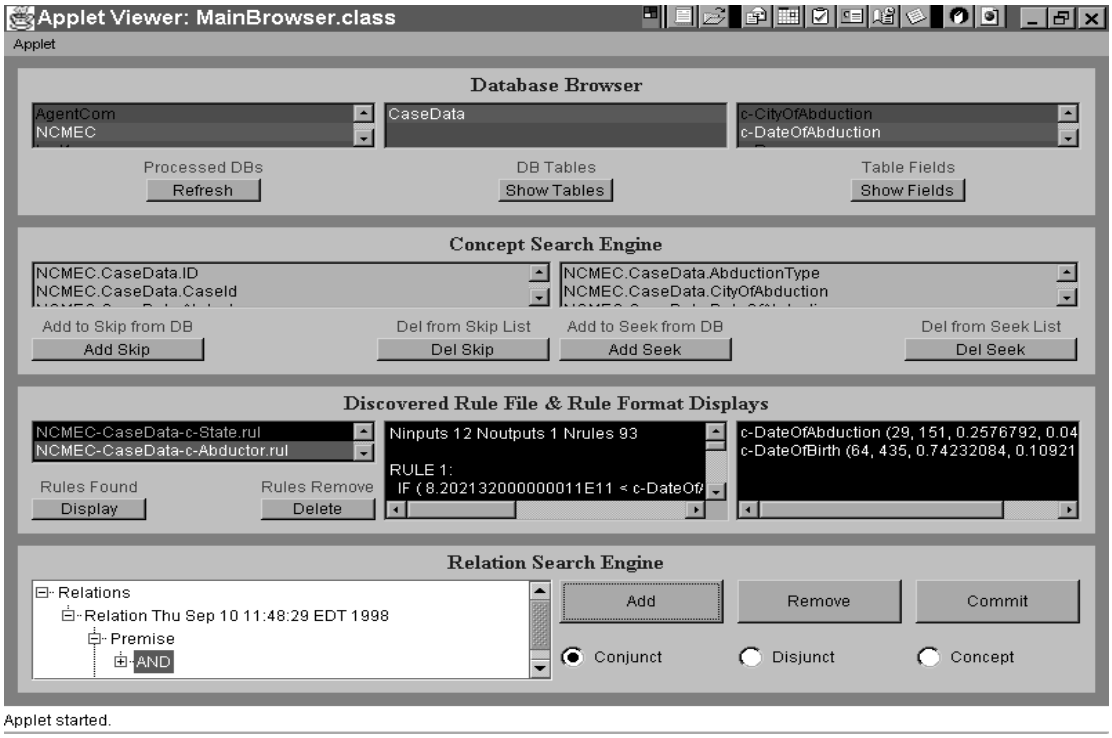

Figure 7. Example snapshot of Knowledge Base Search Engine

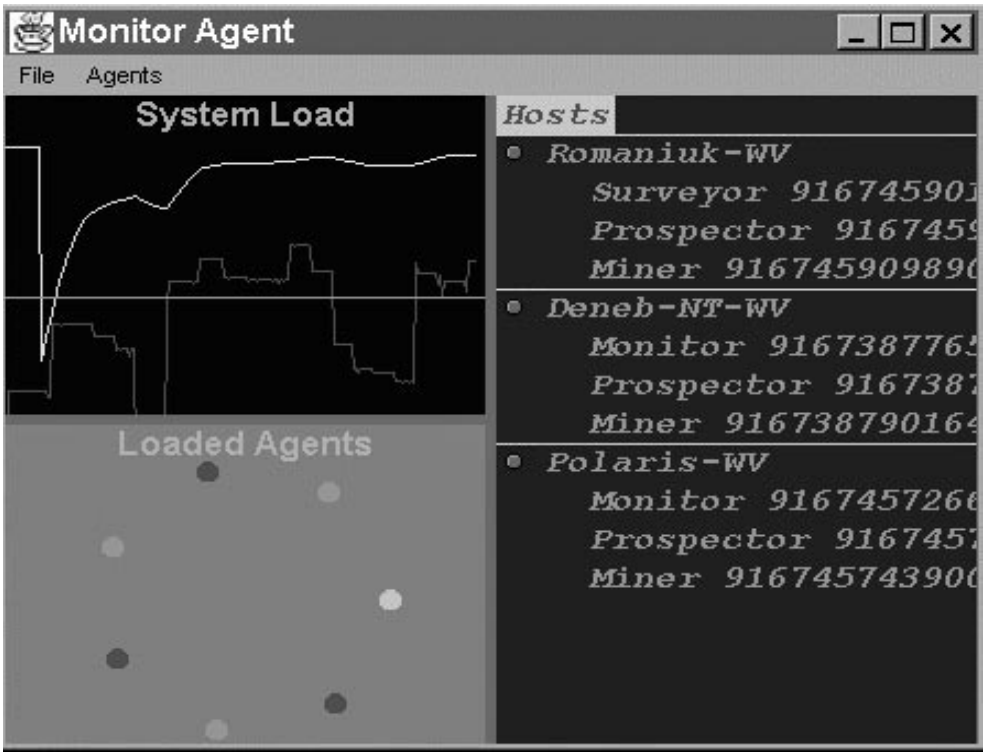

Figure 8. Example snapshot of Monitor Agent

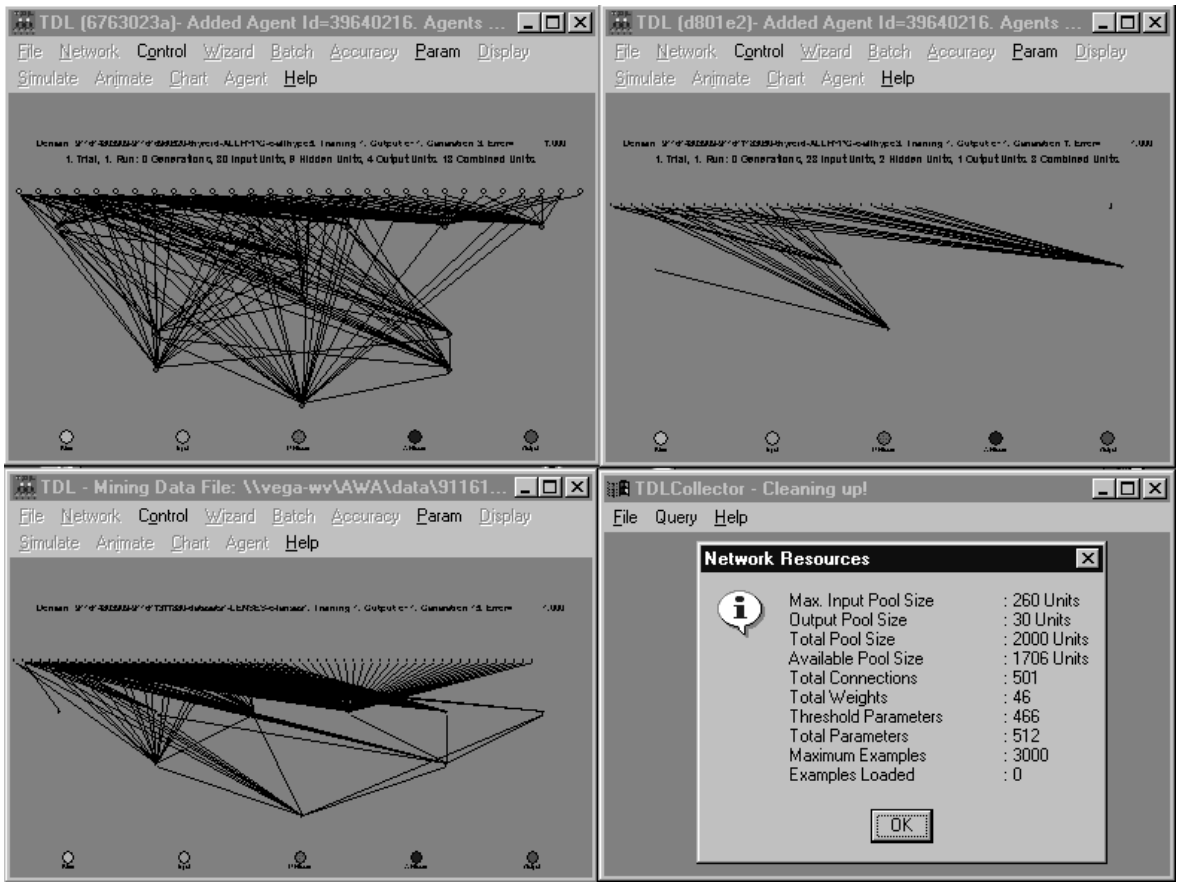

Figure 9. Example snapshot showing 3 TDLMiner Agents and one TDLCollector