KNOWLEDGE RETENTION WITH GENETIC ALGORITHMS BY MULTIPLE LEVELS OF REPRESENTATION

by

Yingjia Ding

Thesis submitted to the faculty of the

Virginia Polytechnic Institute and State University

for partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

APPROVED:

Dr. Jane T. Nutter, Chairman

Dr. Edward A. Fox

Dr. John A. N. Lee

May, 1991

Blacksburg, Virginia

KNOWLEDGE RETENTION WITH GENETIC ALGORITHMS BY MULTIPLE LEVELS OF REPRESENTATIONS

by

Yingjia Ding

Committee Chairman: Dr. Jane T. Nutter

Computer Science

(ABSTRACT)

Low-level representations have proven to be good at certain kinds of adaptive learning. High-level representations make effective use of existing knowledge and perform inference well. To promote using both forms of representation cooperatively rather than engaging in the perennial sectarian debate of supporting one paradigm at the expense of the other, this thesis presents a prototype system demonstrating knowledge retention using genetic algorithms and multiple levels of representation and learning. The prototype uses a mid-level of representation and transformations upward and downward for retaining domain-specific knowledge to bridge the gap between the high-level representation and learning and the genetic algorithm level. The thesis begins with an overview of the work, briefly introduces the principles of genetic algorithms, and states an illustrative domain. Then it reviews related work and two supportive systems. After that, it gives a general description of the prototype system's structure, three levels of representation, two transformations, and three levels of learning. Next, it describes methods of implementing the prototype system in some detail. Finally, it shows results with discussion, and points out conclusions and future work.

Acknowledgements

I would like to express my thanks to Dr. Jane T. Nutter, my supervisor, for her directive guidance during the research and writing of this thesis. Her advice and inspiration have been invaluable.

I would like to thank the other members of my committee, Dr. Edward A. Fox, for his helpful comments and discussion on this thesis, and Dr. John A. N. Lee, for his helpful suggestions on this thesis, and continuous care and encouragement since I came to United States.

I also would like to thank Dr. Mark Pavicic at North Dakota State University and IBM Application Systems Division at Rochester, Minnesota, for giving me the opportunity as a research associate during Oct. 1989 - Aug. 1990 to work on simulator PETSA for the Mayo/IBM PACS project.

Finally, I would like to thank my father Deyen Ding, my mother Xiouying Zhou, and my husband Xudong He, for their love, patience, and support.

Contents

List of Illustrations

Chapter 1. Introduction

1.1. Overview

This thesis investigates methods for combining genetic algorithm learning with traditional high-level inductive learning to retain domain-specific knowledge.

The power of an intelligent system lies in its knowledge. Much work has concentrated on acquiring such knowledge. However, almost all work to date uses only one level of representation -- either low-level representations such as binary strings, or high-level representations such as logical rules. Low-level representations have proven to be good at certain kinds of adaptive learning; high-level representations make effective use of existing knowledge and perform inference well [Carbonell, 1989]. But a debate has arisen between proponents of so-called subsymbolic representations and those of so-called symbolic AI. We contend that this debate is based on a fundamental misapprehension. Representation is *always* symbolic. The only real question is the level which the symbols represent. The answer, then, is not to choose one level, but to teach our systems to use them all cooperatively. This ability to combine multiple representation levels is particularly important for systems that will function as autonomous agents, and so must combine abilities such as vision and inference, a task that seems difficult if not impossible to do in any other way.

A somewhat simpler instance that calls for multilayered representation techniques can be found in a large class of design problems. Researchers in computing applications such as simulation and computer-aided design procedures often face the challenge of meeting time limits and performance/cost requirements, and develop the design by an iterative process: running a simulation program, analyzing the results, and modifying the

inputs to the simulator. To find an optimum design, they need optimization techniques and tools. Genetic algorithms turn out to be a good option (see section 1.2).

However, on the one hand, genetic algorithms do not explicitly take advantage of and retain domain-specific knowledge during their exploration of the parameter space for a solution. On the other hand, one major obstacle to a widened role for genetic algorithms has been its foundation on unstructured problem representations, typically binary strings. Although binary strings are theoretically tractable and form the basis of the theory underlying genetic algorithms, such representations contrast sharply with the high-level representations used in computing applications. When genetic algorithms are integrated with simulators to perform optimization, the binary string is not a natural representation of inputs to simulators. Also, if knowledge retained is represented by binary strings, it will be hard to incorporate it with existing knowledge and high-level learning programs.

In this thesis, a prototype system is developed to address how to bridge low-level genetic algorithm learning and traditional high-level inductive learning by using a mid-level of representation and learning, and transformations upward and downward for retaining domain-specific knowledge.

The rest of the thesis is organized as follows. The remainder of this chapter introduces the principles of genetic algorithms and the illustrative application domain on which the prototype system operates. Chapter 2 presents a brief review of previous related work and two underlying systems on which the implementation of our prototype is based. Chapter 3 provides a general architecture and implementation independent aspects of the prototype system. Chapter 4 describes implementation of the prototype system in some detail. Chapter 5 shows and discusses experimental results. Chapter 6 is devoted to conclusions and future work.

1.2. What are Generic Algorithms?

Genetic algorithms (GA's) are adaptive generate-and-test search procedures derived from principles of natural population genetics [DeJong, 1988] [Goldberg, 1989] [Grefenstette, 1988] [Holland, 1975]. A skeleton of a simple genetic algorithm is shown below.

```
procedure GA
begin
        t = 0;
        initialize P(t);
        evaluate P(t);
        while (not termination condition) do
        begin
                t = t + 1;
                select P(t) from P(t-1);
                recombine P(t);
                evaluate P(t);
        end
end.
```

During a given iteration t, called a *generation,* the genetic algorithm maintains a *population* $P(t)$ of *structures* :

$$P(t) = < x1(t), x2(t), \ldots , xN(t) >,$$

which are chosen from the domain of an objective function f. Each *structure* $xi(t)$, also called an *individual*, is simply a binary string of length L. Generally, each $xi(t)$ represents a vector of parameters for the function $f(xi(t))$, but the semantics associated with the vector are unknown to the GA. The initial population $P(0)$ is usually chosen at random. Each structure $xi(t)$ is *evaluated* by computing $f(xi(t))$. The term *trial* is often used for each such evaluation. This provides a measure of *fitness* of the evaluated structure for the given

problem. When each structure in the population has been evaluated, a new population of structures is formed in two steps.

First, structures in the current population are selected to reproduce on the basis of their relative fitness. That is, the *selection* algorithm chooses structures for replication by a stochastic procedure that ensures that the expected number of offspring associated with a given structure $xi(t)$ is $f(xi(t))/\mu(P(t))$, where $f(xi(t))$ is the observed performance of $xi(t)$ and $\mu(P(t))$ is the average performance of all structures in the population. That is, structures that perform well may be chosen several times for replication and structures that perform poorly may not be chosen at all. In the absence of any other mechanisms, this selective pressure would cause the best-performing structures in the initial population to occupy a larger and larger proportion of the population over time.

Next, the selected structures are recombined using *idealized genetic operators* to form a new set of structures for evaluation. One of the most important genetic operator is *crossover*, which combines the features of two parent structures to form two similar *offspring*. Crossover operates by swapping corresponding segments of a string representation of the parents. For example, let

$$x1 = 100 : 01010$$

$$x2 = 010 : 10100$$

and suppose that the crossover point has been chosen as indicated by the location of the colon. The resulting structures would be

$$y1 = 100 : 10100$$

$$y2 = 010 : 01010$$

Crossover serves two complementary search functions. First, it provides new points for further testing within the schemas already present in the population. In the above example, both $x1$ and $y1$ are representatives of the schema 100#####, where the # means *don't care*.

Thus, by evaluating *y1*, the GA gathers further information about this schema. Second, crossover introduces representatives of new schemas into the population. In the above example, *y2* is a representative of the schema #1001###, which is not represented by either parent. If this schema represents a high-performance area of the search space, the evaluation of *y2* will lead to further exploration in this part of the search space.

However, in generating new structures for testing, the crossover operator draws only on the information present in the structures of the current population. If specific information is missing, due to storage limitations or loss incurred during the selection process of a previous iteration, or because of gaps in the initial population, then crossover may be unable to produce new structures that contain it. A *mutation* operator that arbitrarily alters one or more components of a selected structure provides the means for introducing new information into the population. Its presence ensures that all points in the search space can be reached.

Termination may be triggered by finding an acceptable approximate solution to $f(x)$, by fixing the total number of evaluations, or by some other application dependent criterion.

The basic concepts of GA's were developed by Holland and his students. GA's have been applied to various problems, including numerical function optimization, adaptive control system design, and artificial intelligence task domains [Goldberg, 1989]. Theoretical considerations concerning the allocation of trials to schemas [Holland, 1975] show that genetic techniques provide a highly efficient heuristic for information gathering in complex search spaces. A number of experimental studies [Grefenstette et al., 1991] have shown that GA's exhibit impressive efficiency in practice. While classical gradient search techniques are more efficient for problems that satisfy tight constraints (e.g., continuity, low dimensionality, etc.), GA's consistently outperform both gradient techniques and various forms of random search on more difficult (and more common)

problems, such as optimizations involving discontinuous, noisy, and high dimensional objective functions.

1.3. Illustrative Domain

The prototype of knowledge retention described in this thesis operates on a simplified domain of multicomputer network systems for medical image archival and retrieval [Persons et al., 1990]. Such networks basically have five types of components: token rings, bridges, servers, workstations, and controllers. Hereafter, we called these systems *task systems*. A simple configuration of a task system is shown in Figure 1 (a). The functionality of each component and the parameters which identify a component are briefly described as follows.

- A *token ring* connects servers, workstations, or controllers for communication among these components. Parameters are: (1) ring name; and (2) ring type, e.g., 4Mbps IEEE802.5 (default).
- A *bridge* interconnects two token ring LANs. Parameters are: (1) bridge name; (2) bridge type, e.g., FDDI split bridge (default); and (3) two ring names, which are at each end of the bridge.
- A *server* is a dedicated system for image archival and retrieval, without any capabilities for interactive users. Parameters are: (1) server name; (2) server type, e.g., System/36 optical storage system (default); and (3) ring name, on which the server is attached.
- A *workstation* has capabilities for interactive users. A workload is assigned to each workstation in the network. A workstation user can request a controller/server to archive images which are generated from modalities (e.g., CT units), or can request

a controller/server to retrieve images for clinical review of images. Parameters are: (1) workstation name; (2) workstation type, e.g., IAU (default); and (3) ring name, on which the workstation is attached.

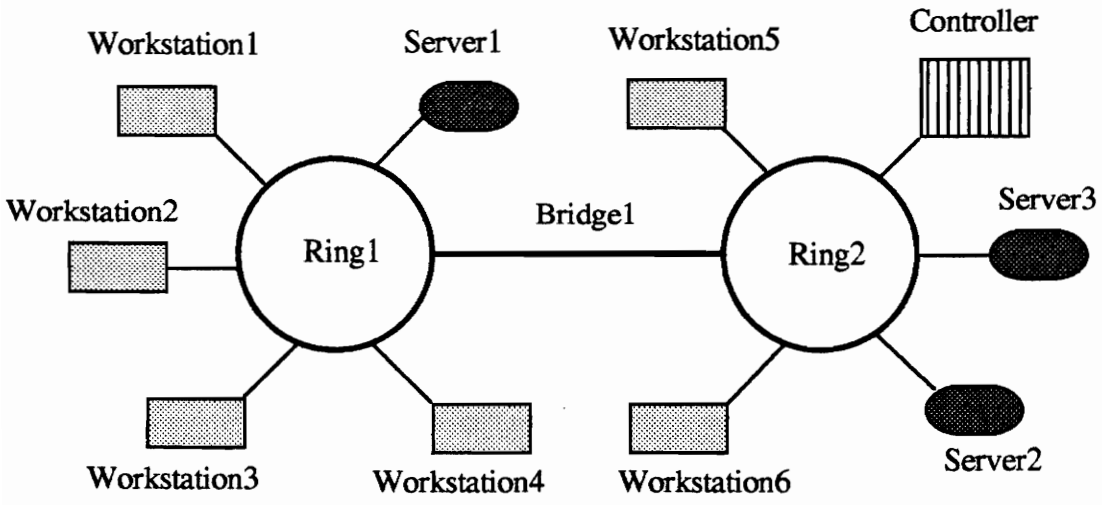
- A *controller* orchestrates the network flow. Only one controller is allowed in the network system. It provides overall system management, communication control, and database management functions for image indexes. Parameters are: (1) controller name; (2) controller type, e.g., AS/400 (default); and (3) ring name, on which the controller is attached.

The information about a system configuration is contained in the values of the above parameters. A daily workload to each workstation can be represented by a 4-tuple:

<start-time, duration, #-of-requests, amount-of-images>,

which means that the workstation starts to work at *start-time*, continues work for *duration* hours, and sends out *#-of-requests* requests to retrieve or archive a total of *amount-of-images* images in megabytes. Typical workload constraints are: (1) *start-time* cannot be earlier than 7 am or later than 2 pm; (2) *duration* ranges from 1 to 8 hours; (3) *#-of-requests* ranges from 4 to 32; and (4) *amount-of-images* ranges from 10 to 150 megabytes. For example, workloads of six workstations in the configuration Figure 1 (a) is shown in Figure 1 (b).

One optimization problem in this domain is as follows: given a task system configuration and constraints, what is a near-maximum workload for each workstation without causing bottlenecks in the system (i.e., determine if the worst response time and average response time are within a certain range)? The focus of this thesis is not on how to solve this optimization problem, but on how to retain knowledge by multiple levels of representation and learning during and after a search for a near-optimal solution.



(a)

Workload / Workstation	Start-time	Duration (hr)	#-of-requests	Amount-of-images (mb)
W1	7	8	32	150
W2	8	7	28	120
W3	9	6	24	100
W4	9	6	24	100
W5	8	7	28	120
W6	7	8	32	150

(b)

Figure 1. A simple configuration of a task system and its workload

Chapter 2. Background

2.1. Previous Work

This section gives a brief review of the previous related work which has partially inspired the current work [Powell et al., 1989] [Zhou, 1990] [Grefenstette el al., 1990] or been used in it [Mitchell, 1982].

Powell et al. [1989] developed an interleaved expert system and genetic algorithm model to take advantage of domain-specific knowledge to improve efficiency. The model starts from a single design point, uses a rule based expert system for maximum efficiency gain, applies specialized control methods to augment the expert system, and finally uses genetic algorithms to supplement knowledge, to avoid constraints, and to escape local optima. But this hybrid model does not address knowledge retention.

Zhou [1990] developed a rule-based, cumulative learning system called CSM (Classifier System with Memory), tested in a robot navigation domain. Classifier systems are a special class of rule-based systems [Holland, 1986]. Like conventional production systems, knowledge is stored in rules in the If-Then form. Unlike conventional production systems, each rule represents its performance by a real number, called strength. The rules interact with each other through a message list and are stored in a temporary knowledge base, i.e., a population. A set of detectors relays external information to the system in the form of messages. The detector messages may trigger eligible rules that in turn generate new messages. A set of effectors takes the message generated by the rules and performs corresponding actions.

Genetic algorithms act as the main learning algorithm in these classifier systems. The classifier system model provides a promising approach toward the development of general purpose learning systems [Goldberg, 1989]. However, the problem of

forgetfulness in current classifier systems jeopardizes their ability to improve their performance incrementally over an extended period of time. In response to this problem, CSM preserves problem solving expertise and tailors it to fit new situations. CSM is concerned with the transfer of learned knowledge within a domain, and the improvements of its learning performance in the long run. But since in CSM or other classifier systems a population consists of If-Then rules, the methods employed by CSM cannot be directly applied to genetic algorithms in optimization, where each individual in a population is a candidate solution for a problem.

Grefenstette et al. [1990] designed a system for learning control strategies with genetic algorithms, called SAMUEL (Strategy Acquisition Method Using Empirical Learning). In a major departure from previous genetic learning systems, SAMUEL learns rules expressed in a restricted high level rule language. But corresponding high level genetic operators for that language have been adapted from basic genetic algorithms, and the methods are only suitable for classifier-like systems.

Mitchell [1982] presents one of the widely studied methods for symbolic learning, called *candidate elimination algorithm*, which induces a general concept description from instances. Given a set of training data and a language in which the desired concept must be expressed (which defines the space of possible generalizations that concept learning will search), Mitchell defines a *version space* to be the set of all concept descriptions within the given language which are consistent with the training instances. Mitchell noted that the generality of concepts imposes a partial order that allows efficient representation of the version space by the boundary sets S and G, representing the most specific and most general concept definitions in the space. The version space contains all concepts at least as general as some element in S and at least specific as some element in G. Given a new instance, some of the concept definitions in the version space for past data may not classify

it correctly. The candidate elimination algorithm manipulates the boundary-set representation of a version space to create boundary sets that represent a new version space consistent with all previous instances plus the new one. The unknown concept is determined when the version space has only one element, which in the boundary-set representation is when the *S*- and *G*-sets have the same single element. But this method has a precondition, which is that the training instances are supplied by a teacher.

2.2. SNePS and OOGA

Two underlying systems, SNePS and OOGA, which form an environment for our prototype system, are briefly described here.

SNePS (the Semantic Network Processing System) [Shapiro and Group, 1989] [Shapiro and Martins, 1989] is a system for building, using, and retrieving from propositional semantic networks. It has been implemented in Common LISP.

A semantic network is a labeled direct graph in which nodes represent concepts, arc labels represent binary relations, and an arc labeled *R* going from node *n* to node *m* represents that the concept represented by *n*, bears the relation represented by *R*, to the concept represented by *m*. SNePS is called a *propositional* semantic network because propositions themselves are treated explicitly as concepts, i.e., every proposition represented in the network is represented by a node, not by an arc. Whenever information is added to the network, it is added in the form of a node with arcs emanating from it to other nodes. Each concept represented in the network is represented by a unique node.

SNePS has four basic sub-systems and a user language: the core of SNePS, SNIP, SNeBR, SNaLPS, and SNePSUL. The core of SNePS is a system for building nodes in the network, retrieving nodes that have a certain pattern of connectivity to other nodes, and performing certain housekeeping tasks, such as dumping a network to a file or loading a

network from a file. SNIP, the SNePS Inference Package, interprets certain nodes as representing reasoning rules, called *deduction rules*. SNIP supports a variety of specially designed propositional connectives and quantifiers, and performs a kind forward/backward *bi-directional inference*. SNeBR, the SNePS Belief Revision system, recognizes when a contradiction exists in the network, and interacts with the user whenever it detects that the user is operating in a contradictory belief space. SNaLPS, the SNePS Natural Language Processing System, consists of a morphological analyzer/synthesizer, and a Generalized Augmented Transition Network (GATN) grammar interpreter/compiler. Using these facilities, we can write natural language interfaces for SNePS. SNePSUL, the SNePS user language, is a Lispish language, which is usually entered at the top-level SNePSUL read-eval-print loop, but can also be called from LISP code or from GATN arcs.

OOGA (the Object-Oriented Genetic Algorithm) [Grefenstette et al., 1991] [Davis, 1991] is an object-oriented genetic algorithm system. It is implemented in Common LISP with the CLOS extension [Keene, 1989], which makes it easier to allow genetic algorithms work together with SNePS. OOGA has been designed for use in genetic algorithm experimentation. Every principal component of a genetic algorithm is an object in OOGA. Replacement of one component by another merely needs to create an object of the desired type and to place it in the appropriate slot. The highly modular OOGA architecture makes it easy to define and use a variety of genetic algorithms techniques for our own purposes.

Chapter 3. General Description of Prototype

3.1. Structure of the Prototype

A prototype system is designed to investigate methods of combining low-level genetic algorithm learning and traditional high-level inductive learning for retaining domain-specific knowledge. The prototype currently operates on the application domain given in section 1.3. The structure of the prototype consists of ten modules and a knowledge base shown in Figure 2 (a) and (b), where *LM-Transformation* means the transformation from a low-level representation of candidate solutions to a mid-level representation of the candidate solutions, and *MH-Transformation* means the transformation from a mid-level representation of accumulated information to a high-level representation of rule instances.

The main algorithm of the prototype is outlined as follows.

begin

1. *In-User-Interface Module.* Read a task system configuration and constraints in a limited subset of English, and build background knowledge into the knowledge base.
2. *Initialization Module.* Give a GA parameter setting (e.g., population size, individual string length, crossover rate, and mutation rate), and initialize the first population randomly (each individual in the population is a vector of workloads for workstations in the task system).
3. **for** each individual in the current population, **begin**

    3.a. *LM-Transformation Module.* Convert a low-level representation of the individual to a mid-level representation of the individual, assisted by the

background knowledge in the knowledge base, and temporarily add this mid-level representation into the knowledge base.

3.b. *Simulation Module.* Use the mid-level representation of the individual, i.e., the configuration and workload of the task system, as an input, simulate the performance of the task system, and output an evaluation as a fitness measureof the individual.

3.c. *Accumulation Module.* Check if the fitness is the same, better, or worse than the best or the worst fitness of individuals among the previous generations. If not, erase the mid-level representation of this individual from the knowledge base. Otherwise, add the fitness with the mid-level representation of the individual into the knowledge base.

end

4. *Termination Module.* Check the termination condition. **If** the condition is not satisfied, **then** invoke Genetic-Operation Module (step 5), **else** invoke MH-Transformation Module (step 6).
5. *Genetic-Operation Module.* Do genetic reproduction and recombination on the current population, produce a new generation, then return to step 3.
6. *MH-Transformation Module.* Analyze good and bad individuals saved in the knowledge base, and convert them from a mid-level representation to a high-level representation of positive and negative rule instances.
7. *Generalization Module.* Generalize the positive and negative high-level rule instances, and produce a rule.
8. *Out-User-Interface Module.* Generate an English text to express the solution and rule.

end

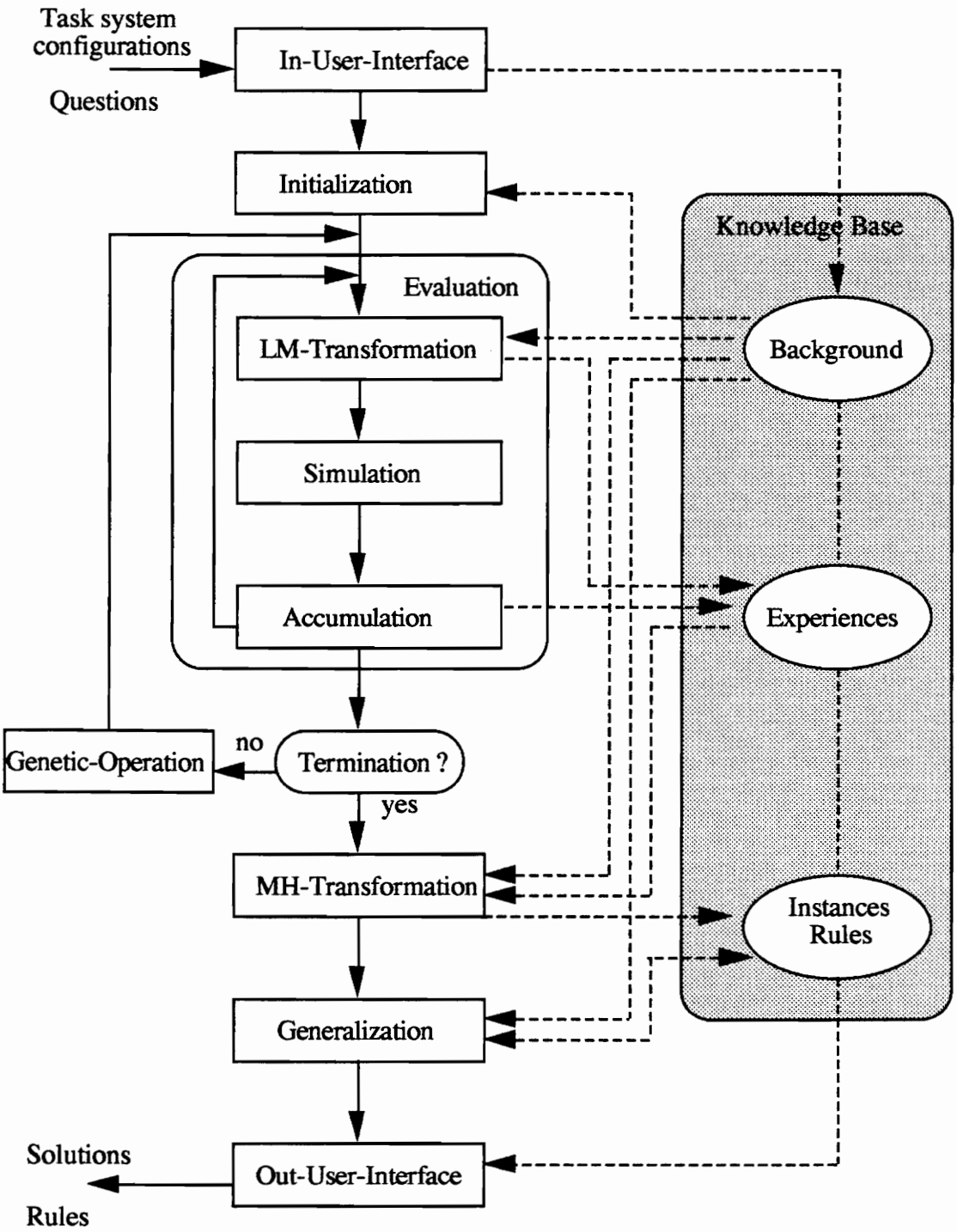


Figure 2 (a). A structure of the prototype: algorithm

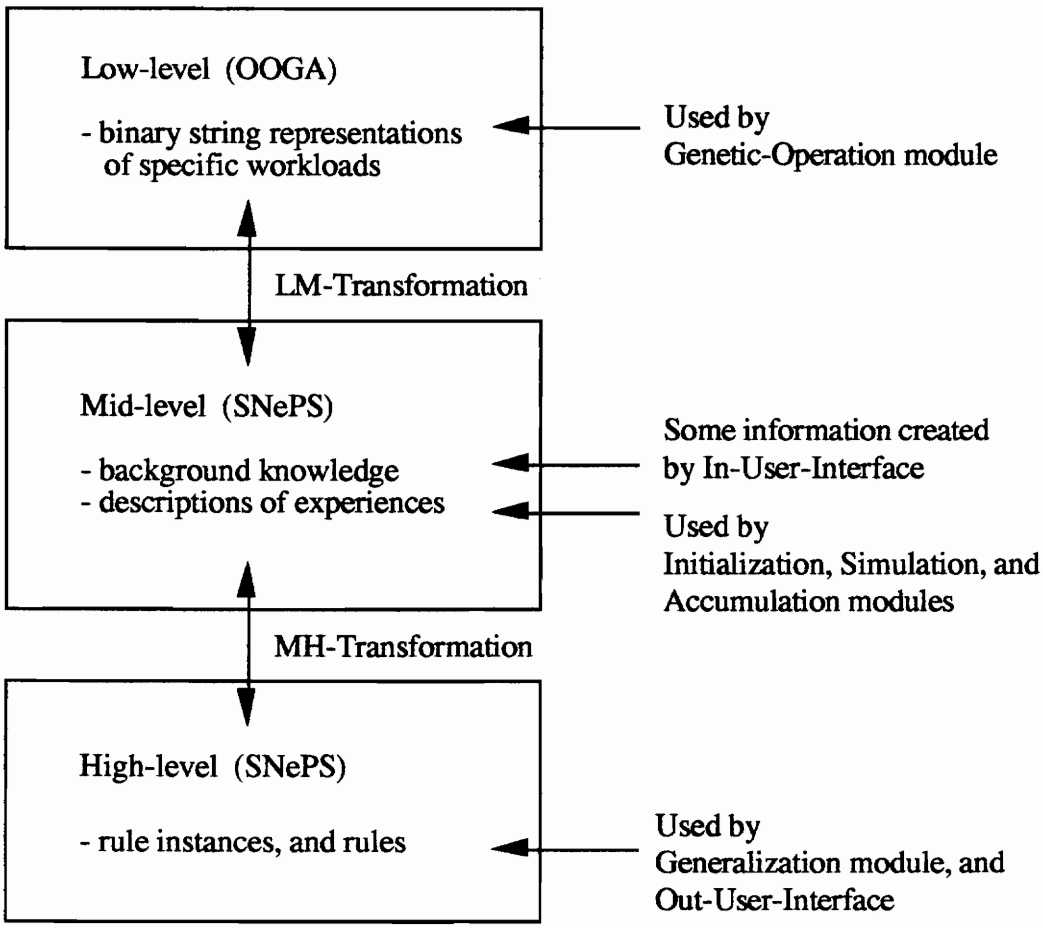


Figure 2 (b). A structure of the prototype: knowledge base

In this prototype, knowledge retention is achieved by using three levels of representation, two transformations, and three levels of learning, which are described in the following two sections.

3.2. Three Levels of Representation and Transformations

The three levels of representation used in this prototype are a low-level representation for candidate solutions; a mid-level representation for the candidate solutions, background knowledge, and accumulated information; and a high-level representation for rule instances.

The low-level representation is in the form of strings over the binary alphabet

$$V = \{0, 1\},$$

which code parameters that control a task system's behavior. These binary strings, i.e., individuals, are accessed by genetic algorithms and by the LM-Transformation module, and are treated as representations of candidate solutions for an optimization problem. For the problem given in section 1.3, a key set of parameters is identified, which is a set of 4-tuples. Each element in the 4-tuple is a parameter. The set size is the number of workstations in a task system; each element in each 4-tuple is a workload parameter for a workstation. The length of the strings depends on the ranges of these parameters and the set size. For example, a coded binary string for a workload on the task system shown in Figure 1 (a) can be:

"011011011111 001011011101 011011011111

001011011101 011011011111 001011011101".

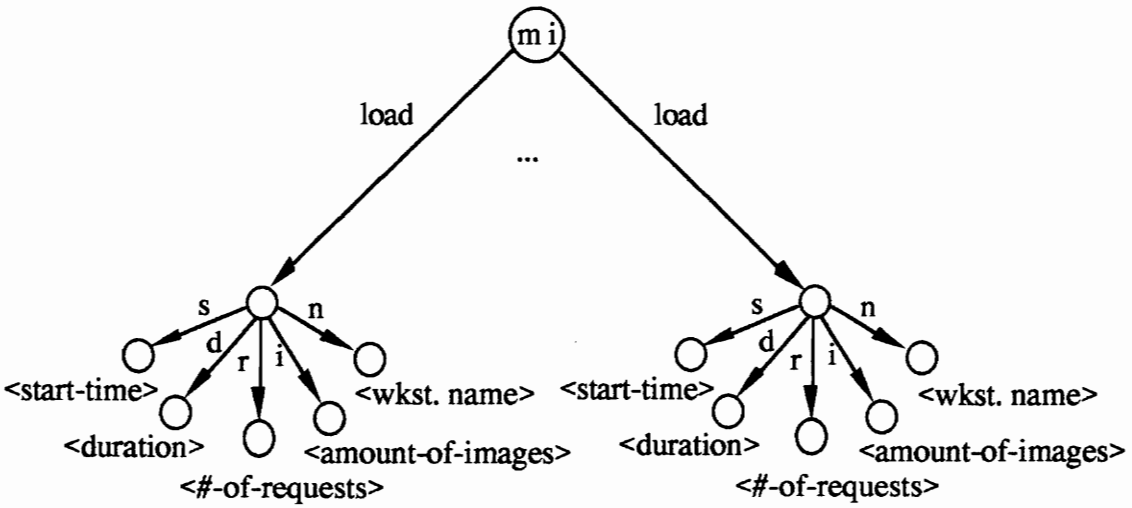
The string length is 12 * 6 = 72, where 12 is the substring length for a 4-tuple (see section 4.2) and 6 is the number of workstations in the task system configuration.

The mid-level representation uses SNePS semantic network frames to provide a simple domain model. Depending on what is represented, the formats of the semantic networks can be different. On the illustrative domain, for representing an individual and an accumulated experience, the formats of semantic networks are defined in Figure 3 (a) and 3 (b); for representing background knowledge, such as a task system configuration, an interpretation of a 4-tuple, and generalization hierarchies, the formats of semantic networks are defined in Figure 4 (a), 4 (b), and 4 (c) respectively. These mid-level representations are accessed by almost all modules except the Genetic-Operation and Termination modules in the prototype system.

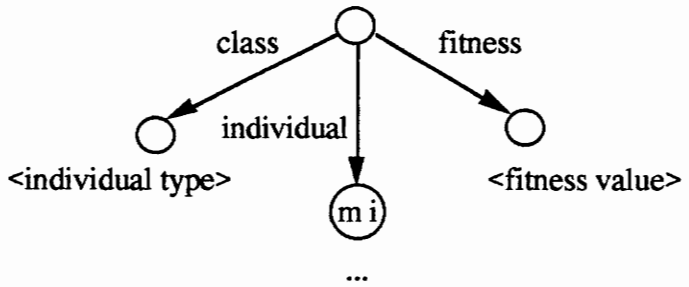
The high-level representation is also in the form of SNePS subnetworks. But instead of providing direct problem modeling, the high level representation units represent vectors of attribute-value pairs for a rule instance in the propositional calculus. The format is defined in Figure 5, where the relation between each arc *arg* is logical *and*. This representation is accessed by Generalization and Out-User-Interface modules.

The above three levels of representation are bridged by two transformations -- the LM-transformation and the MH-transformation. The objective of the LM-transformation is to map the low-level representation of an individual onto its corresponding mid-level representation, letting the genetic exploration stay at the low-level while simulation and accumulation take place at the mid-level. The strategy of LM-transformation is relatively simple. It uses background knowledge (see Figure 4 (a) and 4 (b)) to understand what parameters are coded in a binary string and interpret it in the semantic networks.

The objective of the MH-transformation is to map the mid-level representation of accumulated information to high-level rule instances, letting the experiences of genetic exploration be accumulated and analyzed more accurately at the mid-level while traditional inductive learning takes place at the high-level. The MH-transformation applies domain-

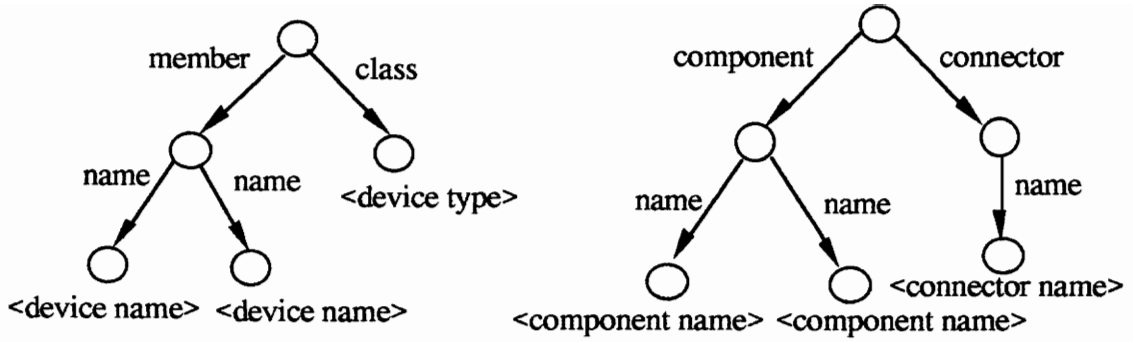


(a)

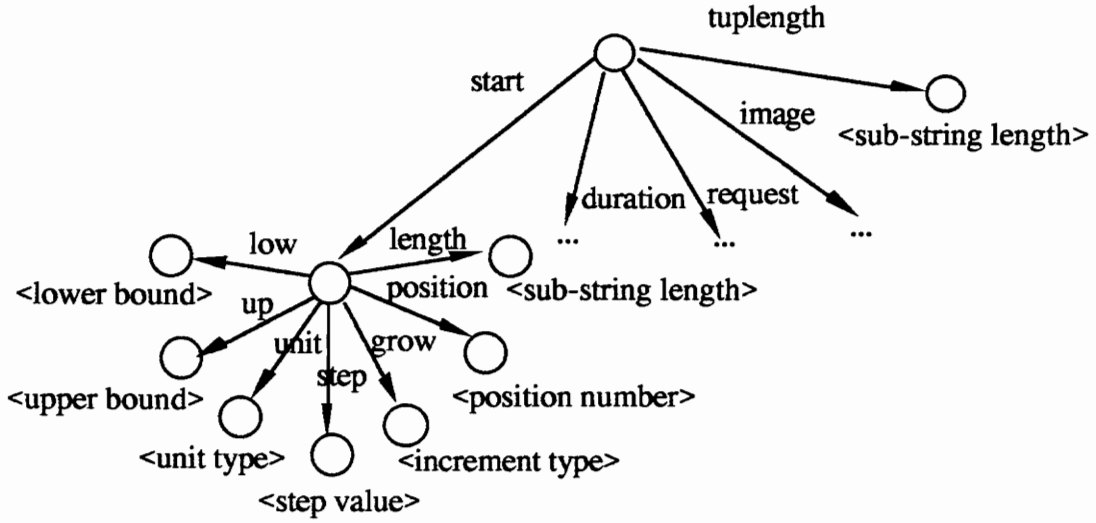


(b)

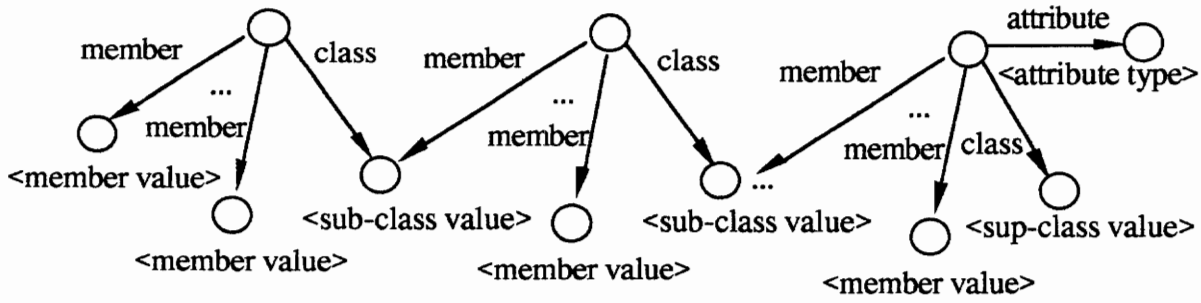
Figure 3. Mid-level representation formats for individuals and experiences



(a)



(b)



(c)

Figure 4. Mid-level representation formats for background knowledge

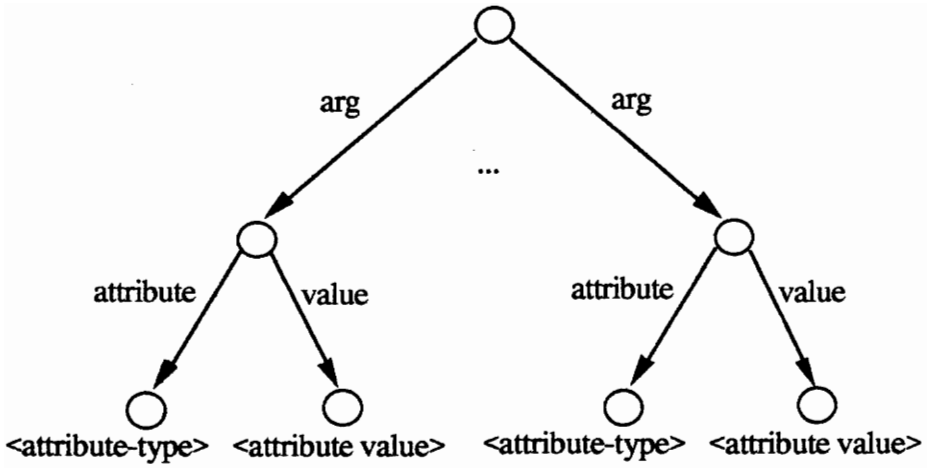


Figure 5. High-level representation format for rule instances

dependent analysis routines to the mid-level accumulated information, and abstracts out high-level rule instances, assisted by generalization hierarchies and a task system configuration (see Figure 4 (a) and (c)). This MH-transformation is also a kind of learning by abstraction, which is described in the next section.

3.3. Three Levels of Learning

Corresponding to the three levels of representation, there are three levels of learning in the prototype -- low-level genetic adaptive search, mid-level accumulation and abstraction, and high-level inductive generalization.

A simple and intuitive approach to effecting behavioral changes in a performance system is to identify a key set of parameters that controls the system's behavior, and to develop a strategy for changing those parameters' value to improve performance. In our prototype, GA's provide this strategy, viewing the parameters as genes and the genetic material of individuals as a fixed-length string of genes, one for each parameter. The crossover operator then generates new parameter combinations from existing good combinations in the current population, and mutation provides new parameter values. It is easy at first glance to discard this approach as trivial and not at all representative of what is meant by "learning". But if we view the adjustable weights and thresholds as parameters of a structurally-fixed neural network, then much of the research on neural network learning also falls into this category.

Why do GA's operate on low-level representations? [Holland, 1975] provides an analysis of GA's which suggests that they are most effective when each gene takes on a small number of values, and that binary genes are in some sense optimal for GA-style adaptive search. For example, rather than representing a 24-parameter problem (six 4-tuples for six workstations in Figure 1 (a)) internally as strings of 24 genes (with each gene

taking on many values), a binary string representation can be used to represent parameters as a *group* of binary-valued genes. Although the two spaces are equivalent in that both represent the same parameter space, GA's perform significantly better on the binary representation. This effect occurs because, in addition to mutation, crossover now generates new parameter values each time it combines a substring of a parameter's bits from one parent with those of another.

Although GA's learn during the exploration of the parameter space for the current problem, they do not retain their knowledge for later use, to avoid having to be rediscovered, for instance in a different design with the same simulator. If domain-specific knowledge from past experience is retained, it can help in automatically seeding the initial population [Davis, 1987] to reduce the number of runs of the simulation code.

There exist many traditional high-level inductive learning algorithms [Carbonell, 1989]. Unfortunately, they cannot be used directly with GA's to retain domain-specific knowledge, because they work at a high-level of representation and because they require an external teacher to supply correct training instances to them. In our prototype, accumulation and MH-transformation provide a mid-level learning device which bridges the gap between low-level genetic learning and high-level inductive learning.

Accumulation retains GA exploration experiences by saving the best and the worst individuals along with their fitness ratings. These individuals basically stay at the parameter level even though they are represented by the mid-level representation. The MH-transformation then analyzes these individuals, using background knowledge, and extracts overall information as rule instances for high-level inductive learning.

The candidate elimination algorithm, employed by the prototype as a high-level inductive learning strategy, finds a rule in the rule space that best covers all positive and no

negative instances, when given a representation language (i.e., generalization hierarchies in background knowledge) for rules, which implicitly defines the rule space.

As a summary of this section, three levels of learning in the prototype work cooperatively to retain domain-specific knowledge as follows: GA's change parameters at the low-level; accumulation collects the best and worst individuals at the mid-level in parallel with GA's search; MH-transformation abstracts rule instances from the mid-level to the high-level; and finally the candidate elimination algorithm induces a rule at the high-level.

Chapter 4. Implementation of the Prototype

4.1. Driver and Interface

The prototype system is implemented in Allegro Common LISP and SNePSUL on a DECstation 5000 model 200 running under Ultrix. This section and the following sections in this chapter describe implementation methods in some detail.

The In-User-Interface and Out-User-Interface modules are implemented as a simple natural language understanding and generating subsystem using a GATN in SNePS. This GATN includes a driver for controlling the execution sequence of the rest of the modules in the prototype, making use of the GATN's ability to evaluate arbitrary LISP forms at GATN grammar load time and to use them as actions on GATN arcs for side effect.

A skeleton of the driver and user interface is shown in Figure 6. To start the prototype system, the user first loads the GATN grammar file into SNePS. The GATN grammar file consists of a loading part and a parsing/generating part. The loading part loads a lexicon, definitions of module functions in Common LISP, and background knowledge representation in SNePSUL. The parsing/generating part uses the lexicon when parsing and generating, and calls the module functions when taking actions.

After loading the GATN grammar file, the user describes task system configurations and asks questions in English sentences. These sentences invoke the GATN to parse them, to build some knowledge into the knowledge base or call some modules to obtain solutions or rules, and finally to generate responses in English back to the user.

The syntax of English sentences for the user interface on the illustrative domain is as follows.

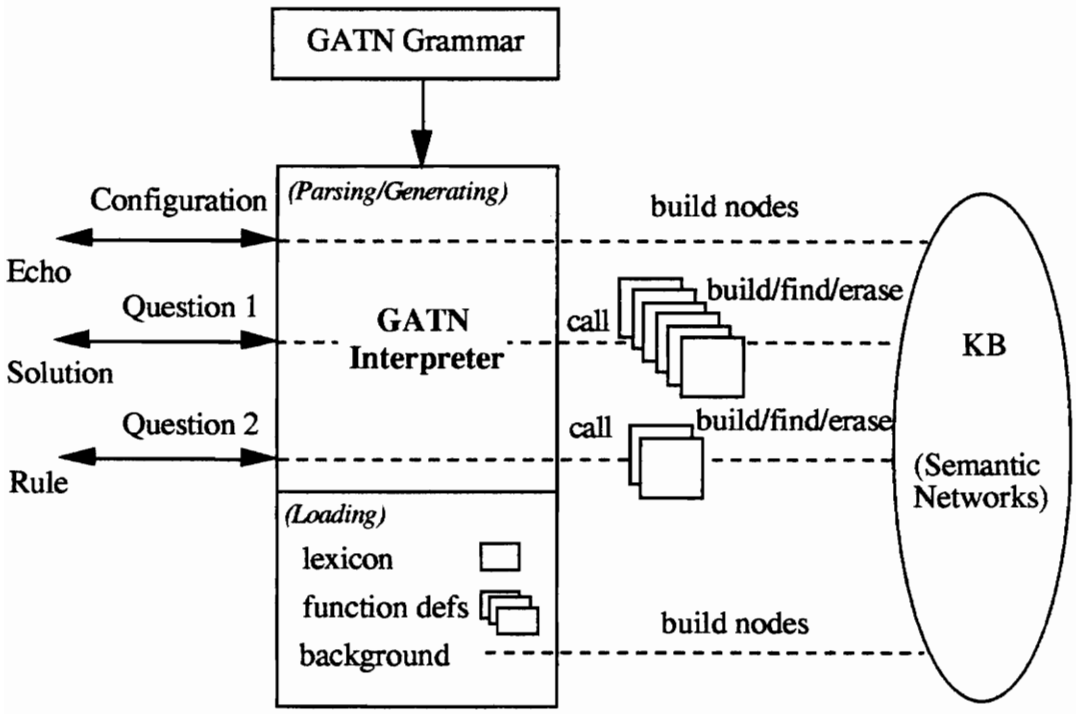


Figure 6. A skeleton of the driver and user interface

S --> *there* verb quant noun *:* NL.
| NL verb NL.
| NL verb *on* NL.
| wh verb [det] [adj] noun *?*
NL --> name {, name}

Note:
| indicates choices;
[ ] indicates options;
{ } indicates zero or more repetitions; and
":", ".", "*?*", ",", "*there*", and "*on*" are terminals.

The lexicon contains the following information:

Word	Category	Type	Num	Root
a	det			
are	verb		plur	be
be	verb			
connect	verb			
general	adj			
is	verb		sing	be
maximum	adj			
rule	noun			
the	det			
there	adv			
what	wh			
workload	noun			
on	prep	ondesc		
bridge	noun	device		
ring	noun	device		
server	noun	device		
controller	noun	device		
workstation	noun	device		
1	quant			
2	quant			
3	quant			
...				
R1	noun	dname		
B1	noun	dname		
C1	noun	dname		
S1	noun	dname		
W1	noun	dname		
...				

:	punc	colon
,	punc	comma
.	punc	final
?	punc	final

A sequence of English sentences for the configuration Figure 1, for example, can be the following:

(1) There are 2 rings: R1, R2.

(2) There is 1 bridge: B1.

(3) There is 1 controller: C1.

(4) There are 3 servers: S1, S2, S3.

(5) There are 6 workstations: W1, W2, W3, W4, W5, W6.

(6) B1 connects R1, R2.

(7) S1, W1, W2, W3, W4 are on R1.

(8) C1, S2, S3, W5, W6 are on R2.

(9) What is a maximum workload?

(10) What is a general rule?

Sentences (1)-(8) state the configuration, which will be converted into the mid-level representation in semantic networks (shown in Figure 7 (a)-(h)) as a consequence of parsing. Although SNePS enforces that each concept is represented by a unique node, we draw some base nodes such as B1 and R1 twice to simplify Figure 7.

Sentences (9)-(10) serve both as questions and as commands. Sentence (9) translates into a sequence of function calls to module *Initialization, LM-Transformation, Simulation, Accumulation, Genetic-Operation and Termination* after its GATN parsing. Sentence (10) translates into a sequence of function calls to module *MH-Transformation and Generalization* after its GATN parsing. A solution and a rule in English will come out from the interface following those function calls and GATN generating.

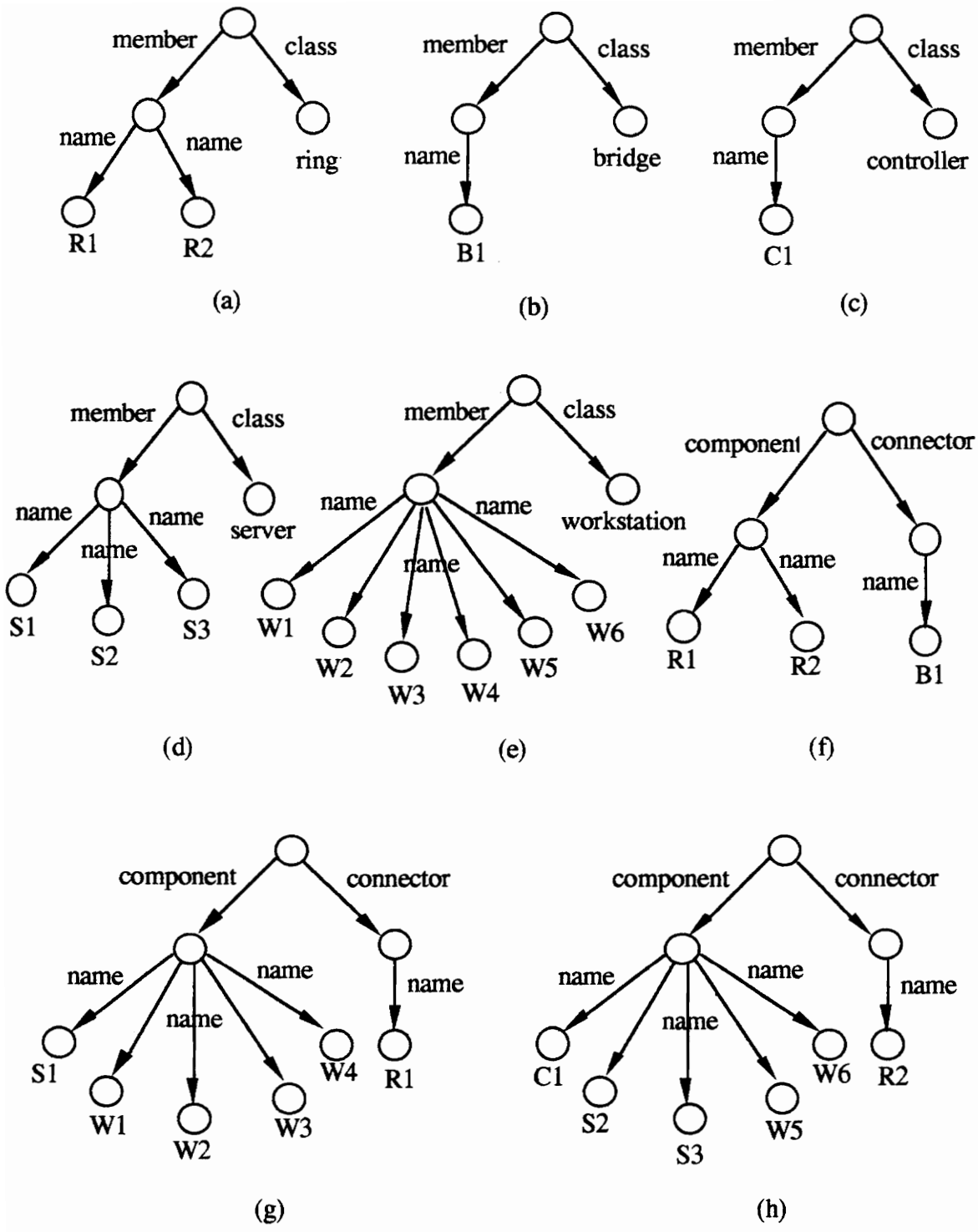


Figure 7. Representation of a configuration in semantic networks

4.2. Initialization, Genetic-Operation and Termination

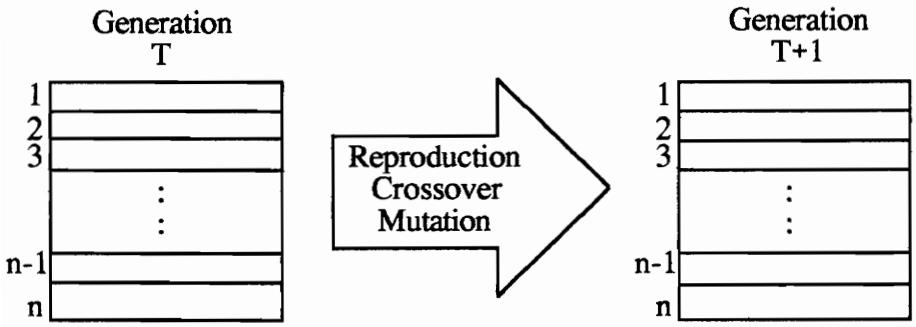
Initialization, Genetic-Operation, and *Termination* modules are most related to the domain independent part of a genetic algorithm. A genetic algorithm processes population strings. The primary data structure for the genetic algorithm is a string population, which is constructed as an array of individuals. Each individual contains a binary string and a fitness value. A schematic of a population is shown in Figure 8 (a).

Before creating an initial population randomly, the initialization module has to set the following GA parameters: (1) string-length, (2) population-size, (3) crossover-rate, and (4) mutation-rate. The population-size affects both the ultimate performance and the efficiency of a GA. The crossover-rate and mutation-rate control the frequencies with which crossover and mutation are applied [Grefenstette, 1986]. These (2)-(4) GA parameters can directly be set by using the standard GA parameters, e.g., population-size = 50, crossover-rate = 0.6, and mutation-rate = 0.001. However, to decide the string-length for each individual, and to save the correspondence between a binary string and a configuration for LM-transformation, the initialization module needs to access the knowledge base.

The knowledge base contains the following background knowledge (Figure 9) for interpreting the 4-tuple of a workload on each workstation (see section 1.3): start time ranges from 7 am to 14 pm in increments of 1 hour, takes 3 bits in a binary string, and its position in the tuple is 1; duration ranges from 1 hour to 8 hours in increments of 1 hour, takes 3 bits in a binary string, and its position in the tuple is 2; the number of requests ranges from 4 to 32 in increments of 4 numbers, takes 3 bits in a binary string, and its position in the tuple is 3; the amount of images varies from 10 to 150 MegaBytes in increments of 20 Megabytes, takes 3 bits in a binary string, and its position in the tuple is 4; and the total length of a binary string for a 4-tuple is 12.

Individual Number	Individuals	
	Binary String	Fitness
1	011011011011110011001100	70
2	100011011011011111100011	140
⋮	⋮	⋮
n	010110011111011111000010	85

(a)



(b)

Figure 8. A GA string population and nonoverlapping populations

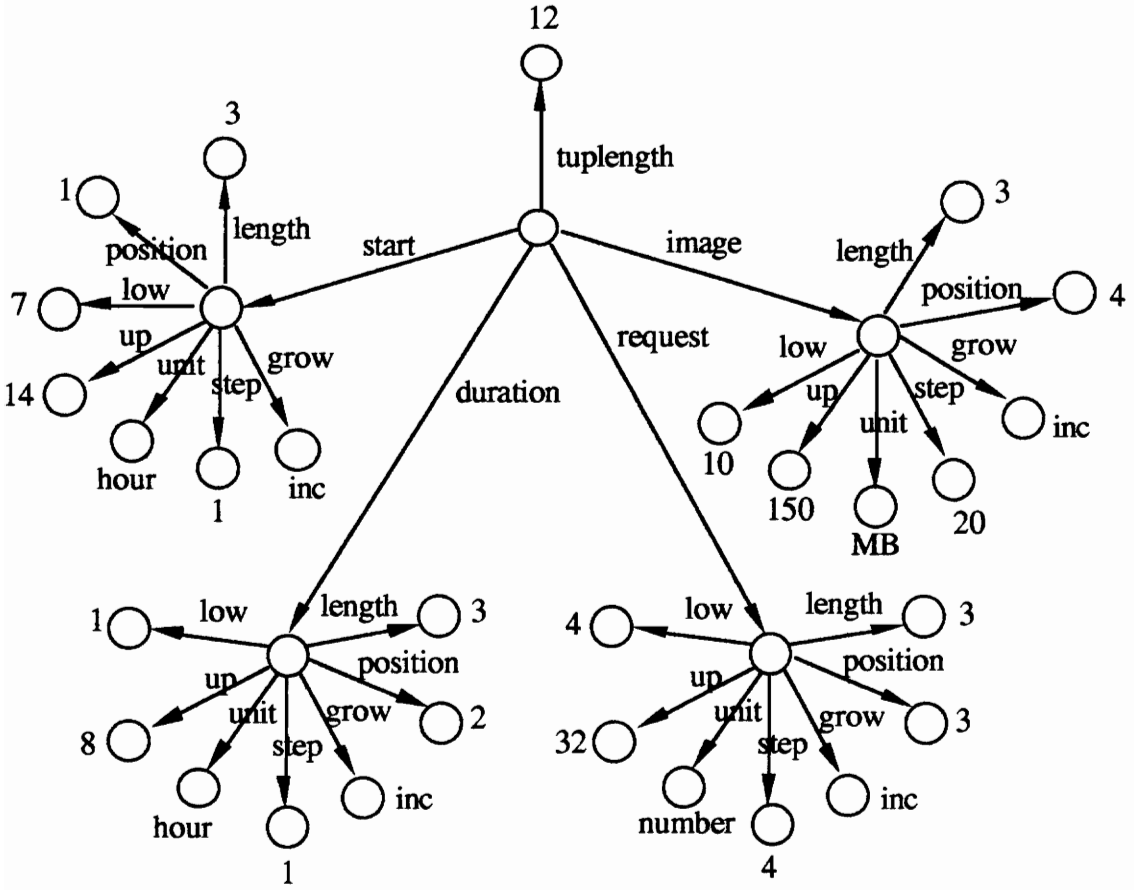


Figure 9. Background knowledge for interpretation

The initialization module will find a set of base nodes representing workstations by using the SNePSUL command *(findbase (compose name- member- class) "workstation")*. The order of the nodes in the node set will be mapped to the segment positions of coded 4-tuples in an individual string. The order will be saved in a global variable **worset** in the package SNePSUL. Then the string-length will be set to the value of (number-of-workstations * 4-tuple-length).

The genetic-operation module consists of three major procedures which correspond to the three genetic operators (reproduction, crossover, and mutation). These operators will be applied to an entire population at each generation as shown in Figure 8 (b). Two nonoverlapping populations are utilized to simplify the birth of offspring and the replacement of parents. In this genetic-operation module, three techniques are used: elitism, linear normalization, and two-point crossover.

The best individual of the population may fail to produce offspring in the next generation because of the stochastic selection, crossover, and mutation. The *elitist* strategy fixes this potential source of loss by copying the best member of each generation into the succeeding generation.

Linear normalization is a solution for stagnating search. For example, after several generations, the current population might contain only structures x for which

$$115 < f(x) < 120.$$

At this point, no structure in the population has a fitness which deviates much from the average. This reduces the selection pressure toward the better structures. The linear normalization solves this problem as follows: order the individuals by decreasing evaluation, and create fitness values that begin with a constant value and decrease linearly. The constant value and the rate of decrement are parameters of this technique, which are 100 and 2 respectively in our prototype system.

One-point crossover was inspired by biological processes, but its algorithmic counterpart has some drawbacks. One of the most important is that one-point crossover cannot combine certain combinations of features encoded on individuals. For example,

Individual-1	11011001011011
Individual-2	00010110111100

The underlined bits comprise the schemata. One-point crossover cannot cause these schemata to be combined on a single individual because the first schema is encoded in bits at both ends of the first individual. No matter where the crossover point is selected, the first schema will be broken up and will not be passed on. The most popular solution for this has been the use of *two-point crossover*. This operator is like one-point crossover, except that two cut points rather than one are selected at random, and two individuals are swapped between the two cut points. For example,

Parent-1	11 01 : 100101 : 1011
Parent-2	0001 : 011011 : 1100
Child-1	11 01 011011 1011
Child-2	0001 100101 1100

The semicolons indicate the two crossover points. The schemata can be combined with two-point crossover.

The termination module is relatively simple. Currently for the prototype, if a given total number of evaluations is met, the genetic operation will end.

4.3. LM-Transformation, Simulation and Accumulation

LM-Transformation, Simulation, and *Accumulation* modules together can be treated as an evaluation function by a genetic algorithm, which is the domain dependent part in GA's. For each individual in the current population, these three modules are called in sequence. These modules evaluate fitness for individuals and put the fitness into the population structure as a normal GA evaluation function does. In addition, they retain GA exploration experiences and deal with different levels of representations.

The LM-transformation module takes an individual string as an input, e.g.,

"011011011111 001011011101 011011011111

001011011101 011011011111 001011011101",

which is a low-level representation for a daily workload on the six workstations (see Figure 7 (e)). It then uses background knowledge (see Figure 9) and the value of the variable **worset**, e.g., (W1, W3, W4, W6, W2, W5) to transform the individual string into the mid-level representation shown in Figure 10. The first 12 bits in the string are converted to a workload for workstation W1. Bits 1-3 "011" is translated into 10 am; bits 4-6 "011" is translated into 4 hours; bits 7-9 "011" is translated into 16 requests; and bits 10-12 "111" is translated into 150 MegaBytes. And so on. The node m7 of the mid-level representation will be passed to the simulation module.

The simulation module currently is a pseudo-PETSA. The original PETSA (Performance Evaluation Tool for System Architecture) [Pavicic and Ding, 1991] is a multicomputer performance simulator and applied to the Mayo/IBM medical image archival and retrieval system. PETSA predicts the performance effects of variation in network configuration and loading characteristics. PETSA has been implemented in Turbo Pascal 5.5, and currently runs on IBM PC only. Since it can not be directly run on DECstation or other available department equipment networked with DECstation, we have replaced it with

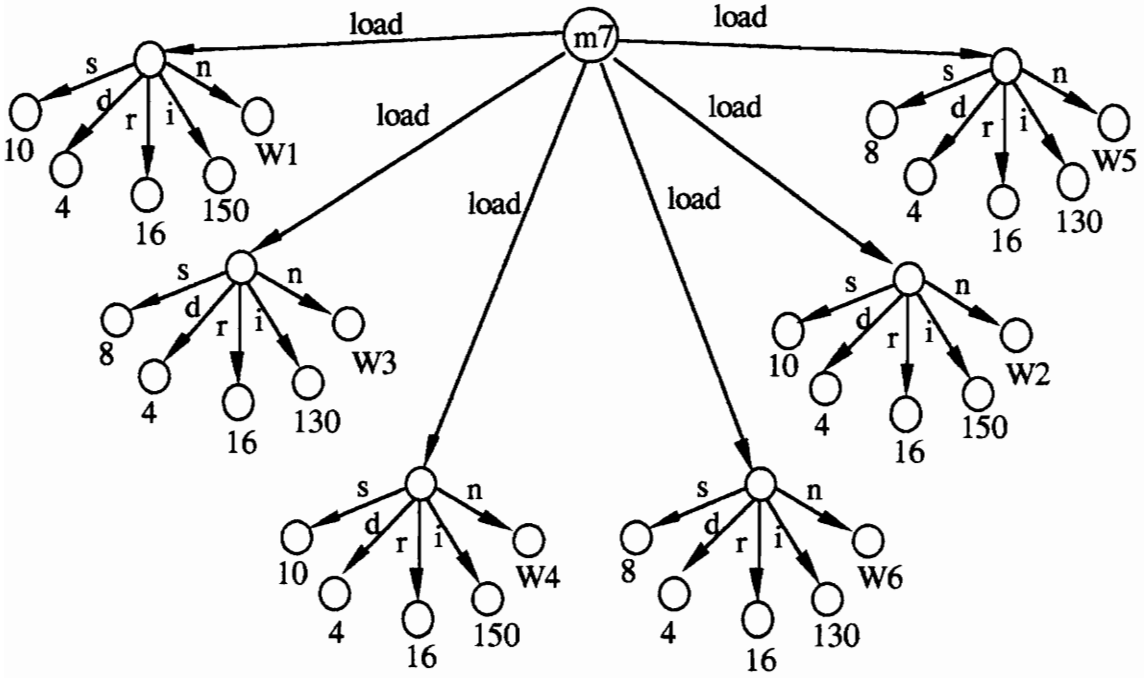


Figure 10. An individual in mid-level representation

a simpler simulator, pseudo-PETSA in Common LISP, for the prototype implementation. Simplifying PETSA only affects the features for which the system optimizes, but does not affect the methods used to retain knowledge in our prototype system.

The simulation module takes the node of the mid-level representation for an individual, and uses the background knowledge in knowledge base such as the task system configuration to get a similar PETSA input. Then it calculates the worst response time on each workstation for a daily workload, calculates overall average worst response time, and finally generates a fitness value which is equal to

C0 * total-amount-of-images, if T1 < average-worst-response-time, or otherwise,
C0 * total-amount-of-images + C1 * (T1 - average-worst-response-time)

where

C0 = [1 / (number-of-workstations)],
C1 = [1 / (number-of-servers * number-of-rings)],
T1 = 100, and
worst-response-time-on-a-workstation = [size-of-images-per-request * number-of-requests-at-same-time].

The fitness value and the node of the mid-level representation for the individual will be passed to the accumulation module.

The accumulation module has a threshold, which is set to be the population size, to limit the total number of experiences saved in the knowledge base, and has a counter for the number of experiences saved so far. It accumulates genetic exploration experiences by the following steps.

(1) If the current generation is the initial generation and the current individual is the first individual evaluated so far, then two experiences (Figure 11) built from this individual

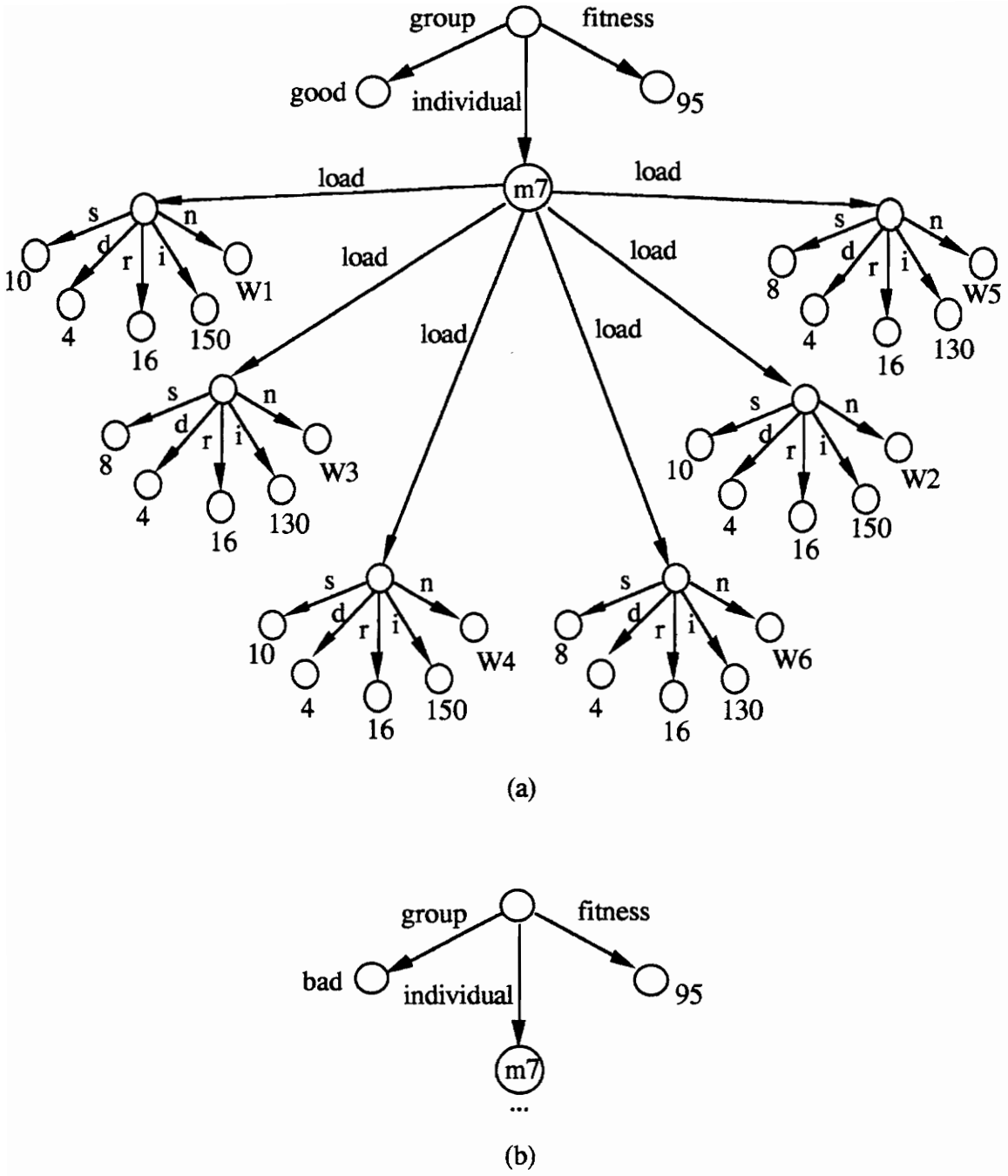


Figure 11. Accumulated experiences in mid-level representation

are added into knowledge base, and the two top nodes are saved in the global variable **prebest** and **preworst** in SNePSUL package respectively.

(2) If the current generation is the initial generation and the current individual is not the first individual, the current fitness is compared with the best and worst fitness in **prebest** and **preworst**. If the current fitness is better than the fitness in **prebest**, the node in **prebest** is erased, the new experience is built, and its top node is assigned to **prebest**. If the current fitness is worse than the fitness in **preworst**, the similar thing will be done.

(3) If the current generation is not the initial generation and the current individual is the first individual, then the experience is built and its top node is saved in **curbest** and **curworst** respectively, and the current fitness is compared with the fitness in **prebest** and **preworst**. If the current fitness is same as or better than the fitness in **prebest**, the experience is added into the knowledge base. If the current fitness is same as or worse than the fitness in **preworst**, the similar thing will be done. The counter may be changed, and the threshold may be checked. If the counter is over the threshold, a search routine will be called to search through the experiences in knowledge base, and either the best in the bad class or the worst in the good class will be removed.

(4) If the current generation is not the initial generation and the current individual is not the first individual, the current fitness is compared with the fitness in **curbest** and **curworst**, and the node in **curbest** or **curworst** may be replaced. The current fitness is also compared with the fitness in **prebest** and **preworst**. If the current fitness is same as or better than the fitness in **prebest**, the experience is added into knowledge base. If the current fitness is same as or worse than the fitness in **preworst**, the similar thing will be done. The counter may be changed, and the

threshold may be checked. If the counter is over the threshold, the similar thing as in (3) will be done. The node in **prebest** or **preworst** will not be replaced unless the end of the current generation is met and the node in **curbest** is better than the node in **prebest** or the node in **curworst** is worse than the node in **preworst**.

4.4. MH-Transformation and Generalization

MH-Transformation, and *Generalization* modules are beyond the range of a genetic algorithm. They deal with experiences, which are collected along as the GA goes, and try to extract a general rule from the raw knowledge.

The MH-transformation module needs substantial background knowledge -- attribute frames and generalization hierarchies, which suggest the focus of generalization, and deduction rules, which tell how to transform an experience in the mid-level representation to a rule instance in the high-level representation for generalization. The module analyzes accumulated experiences and converts them into rule instances by deduction rules, then drops inconsistent instances if they belong to both of positive and negative instance sets.

In the illustrative domain, suppose we are given two attribute frames: (1) *totima*, the total amount of images to be retrieved or archived by all workstations in a task system; and (2) *stadis*, the biggest start-time distance among all workstations. Suppose that totima can be *small*, *medium*, *large*, or *huge*, and that stadis can be *short*, *fairly-long*, or *very-long* (see Figure 12). Suppose that deduction rules are: (1) if total amount of images is between 10 and 99 MegaBytes, totima is *small*; if total amount of images is between 100 and 499 MegaBytes, totima is *medium*; if total amount of images is between 500 and 999 MegaBytes, totima is *large;* and otherwise, totima is *huge.* (2) if the biggest difference among all start-times is between 0 and 2, stadis is *short*; if the biggest difference among all

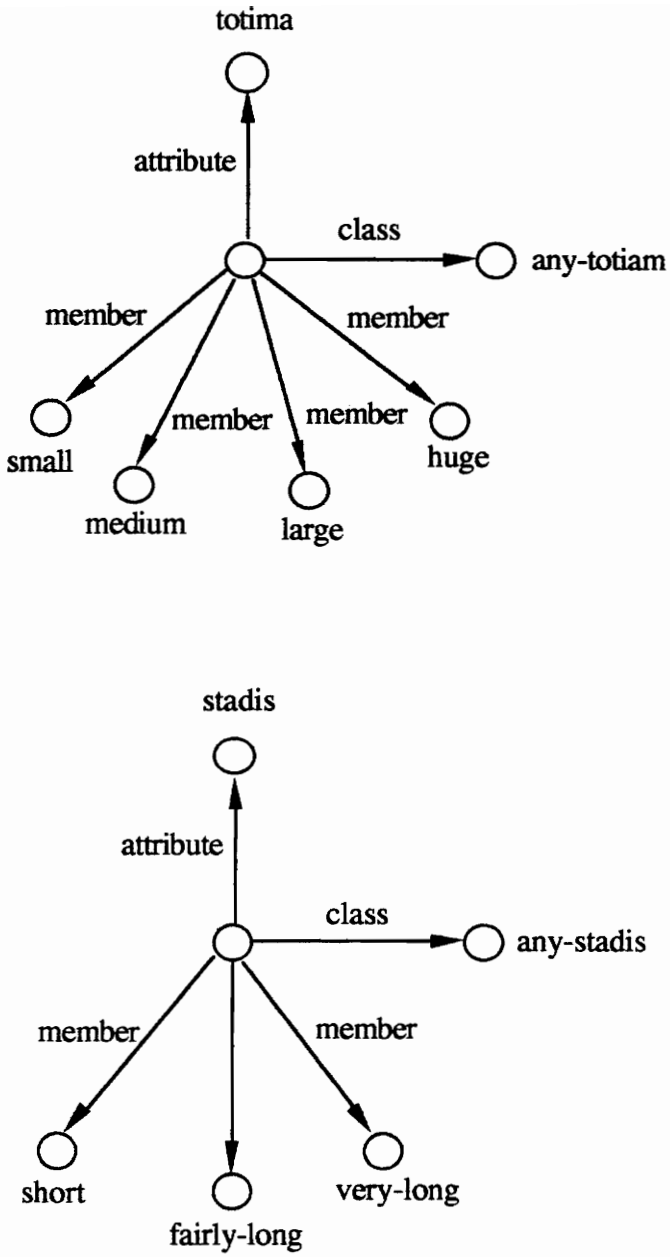


Figure 12. Two generalization hierarchies

start-times is between 3 and 4, stadis is *fairly-long;* and otherwise, stadis is *very-long.* Suppose that a task system has two workstations W1 and W2. And suppose that five experiences have accumulated, abbreviated to:

[class: good, fitness: 130, workload: <W1, 10, 7, 24, 150> <W2, 9, 8, 8, 110>]
[class: good, fitness: 130, workload: <W1, 7, 2, 8, 110> <W2, 10, 8, 4, 150>]
[class: good, fitness: 130, workload: <W1, 10, 7, 24, 150> <W2, 14, 7, 8, 110>]
[class: good, fitness: 130, workload: <W1, 10, 7, 24, 150> <W2, 13, 8, 8, 110>]
[class: bad, fitness: 20, workload: <W1, 9, 3, 8, 10> <W2, 11, 3, 16, 30>]

After the MH-transformation, three rule instances are abstracted out (see Figure 13), which are abbreviated to:

[positive, (totima = medium) and (stadis = short)]
[positive, (totima = medium) and (stadis = fairly-long)]
[negative, (totima = small) and (stadis = short)]

Notice that the second, third, and fourth experiences map to the second rule instance.

The generalization module currently uses the candidate elimination algorithm [Mitchell, 1982] [Hirsh, 1990], which is implemented as follows.

Initialize *S*-set to be the set containing the first positive instance, and *G*-set to be the set of maximally general generalization.
for each subsequent instance, i, **begin**
if i is a positive instance **then begin**
- Remove those elements of the *G*-set that do not cover the new positive instance.
- Generalize the elements of the *S*-set as little as possible so that they cover the new positive instance.
- Remove from the *S*-set those elements that is more general than some other element in *S*-set. Also remove those element that are no longer covered by some element of the *G*-set.

end

else if i is a negative instance **then begin**

- Remove those elements of the *S*-set that cover the new negative instance.
- Specialize the elements of the *G*-set as little as possible so that they no longer cover the new negative instance.
- Remove from *G*-set those elements that are not most general and those no longer cover some element of the *S*-set.

end

end

The rule is determined when the *S*- and *G*- sets have the same single element.

From the above three instances and the two generalization hierarchies shown in Figure 12, the candidate elimination algorithm goes through the following steps:

(1) *S*-set = { (totima = medium) and (stadis = short) }
G-set = { (totima = any-totima) and (stadis = any-stadis) }

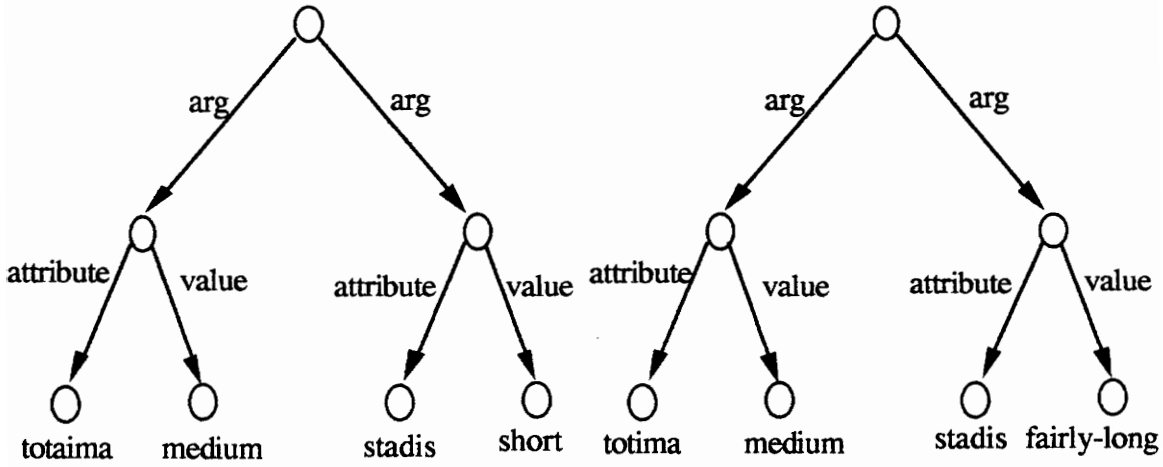
(2) *S*-set = { (totima = medium) and (stadis = any-stadis) }
G-set = { (totima = any-totima) and (stadis = any-stadis) }

(3) *S*-set = { (totima = medium) and (stadis = any-stadis) }
G-set = { (totima = huge) and (stadis = any-stadis),
(totima = large) and (stadis = any-stadis),
(totima = medium) and (stadis = any-stadis),
(totima = any-totima) and (stadis = fairly-long),
(totima = any-totima) and (stadis = very-long) }
G-set = { (totima = medium) and (stadis = any-stadis) }

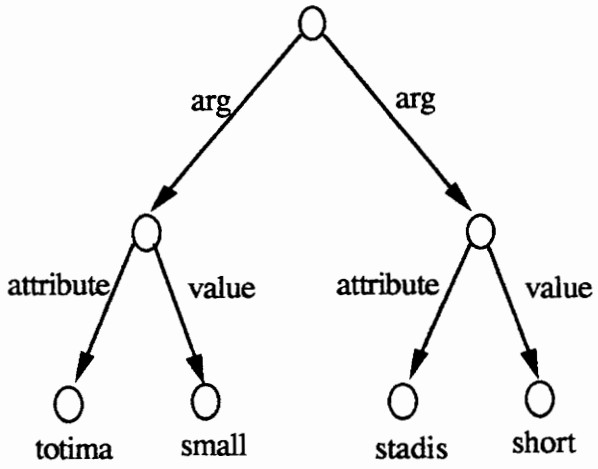
and generates a rule:

(totima = medium) and (stadis = any-stadis)

which means that no matter when each workstation starts to work during a day, the total amount of images retrieved or archived by all the workstations in the task system should not be over 500 MegaBytes.



(a) Positive instances



(b) Negative instance

Figure 13. Three high-level rule instances

Chapter 5. Results

The prototype of knowledge retention with genetic algorithms by three levels of representation and learning has been tested on the illustrative domain. Results of experiments for two different task system configurations are given in some detail in section 5.1 and section 5.2 (refer to appendix A and B for detailed result descriptions and actual I/O scripts). Results from eight further experiments for eight new configurations are summarized in section 5.3.

5.1. Experiment 1

Given Conditions

- A task system configuration: one token ring R1 attached with a controller C1, a server S1, and two workstations W1 and W2.
- Background knowledge: the interpretation for 4-tuples (see Figure 9 and section 4.2), two generalization hierarchies (see Figure 12), and deduction rules (see section 4.4).
- GA parameter settings: the population size is 20, and the total number of trials is 100, i.e., the total number of generations is 5.

Objective

- To retain a rule, which generally tells how to assign workloads to the workstations in the given task system, during and after a search for a near-maximum workload.

Knowledge Retention

(1) *At low-level*

At the low-level genetic exploration, workloads for two workstations in the given configuration are represented by length-24 binary strings. In generation 1, the highest

fitness value is 140; the lowest fitness value is 20; the average fitness value is 85.5; and the median of the twenty fitness values is 85.

After each generation, individuals with high fitness values occur more frequently. In generation 5, the highest fitness value is still 140; the lowest fitness value is now 40; the average fitness value has increased to 95; and the median of the twenty fitness values is 95.

Comparing generation 5 with generation 1, we can see that the average fitness value has improved by 11 percent, and the median has increased by 10. Although the highest fitness value, 140, in generation 5 is the same as that in generation 1, three individuals with fitness value 140 instead of one now exist in generation 5.

(2) *At mid-level*

Through the LM-transformation, seven individuals with fitness value 140, and one individual with a low fitness value 20 have accumulated in the mid-level representation. These seven good experiences and one bad experience are abbreviated as follows:

[class: good, fitness: 140, workload: <W1, 9, 3, 8, 130> <W2, 10, 8, 4, 150>]
[class: good, fitness: 140, workload: <W1, 7, 2, 8, 130> <W2, 13, 5, 24, 150>]
[class: good, fitness: 140, workload: <W1, 11, 2, 8, 130> <W2, 14, 3, 8, 150>]
[class: good, fitness: 140, workload: <W1, 7, 2, 8, 130> <W2, 12, 1, 24, 150>]
[class: good, fitness: 140, workload: <W1, 7, 2, 8, 130> <W2, 12, 8, 4, 150>]
[class: good, fitness: 140, workload: <W1, 11, 2, 8, 130> <W2, 12, 1, 24, 150>]
[class: good, fitness: 140, workload: <W1, 7, 6, 8, 130> <W2, 12, 8, 4, 150>]
[class: bad, fitness: 20, workload: <W1, 9, 3, 8, 10> <W2, 11, 3, 16, 30>]

Comparing the above experiences with generation 5, we can see that generation 5 only contains three individuals with fitness value 140, but the accumulation module has saved four more good experiences and one more bad experience, which form the raw knowledge retained at the mid-level representation during the genetic exploration.

As knowledge retention continues, several solutions for a near-maximum workload have been found at the end of the genetic exploration. The English text describing one such solution from the natural language interface is as follows:

"One near-maximum workload for the configuration is as follows: workstation W1 starts to work at 7, continues work for 6 hours, and submits a total number of 8 requests to archive or retrieve 130 MB images; workstation W2 starts to work at 12, continues work for 8 hours, and submits a total number of 4 requests to archive or retrieve 150 MB images."

(3) *At high-level*

By the MH-transformation, the above accumulated good and bad experiences are extracted to form high-level positive and negative rule instances, which are abbreviated as follows:

[positive, (totima = medium) and (stadis = short)]
[positive, (totima = medium) and (stadis = very-long)]
[positive, (totima = medium) and (stadis = fairly-long)]
[negative, (totima = small) and (stadis = short)]

Looking closely at the abstraction, we can see that the first and sixth good experiences have mapped to the first positive rule instance; the second, fourth, and fifth experiences have mapped to the second rule instance; the third experience has mapped to the third rule instance; and the bad experience has mapped to the negative rule instance.

After generalization on these instances by the candidate elimination algorithm, a rule is induced, which is abbreviated to:

(totima = medium) and (stadis = any-stadis)

This rule covers all above positive instances and no negative instance, and it is consistent with the accumulated experiences. The English text from the natural language interface to explain this rule is as follows:

"A general rule is as follows: -- totima = medium AND stadis = any-stadis -- which means that no matter when each workstation starts to work, the total amount of image retrieved or archived by all the workstations in the current task system should not be over the medium amount (100 - 500 MB)."

5.2. Experiment 2

Given Conditions

- A task system configuration: two token rings R1 and R2; a bridge B1 connecting these two rings; a server S1, workstations W1, W2, and W3 attached to ring R1; a controller C1, servers S2 and S3, and workstations W4 and W5 attached to ring R2.
- Background knowledge: the interpretation for 4-tuples (see Figure 9 and section 4.2), two generalization hierarchies (see Figure 12), and deduction rules (see section 4.4).
- GA parameter settings: the population size is 50, and the total number of trials is 250, i.e., the total number of generations is 5.

Objective

- To retain a rule, which generally tells how to assign workloads to the workstations in the given task system, during and after a search for a near-maximum workload.

Knowledge Retention

(1) *At low-level*

At the low-level genetic exploration, workloads for five workstations in the given configuration are represented by length-60 binary strings. In generation 1, the highest fitness value is 122; the lowest fitness value is 30; the average fitness value is 83.92; and the median of the fifty fitness values is 86.

After each generation, individuals with high fitness values occur more often, and the highest fitness gets higher. In generation 5, the highest fitness value is 138; the lowest fitness value is 58; the average fitness value is 109.12; and the median of the fifty fitness values is 110.

Comparing generation 5 with generation 1, we can see that the average fitness value in generation 5 has improved by 30 percent, the highest fitness value has increased by 16, and the median has increased by 24.

(2) *At mid-level*

Through the LM-transformation, ten individuals with high fitness values and one individual with a low fitness value are accumulated in the mid-level representation during genetic exploration. These good and bad experiences are abbreviated as follows:

[class: good, fitness: 134, workload: <W1, 7, 6, 8, 150> <W2, 13, 2, 16, 150> <W3, 14, 8, 20, 130> <W4, 9, 2, 32, 130> <W5, 10, 7, 24, 130>]

[class: good, fitness: 134, workload: <W1, 14, 2, 16, 130> <W2, 14, 5, 12, 130> <W3, 14, 6, 20, 150> <W4, 9, 7, 24, 150> <W5, 14, 7, 8, 110>]

[class: good, fitness: 138, workload: <W1, 10, 4, 4, 150> <W2, 13, 2, 16, 130> <W3, 14, 8, 20, 130> <W4, 9, 2, 32, 130> <W5, 10, 7, 8, 150>]

[class: good, fitness: 126, workload: <W1, 7, 2, 24, 130> <W2, 13, 7, 32, 150> <W3, 11, 5, 12, 110> <W4, 12, 8, 24, 90> <W5, 11, 7, 24, 150>]

[class: good, fitness: 134, workload: <W1, 14, 2, 16, 130> <W2, 14, 5, 12, 130> <W3, 14, 6, 20, 150> <W4, 9, 7, 24, 150> <W5, 14, 7, 24, 110>]

[class: good, fitness: 134, workload: <W1, 7, 6, 8, 150> <W2, 13, 2, 16, 130> <W3, 14, 8, 20, 130> <W4, 9, 2, 32, 130> <W5, 10, 7, 8, 130>]

[class: good, fitness: 130, workload: <W1, 12, 5, 8, 130> <W2, 9, 5, 16, 130> <W3, 7, 1, 24, 150> <W4, 7, 6, 8, 130> <W5, 14, 7, 8, 110>]

[class: good, fitness: 126, workload: <W1, 7, 2, 24, 130> <W2, 13, 7, 32, 150> <W3, 11, 5, 12, 110> <W4, 12, 8, 24, 90> <W5, 14, 4, 4, 150>]

[class: good, fitness: 126, workload: <W1, 14, 3, 24, 150> <W2, 10, 3, 4, 130> <W3, 11, 5, 12, 150> <W4, 12, 6, 24, 150> <W5, 13, 8, 20, 50>]

[class: good, fitness: 122, workload: <W1, 14, 3, 24, 150> <W2, 10, 3, 4, 130>
<W3, 11, 5, 12, 130> <W4, 12, 6, 24, 150> <W5, 13, 8, 20, 50>]
[class: bad, fitness: 30, workload: <W1, 10, 8, 16, 10> <W2, 7, 2, 28, 10>
<W3, 11, 4, 28, 50> <W4, 11, 8, 32, 70> <W5, 12, 7, 28, 10>]

Comparing the above experiences with generation 5, we can see that accumulation has saved the best experience, the two second best experiences, a few other good experiences, and one more bad experience, which are the raw knowledge retained at the mid-level representation.

As knowledge retention continues, several solutions for a near-maximum workload have been found at the end of the genetic exploration. The English text for describing one such solution from the natural language interface is as follows:

"One near-maximum workload for the configuration is as follows: workstation W1 starts to work at 10, continues work for 4 hours, and submits a total number of 4 requests to archive or retrieve 150 MB images; workstation W2 starts to work at 13, continues work for 2 hours, and submits a total number of 16 requests to archive or retrieve 130 MB images; workstation W3 starts to work at 14, continues work for 8 hours, and submits a total number of 20 requests to archive or retrieve 130 MB images; workstation W4 starts to work at 9, continues work for 2 hours, and submits a total number of 32 requests to archive or retrieve 130 MB images; workstation W5 starts to work at 10 continues work for 7 hours, and submits a total number of 8 requests to archive or retrieve 150 MB images."

(3) *At high-level*

By the MH-transformation, the above accumulated good and bad experiences are extracted to form high-level positive and negative rule instances, which are abbreviated as follows:

[positive, (totima = large) and (stadis = very-long)]
[positive, (totima = large) and (stadis = fairly-long)]
[negative, (totima = medium) and (stadis = very-long)]

Looking closely at the abstraction, we can see that the first to the eighth good experiences have mapped to the first positive rule instance; the ninth and tenth experiences have mapped to the second rule instance; and the bad experience has mapped to the negative rule instance. After generalization on these instances by the candidate elimination algorithm, a rule is induced, which is abbreviated to:

(totima = large) and (stadis = any-stadis)

This rule covers all above positive instances and no negative instance, and it is consistent with the accumulated experiences. The English text for explaining this rule from the natural language interface is as follows:

"A general rule is as follows -- totima = large AND stadis = any-stadis -- which means no matter when each workstation starts to work the total amount of images retrieved or archived by all the workstations in the current task system should not be over the large amount -- 500 - 1000 MB."

5.3. Eight More Experiments

Given Conditions

Eight task system configurations and GA settings are given below. Background knowledge is the same as the one in experiments 1 and 2.

Experiment #	Task System Configurations	GA Parameter Settings
3	token ring R1 attached with workstation W1, server S1, controller C1.	population size = 10 total trials = 50
4	token ring R1 attached with workstations W1, W2 and W3, server S1, controller C1.	population size = 30 total trials = 150
5	token ring R1 attached with workstations W1, W2, W3 and W4, server S1, controller C1.	population size = 40 total trials = 200

6	token ring R1 attached with workstations W1, W2 and W3, server S1; token ring R2 attached with workstations W4, W5 and W6, servers S2 and S3, controller C1; bridge B1 connecting R1 and R2.	population size = 60 total trials = 300
7	token ring R1 attached with workstations W1, W2 and W3, server S1, controller C1; token ring R2 attached with workstations W4, W5, W6 and W7, servers S2 and S3; bridge B1 connecting R1 and R2.	population size = 70 total trials = 350
8	token ring R1 attached with workstations W1, W2,W3 and W4, servers S1 and S2, controller C1; token ring R2 attached with workstations W5, W6,W7 and W8, servers S3 and S4; bridge B1 connecting R1 and R2.	population size = 80 total trials = 400
9	token ring R1 attached with workstations W1, W2 and W3 , server S1; token ring R2 attached with workstations W4, W5 and W6, server S2, controller C1; token ring R3 attached with workstations W7, W8 and W9, server S3; bridge B1 connecting R1 and R2; bridge B2 connecting R2 and R3.	population size = 90 total trials = 450
10	token ring R1 attached with workstations W1, W2 and W3 , server S1; token ring R2 attached with workstations W4, W5, W6 and W7, servers S2 and S3, controller C1; token ring R3 attached with workstations W8, W9 and W10, server S4; bridge B1 connecting R1 and R2; bridge B2 connecting R2 and R3.	population size = 100 total trials = 500

Objective

The same as the one in experiments 1 and 2.

Knowledge Retention

(1) *At low-level*

The average, median, maximum, and minimum fitness values of generation 1 and generation 5 for the eight configurations are shown below, where#>= means the number of individuals in generation 5 whose fitness values are greater or equal to the maximum

fitness value in generation 1 and *%AveImp* means the percentage improvement of the average fitness value.

	Generation 1					Generation 5					
Exp #	Ave	Med	Max	Min	#=Max	Ave	Med	Max	Min	#>=	%AveImp
3	86	80	150	30	2	120	150	150	30	7	39.5
4	91.2	96.7	143.3	23.3	1	96	96.7	143.3	43.3	2	5.3
5	82.4	85	135	40	1	109	120	140	55	6	32.3
6	83.3	83.3	116.7	46.7	1	115.6	116.7	136.7	60	22	38.8
7	81.3	84.3	127.1	41.4	1	114.2	118.6	132.9	44.3	13	40.5
8	81.4	82.5	127.5	45	2	112.8	117.5	137.5	47.5	14	38.6
9	80.9	81.1	121.1	34.4	1	121.6	125.6	141.1	72.2	64	50.3
10	80.8	80	112	50	1	113	120	136	52	66	39.9

Hence over all ten experiments, the average fitness value in generation 5 represents an improvement over that of generation 1 by an average of 32.6%. The standard deviation is somewhat high (14.01); but if we eliminate experiment 4, which represents the only strong outlier on the distribution, the average improvement becomes 35.7%, with a more reasonable standard deviation of 10.8. In addition, all results except that of experiment 4 lie within a single standard deviation of the mean, showing reasonable clustering.

(2) *At mid-level*

The numbers of accumulated good and bad experiences for the eight configurations are shown below.

Exp #	Number of Good Experiences		Number of Bad Experiences	
3	13	(fitness >= 150)	5	(fitness <= 30)
4	5	(fitness >= 143.3)	1	(fitness <= 23.3)
5	4	(fitness >= 135)	3	(fitness <= 40)
6	11	(fitness >= 116.7)	2	(fitness <= 46.7)

7	7	(fitness >= 127.1)	1	(fitness <= 41.4)
8	19	(fitness >= 127.5)	2	(fitness <= 45)
9	24	(fitness >= 121.1)	1	(fitness <= 34.4)
10	44	(fitness >= 112)	1	(fitness <= 50)

(3) *At high-level*

The extracted rule instances, and rules for the eight configurations are as follows.

Exp #	Positive Instances	Negative Instances	Rule
3	totima = medium *and* stadis = short	totima = small *and* stadis = short	*G*-set: totima = medium *and* stadis = any-stadis *S*-set: totima = medium *and* stadis = short
4	totima = medium *and* stadis = short totima = medium *and* stadis = fairly-long totima = medium *and* stadis = very-long	totima = small *and* stadis = fairly-long	totima = medium *and* stadis = any-stadis
5	totima = large *and* stadis = very-long totima = large *and* stadis = short	totima = medium *and* stadis = fairly-long totima = medium *and* stadis = very-long	totima = large *and* stadis = any-stadis
6	totima = large *and* stadis = very-long totima = large *and* stadis = fairly-long	totima = medium *and* stadis = very-long	totima = large *and* stadis = any-stadis
7	totima = large *and* stadis = very-long	totima = medium *and* stadis = very-long	*G*-set: totima = large *and* stadis = any-stadis *S*-set: totima = large *and* stadis = very-long
8	totima = huge *and* stadis = very-long totima = huge *and* stadis = fairly-long	totima = medium *and* stadis = very-long	totima = huge *and* stadis = any-stadis

9	totima = huge *and* stadis = very-long	totima = medium *and* stadis = very-long	*G*-set: totima = huge *and* stadis = any-stadis *S*-set: totima = huge *and* stadis = very-long
10	totima = huge *and* stadis = very-long	totima = large *and* stadis = very-long	*G*-set: totima = huge *and* stadis = any-stadis *S*-set: totima = huge *and* stadis = very-long

The candidate elimination algorithm does not know how many training instances it needs to determine a unique rule before it reaches a point where the *S*- and *G*- sets contain the same single element. Therefore, even after genetic exploration, accumulation, and MH-transformation, the generalization module may not have received enough rule instances, i.e., when the candidate elimination algorithm has gone through all rule instances, *S*- and *G*- sets may not be equal or may contain more than one element. In this case, the prototype system produces a rule in propositional calculus of attribute-value pairs, which uses logical *or* to connect each element in *S*- and *G*- sets for representing the current version space (see section 2.1). This rule then will be refined in later use.

Inductive learning suffers from a fundamental weakness: it is inherently not susceptible to complete validation. However, our prototype system does not have this problem, since when knowledge retained is used to seed an initial population sometime later, the genetic exploration will recover the possible error in the previously retained knowledge automatically.

Chapter 6. Conclusion and Further Work

In this thesis, a prototype system has been developed to combine low-level genetic algorithm with high-level traditional symbolic AI to retain domain-specific knowledge. The keys to using these two paradigms together are as follows:

- Using mid-level representations and two relatively simple transformations to bridge the gap.
- Viewing inputs and outputs of low-level genetic learning as codifications (i.e., representations) of high-level information.
- Adopting techniques that restrict the domain-dependent portions to details of background knowledge represented by mid-level frame structures, and the level projection functions.

The keys to retaining knowledge are as follows:

- GA's change parameters at the low-level.
- The accumulation collects the good and bad experiences at the mid-level in parallel with GA's search.
- The MH-transformation abstracts rule instances from the mid-level to the high-level.
- The candidate elimination algorithm induces a rule at the high-level.

The strategies and methods used in the prototype system are mainly domain-independent, which can easily be adapted to other application domains. Although the background knowledge is domain-dependent, the representation formats for background knowledge still remain domain-independent.

In principle, the same techniques can be used to bridge the gap between neural networks and high-level representations, or to connect neural networks and genetic algorithms in larger systems. In this way, autonomous systems can enjoy the benefits of all levels of representation, with minimal overhead for themselves or designers.

There is a number of interesting directions in which this work can be refined and extended:

- To replace the pseudo simulator by a more realistic simulator in the prototype system to make the knowledge retained more accurate.
- To use retained knowledge in later similar cases and to see how this knowledge can help improve the performance of design systems.
- To theoretically analyze and generalize the methods designed and used in the prototype system for combining low-level and high-level of representation and learning.

Appendix A. Detailed Description of Results

The following gives the detailed descriptions of two experiment results.

A.1. Experiment 1

Given Conditions

- A small task system configuration: only one token ring R1 attached with a controller C1, a server S1, and two workstations W1 and W2.
- Background knowledge: the interpretation for 4-tuples (see Figure 9 and section 4.2), two generalization hierarchies (see Figure 12), and deduction rules (see section 4.4).
- GA parameter settings: the population size is 20, and the total number of trials is 100, i.e., the total number of generations is 5.

Objective

- To retain a rule, which generally tells how to assign workloads to the workstations in the given task system, during and after a search for a near-maximum workload.

Knowledge Retention

(1) *At low-level*

At the low-level genetic search, workloads for two workstations in the given configuration are represented by length-24 binary strings. A portion of individuals and their fitness values in the first generation is shown below:

000101001110101111000111	140
100010101101110000101111	130
011110101111111110001101	130
...	...

011101111000111100001011 40
010010001000100010011001 20

In generation 1, the fitness values of the twenty individuals are given in a descending order:

140	130	130	120	100	100	100	100	90	90
80	80	80	80	80	70	50	50	20	20

After each generation, individuals with high fitness values occur more and more. In generation 5, the fitness values of individuals are given below:

140	140	140	120	120	120	100	100	100	100
90	90	90	90	80	80	60	60	40	40

Comparing generation 5 with generation 1, we can see that the overall fitness values in generation 5 have been improved. Although the highest fitness value, 140, in generation 5 is the same as that in generation 1, three individuals with fitness value 140 instead of one now exist in generation 5.

(2) *At mid-level*

Through the LM-transformation, seven instead of three individuals with fitness value 140, and one individual with a low fitness value 20 are accumulated in the mid-level representation during the five generations' exploration. These seven good experiences and one bad experience are represented by the following top-nodes of their semantic networks:

good: M307! M260! M257! M160! M140! M134! M67!
bad: M39!

Their corresponding semantic networks in text forms can be found in appendix B.1. For example, the semantic network for node M307! is:

(M307! (CLASS (good) FITNESS (140) INDIVIDUAL (M306)))
(M306 (INDIVIDUAL- (M307!) LOAD (M304 M305)))
(M304 (LOAD- (M306) NAME (W1) START (9) DURATION (3)
REQUEST (8) IMAGE (130)))
(M305 (LOAD- (M306) NAME (W2) START (10) DURATION (8)
REQUEST (4) IMAGE (150)))

Their graphical forms are similar to Figure 11, and abbreviated as follows:

M307! --[class: good, fitness: 140, workload: <W1, 9, 3, 8, 130> <W2, 10, 8, 4, 150>]
M260! --[class: good, fitness: 140, workload: <W1, 7, 2, 8, 130> <W2, 13, 5, 24, 150>]
M257! --[class: good, fitness: 140, workload: <W1, 11, 2, 8, 130> <W2, 14, 3, 8, 150>]
M160! --[class: good, fitness: 140, workload: <W1, 7, 2, 8, 130> <W2, 12, 1, 24, 150>]
M140! --[class: good, fitness: 140, workload: <W1, 7, 2, 8, 130> <W2, 12, 8, 4, 150>]
M134!--[class: good, fitness: 140, workload: <W1, 11, 2, 8, 130> <W2, 12, 1, 24, 150>]
M67! -- [class: good, fitness: 140, workload: <W1, 7, 6, 8, 130> <W2, 12, 8, 4, 150>]
M39! -- [class: bad, fitness: 20, workload: <W1, 9, 3, 8, 10> <W2, 11, 3, 16, 30>]

One solution for a near-maximum workload has been found, which is represented by node M67!. The English text for describing the solution from the natural language interface is as follows:

"ONE NEAR-MAXIMUM WORKLOAD FOR THE CONFIGURATION IS AS FOLLOWS: WORKSTATION W1 STARTS TO WORK AT 7 CONTINUES WORK FOR 6 HOURS AND SUBMITS A TOTAL NUMBER OF 8 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES; WORKSTATION W2 STARTS TO WORK AT 12 CONTINUES WORK FOR 8 HOURS AND SUBMITS A TOTAL NUMBER OF 4 REQUESTS TO ARCHIVE OR RETRIEVE 150 MB IMAGES."

(3) *At high-level*

After the MH-transformation, the above accumulated good and bad experiences are extracted to the following high-level positive and negative rule instances, represented by the top-nodes of their semantic networks:

positive: M318! M320! M322! M320! M320! M318! M320!
(i.e., M318! M320! M322!)
negative: M324!

Their semantic networks in text forms are:

(M318! (ARG (M316 M317)))
(M316 (ARG- (M318! ...) ATTRIBUTE (totima) VALUE (medium)))
(M317 (ARG- (M318! ...) ATTRIBUTE (stadis) VALUE (short))) ...

Their graphical forms are similar to Figure 13, abbreviated as follows:

M318! -- [positive, (totima = medium) and (stadis = short)]
M320! -- [positive, (totima = medium) and (stadis = very-long)]
M322! -- [positive, (totima = medium) and (stadis = fairly-long)]
M324! -- [negative, (totima = small) and (stadis = short)]

After the generalization on these instances, a rule is induced by the candidate elimination algorithm, which is represented by node M328! and abbreviated to:

(totima = medium) and (stadis = any-stadis)

The English text from the natural language interface to explain this rule is as follows:

"A GENERAL RULE IS AS FOLLOWS: -- totima = medium AND stadis = any-stadis -- WHICH MEANS THAT NO MATTER WHEN EACH WORKSTATION STARTS TO WORK, THE TOTAL AMOUNT OF IMAGES RETRIEVED OR ARCHIVED BY ALL THE WORKSTATIONS IN THE CURRENT TASK SYSTEM SHOULD NOT BE OVER THE MEDIUM AMOUNT (100 - 500 MB)"

A.2. Experiment 2

Given Conditions

- A relative large task system configuration: two token rings R1 and R2; a bridge B1 connecting these two rings; a server S1, workstations W1, W2, and W3 attached to ring R1; a controller C1, servers S2 and S3, and workstations W4 and W5 attached to ring R2.
- Background knowledge: the interpretation for 4-tuples (see Figure 9 and section 4.2), two generalization hierarchies (see Figure 12), and deduction rules (see section 4.4).
- GA parameter settings: the population size is 50, and the total number of trials is 250, i.e., the total number of generations is 5.

Objective

- To retain a rule, which generally tells how to assign workloads to the workstations in the given task system, during and after a search for a near-maximum workload.

Knowledge Retention

(1) *At low-level*

At the low-level genetic search, workloads for five workstations in the given configuration are represented by length-60 binary strings. A portion of individuals and their fitness values in the first generation is shown below:

```
111010101111011010000110100100010110101101101111110111100010   122
011010001111011000110111000001111101000110111011101001001101   118
100110011111111000101111011011100111011010011001101110010101 1   110
...                                                                ...
011101100101111001000000000110100001011001001000001000101011    46
011111011000000001110000100011110010100111111011101110110000    30
```

In generation 1, the fitness values of the fifty individuals are given below in a descending order:

122	118	110	110	110	106	106	106	106	106
102	98	98	94	94	94	94	94	94	90
90	90	90	90	86	86	82	82	82	82
82	78	78	78	74	70	70	70	70	66
66	62	62	58	58	58	58	50	46	30

After each generation, individuals with high fitness values occur more and more, the highest fitness gets higher. In generation 5, the fitness values of individuals are given below:

138	134	134	134	130	130	130	130	126	126
126	122	122	122	122	118	118	114	114	114
114	114	110	110	110	110	110	110	110	110
110	106	106	106	102	102	102	102	102	102
94	94	94	90	82	82	74	70	66	58

Comparing generation 5 with generation 1, we can see that the overall fitness values in generation 5 have improved significantly, and the highest fitness value has increased from 122 in generation 1 to 138 in generation 5.

(2) *At mid-level*

Through the LM-transformation, ten individuals with high fitness values and one individual with a low fitness value are accumulated in the mid-level representation during the five generations' exploration. These good and bad experiences are represented by the following top-nodes of their semantic networks:

good: M1317! M1314! M1228! M1158! M1084! M1071! M1038! M1017! M738! M162!
bad: M241!

Their corresponding semantic networks in text forms can be found in appendix B.2. For example, the semantic network for node M1317! is:

(M1317! (CLASS (good) FITNESS (134) INDIVIDUAL (M1316)))
(M1316 (INDIVIDUAL- (M1317!) LOAD (M1065 M1066 M1067 M1068 M1315)))
(M1065 (LOAD- (M1316 ...) NAME (W1) START (7) DURATION (6) REQUEST (8) IMAGE (150)))
(M1066 (LOAD- (M1316 ...) NAME (W2) START (13) DURATION (2) REQUEST (16) IMAGE (130)))
(M1067 (LOAD- (M1316 ...) NAME (W3) START (14) DURATION (8) REQUEST (20) IMAGE (130)))
(M1068 (LOAD- (M1316 ...) NAME (W4) START (9) DURATION (2) REQUEST (32) IMAGE (130)))
(M1315 (LOAD- (M1316) NAME (W5) START (10) DURATION (7) REQUEST (24) IMAGE (130)))

Their graphical forms are similar to Figure 11, and abbreviated as follows:

M1317!--[class: good, fitness: 134, workload: <W1, 7, 6, 8, 150> <W2, 13, 2, 16, 150> <W3, 14, 8, 20, 130> <W4, 9, 2, 32, 130> <W5, 10, 7, 24, 130>]
M1314!--[class:good, fitness:134, workload:<W1, 14, 2, 16, 130> <W2, 14, 5, 12, 130> <W3, 14, 6, 20, 150> <W4, 9, 7, 24, 150> <W5, 14, 7, 8, 110>]
M1228!--[class: good, fitness: 138, workload:<W1, 10, 4, 4, 150> <W2, 13, 2, 16, 130> <W3, 14, 8, 20, 130> <W4, 9, 2, 32, 130> <W5, 10, 7, 8, 150>]
M1158!--[class: good, fitness: 126, workload:<W1, 7, 2, 24, 130> <W2, 13, 7, 32, 150> <W3, 11, 5, 12, 110> <W4, 12, 8, 24, 90> <W5, 11, 7, 24, 150>]
M1084!--[class:good, fitness:134, workload:<W1, 14, 2, 16, 130> <W2, 14, 5, 12, 130> <W3, 14, 6, 20, 150> <W4, 9, 7, 24, 150> <W5, 14, 7, 24, 110>]
M1071! --[class: good, fitness: 134, workload: <W1, 7, 6, 8, 150> <W2, 13, 2, 16, 130> <W3, 14, 8, 20, 130> <W4, 9, 2, 32, 130> <W5, 10, 7, 8, 130>]
M1038! --[class: good, fitness: 130, workload: <W1, 12, 5, 8, 130> <W2, 9, 5, 16, 130> <W3, 7, 1, 24, 150> <W4, 7, 6, 8, 130> <W5, 14, 7, 8, 110>]

M1017!--[class: good, fitness: 126, workload:<W1, 7, 2, 24, 130> <W2, 13, 7, 32, 150>
<W3, 11, 5, 12, 110> <W4, 12, 8, 24, 90> <W5, 14, 4, 4, 150>]
M738! --[class: good, fitness: 126, workload: <W1, 14, 3, 24, 150> <W2, 10, 3, 4, 130>
<W3, 11, 5, 12, 150> <W4, 12, 6, 24, 150> <W5, 13, 8, 20, 50>]
M162! --[class: good, fitness: 122, workload: <W1, 14, 3, 24, 150> <W2, 10, 3, 4, 130>
<W3, 11, 5, 12, 130> <W4, 12, 6, 24, 150> <W5, 13, 8, 20, 50>]
M241! --[class: bad, fitness: 30, workload: <W1, 10, 8, 16, 10> <W2, 7, 2, 28, 10>
<W3, 11, 4, 28, 50> <W4, 11, 8, 32, 70> <W5, 12, 7, 28, 10>]

One solution for a near-maximum workload has been found, which is represented by node M1228!. The English text for describing that solution from the natural language interface is as follows:

"ONE NEAR-MAXIMUM WORKLOAD FOR THE CONFIGURATION IS AS FOLLOWS: WORKSTATION W1 STARTS TO WORK AT 10 CONTINUES WORK FOR 4 HOURS AND SUBMITS A TOTAL NUMBER OF 4 REQUESTS TO ARCHIVE OR RETRIEVE 150 MB IMAGES; WORKSTATION W2 STARTS TO WORK AT 13 CONTINUES WORK FOR 2 HOURS AND SUBMITS A TOTAL NUMBER OF 16 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES; WORKSTATION W3 STARTS TO WORK AT 14 CONTINUES WORK FOR 8 HOURS AND SUBMITS A TOTAL NUMBER OF 20 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES; WORKSTATION W4 STARTS TO WORK AT 9 CONTINUES WORK FOR 2 HOURS AND SUBMITS A TOTAL NUMBER OF 32 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES; WORKSTATION W5 STARTS TO WORK AT 10 CONTINUES WORK FOR 7 HOURS AND SUBMITS A TOTAL NUMBER OF 8 REQUESTS TO ARCHIVE OR RETRIEVE 150 MB IMAGES"

(3) *At high-level*

After the MH-transformation, the above accumulated good and bad experiences are extracted to the following high-level positive and negative rule instances, represented by the top-nodes of their semantic networks:

positive: M1410! M1410! M1410! M1410! M1410! M1410! M1410! M1410!
M1412! M1412!
(i.e., M1410! M1412!)
negative: M1414!

Their semantic networks in text forms are:

(M1410! (ARG (M1408 M1409)))
(M1408 (ARG- (M1410! ...) ATTRIBUTE (totima) VALUE (large)))
(M1409 (ARG- (M1410! ...) ATTRIBUTE (stadis) VALUE (very-long))) ...

Their graphical forms are similar to Figure 13, and are abbreviated as follows:

M1410! -- [positive, (totima = large) and (stadis = very-long)]
M1412! -- [positive, (totima = large) and (stadis = fairly-long)]
M1414! -- [negative, (totima = medium) and (stadis = very-long)]

After the generalization on these instances, a rule is induced by the candidate elimination algorithm, which is represented by node M1228! and abbreviated to:

(totima = large) and (stadis = any-stadis)

The English text for explaining this rule from the natural language interface is as follows:

"A GENERAL RULE IS AS FOLLOWS -- totima = large AND stadis = any-stadis -- NO MATTER WHEN EACH WORKSTATION STARTS TO WORK THE TOTAL AMOUNT OF IMAGES RETRIEVED OR ARCHIVED BY ALL THE WORKSTATIONS IN THE CURRENT TASK SYSTEM SHOULD NOT BE OVER THE LARGE AMOUNT -- RANGE 500 - 1000 MB"

Appendix B. Two Experiment Scripts

The following gives the actual I/O from two experiments of the prototype system on DECstation.

B.1. For a Small Task System

```
==> sneps-ga

Allegro CL 3.1.12.2 [DECstation] (11/19/90)
Copyright (C) 1985-1990, Franz Inc., Berkeley, CA, USA


<cl> (sneps)

  Welcome to SNePS-2.1

 Copyright 1984, 88, 89 by Research Foundation of State University of New York

  4/25/1991 14:22:35


* ^(atnin "driver.atn")

--> undefined- (NIL NIL)
; Loading /u2/ding/thesis/kr/myfile.lisp.
; Fast loading /u2/ding/thesis/kr/backgr.fasl.
; Fast loading /u2/ding/thesis/kr/initia.fasl.
; Fast loading /u2/ding/thesis/kr/LM-tra.fasl.
; Fast loading /u2/ding/thesis/kr/simula.fasl.
; Fast loading /u2/ding/thesis/kr/accumu.fasl.
; Fast loading /u2/ding/thesis/kr/evalua.fasl.
; Fast loading /u2/ding/thesis/kr/myooga.fasl.
; Fast loading /u2/ding/thesis/kr/MH-tra.fasl.
; Fast loading /u2/ding/thesis/kr/genera.fasl.
; Fast loading /u2/ding/thesis/kr/learns.fasl.
State S processed.
State SP processed.
State SP/VERB processed.
State SP/OBJ processed.
State SP/1 processed.
```

State SP/2 processed.
State SP/3 processed.
State SP/4 processed.
State SP/5 processed.
State SP/END processed.
State NP processed.
State NP/DET processed.
State NP/ADJ processed.
State NP/END processed.
State NL processed.
State NL/END processed.
State R processed.
State G processed.
State E processed.

Atnin read in states: (E G R NL/END NL NP/END NP/ADJ NP/DET NP SP/END SP/5 SP/4 SP/3 SP/2 SP/1 SP/OBJ SP/VERB SP S NIL NIL NIL)

CPU time : 4.22 GC time : 0.00

* ^(parse)

-->
ATN parser initialization...

Trace level = 0.

Beginning at state 'S'.

Input sentences in normal English orthographic convention.
May go beyond a line by having a space followed by a <CR>
To exit parser, write ^end.

: there is 1 ring: R1.

Time (sec.): 0.183
Resulting parse:

(I UNDERSTAND THAT THERE IS ring - R1)

: there is 1 controller: C1.

Time (sec.): 0.2
Resulting parse:

(I UNDERSTAND THAT THERE IS controller - C1)

: there is 1 server: S1.

Time (sec.): 0.217
Resulting parse:

(I UNDERSTAND THAT THERE IS server - S1)

: there are 2 workstations: W1, W2.

Time (sec.): 0.233
Resulting parse:

(I UNDERSTAND THAT THERE ARE workstation - W1 W2)

: C1, S1, W1, W2 are on R1.

Time (sec.): 0.284
Resulting parse:

(I UNDERSTAND THAT R1 CONNECTS C1 S1 W1 W2)

: what is a maximum workload?

AT 20 BEST 20 CHROMOSOMES ARE:

000101001110101111000111 140
100010101101110000101111 130
011110101111111110001101 130
010100011110001110111101 120
101010011101000110001100 100
001011100011101011001110 100
101110010100011011100101 100
110000100101110011101100 100
000001001110111010110010 90
010110101111101111010001 90
111101110001101010101110 80
101101110111100010101000 80
000001110100000000001011 80
000001111100011010100011 80
010000001001011111000110 80
101110000011011000001011 70
110100101100000010111000 50
111011001001000000000011 50
001110001001001010100000 20
010010001000100010011001 20

AT 40 BEST 20 CHROMOSOMES ARE:

000001001110101111000111 140
100001001110101000101111 140
000101001110101111000111 140
000110101101110111000111 130
000001111100011010001111 120
101110010011101011001111 110
110000100101110011101100 100
001011100100011011100100 90
101101110100000000001011 80
000001110100000000001011 80
000001111100011010100011 80
010000001001011111000110 80
000001101100011010100011 80
000010000011011000001011 70
010110110010110100111100 70
101010011101000110100000 60
010000100000101000000100 50
110100101100000010111000 50
110100101100000010111000 50
110100111100000010111000 50

AT 60 BEST 20 CHROMOSOMES ARE:

000001001110101000101111 140
000001001110101111000111 140
101010011101000110100111 130
000000111100011010001111 120
100001111100011010001111 120
100001110101000011001101 110
010110001110101111111100 110
110001100101110011101100 100
010000100101101000000100 100
000001110010110100000111 100
010111100100011000111100 90
101101110100000000001011 80
010010000001011111101101 70
001010110010110111100100 70
100110101011010100110011 70
100110101010001011011100 70
110000100000110011101100 50
011001001000001011111011 40
101110010011101011001000 40
001110001000001010101010 30

AT 80 BEST 20 CHROMOSOMES ARE:

000001001110110100101111 140
100001001110111010001111 140
000001001110101000101111 140
000001111100001111000111 120
100101111100011010001111 120

```
100001110110110011001101   120
000001001110101111101101   120
001110100011101110011111   110
000000010010010010111111   100
010011100100011000111101   100
000001110001000100000111   90
010010000001011111000111   90
100110100101110011100011   90
101001110100000000001011   80
000001110000001010000111   80
110001101011010100111100   80
011001001010110101111011   60
111110101001011101110100   60
010110000001011111101100   60
011101111000111010001011   40
```

AT 100 BEST 20 CHROMOSOMES ARE:

```
010010001110011111000111   140
100001001110111010001111   140
000001001110110100101111   140
000001001110101000101101   120
000001001110111010001101   120
000000011100001110111111   120
000000010010010010111111   100
000000010010010010111111   100
001000101110010001111011   100
000001110010010011000111   100
000001000001101000101111   90
000001110001000010000111   90
100110110001000000111111   90
000001100001011110000111   90
101001110100000111001011   80
010010010000001011000111   80
011001001010110101111011   60
000110011010010001101011   60
011101111000111100001011   40
011101111000101111101011   40
```

Time (sec.): 76.267
Resulting parse:

(ONE NEAR-MAXIMUM WORKLOAD FOR THE CONFIGURATION IS AS FOLLOWS -- WORKSTATION W1 STARTS TO WORK AT 7 CONTINUES WORK FOR 6 HOURS AND SUBMITS A TOTAL NUMBER OF 8 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES -- WORKSTATION W2 STARTS TO WORK AT 12 CONTINUES WORK FOR 8 HOURS AND SUBMITS A TOTAL NUMBER OF 4 REQUESTS TO ARCHIVE OR RETRIEVE 150 MB IMAGES)

: ^end

ATN Parser exits...

CPU time : 78.05 GC time : 0.00

* ^*good-nod-set*

-->
(M307! M260! M257! M160! M140! M134! M67!)

CPU time : 0.03 GC time : 0.00

* ^*bad-nod-set*

-->
(M39!)

CPU time : 0.02 GC time : 0.00

* ^*prebest-nod*

-->
M67!

CPU time : 0.02 GC time : 0.00

* ^*preworst-nod*

-->
M39!

CPU time : 0.00 GC time : 0.00

* ^(parse)

-->
ATN parser initialization...

Trace level = 0.

Beginning at state 'S'.

Input sentences in normal English orthographic convention.
May go beyond a line by having a space followed by a <CR>
To exit parser, write ^end.

: what is a general rule?

Time (sec.): 3.35
Resulting parse:

(A GENERAL RULE IS AS FOLLOWS -- totima = medium AND stadis = any-stadis -- WHICH MEANS THAT NO MATTER WHEN EACH WORKSTATION STARTS TO WORK THE TOTAL AMOUNT OF IMAGES RETRIEVED OR ARCHIVED BY ALL THE WORKSTATIONS IN THE CURRENT TASK SYSTEM SHOULD NOT BE OVER THE MEDIUM AMOUNT -- RANGE 100 - 500 MB)

: ^end

ATN Parser exits...

CPU time : 3.52 GC time : 0.00

* ^*pos-instances*

-->
(M318! M320! M322! M320! M320! M318! M320!)

CPU time : 0.05 GC time : 0.00

* ^*neg-instances*

-->
(M324!)

CPU time : 0.01 GC time : 0.00

* ^*rule-nod*

-->
M328!

CPU time : 0.03 GC time : 0.00

* (dump *nodes)

(1 (DURATION- (M132) LOW- (M2) POSITION- (M1) STEP- (M1 M2)))
(10 (IMAGE- (M36) LOW- (M4) START- (M305)))
(11 (START- (M131 M37)))
(12 (START- (M132 M65) TUPLENGTH- (M5!)))
(13 (START- (M258)))

(130 (IMAGE- (M131 M138 M304 M64)))
(14 (START- (M255) UP- (M1)))
(140 (FITNESS- (M134! M140! M160! M257! M260! M307! M67!)))
(150 (IMAGE- (M132 M255 M258 M305 M65) UP- (M4)))
(16 (REQUEST- (M37)))
(2 (DURATION- (M131 M138) POSITION- (M2)))
(20 (FITNESS- (M39!) STEP- (M4)))
(24 (REQUEST- (M132 M258)))
(3 (DURATION- (M255 M304 M36 M37) LENGTH- (M1 M2 M3 M4) POSITION (M3)))
(30 (IMAGE- (M37)))
(32 (UP- (M3)))
(4 (LOW- (M3) POSITION- (M4) REQUEST- (M305 M65) STEP- (M3)))
(5 (DURATION- (M258)))
(6 (DURATION- (M64)))
(7 (LOW- (M1) START- (M138 M64)))
(8 (DURATION- (M305 M65) REQUEST- (M131 M138 M255 M304 M36 M64) UP (M2)))
(9 (START- (M304 M36)))
(C1 (NAME- (M10 M16)))
(M1 (START- (M5!) GROW (inc) LENGTH (3) LOW (7) POSITION (1) STEP (1) UNIT (hour) UP (14)))
(M10 (MEMBER- (M11!) NAME (C1)))
(M11! (CLASS (controller) MEMBER (M10)))
(M12 (MEMBER- (M13!) NAME (S1)))
(M13! (CLASS (server) MEMBER (M12)))
(M131 (LOAD- (M133 M256) DURATION (2) IMAGE (130) NAME (W1) REQUEST (8) START (11)))
(M132 (LOAD- (M133 M159) DURATION (1) IMAGE (150) NAME (W2) REQUEST (24) START (12)))
(M133 (INDIVIDUAL- (M134!) LOAD (M131 M132)))
(M134! (CLASS (good) FITNESS (140) INDIVIDUAL (M133)))
(M138 (LOAD- (M139 M159 M259) DURATION (2) IMAGE (130) NAME (W1) REQUEST (8) START (7)))
(M139 (INDIVIDUAL- (M140!) LOAD (M138 M65)))
(M14 (MEMBER- (M15!) NAME (W1 W2)))
(M140! (CLASS (good) FITNESS (140) INDIVIDUAL (M139)))
(M15! (CLASS (workstation) MEMBER (M14)))
(M159 (INDIVIDUAL- (M160!) LOAD (M132 M138)))
(M16 (COMPONENT- (M17!) NAME (C1 S1 W1 W2)))
(M160! (CLASS (good) FITNESS (140) INDIVIDUAL (M159)))
(M17! (COMPONENT (M16) CONNECTOR (M8)))
(M2 (DURATION- (M5!) GROW (inc) LENGTH (3) LOW (1) POSITION (2) STEP (1) UNIT (hour) UP (8)))
(M255 (LOAD- (M256) DURATION (3) IMAGE (150) NAME (W2) REQUEST (8) START (14)))
(M256 (INDIVIDUAL- (M257!) LOAD (M131 M255)))
(M257! (CLASS (good) FITNESS (140) INDIVIDUAL (M256)))
(M258 (LOAD- (M259) DURATION (5) IMAGE (150) NAME (W2) REQUEST (24) START (13)))
(M259 (INDIVIDUAL- (M260!) LOAD (M138 M258)))

(M260! (CLASS (good) FITNESS (140) INDIVIDUAL (M259)))
(M3 (REQUEST- (M5!) GROW (inc) LENGTH (3) LOW (4) POSITION (3) STEP (4) UNIT (number) UP (32)))
(M304 (LOAD- (M306) DURATION (3) IMAGE (130) NAME (W1) REQUEST (8) START (9)))
(M305 (LOAD- (M306) DURATION (8) IMAGE (150) NAME (W2) REQUEST (4) START (10)))
(M306 (INDIVIDUAL- (M307!) LOAD (M304 M305)))
(M307! (CLASS (good) FITNESS (140) INDIVIDUAL (M306)))
(M316 (ARG- (M318! M320! M322! M328!) ATTRIBUTE (totima) VALUE (medium)))
(M317 (ARG- (M318! M324!) ATTRIBUTE (stadis) VALUE (short)))
(M318! (ARG (M316 M317)))
(M319 (ARG- (M320! M334!) ATTRIBUTE (stadis) VALUE (very-long)))
(M320! (ARG (M316 M319)))
(M321 (ARG- (M322! M333!) ATTRIBUTE (stadis) VALUE (fairly-long)))
(M322! (ARG (M316 M321)))
(M323 (ARG- (M324!) ATTRIBUTE (totima) VALUE (small)))
(M324! (ARG (M317 M323)))
(M325 (ARG- (M327! M333! M334!) ATTRIBUTE (totima) VALUE (any-totima)))
(M326 (ARG- (M327! M328! M330! M332!) ATTRIBUTE (stadis) VALUE (any-stadis)))
(M327! (ARG (M325 M326)))
(M328! (ARG (M316 M326)))
(M329 (ARG- (M330!) ATTRIBUTE (totima) VALUE (huge)))
(M330! (ARG (M326 M329)))
(M331 (ARG- (M332!) ATTRIBUTE (totima) VALUE (large)))
(M332! (ARG (M326 M331)))
(M333! (ARG (M321 M325)))
(M334! (ARG (M319 M325)))
(M36 (LOAD- (M38) DURATION (3) IMAGE (10) NAME (W1) REQUEST (8) START (9)))
(M37 (LOAD- (M38) DURATION (3) IMAGE (30) NAME (W2) REQUEST (16) START (11)))
(M38 (INDIVIDUAL- (M39!) LOAD (M36 M37)))
(M39! (CLASS (bad) FITNESS (20) INDIVIDUAL (M38)))
(M4 (IMAGE- (M5!) GROW (inc) LENGTH (3) LOW (10) POSITION (4) STEP (20) UNIT (MB) UP (150)))
(M5! (DURATION (M2) IMAGE (M4) REQUEST (M3) START (M1) TUPLENGTH (12)))
(M6! (CLASS (any-totima) ATTRIBUTE (totima) MEMBER (huge large medium small)))
(M64 (LOAD- (M66) DURATION (6) IMAGE (130) NAME (W1) REQUEST (8) START (7)))
(M65 (LOAD- (M139 M66) DURATION (8) IMAGE (150) NAME (W2) REQUEST (4) START (12)))
(M66 (INDIVIDUAL- (M67!) LOAD (M64 M65)))
(M67! (CLASS (good) FITNESS (140) INDIVIDUAL (M66)))
(M7! (CLASS (any-stadis) ATTRIBUTE (stadis) MEMBER (fairly-long short very-long)))
(M8 (CONNECTOR- (M17!) MEMBER- (M9!) NAME (R1)))
(M9! (CLASS (ring) MEMBER (M8)))
(MB (UNIT- (M4)))
(R1 (NAME- (M8)))
(S1 (NAME- (M12 M16)))

```
(W1 (NAME- (M131 M138 M14 M16 M304 M36 M64)))
(W2 (NAME- (M132 M14 M16 M255 M258 M305 M37 M65)))
(any-stadis (CLASS- (M7!) VALUE- (M326)))
(any-totima (CLASS- (M6!) VALUE- (M325)))
(bad (CLASS- (M39!)))
(controller (CLASS- (M11!)))
(fairly-long (MEMBER- (M7!) VALUE- (M321)))
(good (CLASS- (M134! M140! M160! M257! M260! M307! M67!)))
(hour (UNIT- (M1 M2)))
(huge (MEMBER- (M6!) VALUE- (M329)))
(inc (GROW- (M1 M2 M3 M4)))
(large (MEMBER- (M6!) VALUE- (M331)))
(medium (MEMBER- (M6!) VALUE- (M316)))
(number (UNIT- (M3)))
(ring (CLASS- (M9!)))
(server (CLASS- (M13!)))
(short (MEMBER- (M7!) VALUE- (M317)))
(small (MEMBER- (M6!) VALUE- (M323)))
(stadis (ATTRIBUTE- (M317 M319 M321 M326 M7!)))
(totima (ATTRIBUTE- (M316 M323 M325 M329 M331 M6!)))
(very-long (MEMBER- (M7!) VALUE- (M319)))
(workstation (CLASS- (M15!)))
(1 10 11 12 13 130 14 140 150 16 2 20 24 3 30 32 4 5 6 7 8 9 C1 M1 M10 M11! M12
M13! M131 M132 M133 M134! M138 M139 M14 M140! M15! M159 M16 M160! M17!
M2 M255 M256 M257! M258 M259 M260! M3 M304 M305 M306 M307! M316 M317
M318! M319 M320! M321 M322! M323 M324! M325 M326 M327! M328! M329 M330!
M331 M332! M333! M334! M36 M37 M38 M39! M4 M5! M6! M64 M65 M66 M67! M7!
M8 M9! MB R1 S1 W1 W2 any-stadis any-totima bad controller fairly-long good hour
huge inc large medium number ring server short small stadis totima very-long workstation)

 CPU time : 3.25    GC time : 0.00


* (lisp)
"End of SNePS"


<cl> (exit)
; Exiting Lisp
```

B.2. For a Relatively Large Task System

```
==> sneps-ga

Allegro CL 3.1.12.2 [DECstation] (11/19/90)
Copyright (C) 1985-1990, Franz Inc., Berkeley, CA, USA


<cl> (sneps)

  Welcome to SNePS-2.1

 Copyright 1984, 88, 89 by Research Foundation of State University of New York

  4/25/1991 14:37:48


* ^(atnin "driver.atn")

--> undefined- (NIL NIL)
; Loading /u2/ding/thesis/kr/myfile.lisp.
; Fast loading /u2/ding/thesis/kr/backgr.fasl.
; Fast loading /u2/ding/thesis/kr/initia.fasl.
; Fast loading /u2/ding/thesis/kr/LM-tra.fasl.
; Fast loading /u2/ding/thesis/kr/simula.fasl.
; Fast loading /u2/ding/thesis/kr/accumu.fasl.
; Fast loading /u2/ding/thesis/kr/evalua.fasl.
; Fast loading /u2/ding/thesis/kr/myooga.fasl.
; Fast loading /u2/ding/thesis/kr/MH-tra.fasl.
; Fast loading /u2/ding/thesis/kr/genera.fasl.
; Fast loading /u2/ding/thesis/kr/learns.fasl.
State S processed.
State SP processed.
State SP/VERB processed.
State SP/OBJ processed.
State SP/1 processed.
State SP/2 processed.
State SP/3 processed.
State SP/4 processed.
State SP/5 processed.
State SP/END processed.
State NP processed.
State NP/DET processed.
State NP/ADJ processed.
State NP/END processed.
State NL processed.
```

State NL/END processed.
State R processed.
State G processed.
State E processed.

Atnin read in states: (E G R NL/END NL NP/END NP/ADJ NP/DET NP SP/END SP/5
SP/4 SP/3 SP/2 SP/1 SP/OBJ SP/VERB SP S NIL NIL NIL)

CPU time : 4.33 GC time : 0.00

* ^(parse)

-->
ATN parser initialization...

Trace level = 0.

Beginning at state 'S'.

Input sentences in normal English orthographic convention.
May go beyond a line by having a space followed by a <CR>
To exit parser, write ^end.

: there are 2 rings: R1, R2.

Time (sec.): 0.233
Resulting parse:

(I UNDERSTAND THAT THERE ARE ring - R1 R2)

: there is 1 bridge: B1.

Time (sec.): 0.217
Resulting parse:

(I UNDERSTAND THAT THERE IS bridge - B1)

: there are 3 servers: S1, S2, S3.

Time (sec.): 0.267
Resulting parse:

(I UNDERSTAND THAT THERE ARE server - S1 S2 S3)

: there are 5 workstations: W1, W2, W3, W4, W5.

Time (sec.): 0.3
Resulting parse:

(I UNDERSTAND THAT THERE ARE workstation - W1 W2 W3 W4 W5)

: B1 connects R1, R2.

Time (sec.): 0.184
Resulting parse:

(I UNDERSTAND THAT B1 CONNECTS R1 R2)

: S1, W1, W2, W3 are on R1.

Time (sec.): 0.266
Resulting parse:

(I UNDERSTAND THAT R1 CONNECTS S1 W1 W2 W3)

: C1, S2, S3, W4, W5 are on R2.

Time (sec.): 0.266
Resulting parse:

(I UNDERSTAND THAT R2 CONNECTS C1 S2 S3 W4 W5)

: what is a maximum workload?

AT 50 BEST 50 CHROMOSOMES ARE:

```
11101010111101101000011010010001011010110110111111010111100010  122
01101000111101100011011100000011111010001101110111010010011101  118
10011001111111000101111011011100111011010011001101110010101 1  110
11011100010101011010110010001010010101011111111100100001011 01  110
00001011100010001010110111000010111100010100111010111100011 1  110
01010111011011001101010011101100011111001000110111001000111 00  106
10101010111010010101001001111010110010011101111110010101110 1  106
10000011111011111101110101111111110011000010110101101100110 0  106
01101101111111010100010110011000010110010111101010011000110 1  106
10111001001000011001111011101101010101001101010111100111011 0  106
00110100110001011011011011101000111001001011100000101011111 1  102
11011101111001010100001111011110111001001101110111001001001 0  98
10000000101010011111011111111010111011001000100001110111111 1  98
11010010000100001111011110101011011100000011101000111100010 0  94
10100100111101010011001000110111111111100011010101110100100 11  94
00001001001100100000101110010001000111100010001011111001000 11  94
10000001001010110110000111101100111011000000011001011110111 0  94
```

0110010111101111000101100101011001101000110000110011110100000 94
0111110001100000011111000110101000110101101011111011110100001 94
0001000001000011100011111001000101011011111011001110110100000 90
0000011011101101101100000100101101110100101011001110110110011 90
0110001001010110110100100000111001000110011001100001100001011 90
1011011101111000101010001111011100011010101011100101000111110 90
0110111001010000010011101110101100100000011101000000000001011 90
1101111101011101101110010001010011101110100110011111111110110 86
1011100000110110000010110010111000111010110011101101001011100 86
1011000011100101000100100000011011011001101110001111100000101 82
1011100111000000000101010111000001000100100100010101001101000 82
0100001111000000110000111111101110110001110111010101110110011 82
1100001101111011110000100110000010101011111100110111111100100 82
0110101001000011101111000010101100111000111111101111010010010 82
0100010000101011011101001001000100100000100000010010010010111 78
1011111000111101110110101100000001000010101011010000101000011 78
1110110010010000000000110111101011111111100011010100000001001 78
1011011000101111110001011101111101100101111100011101011100100 74
1111100000001001000101111101101110111001001011010101100000000 70
0000011010011001110011000001111110000010110011010011110101010 70
0010011110010010110100110001010000011111011111001000100011110 70
0011101111010011100010010010101000001010100111010001100011000 70
0000010000111011001011001110111001001011111000110101001110000 66
0100100010001000100110011100001001011100111011001011100101000 66
0101111100000010011110100011001110111101110110101011110011011 62
1010011110001110101101011110100100101011100001101111000110000 62
1000100100101100101000000110111011110000111110011100011100010 58
1111011010100100010000101111110010011111011110011111000111110 58
0101001100010010000000100010101110011000101111101001100110100 58
0010101110000101100000001111001101111100011101000111110110010 58
1001010010010110111110000101101000111000110110110101001010011 50
0111011001011110010000000001101000010110010010000010001010011 46
0111110110000000011100001000111100101001111110111011101100000 30

AT 100 BEST 50 CHROMOSOMES ARE:

1110101011110110100001101001000101101011011011111101111000010 122
0110110111111100010111101101100001011001011110101001100011101 110
0000101110001000101011011100001011110001010011101111110000111 110
0001010010101011101001010111111001100100011111100010100011110 110
0000011011101101101111111001000101011011111011001110110110011 110
0110110111111110101000101100110000101100101111010100110101101 106
1001100111111110101000101100111001110110100110011011100101011 106
1010010011110110001101110000011111010000011010101110100100011 106
0110100011110101001100100011011111111110110111011101001001101 106
0000101110001000101011011100001011110001001011010111110001111 106
0110010111101000101011011100001011110001010011101011110100000 106
1001100111101111000101100101011001101000110100110111001010011 106
0110111001010000010011101110101101011001011111000000000001011 102
0010000101100011111100100110100101101010110110111001111011110 102
0000010111010100001101001011100011010000100111010111111111100 102

```
11011001111111000101111011011100101010111111001101111110010098
11010010010110111011101101010000011001010101101101111010110198
00001011100011110001011001010110011010001100001100111100011198
11000011011110100101111011011100101010111111001101111110010098
10110000111001010001111000000110110110011011100011111000010198
01100101111111000101111011011100111011010010001100111101000098
01101101111111010100010110011000001000000111001010011000110194
01010110111011110100111001001011010110010000010011001010000094
10000010101001100111001101011010111001110111101110011001011194
01100101111011110001011011110110011010001100001100111101000094
10000011011110111100001001100000111011010011001101110010101194
11011101111001010101110000101011001110001111111011111001001094
01111100011001010001110001101010001101011010111110111101000194
01010111011011001101010011101100011110010001100010010001110094
10110000111001010001001000000110110110011011101111111000010194
10011001111111011100001001100000111011010011001101110010101194
10011001111111000101001011011100111011010011001101110010101194
10110111011110001010100011110001001010111000011011110001111094
10010110101001000101110101111111110011000010110101011001100090
01101110010100000100111001001011001000000111010000000000101190
10110110001011111100010111111010110010011101111110010111001090
10101010111010010101001001011111011001011111000111010101110190
10111000001101100000101100101110001110101100111011010000110086
10000100011010110000101100101110101110111000101001000001111086
01101010010000111010001111011110111001001101110111000100100186
01101010010010010001010000101011001110001111111011110100100182
10110000111000000111001000000110110110011011100011111000010182
10011011011101111110100110000111001000101011111010100101000178
11100011111011111100001011111100100111111011110011110001111074
11111000000010010001011111011011101110010100111010011000000074
01011010010110000010000110001010000110100110101011000101111174
11111000000000111011111111011011101110010010110101011000000070
00010000010000111000000001001011011101001010110011101101000070
10100111100011101011010111101001001010111000011010110001100062
10100111100011101011010111101111000110101010111001010001100058
```

AT 150 BEST 50 CHROMOSOMES ARE:

```
11101010111101101000011010010001011110110110111111011110001012126
11101010111101101000011010010001011010110110111111011110001012122
11101010111101101000011010010001011010110110111111011110001012122
01101101111111000101111111001000101011011111010101001100010114114
01111100111010001010110111000010111101011010111110111101000114114
01100101111011110001011011110110011101011010111110111101000014114
10110000111001010001111000000110110100000011101000000000000114114
00100001011000111111001001101001011010101100111011011110110114114
00010100101010111010010101111111001100100011111100111100011104110
00000101110101000100111011101011010110001001110101111111110014110
01101101111111000101111011011000010110010111101010011010110114110
00000110111011011011111110010001010110111110110011101101101114110
00000110111011011011111101101100001011001011111001110110110114106
```

```
1011000011100101000111100000011011011001011110101001101011 106
0000011011101101101111111001000101001011111011001110110111 106
1001101011011100000011010000100101100101101000111110110111 106
0110110111111101010001011001100001011001011110101001100011 106
1001100111111101010001011001110011101101000001001100101000 102
1000010001101011000010110010111001011011111011001110110111 102
0110111001010000010011100100101101011001000001000000000011 102
0111001110111100010011101000101010000101101101110010010000 111 102
1110000001111111100001110100001011101010011010101011111101001 102
1011000010001000101011011100001011110001010011101111100000 101 102
0000101111100101000100100000011011011001101110111111110000 111 102
0110010111111100010111101101101100111000111111101111010010 01 102
1101011101101100110110110101000001100101010110110111101011 01 102
0001101110110001111101111010110101001010000111001001001101 01 102
1011000011110101001100100010011011011001101110111111110000 101 98
0101011011101111010011100100101101011001001100110111001010 10 98
0110010111101111000101101111011000110101101011110011110100 00 98
1100001101111010010111101101110010101011111100111011110101 00 98
0110110111111101010001011001100001011001101110001111100001 01 98
1011001101111011000100100000011011011001101110111111110000 101 98
1101100111111100010111101101110010101011111100110010111001 00 98
0111110001100101000111000110101000110101101011110111111100001 94
0000011011101101101111111001000110111011100010100100001110 11 94
0101011011101111010011101110101101011001011111001100101000 00 94
0110111001010000001101001011100011010001011111000000000011 94
0111110001100101000111000110101001101000110000111011110100 01 90
1110000111000111101101111111001000100000011101100110100010 01 90
1100001111100100101101100001110000100110001100100011000001 00 90
1000000011100101110000100110000011101101001100110111001010 11 90
0101001001011011101101001110110001111001000110001001000111 00 90
0110010101100101000111000110101000110001010011101011110100 00 86
0110101001001001000101000010110011101101001000110011110100 00 78
1100110000100100101010110101101000010011000001111111111111 11 74
0111110001100101000111000110101000101000110000110011110100 01 74
1011110101011000010001001110110110101110100001000100011110 01 74
0110111001010000010011100100101100101001101110001111100010 11 74
1011100000110110000010110010111000111010110110111001000011 00 74
```

AT 200 BEST 50 CHROMOSOMES ARE:

```
1110010111101111000101101111011001110101101011111111101011 01 134
0001010011111100010111101111111001100100011111100111000111 0 134
1011000011100101000111100000000101111000101001110111110001 101 130
0000011011101101101111111001000101011011111011001001101011 11 126
0000011011101101101111111001000101011011111011001110110001 11 126
1110101011110110100001101001000101111011011011111101111000 10 126
1110101011110110100001101001000101101011011011111101111000 10 122
1110000111010110100001101001000101101011011011111101111000 11 118
1110101011110110100001101001000111010000011111111101111000 10 118
1011000011100101000111100000001100110101101100100000000000 101 118
0001100011101000101011111010110101001010000111001001001101 01 114
```

1011000011100101000111100000000001101000001110100000000000101 114
0010000101100011111100100100000101101010110011101101111011110 114
1001101011011100010111111001000101100101101000111110111011101 114
1100011011000010111001101110000011110111010111100101111000010 110
1101011111111101010001011001100001011101010110110111101011101 110
0110110001111100010111101101100001011001011110101001101011101 110
0000010111010100010011101110101101011000100111010111111111100 110
0110010111111100010111101101101100111000111100110010111001011 106
1011000011100101000111100000011011011001011110101001101011011 106
1011000011100101000111100000011011011001011110101001101011011 106
0110110111111100000011010000100101011011111010101001100011011 106
0111111110110001111101011100001011110101101011111011111010001 102
1101011101101100110110110101011001100101010110110111101011011 102
0110010001111111100001101101101100111000111111011110100100 1 102
1110000111111100010111110100001011101010011010101011111101001 102
1011000010001000101011011110101011110001010011101111100001 01 102
0110111001010000010011100100101101011001000001000001100010 11 102
1110000111111111100001110100001011101010011010101011111101001 102
1101100011100101000111100000011011011001011110101001101011001 102
1011000111111100010111101101110010101011111100110010111001 01 102
1011110001100101000111000110101011011001101110111111100000 101 102
1011001101111011000100100000011011011001101110111111000000101 98
1011000011100101000111100000011011010000011101000000000011001 98
0111110001100101000111001001110011101101000001010011110100 01 98
1101100111111100010111101101110010101011111111101111010010 00 94
0110110111111100010111101101100001011001011110101110110 11001 94
1110101011100111101101111111001000100000011101100110100 01000 94
1011110101011000010011100100101101011001000001000100011 11001 94
0110110101101100110110110101000001100001101110001111100 00101 90
1110100100101111001111111101011010110101001001101011111111001 86
0110110110101011101001010101100001011001011110101001101011 01 86
0111001101111011000100100000011001101000110000111011110100 01 86
0101011101101100110110110101000001100101010110110011110100 00 82
0110111001010000010001001110110110101110100001000000000 01011 82
1011000010001000101011011100011011011001011110101001101001 01 78
1001100111111101010001010110101000101000110000101100101000 01 78
1011011110101001111101111001001000100001010111010011110110 00 74
0000001010101100100110110110111010011001100000101100011100 10 50
0110011100000111111101101010001110010101001110111110100110 00 50

AT 250 BEST 50 CHROMOSOMES ARE:

0110110001111110001011110111111100110010001111110011110001111 138
0001010011111100010111101111111001100100011111100111101011 10 134
1110010111101111000101101111011001110101101011111111100011 01 134
1110010111101111000101101111011001110101101011111111101011 01 134
1010101011110110100001101001000101101011010011101111100011 01 130
0000011011101100010111111001000101100111111011001110110001 11 130
0000001101111011011111111001000101011011111011001001101011 11 130
0000011011101101101111111001001001101011011011001001101011 11 130
1011000011100101000111100000000001100100011111100100000 0101 126

```
000001101110110110111111100110000101100101101100100110101111  126
000110001110100010101111101011010010100110111111011110101  126
100110101111110001011111100100010110010110100011111011011101  122
111100001110010100011110000000010111000101101111110111100010  122
000101001111110001011110111111101101000001110100001110001110  122
000110101101110001011111100100010101101111101100100110101111  122
100001101110110110111111100100010110010110100011111011011101  118
011001011111110001011110110100010110010011110011001011100101  118
101100001101110001011111100100010110010110100011111110000101  114
101100110110110110111111100101101101100110111011111000000101  114
101100001110010100011110000000101010110111110010000000000101  114
110101110110110011011011010101100110101011001110110111101101  114
111000011101011010000110100100010110101101101110100110100011  114
011011000111110001011110110110000010110010111101010011010110 1  110
011011000111110001011110110110000010110010111101010011010110 1  110
101100001100010100011110000000110011010110110010000000000101  110
001000010110001111110010010000011101000001001110110111101110  110
100110101101110110111111100100010101100110100011111011011101  110
011011000111110001011110110100010101101111111010100110101101  110
000001101111101100010010000000010101101111101100111011000111  110
111010101111011010000110100100010110101100011100100100100010  110
001001110111100101000001101000100111010110101100101111101110  110
101101110101001000000110010000100011001001001100111100110110  106
100101010101101001100111111000111101100100001110101010100001  106
000101001111110001011110110110000010110010111101010011010110 0  106
110110001110010100011110000000110110110010111101011111000110 0  102
101111000110010100011100011010101101100110111011100110100101  102
101100001110010100011110000000110011010101111010000000000011001  102
100110101000100010101101111010101111000101001110111011011101  102
001000010110001111110010010000010110010101011011011110101110  102
100110101101110001011111100110110011100110100011111011011101  102
100011111111110111011110111000111101001010011011001110110000 1  94
101101101110110100010010000000110110110011011101111100000010 1  94
100100111110000110100101100011111000000111011110010010111100  94
011011011010101110100101010110000101100101111011110111101101  90
101101000011010100011110100011000101010101101000100000010100  82
101011101110110000000001111101101111100011001011010011111001  82
000010101001111110000101110010100010010011100111101111010001  74
101111100110000000110100001010011100100100111000010010001001  70
000010001111111010100100010011110010000001110101110101111000 0  66
000011001100010100011011110101000000101001100100100011011001  58
```

Time (sec.): 302.683
Resulting parse:

(ONE NEAR-MAXIMUM WORKLOAD FOR THE CONFIGURATION IS AS FOLLOWS -- WORKSTATION W1 STARTS TO WORK AT 10 CONTINUES WORK FOR 4 HOURS AND SUBMITS A TOTAL NUMBER OF 4 REQUESTS TO ARCHIVE OR RETRIEVE 150 MB IMAGES -- WORKSTATION W2 STARTS TO WORK AT 13 CONTINUES WORK FOR 2 HOURS AND SUBMITS A TOTAL NUMBER OF 16 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES -- WORKSTATION W3

STARTS TO WORK AT 14 CONTINUES WORK FOR 8 HOURS AND SUBMITS A TOTAL NUMBER OF 20 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES -- WORKSTATION W4 STARTS TO WORK AT 9 CONTINUES WORK FOR 2 HOURS AND SUBMITS A TOTAL NUMBER OF 32 REQUESTS TO ARCHIVE OR RETRIEVE 130 MB IMAGES -- WORKSTATION W5 STARTS TO WORK AT 10 CONTINUES WORK FOR 7 HOURS AND SUBMITS A TOTAL NUMBER OF 8 REQUESTS TO ARCHIVE OR RETRIEVE 150 MB IMAGES)

: ^end

ATN Parser exits...

CPU time : 305.75 GC time : 0.00

* ^*good-nod-set*

-->
(M1317! M1314! M1228! M1158! M1084! M1071! M1038! M1017! M738! M162!)

CPU time : 0.07 GC time : 0.00

* ^*bad-nod-set*

-->
(M241!)

CPU time : 0.02 GC time : 0.00

* ^*prebest-nod*

-->
M1228!

CPU time : 0.00 GC time : 0.00

* ^*preworst-nod*

-->
M241!

CPU time : 0.02 GC time : 0.00

* ^(parse)

-->
ATN parser initialization...

Trace level = 0.

Beginning at state 'S'.

Input sentences in normal English orthographic convention.
May go beyond a line by having a space followed by a <CR>
To exit parser, write ^end.

: what is a general rule?

Time (sec.): 7.284
Resulting parse:

(A GENERAL RULE IS AS FOLLOWS -- totima = large AND stadis = any-stadis --
NO MATTER WHEN EACH WORKSTATION STARTS TO WORK, THE TOTAL AMOUNT OF IMAGES RETRIEVED OR ARCHIVED BY ALL THE WORKSTATIONS IN THE CURRENT TASK SYSTEM SHOULD NOT BE OVER THE LARGE AMOUNT -- RANGE 500 - 1000 MB)

: ^end

ATN Parser exits...

CPU time : 7.40 GC time : 0.00

* ^*pos-instances*

-->
(M1410! M1410! M1410! M1410! M1410! M1410! M1410! M1410! M1412! M1412!)

CPU time : 0.08 GC time : 0.00

* ^*neg-instances*

-->
(M1414!)

CPU time : 0.03 GC time : 0.00

* ^*rule-nod*

-->
M1418!

```
CPU time : 0.02    GC time : 0.00

* (dump *nodes)

(1 (DURATION- (M1034) LOW- (M2) POSITION- (M1) STEP- (M1 M2)))
(10 (IMAGE- (M235 M236 M239) LOW- (M4) START- (M1069 M1225 M1226 M1315
      M157 M235)))
(11 (START- (M1013 M1156 M158 M237 M238 M736)))
(110 (IMAGE- (M1013 M1036 M1082)))
(12 (REQUEST- (M1013 M1079 M158 M736) START- (M1014 M1032 M159 M239)
      TUPLENGTH- (M5!)))
(122 (FITNESS- (M162!)))
(126 (FITNESS- (M1017! M1158! M738!)))
(13 (START- (M1012 M1066 M160)))
(130 (FITNESS- (M1038!) IMAGE- (M1011 M1032 M1033 M1035 M1066 M1067
      M1068 M1069 M1078 M1079 M1315 M157 M158)))
(134 (FITNESS- (M1071! M1084! M1314! M1317!)))
(138 (FITNESS- (M1228!)))
(14 (START- (M1015 M1036 M1067 M1078 M1079 M1080 M1082 M156) UP- (M1)))
(150 (IMAGE- (M1012 M1015 M1034 M1065 M1080 M1081 M1156 M1225 M1226
      M156 M159 M736) UP- (M4)))
(16 (REQUEST- (M1033 M1066 M1078 M235)))
(2 (DURATION- (M1011 M1066 M1068 M1078 M236) POSITION- (M2)))
(20 (REQUEST- (M1067 M1080 M160) STEP- (M4)))
(24 (REQUEST- (M1011 M1014 M1034 M1081 M1082 M1156 M1315 M156 M159)))
(28 (REQUEST- (M236 M237 M239)))
(3 (DURATION- (M156 M157) LENGTH- (M1 M2 M3 M4) POSITION- (M3)))
(30 (FITNESS- (M241!)))
(32 (REQUEST- (M1012 M1068 M238) UP- (M3)))
(4 (DURATION- (M1015 M1225 M237) LOW- (M3) POSITION- (M4) REQUEST-
      (M1015 M1225 M157) STEP- (M3)))
(5 (DURATION- (M1013 M1032 M1033 M1079 M158 M736)))
(50 (IMAGE- (M160 M237)))
(6 (DURATION- (M1035 M1065 M1080 M159)))
(7 (DURATION- (M1012 M1036 M1069 M1081 M1082 M1156 M1226 M1315 M239)
      LOW- (M1) START- (M1011 M1034 M1035 M1065 M236)))
(70 (IMAGE- (M238)))
(8 (DURATION- (M1014 M1067 M160 M235 M238) REQUEST- (M1032 M1035 M1036
      M1065 M1069 M1226) UP- (M2)))
(9 (START- (M1033 M1068 M1081)))
(90 (IMAGE- (M1014)))
(B1 (NAME- (M10)))
(C1 (NAME- (M20)))
(M1 (START- (M5!) GROW (inc) LENGTH (3) LOW (7) POSITION (1) STEP (1) UNIT
      (hour) UP (14)))
(M10 (CONNECTOR- (M16!) MEMBER- (M11!) NAME (B1)))
(M1011 (LOAD- (M1016 M1157) DURATION (2) IMAGE (130) NAME (W1)
      REQUEST (24) START (7)))
```

(M1012 (LOAD- (M1016 M1157) DURATION (7) IMAGE (150) NAME (W2) REQUEST (32) START (13)))
(M1013 (LOAD- (M1016 M1157) DURATION (5) IMAGE (110) NAME (W3) REQUEST (12) START (11)))
(M1014 (LOAD- (M1016 M1157) DURATION (8) IMAGE (90) NAME (W4) REQUEST (24) START (12)))
(M1015 (LOAD- (M1016) DURATION (4) IMAGE (150) NAME (W5) REQUEST (4) START (14)))
(M1016 (INDIVIDUAL- (M1017!) LOAD (M1011 M1012 M1013 M1014 M1015)))
(M1017! (CLASS (good) FITNESS (126) INDIVIDUAL (M1016)))
(M1032 (LOAD- (M1037) DURATION (5) IMAGE (130) NAME (W1) REQUEST (8) START (12)))
(M1033 (LOAD- (M1037) DURATION (5) IMAGE (130) NAME (W2) REQUEST (16) START (9)))
(M1034 (LOAD- (M1037) DURATION (1) IMAGE (150) NAME (W3) REQUEST (24) START (7)))
(M1035 (LOAD- (M1037) DURATION (6) IMAGE (130) NAME (W4) REQUEST (8) START (7)))
(M1036 (LOAD- (M1037 M1313) DURATION (7) IMAGE (110) NAME (W5) REQUEST (8) START (14)))
(M1037 (INDIVIDUAL- (M1038!) LOAD (M1032 M1033 M1034 M1035 M1036)))
(M1038! (CLASS (good) FITNESS (130) INDIVIDUAL (M1037)))
(M1065 (LOAD- (M1070 M1316) DURATION (6) IMAGE (150) NAME (W1) REQUEST (8) START (7)))
(M1066 (LOAD- (M1070 M1227 M1316) DURATION (2) IMAGE (130) NAME (W2) REQUEST (16) START (13)))
(M1067 (LOAD- (M1070 M1227 M1316) DURATION (8) IMAGE (130) NAME (W3) REQUEST (20) START (14)))
(M1068 (LOAD- (M1070 M1227 M1316) DURATION (2) IMAGE (130) NAME (W4) REQUEST (32) START (9)))
(M1069 (LOAD- (M1070) DURATION (7) IMAGE (130) NAME (W5) REQUEST (8) START (10)))
(M1070 (INDIVIDUAL- (M1071!) LOAD (M1065 M1066 M1067 M1068 M1069)))
(M1071! (CLASS (good) FITNESS (134) INDIVIDUAL (M1070)))
(M1078 (LOAD- (M1083 M1313) DURATION (2) IMAGE (130) NAME (W1) REQUEST (16) START (14)))
(M1079 (LOAD- (M1083 M1313) DURATION (5) IMAGE (130) NAME (W2) REQUEST (12) START (14)))
(M1080 (LOAD- (M1083 M1313) DURATION (6) IMAGE (150) NAME (W3) REQUEST (20) START (14)))
(M1081 (LOAD- (M1083 M1313) DURATION (7) IMAGE (150) NAME (W4) REQUEST (24) START (9)))
(M1082 (LOAD- (M1083) DURATION (7) IMAGE (110) NAME (W5) REQUEST (24) START (14)))
(M1083 (INDIVIDUAL- (M1084!) LOAD (M1078 M1079 M1080 M1081 M1082)))
(M1084! (CLASS (good) FITNESS (134) INDIVIDUAL (M1083)))
(M11! (CLASS (bridge) MEMBER (M10)))
(M1156 (LOAD- (M1157) DURATION (7) IMAGE (150) NAME (W5) REQUEST (24) START (11)))
(M1157 (INDIVIDUAL- (M1158!) LOAD (M1011 M1012 M1013 M1014 M1156)))
(M1158! (CLASS (good) FITNESS (126) INDIVIDUAL (M1157)))

(M12 (MEMBER- (M13!) NAME (S1 S2 S3)))
(M1225 (LOAD- (M1227) DURATION (4) IMAGE (150) NAME (W1) REQUEST (4) START (10)))
(M1226 (LOAD- (M1227) DURATION (7) IMAGE (150) NAME (W5) REQUEST (8) START (10)))
(M1227 (INDIVIDUAL- (M1228!) LOAD (M1066 M1067 M1068 M1225 M1226)))
(M1228! (CLASS (good) FITNESS (138) INDIVIDUAL (M1227)))
(M13! (CLASS (server) MEMBER (M12)))
(M1313 (INDIVIDUAL- (M1314!) LOAD (M1036 M1078 M1079 M1080 M1081)))
(M1314! (CLASS (good) FITNESS (134) INDIVIDUAL (M1313)))
(M1315 (LOAD- (M1316) DURATION (7) IMAGE (130) NAME (W5) REQUEST (24) START (10)))
(M1316 (INDIVIDUAL- (M1317!) LOAD (M1065 M1066 M1067 M1068 M1315)))
(M1317! (CLASS (good) FITNESS (134) INDIVIDUAL (M1316)))
(M14 (MEMBER- (M15!) NAME (W1 W2 W3 W4 W5)))
(M1408 (ARG- (M1410! M1412! M1418!) ATTRIBUTE (totima) VALUE (large)))
(M1409 (ARG- (M1410! M1414!) ATTRIBUTE (stadis) VALUE (very-long)))
(M1410! (ARG (M1408 M1409)))
(M1411 (ARG- (M1412! M1423!) ATTRIBUTE (stadis) VALUE (fairly-long)))
(M1412! (ARG (M1408 M1411)))
(M1413 (ARG- (M1414!) ATTRIBUTE (totima) VALUE (medium)))
(M1414! (ARG (M1409 M1413)))
(M1415 (ARG- (M1417! M1423! M1425!) ATTRIBUTE (totima) VALUE (any-totima)))
(M1416 (ARG- (M1417! M1418! M1420! M1422!) ATTRIBUTE (stadis) VALUE (any-stadis)))
(M1417! (ARG (M1415 M1416)))
(M1418! (ARG (M1408 M1416)))
(M1419 (ARG- (M1420!) ATTRIBUTE (totima) VALUE (huge)))
(M1420! (ARG (M1416 M1419)))
(M1421 (ARG- (M1422!) ATTRIBUTE (totima) VALUE (small)))
(M1422! (ARG (M1416 M1421)))
(M1423! (ARG (M1411 M1415)))
(M1424 (ARG- (M1425!) ATTRIBUTE (stadis) VALUE (short)))
(M1425! (ARG (M1415 M1424)))
(M15! (CLASS (workstation) MEMBER (M14)))
(M156 (LOAD- (M161 M737) DURATION (3) IMAGE (150) NAME (W1) REQUEST (24) START (14)))
(M157 (LOAD- (M161 M737) DURATION (3) IMAGE (130) NAME (W2) REQUEST (4) START (10)))
(M158 (LOAD- (M161) DURATION (5) IMAGE (130) NAME (W3) REQUEST (12) START (11)))
(M159 (LOAD- (M161 M737) DURATION (6) IMAGE (150) NAME (W4) REQUEST (24) START (12)))
(M16! (COMPONENT (M8) CONNECTOR (M10)))
(M160 (LOAD- (M161 M737) DURATION (8) IMAGE (50) NAME (W5) REQUEST (20) START (13)))
(M161 (INDIVIDUAL- (M162!) LOAD (M156 M157 M158 M159 M160)))
(M162! (CLASS (good) FITNESS (122) INDIVIDUAL (M161)))
(M17 (COMPONENT- (M19!) NAME (S1 W1 W2 W3)))
(M18 (CONNECTOR- (M19!) NAME (R1)))
(M19! (COMPONENT (M17) CONNECTOR (M18)))

(M2 (DURATION- (M5!) GROW (inc) LENGTH (3) LOW (1) POSITION (2) STEP (1) UNIT (hour) UP (8)))
(M20 (COMPONENT- (M22!) NAME (C1 S2 S3 W4 W5)))
(M21 (CONNECTOR- (M22!) NAME (R2)))
(M22! (COMPONENT (M20) CONNECTOR (M21)))
(M235 (LOAD- (M240) DURATION (8) IMAGE (10) NAME (W1) REQUEST (16) START (10)))
(M236 (LOAD- (M240) DURATION (2) IMAGE (10) NAME (W2) REQUEST (28) START (7)))
(M237 (LOAD- (M240) DURATION (4) IMAGE (50) NAME (W3) REQUEST (28) START (11)))
(M238 (LOAD- (M240) DURATION (8) IMAGE (70) NAME (W4) REQUEST (32) START (11)))
(M239 (LOAD- (M240) DURATION (7) IMAGE (10) NAME (W5) REQUEST (28) START (12)))
(M240 (INDIVIDUAL- (M241!) LOAD (M235 M236 M237 M238 M239)))
(M241! (CLASS (bad) FITNESS (30) INDIVIDUAL (M240)))
(M3 (REQUEST- (M5!) GROW (inc) LENGTH (3) LOW (4) POSITION (3) STEP (4) UNIT (number) UP (32)))
(M4 (IMAGE- (M5!) GROW (inc) LENGTH (3) LOW (10) POSITION (4) STEP (20) UNIT (MB) UP (150)))
(M5! (DURATION (M2) IMAGE (M4) REQUEST (M3) START (M1) TUPLENGTH (12)))
(M6! (CLASS (any-totima) ATTRIBUTE (totima) MEMBER (huge large medium small)))
(M7! (CLASS (any-stadis) ATTRIBUTE (stadis) MEMBER (fairly-long short very-long)))
(M736 (LOAD- (M737) DURATION (5) IMAGE (150) NAME (W3) REQUEST (12) START (11)))
(M737 (INDIVIDUAL- (M738!) LOAD (M156 M157 M159 M160 M736)))
(M738! (CLASS (good) FITNESS (126) INDIVIDUAL (M737)))
(M8 (COMPONENT- (M16!) MEMBER- (M9!) NAME (R1 R2)))
(M9! (CLASS (ring) MEMBER (M8)))
(MB (UNIT- (M4)))
(R1 (NAME- (M18 M8)))
(R2 (NAME- (M21 M8)))
(S1 (NAME- (M12 M17)))
(S2 (NAME- (M12 M20)))
(S3 (NAME- (M12 M20)))
(W1 (NAME- (M1011 M1032 M1065 M1078 M1225 M14 M156 M17 M235)))
(W2 (NAME- (M1012 M1033 M1066 M1079 M14 M157 M17 M236)))
(W3 (NAME- (M1013 M1034 M1067 M1080 M14 M158 M17 M237 M736)))
(W4 (NAME- (M1014 M1035 M1068 M1081 M14 M159 M20 M238)))
(W5 (NAME- (M1015 M1036 M1069 M1082 M1156 M1226 M1315 M14 M160 M20 M239)))
(any-stadis (CLASS- (M7!) VALUE- (M1416)))
(any-totima (CLASS- (M6!) VALUE- (M1415)))
(bad (CLASS- (M241!)))
(bridge (CLASS- (M11!)))
(fairly-long (MEMBER- (M7!) VALUE- (M1411)))
(good (CLASS- (M1017! M1038! M1071! M1084! M1158! M1228! M1314! M1317! M162! M738!)))
(hour (UNIT- (M1 M2)))

```
(huge (MEMBER- (M6!) VALUE- (M1419)))
(inc (GROW- (M1 M2 M3 M4)))
(large (MEMBER- (M6!) VALUE- (M1408)))
(medium (MEMBER- (M6!) VALUE- (M1413)))
(number (UNIT- (M3)))
(ring (CLASS- (M9!)))
(server (CLASS- (M13!)))
(short (MEMBER- (M7!) VALUE- (M1424)))
(small (MEMBER- (M6!) VALUE- (M1421)))
(stadis (ATTRIBUTE- (M1409 M1411 M1416 M1424 M7!)))
(totima (ATTRIBUTE- (M1408 M1413 M1415 M1419 M1421 M6!)))
(very-long (MEMBER- (M7!) VALUE- (M1409)))
(workstation (CLASS- (M15!)))
(1 10 11 110 12 122 126 13 130 134 138 14 150 16 2 20 24 28 3 30 32 4 5 50 6 7 70 8 9
90 B1 C1 M1 M10 M1011 M1012 M1013 M1014 M1015 M1016 M1017! M1032 M1033
M1034 M1035 M1036 M1037 M1038! M1065 M1066 M1067 M1068 M1069 M1070
M1071! M1078 M1079 M1080 M1081 M1082 M1083 M1084! M11! M1156 M1157
M1158! M12 M1225 M1226 M1227 M1228! M13! M1313 M1314! M1315 M1316
M1317! M14 M1408 M1409 M1410! M1411 M1412! M1413 M1414! M1415 M1416
M1417! M1418! M1419 M1420! M1421 M1422! M1423! M1424 M1425! M15! M156
M157 M158 M159 M16! M160 M161 M162! M17 M18 M19! M2 M20 M21 M22! M235
M236 M237 M238 M239 M240 M241! M3 M4 M5! M6! M7! M736 M737 M738! M8
M9! MB R1 R2 S1 S2 S3 W1 W2 W3 W4 W5 any-stadis any-totima bad bridge fairly-
long good hour huge inc large medium number ring server short small stadis totima very-
long workstation)

 CPU time : 5.60    GC time : 0.00


* (lisp)
"End of SNePS"


<cl> (exit)
; Exiting Lisp
```

Generally, a user may take the following steps to run the prototype system:

(1) At UNIX shell prompt, e.g., "=>", type command ***sneps-ga***, which loads SNePS and OOGA systems at the top of Common Lisp.

(2) At prompt "<cl>", type command ***(sneps)***, which brings a user into the SNePS read-eval-print loop.

(3) At prompt "*", type command ***^(atnin "driver.atn")***, which loads the prototype system's driver, user interface, and other modules, included in the GATN grammar file called driver.atn.

(4) At prompt "*", type command ***^(parse)***, which enters the natural language interface of the prototype system.

(5) At prompt ":", the user describes a task system configuration in a limited subset of English (see section 4.1). The prototype system gives responses like "I understand that ..." and semantic networks representing this configuration are built into knowledge base.

(6) At prompt ":", the user asks question ***what is a maximum workload?***, then the prototype system starts doing a genetic search. Along the search goes, individuals (shown on the left-side) and their fitness values (shown on the right-side) are presented to the user. At the end of the genetic search, a description of one near-maximum workload comes out.

(7) If at this point the user likes to check how many experiences or what kind of experiences have been accumulated in knowledge base, then at prompt ":", type ***^end***; at prompt "*", type ***^*good-nod-set****; at prompt "*", type ***^*bad-nod-set****; and etc. Then at prompt "*", type ***^(parse)*** to go back to the natural language interface.

(8) At prompt ":", the user asks question ***what is a general rule?***, then the prototype system starts doing MH-transformation and generalization. At the end of the generalization, a rule comes out in propositional calculus of attribute-value pairs and in English.

(9) If the user likes to see what has been done by MH-transformation, then at prompt ":", type ***^end***; at prompt "*", type ***^*positive-instances****; and at prompt "*", type ***^*negative-instances****.

(10) If the user likes to see what kind of semantic networks has been built into knowledge base, at prompt "*", type ***(dump *nodes)***.

(11) For quit, at prompt "*", type ***(lisp)***; and at prompt "<cl>", type ***(exit)***.

References

[Branchman and Levesque, 1985] R. J. Branchman and H. J. Levesque (Ed.). *Readings in knowledge representation.* Morgan Kaufmann, 1985.

[Carbonell, 1989] J. G. Carbonell. Introduction: Paradigms for machine learning. *Artificial intelligence*, 40, 1-10, 1989.

[Cohen and Feigenbaum, 1982] P. R. Cohen and E. A. Feigenbaum. *The handbook of artificial intelligence, Vol. 3.* Addison-Wesley, 1982.

[Davis, 1987] L. Davis (Ed.). *Genetic algorithms and simulated annealing.* London: Pitman, 1987.

[Davis, 1991] L. Davis (Ed.). *Handbook of genetic algorithms.* Van Nostrand Reinhold, 1991.

[DeJong, 1988] K. DeJong. Learning with genetic algorithms: An overview. *Machine learning*, 3, 121-138, 1988.

[Franz Inc, 1988] Franz Inc. *Common LISP: The reference.* Addison-Wesley, 1988.

[Goldberg, 1989] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989.

[Gordon and Grefenstette, 1990] D. F. Gordon and J. J. Grefenstette. Explanations of empirically derived reactive plans. *Proceedings of the seventh international conference on machine learning,* 198-203, 1990.

[Grefenstette, 1986] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE transactions on systems, man, and cybernetics*, SMC-16(1), 122-128, 1986.

[Grefenstette, 1988] J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine learning*, 3, 101-120, 1988.

[Grefenstette et al., 1990] J. J. Grefenstette, C. L. Ramsey, and A. Schultz. Learning sequential decision rules using simulation models and competition. *Machine learning*, 5, 355-381, 1990.

[Grefenstette et al., 1991] J. J. Grefenstette, L. Davis, and D. Cerys. *GENESIS and OOGA: Two genetic algorithm systems.* TSP, P.O Box 991, Melrose, MA 02176, 1991.

[Hirsh, 1990] H. Hirsh. *Incremental version-space merging: A general framework for concept learning.* Kluwer Academic, 1990.

[Holland, 1975] J. H. Holland. *Adaptation in natural and artificial systems.* University Michigan Press, Ann Arbor, 1975.

[Holland, 1986] J. H. Holland. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. *Machine learning: An artificial intelligence approach, Vol. 2.* Morgan Kaufmann, 1986.

[Keene, 1989] S. E. Keene. *Object-oriented programming in Common Lisp: A programmer's Guide to CLOS.* Addison-Wesley, 1989.

[Mitchell et al., 1982] T. M. Mitchell, P. E. Utgoff, and R. B. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. *Machine learning: An artificial intelligence approach, Vol. 1.* Morgan Kaufmann, 1982.

[Mitchell, 1982] T. M. Mitchell. Generalization as search. *Artificial intelligence*, 18, 203-226, 1982.

[Pavicic and Ding, 1991] M. Pavicic and Y. Ding. Multicomputer performance evaluation tool and its application to the Mayo/IBM image archival system. *Medical imaging V, SPIE Proceedings Vol.1446,* 1991.

[Persons et al., 1990] K. Persons, A. Benson, M. Pavicic, and Y. Ding. Performance characteristics and model for the Mayo/IBM PACS project. *SPIE medical imaging IV,* 1990.

[Powell et al., 1989] D. J. Powell, S. S. Tong, and M. M. Skolnick. EnGENEous domain independent, machine learning for design optimization. *Proceedings of the third international conference on genetic algorithms*, 151-159, 1989.

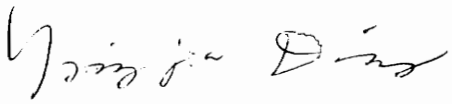
[Shapiro and Group, 1989] S. C. Shapiro and the SNePS implementation group. *SNePS-2.1 user's manual*, 1989.

[Shapiro and Martins, 1989] S. C. Shapiro and J. P. Martins. Recent advances and developments - The SNePS 2.1 report. *Current Trends in SNePS - Semantic network processing system,* LNAI 437, 1-13, 1989.

[Zhou, 1990] H. H. Zhou. CSM: A computational model of cumulative learning. *Machine learning*, 5, 383-406, 1990.

Vita

Yingjia Ding was born in Zhejiang, China on April 26, 1960. She received her B.S. and M.S. degrees in Computer Science and Engineering from Nanjing Institute of Technology (NIT) in China in 1982 and 1984 respectively. She joined NIT as a lecturer from 1985 to 1987. She worked as a research associate in the department of Computer Science at North Dakota State University (NDSU) for a joint research project of IBM and NDSU during October, 1989 - August, 1990.



Yingjia Ding