REES: Reasoning Engine(s) Evaluation Shell

Radu Marinescu, Kalev Kask and Rina Dechter

School of Information and Computer Science
University of California, Irvine
{radum,kkask,dechter@ics.uci.edu}

Abstract

We introduce a software environment to support research and development in the area of both deterministic and non-deterministic reasoning. This environment - REES (Reasoning Engine(s) Evaluation Shell) has a plug-in oriented architecture that promotes reuse of existing software components and allows for the comparison and evaluation of alternative technologies. The third release of the REES system and technical documentation is now available at *www.ics.uci.edu/∼radum/rees.html*.

1 Introduction

In a typical application, a design is implemented that meets the set of requirements at the time of development. Often, after a program is delivered, the user will want added functionality, or different users will require custom functionality based on their specific needs. In order to accommodate these situations without a complete re-write, or causing a develop/compile/test/ship scenario, a framework that allows for future additions of modules without breaking the existing code base needs to be implemented. A Plug-In architecture will meet these needs.

To put it simply, a system using this architecture would be capable of looking for various Plug-In modules when starting up. Once all the Plug-Ins have been located they are loaded by the main application one by one, or selectively so as to use their built-in features. These Plug-Ins are normally DLLs (Dynamic Linked Library) in disguise and many comercial applications, even the Windows operating system, currently use similar technologies to allow third-party developers to integrate with their existing application to add functionality or robustness, otherwise missing from the application.

The REES system was purposely designed in this manner. The main reason behind this is that different research groups in the community usually develop their own libraries of algorithms and in most cases they are incompatible with each other, thus making a joint comparison and evaluation practically impossible. REES provides a common interface that promotes reuse of already existing components and allows for comparison and evaluation of alternative technologies, while using a common workbench.

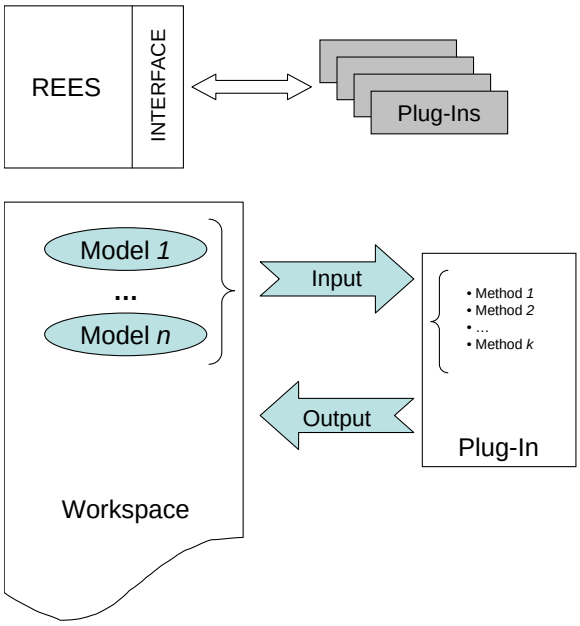


Figure 1: REES Plug-In Architecture

2 REES Architecture

The architecture of REES system is described in Figure 1. Constraint based or probabilistic reasoning problems are locally defined and/or loaded into the main workspace and transferred to the available Plug-Ins for processing. The results produced by the inference algorithms residing in various Plug-Ins are passed back to the REES main workspace for further refining and appropriate display. The existence of a predetermined interface, implemented by each Plug-In, facilitates easy and complete communication between them and REES. We will now discuss the main components of the proposed architecture: *Workspace, Model, Plug-In modules*.

2.1 Workspace and Models

The **Workspace** is the main component of the system. It encapsulates all the problem models defined by the user and available for evaluation, as well as the list of currently loaded Plug-Ins. Using the graphical interface, one has the posibility of defining new problem models, modifying existing ones or selectively loading/unloading Plug-In modules for additional functionality.
A **Model** is an abstract representation of a reasoning problem. Within the framework, such a problem instance may be represented either in parametric form (e.g. we use the well known **(N, K, C, T)** parametric model representation) or as a completely defined instance in terms of variables, domain sizes and relationships between variables (i.e. functions). Depending on the chosen model representation (paramatric or complete),

the graphical interface assits the user in further refining the model. Options as modifing the values of some parameters (parametric model) or altering the graph structure of the network (complete model) are also available.

Together with the problem structure (i.e. constraint/belief network) a list of processing algorithms must also be defined. These inference algorithms may all reside in a single Plug-In library, but in the common case they may be part of different Plug-Ins. The list of selected algorithms together with their control parameters form the *experiment* associated with the problem model. In this way, reasoning algorithms developed within different research groups can be executed and evaluated altogether on the very same problem instances or benchmarks.

Once a problem model has been completely defined in the current workspace, the common interface takes care of creating an object that is *understood* and can be transferred to any attached Plug-In. This sub-process is called *random problem generation* and in both cases it creates a complete problem instance. In case of a parametric model representation, the parameters completely define the graph structure and the functions of the problem. In the other case, there is no need for a problem instance generation and the already existing object can be passed along, as is.

2.2 Plug-Ins

A **Plug-In** is an external module (a DLL in our framework) that implements some functionality. Once installed, it can be loaded at runtime by the main application (REES) to use the functionality provided using exported functions/classes within the DLL. All the Plug-In modules must conform to a pre-defined interface (see Figure 1). The reason for that is determined by the fact that a call to a function residing inside the Plug-In can be issued only after knowing the function name.

As it is defined in this framework, a Plug-In library implements a collection of deterministic and/or non-deterministic reasoning algorithms. A pre-defined header structure ensures the compatibility with the main application (REES). In our implementation, a Plug-In must export the list of implemented algorithms together with their input/output control parameters as well as the list of functions that form the common interface.

3 A Closer Look

This section describes in more detail the main features of the REES environment and shows the basic steps of the entire process, from model creation to experimentation to viewing and interpreting the results. REES provides an easy to use graphical interface that allows intuitive creation/editing of the problem model, direct adjustment of the control parameters for all algorithms involved in some experiment as well as user friendly display of the results produced by the experiments. REES also provides support for saving either the entire workspace or individual models to a file for later use.

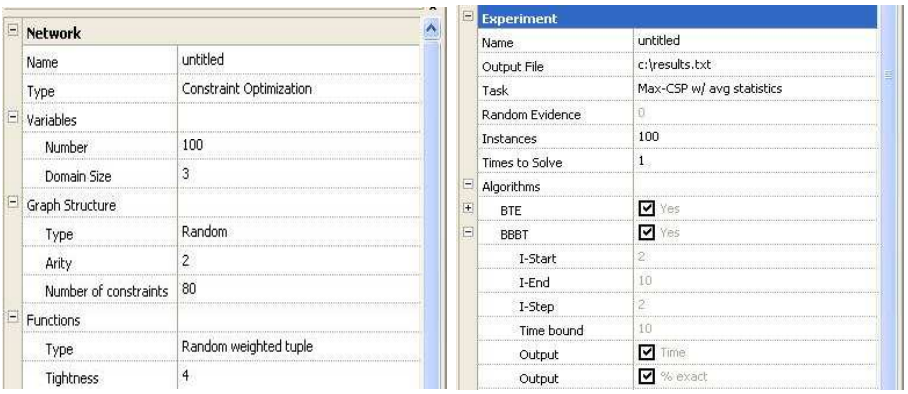


Figure 2: REES Graphical Interface. (a) Model. (b) Experiment

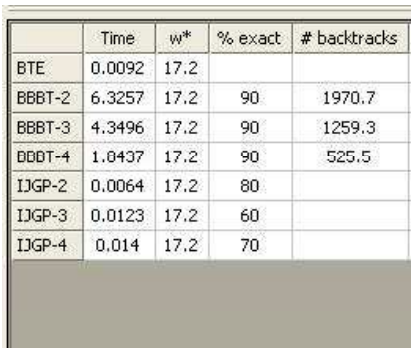
3.1 Model Definition

The first step in any deterministic/probabilistic reasonig problem is defining the problem model. To create a new model, simply select *Add Model* from the main menu or select the appropriate icon from the toolbar. REES will then begin the process of helping the user to create the model. The underlying network of the model (either a constraint network for deterministic reasoning problems or a belief network for probabilistic reasoning problems) can be specified in two ways:

1. *Parametric Form.* In this case, a parametric model of the problem is created. The user has the options of specifying the size of the network as number of variables and domain size, the type of graph structure to be generated (e.g. *random, grid, etc.*) as well as the type of functions to be defined over subsets of variables (e.g. *random weighted tuple*). Figure 2(a) displays an example of model definition using REES Graphical Interface. At any time, the system allows the user to modify the values of all these parameters. Later, the *random problem generator* will use all these user defined parameters to build a complete model, as described in Section 2.1.

2. *Complete Form.* In this case, the entire model, as represented by the graph structure and functions defined over subsets of variables, resides in a text file that will be loaded into the workspace. At this moment, REES system is able to parse several file formats (e.g. DIMACS), as well as a proprietary format that may contain additional information for a graphical display of the network[1]. REES Net Editor provides easy graphical editing of constraint/belief networks including cut/paste/duplicate nodes and edges. All these options and many others are available from the main menu and toolbar of the application. In this way, the user is offered the possibility of creating its own model either from scratch or modifying an existing one.

[1]REES Net Editor is currently available only for belief networks.



	Time	w*	% exact	# backtracks
BTE	0.0092	17.2		
BBBT-2	6.3257	17.2	90	1970.7
BBBT-3	4.3496	17.2	90	1259.3
BBBT-4	1.0437	17.2	90	525.5
IJGP-2	0.0064	17.2	80	
IJGP-3	0.0123	17.2	60	
IJGP-4	0.014	17.2	70	

Figure 3: REES Results Display Window.

The model definition is completed once an experiment is defined and attached to it. Details on how to do it can be followed in the next section.

3.2 Running Experiments

Once a problem model is created, the knowledge it contains can be transferred to the available Plug-Ins, each one of them implementing a set of inference algorithms as described in Section 2.2. To create a new experiment, using the current problem model, simply select *New Experiment* from the main menu or select the appropriate icon from the toolbar. A REES wizard will then assist the user in the process of creating the experiment. A tipical experiment must specify the *task* it will perform, the number of problem *instances* to be generated as well as the set of *algorithms* together with their control parameters to be executed. Figure 2(b) shows an example of an experiment defined on a constraint-based model.

1. *Task*: Depending on the problem model on which the experiment is defined, several tasks may be available (e.g. *Max-CSP, Solution Counting, etc.* for constraint-based models, *Belief, Most Probable Explanation, etc.* for probabilistic models). Each algorithm exported by a Plug-In must have its header information containing the task type it is able to perform.

2. *Instances*: If there is no random behavior specified for this particular model (i.e. *complete model*), then there can only be one instance of the underlying network. If there is either a random structure definition and/or a random function definition (i.e. *parametric model*) then REES can create as many problem instances as indicated by the parameter value.

3. *Algorithms*: Each algorithm exported by some Plug-In library has a set of control parameters associated with. The user must set values for all *input* parameters (if there are any) and may select one or more *output* parameters for visualization. After the execution of the experiment has sucessfully completed, the average values of the output parameters will be displayed for further analysis.

Once the experiment is created, REES can be instructed to execute it. A detailed log of the execution can also be recorded so as the user to be able to abort the experiment once an error is signaled. The results produced by an experiment that completed sucessfully are displayed in a spreadsheet, each column representing one of the selected output parameters. This should make comparison between algorithms quite simple and intuitive, where such a comparison is appropriate.

In Figure 3 we provide an example of results produced by an *MPE* experiment, that is finding the *Most Probable Explanation* in Bayesian models. The problem was represented as a parametric model that generated 10 random instances of a binary belief network with 100 variables and 90 conditional probability tables. Three algorithms were chosen for evaluation: **BTE** (i.e. *Bucket Tree Elimination*), an exact inference algorithms based on the well known variable elimination mechanism, **BBBT(**i**)** (i.e. *Branch and Bound with mini-Bucket Tree* heuristics) a complete Branch and Bound search algorithm that uses dynamic heuristics generated by a Mini-Bucket Tree Elimination algorithm to guide the search and **IJGP(**i**)** (i.e. *Iterative Join Graph Propagation*) an iterative version of graph propagation algorithms. The latter two algorithms are controlled by a parameter called i-bound. For each algorithm, REES displays the average values of the selected output parameters, columnwise. They are: average running time (*Time*), average induced width of the problem (*w**), average accuracy as percent of exaclty solved instances (*% exact*) as well as the average number of backtracks for the branch and bound search algorithm (*# backtracks*).

4 Conclusions

We have described the REES system as a powerful, easy-to-use, complete application for working with deterministic and non-deterministic reasoning problems. Its plug-in oriented arhitecture allows one to experiment with already existing inference algorithms as well as to extend the currently existing collection of algorithms with new ones. When developing a new algorithm it is very important to rapidly evaluate and compare its performance against alternative approaches. We think that promoting and contributing to the REES system would offer a common workbench and ease collaboration among different research groups in the automated reasoning community.

Acknowledgements

This work was supported in part by the NSF grant IIS-0086529 and MURI ONR award N00014-00-1-0617.