UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

DIOGO CESAR BIF

ANÁLISE DE DESEMPENHO DA SHELL PEGASUS UNCERTAINITY MODELING

CRICIÚMA, JUNHO DE 2010

DIOGO CESAR BIF

ANÁLISE DE DESEMPENHO DA SHELL PEGASUS UNCERTAINITY MODELING

Trabalho de Conclusão de Curso para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense

Orientador: Prof. MSc. Kristian Madeira

CRICIÚMA, JUNHO DE 2010

DIOGO CESAR BIF

Análise de desempenho da Shell Pegasus Uncertainity Modeling

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Profa. MSc. Ana Claudia Garcia Barbosa
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

Prof. MSc. Kristian Madeira (UNESC)
Orientador

Profª. MSc. Priscyla Waleska Targino de Azevedo Simões (UNESC)

Prof. MSc. Paulo João Martins (UNESC)

A todos que assim como eu acreditaram que seria possível.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, sem os quais com certeza não conseguiria finalizar essa corrida.

Os meus mais sinceros votos de agradecimento e sucesso aos professores que me auxiliaram nesse trabalho, em especial ao professor Kristian e a professora Priscyla, que me guiaram durante o desenvolvimento e me mostraram como fazer.

Meus sinceros votos também ao professor Paulo e a professora Ana que me auxiliaram no desenvolvimento de minha pesquisa.

"A preguiça caminha tão devagar, que a pobreza não tem dificuldade em a alcançar."

(Confúcio)

RESUMO

Desde o início da aplicação de máquinas para a resolução de problemas complexos a Ciência da Computação começou a criar áreas para se adaptar as novas necessidades. Uma dessas áreas foi a Inteligência Artificial, que tinha como intuito criar entidades inteligentes para a resolução de problemas. Com o crescimento da complexidade desses problemas, a Inteligência Artificial teve que lidar com situações cujos dados enviados para as máquinas não eram completos ou confiáveis, abrindo o campo para a Modelagem da Incerteza. Nessa direção, percebe-se que diferentes problemas podem lidar com tipos de dados distintos, o que acaba por suscitar a existência de várias técnicas para a Modelagem de Incerteza, entre as quais destaco as mais frequentemente utilizadas: Lógica Difusa, Redes Bayesianas, Teoria da Evidência de Dempster-Shafer e Teoria dos Fatores de Certeza. Essas teorias envolvem grande carga matemática, além de revelarem regras complexas para implementação, o que provocou o surgimento de softwares com a finalidade de auxiliar no desenvolvimento e resolução de problemas que envolvem a incerteza, esses programas são chamados *Shells*. De outro lado, com a expansão e aumento da demanda de softwares pela população em geral, a necessidade de se desenvolver softwares mais ágeis e que tivessem seus resultados fornecidos de modo seguro e sem erro fez com que a Ciência da Computação criasse uma área para desenvolver e estudar metodologias de desenvolvimento e promover também processos que garantissem, através de testes, a qualidade do software desenvolvido, a essa área denominou-se Engenharia de Software. A proposta dessa pesquisa foi a de avaliar o desempenho da Shell Pegasus Uncertainty Modeling, desenvolvida na Universidade do Extremo Sul Catarinense (UNESC), submetendo-a a uma série de testes de qualidade e agilidade, afim de se criar um relatório e divulgá-la com mais amplitude e confiabilidade dentro da comunidade científica. A metodologia dos testes baseou-se na captura e análise dos tempos de execução das inferências nos módulos da Shell. Para executar tal captura foi utilizada a ferramenta de análise de desempenho JProfiler, a qual auxilia na observação da execução de softwares, mostrando os tempos de execução dos processos, a quantidade de memória utilizada em cada processo e as classes e funções utilizadas. Após a captura dos tempos de execução, esses tempos foram inseridos em um gráfico de dispersão, a fim de se analisar sua oscilação. Logo após foram criadas médias de cada módulo, as quais também foram inseridas em um gráfico de dispersão aplicando os formalismos de análise de regressão (linear e polinomial) e de coeficiente de determinação (r^2), a fim de se analisar sua taxa de crescimento. Baseando-se nos tempos de execução, foi feita uma pesquisa sobre algoritmos, citanda suas classes e níveis de complexidade. Utilizando-se dos tempos capturados, chegou-se a conclusão que a *SHELL* cresce de modo linear, com um algoritmo O(n), o qual é o terceiro em nível de complexidade e é um dos algoritmos com maior eficiência.

Palavras chave: Engenharia de software, Modelagem de Incerteza, Teste de Software, Análise de Desempenho, Shell Pegasus Uncertainity Modeling.

ABSTRACT

Since the early application of machines to solve complex problems, the computer science began to create areas to adapt to new needs. One such area was the AI, which had the intention to create intelligent entities to solve problems. With the increasing complexity of these problems, the AI had to deal with situations for which data sent to the machines were not complete or accurate, opening the field for the Modelling of Uncertainty. In this direction, we realize that different problems can handle different data types, which ultimately lead to the existence of various techniques for modeling uncertainty, among which I highlight the most frequently used: Fuzzy Logic, Bayesian Networks, Dempster-Shafer Evidence Theory and Certainty Factors. These theories involve large load math, and reveal complex rules for implementation, which prompted the development of software for the purpose of assisting in the development and solving problems that involve uncertainty, these programs are called Shells. On the other hand, with the expansion and increased demand for software in general population, the need to develop software more flexible and that its results had provided safely and without error caused the Computer Science created an area to develop and study development methodologies and also promote processes that would ensure, through testing, the quality of software developed, this area was called Software Engineering. The purpose of this study was to evaluate the performance of Shell Pegasus Uncertainty Modeling, developed at the University of Southern Santa Catarina (UNESC), subjecting it to a series of quality tests and agility in order to create a report and disseminate it with more breadth and reliability within the scientific community. The methodology of the tests was based on capturing and analyzing the execution times of the inferences in the modules from Shell. To run this tool was used to capture performance analysis JProfiler which helps in observing the implementation of software, showing the execution times of processes, the amount of memory used in each case, the classes and functions used. After the capture of execution times, these times were entered into a scatter chart, in order to analyze their swing. They were soon established means of each module, which was inserted in a scatter plot by applying the formalism of regression analysis (linear and polynomial) and coefficient of determination (r^2) in order to analyze its growth rate. Based on the execution times, was made a research on algorithms, citanda their classes and levels of complexity. Using captured the times, we reached the conclusion that Shell grows in linear fashion, an algorithm with O (n), which is the third level of complexity and is one of the algorithms more efficiently.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IA Inteligência Artificial
MC Medida de Crença
MD Medida de Descrença
MTBF Mean Time Between Failure
RB - Redes Bayesianas
TDS Teoria da Evidência de Dempster-Shafer
UNESC Universidade do Extremo Sul Catarinense

SUMÁRIO

1. INTRODUÇÃO

A Inteligência Artificial (IA) é um campo que tenta construir entidades inteligentes, sistematizando e automatizando tarefas intelectuais. (RUSSEL; NORVIG, 2004).

Conforme visto nos últimos anos, é alto o valor que o conhecimento específico tem na resolução de problemas complexos, pois para resolvê-los os computadores necessitam do conhecimento que os desenvolvedores humanos têm sobre o problema Os problemas de IA abrangem um espectro muito amplo e aparentemente têm muito pouco em comum. Para resolver diversos desses problemas, a IA possui técnicas que exploram o conhecimento que deve ser representado de tal forma que capture generalizações, agrupando situações com características comuns e que seja compreendido pelas pessoas que fornecem esse conhecimento (REZENDE, 2005).

A IA costuma contribuir em diferentes áreas de aplicação quando existem problemas complexos que exigem grande quantidade de conhecimento e raciocínio.

Para auxiliar na resolução desses problemas, muitas vezes são desenvolvidos os denominados Sistemas Inteligentes, que geralmente buscam resolver problemas de domínios específicos.

Para auxiliar no desenvolvimento desses tipos de sistemas, foram criadas ferramentas denominadas *Shells*. Essas ferramentas geralmente oferecem uma interface gráfica para facilitar a implementação das técnicas de IA, fornecendo assim um ambiente onde se pode aplicar variados algoritmos, os quais ajudam na resolução de problemas como a modelagem da incerteza.

Na Universidade do Extremo Sul Catarinense (UNESC) encontra-se em desenvolvimento pelo Grupo de Pesquisa em Inteligência Computacional Aplicada a

Shell Pegasus Uncertainity Modeling, que busca apresentar em um único ambiente vários formalismos utilizados na modelagem da incerteza. Esta *Shell* busca auxiliar na solução de problemas incompletos ou inconsistentes, e já oferece os seguintes algoritmos: Teoria da Evidência de Dempster-Shafer (TDS) pelo algoritmo clássico (SILVA, 2008), Redes Bayesianas pelo algoritmo de passagem de mensagens de Judea Pearl (ROCHA, 2008) e Fatores de Certeza pelo algoritmo de Buchanan (GONÇALVES, 2008).

Cada um dos algoritmos de IA implementados na *Shell* Pegasus Uncertainity Modeling têm por função facilitar a resolução de determinada categoria de problema com incerteza associada. Assim, os algoritmos utilizados neste contexto tentam auxiliar no entendimento e no processamento das informações para o sistema, fazendo assim com que o acerto seja maior, a interpretação dos dados não seja errônea e o entendimento do problema e de suas variáveis seja facilitado.

Devido ao fato de utilizarem o conceito de probabilidade condicional, as Redes Bayesianas (RB) sugerem um mecanismo unidirecional para representação de modelos causais, ou seja, baseados na relação causa-efeito. Esse tipo de relação é amplamente utilizada em várias áreas da ciência, por exemplo, um médico utiliza desse modelo para seu diagnóstico pois a partir de sintomas o diagnóstico é estabelecido como hipótese com maior valor de probabilidade associado. As Redes Bayesianas vêm sendo aplicadas em problemas de diversas áreas do conhecimento, como diagnóstico médico, aprendizado de mapas, interpretação de linguagem e visão computacional.

Assim como as Redes Bayesianas, a Teoria dos Fatores de Certeza tem por função auxiliar na resolução de incertezas apresentadas dentro de um problema. Ela se utiliza de valores que variam de 0 até 1 para quantificar o grau de força que uma

hipótese pode ter. O Fator de Certeza é dividido em duas partes: Medida de Crença (MC) e Medida de Descrença (MD).

A TDS utiliza graus de crença com intervalos de probabilidade para representar o conhecimento incerto, e ao contrário de calcular a probabilidade de uma proposição, estima a probabilidade de uma evidência assumir uma determinada hipótese. TDS visa tratar as informações por meio de formalismos matemáticos, lidando de forma concreta com o problema da ignorância.

O meio com que a TDS trata a informação é muito parecido com a forma com que o cérebro de uma pessoa processa uma informação, o que ajuda muito na inserção de novas crenças dentro do sistema.

Para que os sistemas computacionais baseados na IA tenham maior aplicabilidade e confiabilidade são utilizadas métricas de avaliação que buscam analisar o desempenho de um sistema na resolução do problema onde este é aplicado.

O processo de medição pode ser caracterizado por cinco atividades: formulação, que é a derivação de medidas e métricas de software adequadas para a representação do software que está sendo considerado; coleta dos dados necessários para derivar as métricas formuladas; análise, por meio do cálculo de métricas e aplicação das ferramentas matemáticas; interpretação dos resultados buscando uma representação com maior qualidade e mais detalhada; e realimentação por meio de recomendações derivadas da interpretação das métricas de produtos transmitidos à equipe de software (PRESSMAN, 2006).

Entre as técnicas de avaliação de desempenho descritas por Pressman (2006), pode-se citar o teste de carga, no qual o sistema é submetido a diversos níveis de carga e em diversas combinações, dentro dos limites do sistema; e o teste de esforço ou

estresse, no qual o sistema é forçado a atingir e depois exceder os limites operacionais. Esses testes têm como objetivo avaliar o comportamento do sistema em situações normais e também onde são requeridos muitos recursos do mesmo.

Considerando o problema apresentado, esta pesquisa busca avaliar o desempenho da *Shell Pegasus Uncertainty Modeling* e verificar se ela tem um desempenho aceitável.

1.1 OBJETIVO GERAL

Avaliar o desempenho da *Shell Pegasus Uncertainty Modeling* desenvolvida na Universidade do Extremo Sul Catarinense.

1.2 OBJETIVOS ESPECÍFICOS

a) Entender sobre os sistemas inteligentes;

b) Compreender os critérios de avaliação de desempenho de software;

c) Abranger as técnicas para modelagem de incerteza utilizadas na *Shell Pegasus*;

d) Aplicar métodos estatísticos para avaliação do desempenho na *Shell Pegasus*;

e) Disponibilizar um protocolo de atividades para a avaliação de desempenho de sistemas inteligentes.

1.3 JUSTIFICATIVA

Muitos dos problemas que profissionais de várias áreas precisam resolver são complexos e necessitam de conhecimentos que normalmente são incertos. Para que se possa encontrar possíveis soluções para esses problemas, podem ser utilizadas técnicas de IA.

Nos diversos domínios de aplicação existentes em que a incerteza é presente, é comum que as informações não estejam claras ou completas, dando margem a várias soluções para o problema. Assim, para que uma determinada aplicação possa disponibilizar um resultado mais confiável, é necessária a modelagem adequada da forma de incerteza apresentada.

Algumas técnicas utilizadas neste contexto buscam reduzir a dúvida gerada entre as hipóteses permitindo uma maior crença do especialista em uma das apresentadas, sendo que, por exemplo, quanto maior o grau de confiança na hipótese, maior é a relevância da mesma reduzindo a possibilidade de múltiplos resultados (COSTA; SIMÕES, 2004).

Dentro dos algoritmos de IA que tratam do contexto de incerteza da informação três formalismos distintos foram abordados na *Shell Pegasus Uncertainty Modeling*: Redes

Bayesianas, Fatores de Certeza e Teoria da Evidência de Dempster-Shafer.

Devido a sua estrutura eficiente para propagação de probabilidades explorando as relações de independência condicional existentes entre os nós do grafo, as Redes Bayesianas possibilitam a representação fatorizada da distribuição de probabilidade conjunta e tornam o processo de inferência em modelos probabilísticos computacionalmente tratáveis.

A modelagem dos Fatores de Certeza faz com que os dados apresentados ao sistema sejam interpretados de maneira mais clara, fazendo com que assim a resposta do processamento desses dados tenha uma maior chance de ser correta e a computabilidade do problema seja facilitada.

A TDS visa o auxílio no aprendizado e no processamento das informações do sistema, pois devido ao fato de ela utilizar formalismos matemáticos para o tratamento da informação acaba ajudando no descobrimento de novas crenças. Outro fator interessante na TDS é que nela o resultado não depende da ordem da apresentação das evidências e nem necessita que estas expressem certeza, o que facilita na adição de novas crenças ao sistema.

Assim, conforme Nassar (2007) descreve, não basta somente desenvolver uma aplicação ou ferramenta baseada na IA, é de fundamental importância que o desempenho dos sistemas inteligentes sejam estatisticamente avaliados considerando que estes pretendem emular o comportamento de um especialista em um domínio de aplicação. Para determinados tipos de sistemas, o software pode fornecer as funções desejadas, mas não satisfazer os requisitos de desempenho, diminuindo assim a produtividade do software ou tornando necessário o processamento maior que o necessário para executá-lo.

A avaliação de desempenho é projetada para testar essa característica do software durante a execução no contexto de um sistema integrado, ocorrendo ao longo de todos os passos do processo de teste. O desempenho de módulos individuais pode ser avaliado à medida que testes são conduzidos, porém o desempenho do sistema como um todo - ou o verdadeiro desempenho do sistema - só pode ser avaliado após a integração plena de todos os seus elementos (PRESSMAN, 2006).

Conforme citado, esse trabalho busca testar a *Shell Pegasus*, comprovando sua capacidade através de testes de desempenho, fazendo assim com que seu significado para a comunidade científica seja detalhado.

1.4 ESTRUTURA DO TRABALHO

O trabalho foi dividido em 11 capítulos seguindo a seqüência e as especificações citadas abaixo:

O primeiro capítulo contém uma introdução, os objetivos gerais e específicos, a justificativa e a estrutura do trabalho.

O capítulo seguinte faz uma introdução à Modelagem de Incerteza dentro da inteligência artificial, citando os quatro algoritmos de modelagem implementados na Shell Pegasus Uncertainity Modeling.

O terceiro capítulo faz uma introdução a Engenharia de Software, definindo os conceitos de qualidade.

O quarto capítulo fala sobre teste e análise de software, explicando os tipos de teste que são feitos afim de verificar a presença de erros num sistema, fala também sobre testes e análise de desempenho.

O quinto capítulo fala sobre as ferramentas de análise de desempenho.

O sexto capítulo fala sobre complexidade de algoritmos.

O sétimo capítulo traz o trabalho desenvolvido, falando sobre as análises e feitas com a *Shell Pesasus*.

O oitavo capítulo traz uma comparação entre os módulos da *Shell* com outras *Shells*

O nono capítulo traz a os trabalhos correlatos

O décimo capítulo traz a conclusão.

O último capítulo apresenta as referências

2. MODELAGEM DE INCERTEZA

Dentro do contexto do processamento de dados para geração de respostas / ações a IA acabou tendo que lidar com uma série de problemas. Um dos problemas mais críticos se refere á imperfeição ou incompletude dos dados com os quais tanto os sistemas inteligentes quantos as máquinas que tem algum tipo de raciocínio programado

têm que processar, a isso costuma-se chamar Incerteza. Modelar a incerteza consiste em criar uma representação dela utilizando uma teoria ou modelo de modelagem. Em seguida costuma-se criar um motor de inferência sobre essa modelagem, para então aplicá-lo dentro do sistema (SALICONE, 2007, tradução nossa).

Devido ao fato de não existir nenhum tipo de teoria ou modelo genérico que trabalhe com os variados tipos de incerteza, a escolha do modelo utilizado dentro do sistema é um grande problema enfrentado dentro do domínio dos Sistemas Inteligentes.

Assim, foram criadas aplicações que se baseiam na modelagem da incerteza de acordo com uma teoria específica. A grande facilidade que as Shells[1] trazem é o fato de oferecerem uma interface, na qual fica mais fácil se desenhar e entender o problema (RUSSEL; NORVIG, 2004).

2.1 TEORIAS DE MODELAGEM DE INCERTEZA

Dentre as várias teorias de modelagem de Incerteza, as que mais são utilizadas pelas comunidades científicas são: Teoria da Evidência de Dempster-Shafer, Lógica Nebulosa (Fuzzy), Redes Bayesianas e Teoria dos Fatores de Certeza.

2.1.1 TEORIA DA EVIDÊNCIA DE DEMPSTER SHAFER

A teoria matemática da evidência foi introduzida por Shafer em 1970 como uma reinterpretação das Inferências Estatística de Dempster. Sua teoria começa com a

[1] Shells são ferramentas que oferecem uma interface gráfica para facilitar a implementação de técnicas de IA em sistemas (RUSSEL; NORVIG , 2004).

idéia de utilizar um número entre 0 e 1 para indicar o nível de confiança a partir de uma proposição baseada em uma evidência (SALICONE, 2007, tradução nossa).

A TDS é uma teoria de evidência matemática, que quando aplicada dentro de um espaço finito e discreto pode ser interpretada como uma generalização da teoria da probabilidade. Uma de suas principais vantagens é o fato de possibilitar o desenho de um modelo que consegue lidar com níveis de precisão da informação variáveis. A TDS oferece uma vantagem perante as outras teorias de modelagem de incerteza por tratar a incerteza e a ignorância de modos diferenciados (SALICONE, 2007, tradução nossa).

Para tratar a incerteza, a TDS utiliza-se de dois artifícios: um conjunto de proposições verdadeiras chamadas de evidências e um conjunto de hipóteses que podem ser falsas o qual é chamado de quadro de discernimento. O estudo da relação entre as hipóteses e o quadro de discernimento é a base de trabalho da TDS (YAGER, 1994).

Segundo Klir (1994), a TDS pode ser descrita sucintamente por meio de três funções básicas: a função de alocação de massa, a função de credibilidade Bel e a função de plausibilidade Pl

A função de alocação de massa tem por objetivo expressar a evidência disponível e relevante suportada por um elemento do quadro de discernimento X (SALICONE, 2007, tradução nossa).

O valor da equação varia no intervalo entre 0 e 1, onde 1 significa que o elemento analisado pertence ao conjunto de discernimento X e 0 significa que o elemento não pertence.

A partir do cálculo de alocação de massa, é possível definir um limite inferior (credibilidade) e superior (plausibilidade) (SALICONE, 2007, tradução nossa).

A principal função da credibilidade (Bel (A)) é indicar em que ponto as informações fornecidas se sustentam, já a função de plausibilidade (P1(A)) mostra até que ponto as informações fornecidas não se contradizem (SALICONE, 2007, tradução nossa).

Devido ao fato de utilizar formalismos matemáticos para representar o conhecimento, e também a capacidade de que a mesma suporta o uso de probabilidades subjetivas ao raciocínio, a TDS é a teoria que mais se assemelha ao raciocínio humano (WU 2003).

2.1.2 LÓGICA FUZZY

A teoria dos conjuntos Fuzzy é uma teoria de modelagem de incerteza a qual trabalha baseada na teoria de conjuntos (NICOLETTI; CAMARGO, 2004).

A qualidade que mais diferencia a lógica Fuzzy (ou Lógiga Difusa) das outras técnicas de modelagem é a possibilidade de se captar conceitos intuitivos, chamados de variáveis linguísticas, dentro de um modelo matemático, tais quais graus de satisfação, conforto entre outros (AGUIAR; OLIVEIRA, 1999).

Os conjuntos são caracterizados por um grupo de objetos distinguíveis que compartilham de alguma característica, fato esse que faz um objeto pertencer ou não a um conjunto (NICOLETTI; CAMARGO, 2004).

Variáveis Linguísticas são variáveis que expressam algum tipo de situação, por exemplo, em outras teorias uma variável utilizada para se representar velocidade seria criada sob a forma de Km/h ou Cm/h, já na teoria de conjuntos Fuzzy estas variáveis possuem valores como rápido ou lento, forte ou fraco, alto ou baixo (WEBER; KLEIN, 2003).

Segundo Aguiar e Oliveira (2004), o campo de aplicação da lógica Fuzzy é virtualmente ilimitado, porém nota-se que a maior aplicabilidade da mesma vem nos sistemas relacionados à área de engenharia. Um pequeno exemplo de sistemas que utilizam lógica Fuzzy são os sistemas de controle econômico de condicionamento de temperatura, controle de máquinas de lavar, controle de freios ABS entre outros.

2.1.3 REDES BAYESIANAS

As Redes Bayesianas (RB) surgiram com o intuito de resolver problemas nos quais é necessária uma análise de mais de um elemento dentro do domínio de conhecimento, ou em situações onde os dados relativos ao problema não se encontram em sua ordem lógica (NEAPOLITAN, 2004).

Uma Rede Bayesiana é um formalismo baseado na teoria dos grafos e na teoria da probabilidade e possibilita a representação gráfica do conhecimento incerto e a propagação de probabilidades em sua estrutura por meio de algoritmos de inferência (CASTILHO; GUTIÉRREZ; HADI,1998).

São redes de conhecimento que para sua representação utilizam grafos direcionados acíclicos, onde os nós representam as variáveis e os arcos representam a

existência de uma dependência probabilística entre os mesmos. Essa dependência é expressa através de probabilidades condicionais[2] (NEAPOLITAN, 2004).

Por explorar as dependências condicionais criadas pelas cadeias de influência, as Redes Bayesianas tornam os modelos computacionais aptos a representar uma larga instância usando pouco espaço, e relata ocasionalmente fazer inferências probabilísticas na mesma (NEAPOLITAN, 2004, tradução nossa).

Uma Rede Bayesiana pode ser construída a partir de uma base de dados utilizando-se de seus algoritmos de aprendizagem ou através do conhecimento de um especialista humano (TARONI, 2006).

Segundo Spiegelhalter e Abrams (2004) tal formalismo tem sido usado principalmente na área do conhecimento médico, por exemplo, onde é usada na detecção de doenças.

As variáveis de uma RB podem ser tanto de valor discreto como de valor contínuo (JENSEN, 2006).

A tarefa mais comum é utilizar a rede bayesiana para inferência probabilística, o que consiste em obter conclusões à medida que novas informações ou evidências são conhecidas (NEAPOLITAN, 2004).

2.1.4 FATORES DE CERTEZA

[2] Segundo Moura, et al (1986) a probabilidade condicional é a probabilidade de um evento ocorrer dado que outro evento tenha acontecido

Segundo Costa e Simões (2004), a Teoria dos Fatores de Certeza (TFC) foi desenvolvida para um sistema de diagnóstico médico chamado MYCIN.

A TFC tem uma abordagem parecida com a das Redes Bayesianas, pois também quantifica a incerteza através de um único valor numérico chamado Fator de Certeza, o qual quantifica o grau de certeza que depositamos em tal afirmação ou regra (COSTA; SIMÕES, 2004).

Assim como nas redes Bayesianas, a teoria dos Fatores de certeza também atribui valores numéricos para indicar confiança em uma hipótese. Diferentemente das outras teorias, este formalismo utiliza seus valores para demonstrar quão confiante é uma hipótese dada à ocorrência de uma evidência (RUSSEL; NORVIG, 2004).

Na TFC atribuir fator de probabilidade 0 a uma sentença significa a crença inequívoca de que essa sentença é falsa, enquanto que atribuir fator de probabilidade 1 a uma sentença significa o seu contrário, ou seja, a crença de que essa sentença está totalmente correta.

A TFC tem como maior função ser usada em sistemas nos quais o especialista possui poucas informações sobre o problema, o que faz com que ele tenha que considerar a sua crença nas hipóteses (RUSSEL; NORVIG, 2004).

Segundo Costa e Simões (2004), uma das particularidades da TFC que a diferencia da abordagem probabilística é que nela não há igualdade, ou seja, a soma das probabilidades do conjunto exaustivo e mutuamente exclusivo não soma 1. Levando-se em consideração que a TFC também é uma teoria baseada em regras, a representação da mesma é mostrada pela figura 1, onde k significa a crença nessa regra

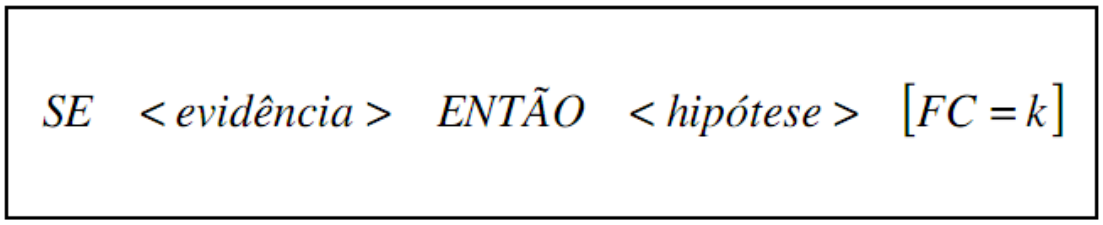


Figura 1. Regra se então
Fonte: COSTA, E.; SIMÕES, A. (2004)

2.2 COMPARAÇÃO DAS TEORIAS

A fim de resumir o capítulo sobre as teorias de modelagem, a tabela 1 apresenta uma comparação entre pos formalismo de modelagem, mostrando o nome da técnica, o tipo de incerteza que ela trata, seus autores, o ano de criação, o país e o que ela utiliza para representar o conhecimento:

Tabela 1. Técnicas de representação de conhecimento

Técnica	Incerteza tratada	Autores	Ano	País	Representação do conhecimento
Lógica *Fuzzy*	Imprecisão	Lotfi Asker Zadeh	1965	EUA	Conjuntos *Fuzzy*
Inferência Bayesianas	Aleatoriedade	Judea Pearl	1988	EUA	Redes Bayesianas
Teoria de Dempster-Shafer	Ignorância	Arthur Pentland Dempster e Glenn Shafer	1979	EUA	Funções de Crença
Fatores de Certeza	Confiança	Edward Hance Shortliffe e Bruce G. Buchanan	1970	EUA	Regras

Fonte: SILVANO, F. (2008)

Mesmo com a utilização da técnica de modelagem apropriada, se um software não for bem desenvolvido poderá não se comportar do modo necessário e acabar ocasionando resultados falhos, assim, a ciência que estuda o modo com o qual os softwares são construídos e, por conseguinte seus devidos testes de chama Engenharia de Software (PEZZÉ, 2008).

3. **ENGENHARIA DE SOFTWARE**

O software teve uma grande evolução desde seus primórdios nos anos 80, tornando-se uma ferramenta chave para o sucesso de empresas e tendo suas aplicações ampliadas. Hoje o software não é um artefato importante apenas para as empresas, mas também atua dentro das áreas científicas auxiliando em pesquisas e desenvolvimento de novas tecnologias em todas as áreas (PRESSMAN, 2005).

Com o crescimento da necessidade do software estar dentro dos mais diversos contextos começaram a se fazer necessárias novas pesquisas que auxiliassem os desenvolvedores no âmbito de projeto, confecção e controle de qualidade dos produtos. Essa nova área de pesquisas ficou conhecida como Engenharia de Software. Tal quais as outras engenharias, as disciplinas de engenharia de software aliam projeto, e verificação dos produtos final e intermediário, afim de que os defeitos sejam identificados e removidos e que sua qualidade seja explorada (PEZZÉ, 2008).

> "A engenharia de software é um rebento da engenharia de sistemas e de hardware. Ela abrange um conjunto de três elementos fundamentais - métodos, ferramentas e procedimentos - que possibilita ao gerente o controle do processo de desenvolvimento do software e oferece ao profissional uma base para a construção de software de alta qualidade produtivamente" (Pressman, 2005 p. 31)

Segundo Pezzé (2008), todos os produtos que são fabricados necessitam passar por uma série de testes para ter sua qualidade e eficiência comprovadas de modo formal.

Alguns aspectos de qualidade dentro do produto de software serão abordados na próxima seção.

3.1 QUALIDADE DE SOFTWARE

Um produto tem sua qualidade garantida quando passa por uma série de testes criados para se descobrir falhas e avaliar desempenho. No software isso não é diferente, ou seja, um software com qualidade é um software que teve suas características testadas afim de se avaliar seu desempenho e avaliar se o mesmo não possui nenhum tipo de erro (PEZZÉ, 2008).

Os aspectos de qualidade de um produto de software podem ser divididos em dois segmentos: os aspectos que são avaliados durante o processo de desenvolvimento do software e os aspectos que são avaliados após a entrega do mesmo ao cliente final. Os aspectos relativos ao desenvolvimento são chamados de aspectos internos, os aspectos que tangem o uso do produto são chamados aspectos externos (PRESSMAN, 2005).

Tanto as propriedades internas como as externas estão intimamente ligadas, pois com um bom projeto e um desenvolvimento organizado e acompanhado dificilmente aparecerão erros ou situações não tratadas no software (PEZZÉ, 2008).

Devido ao fato de que qualidade é algo que não é mensurável, foram criadas algumas métricas de qualidade de software. Essas métricas são características que um software precisa possuir para que seja considerado um software de qualidade (PRESSMAN, 2005).

A próxima seção abordará algumas dessas métricas de qualidade de software.

3.1.1 Métricas de qualidade de software

Conforme Pezzé (2008) descreve, um software para possuir qualidade precisa ter sua confiança comprovada. Confiança é a característica que faz com que o usuário possa utilizar o produto sem ter problemas. Dentre algumas características, as mais relevantes são: corretude, confiabilidade, disponibilidade, Tempo Médio Entre Falhas (do inglês Mean Time Between Failures - MTBF), riscos e robustez.

a) Corretude é o princípio de que um software é correto e condizente com as suas especificações;
b) Confiabilidade é uma medida de probabilidade do funcionamento correto do software dentro de uma unidade de tempo. É uma aproximação estatística da corretude. Um software 100% confiável é também um software 100% correto;
c) Disponibilidade é uma métrica utilizada para se representar quanto tempo um software pode ficar indisponível devido a uma falha, ou pode ser utilizada para representar os níveis de confiabilidade do software também, ilustrando assim uma medida que indique quanto tempo o software está disponível ao usuário. A unidade básica de medição mais utilizada para representar disponibilidade é o tempo;
d) MTBF também é uma métrica utilizada para demonstrar a disponibilidade do software. MTBF significa o tempo médio entre falhas, ou seja, se um software falha uma vez por dia e o tempo de indisponibilidade do mesmo é de 1 hora esse software possui um MTBF de 23 horas;
e) Corretude e Confiabilidade são dois aspectos muito importantes do software no âmbito das falhas, porém ambas não fazem distinções do que uma falha pode

causar no sistema. Uma falha que vai causar apenas um incômodo ao usuário deve ser distinguida de uma falha que pode causar um colapso no sistema e corromper uma base de dados;

f) Robustez basicamente é conceito de um software se degradar de maneira elegante, ou seja, conhecendo-se uma situação que pode vir a alavancar algum tipo de falha ou comportamento ineficiente o software deve se comportar de forma a sanar essa falha, mas não de uma maneira abrupta, por exemplo, um site que está com a sua capacidade de usuários excedida não pode ficar lento ou travar seu funcionamento, mas sim não aceitar mais usuários conectados ao mesmo, deixando assim uma margem de segurança e bom funcionamento para os usuários que já estão conectados ao mesmo.

    a. Segurança em um software pode ser considerada um tipo de robustez, mas robustez vai além da eliminação de riscos e trabalha também a funcionalidade do software em condições não usuais.

Além das métricas citadas acima por Pezzé, McCal (1997) cita mais algumas métricas de qualidade de software. Elas são: auditabilidade, acurácia, inteireza, concisão, consistência, padrões de dados, tolerância a erros, eficiência de execução, expansibilidade, generalidade, independência de Software e Hardware, modularidade, operabilidade, autodocumentação, simplicidade, e treinamento.

a) Auditabilidade é um índice que indica a o quão conforme o projeto está em relação aos padrões de desenvolvimento.
b) Acurácia mede a precisão dos tratamentos e dos controles do software.
c) Inteireza mede o quanto da implementação planejada foi alcançado.

d) Consistência mede o uso de técnicas e documentação do projeto ao longo do processo de desenvolvimento.
e) Padrões de dados verificam se as estruturas de dados foram padronizadas durante a implementação
f) Tolerância a erros verifica como foram feitos os tratamentos de erros de execução e como o sistema se comporta nesse tipo de situação
g) Eficiência de execução Mede utilizando alguma variável (tempo) se a execução do sistema foi eficiente ou não.
h) Expansibilidade analisa a capacidade de serem mais adicionados mais módulos á aplicação
i) Generalidade mede a utilidade dos componentes dos módulos do sistema.
j) Independência de software e hardware verifica o quanto o software é independente em relação a plataforma no qual ele está sendo executado.
k) Modularidade mede o nível de independência dos módulos do software.
l) Segurança Verifica os mecanismos de proteção dos dados e do software
m) Autodocumentação Verifica o quanto o sistema tem documentação explicativa e código fonte comentado.
n) Simplicidade Verifica se o programa é entendido facilmente.
o) Treinamento Mede o quanto o software auxilia novos usuários.

As métricas de qualidade devem envolver tanto os aspectos internos do software quanto os aspectos externos.

A partir destes conceitos o ato de se expor um software á uma série de testes que podem comprovar quais quesitos dos acima citados ele possui é chamado teste de Software (PEZZÉ, 2008).

A próxima seção trará uma comparação de funcionalidades entre os módulos da Shell Pegasus e os módulos de outras Shells que já atuam no mercado.

4. TESTE E ANÁLISE DE SOFTWARE

O teste e análise de software tem sido uma prática comum aplicada desde os primeiros projetos de software. Durante muito tempo essas práticas se basearam no senso comum e nas habilidades individuais, fazendo com que a área emergisse como uma disciplina somente há 3 décadas atrás (PEZZÉ, 2008).

Qualquer produto pode ser testado de duas maneiras: pela função específica que um produto projetado deve executar ou conhecendo o funcionamento interno do produto.

Devido ao fato de o software ser um produto complexo e também por não ser um produto produzido em série o teste de software muitas vezes é um processo complexo, de alto custo e que muitas vezes requer algum outro software para simular situações (PRESSMAN, 2005).

Devido ao fato de o software não ser um produto produzido em série, cada pacote de software é, ao menos, parcialmente único em seu projeto e funcionalidade. Os testes em cada pacote são feitos individualmente, tanto durante como depois de sua produção, a fim de identificar e eliminar possíveis erros. Devido também ao fato de o software ser um dos mais complexos e variáveis artefatos construídos de forma regular, o seu teste exige uma complexidade maior do que a de outros produtos (PEZZÉ, 2008).

Requisitos de qualidade que funcionam em um ambiente de execução e não funcionam em outro, estruturas que se deterioram na medida em que o software cresce, a não-linearidade inerente aos sistemas de software e a distribuição irregular dos erros são problemas que devem ser enfrentados no teste de software.

> Se um elevador pode transportar com segurança 1000kg, então pode transportar com segurança qualquer carga mais leve, mas se um procedimento ordena corretamente um conjunto de 256 elementos, o mesmo

> pode falhar para um outro conjunto de 255, 53 ou 15 elementos, bem como para 257 ou 1023.(PEZZÉ, 2008, p. 341)

A atividade de teste não pode mostrar a ausência de bugs; ela só pode mostrar se defeitos de software estão presentes. À medida que os dados dos testes vão sendo reunidos e avaliados, uma indicação da qualidade e confiabilidade do software começa a ser feita. Se durante o processo de teste, erros de grande complexidade começarem a aparecer com regularidade a qualidade e confiabilidade do software vão perdendo pontos, já se não foram encontrados grandes erros e/ou se os erros encontrados forem de fácil solução pode-se chegar a conclusão de que ou a qualidade e confiabilidade são altas ou os testes são inapropriados. Se os testes não encontrarem nenhum erro é sinal de que a configuração do teste não é adequada. Geralmente os erros não encontrados na fase de teses são detectados pelos usuários e corrigidos na fase de manutenção. O grande problema disso é que os custos de correção na fase de manutenção podem ser de 60 a 100 vezes maiores que na fase de teste (PRESSMAN, 2005).

Para se testar um software, selecionam-se alguns pontos específicos do mesmo, pois para se efetuar um teste completo em todo o programa seria necessário muito tempo. Por exemplo, um programa que executa uma função x^y, se levarmos em consideração que o programa utiliza números inteiros, e utilizando as regras das probabilidades, teríamos 18446544073709551616 possibilidades. Se cada caso desse pudesse ser executado em 1 milissegundos, precisaríamos de 5.849.424 séculos para executar o teste. O processo de teste necessita dos seguintes passos: Procedimentos iniciais, Planejamento, Preparação, Especificação, Execução e Entrega, onde os procedimentos iniciais consistem na elaboração do documento Guia Operacional de Testes (GOT), Planejamento e Preparação contém a parte de desenvolvimento de

estratégia e preparação do ambiente dos testes, a Especificação elabora e revisa os casos de testes, Execução consiste em executar os testes e a Entrega conclui o processo entregando o sistema para seu ambiente de produção (RIOS; MOREIRA, 2006).

4.1 TIPOS DE TESTE DE SOFTWARE

Assim como nas outras áreas da engenharia, um produto pode ser testado de duas maneiras: conhecendo as funções específicas que esse produto precisa executar e conhecendo-se o modo como ele funciona. Baseando-se nesse conceito existem duas abordagens de teste: teste funcional (black box) e teste estrutural (white box) (PRESSMAN, 2005) .

4.1.1 Teste Funcional

Os testes funcionais já vem sendo citados na literatura desde os anos 70. Eles têm como metodologia básica analisar o software como se o mesmo fosse uma caixa preta, para a qual são fornecidas entradas para o software e sua saída é comparada com a desejada. A partir desse teste é possível saber se o software condiz com seus requisitos e se ele está funcionando da maneira desejada no projeto (PRESSMAN, 2005) .

Eles analisam o programa de modo exaustivo, ou seja, jogando todas as entradas possíveis para assim testar todas as saídas e efetuando suas devidas comparações. Nem sempre esse tipo de teste é possível devido ao fato de que o domínio de entrada muitas vezes é complexo demais ou até em alguns casos define-se de modo infinito. O núcleo do projeto de teste funcional consiste em dividir os possíveis

comportamentos do software em xeque em várias classes, comparando-as e analisando se elas estão corretas ou não (conforme as especificações)

Em alguns casos o teste pode ser totalmente automatizado, como nos casos onde as especificações são dadas através de algum formalismo como uma gramática por exemplo (PEZZÉ, 2008).

Devido ao fato do teste funcional ser baseado todo em requisitos, se os mesmos não forem bem definidos o teste funcional pode não ser tão eficiente. Além de testar as entradas e saídas, é feita uma análise de integridade também nos arquivos gerados pelo sistema (RIOS; MOREIRA 2006).

4.1.2 Teste Estrutural

A estrutura do software por si já é uma fonte valiosa de informações para se escolher quais tipos de teste aplicar e se esses testes serão rigorosos o bastante para validar o software (PEZZÉ, 2008).

Os testes feitos utilizando-se de situações que provem que cada função do produto é totalmente operacional são chamados de testes de caixa branca (white box). O teste estrutural se baseia no conhecimento da estrutura interna do programa, sendo os aspectos de implementação fundamentais para a geração/seleção dos casos de teste associados (PRESSMAN, 2005) .

Diferente do teste funcional que analisa apenas as entradas e saídas do software, o teste estrutural estuda a fundo a aplicação, analisando seu conjunto de dados, seus caminhos lógicos e seus laços.

Os primeiros critérios de testes estruturais foram definidos por McCabe e eram baseados essencialmente no fluxo de controle dos programas. Dentro da classe de

testes Estruturais, estão inseridos de forma unitária alguns critérios de testes. Os principais critérios de teste estrutural são: critérios baseados na complexidade, critérios baseados em fluxo de controle, critérios baseados em fluxo de dados, critérios de Rapps e Weyuker e critérios Potenciais-Usos. Dentre eles, os quais recebem mais atenção são os critérios baseados em fluxo de dados (PRESSMAN, 2005) .

Antes de ser iniciado o capítulo que trata da aplicação dos testes na Shell Pegasus será feita uma pequena comparação dos módulos da Shell citada com outras Shells que já existem no mercado.

Além de testar o nível de confiança de um software, é possível também testar sua eficiência de execução. Para tal, devem ser definidos parâmetros e metodologias que auxiliarão na documentação e certificação desses testes, para isso dá-se o nome de análise de desempenho. A próxima seção abordará o tema.

4.2 DESEMPENHO DE SOFTWARE

Define-se por desempenho de software o conjunto de requisitos não-funcionais que são relacionados á velocidade de operação ou as restrições de uso de um sistema. O desempenho computacional descreve a velocidade com a qual um determinado sistema pode executar um programa ou um conjunto de programas (CARTER, 2003, p.12), porém dentro do contexto de desempenho existem diferentes tipos de requisitos que podem ser analisados e especificados,são eles: Tempo de resposta, Throughput e Temporização.

Tempo de resposta é um requisito que especifica um tempo aceitável para que uma operação seja concluída no sistema, Throughput especifica a quantidade de

dados que o sistema precisa processar dada uma quantidade de tempo e a Temporização indica quanto tempo o sistema deve levar para processar uma entrada advinda de algum sensor, antes que o próprio sistema a sobrescreva.

Apesar do tempo ser considerado a única medida de desempenho confiável e coerente quando fazemos uma medição de tempo de execução precisamos levar em consideração todos os fatores que influenciarão na execução do sistema. Somente com uma análise completa do ambiente e dos tempos é que podemos afirmar se um sistema pode ser considerado de alto desempenho (HENESSY; PATTERSON, 2003).

O desempenho de software está totalmente ligado a usabilidade do mesmo, pois se um sistema é lento ele acaba diminuindo a produtividade do usuário, e se um sistema ocupa muita memória da máquina ele pode afetar as outras aplicações que estão sendo executadas no sistema operacional, ou pode se tornar tão lento que o sistema operacional precisará usar técnicas de escalonamento para executá-lo (SOMMERVILLE, 2003).

4.2.1 Testes de desempenho

Assim como nos testes de qualidade, onde existem métodos de teste para assegurar que a execução do software não possui erros e possui os requisitos de qualidade pelos mesmos especificados, também existem testes que têm por função analisar os requisitos de desempenho de um sistema.

Apesar de os testes poderem ser aplicados enquanto o software ainda está sendo desenvolvido, só podemos ter uma medida real de desempenho do mesmo após todos os seus componentes serem integrados (PRESSMAN, 2005).

Existem basicamente 3 tipos de teste de desempenho de software: teste de carga, testes de estresse e teste de volume.

Os testes de carga têm por função testar o software no sentido de quantos usuários simultâneos o mesmo suporta. Esse teste auxilia na observação dos tempos de resposta para cada usuário do sistema e observa também a ocorrência de erros em situações de uso (PRESSMAN, 2005).

Testes de estresse são parecidos com o teste de carga, porém os testes de estresse inserem no sistema uma carga de usuários além da qual o sistema foi projetado, para assim analisar o comportamento e as falhas que o sistema pode apresentar quando exposto a situações de uso extremas. Esse tipo de teste tem mais relevância em sistemas distribuídos, pois com a sobrecarga das unidades de processamento da rede a mesma tem seu tráfego mais lento, o que acaba tornando os processos que necessitam de dados da rede mais lentos e mais suscetíveis a erros (SOMMERVILLE, 2003).

Testes de volume testam a quantidade de dados que um dado sistema pode processar, para testar assim se o volume de dados que o sistema suporta é o mesmo que foi especificado no desenvolvimento do sistema. Esse teste opera também com uma quantidade maior do que a qual o sistema foi projetado, para assim definir os limites de volume que o sistema consegue trabalhar sem ter erros.

O teste de volume é o único que não leva em consideração o número de usuários que estão usando o sistema simultaneamente.

Para auxiliar no desenvolvimento desses testes foram desenvolvidas ferramentas que auxiliam nos mesmos, também chamadas de ferramentas de teste ou Benchmarks.

5. FERRAMENTAS DE TESTE

As ferramentas de teste auxiliam na análise do processo de execução de um software, mostrando o consumo de memória, os tempos de execução entre outras funções.

Existem diferentes ferramentas de teste no mercado, algumas com licença gratuita e outras não, existem também algumas que são grátis apenas por um tempo limitado.

Para os testes com a shell Pegasus, foi escolhida a ferramenta de teste JProfiler produzida pela empresa EJ Technologies na sua versão 6.02, essa ferramenta pode ser obtida no site da empresa que a produziu.[3]

5.1 A FERRAMENTA JPROFILER 6.02

JProfiler é uma ferramenta desenvolvida pela empresa alemã EJ-Technologies, a qual desenvolve soluções para auxiliar desenvolvedores a sanar problemas em relação á desempenho e distribuição de software.

Suas principais vantagens são ter uma licença gratuita por 10 dias, uma interface de fácil utilização, mostrar resultados em tempo integral, ter um baixo consumo de memória e poder ser integrada em várias IDEs. Ela é compatível também com ambientes tanto 32 quanto 64 bits.

[3] http://www.ej-technologies.com/download/jprofiler/trial

A ferramenta também mostra o consumo de todas as classes e métodos da aplicação, e oferece no final de cada sessão os métodos que tiveram sobrecarga de chamados, auxiliando assim o desenvolvedor a desenvolver um código mais eficiente.

Devido ao fato de um software ser um conjunto de algoritmos, o próximo capítulo falará sobre análise de algoritmos, suas classes de custo e seu desempenho.

6. COMPLEXIDADE DE ALGORITMOS

Segundo Dijkstra, algoritmos são a representação de um comportamento através de um conjunto finito de ações.

Devido ao fato de um software ser um conjunto de algoritmos, o nível de desempenho de um software está intimamente ligado aos algoritmos utilizados no mesmo (PRESSMAN, 2005).

Para se analisar o desempenho de um algoritmo é necessário analisar o custo do mesmo, ou seja, o tempo de execução dele e a quantidade de memória que ele ocupa (TOSCANI; PAULO, 2005).

A análise do custo pode ser feita de dois modos: através da execução do programa ou através de um modelo matemático.

Analisar o custo de um algoritmo através da execução do programa não é considerada uma metodologia confiável, pois vários fatores podem influenciar no resultado, como o compilador usado e o hardware da máquina, já a análise através de um modelo matemático é a mais confiável, pois ela utiliza modelos matemáticos dentro de um computador idealizado (TOSCANI; PAULO, 2005).

Para medir o custo de execução, comumente se define uma função f para representar o custo que esse algoritmo representa, também chamada de função de custo ou função de complexidade (ZIVIANI, 1993).

Conforme a evolução dos tempos de execução de um algoritmo, podemos definir uma função de custo que represente a complexidade do mesmo.

Conforme o comportamento da execução de um dado algoritmo, ele pode pertencer a uma das 8 classes de algoritmos, que serão listadas logo abaixo:

a) A classe f(n) = O(1), possui os chamados algoritmos de complexidade constante, pois são algoritmos os quais não importa o número de instruções o algoritmo é executado em um número fixo de vezes, portanto é a classe com a menor complexidade e custo (ZIVIANI, 1993).
b) A classe f(n) = O(logn) é chamada de classe dos algoritmos de complexidade logaritmica. É Típica de algoritmos que quebram um problema grande em outros menores, esse algoritmo executa uma instrução em uma quantidade menor do que n (ZIVIANI, 1993).
c) Algoritmos da classe f(n) = O(n) são os algoritmos de complexidade linear, ou seja, o número de execuções do algoritmo equivale ao número de vezes da instrução (ZIVIANI, 1993).
d) Algoritmos da classe f(n) = ON(nlogn) são típicos de algoritmos que quebram um problema em outros menores, resolvem cada um deles independentemente e junta as soluções depois (ZIVIANI, 1993).
e) A classe f(n) = O(n^2) são os algoritmo de complexidade quadrática. Ocorre quando os itens de dados são processados aos pares (ZIVIANI, 1993).
f) Algoritmos da classe f(n) = O(n^3) são algoritmos de complexidade cúbica. São úteis apenas para resolver pequenos problemas (ZIVIANI, 1993).
g) A classe f(n) = O(2^n) possui algoritmos de complexidade exponencial. São utilizados em problemas que envolvem força bruta. (ZIVIANI, 1993).

h) A classe f(n) = O(n!) é dita de complexidade exponencial, apesar de ter um comportamento muito pior do que O(2^n) (ZIVIANI, 1993).

A figura 2 ilustra algumas classes de complexidade de algoritmo e seu custo em relação ao tamanho do problema:

Função de custo	Tamanho n					
	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0.35 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0.316 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
2^n	0,001 s	1 s	17,9 min	12,7 dias	35,7 anos	366 séc.
3^n	0,059 s	58 min	6,5 anos	3855 séc.	10^8 séc.	10^{13} séc.

Figura 2 - Relação de custo das classes de algoritmos
Fonte: (ZIVIANI, 1993)

7. TRABALHO DESENVOLVIDO

Esse capítulo trará todo o trabalho que foi desenvolvido durante a pesquisa:

7.1 ANÁLISE DE QUALIDADE E DESEMPENHO DA SHELL PEGASUS

Essa seção abordará dois temas: a análise de alguns dos ítens de qualidade de software estudados anteriormente dentro da shell Pegasus e a análise de desempenho da mesma. Cada seção abordará as metodologias empregadas e trará seus respectivos resultados.

7.1.1 Itens de qualidade analisados na *Shell Pegasus*

Baseando-se em todos os conceitos discutidos nos capítulos anteriores, foi analisado o funcionamento da Shell Pegasus e quais ítens de qualidade caberiam ser analisados na Shell.

Devido ao fato de a Shell não utilizar banco de dados e ser um software que roda localmente sem o uso de mais de um usuário simultâneo, os ítens escolhidos para a análise foram os que tangiam a parte funcional.

Dentre todas as métricas foram escolhidas 14, as quais estão listadas a seguir: auditabilidade, acurácia, inteireza, consistência, padrões de dados, tolerância a erros, eficiência de execução, expansibilidade, generalidade, independência de Software e Hardware, instrumentação, modularidade, operabilidade, autodocumentação.

7.1.2 Análise de qualidade da *Shell Pegasus*

Devido ao fato de o objetivo dessa pesquisa ser avaliar o desempenho da Shell Pegasus, a análise de qualidade da mesma não foi executada, pois para fazê-la

seria necessária uma pesquisa sobre o significado exato de cada métrica e sobre as metodologias de análise das mesmas.

7.2 ANÁLISE DE DESEMPENHO DA SHELL PEGASUS

Essa seção abordará as metodologias utilizadas para a análise de desempenho da Shell Pegasus, assim como seus resultados e conclusões.

7.2.1 Metodologia utilizada

A metodologia utilizada para a análise de desempenho foi baseada no fator tempo.

Utilizando-se a ferramenta de análise de desempenho JProfiler 6.02, foram realizadas inferências em cada módulo da shell documentando-se o tempo de execução de cada inferência para uma análise posterior.

A fim de se analisar a taxa de crescimento do tempo de execução da shell, as inferências foram divididas em sessões, cada qual com um volume de dados crescente.

Após o fim dos testes, os tempos de execução foram catalogados, obtendo-se a média do tempo com cada volume de dados para efetuar suas devidas comparações.

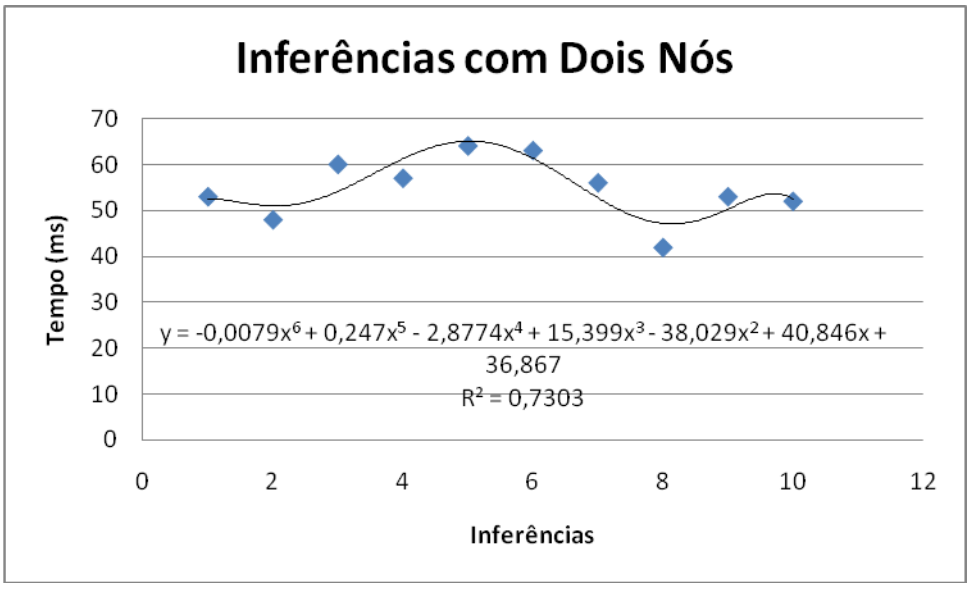
7.2.2 Ambiente de execução

O ambiente utilizado para os testes possuía o sistema operacional Windows Vista Home Premium 64 bits e a máquina possuía a seguinte configuração: processador Intel Core 2 Quad Q6600 2.4 GHZ, 8 GB de memória RAM e 500 GB de disco rígido.

7.3 ANÁLISE DE DESEMPENHO DO MÓDULO DE REDES BAYESIANAS DA SHELL PEGASUS

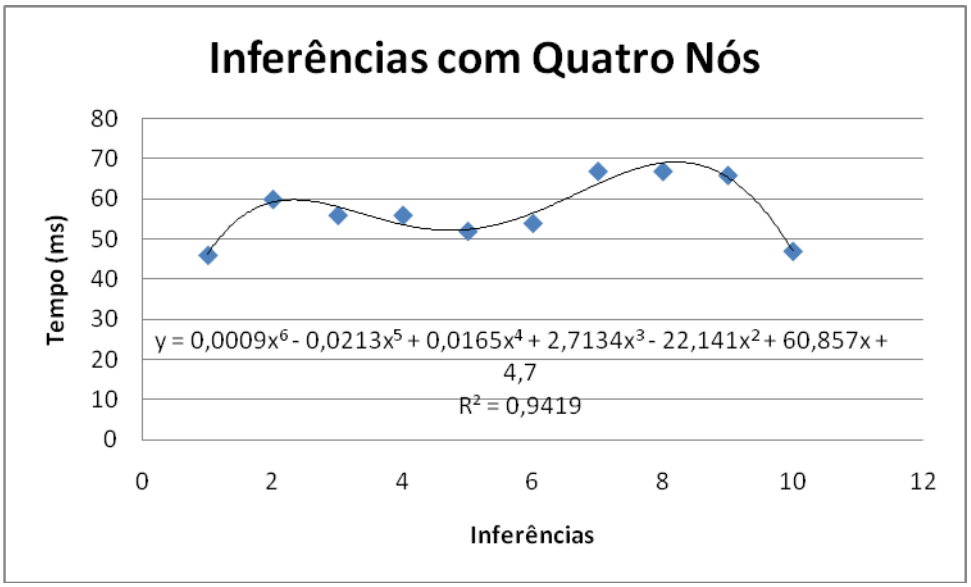
Para a análise de desempenho do módulo de redes Bayesianas da shell Pegasus, foram criadas 3 redes Bayesianas com as seguintes configurações: 2 nós, 4 nós e 6 nós.

Em seguida procedeu-se a realização da análise estatística com os mesmos números de inferências (dez) para cada modelo. Os gráficos abaixo mostram os tempos de cada teste, a equação da função polinomial que melhor se ajusta aos tempos obtidos e o coeficiente de determinação (r^2), que indica o quando o tempo é explicado pelo número de inferências, sendo que quanto mais próximo de 1 (um) for o seu valor, melhor o modelo matemático explica essa relação.



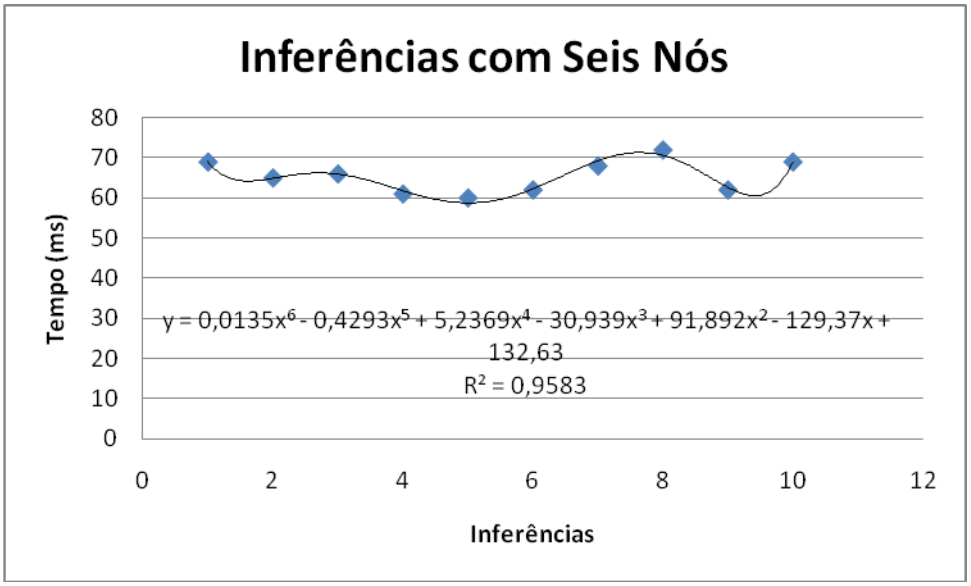
Fonte: Elaborado pelo autor.

Figura 3. Resultado das inferências em uma rede com 2 nós



Fonte: Elaborado pelo autor.

Figura 4. Resultado das inferências em uma rede com 4 nós



Fonte: Elaborado pelo autor.

Figura 5. Resultado das inferências em uma rede com 6 nós

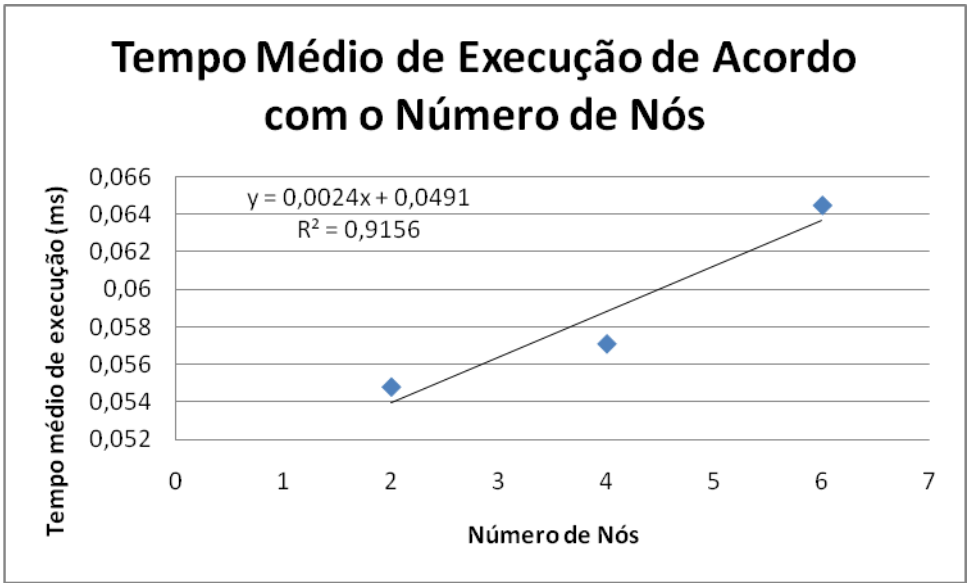
Pode-se perceber que a curva que melhor se ajusta aos dados é a originada de uma função polinomial do sexto grau, o que fica mais evidente quanto mais nós acrescentam-se à análise, chegando a um coeficiente de determinação de $r^2 = 0,9583$. A questão que se segue é de que forma se comporta o tempo médio de cada análise.

Nessa direção, após calcular as médias de tempo de execução com os 3 tipos diferentes de redes, foi calculado a média das médias afim de achar o tempo médio total de execução e se avaliar como é o crescimento dos tempos. A tabela 2 ilustra os dados:

Tabela 2. Média dos tempos de execução do Módulo Bayesiano da Shell Pegasus

Tipo de rede (nós)	Tempo médio de execução (ms)
2	0.0548
4	0.0571
6	0.0645
Média total	0.0588

Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.

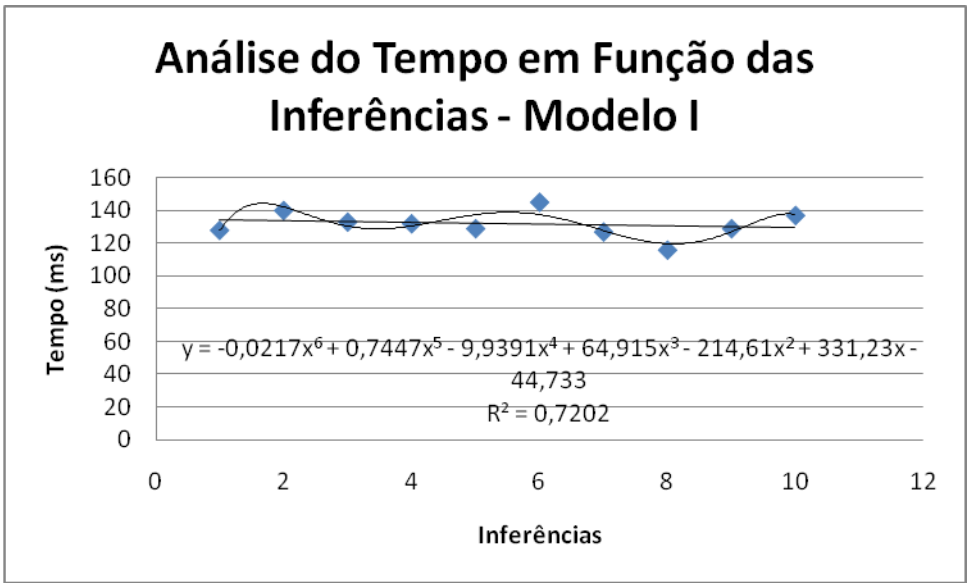
Figura 6. Média dos tempos de execução das inferências

A observação que se faz é de que embora durante os testes onde fixaram-se o número de nós e realizaram-se as inferências, o modelo explicativo é polinomial, porém, quando tomadas as médias das inferências pode-se ter a clara visão de que o comportamento das médias ao acrescentarmos nós no modelo é linear, com um coeficiente de determinação $r^2 = 0,9156$.

7.4 ANÁLISE DE DESEMPENHO DO MÓDULO DE FATORES DE CERTEZA DA SHELL PEGASUS

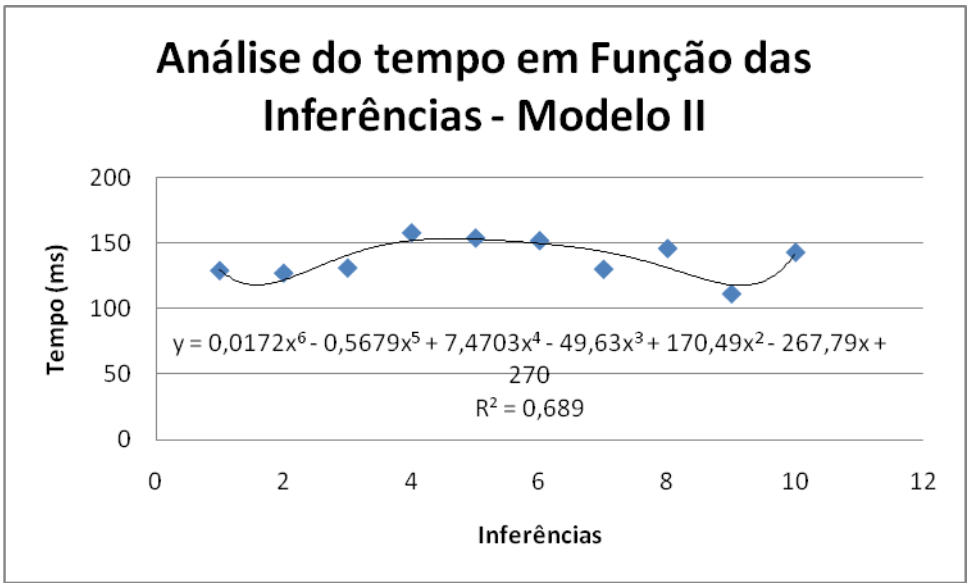
Para a análise do módulo de Fatores de Certeza da Shell Pegasus, foram criados 3 modelos com as seguintes características: 3 variáveis, 5 variáveis e 7 variáveis.

Foram executadas dez inferências em cada modelo, os tempos foram catalogados e inseridos em um gráfico, conforme ilustram as figuras que seguem.



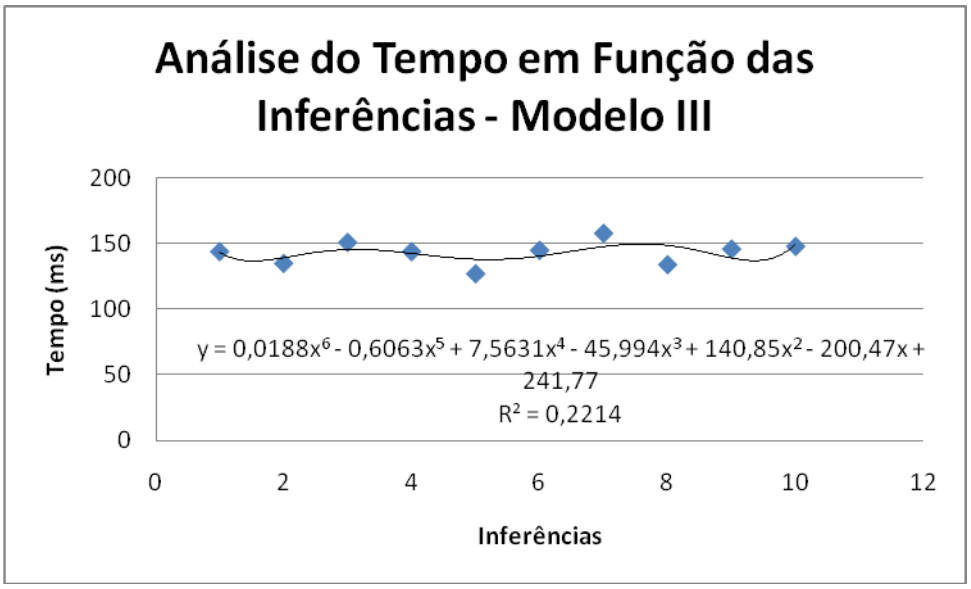
Fonte: Elaborado pelo autor.

Figura 7. Tempos de execução das inferências em um modelo com 3 variáveis



Fonte: Elaborado pelo autor.

Figura 8. Tempos de execução das inferências em um modelo com 5 variáveis



Fonte: Elaborado pelo autor.

Figura 9. Tempos de execução das inferências em um modelo com 7 variáveis

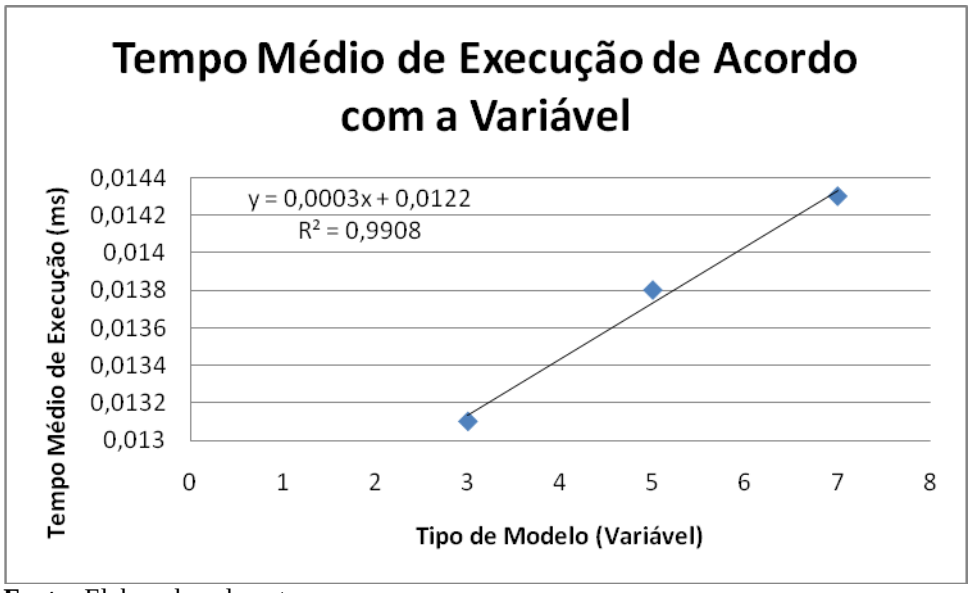
Após as execuções, foi calculado o tempo médio de cada modelo e o tempo médio total de execução do módulo conforme ilustra a tabela . Esses dados foram inseridos em um gráfico conforme a figura 30 e a tabela 6:

Tabela 3. Média dos tempos de execução do módulo de Fatores de Certeza da *shell* pegasus

Tipo de modelo (variável)	Tempo médio de execução (ms)
3	0.0131
5	0.0138
7	0.0143
Média total	0.0137

Fonte: Elaborado pelo autor.

Médias de tempo de execução das inferências do módulo de fatores de certeza.



Fonte: Elaborado pelo autor.

Figura 10. Médias de tempo de execução das inferências do módulo de fatores de certeza

Pode-se perceber que quando analisados cada modelo de forma separada, quanto ao tempo em relação ao número de inferências ficou evidente novamente que o modelo que mais adéqua a situação é o polinomial, porém, a medida em que se inserem novas variáveis ele vai perdendo a sua força de determinação. Já quando são analisadas as médias dos grupos de inferências, percebe-se uma tendência linear, com coeficiente de determinação $r^2 = 0,9908$.

7.5 ANÁLISE DE DESEMPENHO DO MÓDULO DE DEMPSTER-SHAFER DA SHELL PEGASUS

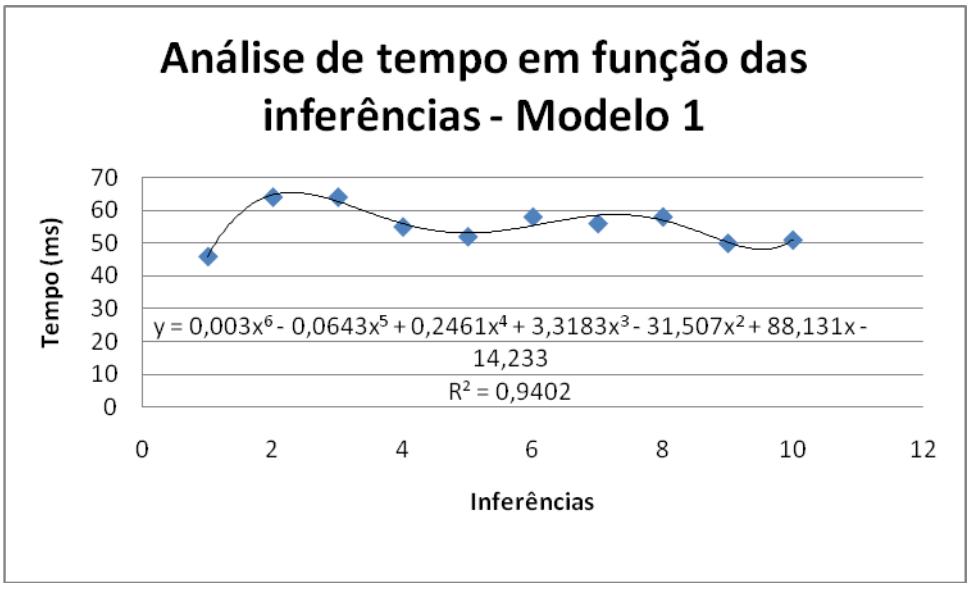
Para a análise do módulo de Dempster-Shafer da Shell Pegasus, foram criados 3 modelos com as seguintes características:

a) Modelo 1: 4 evidências, 2 hipóteses e 2 especialistas

b) Modelo 2: 6 evidências, 4 hipóteses e 2 especialistas

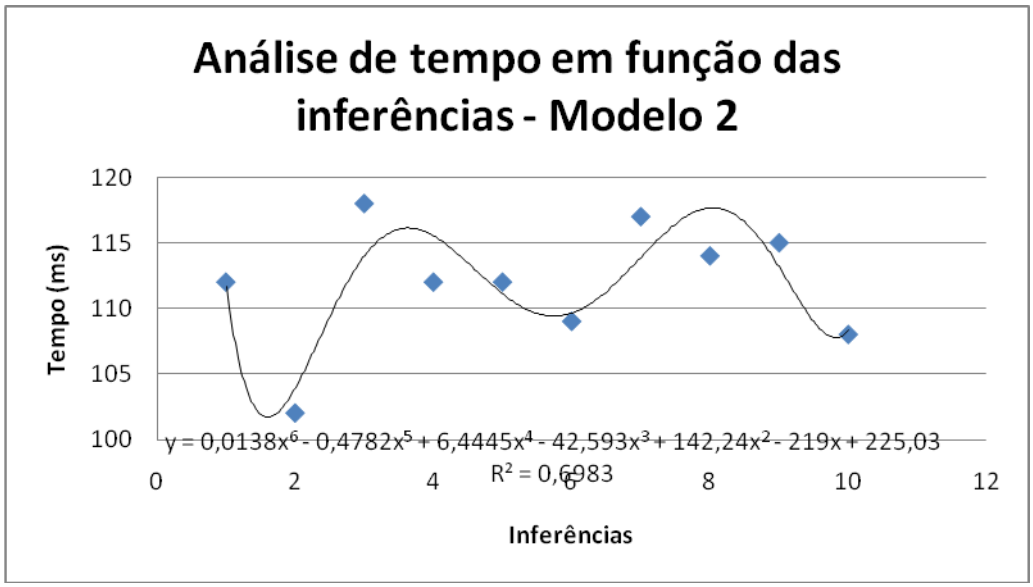
c) Modelo 3: 8 evidências, 6 hipóteses e 2 especialistas

Para cada modelo foram executadas 10 inferências, seus tempos de execução foram catalogados e utilizados para calcular as médias de tempo. As figuras mostram os tempos de execução das inferências:



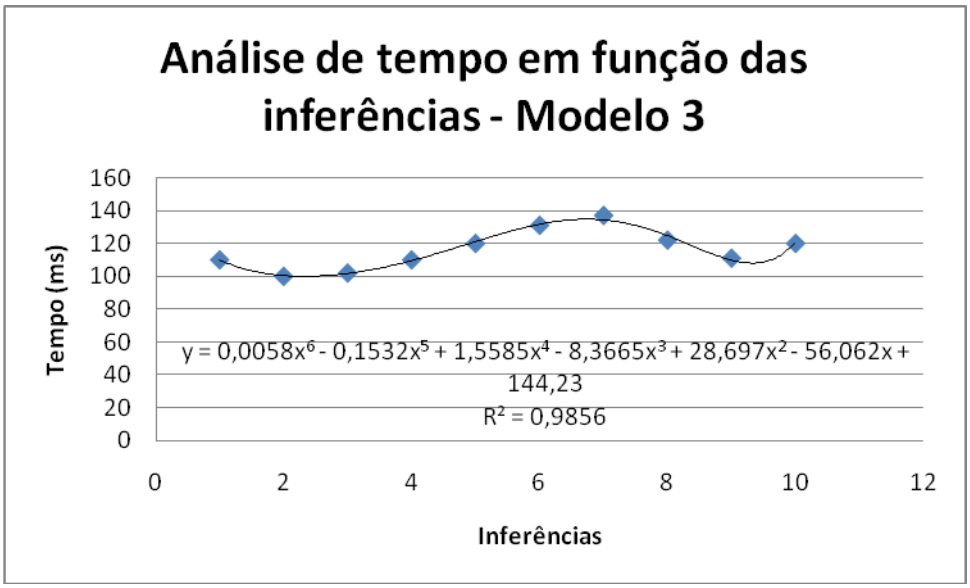
Fonte: Elaborado pelo autor.

Figura 11. Tempos de execução das inferências em um modelo com 6 evidências, 4 hipóteses e 2 especialistas



Fonte: Elaborado pelo autor.

Figura 12. Tempos de execução das inferências em um modelo com 6 evidências, 4 hipóteses e 2 especialistas



Fonte: Elaborado pelo autor.

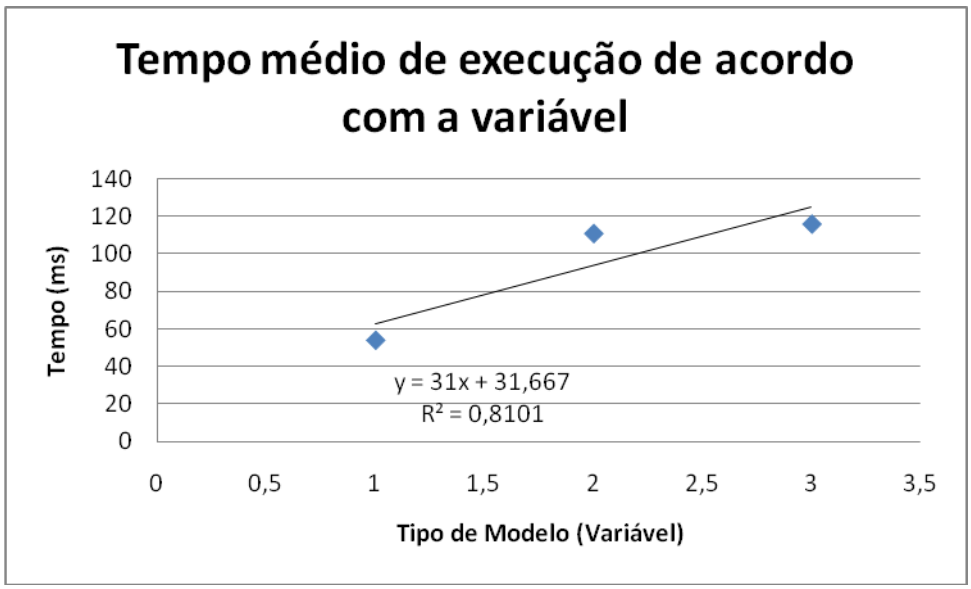
Figura 13. Tempos de execução das inferências em um modelo com 8 evidências, 6 hipóteses e 2 especialistas

Após as execuções, foram calculados os tempos médios de cada modelo e analisados em um gráfico a fim de visualizar sua taxa de crescimento, conforme mostram a tabela 7 e a figura 34:

Tabela 4. Tabela média dos tempos de execução dos modelos no módulo de Dempster-Shafer da Shell Pegasus

Tipo de modelo			Tempo médio de execução (ms)
Evidências	Hipóteses	Especialistas	
4	2	2	0.0054
6	4	2	0.0111
8	6	2	0.0116
Média total			0.0093

Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.

Figura 14. Média dos tempos de execução dos modelos no módulo de Dempter-Shafer da ShellPegasus

Pode-se perceber que ao analisarmos os modelos isoladamente em relação às inferências realizadas, que a curva que melhor se ajustou a distribuição nos três casos foi a curva polinomial, porém com um coeficiente de determinação instável (moderado à forte).

Tomando a média de desempenho de cada modelo, pode-se perceber claramente uma tendência linear, com coeficiente de determinação $r^2 = 0,8101$.

7.6 FATORES DE QUALIDADE QUE PODEM AFETAR NO DESEMPENHO

Conforme já foi citado em outras seções do trabalho, vários aspectos do software podem influenciar no desempenho do mesmo. Aspectos como a linguagem utilizada no desenvolvimento, tipos de dados utilizados, algoritmos usados e vários outros fatores.

A Shell Pegasus foi desenvolvida utilizando-se a linguagem Java, que é multiplataforma. Para que a linguagem pudesse ser multiplataforma ela foi construída de modo que interpretasse seus códigos por meio dos esquemas de *bytecodes*, o que torna seu desempenho inferior as linguagens que utilizam códigos nativos.

Alguns aspectos de inconsistência de código também foram observados, pode-se citar, por exemplo, a possibilidade de se inserir duas regras idênticas no módulo de Fatores de Certeza.

7.7 ANÁLISE DE DESEMPENHO GERAL DOS MÓDULOS DA SHELL PEGASUS

Conforme os resultados obtidos com as inferências, e analisando os gráficos das médias de tempo de execução de cada módulo, os módulos crescem linearmente conforme sua carga de dados aumenta.

Devido ao seu crescimento linear, o algoritmo da Shell Pegasus pertence a uma dada classe de complexidade de algoritmos, essa classe será estudadas na próxima seção, e ao final será a classificação da Shell em estudo.

7.8 ANÁLISE DO ALGORITMO DA SHELL PEGASUS

Conforme análise estatística dos tempos de execução dos módulos da Shell, chegou-se a constatação que o algoritmo utilizado nos módulos da *Shell Pegasus* tem uma progressão linear, o que o torna de nível de complexidade O(n).

Conforme analisado na pesquisa, os algoritmos dessa classe são considerados de baixo custo, portanto podemos considerar que o mesmo possui um desempenho aceitável.

8. COMPARAÇÃO DAS SHELLS

Nesse capítulo será feita uma comparação dos módulos da Shell Pegasus com outras *shells* que já atuam no mercado, afim de uma exposição de seus atributos.

8.1 O MÓDULO BAYESIANO DA SHELL PEGASUS

Nesta seção será feita uma comparação entre o módulo Bayesiano da *Shell Pegasus* e a *Shell* Hugin Lite 7.2

8.1.1 A *Shell* Hugin Lite 7.2

Hugin é uma *shell* que foi desenvolvida pela empresa HUGIN EXPERT A/S, a qual existe desde 1989 e é líder em distribuição de softwares para decisões avançadas baseadas em modelos estatísticos.

O Hugin Lite 7.2 é um software que foi desenvolvido para tratar a modelagem de incerteza utilizando os modelos de Redes Bayesianas, sua licensa é *free* e o software pode ser baixado no site da empresa desenvolvedora.

Na tela inicial da *shell*, o menu superior oferece algumas funcionalidades, tais quais salvar um modelo, abrir um modelo já existente, criar um novo modelo e algumas configurações para o aprendizado da rede, conforme ilustra a figura 2. A *shell* oferece também algumas ferramentas de criação da rede como criação de nós, ligação entre nós, nó de decisão e outros, além de oferecer as funções de edição mais básicas como exclusão e edição.

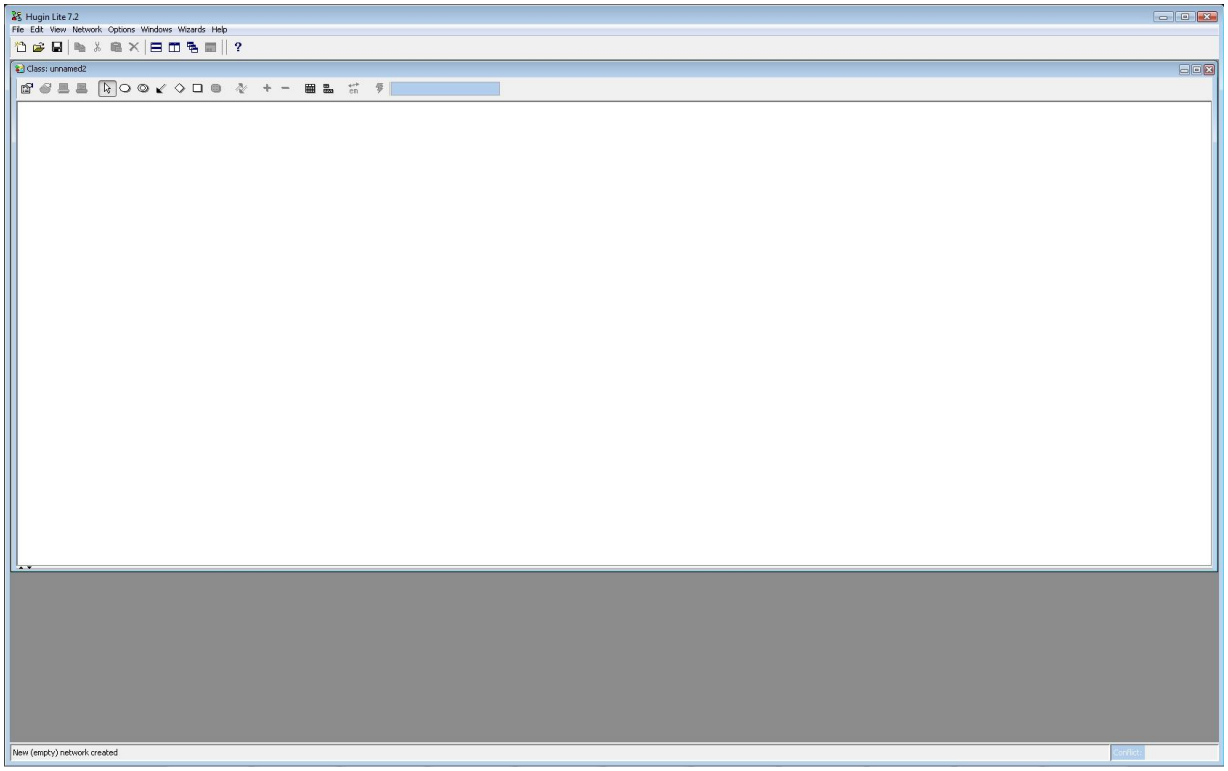


Figura 15. Tela principal do Hugin Lite 7.2

No menu principal da ferramenta existem mais algumas funções relativas á árvore, como opções de alinhamento dos nós, modos de exibição das janelas do software, importação da árvore em forma de figura e outros.

O menu secundário traz algumas funcionalidades que estão presentes no menu principal,mas de modo resumido.

A ferramenta permite através do menu da área de criação da árvore inserir todos os tipos de nós existentes em uma rede Bayesiana, como mostra a imagem 3 :

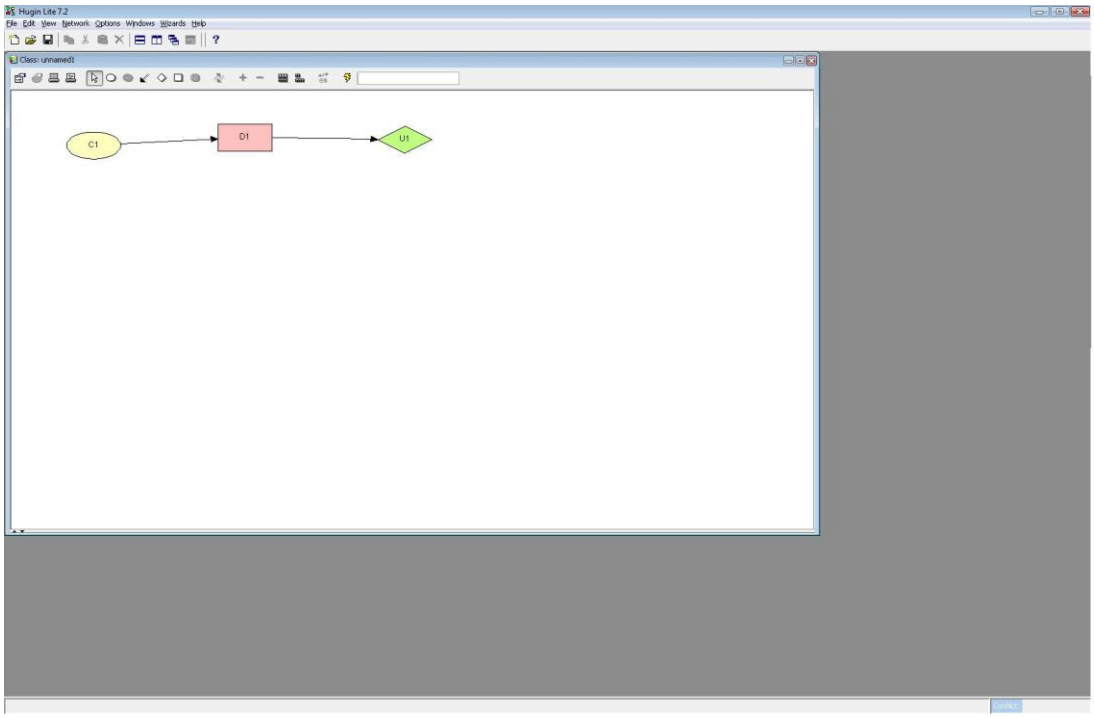


Figura 16. Uma rede Bayesiana com tipos de nós distintos

Após o nó ser inserido é possível acessar suas propriedades através da função de edição no menu da área de desenvolvimento, dentre as propriedades editáveis do nó estão ítens como: nome, *label*, tipo (pode ser numérico, booleano ou intervalo), tamanho, descrição, estados, funções e atributos, conforme ilustra a figura :

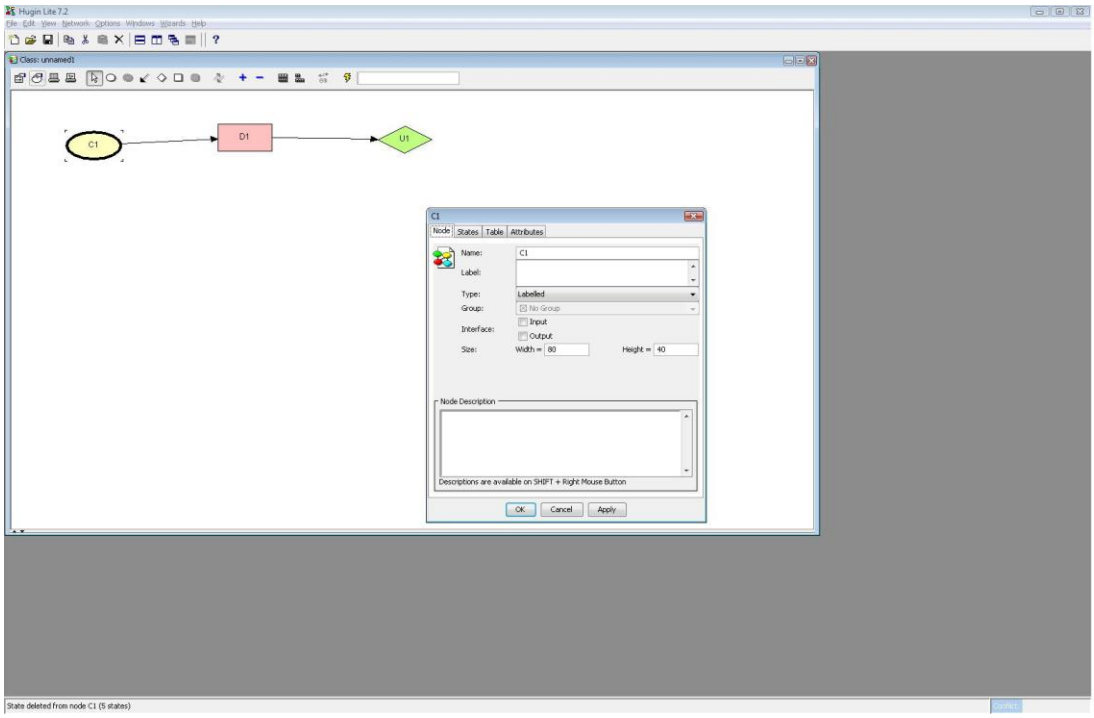


Figura 17. Menu de edição de um nó da árvore

Após inserir todos os dados da árvore, é possível se executar o modelo para verificar seus resultados clicando na opção run.

Após a execução do modelo o software oferece uma nova barra de menu, na qual estão algumas opções de visualização dos resultados, como ilustra a figura 5:

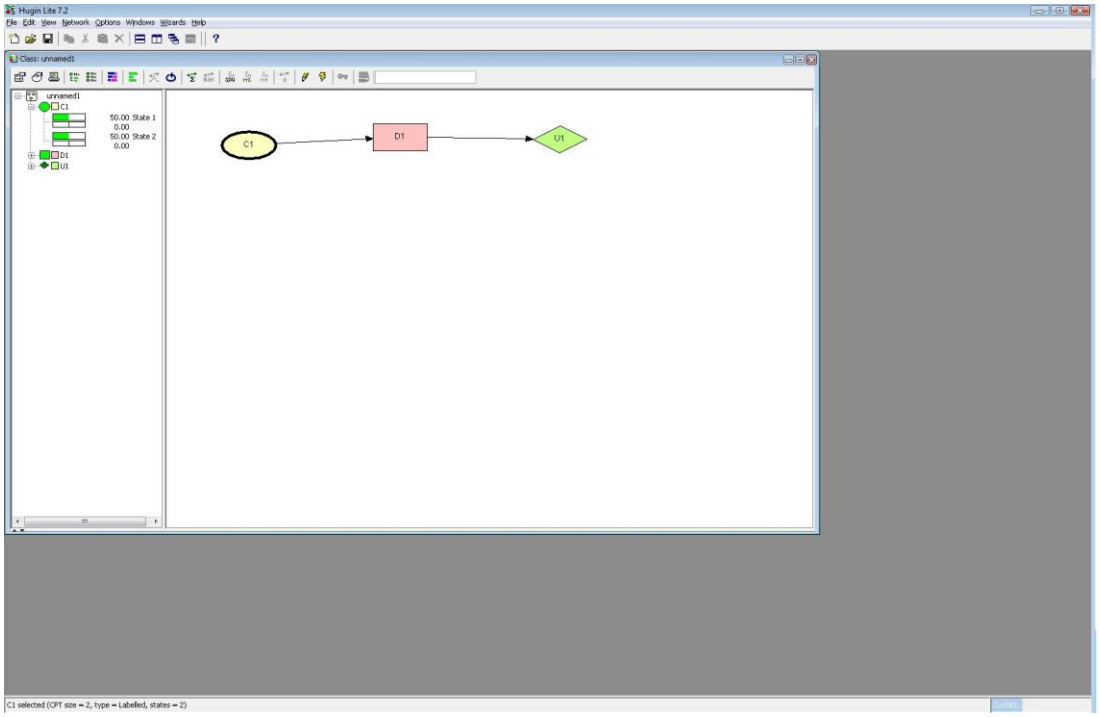


Figura 18. Opções de exibição do resultado

8.1.2 O módulo de Bayes da *Shell Pegasus*

O módulo de Bayes da Shell Pegasus foi desenvolvido por Edroaldo Lummertz da Rocha, o qual faz parte do grupo de Pesquisa em Inteligência Computacional Aplicada da Universidade do Extremo Sul Catarinense - UNESC - em 2008.

O módulo de Bayes da *shell* Pegasus, possui em sua tela inicial uma barra de menu principal, uma barra de menu secundária para navegação dentro da Shell Pegasus e uma área para o desenvolvimento da árvore, a qual possui um menu próprio para a criação da mesma, como ilustra a figura 6. O menu principal oferece também uma área de ajuda, na qual são exibidos os dados dos criadores do

software.

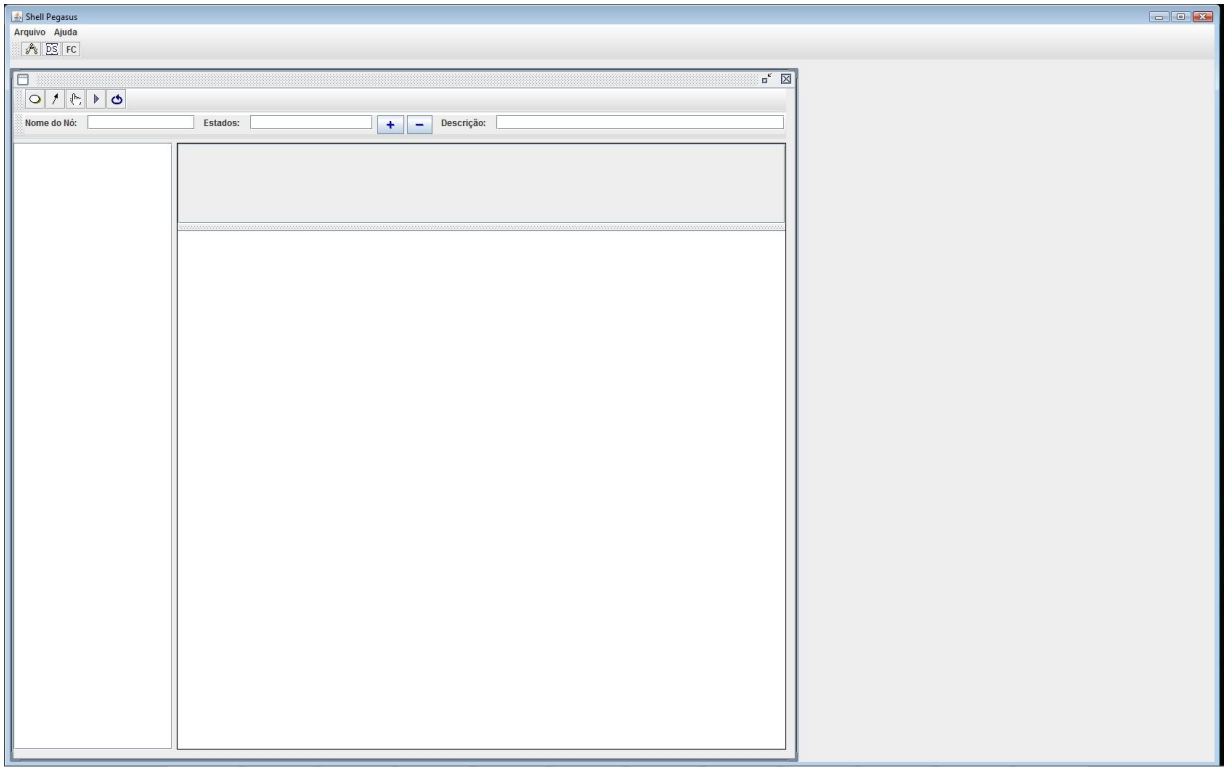


Figura 19. Tela inicial do módulo de Bayes da *Shell* Pegasus

Para a criação da árvore é possível inserir nós, conectar os nós, inserir estados, inserir uma descrição e executar o modelo a fim de se obter seus resultados, ou seja, todas as funcionalidades básicas para se trabalhar com esse tipo de modelagem, conforme ilustra a figura . Após a criação e execução da árvore, a *shell* mostra os resultados no lado esquerdo da área de criação, conforme ilustra a figura 7.

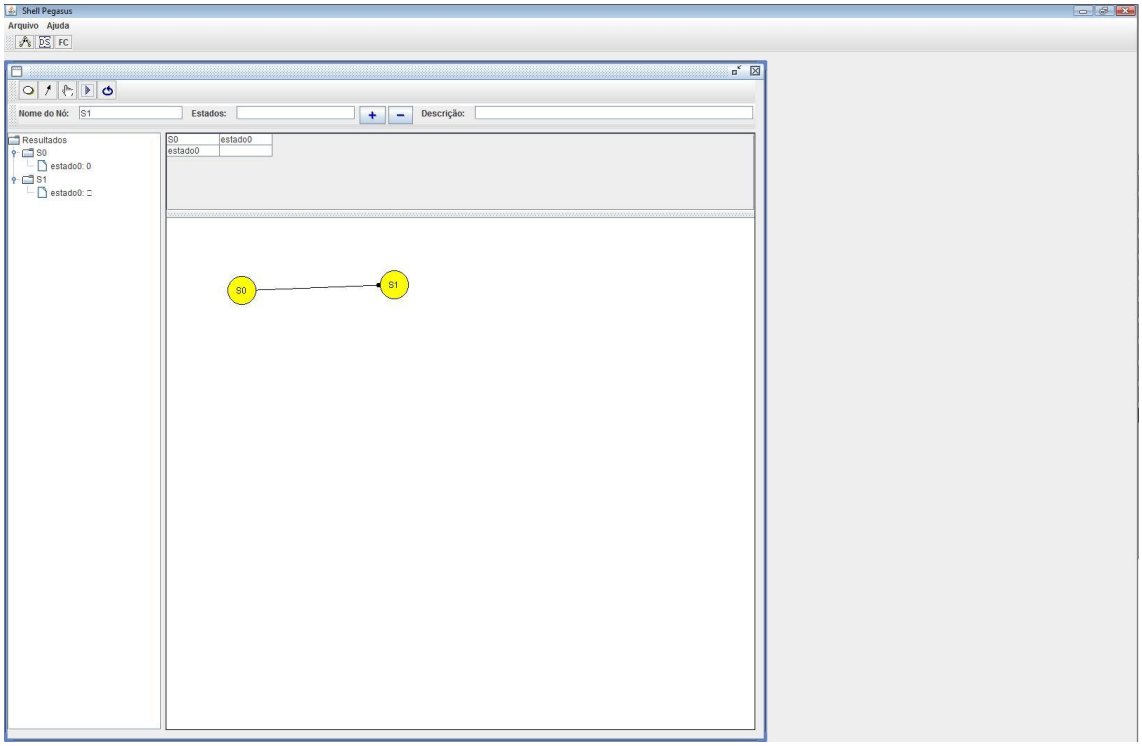


Figura 20. Tela principal do módulo de redes Bayesianas da *Shell* Pegasus

8.1.3 Comparativo entre a *Shell* Hugin Lite 7.2 e a *Shell* Pegasus

A tabela abaixo ilustra a comparação de funcionalidades entre as duas shells:

Funcionalidade	Shell	
	Hugin Lite 7.2	Módulo Bayes Pegasus
Criar novo modelo	x	x
Salvar modelo	x	
Inserir nós no modelo	x	x
Abrir um modelo já existente	x	
Criação de estados	x	x
Execução do modelo	x	x
Edição de nós	x	x

Tabela 5. Comparação entre as *shells* Hugin Lite 7.2 e o módulo Bayes da *shell* Pegasus

8.2 O MÓDULO DE FATORES DE CERTEZA DA SHELL PEGASUS

Essa seção trará uma comparação entre a *Shell* Expert Sinta e o módulo de Fatores de Certeza da *Shell Pegasus*

8.2.1 A *Shell* Expert Sinta

A *shell* Expert Sinta foi desenvolvida pelo Grupo de Sistemas Inteligentes Aplicados do laboratório de Inteligência Artificial, LIA/UFC-UECE, em 1997.

O sistema permite a criação de uma base de dados para sistemas inteligentes A shell utilizando a modelagem dos Fatores de Certeza, permitindo a inserção de regras, variáveis e objetivos. A shell permite também uma configuração de interface e a inserção de algumas informações sobre o sistema modelado.

Conforme ilustra a figura, a tela inicial da shell oferece um menu principal onde pode ser criado um novo modelo ou editado algum modelo já existente

Além do menu principal, a tela inicial da shell oferece também um menu secundário com algumas das funções oferecidas pelo menu principal (as mais importantes) e uma área para a criação do sistema conforme mostra a figura :

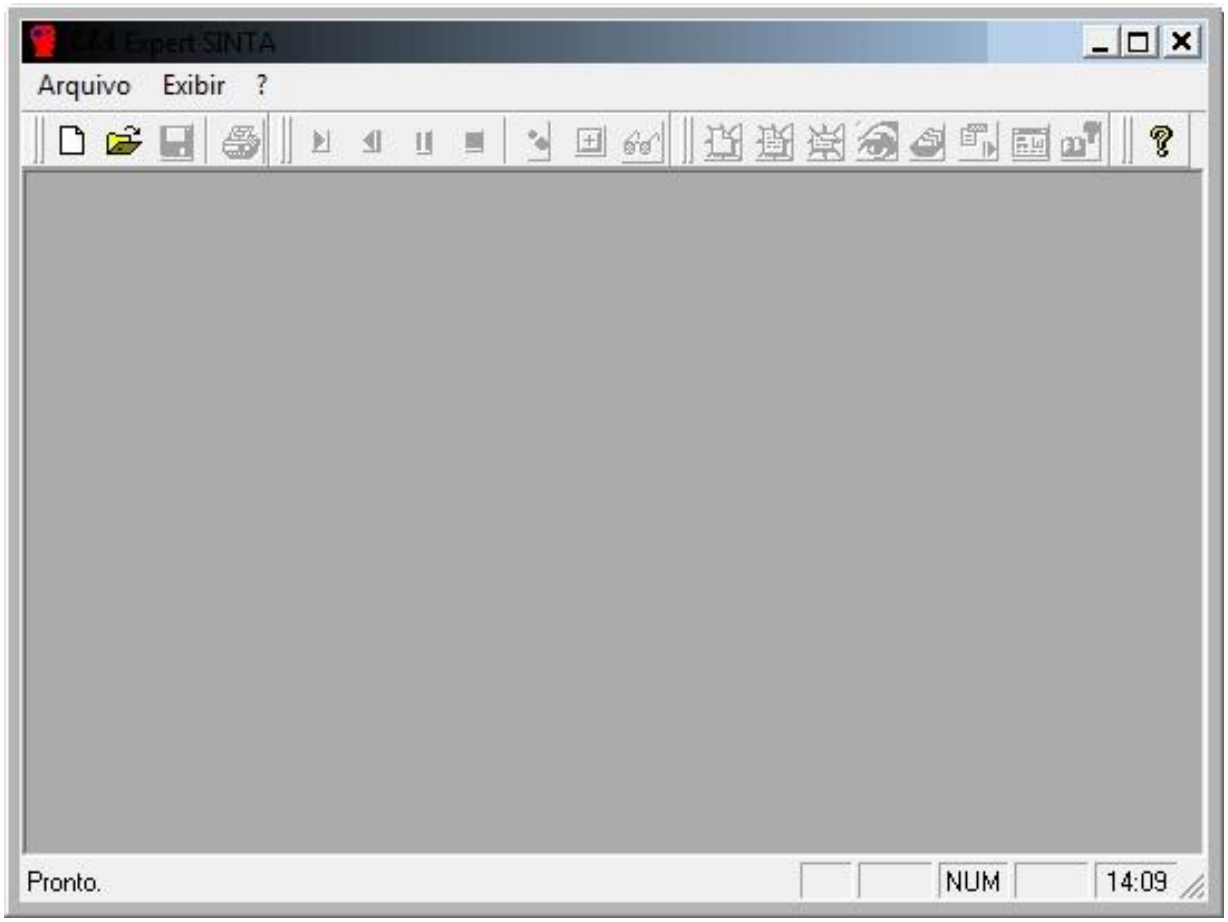


Figura 21. Tela inicial da *shell* Expert Sinta

Após se iniciar um modelo, a shell mostra uma área para a criação dos ítens do modelo, tais quais as regras, a interface, as variáveis e os objetivos, como ilustra a figura 9.

Para se inserir as regras, primeiro é necessário que sejam criadas as variáveis e suas interfaces, pois somente após definidas as variáveis é que é possível integrá-las nas regras.

A tela de inserção de regras pede os dados básicos necessários para a criação da mesma, como ilustra a figura 10.

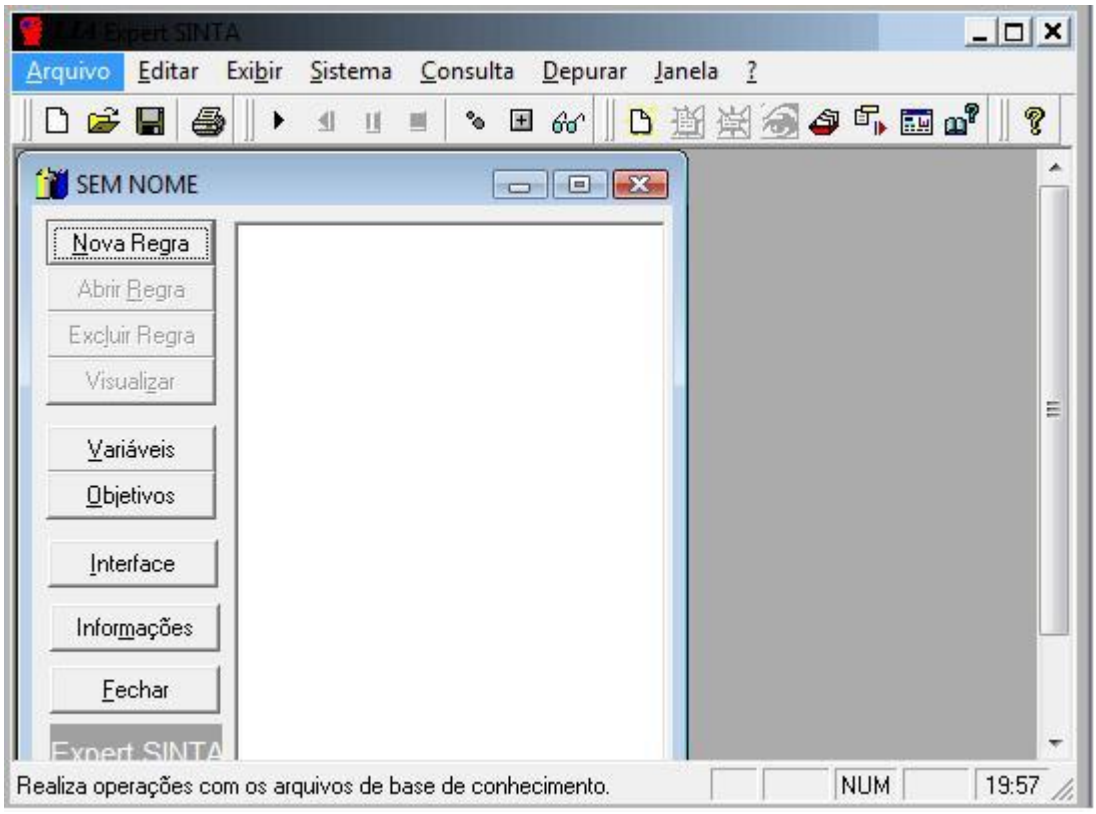


Figura 22. Tela de criação de modelo

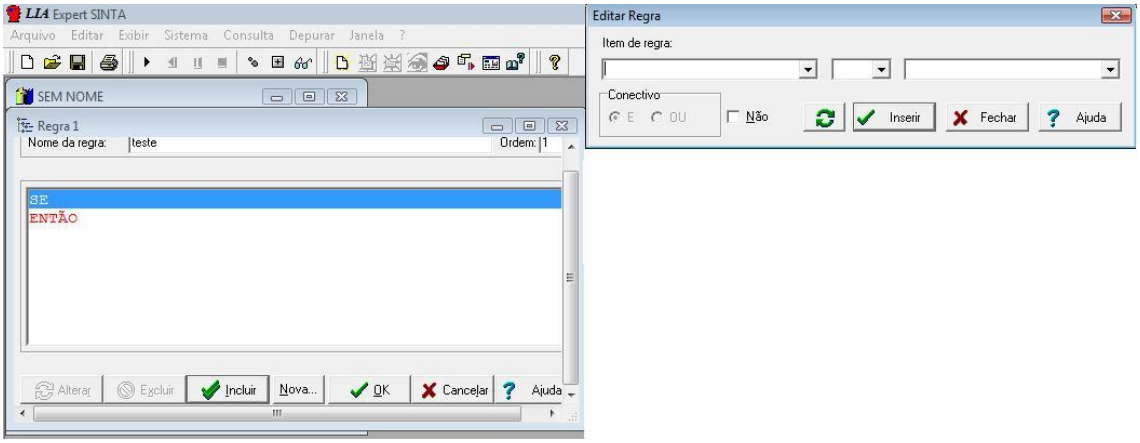


Figura 23. Tela de inserção de regras

8.2.2 O módulo de Fatores de Certeza da *Shell Pegasus*

O módulo de Fatores de certeza da *shell* Pegasus foi criado por Fernando Silvano Gonçalves, que faz parte do Grupo de Pesquisa em Inteligência Computacional Aplicada da Universidade do Extremo Sul Catarinense - UNESC - em 2008.

A *shell* oferece em sua tela inicial um menu principal, no qual são criadas as novas bases de dados, logo abaixo do mesmo existe uma área para a criação da base de dados, a qual permite a inserção de regras, variáveis, objetivos e configuração de interface, como ilustra a figura11.

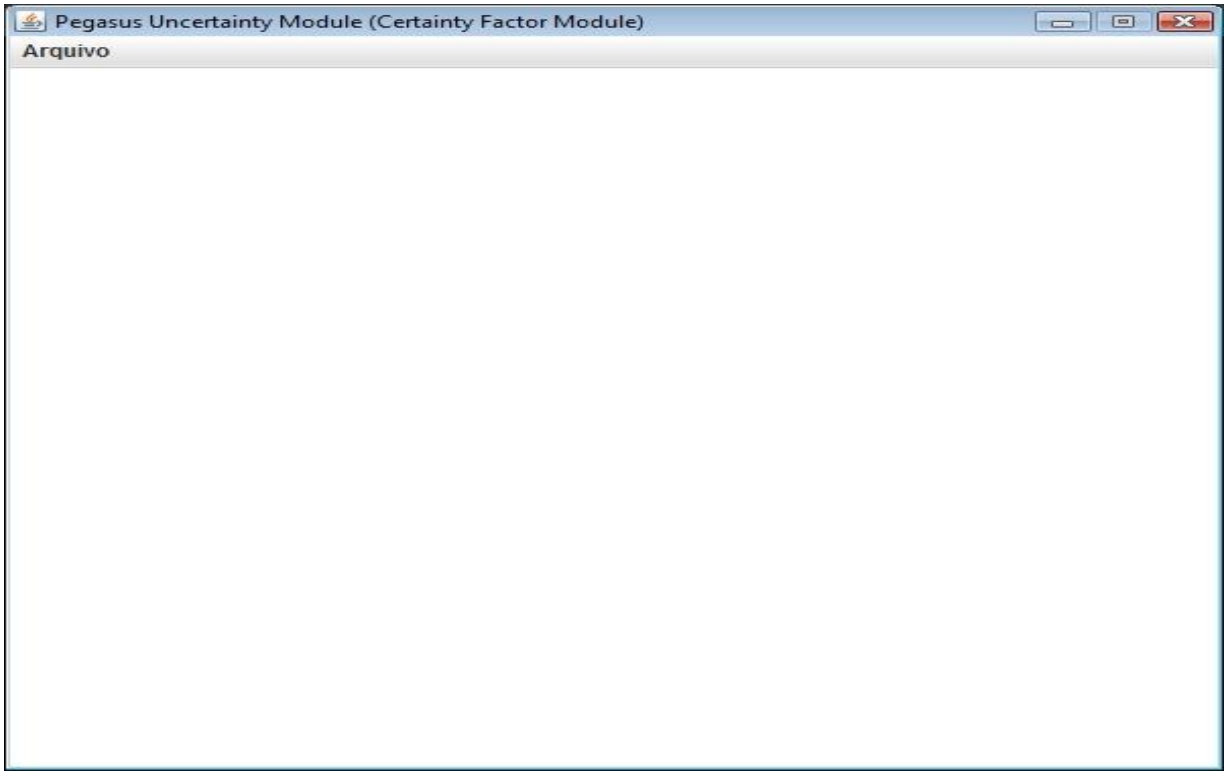


Figura 24. Tela inicial do módulo de Fatores de Certeza da Shell Pegasus

Figura 25. Tela de criação de um modelo

Para a criação da base é necessário criar as variáveis, suas respectivas interfaces e suas regras.

Para a criação de variáveis, a shell mostra uma tela onde é possível se inserir ou remover variáveis, com seus respectivos nomes, como mostra a figura 13.

Após criar as variáveis, é possível criar as regras e integrá-las as variáveis. As regras criadas são do tipo Se Então e são mostradas na figura 14.

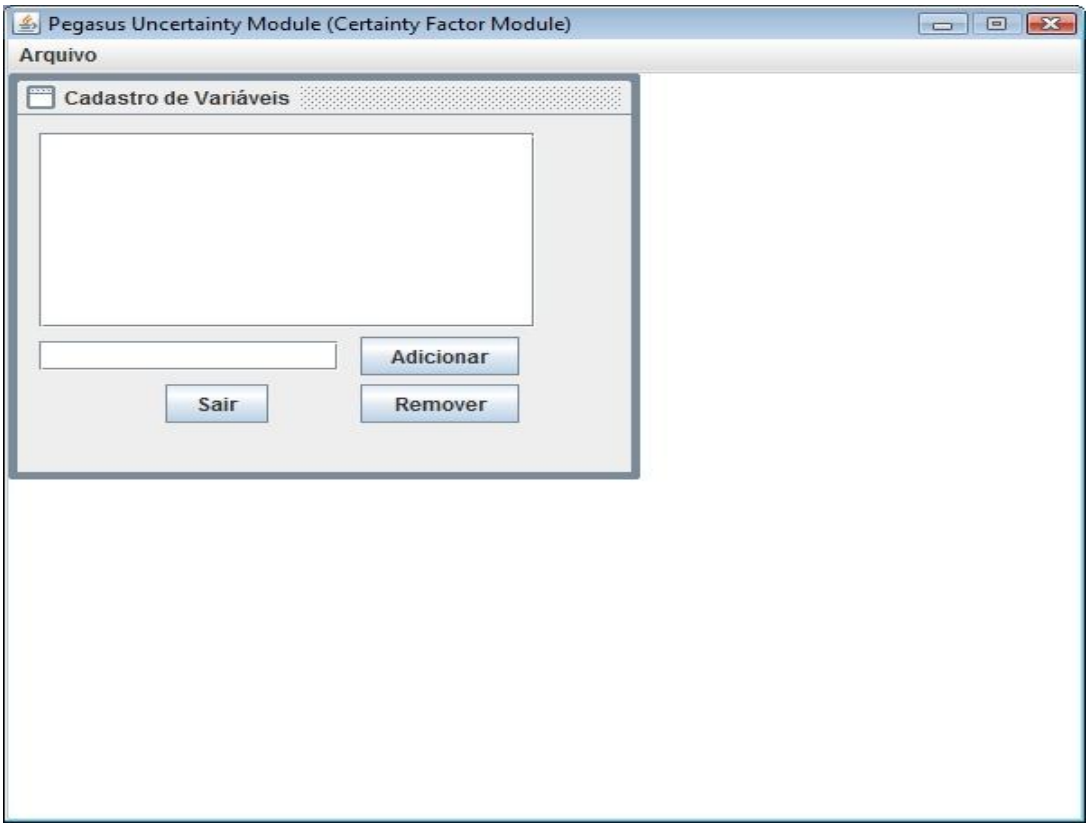


Figura 26. Tela de cadastro de variáveis

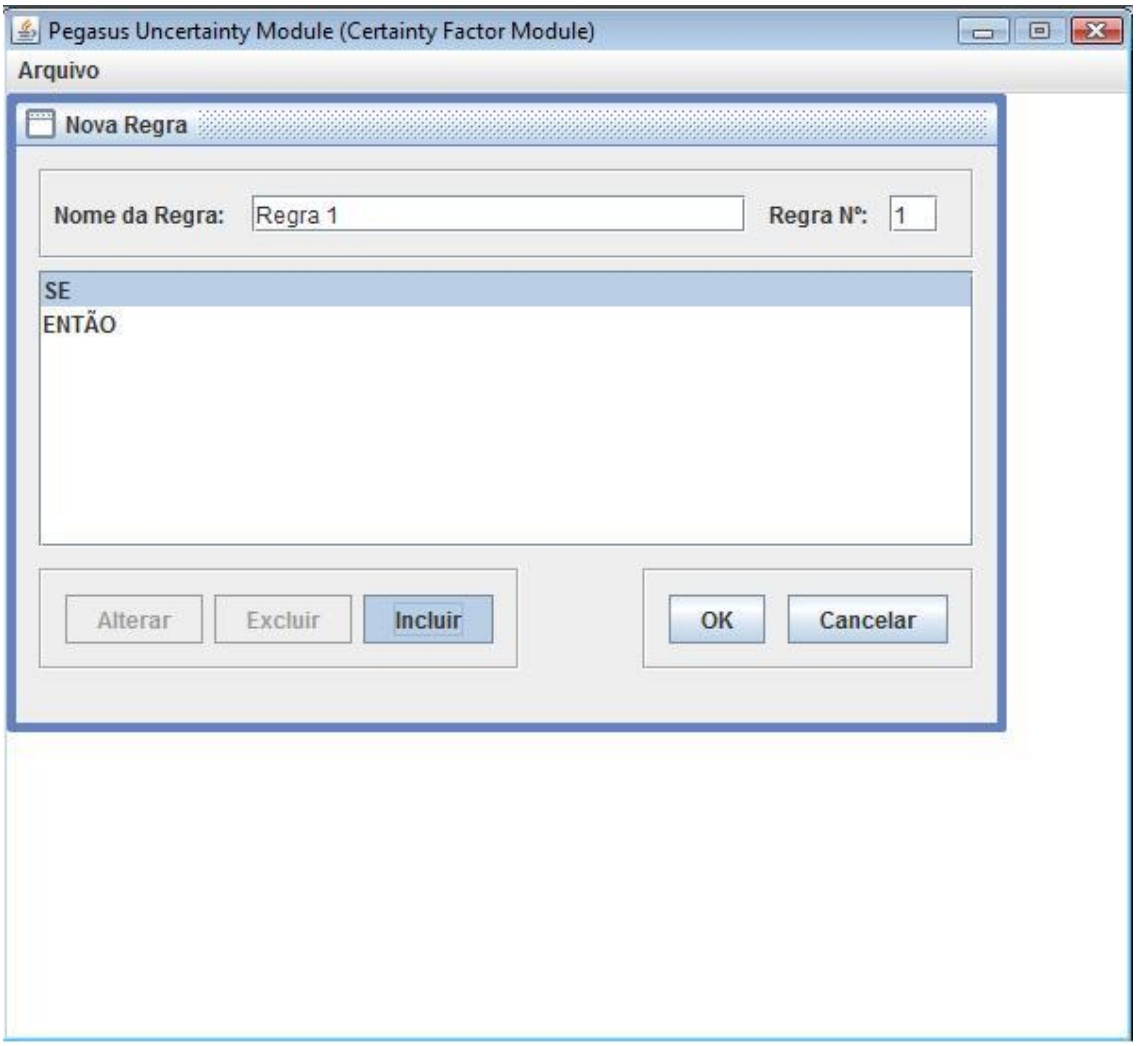


Figura 27. Tela cadastro de regras

Após o cadastro das regras e das variáveis, é possível executar o sistema, o qual carregará os dados, executará as perguntas via interface e no final mostrará o resultado da execução.

8.2.3 Comparação entre a *Shell* Expert Sinta e o módulo de Fatores de Certeza da *Shell* Pegasus

A tabela 3 ilustra as funcionalidades presentes nas duas *Shells*:

Tabela 6. Comparação entre as *shells* Expert Sinta e o módulo de Fatores de Certeza da *shell* pegasus

Funcionalidade	Shell	
	Expert Sinta	Shell Pegasus
Criação de regras	X	x
Edição de regras	X	x
Exclusão de regras	X	x
Criação de Variáveis	X	x
Edição de variáveis	X	x
Exclusão de variáveis	X	x
Criação de variáveis-objetivos	X	x
Edição de variáveis-objetivos	X	x
Exclusão de variáveis-objetivos	X	x
Criação de interface	X	x
Visualização de resultados de forma numérica	X	x
Visualização de resultados de forma gráfica		x

Fonte: Autor

8.3 O MÓDULO DE DEMPSTER-SHAFER DA SHELL PEGASUS

Esta seção traz a comparação entre a *Shell* Dempster-Shafer Engine e o módulo de Dempster-Shafer da *Shell Pegasus*

8.3.1 A *Shell* Dempster-Shafer Engine

A *Shell* Dempster-Shafer engine (DSE) foi desenvolvida por Adrian O'neil e lançada em 1999, sua licensa é free e é fácilmente achada para download na internet.

A DSE tem como função auxiliar na resolução de problemas de modelagem de incerteza que envolvam ignorância, sendo assim uma ferramenta que oferece uma interface visual para a criação das funções de crença da teoria de Dempster-Shafer.

A figura 15 mostra a tela inicial, a qual contem uma barra de menu com as opções principais, uma barra de menu secundária, um campo que mostra as hipóteses atuais (criadas ou de algum arquivo exportado) e uma área para criação de novas hipóteses.

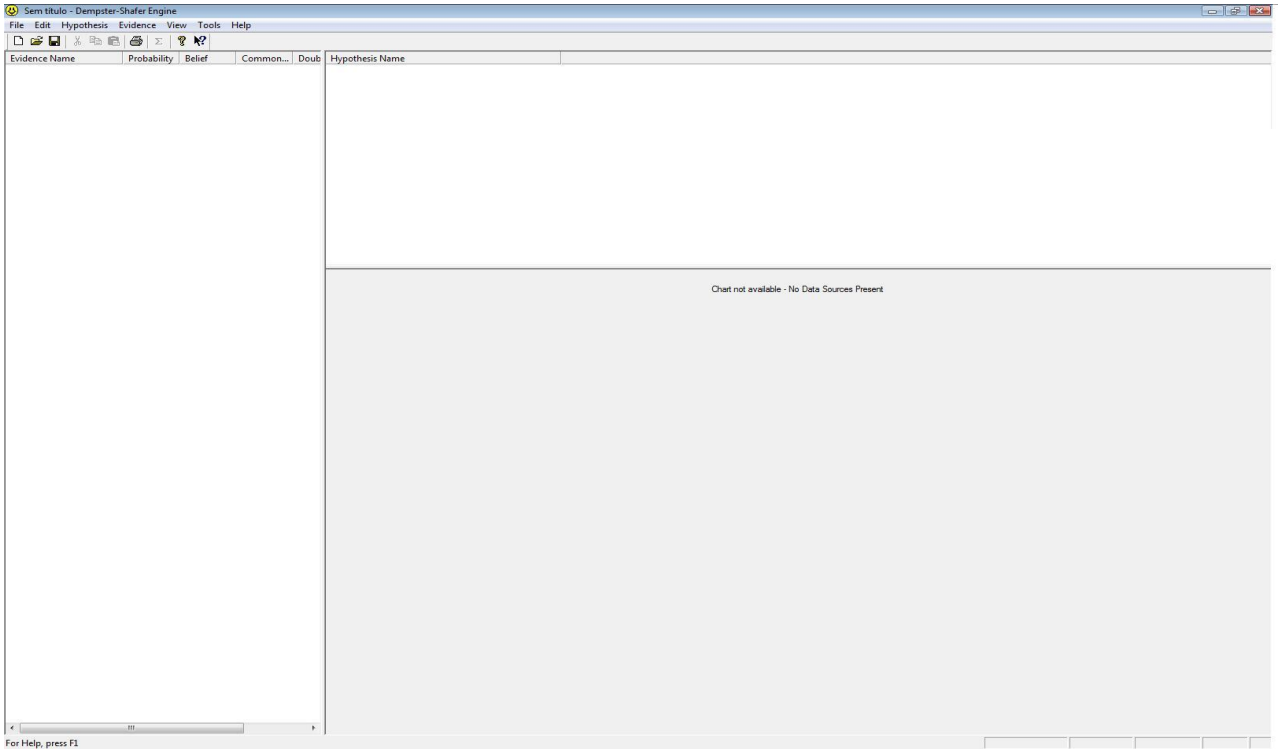


Figura 28. Tela inicial da DSE

Na sessão file do menu principal temos algumas opções relativas ao modelo que estamos construindo como: abrir um arquivo, criar um novo arquivo, imprimir o modelo, acessar propriedades do modelo tais quais nome, autor e descrição.

Existe também a opção de criação e impressão de um documento preview do modelo, esse documento que é mostrado na figura 16 contém os dados do modelo como nome, hipóteses e outros dados.

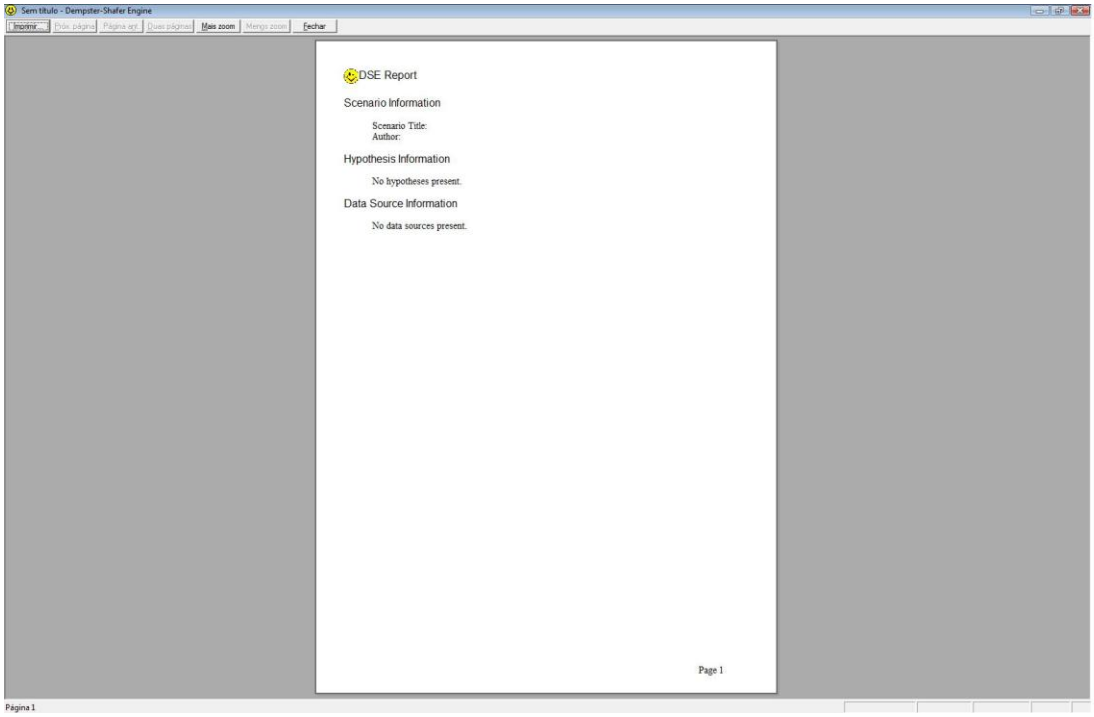


Figura 29. Preview do modelo

A sessão Hypotesis do menu principal trata de todas as ações que podem ser feitas em relação as hipóteses do modelo, tais como:criação de uma nova hipótese, edição de uma hipótese, exclusão e edição de relacionamentos.

Conforme mostra a figura 17, ao se criar uma nova hipótese é necessária a inserção de alguns dados básicos sobre ela como nome e descrição.

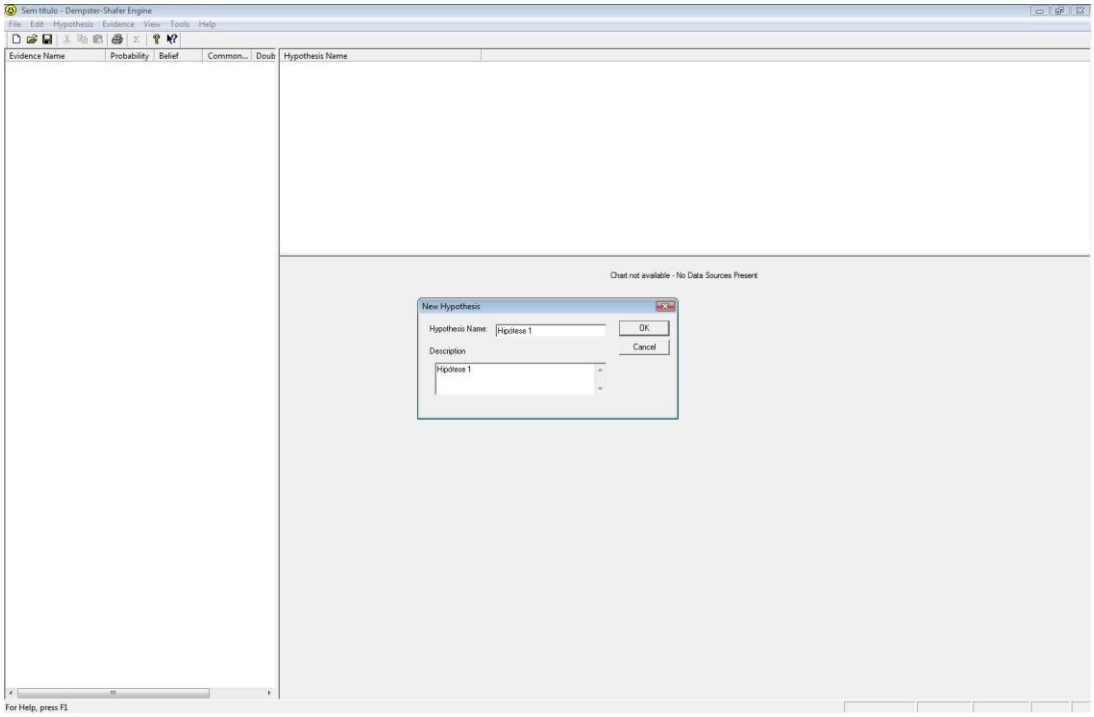


Figura 30. Criação de uma nova hipótese

A sessão Evidence do menu principal oferece as mesmas opções da sessão Hypotesis ao usuário, ou seja, criar, editar e excluir alguma evidência do modelo.

Como é mostrado na figura 18, para a criação de uma nova evidência são necessários alguns dados como nome e descrição. É necessário também escolher á qual hipótese essa evidência pertence e o grau de probabilidade dela que varia de 0 a 1.

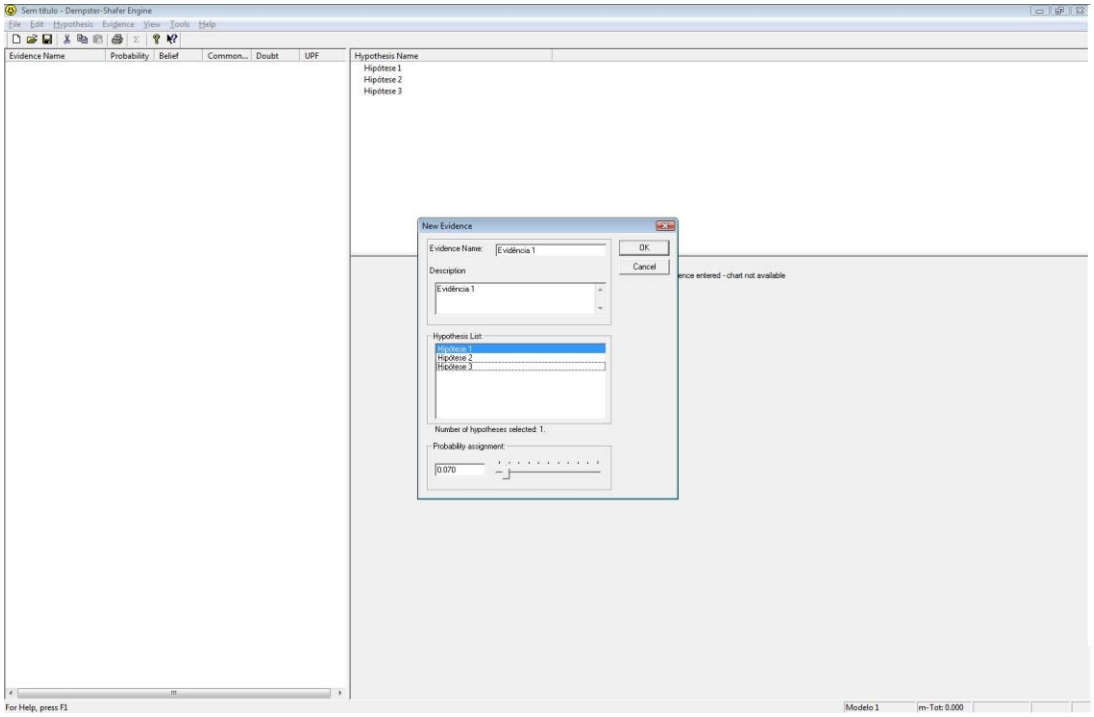


Figura 31. Inserção de uma nova evidência no modelo

Após a inserção dos dados do modelo, a *Shell* mostra em sua tela inicial os dados que foram inseridos, conforme a figura 19.

Na área A são exibidas as evidências, na área B são exibidas as hipóteses e na área C é mostrado um gráfico ilustrando as evidências cadastradas.

Figura 32. Tela inicial da DSE com os dados inseridos

Na sessão Tools, a shell oferece a opção para combinação das evidências (que pode ser feita através da tecla F5), um esquema de normalização das probabilidades e a opção de exportação do modelo no formato .csv.

O menu também oferece uma sessão de ajuda - Help - a qual contém uma área que fala sobre o software e outra que oferece alguns tópicos de aprendizagem do software.

A sessão View oferece algumas opções de visualização do menu principal.

O menu secundário da shell oferece as mesmas opções do menu principal, mas de um modo mais intuitivo.

8.3.2 O MÓDULO DE DEMPSTER-SHAFER DA SHELL PEGASUS

O módulo da teoria de evidência de Dempster-Shafer da Shell pegasus oferece em suam tela principal opções que tangem tanto as hipóteses quanto as evidências do modelo.

Através da shell é possível inserir e remover hióteses, inserir e remover as evidências relativas a uma dada hipótese, cadastrar o valor de crença do(s) especialista(s) nessa hipótese e combinar as evidências, assim como mostra a figura 20 :

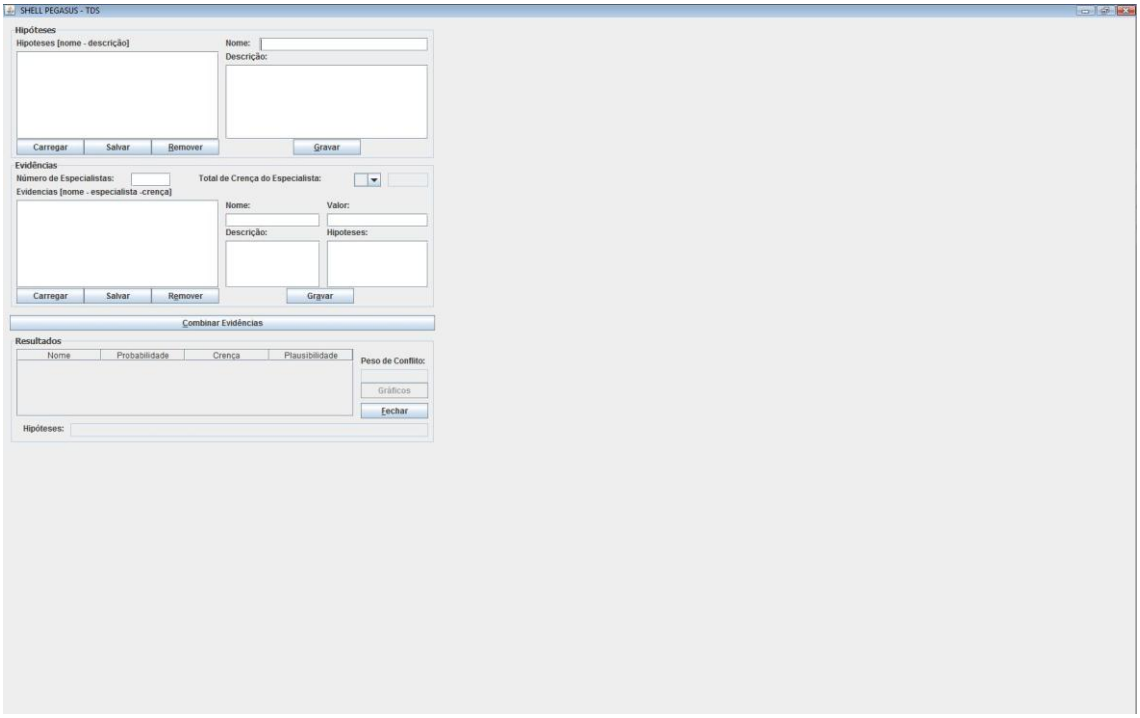


Figura 33. Tela principal do módulo de Dempster-Shafer da shell Pegasus

A shell oferece também uma área para exibição do resultado da combinação das hipóteses, além de oferecer a opção de mostrar os resultados em forma de gráfico como mostram as figuras 21 e 22:

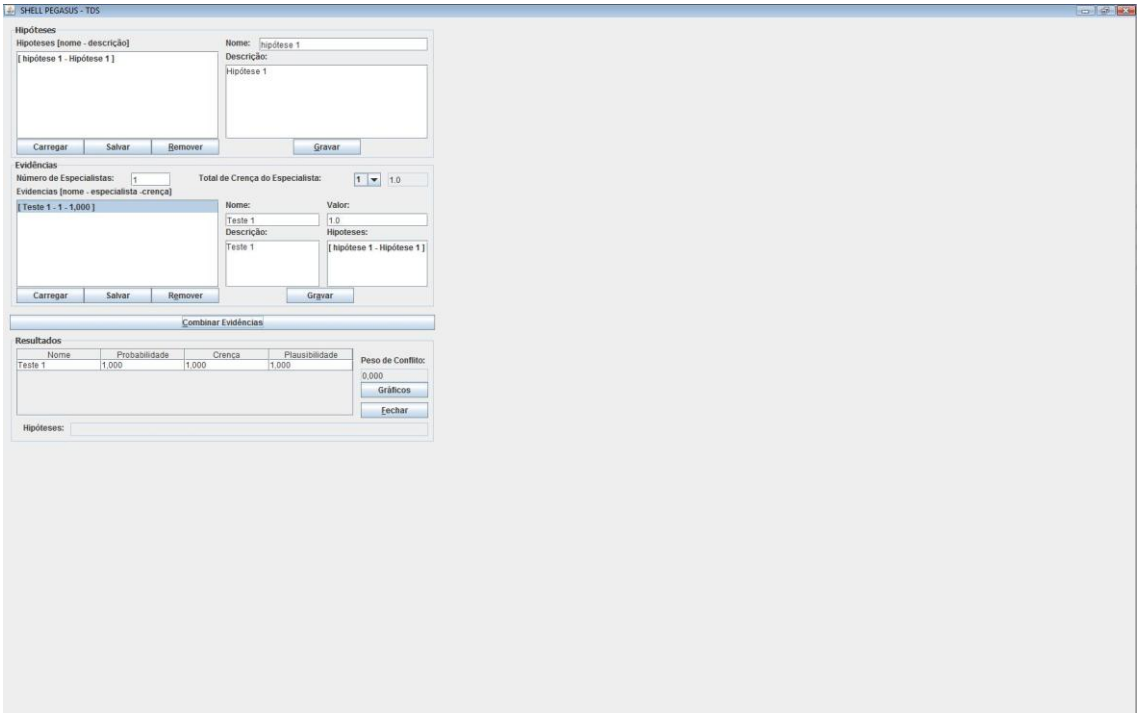


Figura 34. Exibição do resultado da combinação das hipóteses

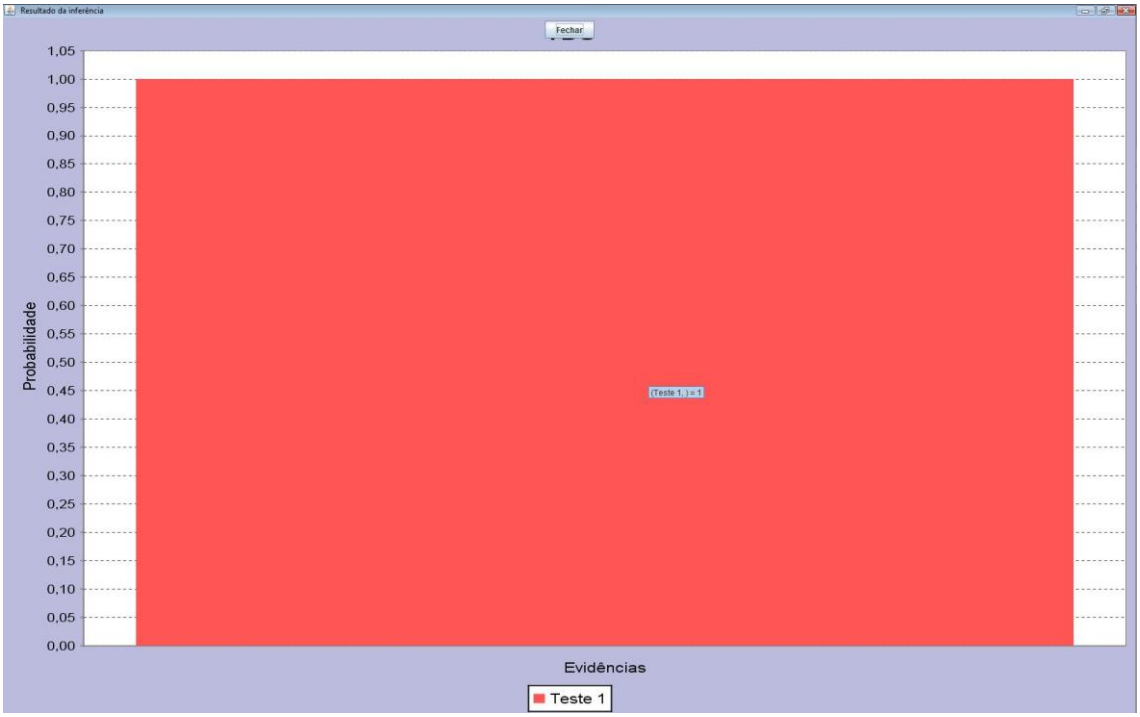


Figura 35. Modo de exibição gráfico do módulo

8.3.3 COMPARAÇÃO ENTRE A SHELL DEMPSTER-SHAFER ENGINE E O MÓDULO DE DEMPSTER-SHAFER DA SHELL PEGASUS

A tabela 4 ilustra a comparação entre as duas shells:

Tabela 7. Comparação de funcionalidades entre a *shell* Dempster-Shafer Engine e o módulo de Dempster-Shafer da *shell* Pegasus

Funcionalidade	Shell	
	Dempster-shafer engine	Shell Pegasus
Criação de hipóteses	X	x
Edição de hipóteses	X	x
Exclusão de hipóteses	X	x
Criação de Evidências	X	x
Edição de Evidências	X	x
Exclusão de Evidências	X	x
Exclusão de variáveis-objetivos	X	x

Fonte: Autor

9. TRABALHOS CORRELATOS

Os trabalhos apresentados nesse capítulo fazem referência as técnicas de modelagem de incerteza e suas aplicações, sendo descritos seus objetivos, resultados entre outros.

Alguns objetivos abordados são: aplicação de IA incerteza na área de diagnóstico médico, controle de riscos em economia, fusão de sensores em robôs autônomos e detecção de falhas em redes sem fio.

9.1 APLICAÇÕES

As aplicações descritas nesse capítulo utilizam as técnicas de modelagem de incerteza para auxiliar na solução de alguns problemas, grande parte na área médica.

São 3 pesquisas nacionais, duas na área do diagnóstico médico e uma pesquisa na área de economia.

9.1.1 Construção de uma rede Bayesiana aplicada ao diagnóstico de doenças cardíacas

Essa dissertação de mestrado do curso de Engenharia Mecatrônica foi desenvolvida na Escola Politécnica da Universidade de São Paulo em 2005, apresenta a construção de uma Rede Bayesiana aplicada aos diagnósticos de doenças cardíacas (SAHEKI, 2005).

O objetivo do sistema construído foi auxiliar o diagnóstico de doenças cardíacas, especificamente em pessoas com dificuldades respiratórias. De um modo

geral, a rede foi modelada para um diagnóstico que se baseia nos mecanismos (doenças e pré-disposições) que possam levar o indivíduo a ter problema de insuficiência cardíaca (SAHEKI, 2005).

Como pode ser observado na figura 4, a Rede Bayesiana foi contruída de modo que os nós representam características que podem configurar uma pré-disposição as doenças cardíacas, por exemplo, mal de chagas, Fumo e etc (SAHEKI, 2005).

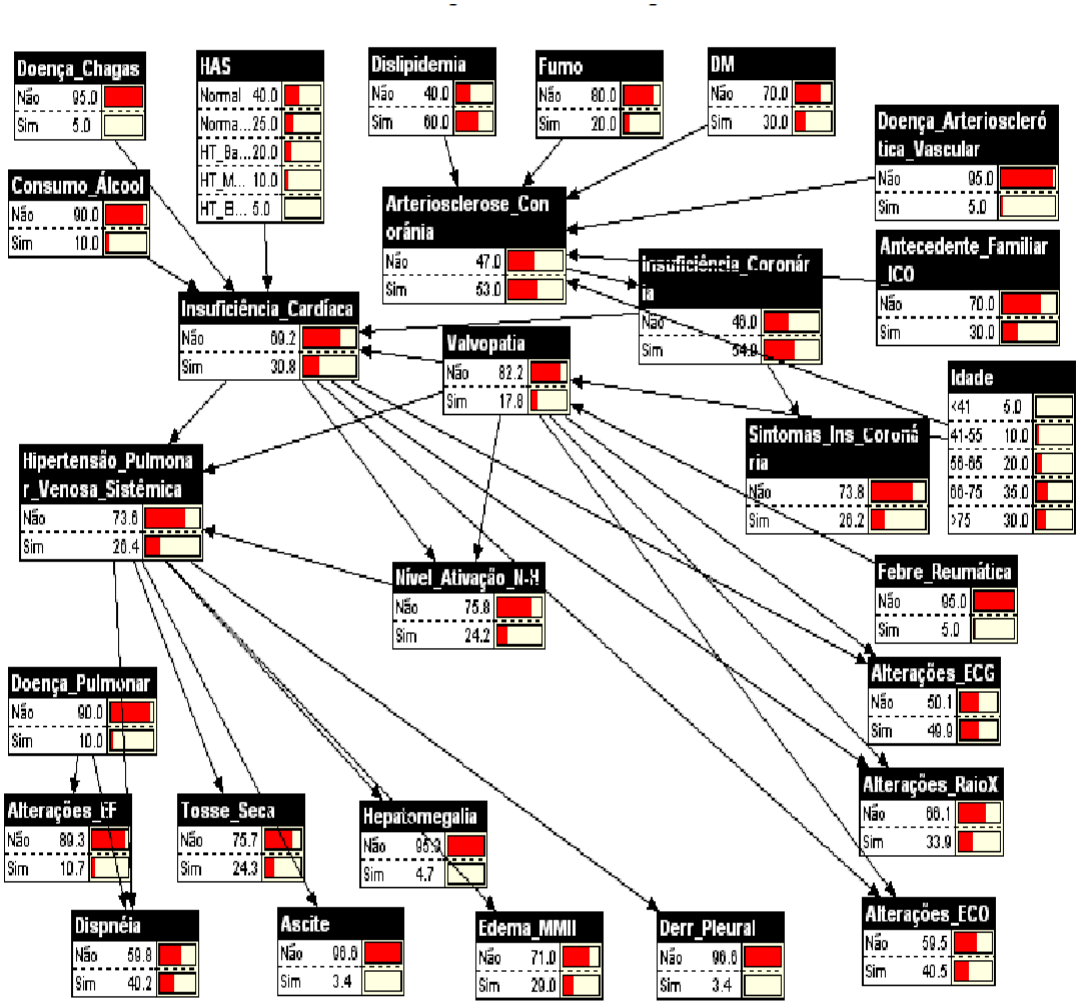


Figura 36. A Rede Bayesiana construída no sistema
Fonte: Saheky (2005)

De acordo com o autor, após uma série de testes feitos com a presença do especialista, os resultados obtidos foram considerados adequados (SAHEKI, 2005).

9.1.2 Aplicando a teoria da Evidência na detecção de anomalias

O presente artigo apresenta uma ferramenta de fusão de dados baseada na teoria da evidência de Dempster-Shafer para a detecção de tráfego anômalo em uma rede (LINS; FEITOSA; SADOK, 2009).

Para efetuar os testes foi usada uma ferramenta chamada ADS-Fusion, a qual foi desenvolvida em JAVA e tem como função analisar anomalias em uma rede utilizando a técnica de fusão de dados utilizando TDS. Essa ferramenta se difere das outras pela capacidade de gerar inferências a partir da fusão de dados com um grau de certeza muito maior do que a gerada por outras ferramentas como a ferramenta Profiling e a TCP Model.(LINS; FEITOSA; SADOK, 2009).

Após uma série de testes feitos na rede, foi constatado que a ferramenta ADS-Fusion traz resultados mais concisos na avaliação de ataques do tipo TCP SYN Flood, SPAM e TCP SYN de baixa carga (LINS; FEITOSA; SADOK, 2009).

9.1.3 Building and Testing the SHYSTER-MYCIN Hybrid Legal Expert System

Shyster-MYCIN é um sistema especialista híbrido, aplicado a área jurídica, que foi criado pela combinação de raciocínio baseado em regras e raciocínio baseado em casos a fim de se criar um sistema para análise de casos onde se aplicam leis autorais (CALLAGHAN; POPPLE; MCCREATH, 2003).

Para fazer os testes, foi usada uma base de dados com algumas questões relativas as leis de direito autoral, essas questões foram aplicadas dentro do sistema e suas respostas foram analisadas (CALLAGHAN; POPPLE; MCCREATH, 2003).

As conclusões mostraram que os resultados alcançados com a utilização do sistema são aceitáveis, e que o sistema possui uma capacidade ampla de resolução das questões relativas as leis de cópia. O sistema possui também uma base de dados completa (CALLAGHAN; POPPLE; MCCREATH, 2003).

10. CONCLUSÃO

Com essa pesquisa, pode-se verificar como a IA é útil em nosso meio, o quão complexo são os problemas que ela lida e como ela está disseminada em todas as áreas da ciência.

Pode-se perceber também como as Shells são fatores importantes nas pesquisas na área de Modelagem de Incerteza, haja vista que a carga matemática é altíssima e de implementação complexa.

A Shell Pegasus Uncertainity Modeling sem dúvida alguma é uma ferramenta completa, pois alia três das quatro principais técnicas de Modelagem de Incerteza, o que faz com que a mesma acabe preenchendo uma lacuna que ainda não foi preenchida no âmbito de Shells.

A Engenharia de Software possui um papel crucial na nossa realidade, pois com todo o volume de software que está sendo produzido, e devido à alta complexidade que o mesmo emprega em seu desenvolvimento é totalmente necessário se criar metodologias científicas que tratem de garantir a qualidade no desenvolvimento e no uso do produto final.

Os objetivos do trabalho foram alcançados, pois além de fazer a análise de desempenho da Shell foi feito também uma verificação de qualidade da mesma, apesar de que este não era o foco principal do trabalho, portanto foi realizada de modo breve e simples.

As maiores dificuldades encontradas no desenvolvimento desse trabalho foi o fato de que qualidade e desempenho são termos muito subjetivos, pois não existem métricas numéricas para exprimir qualidade, fato esse que obriga o analista a ter que

gerar uma interpretação pessoal de alguns desses itens baseada apenas em seu conhecimento e experiência.

Os resultados obtidos foram satisfatórios e com certeza trarão uma boa base para os que se interessarem em continuar essas análises.

Para os trabalhos futuros poderá ser explorada com um pouco mais de detalhes a área que tange a qualidade de software da *Shell*.

11. BIBLIOGRAFIA

ALMEIDA, P. E.; EVSUKOFF, A. G. Sistemas Fuzzy. In: REZENDE, S.O.(Org.) Sistemas Inteligentes: fundamentos e aplicações. Barrueri: Manole, 2005. p. 169-201.

ANTUNES, Jerônimo. **Modelo de avaliação de risco de controle utilizando a lógica nebulosa.** 2004. 162 f. Dissertação (Título de Doutorado) - Departamento de Contabilidade e Atuaria, Universidade de São Paulo, São Paulo, 2004.

CALLAGHAN, Thomas A; POPPLE, James; MCCREATH, Eric **Building and Testing the SHYSTER-MYCIN Hybrid Legal Expert System** Australian National University 2003. Diponível em < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.7.1228&rep=rep1&type=pd> Acesso em: 19 de Junho de 2010

CARTER, Nicholas. Arquitetura de Computadores - Coleção Schaum. Porto Alegre: Bookman, 2003. 240p.

CASTILHO, Enrique; GUTIÉRREZ, José Manuel; HADI, Ali S. Sistemas Expertos y Modelos de Redes Probabilísticas. Madrid: Academia Española de Ingeniería, 1998.

CHARNIAK, EUGENE. “Bayesians Networks without Tears”. IA Magazine, 1991.

COSTA, Ernesto; SIMÕES, Anabela. **Inteligência artificial:** fundamentos e aplicações. Lisboa: FCA, 2004.

COWELL, Robert G. (...[et al.]). Probabilistic networks and expert systems. New York, USA: Springer, 1999.

COX, Earl. Fuzzy modeling and genetic algorithms for data mining and exploration. California: Morgan Kaufmann, c2005.

EJ-TECHNOLOGIES. JProfiler Trial Dowload. Disponível em http://www.ej-technologies.com/download/jprofiler/trial.php. Recuperado em 02/04/2007.

GEORGESCU, Irina. Fuzzy choice functions: a revealed preference approach. New York: Springer, c2007.

HENNESY, John L.; PATTERSON, David A. Arquitetura de computadores: uma abordagem quantitativa. 3 . ed. Rio de Janeiro: Campus, 2003. 827 p.

JENSEN, Finn V. Bayesian networks and decision graphs. New York, USA: Springer, c2001.

KLIR, George J.; YUAN, Bo. Fuzzy sets and fuzzy logic; theory and applications. New Jersey: Prentice Hall, 1995.

KOEHLER, Cristiane. **Desenvolvimento de um Sistema Inteligente para Apoio à Decisão em Saúde** Universidade de Caxias do Sul. 2005. Diponível em < http://www.informedicajournal.org/a1n2/files/papers_csbis/koehler1.pdf> Acesso em: 15 de Outubro de 2009

KORB, Kevin B.; NICHOLSON, Ann E. Bayesian artificial intelligence. Florida: Chapman & Hall, 2004.

LINS, Bruno F O; FEITOSA, Eduardo L; SADOK, Djamel F H. Aplicando a teoria da evidência na detecção de anomalias. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 27., 2009, Recife. **Aplicando a teoria da evidência na detecção de anomalias.**

MARQUES, Ligeiro R.; DUTRA, Inês. **Redes Bayesianas: o que são, para que servem**: **algoritmos e exemplos de aplicações**. Disponível em

<http://www.cos.ufrj.br/~ines/courses/cos740/leila/cos740/Bayesianas.pdf> Acesso em: 15 de Outubro de 2009

MOURA, Jacques Philippe Sauvé; GIOZZA, William Ferreira; ARAÚJO, José Fábio Marinho de. **Redes locais de computadores:** protocolos de alto nível e avaliação de desempenho. São Paulo: Mcgraw-hill, 1986.

NEAPOLITAN, Richard E. Learning bayesian networks. New Jersey: Pearson Prentice Hall, c2004.

NICOLETTI, Maria do Carmo; CAMARGO, Heloisa de Arruda. Fundamentos da teoria de conjuntos fuzzy. São Carlos, SP: EDUFSCAR, 2004.

PEZZÉ, MAURO ; YOUNG, MICHAEL **Teste e análise de software: processo, princípios e técnicas** Porto Alegre: Bookman, 2008. 512 p.

PRESSMAN, S. ROGER **Engenharia de Software** São Paulo: Makron Books, 1995. 1063 p.

RIOS, Emerson. Moreira, Trayahú. Teste de Software, Rio de Janeiro: Alta Books, 1996.

ROSS, Timothy J. Fuzzy logic with engineering applications. 2nd ed New Jersey: John Wiley & Sons, c2004.

RUSSEL, S. J. ; NORVIG, P. Inteligência Artificial . Rio de Janeiro: Elsevier, 2004

SAHEKI, Hideaki André. Construção de uma Rede Bayesiana Aplicada ao Diagnóstico de Doenças Cardíacas Universidade de São Paulo. Diponível em

<http://www.poli.usp.br/p/fabio.cozman/Publications/Article/saheki-sharovsky-cozman-coupe-enia2003.pdf> Acesso em: 16 de Outubro de 2009

SALICONE, Simona. Measurement uncertainty: an approach via the mathematical theory of evidence. New York: Springer, c2007.

SILVANO, Fernando. O formalismo dos Fatores de Certeza na shell Pegasus

Uncertainity

SOMMERVILLE, I. Engenharia de software. 6ª ed. São Paulo: Addison Wesley, 2003.

SPIEGELHALTER, D. J.; ABRAMS, K. R.; MYLES, Jonathan P. Bayesian approaches to clinical trials and health care evaluation. New Jersey: Wiley, c2004.

SUN Microsystems. Tecnologia Java: breve histórico da tecnologia Java. Disponível em
http://www.java.com/pt_BR/about/. Recuperado em 16/04/2007.

TARONI, Franco. Bayesian networks and probabilistic inference in forensis science. New Jersey: John Wiley & Sons, c2006.

TOSCANI, Laira Vieira; VELOSO, Paulo A S. **Complexidade de algoritmos.** 2. ed. Porto Alegre: Sagra Luzzato, 1993. 264 p.

WANG, Li-Xin. A couse in fuzzy systems and control. London: Prentice Hall, 1997.

WEBER, Leo; KLEIN, Pedro Antonio Trierweiler. Aplicação da lógica Fuzzy em software e hardware. Canoas, RS: ULBRA - Universidade Luterana do Brasil, 2003.

WILLIAMSON, Jon. Bayesian nets and causality: philosophical and computational foundations. New York, USA: Oxford University Press, 2005.

YAGER, Ronald R; KACPRZYK, Janusz; FEDRIZZI, Mario. Advances in the dempster-shafer theory of evidence. New York: John Wiley & Sons, 1994.

ZIVIANI, Nivio. **Projeto de Algoritmos:** com implementações em pascal e c. Campinas: Pioneira, 1993.

APÊNDICE A - DESCRIÇÃO DAS REDES BAYESIANAS UTILIZADAS NOS TESTES

A tabela 8 traz uma descrição das redes bayesianas utilizadas nos testes da Shell Pegasus:

tabela 8. Descrição das Redes Bayesianas utilizadas nos testes da Shell Pegasus

Modelo	Nós	Estados por nó	Estados
Modelo 1	2	2	estado_1, estado_2
Modelo 2	4	2	estado_1, estado_2
Modelo 3	6	2	estado_1, estado_2

APÊNDICE B - DESCRIÇÃO DAS REGRAS UTILIZADAS NO TESTE DO MÓDULO DE FATORES DE CERTEZA DA SHELL PEGASUS

A tabela 9 ilustra as regras usadas nos testes de cada modelo:

Tabela 9. Regras utilizadas nos testes da Shell Pegasus

Modelo	Variáveis	Descrição	Regra	FC
Modelo 1	3	var1, var2, var_ob	se var1 e var 2 então var_obj	100%
Modelo 2	5	var1, var2, var3, var4, var_ob	se var1 e var2 e var3 e var 4 então var_obj	100%
Modelo 3	7	var1, var2, var3, var4, var5, var6, var_ob	se var1 e var2 e var3 e var 4 e var5 e var6 então var_obj	100%

APÊNDICE C - MODELOS UTILIZADOS NO TESTE DO MÓDULO DE FATORES DE CERTEZA DA SHELL PEGASUS

A tabela 10 mostra os modelos utilizados para os testes no módulo de Fatores de Certeza da Shell Pegasus:

Tabela 10. Modelos utilizados nos testes do módulo de Fatores de Certeza da Shell Pegasus

Modelo	Evidências	Hipóteses	Especialistas	Crença
Modelo 1	4	2	2	0,5
Modelo 2	6	4	2	0,5
Modelo 3	8	6	2	0,5

APÊNDICE D - ARTIGO

Análise de desempenho da Shell Pegasus Uncertainty Modeling

Diogo Cesar Bif[1], Priscyla Waleska Targino de Azevedo Simões[1], Kristian Madeira[1]

[1]Departamento de Ciência da Computação
Universidade do Extremo Sul Catarinense (UNESC) – Criciúma, SC – Brazil

diogo_bif@yahoo.com.br, {pry, kma}@unesc.net

Abstract. *This article describes the performance evaluation of Shell Pegasus Uncetarinty modeling, including the methodologies used in evaluating the same and the results obtained through this evaluation.*

Resumo. *O presente artigo descreve a avaliação de desempenho da Shell de modelagem de Incerteza Pegasus Uncetarinty modeling, compreendendo as metodologias usadas na avaliação da mesma e os resultados obtidos através dessa avaliação.*

1. Introdução

Muitos dos problemas que profissionais de várias áreas precisam resolver são complexos e necessitam de conhecimentos que normalmente são incertos. Para que se possa encontrar possíveis soluções para esses problemas, podem ser utilizadas técnicas de IA.

Nos diversos domínios de aplicação existentes em que a incerteza é presente, é comum que as informações não estejam claras ou completas, dando margem a várias soluções para o problema. Assim, para que uma determinada aplicação possa disponibilizar um resultado mais confiável, é necessária a modelagem adequada da forma de incerteza apresentada.

Algumas técnicas utilizadas neste contexto buscam reduzir a dúvida gerada entre as hipóteses permitindo uma maior crença do especialista em uma das apresentadas, sendo que, por exemplo, quanto maior o grau de confiança na hipótese, maior é a relevância da mesma reduzindo a possibilidade de múltiplos resultados (COSTA; SIMÕES, 2004).

Dentro dos algoritmos de IA que tratam do contexto de incerteza da informação três formalismos distintos foram abordados na *Shell Pegasus Uncertainty Modeling*: Redes Bayesianas, Fatores de Certeza e Teoria da Evidência de Dempster-Shafer.

Conforme Nassar (2007) descreve, não basta somente desenvolver uma aplicação ou ferramenta baseada na IA, é de fundamental importância que o desempenho dos sistemas inteligentes sejam estatisticamente avaliados considerando

que estes pretendem emular o comportamento de um especialista em um domínio de aplicação. Para determinados tipos de sistemas, o software pode fornecer as funções desejadas, mas não satisfazer os requisitos de desempenho, diminuindo assim a produtividade do software ou tornando necessário o processamento maior que o necessário para executá-lo.

A avaliação de desempenho é projetada para testar essa característica do software durante a execução no contexto de um sistema integrado, ocorrendo ao longo de todos os passos do processo de teste. O desempenho de módulos individuais pode ser avaliado à medida que testes são conduzidos, porém o desempenho do sistema como um todo - ou o verdadeiro desempenho do sistema - só pode ser avaliado após a integração plena de todos os seus elementos (PRESSMAN, 2006).

2. Análise de desempenho da *Shell Pegasus Uncertainty Modeling*

Define-se por desempenho de software o conjunto de requisitos não-funcionais que são relacionados á velocidade de operação ou as restrições de uso de um sistema. O desempenho computacional descreve a velocidade com a qual um determinado sistema pode executar um programa ou um conjunto de programas (CARTER, 2003, p.12), porém dentro do contexto de desempenho existem diferentes tipos de requisitos que podem ser analisados e especificados,são eles: Tempo de resposta, Throughput e Temporização.

Tempo de resposta é um requisito que especifica um tempo aceitável para que uma operação seja concluída no sistema, Throughput especifica a quantidade de dados que o sistema precisa processar dada uma quantidade de tempo e a Temporização indica quanto tempo o sistema deve levar para processar uma entrada advinda de algum sensor, antes que o próprio sistema a sobrescreva.

Apesar do tempo ser considerado a única medida de desempenho confiável e coerente quando fazemos uma medição de tempo de execução precisamos levar em consideração todos os fatores que influenciarão na execução do sistema. Somente com uma análise completa do ambiente e dos tempos é que podemos afirmar se um sistema pode ser considerado de alto desempenho (HENESSY; PATTERSON, 2003).

2.1. Metodologia de análise

A metodologia utilizada para a análise de desempenho foi baseada no fator tempo.

Utilizando-se a ferramenta de análise de desempenho JProfiler 6.02, foram realizadas inferências em cada módulo da shell documentando-se o tempo de execução de cada inferência para uma análise posterior.

A fim de se analisar a taxa de crescimento do tempo de execução da *Shell*, as inferências foram divididas em sessões, cada qual com um volume de dados crescente.

Após o fim dos testes, os tempos de execução foram catalogados, obtendo-se a média do tempo com cada volume de dados. Após a obtenção das médias, esses tempos foram inseridos em um gráfico de dispersão e neles foram aplicados os testes de regressão e de coeficiente de determinação, afim de se obter o tipo de crescimento e a taxa de crescimento dos mesmos.

2.2. Ferramenta de teste

Para a análise e obtenção dos tempos de execução das inferências, foi utilizada a ferramenta de teste JProfiler em sua versão 6.02.

Suas principais vantagens são ter uma licença gratuita por 10 dias, uma interface de fácil utilização, mostrar resultados em tempo integral, ter um baixo consumo de memória e poder ser integrada em várias IDEs. Ela é compatível também com ambientes tanto 32 quanto 64 bits.

A ferramenta também mostra o consumo de todas as classes e métodos da aplicação, e oferece no final de cada sessão os métodos que tiveram sobrecarga de chamados, auxiliando assim o desenvolvedor a desenvolver um código mais eficiente.

2.3. Ambiente de execução dos testes

Para a análise e obtenção dos tempos de execução das inferências, foi utilizada a ferramenta de teste JProfiler em sua versão 6.02.

Suas principais vantagens são ter uma licença gratuita por 10 dias, uma interface de fácil utilização, mostrar resultados em tempo integral, ter um baixo consumo de memória e poder ser integrada em várias IDEs. Ela é compatível também com ambientes tanto 32 quanto 64 bits.

A ferramenta também mostra o consumo de todas as classes e métodos da aplicação, e oferece no final de cada sessão os métodos que tiveram sobrecarga de chamados, auxiliando assim o desenvolvedor a desenvolver um código mais eficiente.

3. Resultados obtidos

Os módulos da *Shell Pegasus* foram analisados inicialmente de forma individual, para após a análise dos três se criar uma conclusão sobre a *Shell.*Os resultados serão discutidos nas próximas seções

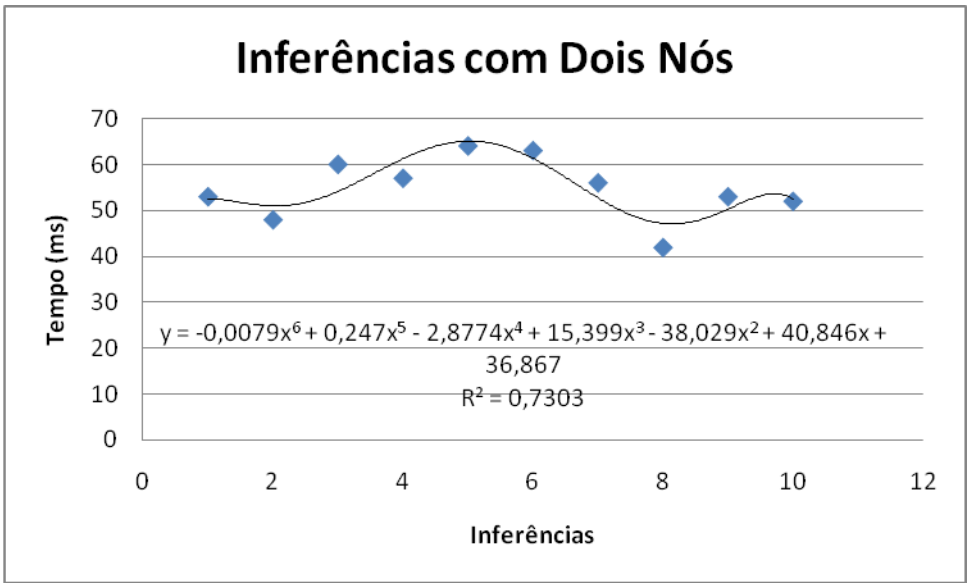
3.1. Análise do módulo Bayesiano da Shell Pegasus

Para a análise de desempenho do módulo de redes Bayesianas da shell Pegasus, foram criadas 3 redes Bayesianas com as seguintes configurações: 2 nós, 4 nós e 6 nós.

Em seguida procedeu-se a realização da análise estatística com os mesmos números de inferências (dez) para cada modelo. Os gráficos abaixo mostram os tempos de cada teste, a equação da função polinomial que melhor se ajusta aos tempos obtidos e o coeficiente de determinação (r^2), que indica o quando o tempo é explicado pelo número de inferências, sendo que quanto mais próximo de 1 (um) for o seu valor, melhor o modelo matemático explica essa relação.

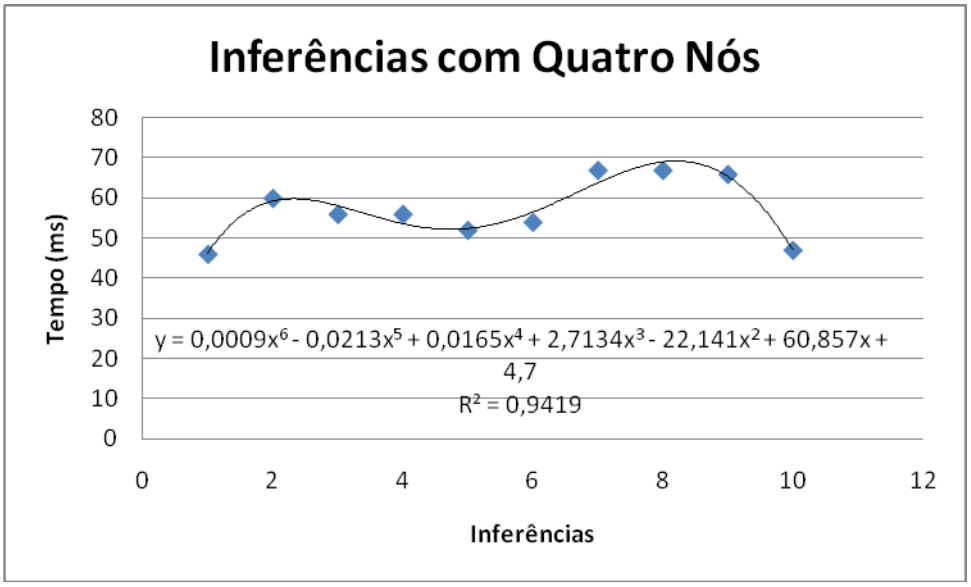
Para a análise de desempenho do módulo de redes Bayesianas da shell Pegasus, foram criadas 3 redes Bayesianas com as seguintes configurações: 2 nós, 4 nós e 6 nós.

Em seguida procedeu-se a realização da análise estatística com os mesmos números de inferências (dez) para cada modelo. Os gráficos abaixo mostram os tempos de cada teste, a equação da função polinomial que melhor se ajusta aos tempos obtidos e o coeficiente de determinação (r^2), que indica o quando o tempo é explicado pelo número de inferências, sendo que quanto mais próximo de 1 (um) for o seu valor, melhor o modelo matemático explica essa relação.



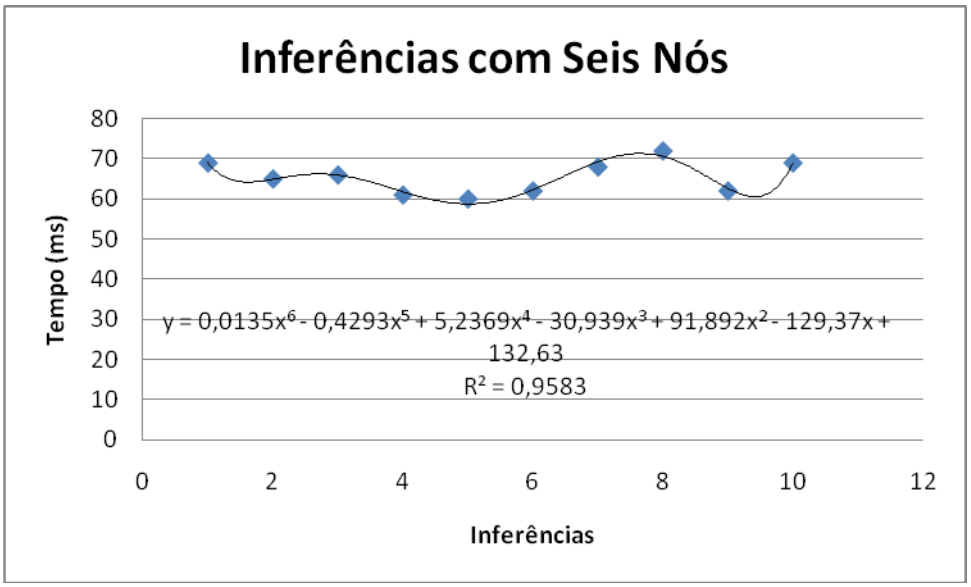
Fonte: Elaborado pelo autor.

Figura 1. Resultado das inferências em uma rede com 2 nós



Fonte: Elaborado pelo autor.

Figura 2. Resultado das inferências em uma rede com 4 nós



Fonte: Elaborado pelo autor.

Figura 3. Resultado das inferências em uma rede com 6 nós

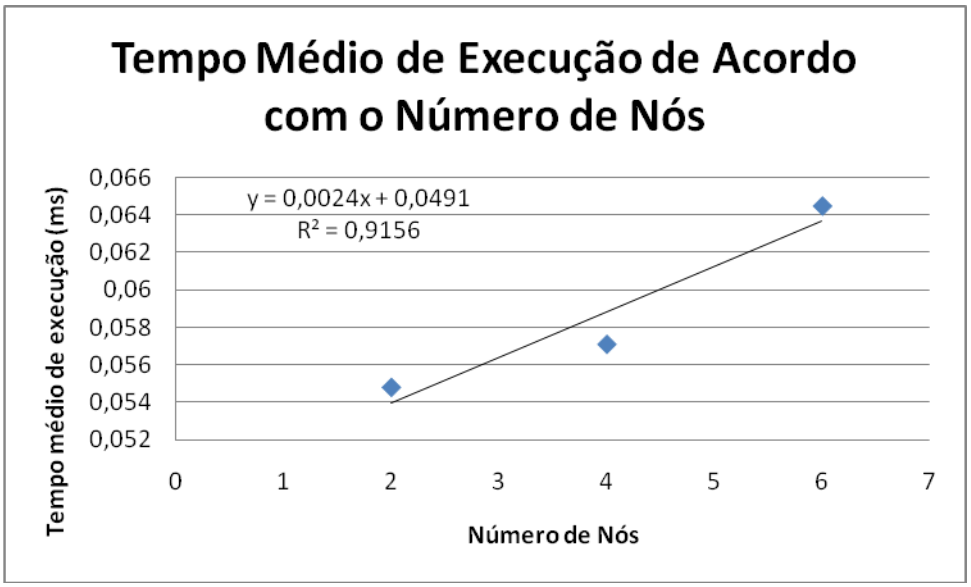
Pode-se perceber que a curva que melhor se ajusta aos dados é a originada de uma função polinomial do sexto grau, o que fica mais evidente quanto mais nós acrescentam-se à análise, chegando a um coeficiente de determinação de $r^2 = 0,9583$. A questão que se segue é de que forma se comporta o tempo médio de cada análise.

Nessa direção, após calcular as médias de tempo de execução com os 3 tipos diferentes de redes, foi calculado a média das médias afim de achar o tempo médio total de execução e se avaliar como é o crescimento dos tempos. A tabela 1 e a figura 4 ilustram os dados:

Tabela 1. Média dos tempos de execução do Módulo Bayesiano da Shell Pegasus

Tipo de rede (nós)	Tempo médio de execução (ms)
2	0.0548
4	0.0571
6	0.0645
Média total	0.0588

Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.

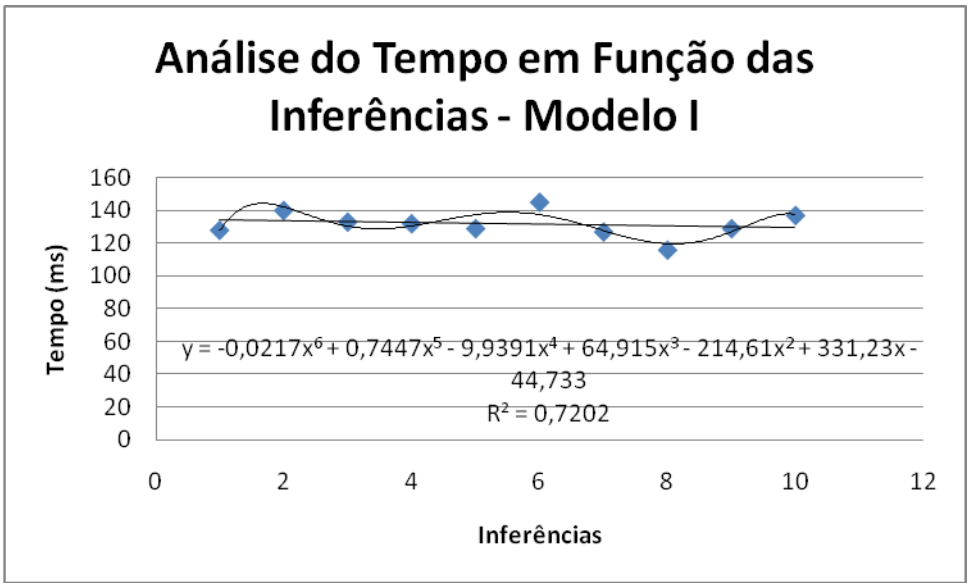
Figura 4. Média dos tempos de execução das inferências

A observação que se faz é de que embora durante os testes onde fixaram-se o número de nós e realizaram-se as inferências, o modelo explicativo é polinomial, porém, quando tomadas as médias das inferências pode-se ter a clara visão de que o comportamento das médias ao acrescentarmos nós no modelo é linear, com um coeficiente de determinação $r^2 = 0,9156$.

3.2. Análise do módulo de Fatores de Certeza da Shell Pegasus

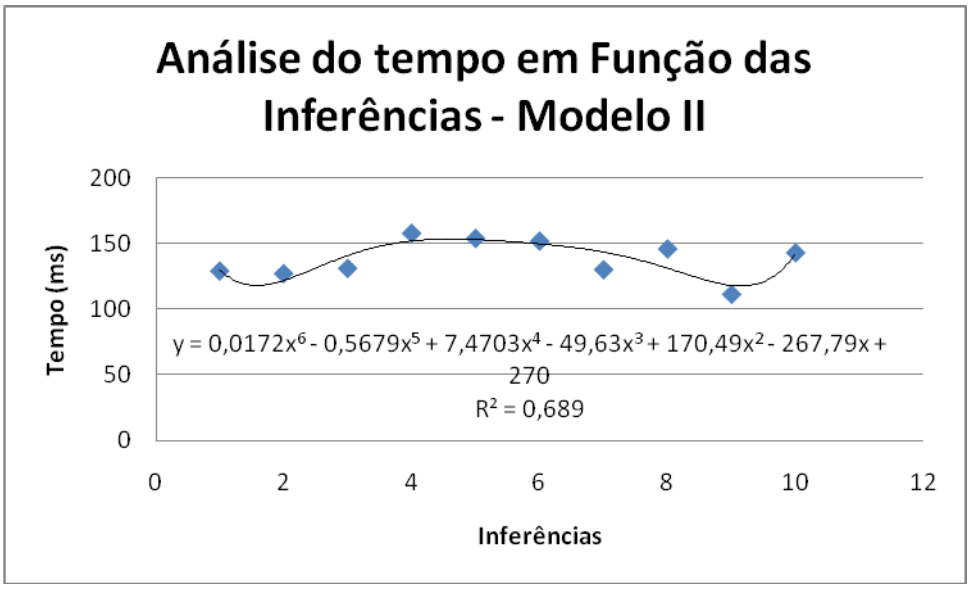
Para a análise do módulo de Fatores de Certeza da Shell Pegasus, foram criados 3 modelos com as seguintes características: 3 variáveis, 5 variáveis e 7 variáveis.

Foram executadas dez inferências em cada modelo, os tempos foram catalogados e inseridos em um gráfico, conforme ilustram as figuras que seguem.



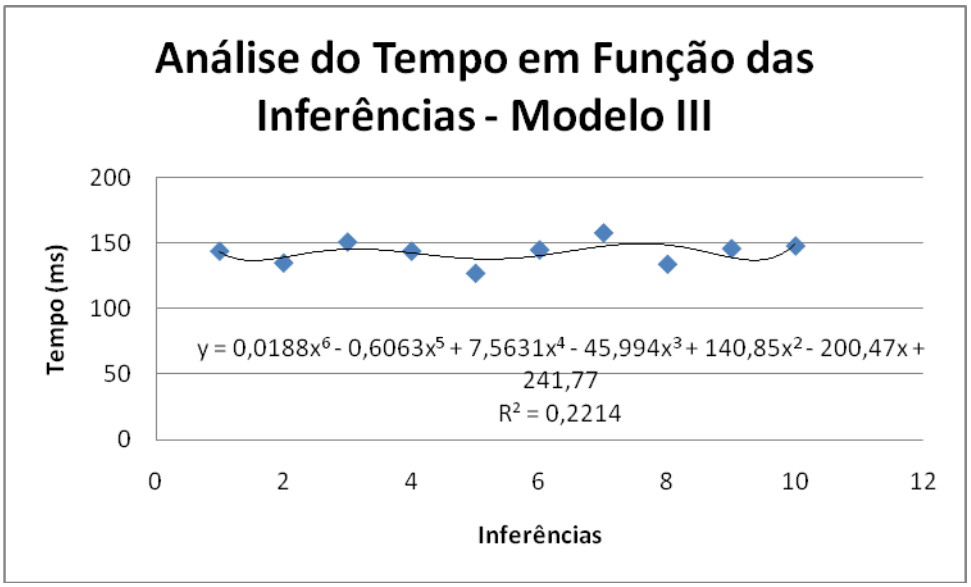
Fonte: Elaborado pelo autor.

Figura 5. Tempos de execução das inferências em um modelo com 3 variáveis



Fonte: Elaborado pelo autor.

Figura 6. Tempos de execução das inferências em um modelo com 5 variáveis



Fonte: Elaborado pelo autor.

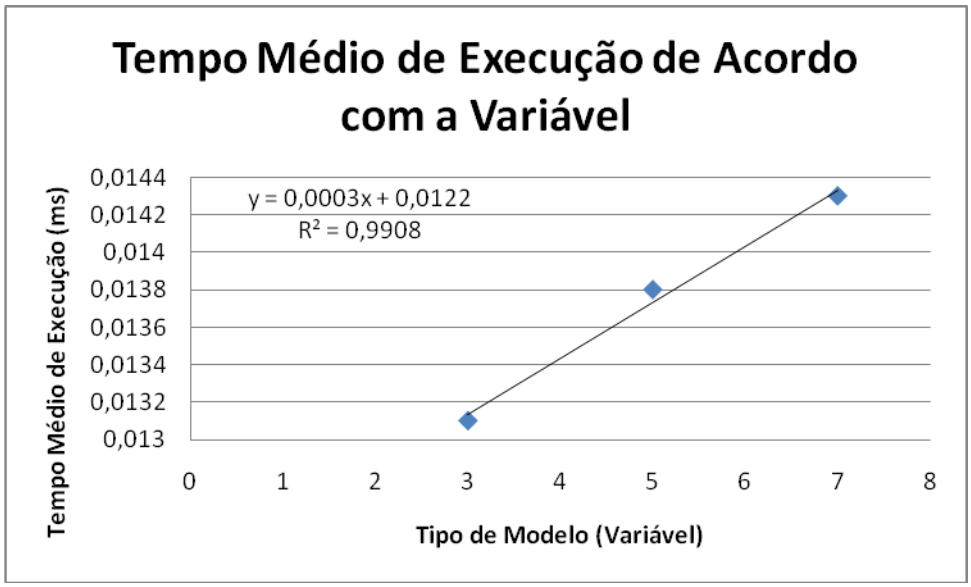
Figura 7. Tempos de execução das inferências em um modelo com 7 variáveis

Após as execuções, foi calculado o tempo médio de cada modelo e o tempo médio total de execução do módulo conforme ilustra a tabela . Esses dados foram inseridos em um gráfico conforme a figura 8 e a tabela 2:

Tabela 2. Média dos tempos de execução do módulo de Fatores de Certeza da *Shell* pegasus

Tipo de modelo (variável)	Tempo médio de execução (ms)
3	0.0131
5	0.0138
7	0.0143
Média total	0.0137

Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.

Figura 8. Médias de tempo de execução das inferências do módulo de fatores de certeza

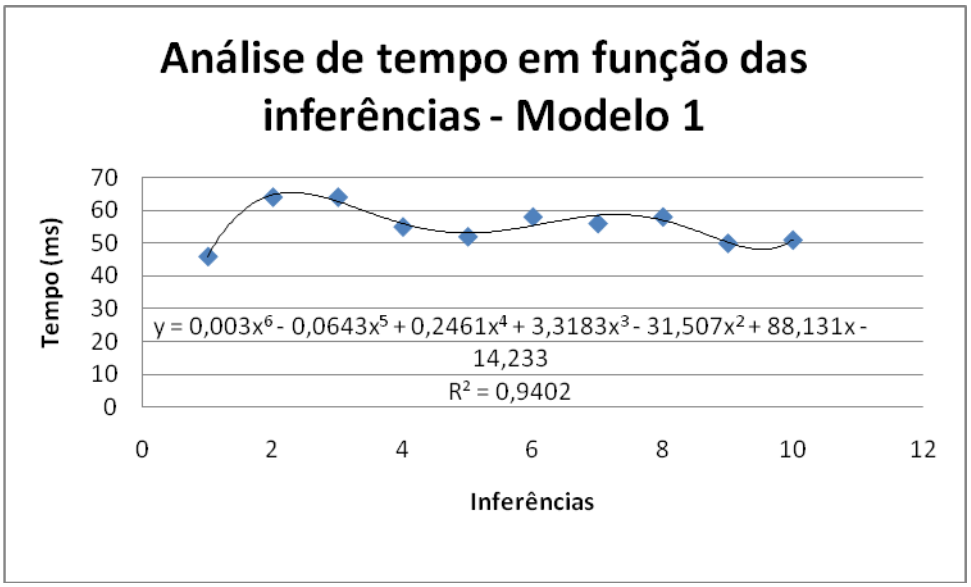
Pode-se perceber que quando analisados cada modelo de forma separada, quanto ao tempo em relação ao número de inferências ficou evidente novamente que o modelo que mais adéqua a situação é o polinomial, porém, a medida em que se inserem novas variáveis ele vai perdendo a sua força de determinação. Já quando são analisadas as médias dos grupos de inferências, percebe-se uma tendência linear, com coeficiente de determinação $r^2 = 0,9908$.

3.3. Análise do módulo de Dempster-Shafer da Shell Pegasus

Para a análise do módulo de Dempster-Shafer da Shell Pegasus, foram criados 3 modelos com as seguintes características:

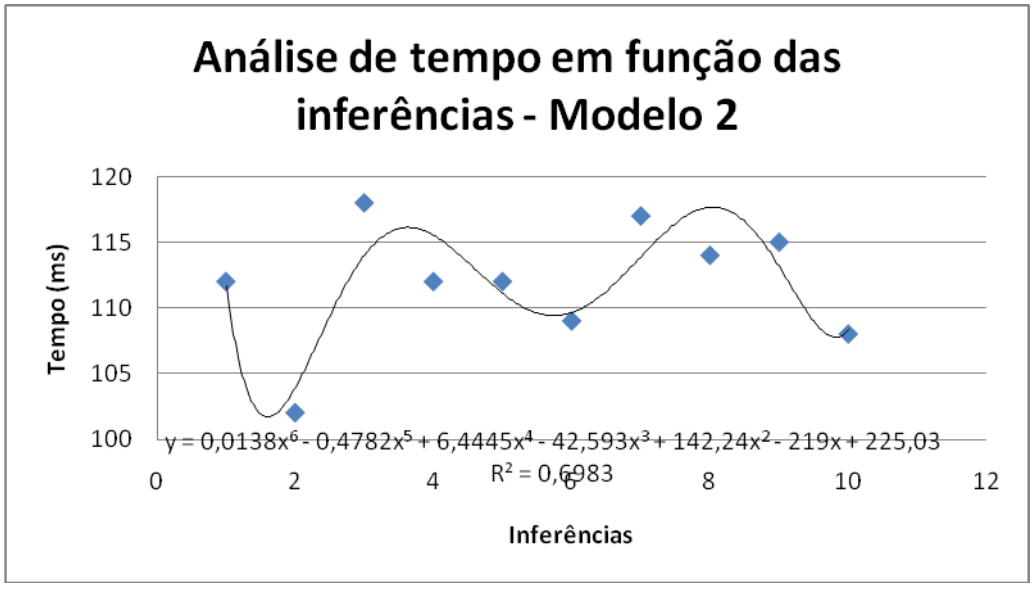
a) Modelo 1: 4 evidências, 2 hipóteses e 2 especialistas
b) Modelo 2: 6 evidências, 4 hipóteses e 2 especialistas
c) Modelo 3: 8 evidências, 6 hipóteses e 2 especialistas

Para cada modelo foram executadas 10 inferências, seus tempos de execução foram catalogados e utilizados para calcular as médias de tempo. As figuras mostram os tempos de execução das inferências:



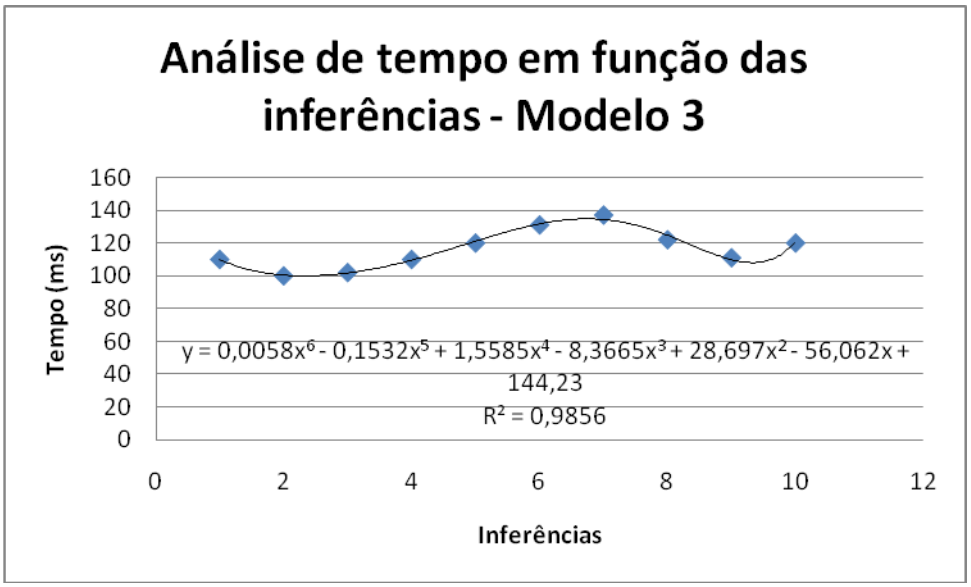
Fonte: Elaborado pelo autor.

Figura 9. Tempos de execução das inferências em um modelo com 6 evidências, 4 hipóteses e 2 especialistas



Fonte: Elaborado pelo autor.

Figura 10. Tempos de execução das inferências em um modelo com 6 evidências, 4 hipóteses e 2 especialistas



Fonte: Elaborado pelo autor.

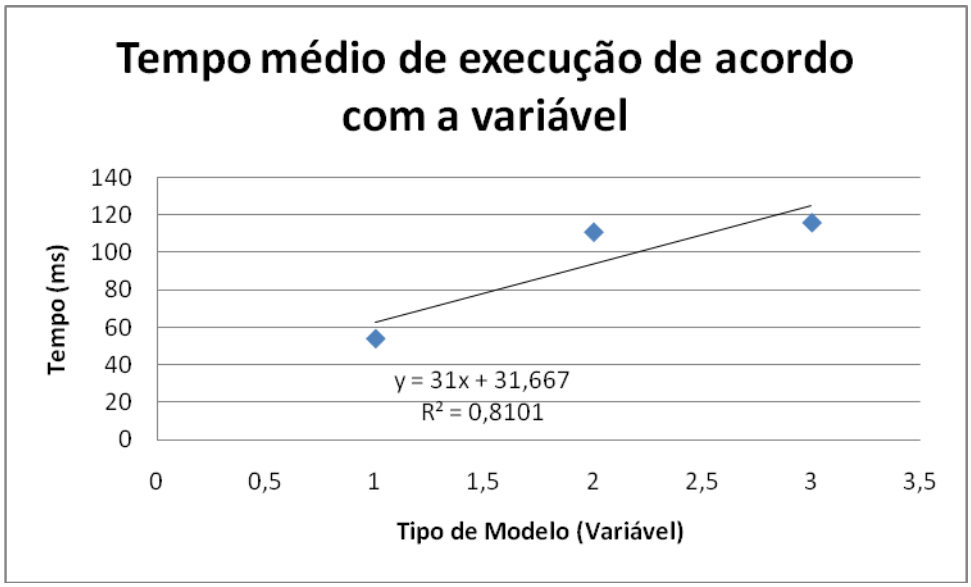
Figura 11. Tempos de execução das inferências em um modelo com 8 evidências, 6 hipóteses e 2 especialistas

Após as execuções, foram calculados os tempos médios de cada modelo e analisados em um gráfico a fim de visualizar sua taxa de crescimento, conforme mostram a tabela 3 e a figura 12:

Tabela 3. Tabela média dos tempos de execução dos modelos no módulo de Dempster-Shafer da Shell Pegasus

Tipo de modelo			Tempo médio de execução (ms)
Evidências	Hipóteses	Especialistas	
4	2	2	0.0054
6	4	2	0.0111
8	6	2	0.0116
Média total			0.0093

Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.

Figura 12. Média dos tempos de execução dos modelos no módulo de Dempter-Shafer da *Shell* Pegasus

Pode-se perceber que ao analisarmos os modelos isoladamente em relação às inferências realizadas, que a curva que melhor se ajustou a distribuição nos três casos foi a curva polinomial, porém com um coeficiente de determinação instável (moderado à forte).

Tomando a média de desempenho de cada modelo, pode-se perceber claramente uma tendência linear, com coeficiente de determinação $r^2 = 0,8101$.

3.4. Análise geral dos módulos

Conforme análise estatística dos tempos de execução dos módulos da Shell, chegou-se a constatação que o algoritmo utilizado nos módulos da *Shell Pegasus* tem uma progressão linear, o que o torna de nível de complexidade O(n).

Conforme analisado na pesquisa, os algoritmos dessa classe são considerados de baixo custo, portanto podemos considerar que o mesmo possui um desempenho aceitável.

4. Conclusão

Com essa pesquisa, pode-se verificar como a IA é útil em nosso meio, o quão complexo são os problemas que ela lida e como ela está disseminada em todas as áreas da ciência.

Pode-se perceber também como as Shells são fatores importantes nas pesquisas na área de Modelagem de Incerteza, haja vista que a carga matemática é altíssima e de implementação complexa.

A Shell Pegasus Uncertainity Modeling sem dúvida alguma é uma ferramenta completa, pois alia três das quatro principais técnicas de Modelagem de

Incerteza, o que faz com que a mesma acabe preenchendo uma lacuna que ainda não foi preenchida no âmbito de Shells.

Conforme a análise de desempenho feita, chegou-se a conclusão de que a Shell possui um desempenho considerado aceitável.

Referências

Costa, Ernesto; Simões, Anabela. Inteligência artificial: fundamentos e aplicações. Lisboa: FCA, 2004.

PRESSMAN, S. ROGER **Engenharia de Software** São Paulo: Makron Books, 1995. 1063 p.

CARTER, Nicholas. Arquitetura de Computadores - Coleção Schaum. Porto Alegre: Bookman, 2003. 240p.

HENNESY, John L.; PATTERSON, David A. Arquitetura de computadores: uma abordagem quantitativa. 3 . ed. Rio de Janeiro: Campus, 2003. 827 p.