Universidade do Estado do Pará
Centro de Ciências Naturais e Tecnologia
Curso de Graduação em Engenharia de Produção

CLEVER RONNI MONTEIRO MOREIRA

DESENVOLVIMENTO DE UM SISTEMA COMPUTACIONAL PARA AUXILIAR À TOMADA DE DECISÃO MULTICRITÉRIO PELO MÉTODO AHP

Belém
2017

Clever Ronni Monteiro Moreira

DESENVOLVIMENTO DE UM SISTEMA COMPUTACIONAL PARA AUXILIAR À TOMADA DE DECISÃO MULTICRITÉRIO PELO MÉTODO AHP

Trabalho de Conclusão de Curso apresentado à Universidade do Estado do Pará como requisito final para obtenção do título de Engenheiro de Produção.

Orientadora: Prof. Dra. Yvelyne Bianca Iunes Santos

BELÉM - PA
2017

UNIVERSIDADE DO ESTADO DO PARÁ
CENTRO DE CIÊNCIAS NATURAIS E TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

"DESENVOLVIMENTO DE UM SISTEMA COMPUTACIONAL PARA AUXILIAR À TOMADA DE DECISÃO MULTICRITÉRIO PELO MÉTODO AHP". Trabalho de Conclusão de Curso apresentado como requisito necessário para obtenção do título de Engenheiro de Produção pelo aluno **Clever Ronni Monteiro Moreira**, em 07 de dezembro de 2017, no Centro de Ciências Naturais e Tecnologia da Universidade do Estado do Pará - CCNT/UEPA, e aprovado pela Banca Examinadora, formada pelos seguintes membros:

Dra. Yvelyne Bianca Iunes Santos – UEPA
Orientadora

Dr. José Alberto Silva de Sá - UEPA
Avaliador 1

Esp. Cláudio Mauro Vieira Serra - UEPA
Avaliador 2

Belém/PA, 07 de dezembro de 2017.

AGRADECIMENTOS

Primeiramente agradeço a Deus por permitir que pudesse desenvolver esta pesquisa, e aos meus familiares que apoiaram nesta jornada que é a graduação. Agradeço principalmente à minha mãe e sua constante preocupação com meu bem-estar e ao meu pai que sempre me deu forças e conselhos ao longo da graduação.

Agradeço também à minha orientadora, a Profa. Dra. Yvelyne Bianca Iunes Santos que teve muita paciência e mesmo quando eu não esperava, acreditou que eu pudesse completar este trabalho.

Agradeço também ao Prof. Dr. José Alberto Silva de Sá, que apesar de não ser meu orientador, deu-me algumas dicas sobre a redação do trabalho.

Gostaria de agradecer também aos meus amigos de faculdade que sempre me animaram e cobraram resultados da minha parte, especialmente Murillo Cavalleiro, Herbert Barbosa, Caio Miranda e Kathleen Ferreira que por vezes tentaram me ajudar com o trabalho, sempre marcando reuniões para escrevermos ou trabalharmos juntos, cada um em seu projeto.

Agradeço também á Thayse Brito, por sempre ter estado disposta a ajudar, dando-me sugestões e me cedendo alguns trabalhos de referências. Além disso, ela também me ajudou com conversas, pois sempre ouvia meus pontos, mesmo sem entender nada. De forma análoga, agradeço à Thayris Brito por se disponibilizar a ler meu trabalho, mesmo que eu nunca tenha enviado algo para que ela corrigisse, e por ser sempre uma boa ouvinte.

Agradeço à Debora Bezerra por ter oferecido ajuda com o trabalho quando eu mais estava desesperado, mesmo que no fim a ajuda não tenha sido necessária. Entretanto devo agradecimentos não só por esse fato, mas também pela forma de como ela me ajudou ao longo do curso inteiro.

Por fim, meus sinceros agradecimentos à UEPA e todo seu corpo docente de Engenharia de Produção por fornecer o conhecimento necessário para a formação de ótimos profissionais.

“Pensar é o trabalho mais difícil que existe, essa provavelmente é a razão pela qual poucos se propõem a isso”.

(Henry Ford)

Resumo

MOREIRA, Clever Ronni Monteiro. **Desenvolvimento de um sistema computacional para auxiliar a tomada de decisão multicritério pelo método AHP.** 69 f. Trabalho de Conclusão de Curso (Bacharel em Engenharia de Produção) — Universidade do Estado do Pará. Belém 2017.

A tomada de decisão é um processo complexo que muitas vezes envolve diversas variáveis a serem estudadas. Por esse motivo existem diversos métodos, para organizar e auxiliar a tomada de decisão. Um dos métodos muito difundido e aplicado em diversos campos do conhecimento é a Análise Hierárquica de Processo (AHP) criada por Saaty. Por ser uma ferramenta efetiva e de simples entendimento, a AHP é amplamente utilizada em trabalhos acadêmicos. Em consequência de efetuar cálculos com matrizes, quanto maior o número de critérios e subcritérios, mais complexo e trabalhoso o uso do método se torna. Assim, para facilitar a aplicação da AHP, utilizam-se *softwares*, os quais são muitas vezes proprietários e podem ser inacessível para a maioria dos estudantes que não tem condições de adquirir uma licença. Dessa forma, este trabalho tem como objetivo desenvolver e validar um *software* que auxilie a tomada de decisão através do método AHP de Saaty, para que posteriormente ele seja disponibilizado com uma licença do tipo *Open Source*, de forma que qualquer pessoa possa testá-lo, modificá-lo e usá-lo gratuitamente. O *software* foi desenvolvido em uma distribuição Mint do Linux, utilizando uma IDE chamada Vim (Vi IMproved), e a implementação do algoritmo foi feita na linguagem Python. Para validação do *software*, foram usados os dados de entrada de dois trabalhos, de autores diferentes, e comparados os resultados destes com aqueles apresentados pelo *software* desenvolvido. O *software* conseguiu resolver ambos de forma satisfatória, mostrando que pode ser utilizado para cálculo de AHP com mais de um nível de critérios.

Palavras-Chave: Análise hierárquica de processo; *Software* livre; Tomada de decisão multicritério; Python.

Abstract

Decision making is something complex that, many times, involves several variables to be studied. For this motive, there are various methods, technical and non-technical, to organize and support decision making. One of those methods, which is very well known and widely applied in many fields of knowledge is named Analytic Hierarchy Process (AHP) and was created by Thomas L. Saaty. Being an effective and simple to understand tool, the AHP is widely applied in academic works. As a consequence of making calculations with matrixes, the more substantial is the number of criteria and subcriteria, the more complex and demanding the method use becomes. So, to facilitate the application of AHP, it is utilized some software, which are often proprietary and might be inaccessible for the majority of the students that cannot afford a license. Thus, this work aims to develop and validate a computer program that assists decision making through the method AHP by Saaty, and posteriorly made it available with an Open Source license, in a way that anyone can test it, modify it and use it for free. The software was developed in Mint, a Linux distribution, using an IDE called Vim (Vi IMproved), and the algorithm implementation was developed in the Python programming language. To validate the software, it was extracted input data from two works, written by different authors. Then, the outputs generated by the software built in this work were compared to the result found by the other authors in their works. The software managed to solve both satisfactorily, showing that is can calculate an AHP hierarchy with more than one level of criteria.

Keywords: Analytic Hierarchy Process; Open Source Software; Multi-criteria decision making; Python.

LISTA DE FIGURAS

Figura 1 – Diagrama Hierárquico Genérico 19
Figura2 - Hello World em C++ 26
Figura 3 - Hello World em Python 27
Figura 4 - Hello World em C# 27
Figura 5 - Comando de instalação do Vim no Linux 29
Figura 6 - Executando o Vim no Linux 30
Figura 7 - Interface do Vim no sistema Linux 30
Figura 8 - Janela 1 de instalação do Vim no Sistema Windows 31
Figura 9 - Janela 2 de instalação do Vim no Sistema Windows 31
Figura 10 - Janela 3 de instalação do Vim no Sistema Windows 32
Figura 11 - Interface do Vim em um sistema Windows 32
Figura 12 - Diagrama de Etapas da pesquisa 38
Figura 13 - Primeira estrutura do programa 39
Figura 14 - Segunda estrutura do programa e representação do cálculo de pesos 40
Figura 15 - Exemplo de escolha de pesos para a montagem do vetor final de pesos 40
Figura 16 - Modelo para a descrição de objetivo e critérios principais 43
Figura 17 – Modelo de descrição de subcritérios 44
Figura 18 - Arquivo de alternativas 45
Figura 19 - Input do arquivo de critérios no Odin 45
Figura 20 - Erro de arquivo não encontrado 46
Figura 21 - Erro ao entrar um arquivo de texto 46
Figura 22 - Matriz recíproca de critérios gerada pelo software Odin para Winston 48
Figura 23 - Matriz recíproca de análise pareada de critérios do exemplo de Winston. 48

Figura 24 - Arquivo com os dados de julgamento dos critérios para teste 1 49

Figura 25 - Matriz normalizada calculada por Winston.....49

Figura 26 - Pesos dos critérios calculados por Winston. 50

Figura 27 - Matriz normalizada, pesos dos critérios e consistência calculados por Odin.....50

Figura 28 - Arquivo com os dados de comparação das alternativas para teste 1 51

Figura 29 - Matriz de comparação para o critério SAL montada por Winston 51

Figura 30 - Matriz de comparação para o critério SAL montada por Odin 52

Figura 31 - Matriz normalizada de pesos para SAL montada por Winston 52

Figura 32 - Matriz normalizada de pesos para SAL montada por Odin. 52

Figura 33 - Pesos parciais das alternativas para SAL calculados por Winston 53

Figura 34 - Pesos parciais das alternativas para SAL calculados em Odin 53

Figura 35 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério QL calculados por Winston 54

Figura 36 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério QL calculados em Odin. 54

Figura 37 - Matrizes de julgamento e de pesos e prioridade relativa para o critério IW calculados por Winston 55

Figura 38 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério IW calculados em Odin 55

Figura 39 - Matrizes de julgamento e de pesos, e prioridade relativa para o critério NF calculados por Winston.....56

Figura 40 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério NF calculados em Odin 56

Figura 41 - Prioridade (resultado) final das alternativas por Winston 57

Figura 42 - Prioridade (resultado) final das alternativas por Odin 57

Figura 43 - Arquivo alterado de alternativas de trabalho.....58

Figura 44 - Nova matriz para o critério NF 59

Figura 45 - Nova matriz normalizada de pesos, prioridade relativa e julgamentos para NF 59

Figura 46 - Resultados finais do teste 1 recalculados 59

Figura 47 - Hierarquia de critérios para a Seleção de projetos da ACME 60

Figura 48 - Matriz principal de critérios para o problema de Vargas montadas por Odin.....62

Figura 49 - Matriz normalizada de pesos dos critérios para o problema de Vargas montadas por Odin.....62

Figura 50 - Pesos dos critérios principais para o problema de Vargas calculados por Odin.....63

Figura 51 - Pesos dos subcritérios para o problema de Vargas calculados por Odin63

Figura 52 - Pesos dos critérios e subcritérios calculados por Vargas 64

Figura 53 - Inconsistência na matriz de opções para o critério CIIM.....65

Figura 54 - Inconsistência na matriz de opções para o critério LRO 66

Figura 55 - Inconsistência na matriz de opções para o critério URGE 66

Figura 56 - Inconsistência na matriz de opções para o critério ITK.....67

Figura 57 - Resultado final para o problema de Vargas 67

Figura 58 - Resultado final calculado pelo Expert Choice. 68

LISTA DE QUADROS

Quadro 1 - Escala Saaty 20

Quadro 2 - Índices randômico de consistência..... 22

Quadro 3 - Comandos básicos do Vim..... 33

Quadro 4 - Projeto a serem escolhidos e siglas usadas no Programa Odin. 61

Quadro 5 - Critérios para escolha de projetos e siglas aplicadas no *software* Odin.. 61

Quadro 6 - Matriz de critérios principais (Nível 1) por Vargas 62

Quadro 7 - Matriz normalizada de pesos para os critérios principais (Nível 1) por Vargas 62

Quadro 8 - Pesos dos critérios e subcritérios para o problema de Vargas calculados por Odin 64

Quadro 9 - Resultado final ordenado para o problema de Vargas 68

Quadro 10 - Regressão Simples *Odin* X *Expert Choice* 69

Quadro 11 - Análise de variância parte 1 69

Quadro 12 - Análise de variância parte 2 70

SUMÁRIO

CAPÍTULO I – INTRODUÇÃO 14

1.1 Tema e problema 14

1.2 Justificativa 15

1.3 Objetivos 16

1.3.1 Objetivo Geral 16

1.3.2 Objetivos específicos 16

1.4 Estrutura do trabalho 17

CAPITULO II – REFERENCIAL TEÓRICO 18

2.1 Análise hierárquica de processo (AHP) 18

2.2 *Software open source* 22

2.3 Linguagens de programação 24

2.3.1 O sistema de tipos 24

2.3.2 Linguagens de baixo e alto nível 25

2.3.3 Linguagens compiladas e interpretadas. 25

2.4 A Linguagem Python 25

2.5 IMPROVED VI (VIM). 29

2.6 Análise de Regressão 34

CAPITULO III – METODOLOGIA DE PESQUISA 36

3.1 Estratégia e classificação de pesquisa 36

3.2 Delineamento da pesquisa 36

3.4 Funcionamento do algoritmo desenvolvido 38

3.5 Atribuição do nome do *software* 41

CAPITULO IV – FUCIONAMENTO E TRATAMENTO DE EXCEÇÕES 42

4.1 Montagem dos arquivos para a entrada de dados 42

4.1.1 Arquivo de critérios 42

4.1.2 Arquivo de Alternativas.....44

4.2 Funcionamento do programa.....45

CAPÍTULO V – VALIDAÇÃO 48

5.1 Primeiro teste 48

5.1 Segundo teste.....60

5.3 Explicação sobre os arredondamentos.....70

CAPÍTULO VI – CONCLUSÃO E SUGESTÕES.....70

6.1 Conclusão.....70

6.2 Trabalhos futuros.....72

REFERÊNCIAS.....74

APÊNDICE 1.....77

CAPÍTULO I – INTRODUÇÃO

1.1 Tema e problema

A tomada de decisão é um processo complexo que por vezes exigem uma série de critérios a serem analisados cuidadosamente pelos tomadores de decisão. Segundo San Cristobal (2012), a tomada de decisão é o estudo feito para identificar e escolher as melhores alternativas possíveis (solução ótima) provenientes de diversos fatores e visando atender as expectativas dos tomadores de decisão. Toda decisão é estudada em um ambiente, no qual há uma séria de informações, alternativas e preferências disponíveis enquanto a decisão está em análise. Por esse motivo, a tomada de decisão pode se tornar um tanto quanto conflitante e complexa devido à multiplicidade de critérios usados para escolher as alternativas, e também à necessidade de envolvimento de vários grupos na tomada de decisão. Assim, para facilitar a esse tipo de análise, um grupo de ferramentas nomeadas de Análise de Multicritérios ganhou espaço em consequência da necessidade de formalizar métodos auxiliadores de tomadas de decisão quando as situações evolvem multicritérios.

Uma das ferramentas de Análise Multicritério é a Análise Hierárquica de Processo (AHP) criada por Thomas L. Saaty que por sua simplicidade, facilidade de uso e poder, tem sido usada em diversos tipos de aplicações pelo mundo (BHUSHAN; RAI, 2004, p. 15).

De acordo com o tamanho da hierarquia usada, o processo AHP fica cada vez mais complexo de calcular, por esse motivo é necessário que o processo de decisão seja assistido por um *software* que calcule os pesos de todos os critérios para enfim decidir entre as alternativas disponíveis.

Apesar de haverem diversos *softwares* no mercado que possibilitam o cálculo da AHP, como o *Expert Choice* e o *Microsoft Excel*, a grande maioria deles são soluções pagas, e mesmo quando os desenvolvedores disponibilizam uma versão grátis da sua aplicação, ela geralmente contém limitações como entrada de critérios limitada, ou licença de até trinta dias para teste.

Assim, o trabalho tem o objetivo de desenvolver um código gratuito e *open source,* para que pesquisadores possam ter uma alternativa aos *softwares* pagos,

sem que o mesmo tenha nenhum tipo de prejuízo na sua pesquisa e nem custos desnecessários.

Portanto o presente trabalho tem como tema: o desenvolvimento de um *software* para o cálculo do método AHP.

1.2 Justificativa

Para a efetuação de uma pesquisa científica faz-se uso de diversos tipos de ferramentas para medição, controle e trabalho de dados coletados (Silva; Menezes, 2005). Dentre tais ferramentas, os programas computacionais (ou *softwares*) que podem ser desenvolvidos tanto por demanda ou para um mercado específico (SOMMERVILLE, 2015, p. 6), auxiliam nos estudos acadêmicos em diversas áreas do conhecimento.

No entanto, os *softwares* em sua grande maioria não são distribuídos gratuitamente pelos seus desenvolvedores, principalmente aqueles desenvolvidos por organizações, e algumas soluções podem até mesmo ter preço elevado. Assim, como uma maneira de economizar, as universidade têm a opção de utilizar *software* livres, podendo esses serem desenvolvidos por alunos da própria instituição com o proposito de disponibilizá-los gratuitamente e dando a oportunidade para que os usuários os modifiquem de forma a melhorar o *software* com o passar do tempo.

A ferramenta escolhida para o estudo foi a Análise Hierárquica de Processo (AHP) a qual é um método de tomada de decisão multicritério que calcula pesos para os critérios de decisão a partir da opinião de um especialista, ou um grupo de pessoas, de modo a atender as necessidades dos envolvidos no processo. Ela basicamente ajuda a estruturar a complexidade, medida e a síntese de *rankings* (BHUSHAN; RAI, 2004, p. 15).

Esse método tem aplicação nas mais diversas áreas do conhecimento, segundo Saaty (2008), a AHP já foi utilizada em diversos tópicos na administração pública, na escolha do melhor local para a realocação de uma cidade devastada por um terremoto e até mesmo para a escolha de projetos a serem financiados pela empresa Xerox.

Atualmente, o *software* referência no cálculo da AHP é o *Expert Choice*, criado por Thomas L. Saaty e Ernest Forman e mantido pela *Expert Choice Inc*. Entretanto, outros *softwares* já foram produzidos para tomar decisões com a mesma

metodologia. Contudo, a maioria desses *softwares* são proprietários e geram custo para os pesquisadores que o utilizam para desenvolver seus estudos, acarretando em uma possível desistência da implementação do método e uma perda na contribuição científica que o cientista poderia proporcionar. Baseado nessa motivação, o trabalho terá foco no desenvolvimento de uma aplicação que auxilia a tomada de decisão através do método AHP.

A linguagem utilizada para a construção do programa foi Python devido seu poder e abrangência de uso. Além disso, o Python é uma linguagem que tem portabilidade, ou seja, é possível executar códigos sem a preocupação com a plataforma, tendo em vista que seu interpretador tem versão para os três principais sistemas operacionais *Windows*, *Mac OS*, e distribuições *Linux*, há também um interpretador para *Android* (o qual também é uma distribuição Linux, porém pertencente a uma empresa).

1.3 Objetivos

1.3.1 Objetivo Geral

Desenvolver uma ferramenta computacional voltada para a tomada de decisão de situações que evolvem multicritérios, com base no método de Análise Hierárquica de Processo (AHP).

1.3.2 Objetivos específicos.

- Planejar um modelo de organização dos arquivos de opções (alternativas) e critérios para serem interpretados pelo programa com o intuito de construir as matrizes automaticamente.
- Desenvolver uma biblioteca básica para o trabalho com matrizes.
- Desenvolver código para leitura de um arquivo de texto e cálculo dos pesos de cada matriz de critérios, subcritérios e alternativas.
- Validar o *software* desenvolvido;
- Documentar todas as características e funcionalidades do *software*.

1.4 Estrutura do trabalho

Este trabalho está estruturado em seis capítulos. O primeiro discute o tema e problema a ser abordado pelo trabalho, assim como a justificativa para a efetuação do próprio, seguido dos objetivos gerais e específicos.

O segundo capítulo descreve toda a base necessária para o entendimento do trabalho através da descrição do método AHP de Saaty. Em seguida, o capítulo busca explicar o que é o *software* livre e quais as regras usadas para criar licenças para estes tipos de aplicações. O capítulo finaliza com uma breve explicação sobre a linguagem Python e sobre o ambiente de desenvolvimento onde o programa foi trabalhado.

O terceiro capítulo descreve primeiramente a estratégia e classificação da pesquisa, após isso ela delimita a pesquisa falando sobre as ferramentas utilizadas na construção do *software* assim como o algoritmo base utilizado.

No quarto capítulo é explicado todas as características do *software* e o funcionamento do mesmo, desde a criação dos arquivos para a entrada de dados até o modo de abrir os arquivos através do programa.

No capítulo cinco é feita a validação do *software* através de dois exemplos, o primeiro retirado de um livro e o segundo de um artigo científico.

Por fim, o capítulo seis conclui o trabalho expressando algumas reflexões sobre os resultados obtidos. O mesmo capítulo explicita limitações do programa e em seguida sugere melhoras. Por fim, o capítulo é finalizado com sugestão de trabalho futuro.

CAPITULO II – REFERENCIAL TEÓRICO

2.1 Análise hierárquica de processo (AHP)

De acordo com Bushan e Rai (2004, p. 15), a Análise Hierárquica de Processo foi desenvolvida por Thomas L. Saaty que na época dirigia projetos de pesquisa na Agência de Controle de Armas e Desarmamento dos Estados Unidos da América (*US Arms Control and Disarmament Agency*) aonde havia uma grande carência de uma metodologia comum de simples entendimento e implementação, e capaz de auxiliar na tomada de decisões complexas.

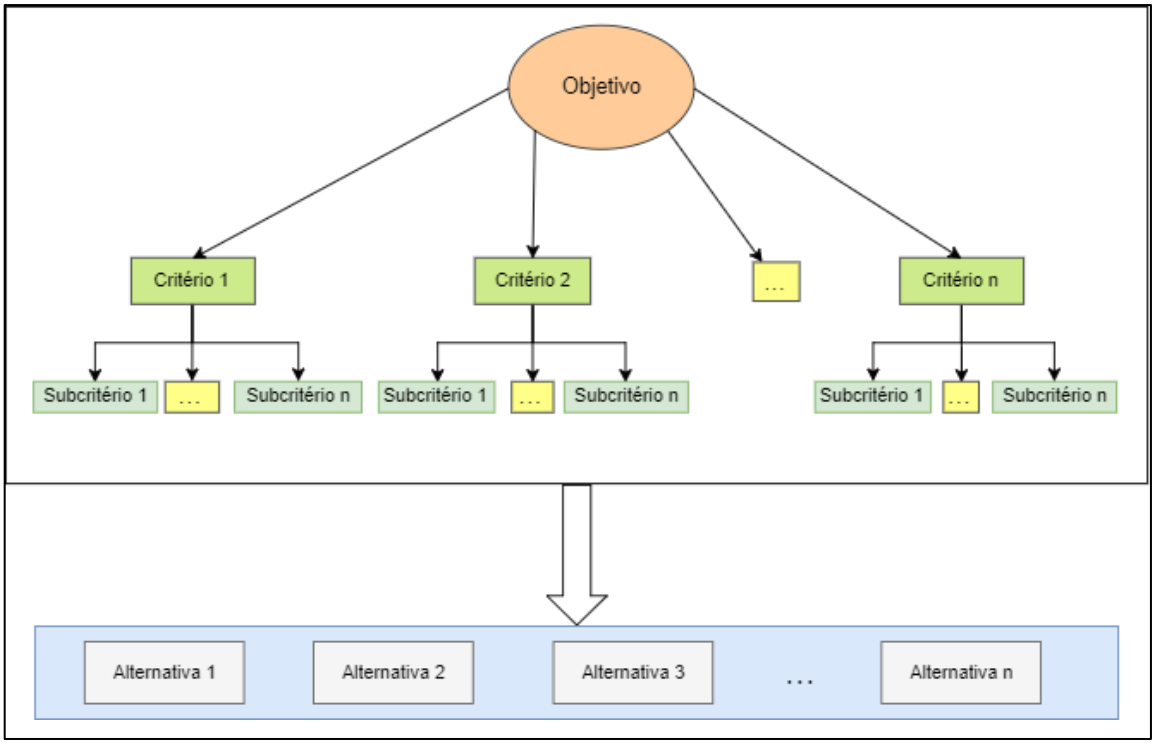
Segundo Saaty (1987), o método AHP é uma teoria que deriva escalas de comparação entre pares discretos e contínuos para fazer mensurações. Ela considera uma série de alternativas a serem escolhidas considerando um objetivo principal ligado a vários critérios e subcritérios (MU; PEREYRA-ROJAS, 2017).

A Análise Hierárquica de Processo é composta basicamente de três etapas:

1ª Etapa: Decomposição do problema

A decomposição do sistema se dá através de critérios, atributos e alternativas, todos organizados em uma estrutura hierárquica descendente (VACLAVIK, 2011). Esta é uma das mais importantes etapas do processo (SAATY, 2008). Essa estrutura hierárquica pode conter diversos níveis com subcritérios e agrupamentos de alternativas de acordo com Tramarico (2012). A figura 1 ilustra um processo hierárquico genérico.

Figura 1 – Diagrama Hierárquico Genérico



Fonte: Autor (2017)

2ª Etapa: Atribuição de pesos para os critérios

A importância de cada critério é comparada em pares através de matrizes de comparação Aij, sendo i igual a j, tal que cada elemento na matriz é um valor de comparação entre os critérios de acordo com uma escala. Assim, é possível perceber a importância relativa de cada critério em comparação ao outro para que enfim seja calculado o peso de importância de cada critério (WINSTON, 2004).

A escala absoluta de Saaty (1987) é a mais comumente usada para o processo. Essa escala contém valores de 1 (igual importância) a 9 (extrema importância) que indicam a intensidade de importância de um item em comparação ao outro. Cada valor usado na escala está descrito no quadro 1.

Quadro 1 - Escala Saaty

Valores	**Interpretação**
1	Os critérios i e j têm a mesma importância
3	O critério i é ligeiramente mais importante que j
5	O critério i é fortemente mais importante que j
7	O critério i é muito fortemente ou comprovadamente mais importante que j
9	O critério i é absolutamente mais importante que j
2, 4, 6, 8	Valores intermediários que estão estre uma definição e outra

Fonte: Winston (2012)

A montagem da matriz recíproca será baseada no autor Saaty (2008), pois em seu artigo *The Seven Pillars of the Analitic Hierarchy Process*, ele descreve e comprova a técnica detalhadamente.

A montagem da matriz recíproca de comparação pareada de julgamentos A = a_{ij} obedece a seguinte regra:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1j} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ij} \end{bmatrix} (1)$$

$$a_{ij} = \frac{1}{a_{ji}} (2)$$

$$a_{ij} > 0 (3)$$

Sendo (2) a representação da propriedade recíproca.

Considerando as propriedades supracitadas, a solução da matriz de comparação é um autovetor direito principal. Não é intenção do trabalho fazer a descrição completa sobre a matemática por trás de autovetores, pois a mesma é dispensável para a compreensão do funcionamento do método.

A AHP se baseia em escalas racionais, que são uma série de números invariantes sobre uma transformação similar, ou seja, a multiplicação dos mesmos

por uma constante positiva. A constante é cancelada quando a razão de dois números é formada. Para exemplificar, considere duas distâncias, a primeira do ponto A ao ponto B sendo x em metros e y quilômetros e a do ponto C ao ponto D sendo w em metros e z em quilômetros. Se calcularmos a razão entre x e w, ela irá ser uma valor v, o mesmo valor da razão entre y e z. Ou seja, v é a constante de proporcionalidade entre as duas razões, em outras palavras a constante não irá mudar independente da escala usada.

Com base no mesmo raciocínio, é possível calcular a escala racional de uma Matriz recíproca de comparação pareada de julgamentos através da relação:

$$\sum_{j=1}^{n} a_{ij} w_j = \lambda_{max} w_s \quad (4)$$

$$\sum_{i=1}^{n} w_i = 1 \quad (5)$$

3ª Etapa: Cálculo de consistência

Para cada matriz de comparação construída pode haver inconsistências, por exemplo, suponha-se que ao comparar o critério (1) com um critério (4) entenda-se que o critério 3 é mais importante que o critério 1, da mesma forma quando comparados os critérios 1 e 2 observa-se que 1 é mais importante, entretanto o tomador de decisão definiu o critério 3 como menos importante que o critério 2, o que pela lógica seria impossível, pois 3 também deveria ser mais importante que 2. Essas incoerências acontecem naturalmente e dependendo do nível de inconsistência (caso seja um alto nível) a análise deve ser refeita (WINSTON, 2004).

Para calcular a consistência, primeiro multiplica-se a matriz recíproca A pelo autovetor Wt^T para achar o vetor de soma dos pesos Ws e em seguida determina-se o valor do máximo autovalor da matriz (λ_{max}). Assim como exposto na equação descrita por Winston (2004).

$$\lambda_{max} = \sum_{i=1}^{n} \frac{Wt^T{}_i}{Ws_i} \quad (6)$$

Em seguida, computa-se o Índice de Consistência (CI).

$$CI = \frac{\lambda_{max} - n}{n - 1} \quad (7)$$

Finalmente, o valor da Razão de Consistência é determinado pela razão do índice de consistência pelo índice randômico (RI) que está descrito no quadro 2 sendo n a dimensão da matriz recíproca.

Quadro 2 - Índices randômico de consistência

N	1	2	3	4	5	6	7	8	9	10
RI	0	0	0,52	0,89	1,11	1,25	1,35	1,40	1,45	1,59

Fonte: Saaty (2008)

2.2 *Software open source*

Um *software open source* é um *software* distribuído gratuitamente que tem código aberto para estudo, aperfeiçoamento e modificações do mesmo por parte de terceiros. Porém o conceito de *Open Source* não é apenas isso, para ser considerado *Open Source* o *software* deve atender os seguintes critérios de acordo com a organização OSI (*Open Source Initiative*):

a) Redistribuição Livre:

A licença não irá restringir nenhum grupo de vender ou fornecer o *software* como um componente de uma distribuição de *software* agregado contendo programas de várias fontes diferentes. A licença não deverá pedir royalties ou outros tipos de taxa para cada venda.

b) Código fonte:

O programa deve necessariamente incluir o código fonte, e permitir a distribuição do código fonte e do programa compilado. Mesmo que alguma forma de um produto não for distribuída com o código-fonte, deverá haver um meio muito bem definido de obter o código-fonte para que a os usuários não tenham custos adicionais além do tempo gasto com a reprodução do algoritmo (desenvolver a própria versão do código), ou de preferência baixando o código na Internet sem nenhum tipo de cobrança. O código-fonte deve ser o meio preferível no qual um programador fará modificações. Formas intermediárias como a saída de um pré-processador ou tradutor não são permitidas.

c) Trabalhos Derivados

A licença deve permitir modificações e trabalhos derivados do código-fonte, e também assentir a distribuição dos mesmos sobre os mesmos termos da licença do *software* original.

d) Integridade do autor do código fonte

A licença deve restringir o código-fonte de ser distribuído na sua forma modificada apenas se a licença permitir a distribuição de “pacotes de arquivos” com o código-fonte com o propósito de modificar o programa no tempo de compilação. A licença precisa consentir explicitamente a distribuição do *software* compilado a partir do código-fonte modificado. A licença pode requerer trabalhos derivados para ter um nome ou número de versão diferentes do *software* original.

e) Sem discriminação contra pessoas ou grupos

A licença não pode discriminar nenhuma pessoa ou grupo de pessoas.

f) Sem discriminação contra campos de trabalho ou objetivos

A licença não pode restringir ninguém de usar o programa em um campo específico de trabalho. Por exemplo, ele não poderá proibir o programa de ser usado em um negócio ou em uma pesquisa genética.

g) Licença de Distribuição.

Os direitos anexados ao programa devem ser aplicados a todos a quem o programa e redistribuído sem a necessidade de conceber uma licença adicional por esses grupos.

h) A licença não é específica à um produto

Os direitos atribuídos ao programa não podem depender do programa sendo uma parte de um *software* de distribuição particular. Se o programa é extraído de uma distribuição e usado ou distribuído dentro os termos da licença do programa, todos os grupos para quem o programa é redistribuído devem ter os mesmo direitos àqueles que foram concedidos em conjunto com a distribuição original do *software*.

i) A licença não deve restringir o uso de outros *softwares*

A licença não deve colocar restrições em outros *softwares* que são distribuídos junto com os *softwares* licenciados. Por exemplo, a licença não deverá

insistir que todos os outros programas distribuídos em um mesmo meio devam ser *softwares open-source*.

j) A licença deve ser neutra quanto à tecnologia utilizada.

Na provisão da licença precisa-se serem previstas quaisquer tecnologia individual ou estilo de interface.

2.3 Linguagens de programação

Atualmente, uma diversidade de linguagens de programação são desenvolvidas para os mais variados propósitos. Essas linguagens existem devido a necessidade de padronizar o desenvolvimento para que os *softwares* possam ser executados por qualquer tipo de CPU, tendo em vista que a linguagem binária, também chamada de linguagem de máquina não é padronizada e varia de acordo com o tipo de máquina (FRIEDMAN; KOFFMAN, 2011). Os tópicos a seguir descrevem mais detalhadamente sobre as linguagens de programação no geral, desde de a primitiva linguagem *Assembly* até as linguagem de mais auto nível como Ruby e Python.

2.3.1 O sistema de tipos

Mesmo antes da invenção dos computadores, matemáticos já produziam as suas próprias linguagens de programação que originalmente foram desenvolvidas pelo conceito de sistemas de provas, os quais são um conjuntos de regras que serviam para derivar verdades lógicas mecanicamente (GRANT et al., 2011).

Com o estudo dos sistemas de provas, deu-se origem ao conceito de sistemas de tipo, que para Pierce (2002) é método de tratamento sintático que prova a não existência de certo comportamento em um programa através da classificação de frases de acordo com o tipo de valores que eles computam.

Os sistemas de tipos são hoje a base para as linguagens de programação computacionais devido ao poder de detecção de erros, eficiência e abstração. As linguagens baseadas em sistemas de tipo são aquelas tratadas por tradutores e interpretadores (PIERCE, 2002).

2.3.2 Linguagens de baixo e alto nível

Linguagens de baixo nível são aquelas que mais se aproximam da linguagem nativa da máquina. Com essas linguagens, é possível dar instruções para a máquina diretamente (OLIVEIRA, 2017).

Um exemplo de linguagem de baixo nível é linguagem nativa de máquina ou binária, que apesar de ter um alto desempenho de execução, varia de acordo com a CPU da máquina, portanto não tem uma sintaxe padrão. Outra desvantagem de usar a linguagem binária é o tempo de desenvolvimento de aplicações longas, tendo em vista que ela é mais complexa, podendo levar tempo para escrever programas funcionais ou detectar erros existentes (FRIEDMAN; KOFFMAN, 2011).

Por outro lado, as linguagens de alto nível são aquelas que misturam expressões algébricas com a linguagem de escrita humana e também podem fazer a CPU executar várias instruções com apenas uma linha de código, ao contrário das linguagens de baixo nível onde cada linha é uma instrução (FRIEDMAN; KOFFMAN, 2011). Entre as linguagens de alto nível estão as linguagens compiladas, como C, C++ e Lisp, e entre as interpretadas estão Java, Python e Ruby.

2.3.3 Linguagens compiladas e interpretadas.

As linguagens compiladas são aquelas que para serem executadas, primeiro são traduzidas para a linguagem de máquina criando assim um arquivo executável pelo sistema operacional (FRIEDMAN; KOFFMAN, 2011). Elas têm um desempenho melhor quando comparadas às linguagens interpretadas, pois quando executadas (em versão binária) lidam diretamente com o processador e a memória (BASTOS, 2010).

Já as linguagens interpretadas são traduzidas para outra linguagem intermediária que é lida por uma máquina virtual que traduzirá o código da linguagem intermediária para a linguagem binária para enfim executar o programa. Contudo, ao contrário da linguagem compilada, que transforma o arquivo de texto em um executável, as linguagens interpretadas são lidas, traduzidas e executadas linha por linha sempre que o programa é rodado (BASTOS, 2010).

2.4 A Linguagem Python

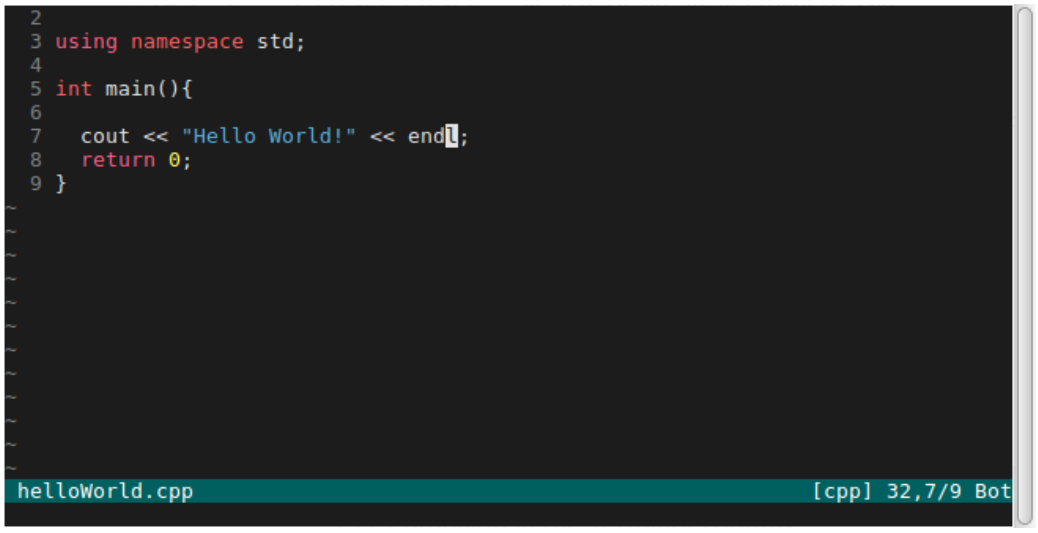
Python é uma linguagem de programação de código aberto (*open source*) para objetivos gerais desenvolvida em 1989 (VENNERS, 2003) por Guido Van

Rossum enquanto trabalhava no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI) na Holanda. O principal foco da linguagem é a produtividade e legibilidade.

Segundo Lutz (2013), a linguagem tem como principal característica a otimização para:

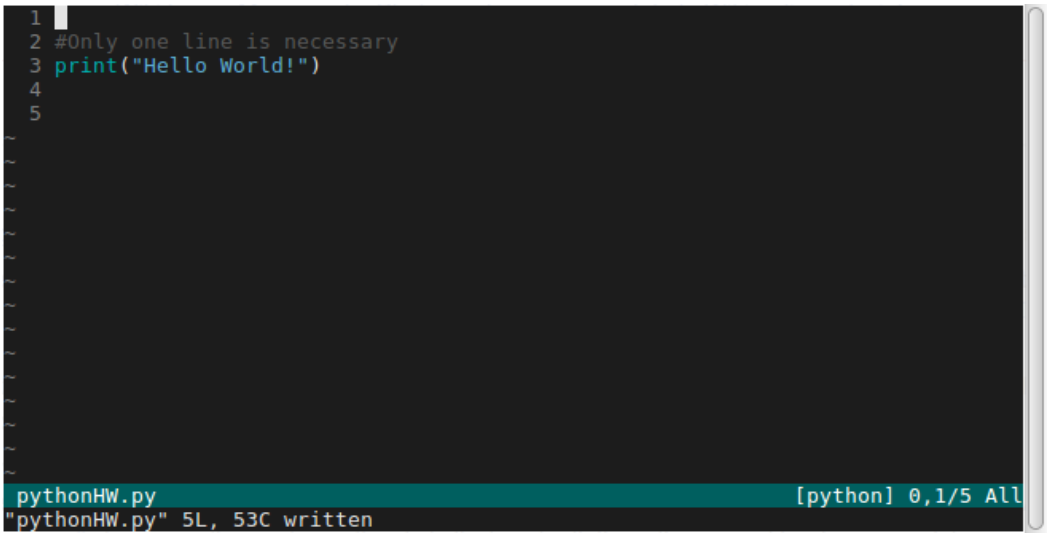
a) Qualidade de *Software*: além de rapidamente interpretada, a estrutura da linguagem permite a manutenção de forma mais simples a um programa, pois os erros podem ser encontrados mais facilmente, se comparado a outras linguagens, pela simplicidade de leitura dos códigos em Python.
b) Produtividade de desenvolvimento: Por ser estruturada de uma forma que os códigos sejam mais enxutos, a linguagem Python permite que um desenvolvedor gaste menos tempo programando uma aplicação. As figuras 2, 3 e 4 mostram códigos em três linguagens diferentes que executam a mesma tarefa:

Figura2 - *Hello World* em C++



Fonte: O autor (2017)

Figura 3 - *Hello World* em Python



Fonte: O autor (2017)

Figura 4 - *Hello World* em C#

Fonte: O autor (2017)

Os três programas mostrados apenas escrevem “Hello World” na tela do computador. Ao comparando os três códigos, é possível observar que em Python se utiliza um menor número de linhas que as outras linguagens apresentadas, o que aumenta bastante a produtividades quando construindo aplicações de porte maior.

c) Portabilidade: por ser uma linguagem interpretada, scripts feitos em Python podem ser executados em diferentes sistemas operacionais sem

que haja a necessidade de fazer portabilidade ou executar outro projeto.

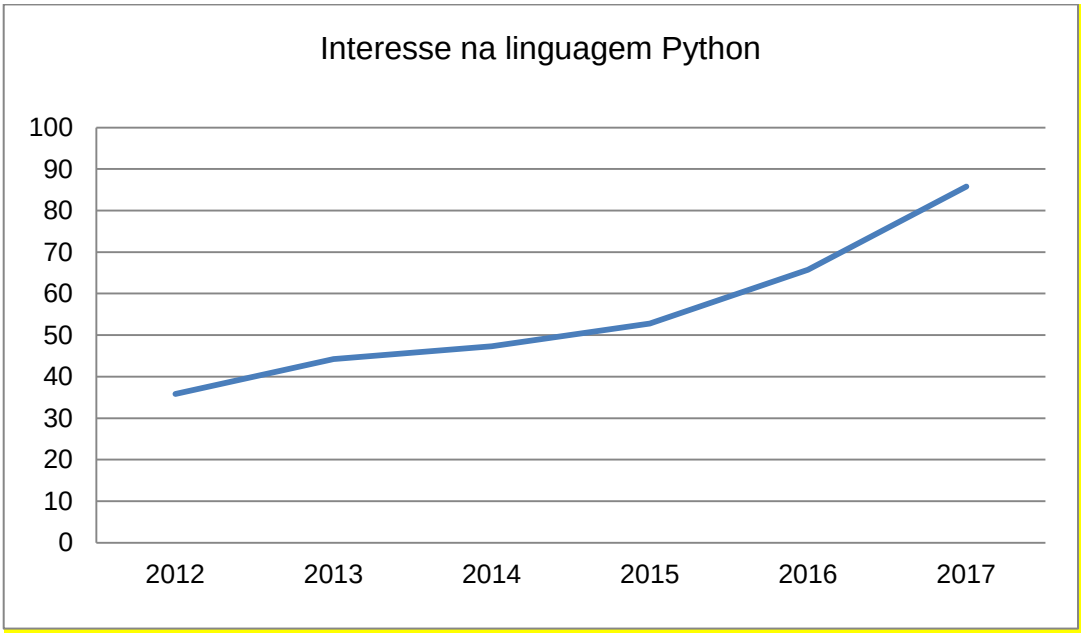
Dessa forma, a linguagem Python foi escolhida por se tratar de uma linguagem Orientada a Objetos (POO) acessível e poderosa.

O uso do Python é abrangente nos mais diversos campos do conhecimento. De acordo com a organização Python, existem diversos casos de sucesso no campo da Engenharia, ciência, arte, tecnologia e até mesmo em *business*.

Atualmente, Python é uma das linguagens de programação que mais cresce em países desenvolvidos do mundo, segundo Robinson (2017), o acesso às *tags* sobre a linguagem python no *website stack overflow*, um dos maiores fóruns mundial sobre programação, vêm crescendo rapidamente. Estima-se que o crescimento das pesquisas sobre a linguagem é de 27% ao ano.

Segundo os dados coletados na ferramenta *Google trends*, que mede o nível de interesse, em uma escala de 0 a 100, por determinado tópico pesquisado no *google* e em sites relacionados à ele, é possível notar um crescimento das pesquisas acerca da linguagem nos últimos cinco anos assim como mostra o gráfico abaixo.

Gráfico 1 - Pesquisas sobre a linguagem *Python* nos últimos 5 anos



Fonte: Elaborado pelo autor baseado em dados do Google Trends

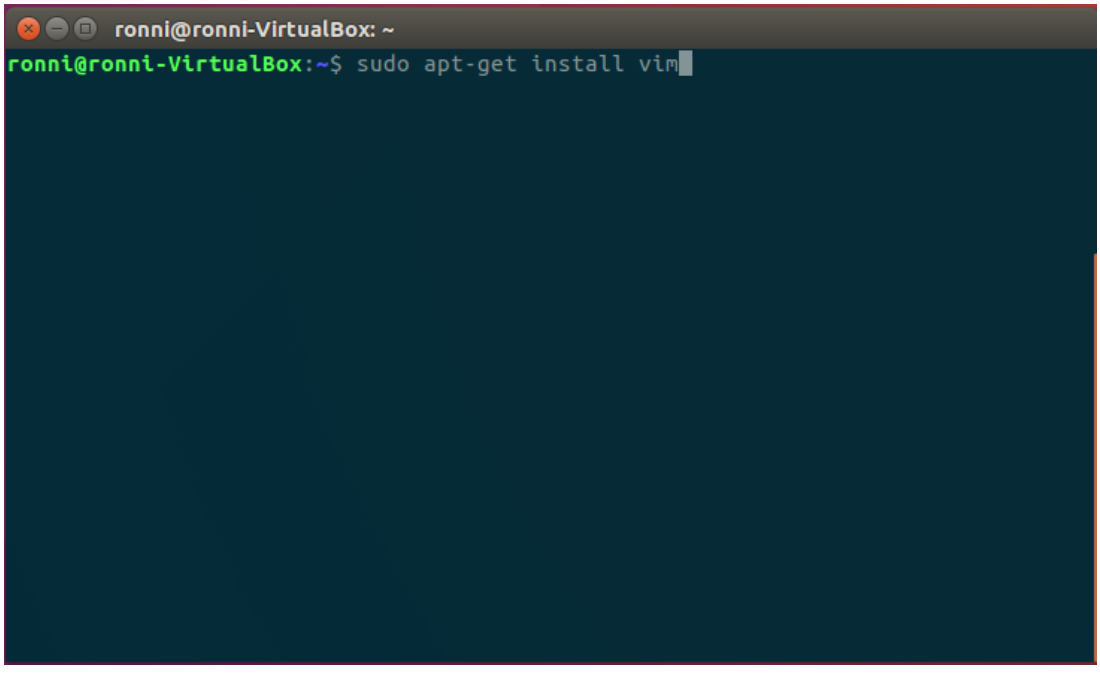
2.5 IMPROVED VI (VIM).

O *Vim* sigla para Vi melhorado (*Vi Improved*) é um editor de texto desenvolvido por Bram Moolenaar que permite a criação e edição de textos.

O editor está disponível para o uso gratuito de qualquer pessoa ou organização e é descrito pelos próprios desenvolvedores como um *charityware (*ou *careware)*, que em uma tradução livre significa *software* de caridade. De acordo com Lutus (2014), um *charityware* é um *software* disponibilizado por alguma entidade ou pessoa que pede aos usuários dos produtos qualquer tipo de pagamento exceto dinheiro. Este tipo de pagamento se estende à qualquer coisa, no entanto geralmente pede-se uma boa ação em troca do uso do programa como por exemplo: ajudar uma instituição de caridade, adotar um animal de estimação ou sempre dormir oito horas por dia.

O editor Vim já está incluso em sistemas UNIX e no Apple OS X. Para inicia-lo basta abrir o terminal de comando e digitar "*vim*". Apesar de o sistema operacional Linux ser baseado em UNIX, ele não inclui o *Vim* em sua instalação, desse modo é necessário que ele seja instalado através da ferramenta de manipulação de pacotes *apt-get*. Para efetuar a instalação basta digitar "*sudo apt-getinstall vim*" no terminal de comando da distribuição Linux.

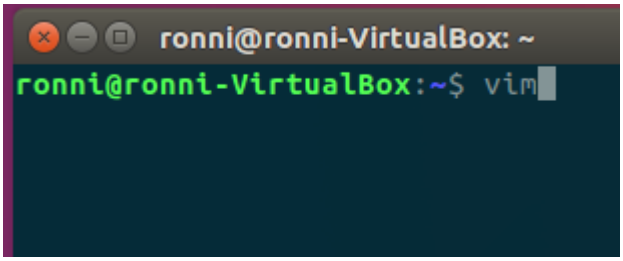
Figura 5 - Comando de instalação do Vim no Linux



Fonte: O autor (2017)

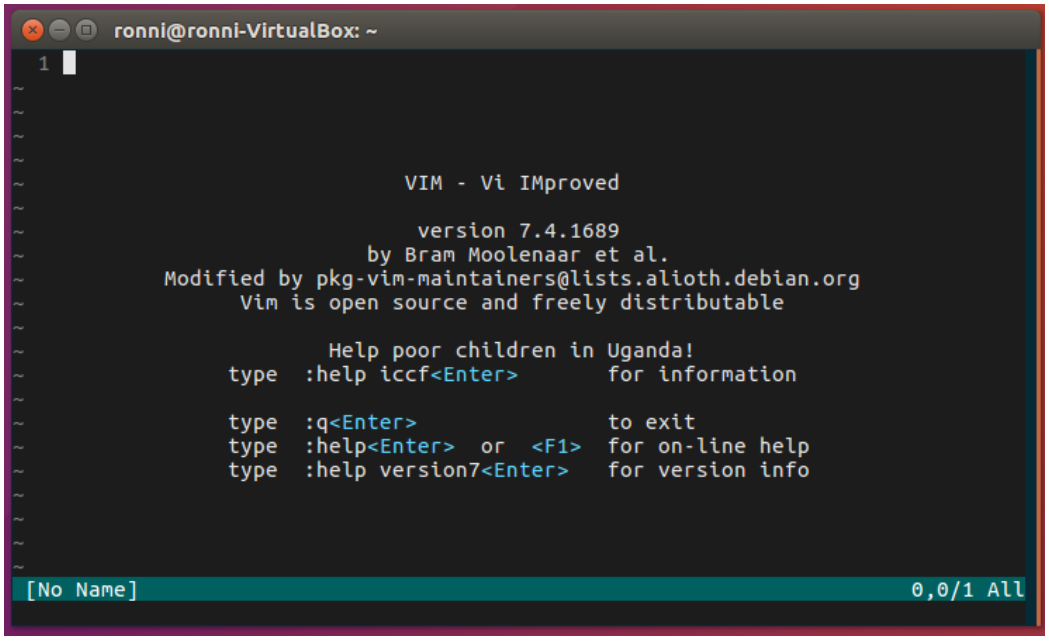
Feito isso, basta “digitar” vim no terminal de comando e o editor irá abrir assim como mostra a figura 6.

Figura 6 - Executando o Vim no Linux



Fonte: O autor (2017)

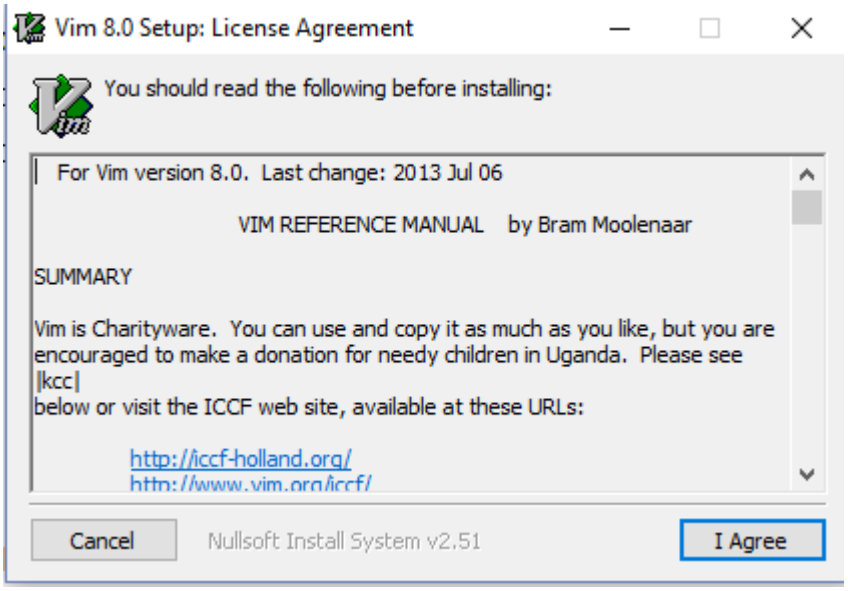
Figura 7 - Interface do Vim no sistema Linux



Fonte: O autor (2017)

Há também uma versão para Windows que pode ser encontrada no endereço https://vim.sourceforge.io/download.php. Ao entrar nesse endereço, terá uma opção no site para baixar um executável para MS Windows. Após o *Download*, basta executar o arquivo baixado. Uma janela com o manual de referência irá abrir (figura 8), para aceitar os termos se deve clicar em “*I agree*”.

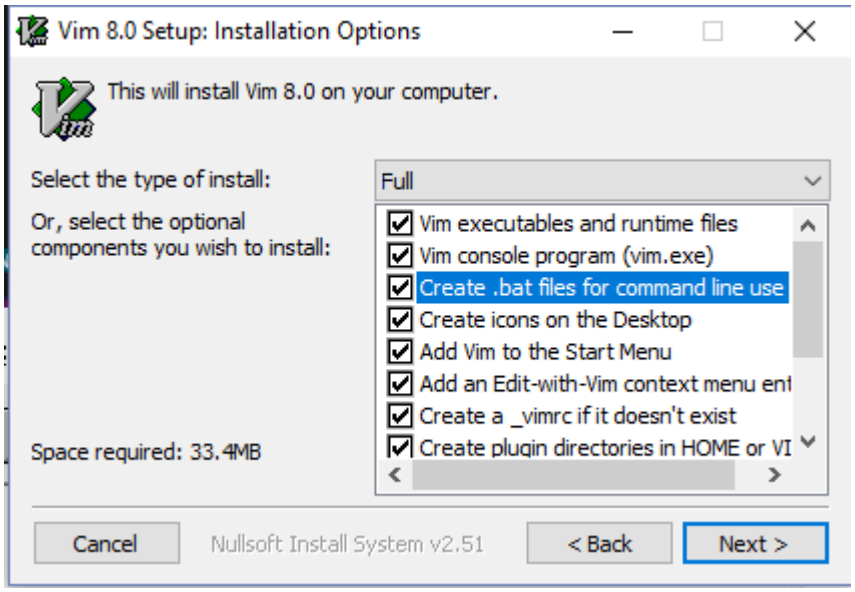
Figura 8 - Janela 1 de instalação do Vim no Sistema Windows



Fonte: O autor (2017)

Em seguida, uma janela para selecionar o tipo de instalação irá aparecer. O recomendado é que na parte "*Select the type of Install:*" (selecione o tipo de instalação), seja escolhida a opção *full* (Figura 9).

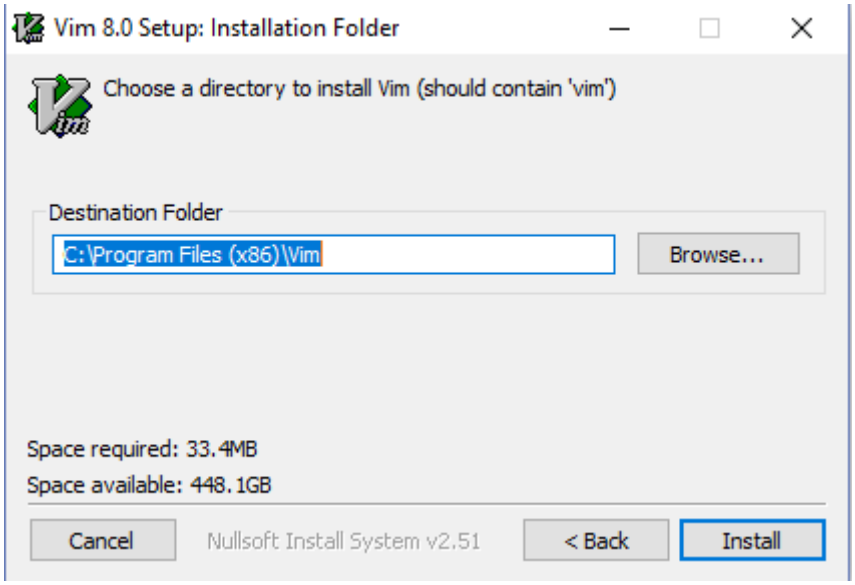
Figura 9 - Janela 2 de instalação do Vim no Sistema Windows



Fonte: O autor (2017)

Para finalizar a instalação, clique em *Next* e quando a próxima janela abrir, e seguidamente basta clicar em *install*.

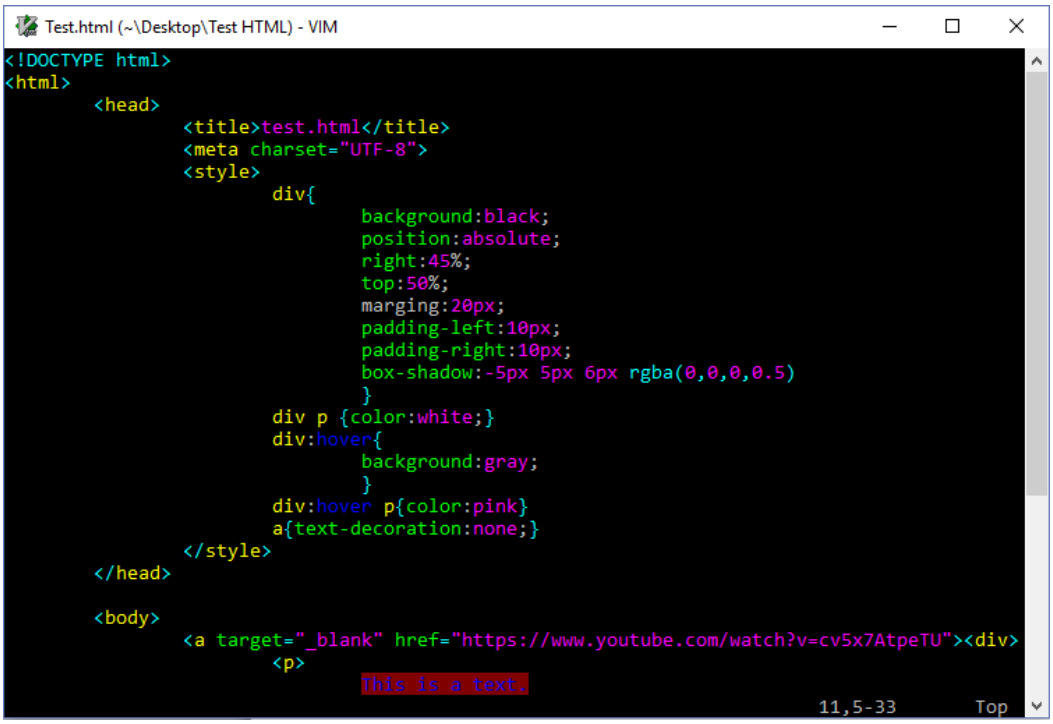
Figura 10 - Janela 3 de instalação do Vim no Sistema Windows



Fonte: O autor (2017)

Após a intalação, basta abrir o prompt de comando do *Windows* e digital "*Vim*", analogamente às versões de Mac OS X e Linux.

Figura 11 - Interface do Vim em um sistema Windows



Fonte: O autor (2017)

O *Vim* possui dois modos de trabalho:

a) Modo de comando: modo onde são executados comandos. Os três comandos mais básicos são ":w", que salva o arquivo sendo editado; ":q", que fecha o editor; e ":q!", que fecha o editor sem salvar as alterações no arquivo. Outros atalhos de teclado podem ser usados para navegar pelo texto. Alguns comandos são mostrados no quadro 3.
b) Modo de edição: para entrar no modo de edição basta apertar a tecla "i", ele permite que o arquivo seja editado. Para voltar ao modo de comando a tecla "*ESC*" deve ser pressionada.

Quadro 3 - Comandos básicos do Vim

Comando (atalho de teclado)	Função
A	Inicia o modo de edição colocando o cursor um caractere à frente de onde ele estava inicialmente.
Dd	Apaga uma linha
Yy	Copia uma linha
P	Cola uma linha
[número]gg	Vai para a linha de número especificado
[número]dd	Apaga um número de linhas especificado
[número]yy	Copia um número de linhas especificado
X	Apaga um caractere
[número]x	Apaga um número de caracteres especificado
U	Desfaz última modificação
Ctrl+r	Refaz última modificação

Fonte: O autor (2017)

2.6 Análise de Regressão

A análise de regressão é uma ferramenta estatística que visa verificar a existência de uma relação funcional entre uma variável dependente com uma ou mais variáveis independentes. Ou seja, ela faz a obtenção de uma equação que tenta explicar a variação da variável dependente através da variação das variáveis independentes (MONTGOMERY; RUNGER, 2009).

Tendo em vista que os dados plotados em um gráfico de dispersão de uma amostra ou de uma população são dados aleatórios, ou seja, acontecem ao acaso. Por isso, o gráfico da função de regressão não ira se ajustar perfeitamente aos dados apresentados no gráfico. Assim, a regressão tem o intuito de encontrar um modelo matemático que melhor se ajuste aos valores de Y em relação a X (MONTGOMERY; RUNGER, 2009).

O modelo trabalhado na presente obra é o linear simples, levando em consideração que será utilizada apenas uma variável dependente (capítulo V). Portanto, a fórmula usada para o cálculo de regressão está descrita abaixo:

$$Y_i = \beta_0 + \beta_1 X_i + e_i \ (8)$$

Sendo Y_i o valor da variável dependente observado no i-ésimo nível da variável independente; β_0 a constante de regressão que representa a interceptação da reta no eixo Y; β_1 o coeficiente de regressão, ou a variação de Y em relação à variação de unidades da variável X; X_i o i-ésimo valor da variável independente e e_i é o erro observado na distância entre o valor de Y_i e o ponto correspondente da curva para o mesmo nível i de X. A análise de regressão será feita em conjunto com a Análise de variância para validar o modelo de análise (CORREIA, 2003).

A análise de variância é um conjunto de modelos estatísticos para analisar a diferença entre grupos médios. Existem diversas variações de ANOVA e essas variações são aplicadas de acordo com o experimento realizado (PORTAL ACTION, 2017).

Para comprovar a validade do modelo de regressão será calculado através do *software Microsoft Excel* o Valor-p que é a probabilidade de uma hipótese H_0 ser verdadeira. Em outras palavras, quando o valor-p se aproxima de zero, então ele

aponta que pelo menos uma das variáveis dependentes contribuem para a variação da variável dependente. Além disso, será calculado o teste f de significância, que testa se há relação linear entre as variáveis Y e X, o que reforça a hipótese aceita ou rejeitada pelo valor-p (PORTAL ACTION, 2017).

CAPITULO III – METODOLOGIA DE PESQUISA

3.1 Estratégia e classificação de pesquisa

A estratégia e classificação de pesquisa foram baseadas nos conceitos descritos por Silva e Menezes (2005).

A presente pesquisa é classificada, quanto à natureza, como pesquisa aplicada, pois busca adquirir conhecimento para o desenvolvimento de uma aplicação para a resolução de um problema específico (método de tomada de decisão por AHP).

Já quanto à forma de abordagem do projeto, a pesquisa se classifica como qualitativa, tendo em vista que o trabalho busca descrever o desenvolvimento de uma ferramenta, o que não se pode ser medido em números ou através de métodos estatísticos.

Também, quanto ao objetivo do estudo a pesquisa é uma pesquisa descritiva, já que o trabalho descreve as características de um programa computacional, suas funcionalidades e possíveis problemas.

Por fim, quanto aos procedimentos técnicos, o estudo se enquadra em estudo de caso em razão de envolver um estudo profundo e exaustivo no método AHP de maneira detalhada para o desenvolvimento de um *software* para a tomada de decisão baseado no método.

3.2 Delineamento da pesquisa

O trabalho tem como foco o desenvolvimento de um *software* para tomada de decisões através do método AHP, o qual pode ser empregado em diversas áreas do conhecimento, como por exemplo, na decisão do melhor projeto para uma organização investir, problemas de localizações e logísticos, e até na escolha de uma planta de layout.

O *software* foi desenvolvido em uma distribuição *Mint* do Linux, que é um sistema operacional baseado em Linux Ubuntu, utilizando uma IDE chamada *Vim* (sigla para *VIl Mproved*).

O programa irá normalizar as matrizes e em seguida calcular suas respectivas consistências de modo que a mostrar o máximo autovalor (λ_{max}) de cada matriz,

seus índices de consistência (CI) e a Razão de Consistência (CR), a qual deve ser menor que 0,1 para que as matrizes sejam ditas consistentes.

Para a entrada de dados, serão utilizados dados do livro *Operational Research* do autor Wayne L. Winston e de Ricardo Viana Vargas para que seja feita a comparação com os resultados calculados pelo *software* (WINSTON, 2004; VARGAS, 2010).

3.3 Etapas de Pesquisa

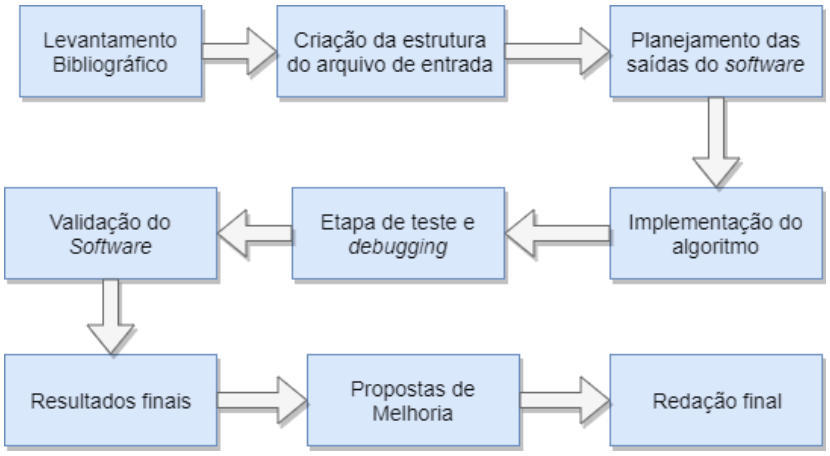
A pesquisa foi realizada em nove etapas principais. A primeira foi o levantamento bibliográfico para a completa compreensão do funcionamento e aplicações da ferramenta AHP, em seguida, precedendo a codificação do *software*, foram criadas a estrutura e mini sintaxe usadas para organizar os dados em um arquivo de texto (extensão .txt). Após a criação da estrutura do arquivo de entrada, foi planejado como o programa mostraria os resultados na tela, e quais etapas seriam apresentadas para o usuário.

Por conseguinte, fez-se a implementação do algoritmo na linguagem Python. Ao terminar a codificação, houve a etapa de *Debugging* para o conserto dos erros. Com todos os erros tratados, iniciou-se a validação do *software* através da entrada de dados de outros dois trabalhos (dados esses já tratados anteriormente pelos autores que os produziram). No primeiro teste, o foco da validação foi comparar detalhadamente os valorem encontrados por Winston (2004) em todas as etapas intermediárias de cálculo, enquanto que no último teste, atentou-se mais para os resultados finais, pois se buscou comparar estatisticamente a solução de *Odin* com a resolução feita por Vargas (2010) através do *Expert Choice,* tendo em conta que o último é uma solução consolidada e amplamente utilizada.

Para a análise estatística do segundo teste, aplicou-se a plotagem de um gráfico de dispersão, seguido de uma regressão linear e, por fim, uma análise de básica variância.

Após a validação do *software*, foi escrito os resultados finais e algumas propostas de melhorias futuras para o *software*, seguido da redação final do trabalho.

Figura 12 - Diagrama de Etapas da pesquisa



Fonte: O autor (2017)

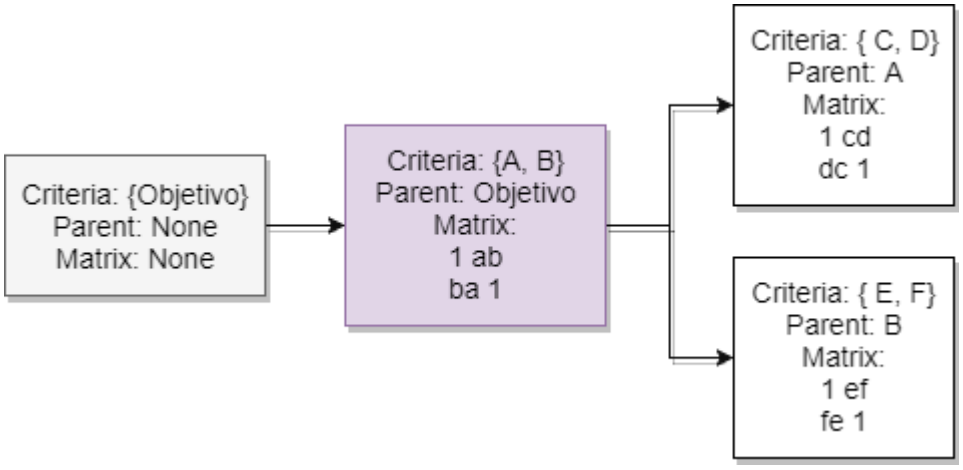
3.4 Funcionamento do algoritmo desenvolvido

Primeiramente criou-se uma classe chamada Matrizes. A classe faz as seguintes operações:

- Soma de cada coluna da matriz;
- Normalização de matrizes;
- Transposição
- Cálculo da média das linhas
- Checagem de consistência da matriz.

Após isso, criou-se duas estruturas de dados para facilitar o processo, a primeira é uma espécie de lista que divide as matrizes por níveis na arvore hierárquica permitindo guardar todos os nome dos critérios representados naquela matriz, o pai dos critérios da matriz e a matriz recíproca em si (figura 13).

Figura 13 - Primeira estrutura do programa



Fonte: O autor (2017)

A segunda estrutura é uma arvore genérica que receberá os pesos calculados para cada critério assim como mostra a figura 14.

Após a criação das estruturas e da classe matriz, foi escrito o código principal. Nele primeiro é perguntado ao usuário o *path* do arquivo que ele deseja abrir. O *software* irá tentar abrir o arquivo, mas caso não consiga, ele avisará ao usuário que o arquivo não existe. Por outro lado, caso exista, ele irá ler o arquivo e testar se não há nenhuma irregularidade na sintaxe do texto escrito no arquivo. Caso haja, o programa irá pedir ao usuário que conserte o problema e em seguida entre o arquivo outra vez. Caso o arquivo esteja sintaticamente correto, o texto ali presente será copiado para uma variável do tipo *string* e em seguida o arquivo será fechado.

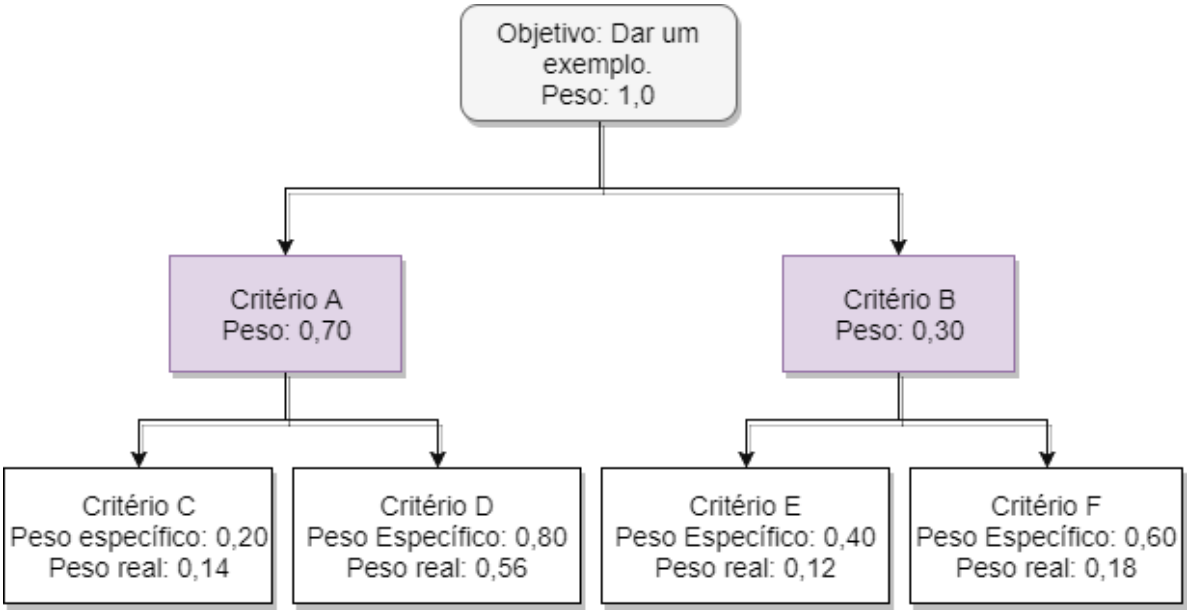
Com a variável do tipo *string*, o programa irá interpretar o arquivo por níveis, e irá armazenar cada matriz junto com a informação da descendência do critério, por exemplo, considere uma matriz de três critérios x, y e z que deriva de um critério M de uma outra matriz do nível 2, assim será guardado o nome dos três critérios mais a matriz recíproca de julgamento pareado, e o nome do parente M. Feito isso, cada matriz será exibida na tela para o usuário.

Após a exibição das matrizes brutas, o programa irá normaliza-las e verificar as consistências para cada uma delas. Esse processo será feito por níveis de hierarquia.

Para a finalização do cálculo dos critério, o programa irá montar uma hierarquia com todos os pesos dos critérios calculados. É importante frisar que quando um critério é filho de outro critério, o resultado parcial calculado para aquele

critério será multiplicado pelo resultado do critério anterior. Por exemplo, considere que o critério M, cujo valor parcial é 0,7 seja filho de um critério L, que vale 0,5. Então o peso geral de M é 0,35 como mostrado na Figura 14.

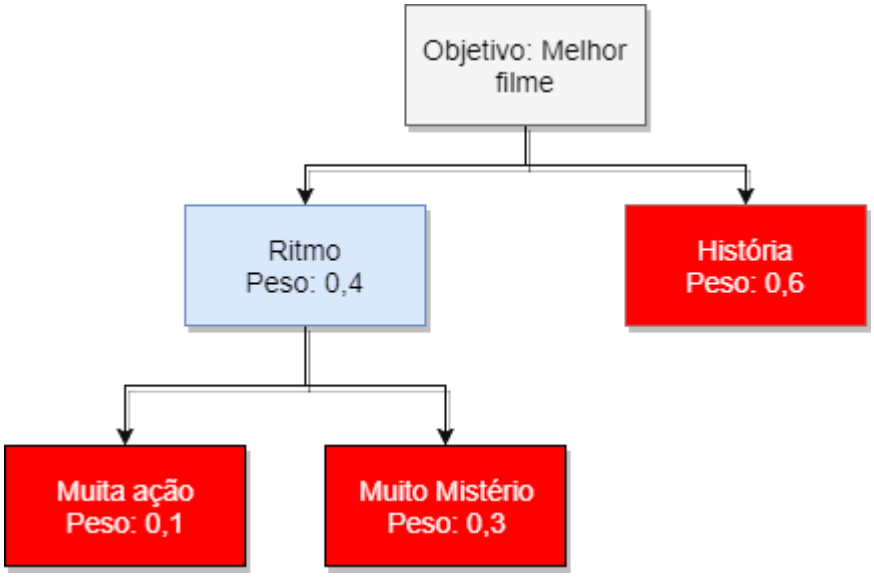
Figura 14 - Segunda estrutura do programa e representação do cálculo de pesos



Fonte: O autor (2017)

Para finalizar a parte dos critérios o código criará um vetor com a os valores das pontas da hierarquia como mostra os nódulos em vermelho da figura 15. Nesse caso o vetor seria (0,1; 0,3; 0,6).

Figura 15 - Exemplo de escolha de pesos para a montagem do vetor final de pesos



Fonte: O autor (2017)

Como próximo passo, o programa irá outra vez perguntar para o usuário digitar o nome de outro arquivo. Esse arquivo será interpretado de forma análoga ao arquivo de critérios e dele será montado uma lista com todas as matrizes de alternativas para cada critério da ponta da hierarquia. O mesmo procedimento de normalização, e calculo de pesos para cada alternativa de acordo com o critério analisado. Todos os resultados serão montados em uma matriz onde as linhas i representarão as opções e as colunas j representarão os critérios, assim cada item da matriz será o peso da opção i em relação ao critério j. Essa matriz também será mostrada ao usuário no fim do processo.

Para finalização, a matriz de resultados para cada critério será transposta e multiplicada pelo vetor com os pesos dos critérios da ponta da hierarquia. Essa multiplicação irá gerar o peso global para cada alternativa e o programa volta a pedir outro arquivo para efetuar o método novamente.

O programa finaliza apenas quando o usuário escrever a palavra “*end*” no *prompt*. Ou seja, quando ao invés do nome do arquivo, o usuário digitar a palavra “*end*”, o programa chamará um método que irá fecha-lo. Caso a palavra “*end*” não seja escrita, o *software* continuará funcionando.

3.5 Atribuição do nome do *software*

Por se tratar de uma aplicação que auxilia na tomada de decisão, o *software* recebeu o nome de *Odin*, que segundo a mitologia nórdica, era o deus associado à sabedoria e ao conhecimento. Segundo Gaiman (2017), *Odin* era o mais velho dos deuses e pai de todos. Ele sabia de muitos segredos e perdeu um dos olhos em troca de sabedoria.

CAPITULO IV – FUCIONAMENTO E TRATAMENTO DE EXCEÇÕES

Neste capítulo serão abordados as funcionalidades do *software* e alguns tratamentos de exceções.

4.1 Montagem dos arquivos para a entrada de dados

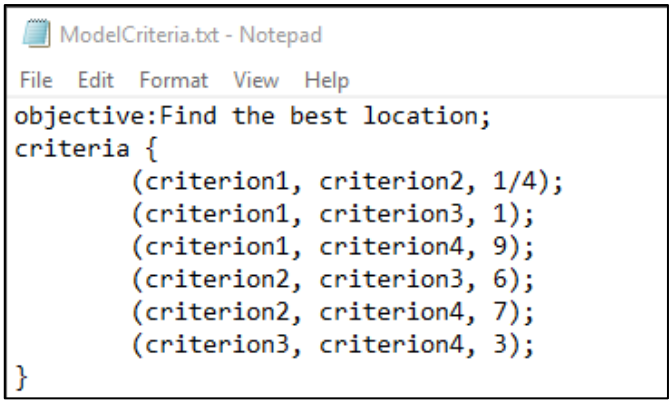
Para fazer o cálculo completo de uma hierarquia AHP, o *software* irá pedir ao usuário para indicar o caminho de dois arquivos de texto. O primeiro é o arquivo aonde vai estar representada a hierarquia com o objetivo, os critérios e os subcritérios. Enquanto que o segundo informará as alternativas, ou opções, que serão julgadas após a determinação dos pesos.

4.1.1 Arquivo de critérios

O arquivo de critérios começa primeiramente com a indicação do objetivo. Para isso é necessário escrever "*objective:*" seguido de qualquer tipo de texto e terminando com um ponto e vírgula. Após isso, vem a descrição dos critérios principais (segundo nível da hierarquia). Ela é feita primeiramente escrevendo *criteria* e entre as chaves devem estar todos os julgamentos possível para a construção da matriz de comparação. A palavra *criteria* indica que tudo o que está entre chaves diz respeito aos critérios principais da hierarquia. Todos os julgamentos devem ser feitos entre parênteses e finalizados com um ponto-e-vírgula. Por exemplo, se os critérios a serem comparados são "beber água" e "correr", a comparação será descrita como "(beber_agua, correr, 7);", assim interpreta-se, de acordo com a escala Saaty, que beber água é fortemente mais importante (conceito 7) que correr. Caso contrário, então a comparação seria "(beber_agua, correr, 1/7);", ou seja, correr é fortemente mais importante (conceito 7) que beber água.

É de extrema importância ressaltar que o programa sempre irá comparar o primeiro critério escrito com o segundo e nunca o contrário. A figura 16 representa todos os passos até o momento descritos.

Figura 16 - Modelo para a descrição de objetivo e critérios principais



Fonte: O autor (2017)

Caso haja subcritérios, eles devem ser colocados de acordo com o nível em que se encontram depois do nível de critérios principais. Para indicar a existência de subcritérios deve ser escrito "*sub-criteria-(nível)*". Para exemplificar, digamos que o primeiro nível de subcritérios está sendo construído, então a palavra chave será escrita como "*sub-criteria-1*", caso seja o segundo, então "*sub-criteria-2*". Dentro das chaves são colocados os julgamentos de maneira similar ao primeiro passo, porém dessa vez deve-se indicar os critérios dos quais estes subcritérios são derivados, para isso usa-se a palavra-chave "*parent*", a qual significa pai, seguido de dois pontos mais o nome do critério que derivou aqueles subcritérios que vão ser comparados. Em seguida, abrem-se chaves e dentro é feita as comparações entre os critérios. Caso os critérios principais não tenham subcritérios, a interpretação deste passo é dispensada e o programa apenas trabalhará com os critérios principais.

A figura 17 descreve como os subcritérios devem estar representados.

Figura 17 – Modelo de descrição de subcritérios

```
sub-criteria-1{
        parent: criterion1{
                (sub-criterion1, sub-criterion2, 4);
        }
        parent: criterion2{
                (sub-criterion3, sub-criterion4, 4);
        }
        parent: criterion3{
                (sub-criterion5, sub-criterion6, 4);
        }
        parent: criterion4{
                (sub-criterion7, sub-criterion8, 4);
        }
}
sub-criteria-2{
        parent: sub-criterion1{
                (sub-sub-criterion1, sub-sub-criterion2, 9);
        }
        parent: sub-criterion3{
                (sub-sub-criterion3, sub-sub-criterion4, 2);
                (sub-sub-criterion3, sub-sub-criterion5, 8);
                (sub-sub-criterion3, sub-sub-criterion6, 1/4);
                (sub-sub-criterion4, sub-sub-criterion5, 1/7);
                (sub-sub-criterion4, sub-sub-criterion6, 9);
                (sub-sub-criterion5, sub-sub-criterion6, 7);
        }
}
```

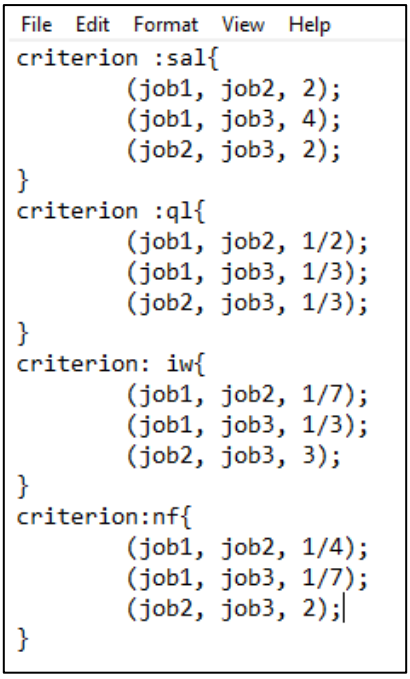
Fonte: O autor (2017)

O exemplo mostrado nas figuras a e b pode ser processado pelo programa, mas não se chegará a nenhum resultado significativo, tendo em conta que os valores nele presente são inventados e meramente ilustrativos. Exemplos de estudos reais serão mostrados no próximo capítulo onde é feita a validação do *software*.

4.1.2 Arquivo de Alternativas

No arquivo de alternativas estará a comparação de cada alternativa em relação à outra para cada critério dos últimos níveis da hierarquia. Para indicar a qual critério cada grupo de comparação pertence é necessário que se use a palavra-chave “*criterion*” seguido de dois pontos mais o nome do critério. Após isso, abre-se chaves e entre elas são escritas todas as comparações possíveis entre as alternativas. A figura 18 mostra um exemplo de um arquivo de alternativas.

Figura 18 - Arquivo de alternativas



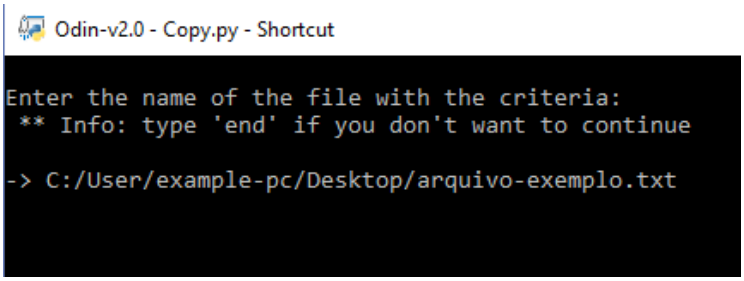
```
File  Edit  Format  View  Help
criterion :sal{
        (job1, job2, 2);
        (job1, job3, 4);
        (job2, job3, 2);
}
criterion :ql{
        (job1, job2, 1/2);
        (job1, job3, 1/3);
        (job2, job3, 1/3);
}
criterion: iw{
        (job1, job2, 1/7);
        (job1, job3, 1/3);
        (job2, job3, 3);
}
criterion:nf{
        (job1, job2, 1/4);
        (job1, job3, 1/7);
        (job2, job3, 2);
}
```

Fonte: O autor (2017)

4.2 Funcionamento do programa

Ao abrir o programa, ele irá pedir primeiramente o arquivo com os dados da hierarquia de critérios, assim como na figura 19.

Figura 19 - Input do arquivo de critérios no Odin

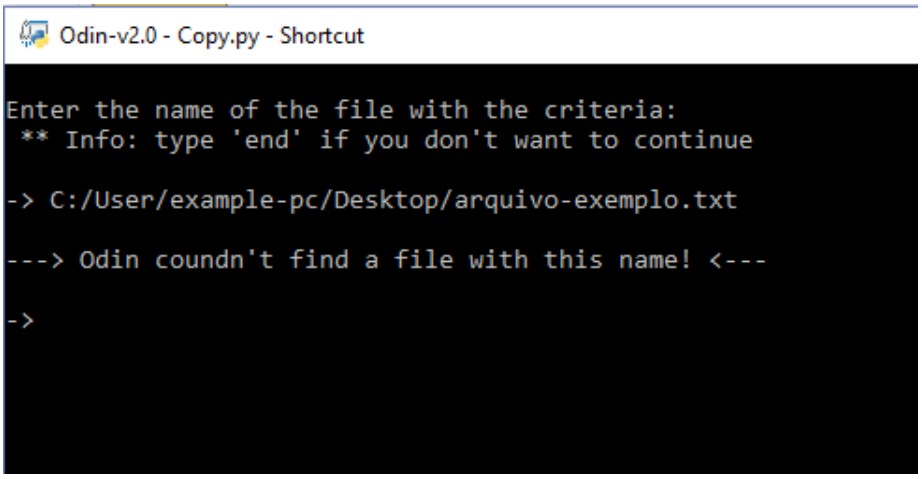


```
Odin-v2.0 - Copy.py - Shortcut

Enter the name of the file with the criteria:
 ** Info: type 'end' if you don't want to continue

-> C:/User/example-pc/Desktop/arquivo-exemplo.txt
```

Fonte: O autor (2017)

Para abrir o arquivo, o programa necessitará do path completo. Caso o arquivo não exista, o *software* irá avisar o usuário e pedir para que o mesmo tente outra vez.

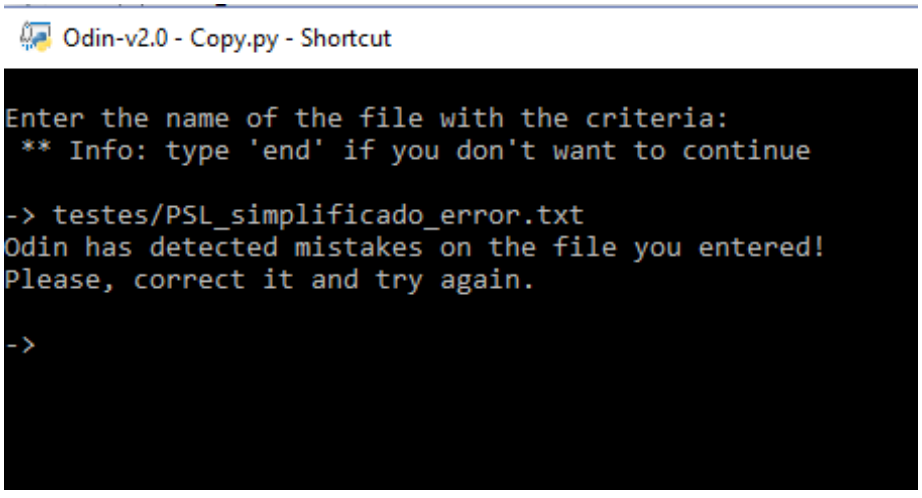
Figura 20 - Erro de arquivo não encontrado



Fonte: O autor (2017)

Caso haja algum erro na formatação tanto do arquivo de critérios quanto no arquivo de alternativas, como por exemplo, a falta de alguma pontuação, ou o erro na digitação de alguma palavra-chave, ou até mesmo se o usuário entrar com nomes diferentes de um mesmo critério, o programa irá dizer que há algum problema com o arquivo e pedirá para o usuário o corrigir antes de tentar abri-lo outra vez.

Figura 21 - Erro ao entrar um arquivo de texto



Fonte: O autor (2017)

Por fim, caso tudo ocorra bem, o *software* irá calcular as matrizes normalizadas, os pesos de cada critério, a consistência de cada matriz e o resultado final listando o peso final para cada alternativa.

O programa continuará funcionando para que sejam imputados mais arquivos, assim, é possível fazer diversas análises sem ter que executar o programa outra vez. Para fechar *Odin*, basta escrever “*end*” em qualquer uma das vezes que for possível entrar com o *path* dos arquivos, por esse motivo é sábio não nomear arquivos com a palavra “*end*”.

CAPÍTULO V – VALIDAÇÃO

O presente capítulo apresentará a etapa de validação do *software*, onde foram imputados dados de outras obras para comprovar o funcionamento e efetividade do mesmo em relação a outros *softwares* já amplamente utilizados no mercado. O primeiro teste foi feito com dados retirados de Winston (2004) e o segundo extraiu dados de Vargas (2010).

5.1 Primeiro teste

Ao falar de AHP, Winston usa um exemplo em seu livro para apresentar o método de forma didática. O exemplo diz que Jane, uma pessoa que está em busca de um emprego, recebe três propostas de trabalho de acordo com quatro critérios: Valor de salário inicial alto (SAL), qualidade de vida na cidade onde o emprego foi oferecido (QL), interesse naquele tipo de trabalho (IW), e a proximidade da família (NF).

Figura 22 - Matriz recíproca de critérios gerada pelo *software* Odin para Winston

```
Objective: FIND THE BEST JOB OPTION

---------------------------------------
Parent:  OBJECTIVE
SAL QL IW NF

1.0|5.0|2.0|4.0|
0.2|1.0|0.5|0.5|
0.5|2.0|1.0|2.0|
0.25|2.0|0.5|1.0|
```

Fonte: O autor (2017)

Figura 23 - Matriz recíproca de análise pareada de critérios do exemplo de Winston.

	SAL	QL	IW	NF
SAL	1	5	2	4
QL	$\frac{1}{5}$	1	$\frac{1}{2}$	$\frac{1}{2}$
IW	$\frac{1}{2}$	2	1	2
NF	$\frac{1}{4}$	2	$\frac{1}{2}$	1

Fonte: Winston (2004)

O programa monta as matrizes de forma automática, porém a única diferença notada é a resolução imediata das frações nela contida, o que não é um problema para a análise.

A matriz foi montada a partir do arquivo de texto demonstrado na figura 24. Todos os detalhes de como construir os arquivos de entrada de dados já foram descritos no capítulo anterior.

Figura 24 - Arquivo com os dados de julgamento dos critérios para teste 1

```
objective:Find the best job option;

criteria {
        (SAL, QL, 5);
        (SAL, IW, 2);
        (SAL, NF, 4);
        (QL, IW, 1/2);
        (QL, NF, 1/2);
        (IW, NF, 2);
}
```

Fonte: O autor (2017)

Após a montagem da matriz recíproca, há a fase de normalização, a figura 25 mostra os resultados da normalização feita por Winston, e em seguida, na figura 27 está a matriz normalizada pelo *software*. Como pode ser observado, os valores estão muito próximos assim como os valore de λ, que para Winston foi de 4,05 e para o *software* ficou em 4,0476; CI, que para Winston é 0,017 e para o *software Odin* é de 0,0159; e o CR, que para Winston é 0,019 e para o *software Odin* é de 0,0176. Em ambos os casos, os cálculos consideraram a matriz como consistente.

Figura 25 - Matriz normalizada calculada por Winston

$$A_{\text{norm}} = \begin{bmatrix} .5128 & .5000 & .5000 & .5333 \\ .1026 & .1000 & .1250 & .0667 \\ .2564 & .2000 & .2500 & .2667 \\ .1282 & .2000 & .1250 & .1333 \end{bmatrix}$$

Fonte: Winston (2004)

Figura 26 - Pesos dos critérios calculados por Winston.

$$w_1 = \frac{.5128 + .5000 + .5000 + .5333}{4} = .5115$$

$$w_2 = \frac{.1026 + .1000 + .1250 + .0667}{4} = .0986$$

$$w_3 = \frac{.2564 + .2000 + .2500 + .2667}{4} = .2433$$

$$w_4 = \frac{.1282 + .2000 + .1250 + .1333}{4} = .1466$$

Fonte: Winston (2004)

Figura 27 - Matriz normalizada, pesos dos critérios e consistência calculados por Odin

```
############### Solving Main Weights ##############


SAL QL IW NF
0.5128|0.5|0.5|0.5333|
0.1026|0.1|0.125|0.0667|
0.2564|0.2|0.25|0.2667|
0.1282|0.2|0.125|0.1333|
SAL :  0.5115

QL :  0.0986

IW :  0.2433

NF :  0.1466

Lambda:  4.0476
CI:  0.0159
CR:  0.0176
```

Fonte: O autor (2017)

Para concluir, fez-se a comparação dos cálculos das matrizes de opções. Infelizmente, o cálculo das consistências das matrizes de opções não foi revelado em Winston (2004), portanto, torna-se impossível fazer a comparação destas, deste modo, serão feitas apenas as comparações dos resultados parciais para cada critério. De forma análoga, as matrizes normalizadas de pesos só foram mostradas para os critérios salário e qualidade de vida, assim, apenas essas matrizes poderão ser comparadas.

Para a montagem das matrizes, um arquivo de texto também foi gerado como mostra a figura 28.

Figura 28 - Arquivo com os dados de comparação das alternativas para teste 1

```
criterion :sal{
        (job1, job2, 2);
        (job1, job3, 4);
        (job2, job3, 2);
}
criterion :ql{
        (job1, job2, 1/2);
        (job1, job3, 1/3);
        (job2, job3, 1/3);
}
criterion: iw{
        (job1, job2, 1/7);
        (job1, job3, 1/3);
        (job2, job3, 3);
}
criterion:nf{
        (job1, job2, 1/4);
        (job1, job3, 1/7);
        (job2, job3, 2);
}
```

Fonte: O autor

As figuras 29 e 30 mostram as matrizes geradas por Winston e pelo *software* Odin respectivamente. Da mesma forma, a figura 31 mostra a matriz normalizada de pesos calculada por Winston enquanto que a figura 32 monstra o mesmo calculado pelo programa.

Figura 29 - Matriz de comparação para o critério SAL montada por Winston

	Job 1	Job 2	Job 3
Job 1	1	2	4
Job 2	$\frac{1}{2}$	1	2
Job 3	$\frac{1}{4}$	$\frac{1}{2}$	1

Figura H:. Fonte: Winston (2004)

Figura 30 - Matriz de comparação para o critério SAL montada por Odin

```
Criterion:  SAL

JOB1    JOB2    JOB3
1.0   2.0   4.0
0.5   1.0   2.0
0.25   0.5   1.0
```

Fonte: O autor (2017)

Figura 31 - Matriz normalizada de pesos para SAL montada por Winston

$$A_{\text{norm}} = \begin{bmatrix} .571 & .571 & .571 \\ .286 & .286 & .286 \\ .143 & .143 & .143 \end{bmatrix}$$

Fonte: Winston (2004)

Figura 32 - Matriz normalizada de pesos para SAL montada por Odin.

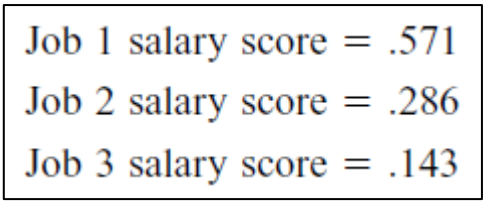
```
Criterion:  SAL
JOB1    JOB2    JOB3
0.5714   0.5714   0.5714
0.2857   0.2857   0.2857
0.1429   0.1429   0.1429
```

Fonte: O autor (2017)

Os valores de ambas as matrizes estão bem próximos. Agora quando a consistência é calculada para a matriz correspondente ao critério salário, achamos que o λ é igual a três e o valor de CI é 0 assim como o CR, ou seja, a inconsistência dessa matriz é de 0%.

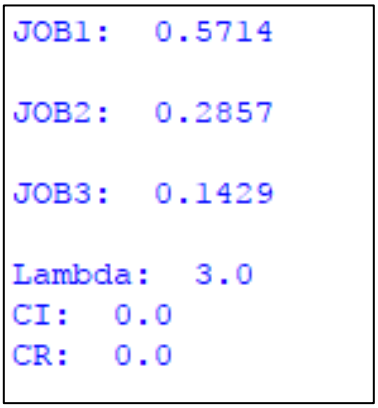
Quanto aos resultados parciais para o critério SAL a proposta de emprego 1 é a preferida, seguida da proposta 2 como a segunda melhor e a 3 como a proposta menos favorável como exibido em ambas as imagens 33 e 34, sendo 33 calculado por Winston (2004) e outro 34 calculado pela ferramenta computacional.

Figura 33 - Pesos parciais das alternativas para SAL calculados por Winston



Fonte: Winston (2004)

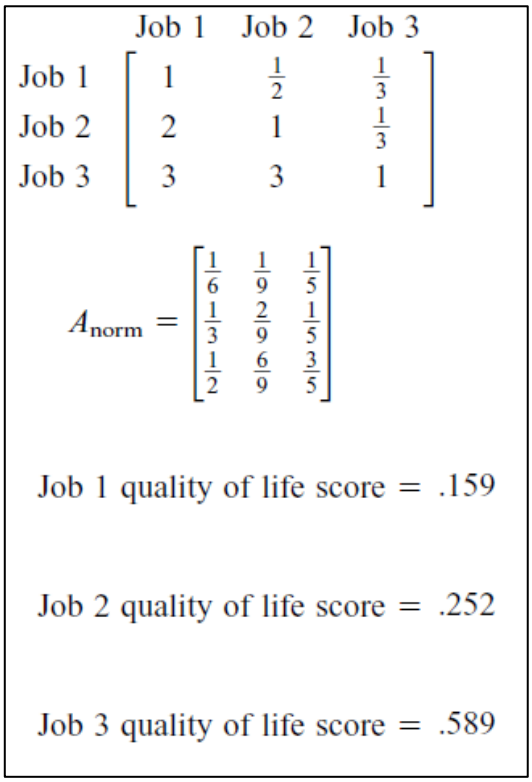
Figura 34 - Pesos parciais das alternativas para SAL calculados em *Odin*



Fonte: O autor (2017)

A figura 35 mostra a matriz e comparação para o critério qualidade de vida assim como a matriz normalizada de pesos para o mesmo critério e os resultados parciais para esse critério. No critério qualidade de vida na cidade, a proposta de emprego 3 teve um conceito maior, logo em seguida vêm as opções 2 e 1. Na imagem 36 é mostrado o mesmo cálculo, agora realizado com o *software Odin*, mais as variáveis de consistência. Considerando que o CR é igual a 0,0465, então a matriz é consistente.

Figura 35 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério QL calculados por Winston



$$\begin{array}{c} \\ \text{Job 1} \\ \text{Job 2} \\ \text{Job 3} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Job 1} & \text{Job 2} & \text{Job 3} \end{array} \\ \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 2 & 1 & \frac{1}{3} \\ 3 & 3 & 1 \end{bmatrix} \end{array}$$

$$A_{\text{norm}} = \begin{bmatrix} \frac{1}{6} & \frac{1}{9} & \frac{1}{5} \\ \frac{1}{3} & \frac{2}{9} & \frac{1}{5} \\ \frac{1}{2} & \frac{6}{9} & \frac{3}{5} \end{bmatrix}$$

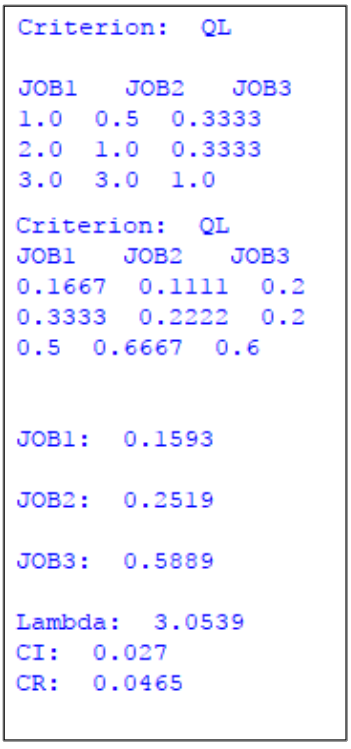
Job 1 quality of life score = .159

Job 2 quality of life score = .252

Job 3 quality of life score = .589

Fonte: Winston (2004)

Figura 36 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério QL calculados em *Odin*.

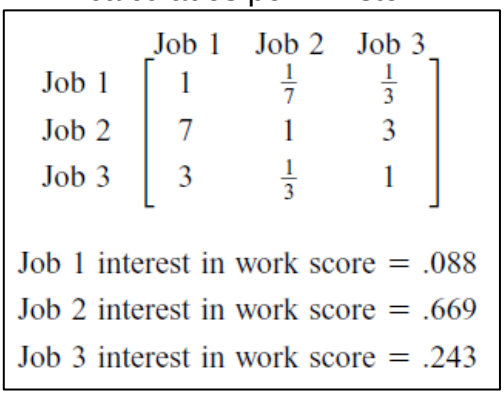


```
Criterion:  QL

JOB1    JOB2    JOB3
1.0   0.5   0.3333
2.0   1.0   0.3333
3.0   3.0   1.0

Criterion:  QL
JOB1    JOB2    JOB3
0.1667   0.1111   0.2
0.3333   0.2222   0.2
0.5   0.6667   0.6


JOB1:   0.1593

JOB2:   0.2519

JOB3:   0.5889

Lambda:   3.0539
CI:   0.027
CR:   0.0465
```

Fonte: O autor (2017)

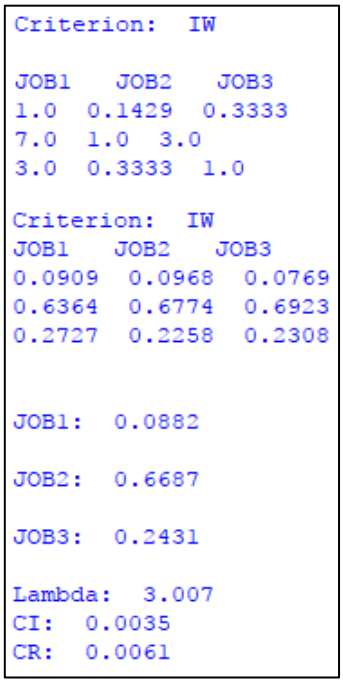
Ulteriormente, tem-se o resultado para o critério interesse no tipo de trabalho. A imagem 37 mostra a matriz de julgamento para IW e suas respectivas prioridades parciais calculados por Winston (2004). Já a imagem 38 expõe a montagem automática da matriz de comparação para IW e a matriz normalizada e suas prioridades parciais para o mesmo critério. A matriz, segundo o programa, é consistente, pois o valor do CR é menor que 0,1. Quanto às prioridades parciais para Interesse no trabalho, o trabalho 2 foi o mais indicado, seguido do trabalho 3 e o menos favorável é o trabalho 1. O programa obteve o mesmo resultado para as prioridades parciais de IW.

Figura 37 - Matrizes de julgamento e de pesos e prioridade relativa para o critério IW calculados por Winston



Fonte: Winston (2004)

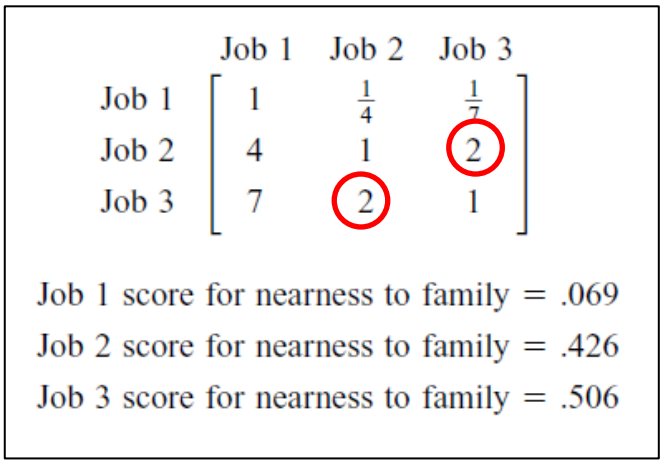
Figura 38 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério IW calculados em Odin



Fonte: O autor (2017)

Finalmente, é analisado o critério proximidade da família NF nas figuras 39 e 40. Para este último critério, as prioridades relativas foram, segundo Winston, proposta 3, em seguida proposta 2, e por último a proposta 1. Já segundo o cálculo efetuado pelo programa *Odin*, a prioridade foi proposta 2, proposta 3 e em último lugar a proposta 1.

Figura 39 - Matrizes de julgamento e de pesos, e prioridade relativa para o critério NF calculados por Winston.

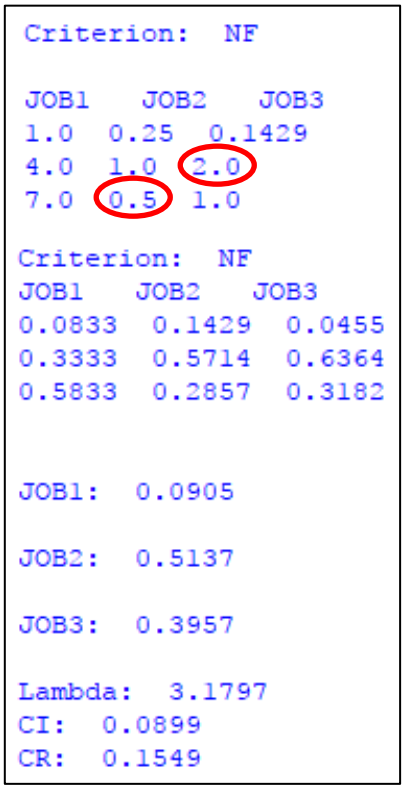


	Job 1	Job 2	Job 3
Job 1	1	$\frac{1}{4}$	$\frac{1}{7}$
Job 2	4	1	2
Job 3	7	2	1

Job 1 score for nearness to family = .069
Job 2 score for nearness to family = .426
Job 3 score for nearness to family = .506

Fonte: Winston (2004)

Figura 40 - Matrizes de julgamento e de pesos, prioridade relativa e consistência das alternativas para o critério NF calculados em Odin

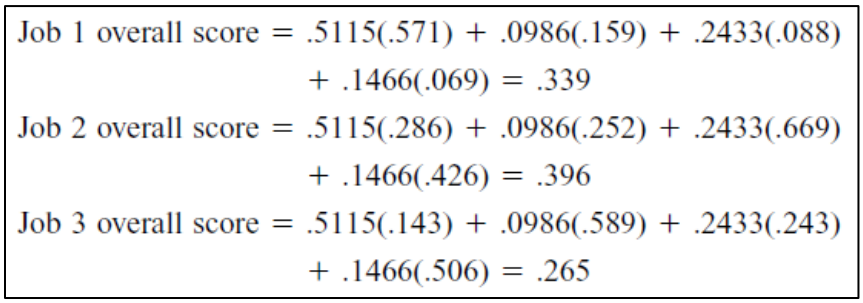


```
Criterion:  NF

JOB1    JOB2    JOB3
1.0   0.25   0.1429
4.0   1.0    2.0
7.0   0.5    1.0

Criterion:  NF
JOB1    JOB2    JOB3
0.0833   0.1429   0.0455
0.3333   0.5714   0.6364
0.5833   0.2857   0.3182


JOB1:   0.0905

JOB2:   0.5137

JOB3:   0.3957

Lambda:   3.1797
CI:   0.0899
CR:   0.1549
```

Fonte: O autor (2017)

Com relação ao resultado final, foi encontrado que a proposta de trabalho 2tem prioridade maior, seguido do trabalho 1 e, como menos desejável, o trabalho 3 tanto para o *software Odin* quanto para Winston. Porém os valores finais para cada proposta foram totalmente diferentes, dado que para a proposta1 o valor calculado por Winston foi de 0.339, para proposta dois foi 0,396 e para a proposta 3 foi de 0,265. Enquanto que o *Software Odin* constatou que a proposta 1 tem valor de 0,3427, a proposta 2 tem um valor de 0,4090 e a proposta três tem o valor de 0,2483.

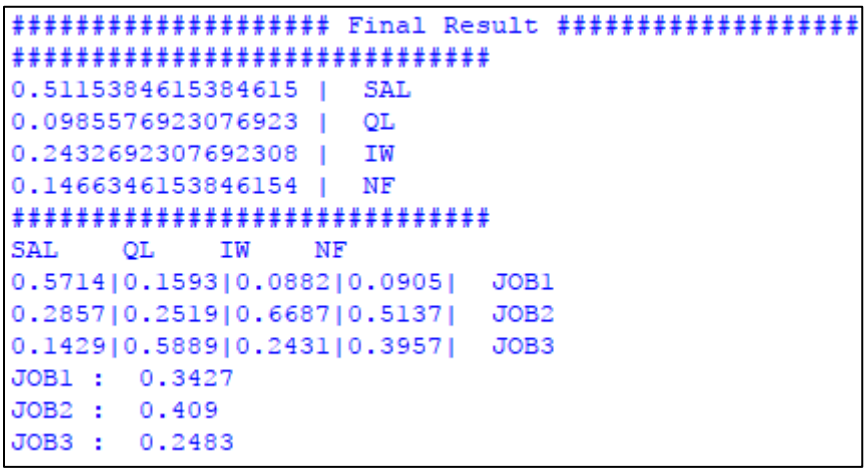
Figura 41 - Prioridade (resultado) final das alternativas por Winston



Job 1 overall score = .5115(.571) + .0986(.159) + .2433(.088)
+ .1466(.069) = .339
Job 2 overall score = .5115(.286) + .0986(.252) + .2433(.669)
+ .1466(.426) = .396
Job 3 overall score = .5115(.143) + .0986(.589) + .2433(.243)
+ .1466(.506) = .265

Fonte: Winston (2004)

Figura 42 - Prioridade (resultado) final das alternativas por *Odin*



```
#################### Final Result ####################
##############################
0.5115384615384615 |  SAL
0.0985576923076923 |  QL
0.2432692307692308 |  IW
0.1466346153846154 |  NF
##############################
SAL     QL     IW     NF
0.5714|0.1593|0.0882|0.0905|  JOB1
0.2857|0.2519|0.6687|0.5137|  JOB2
0.1429|0.5889|0.2431|0.3957|  JOB3
JOB1 :  0.3427
JOB2 :  0.409
JOB3 :  0.2483
```

Fonte: O autor (2017)

Os resultados encontrados pelo programa na análise do último critério (NF), e também nos resultados finais foram totalmente diferentes daqueles apresentados por Winston. Este fato aconteceu devido a um erro por parte de Winston que foi descoberto pelo *software* após a construção automática da matriz de comparação.

Ao especificar na matriz que a proposta 2 teria uma importância maior que a proposta 3 (A_{23} = 2), logo a importância da proposta 3 em relação à proposta 2 deveria ser ½ (A_{32} = ½). Todavia, o autor acaba atribuindo 2 às duas comparações, o que quebra a regra da reciprocidade. Esse erro foi destacado nas figuras 39 e 40.

É importante frisar que o sistema *Odin* constatou que a matriz para NF é inconsistente, tendo em vista que seu CR foi de 0.1549, um valor maior que 0.1.

Para propor uma solução para o cálculo de Winston, ao considerar que tanto em A_{23} quanto A_{32} são iguais a 2, então pode ser considerado que as propostas 2 e 3 têm a mesma importância quando à esse critério, o que nos leva entender que A_{23} = A_{32} = 1. Assim o arquivo de comparação das alternativas for alterado como mostra a figura 43.

Figura 43 - Arquivo alterado de alternativas de trabalho

```
criterion :sal{
        (job1, job2, 2);
        (job1, job3, 4);
        (job2, job3, 2);
}
criterion :ql{
        (job1, job2, 1/2);
        (job1, job3, 1/3);
        (job2, job3, 1/3);
}
criterion: iw{
        (job1, job2, 1/7);
        (job1, job3, 1/3);
        (job2, job3, 3);
}
criterion:nf{
        (job1, job2, 1/4);
        (job1, job3, 1/7);
        (job2, job3, 1);
}
```

Fonte: O autor (2017)

Na figura 44, pode-se perceber que a matriz montada para o critério NF foi alterada. Além disso, o programa constatou que a matriz não está mais inconsistente (figura 45). O resultado final também foi alterado e seus valores se aproximam mais dos resultados calculados por Winston, como pode ser observado na figura 46.

Figura 44 - Nova matriz para o critério NF

```
Criterion:  NF

JOB1    JOB2    JOB3
1.0   0.25   0.1429
4.0   1.0   1.0
7.0   1.0   1.0
```

Fonte: O autor (2017).

Figura 45 - Nova matriz normalizada de pesos, prioridade relativa e julgamentos para NF

```
Criterion:  NF
JOB1    JOB2    JOB3
0.0833   0.1111   0.0667
0.3333   0.4444   0.4667
0.5833   0.4444   0.4667


JOB1:   0.087

JOB2:   0.4148

JOB3:   0.4981

Lambda:   3.035
CI:   0.0175
CR:   0.0302
```

Fonte: O autor (2017).

Figura 46 - Resultados finais do teste 1 recalculados

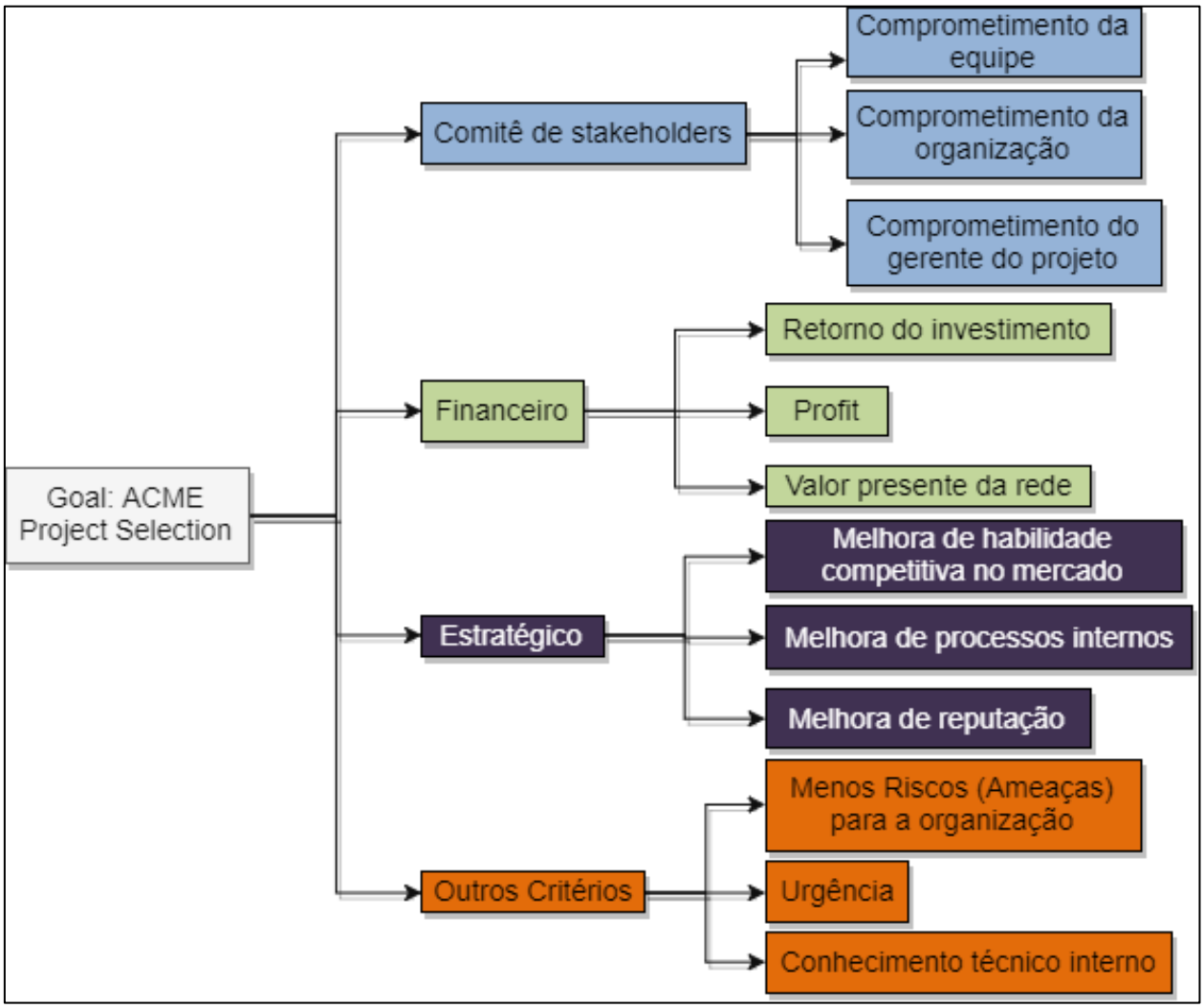
```
#################### Final Result ###################
##############################
0.5115384615384615 |  SAL
0.0985576923076923 |  QL
0.2432692307692308 |  IW
0.1466346153846154 |  NF
##############################
SAL     QL     IW     NF
0.5714|0.1593|0.0882|0.087|  JOB1
0.2857|0.2519|0.6687|0.4148|  JOB2
0.1429|0.5889|0.2431|0.4981|  JOB3
JOB1 :   0.3422
JOB2 :   0.3945
JOB3 :   0.2633
```

Fonte: O autor (2017).

5.1 Segundo teste

O segundo teste será baseado em um exemplo de Vargas (2010), no qual foi usado o *software Expert Choice* para a realização dos cálculos. Nesse exemplo, Vargas considerou uma empresa fictícia chamada ACME, a qual precisava definir uma prioridade de projetos à investir. A Figura 47 mostra a hierarquia dos critérios construída por Vargas.

Figura 47 - Hierarquia de critérios para a Seleção de projetos da ACME



Fonte: Vargas (2010)

A fim de simplificar a leitura pelo programa, fora criadas siglas para cada opção assim como para todos os critérios e subcritérios usados pelo programa *Odin*. As siglas para as alternativas estão descritas no quadro 4, e as siglas para os critérios estão descritas no quadro 5.

Quadro 4 - Projeto a serem escolhidos e siglas usadas no Programa Odin.

Projetos	Siglas
Mudança para um novo escritório	N.O (*New Office*)
Novo sistema ERP	ERP
Abertura de um escritório na China	CHINO (*Chinese Office*)
Desenvolvimento de um novo produto visando o mercado internacional	INTERN.P (*International Product*)
Terceirização da infraestrutura de TI	ITO (*IT Outsourcing*)
Nova campanha de marketing local	NLC (*New Local Marketing*)

Fonte: O Autor (2017)

Quadro 5 - Critérios para escolha de projetos e siglas aplicadas no *software Odin*

	Critérios	Siglas
Critérios(Nv.1)	Comitê de stakeholders	SHC
	Financeiro	FINAN
	Estratégico	STRA
	Outros critérios	OC
Subcritérios (Nv.2)	Comprometimento da equipe	TEAMC
	Comprometimento da organização	ORGC
	Comprometimento do gerente do projeto	PROJMC
	Retorno do investimento	ROI
	Profit	PROFIT
	Valor presente da rede	NPV
	Melhora de habilidade competitiva no mercado.	CIIM
	Melhora de processos internos	IP
	Melhora de reputação	REP
	Menos riscos (ameaças) para a organização	LRO
	Urgência	URGE
	Conhecimento técnico interno	ITK

Fonte: O Autor (2017)

Após a criação das siglas, iniciou-se a comparação da montagem das matrizes de comparação para os critérios principais.

Quadro 6 - Matriz de critérios principais (Nível 1) por Vargas

	SHC	**FINAN**	**STRA**	**OC**
SHC	1	1/5	1/9	1
FINAN	5	1	1	5
STRA	9	1	1	5
OC	1	1/5	1/5	1

Fonte: Vargas (2010)

Figura 48 - Matriz principal de critérios para o problema de Vargas montadas por *Odin*

```
Parent:  OBJECTIVE
SHC FINAN STRA OC

1.0|0.2|0.1111|1.0|
5.0|1.0|1.0|5.0|
9.0|1.0|1.0|5.0|
1.0|0.2|0.2|1.0|
```

Fonte: O autor

Quadro 7 - Matriz normalizada de pesos para os critérios principais (Nível 1) por Vargas

	SHC	**FINAN**	**STRA**	**OC**
SHC	0,063	0,83	0,048	0,083
FINAN	0,313	0,417	0,433	0,417
STRA	0,563	0,417	0,433	0,417
OC	0,063	0,083	0,087	0,083

Fonte: Vargas (2010)

Figura 49 - Matriz normalizada de pesos dos critérios para o problema de Vargas montadas por *Odin*

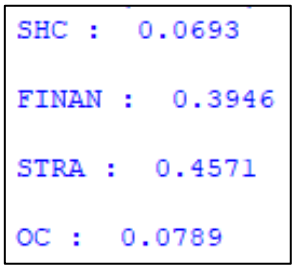
```
SHC FINAN STRA OC
0.0625|0.0833|0.0481|0.0833|
0.3125|0.4167|0.4327|0.4167|
0.5625|0.4167|0.4327|0.4167|
0.0625|0.0833|0.0865|0.0833|
SHC :  0.0693
```

Fonte: O Autor (2017)

Por ação da quantidade de critérios e opções, o número de matrizes geradas será grande, assim, para sintetizar a análise, apenas os resultados dos pesos de cada critério serão apresentados, bem como os resultados parciais para cada critério até que enfim os resultados finais sejam comparados.

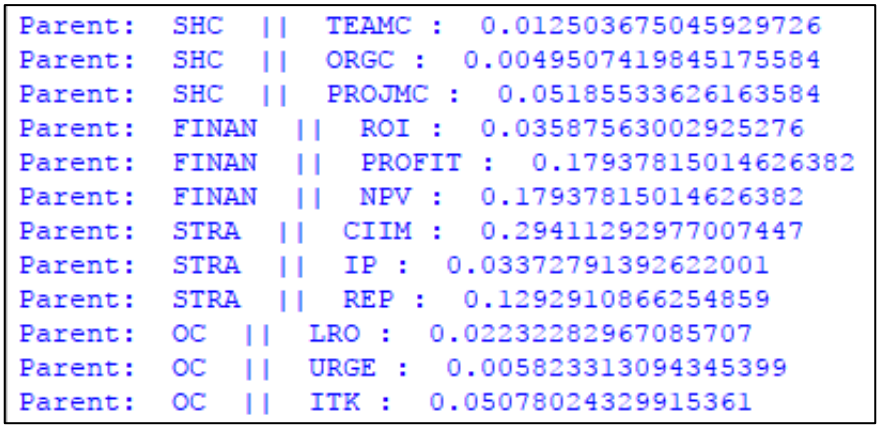
A partir dos pesos calculados pelo programa (Figura 50 e 51), gerou-se o quadro 8. Esse quadro foi comparado com a figura elaborada por Vargas através do *software Expert Choice* (Figura 52).

Figura 50 - Pesos dos critérios principais para o problema de Vargas calculados por *Odin*



Fonte: O Autor (2017)

Figura 51 - Pesos dos subcritérios para o problema de Vargas calculados por *Odin*

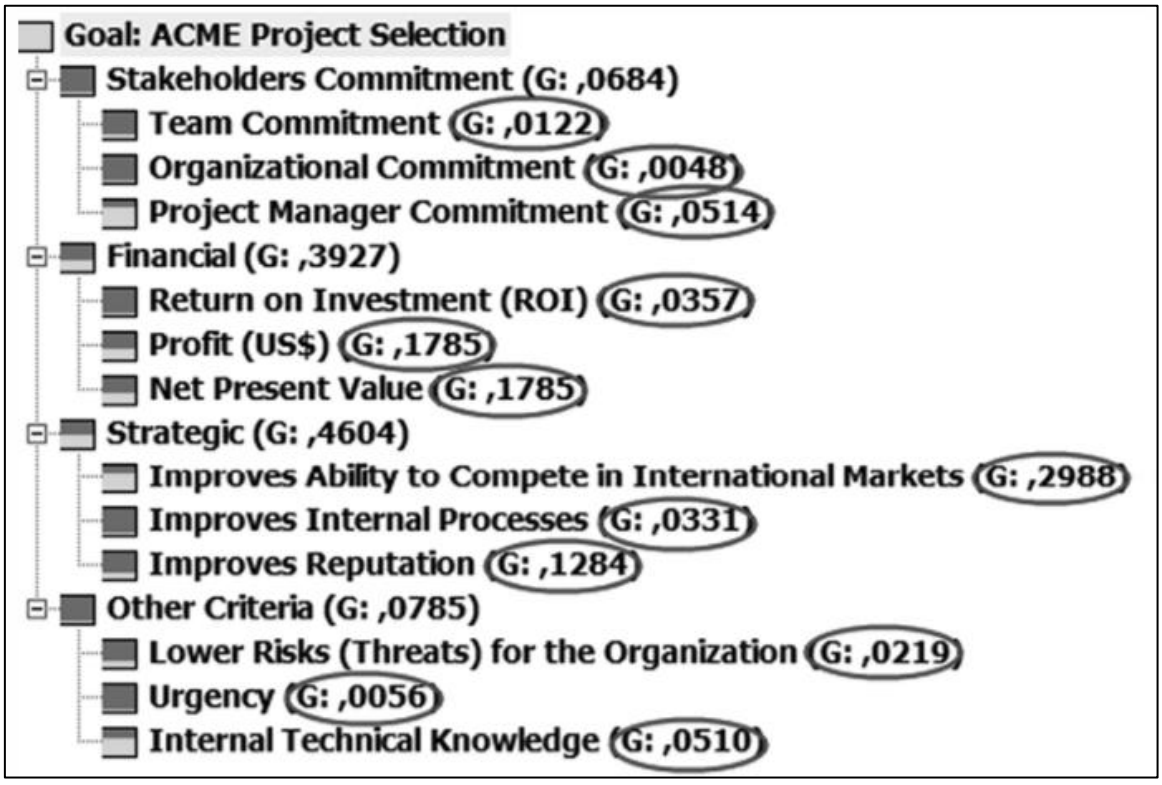


Fonte: O Autor (2017)

Quadro 8 - Pesos dos critérios e subcritérios para o problema de Vargas calculados por *Odin*

Critérios	Prioridades	Subcritérios	Prioridades
SHC	0.0693	TEAMC	0.01250368
		ORGC	0.00495074
		PROJMC	0.05185534
FINAN	0.3946	ROI	0.03587563
		PROFIT	0.17937815
		NPV	0.17937815
STRA	0.4571	CIIM	0.29411293
		IP	0.03372791
		REP	0.12929109
OC	0.0789	LRO	0.02232283
		URGE	0.00582331
		ITK	0.05078024

Figura 52 - Pesos dos critérios e subcritérios calculados por Vargas



Fonte: Vargas (2010)

Ao visualizar os dois resultados, é possível perceber que os valores são bem aproximados, porém não são iguais. Apesar da diferença nos valores, a ordem de importância dos critérios não é alterada, desta maneira o resultado final não será comprometido.

Tendo calculado todos os pesos para os critérios, os dados das alternativas serão agora fornecidos ao programa.

Após a entrada dos dados, as matrizes são normalizadas e em seguida é calculada a consistência. Como pode ser verificado nas figuras 53, 54, 55 e 56, as matrizes para os critérios CIIM, LRO, URGE e ITK estão inconsistentes, e necessitariam de uma revisão para serem rodadas novamente. Porém considerando que estes dados são fictícios e que não é o intuito deste trabalho corrigir outros trabalhos, a informação da inconsistência apenas foi citada para reafirmar que o *software* consegue detecta-las.

Figura 53 - Inconsistência na matriz de opções para o critério CIIM

```
Criterion:  CIIM
NO    ERP    CHINO    INTERN.P    ITO    NLC
0.0564   0.1184   0.0455   0.0577   0.2027   0.1667
0.0188   0.0395   0.0455   0.0449   0.0135   0.1
0.5075   0.3553   0.4091   0.4038   0.3649   0.3
0.3947   0.3553   0.4091   0.4038   0.3649   0.3
0.0113   0.1184   0.0455   0.0449   0.0405   0.1
0.0113   0.0132   0.0455   0.0449   0.0135   0.0333


NO:  0.1079

ERP:  0.0437

CHINO:  0.3901

INTERN.P:  0.3713

ITO:  0.0601

NLC:  0.0269

Lambda:  6.7396
CI:  0.1479
CR:  0.1193
```

Fonte: O Autor (2017)

Figura 54 - Inconsistência na matriz de opções para o critério LRO

```
Criterion:  LRO
NO    ERP    CHINO    INTERN.P    ITO    NLC
0.3477  0.3606  0.25  0.2064  0.2143  0.3987
0.0695  0.0721  0.1786  0.2064  0.1286  0.057
0.0497  0.0144  0.0357  0.0229  0.0143  0.0443
0.1159  0.024  0.1071  0.0688  0.2143  0.057
0.0695  0.024  0.1071  0.0138  0.0429  0.0443
0.3477  0.5048  0.3214  0.4817  0.3857  0.3987


NO:  0.2963

ERP:  0.1187

CHINO:  0.0302

INTERN.P:  0.0979

ITO:  0.0503

NLC:  0.4067

Lambda:  6.6745
CI:  0.1349
CR:  0.1088
```

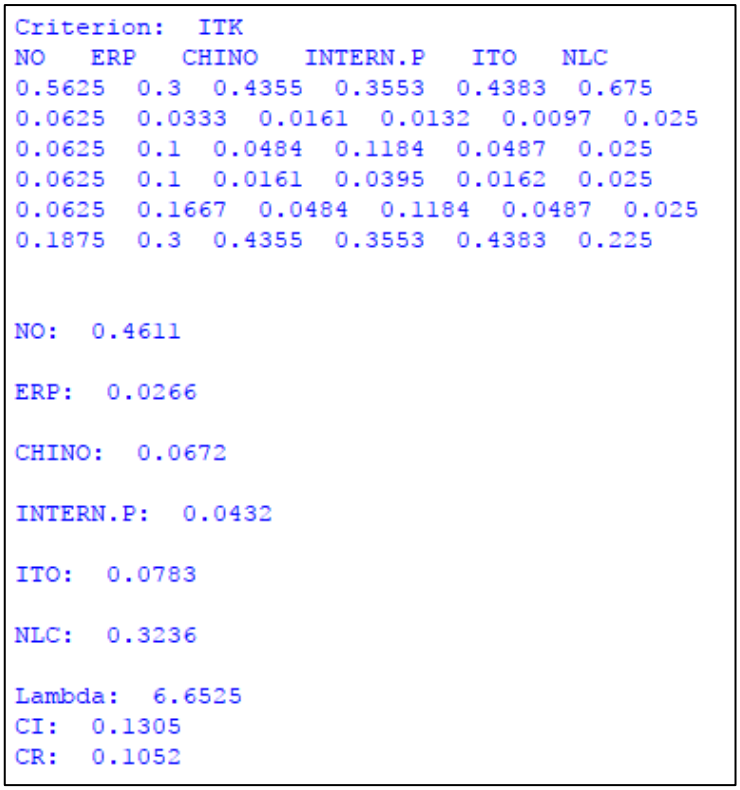
Fonte: O Autor (2017)

Figura 55 - Inconsistência na matriz de opções para o critério URGE

```
Criterion:  URGE
NO    ERP    CHINO    INTERN.P    ITO    NLC
0.0577  0.0185  0.0427  0.0763  0.1364  0.0517
0.1731  0.0556  0.0305  0.0593  0.1364  0.1552
0.2885  0.3889  0.2134  0.178  0.2273  0.3621
0.4038  0.5  0.6402  0.5339  0.3182  0.3621
0.0192  0.0185  0.0427  0.0763  0.0455  0.0172
0.0577  0.0185  0.0305  0.0763  0.1364  0.0517


NO:  0.0639

ERP:  0.1017

CHINO:  0.2763

INTERN.P:  0.4597

ITO:  0.0366

NLC:  0.0618

Lambda:  6.6671
CI:  0.1334
CR:  0.1076
```

Fonte: O Autor (2017)

Figura 56 - Inconsistência na matriz de opções para o critério ITK

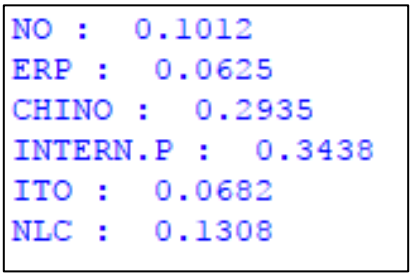


```
Criterion:  ITK
NO    ERP    CHINO    INTERN.P    ITO    NLC
0.5625   0.3   0.4355   0.3553   0.4383   0.675
0.0625   0.0333   0.0161   0.0132   0.0097   0.025
0.0625   0.1   0.0484   0.1184   0.0487   0.025
0.0625   0.1   0.0161   0.0395   0.0162   0.025
0.0625   0.1667   0.0484   0.1184   0.0487   0.025
0.1875   0.3   0.4355   0.3553   0.4383   0.225


NO:  0.4611

ERP:  0.0266

CHINO:  0.0672

INTERN.P:  0.0432

ITO:  0.0783

NLC:  0.3236

Lambda:  6.6525
CI:  0.1305
CR:  0.1052
```

Fonte: O Autor (2017)

Enfim, após o cálculo dos pesos, restou apresentar o resultado final calculado por Odin (Figura 57) e compará-lo com o valor encontrado por Vargas. Com o propósito de simplificar a análise, organizou-se os resultados em uma tabela, ordenados do mais apropriado ao menos apropriado (Quadro 9) e comparou-se com a figura 55, onde consta os resultados encontrados por Vargas.

Figura 57 - Resultado final para o problema de Vargas



```
NO :  0.1012
ERP :  0.0625
CHINO :  0.2935
INTERN.P :  0.3438
ITO :  0.0682
NLC :  0.1308
```

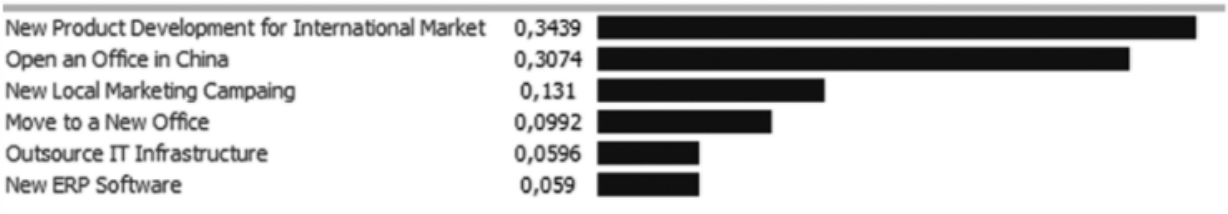
Fonte: O Autor (2017)

Quadro 9 - Resultado final ordenado para o problema de Vargas

INTERN.P	0.3438
CHINO	0.2935
NLC	0.1308
NO	0.1012
ITO	0.0682
ERP	0.0625

Fonte: O Autor (2017)

Figura 58 - Resultado final calculado pelo *Expert Choice.*

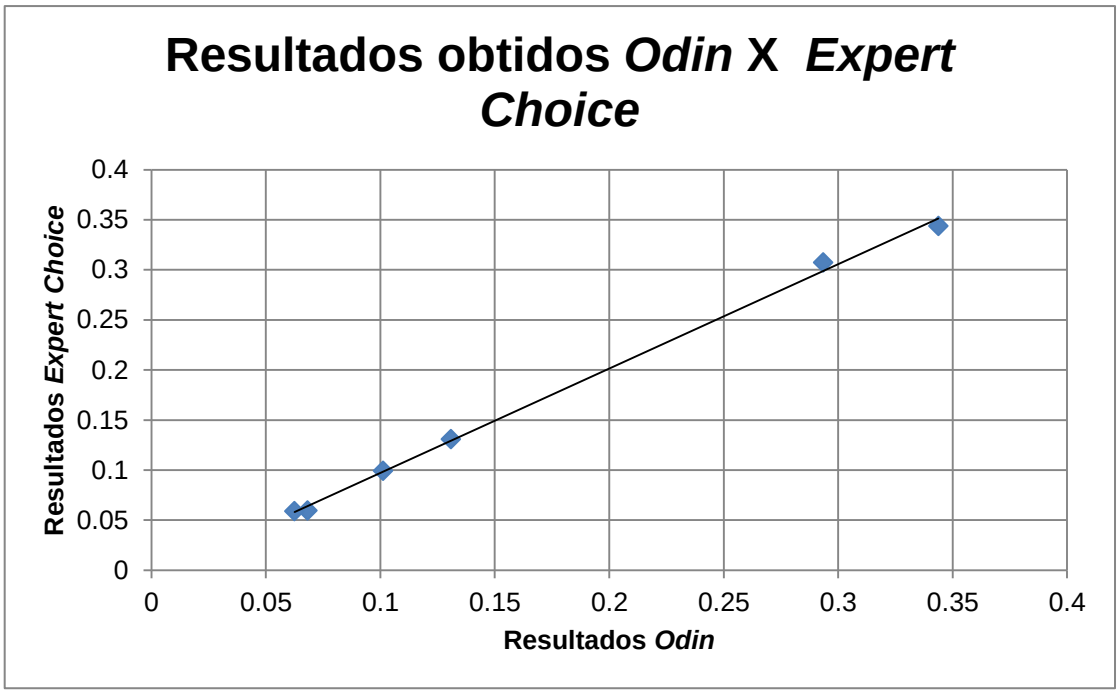


Fonte: Vargas (2010)

Apesar de alguns valores serem diferentes, a ordem de importância calculada por *Odin* é a mesma apresentada por Vargas em seu exemplo, o que comprova que o *software* pode ser usado para mais de um nível de critérios.

Para comprovar a proximidade dos valores calculados pelo programa *Expert Choice* e *Odin*, foi feita uma análise estatística, como mostra o gráfico 2 de dispersão.

Gráfico 2 - Gráfico de dispersão *Odin* x *Expert Choice*



Fonte: O autor (2017)

Além da construção do gráfico, foi feita uma regressão linear, onde considerou-se os resultados obtidos pelo *Expert Choice* os números de referência padrão, tendo em vista que ele é uma solução desenvolvida pelo criador do método AHP, além de ser um *software* já consolidado no mercado. Os quadros 10, 11 e 12 mostram os resultados computados utilizando a ferramenta *Microsoft Excel*.

Quadro 10 - Regressão Simples *Odin* X *Expert Choice*

Estatística de regressão	
R múltiplo	0.99905366
R-Quadrado	0.998108216
R-quadrado ajustado	0.99763527
Erro padrão	0.006154791
Observações	6

Fonte: O autor (2017)

Quadro 11 - Análise de variância parte 1

	gl	***SQ***	***MQ***	***F***	***F de significação***
Regressão	1	0.079945243	0.079945243	2110.406115	1.34291E-06
Resíduo	4	0.000151526	3.78814E-05		
Total	5	0.080096768			

Fonte: O autor (2017)

Quadro 12 - Análise de variância parte 2

	Coeficientes	*Erro padrão*	*Stat t*	*valor-P*
Interseção	-0.0070	0.0045	-1.5508	0.1959
Odin	1.0423	0.0227	45.9392	1.34E-06

Fonte: O autor (2017)

O erro padrão descrito no quadro 10 é menor que 0,05; o que indica que não há muita variação entre o *Expert Choice* e *Odin*. Esse mesmo argumento é comprovado pelo valor-p no quadro 12 que está próximo de zero. O R^2 foi quase 100%, ou seja, o modelo é capaz ser explicado pelos regressores.

Quanto ao teste f de significância no quadro 2, foi possível observar que há evidências que os resultados de Odin têm relação com os resultados do programa usado por Vargas (2010), em outras palavras, o modelo é adequado.

5.3 Explicação sobre os arredondamentos.

Foi possível notar que alguns valores encontrados pelo programa, são diferentes daqueles calculados pelos autores, isso acontece porque o *Odin* foi desenvolvido de uma maneira que os valores são arredondados para quatro casas decimais apenas no fim das operações que ocorrem internamente. Em outras palavras, a solução trabalha com 18 casas decimais até o fim dos cálculos e apenas arredonda os valores para facilitar a visualização pelo usuário.

Essa característica faz com que os números calculados por ele se diferencie ligeiramente do número calculado em outras soluções, entretanto os valores gerados por ele são tão precisos ou até mais precisos que valores obtidos por outros *softwares*.

Apesar da ligeira diferença nos cálculos, essa característica não interfere na tomada de decisão, assim como foi mostrado nos dois exemplos anteriores.

CAPÍTULO VI – CONCLUSÃO E SUGESTÕES

6.1 Conclusão

O objetivo do trabalho foi desenvolver uma ferramenta nova para a tomada de decisão em situações que evolvem multicritérios, baseada no método de Análise hierárquica do processo (AHP), de Saaty, por se tratar de uma poderosa ferramenta

que organiza critérios escolhidos pelos tomadores de decisão e, por meio da atribuição de pesos, calcula a prioridade para cada opção.

Por ter sido calculado em Python, uma linguagem interpretada de alto nível, o programa pode ser executado, sem que se faça qualquer tipo de portabilidade, em diferentes sistemas operacionais, pois único requisito para sua execução é que o computador tenha a máquina virtual do Python instalada no sistema.

Foram usadas duas situações propostas por autores diferentes para que os dados adquiridos por eles fossem rodados com a ferramenta desenvolvida neste trabalho. A primeira situação a qual utilizava apenas um nível de critérios na hierarquia mostrou que o programa consegue montar as matrizes automaticamente evitando o erro por parte do usuário.

Já a segunda atentou-se para o tamanho do problema, pois sua hierarquia apresentava dois níveis de critérios mais um total de seis opções.

O *software* consegue resolver, satisfatoriamente, o cálculo de AHP para mais de um nível de critérios e os escreve na tela. Além de calcular o resultado final, o usuário pode acompanhar etapas do cálculo como a normalização das matrizes e os resultados parciais para cada critério.

Além disso, é possível saber a consistência de cada matriz trabalhada no programa, permitindo que o usuário faça edições nos dados para trata-los novamente, evitando que a decisão seja comprometida.

Uma característica interessante é que o programa faz cálculos utilizando 18 casas decimais para todos os números, este fator aumenta a precisão dos cálculos, porém quando comparado com outras aplicações, os valores podem ser diferentes daqueles calculados por outros *softwares*, o que não afeta a ordem de importância das opções trabalhadas.

Para aumentar a precisão do cálculo, o método de normalização da matriz poderia fazer uso de cálculos matemático mais avançado, desse modo a tomada de decisão seria ainda mais confiável e precisa.

Odin consegue detectar quando o usuário tenta abrir um arquivo não existente, assim como pode detectar quando há erros no arquivo, porém o *software*

é incapaz de dizer ao usuário o lugar onde provavelmente o erro se encontra o que pode dificultar a correção do texto presente no arquivo e acarretar uma perda considerável, e desnecessária, de tempo e até frustração por parte do usuário. Portanto, sugere-se que seja criado um interpretador de arquivos que possa encontrar erros de digitação feitos pelo usuário.

Ainda que o *software* esteja fazendo o que foi planejado, ele sempre necessitará de suporte, para corrigir possíveis *bugs* não detectados na fase de teste, e de atualizações, pois um *software* não é um produto estável e sempre pode ser adicionado novas características ou funções ao mesmo.

Para concluir, alguns aspectos nos códigos estão redundantes. Em outras palavras, existem repetições desnecessárias de alguns loops e duplicação de uma função. A primeiro momento, tais aspectos não geram nenhum tipo de transtorno, como por exemplo, o aumento considerável do tempo de execução, porém com a implementação de tarefas (adicionais) ou com a entrada de uma massa de dados mais extensa, a importância de um código mais enxuto será certamente notada.

6.2 Trabalhos futuros

Para trabalhos futuros, sugere-se o desenvolvimento de uma Interface Gráfica, para que o uso do *software* se torne mais amigável, tendo em vista que a maioria das pessoas não usa e não deseja usar *prompts* de comando para realizar suas tarefas nos dias atuais.

Além da interface gráfica, seria útil o desenvolvimento de um sistema de exportação de arquivo que permitisse que o usuário salvasse o output em PDF, HTML e algum outro tipo de arquivo proprietário. Isso possibilitaria que o usuário consulte os dados de saída sem ter que trata os dados de entrada novamente ou abrir o programa sempre que outra consulta seja necessária.

Quanto ao código do *software*, é importante que redundâncias sejam retiradas e também seria interessante o estudo de mais algoritmos que possam ser implementados de forma a reduzir o tempo de execução, o qual já é bem rápido, para que a entrada de mais ou menos dados não interfira de forma significativa no tempo de execução do programa.

Ainda, um aspecto que pode ser desenvolvido como forma de facilitação da visualização dos resultados é a ordenação da alternativa mais importante para a menos importante, assim como feito no *software Expert Choice.*

Por fim, é interessante que o *software* seja usado para a tomada de decisão de problemas reais. Logo, sugere-se que o software seja usado para auxiliar a tomada de decisão em trabalhos acadêmicos relacionados a estudos de caso que envolvam tomada de decisão multicritério.

REFERÊNCIAS

ABREU, Edson. **Editor Vim:** Introdução e trabalhando com Vim. 2011. Disponível em: <https://www.vivaolinux.com.br/dica/Editor-Vim-Introducao-e-trabalhando-com-Vim>. Acesso em: 20 nov. 2017.

BASTOS, Henrique**. Diferenças entre linguagem compilada e linguagem interpretada**. 2010. Disponível em: <https://www.oficinadanet.com.br/artigo/programacao/diferencas_entre_linguagem_compilada_e_linguagem_interpretada>. Acesso em: 27 nov. 2017.

BHUSHAN, N.; RAI, K.The Analytic Hierarchy Process.In: BHUSHAN, N.; RAI, K.. **Strategic Decision Making:** Applying the Analytic Hierarchy Process. Londres: Springer-verlag, 2004. Cap. 2. p. 11-21. Disponível em: <http://www.springer.com/br/book/9781852337568>. Acesso em: 8 nov. 2017.

CORREIA, Sonia Maria Barros Barbosa. **Probabilidade e estatística** . 2. ed. Belo Horizonte: PUC Minas Virtual, 2003. 116 p. Disponível em: <http://estpoli.pbworks.com/f/livro_probabilidade_estatistica_2a_ed.pdf>. Acesso em: 12 dez. 2017.

FRIEDMAN, Frank L.; KOFFMAN, Elliot B. **Problem solving, abstraction and design using C++**. 6. ed. Massachussets: Pearson, 2011. 869 p.

GAIMAN, Neil. **Mitologia Nórdica**. 1. ed. Rio de Janeiro: Intrínseca, 2017. 288 p.

GRANT, Mike; PALMER, Zachary; SMITH, Scott. **Principles of programming languages**. 7. ed. San Francisco: Creative Commons, 2011. 187 p.

LUTUS, Paul. **The careware idea.** 2014. Disponível em: <https://arachnoid.com/careware/index.html>. Acesso em: 25 nov. 2017.

LUTZ, Mark. **Learning Python.** 5. ed. Sebastopol: O'reilly, 2013. 1648 p.

MOOLENAAR, Bram. **Vim documentation**. 2010. Disponível em: <http://vimdoc.sourceforge.net/htmldoc/uganda.html#license>. Acesso em: 26 nov. 2017.

MOOLENAAR, Bram. **Downloading vim**. 2010. Disponível em: <https://vim.sourceforge.io/download.php>. Acesso em: 22 nov. 2017.

MONTGOMERY, D. C.; RUNGER, G. C. **Estatística aplicada e probabilidade para Engenheiros**. 4. ed., Editora LTC, 2009.

MU, E.; PEREYRA-ROJAS, M.. **Practical Decision Making**: An Introduction to the Analytic Hierarchy Process (AHP) Using Super Decisions V2. Londres: Springer-verlag, 2017. 111 p. (*Springer Briefs in Operations Research*).Disponível em: <http://www.springer.com/br/book/9783319338606>. Acesso em: 2 set. 2017

PIERCE, Benjamin C. **Types and programming languages**. Massachussets: The Mit Press, 2002. 645 p.

PYSCIENCE-BRASIL. **Python: O que é? Por que usar?**. 12 Set 2008. Disponível em: <http://pyscience-brasil.wikidot.com/python:python-oq-e-pq>. Acessoem: 18 nov. 2017.

PORTAL ACTION. **Análise de variância**. Disponível em: <http://www.portalaction.com.br/analise-de-regressao/15-analise-de-variancia>. Acesso em: 09 dez. 2017.

ROBINSON, David. **The incredible growth of python**. 2017. Disponível em: <https://stackoverflow.blog/2017/09/06/incredible-growth-python/>. Acesso em: 27 nov. 2017.

SAATY, R. W. **The analytic hierachy process**—What it is and how it is used, *Mathematical modeling*, v.9, n. 3-5, p. 161-176. 1987

SAATY, T. L. **Decision making with the analitic hierarchy process**. *International journal of services sciences*, v. 1, n. 1, p. 83-98, 2008.

SAN CRISTÓBAL, J. R. (2012). **Contractor selection using multicriteria decision-making methods**. Journal of Construction Engineering and Management 138(6), 751-758

SILVA, Edna Lúcia da; MENEZES, EsteraMuszkat. **Metodologia de pesquisa e elaboração de dissertação.** 4. ed. Florianópolis: Ufsc, 2005. 138 p.

SOMMERVILLE, I. **Introduction to *software* engineering**. In: SOMMERVILLE, I. ***Software Engineering.*** 10. ed. Boston: Pearson, 2015. Cap. 1, p. 6.

The Open Source Definition (Annotated). 2017. Elaborado por *The Open Source Initiative*. Disponível em: <https://opensource.org/osd-annotated>. Acesso em: 10 nov. 2017.

TRAMARICO, Claudemir F. **Avaliação multicritério de prestadores de serviço logísticos.** Dissertação (Mestrado). Faculdade de Engenharia, Universidade Estadual Paulista Júlio de Mesquita Filho, Guaratinguetá, 2012, 55p.

VAKLAVIK, M. C. **Propota de um modelo de avaliação de prestadores de serviços logísticos utilizando o AHP**: O caso de uma indústria de motores. Escola de Administração, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2011.

VENNERS, Bill. **The Making of Python**. 13 Jan 2003. Disponível em: <http://www.artima.com/intv/pytho*VENNERS, Bill. **The Making of Python**. 13 Jan 2003.* Disponível em: <http://www.artima.com/intv/pythonP.html>. Acesso em: 2 mai. 2016.nP.html>. Acesso em: 17 nov. 2017.

VARGAS, Ricardo. **Using the Analytic Hierarchy Process (AHP) to Select and Prioritize Projects in a Portfolio.** 2010. Disponível em: <https://ricardo-vargas.com/articles/analytic-hierarchy-process/>. Acesso em: 10 nov. 2017.

WINSTON, Wayne L. **Operational Research:** Applications and Algorithms. 4. ed. Belmont:Brooks/cole, 2004.1440p

OLIVEIRA, William. **O que é linguagem de programação de alto/baixo nível?** Disponível em: <https://woliveiras.com.br/posts/o-que-e-linguagem-de-programacao-de-alto-nivel/>. Acesso em: 26 nov. 2017

Apêndice 1

O objetivo desde apêndice é fornecer a resolução completa de ambos os problemas apresentados no capítulo 5.

Problema 1 (Winston):

Arquivo de entrada para os critérios:

```
objective:Find the best job option;

criteria {
      (SAL, QL, 5);
      (SAL, IW, 2);
      (SAL, NF, 4);
      (QL, IW, 1/2);
      (QL, NF, 1/2);
      (IW, NF, 2);
}
```

Arquivo de entrada de alternativas:

```
criterion :sal{

      (job1, job2, 2);

      (job1, job3, 4);

(job2, job3, 2);
}

criterion :ql{

      (job1, job2, 1/2);

      (job1, job3, 1/3);

(job2, job3, 1/3);
}

criterion: iw{

      (job1, job2, 1/7);

      (job1, job3, 1/3);

(job2, job3, 3);

}

criterion:nf{

      (job1, job2, 1/4);

      (job1, job3, 1/7);

      (job2, job3, 1);}
```

Resolução:

```
Objective: FIND THE BEST JOB OPTION

-------------------------------------------------
Parent:  OBJECTIVE
SAL QL IW NF

1.0|5.0|2.0|4.0|
0.2|1.0|0.5|0.5|
0.5|2.0|1.0|2.0|
0.25|2.0|0.5|1.0|
########################################################
############### Solving Main Weights ###################


SAL QL IW NF
0.5128|0.5|0.5|0.5333|
0.1026|0.1|0.125|0.0667|
0.2564|0.2|0.25|0.2667|
0.1282|0.2|0.125|0.1333|
SAL :  0.5115

QL :  0.0986

IW :  0.2433

NF :  0.1466

Lambda:  4.0476
CI:  0.0159
CR:  0.0176
############################## END ################################
######################### Working on options ###########################

Enter the name of the file with the alternatives:
 ** Info: type 'end' if you don't want to continue

-> testes/alternatives.txt

Criterion:  SAL

JOB1   JOB2   JOB3
1.0  2.0  4.0
0.5  1.0  2.0
0.25  0.5  1.0

Criterion:  QL

JOB1   JOB2   JOB3
1.0  0.5  0.3333
2.0  1.0  0.3333
3.0  3.0  1.0

Criterion:  IW

JOB1   JOB2   JOB3
1.0  0.1429  0.3333
7.0  1.0  3.0
3.0  0.3333  1.0
```

Criterion: NF

JOB1 JOB2 JOB3
1.0 0.25 0.1429
4.0 1.0 1.0
7.0 1.0 1.0
###################### Solving Options ####################
Criterion: SAL
JOB1 JOB2 JOB3
0.5714 0.5714 0.5714
0.2857 0.2857 0.2857
0.1429 0.1429 0.1429

JOB1: 0.5714

JOB2: 0.2857

JOB3: 0.1429

Lambda: 3.0
CI: 0.0
CR: 0.0

Criterion: QL
JOB1 JOB2 JOB3
0.1667 0.1111 0.2
0.3333 0.2222 0.2
0.5 0.6667 0.6

JOB1: 0.1593

JOB2: 0.2519

JOB3: 0.5889

Lambda: 3.0539
CI: 0.027
CR: 0.0465

Criterion: IW
JOB1 JOB2 JOB3
0.0909 0.0968 0.0769
0.6364 0.6774 0.6923
0.2727 0.2258 0.2308

JOB1: 0.0882

JOB2: 0.6687

JOB3: 0.2431

Lambda: 3.007
CI: 0.0035
CR: 0.0061

Criterion: NF

```
JOB1   JOB2   JOB3
0.08330.1111  0.0667
0.33330.4444  0.4667
0.58330.4444  0.4667


JOB1:0.087

JOB2:0.4148

JOB3:0.4981

Lambda:3.035
CI:0.0175
CR:0.0302

###################### Final Result ####################
################################
0.5115384615384615 |SAL
0.0985576923076923 |QL
0.2432692307692308 |IW
0.1466346153846154 |NF
################################
SALQL    IW    NF
0.5714|0.1593|0.0882|0.087|  JOB1
0.2857|0.2519|0.6687|0.4148|  JOB2
0.1429|0.5889|0.2431|0.4981|  JOB3
JOB1 :  0.3422
JOB2 :  0.3945
JOB3 :  0.2633
```

Problema 2 (Vargas):

Arquivo de entrada para os critérios:

```
objective: find the best project;
criteria{
        (shc, finan, 1/5);
        (shc, stra, 1/9);
        (shc, oc, 1);
        (finan, stra, 1);
        (finan, oc, 5);
        (stra, oc, 5);
}
sub-criteria-1{
        parent:shc{
                (teamc, orgc, 3);
                (teamc, projmc, 1/5);
                (orgc, projmc, 1/9);
        }
        parent:finan{
                (roi, profit, 1/5);
                (roi, npv, 1/5);
                (profit, npv, 1);
        }
        parent:stra{
                (ciim, ip, 7);
                (ciim, rep, 3);
                (ip, rep, 1/5);
        }
```

```
        parent:oc{
                (lro, urge, 5);
                (lro, itk, 1/3);
                (urge, itk, 1/7);
        }
}
```

Arquivo de entrada de alternativas:

```
criterion:teamc{
        (no, erp, 5);
        (no, chino, 3);
        (no, intern.p, 1/3);
        (no, ito, 9);
        (no, nlc, 7);
        (erp, chino, 1/5);
        (erp, intern.p, 1/7);
        (erp, ito, 1);
        (erp, nlc, 1/3);
        (chino, intern.p, 1/3);
        (chino, ito, 7);
        (chino, nlc, 3);
        (intern.p, ito, 5);
        (intern.p, nlc, 5);
        (ito, nlc, 1/3);
}
criterion:orgc{
        (no, erp, 3);
        (no, chino, 1/9);
        (no, intern.p, 1/5);
        (no, ito, 5);
        (no, nlc, 3);
        (erp, chino, 1/9);
        (erp, intern.p, 1/7);
        (erp, ito, 1);
        (erp, nlc, 1/3);
        (chino, intern.p, 3);
        (chino, ito, 7);
        (chino, nlc, 7);
        (intern.p, ito, 9);
        (intern.p, nlc, 7);
        (ito, nlc, 1/3);
}
criterion:projmc{
        (no, erp, 7);
        (no, chino, 1/3);
        (no, intern.p, 1/3);
        (no, ito, 5);
        (no, nlc, 3);
        (erp, chino, 1/9);
        (erp, intern.p, 1/7);
        (erp, ito, 3);
        (erp, nlc, 1/3);
        (chino, intern.p, 1);
        (chino, ito, 7);
        (chino, nlc, 7);
        (intern.p, ito, 7);
        (intern.p, nlc, 9);
        (ito, nlc, 1/5);
}
```

```
criterion:roi{
        (no, erp, 1/3);
        (no, chino, 1/7);
        (no, intern.p, 1/9);
        (no, ito, 1/3);
        (no, nlc, 1/3);
        (erp, chino, 1/9);
        (erp, intern.p, 1/9);
        (erp, ito, 1/3);
        (erp, nlc, 1/3);
        (chino, intern.p, 1/3);
        (chino, ito, 7);
        (chino, nlc, 5);
        (intern.p, ito, 7);
        (intern.p, nlc, 5);
        (ito, nlc, 1/3);
}
criterion:profit{
        (no, erp, 1);
        (no, chino, 1/7);
        (no, intern.p, 1/9);
        (no, ito, 1/5);
        (no, nlc, 1/3);
        (erp, chino, 1/7);
        (erp, intern.p, 1/9);
        (erp, ito, 1/3);
        (erp, nlc, 1/5);
        (chino, intern.p, 1/3);
        (chino, ito, 7);
        (chino, nlc, 5);
        (intern.p, ito, 9);
        (intern.p, nlc, 5);
        (ito, nlc, 1/3);
}
criterion:npv{
        (no, erp, 1/3);
        (no, chino, 1/5);
        (no, intern.p, 1/7);
        (no, ito, 1/3);
        (no, nlc, 1/3);
        (erp, chino, 1/5);
        (erp, intern.p, 1/7);
        (erp, ito, 1);
        (erp, nlc, 1/3);
        (chino, intern.p, 1/3);
        (chino, ito, 5);
        (chino, nlc, 3);
        (intern.p, ito, 5);
        (intern.p, nlc, 7);
        (ito, nlc, 1/3);
}
criterion:ciim{
        (no, erp, 3);
        (no, chino, 1/9);
        (no, intern.p, 1/7);
        (no, ito, 5);
        (no, nlc, 5);
        (erp, chino, 1/9);
        (erp, intern.p, 1/9);
        (erp, ito, 1/3);
```

```
        (erp, nlc, 3);
        (chino, intern.p, 1);
        (chino, ito, 9);
        (chino, nlc, 9);
        (intern.p, ito, 9);
        (intern.p, nlc, 9);
        (ito, nlc, 3);
}
criterion:ip{
        (no, erp, 1/5);
        (no, chino, 3);
        (no, intern.p, 5);
        (no, ito, 1);
        (no, nlc, 7);
        (erp, chino, 7);
        (erp, intern.p, 7);
        (erp, ito, 1);
        (erp, nlc, 7);
        (chino, intern.p, 1);
        (chino, ito, 1/7);
        (chino, nlc, 1);
        (intern.p, ito, 1/7);
        (intern.p, nlc, 1/3);
        (ito, nlc, 7);
}
criterion:rep{
        (no, erp, 1/3);
        (no, chino, 1/7);
        (no, intern.p, 1/5);
        (no, ito, 3);
        (no, nlc, 1/7);
        (erp, chino, 1/9);
        (erp, intern.p, 1/5);
        (erp, ito, 5);
        (erp, nlc, 1/7);
        (chino, intern.p, 3);
        (chino, ito, 7);
        (chino, nlc, 1);
        (intern.p, ito, 7);
        (intern.p, nlc, 1/3);
        (ito, nlc, 1/9);
}
criterion:lro{
        (no, erp, 5);
        (no, chino, 7);
        (no, intern.p, 3);
        (no, ito, 5);
        (no, nlc, 1);
        (erp, chino, 5);
        (erp, intern.p, 3);
        (erp, ito, 3);
        (erp, nlc, 1/7);
        (chino, intern.p, 1/3);
        (chino, ito, 1/3);
        (chino, nlc, 1/9);
        (intern.p, ito, 5);
        (intern.p, nlc, 1/7);
        (ito, nlc, 1/9);
}
criterion:urge{
```

```
                (no, erp, 1/3);
                (no, chino, 1/5);
                (no, intern.p, 1/7);
                (no, ito, 3);
                (no, nlc, 1);
                (erp, chino, 1/7);
                (erp, intern.p, 1/9);
                (erp, ito, 3);
                (erp, nlc, 3);
                (chino, intern.p, 1/3);
                (chino, ito, 5);
                (chino, nlc, 7);
                (intern.p, ito, 7);
                (intern.p, nlc, 7);
                (ito, nlc, 1/3);
        }
        criterion:itk{
                (no, erp, 9);
                (no, chino, 9);
                (no, intern.p, 9);
                (no, ito, 9);
                (no, nlc, 3);
                (erp, chino, 1/3);
                (erp, intern.p, 1/3);
                (erp, ito, 1/5);
                (erp, nlc, 1/9);
                (chino, intern.p, 3);
                (chino, ito, 1);
                (chino, nlc, 1/9);
                (intern.p, ito, 1/3);
                (intern.p, nlc, 1/9);
                (ito, nlc, 1/9);
        }
```

Resolução:

```
Objective:  FIND THE BEST PROJECT

-----------------------------------------------------
Parent:  OBJECTIVE
SHC FINAN STRA OC

1.0|0.2|0.1111|1.0|
5.0|1.0|1.0|5.0|
9.0|1.0|1.0|5.0|
1.0|0.2|0.2|1.0|
#########################################################
Parent: SHC

TEAMC  ORGC  PROJMC

1.0|3.0|0.2|
0.3333|1.0|0.1111|
5.0|9.0|1.0|
Parent: FINAN

ROI  PROFIT  NPV
```

1.0|0.2|0.2|
5.0|1.0|1.0|
5.0|1.0|1.0|
Parent: STRA

CIIM IP REP

1.0|7.0|3.0|
0.1429|1.0|0.2|
0.3333|5.0|1.0|
Parent: OC

LRO URGE ITK

1.0|5.0|0.3333|
0.2|1.0|0.1429|
3.0|7.0|1.0|
##
############### Solving Main Weights ###################

SHC FINAN STRA OC
0.0625|0.0833|0.0481|0.0833|
0.3125|0.4167|0.4327|0.4167|
0.5625|0.4167|0.4327|0.4167|
0.0625|0.0833|0.0865|0.0833|
SHC : 0.0693

FINAN : 0.3946

STRA : 0.4571

OC : 0.0789

Lambda: 4.0436
CI: 0.0145
CR: 0.0162
############################### END ################################
############### Solving sub-criteria-1 ###################
Parent: SHC
TEAMC ORGC PROJMC
0.1579|0.2308|0.1525|
0.0526|0.0769|0.0847|
0.7895|0.6923|0.7627|

Lambda: 3.0292
CI: 0.0146
CR: 0.0252

Parent: FINAN
ROI PROFIT NPV
0.0909|0.0909|0.0909|
0.4545|0.4545|0.4545|

0.4545|0.4545|0.4545|

Lambda: 3.0
CI: 0.0
CR: 0.0

Parent: STRA
CIIM IP REP
0.6774|0.5385|0.7143|
0.0968|0.0769|0.0476|
0.2258|0.3846|0.2381|

Lambda: 3.0656
CI: 0.0328
CR: 0.0566

Parent: OC
LRO URGE ITK
0.2381|0.3846|0.2258|
0.0476|0.0769|0.0968|
0.7143|0.5385|0.6774|

Lambda: 3.0656
CI: 0.0328
CR: 0.0566

Parent: SHC || TEAMC : 0.012503675045929726
Parent: SHC || ORGC : 0.0049507419845175584
Parent: SHC || PROJMC : 0.05185533626163584
Parent: FINAN || ROI : 0.03587563002925276
Parent: FINAN || PROFIT : 0.17937815014626382
Parent: FINAN || NPV : 0.17937815014626382
Parent: STRA || CIIM : 0.29411292977007447
Parent: STRA || IP : 0.03372791392622001
Parent: STRA || REP : 0.1292910866254859
Parent: OC || LRO : 0.02232282967085707
Parent: OC || URGE : 0.005823313094345399
Parent: OC || ITK : 0.05078024329915361
######################### Working on options ############################

Enter the name of the file with the alternatives:
 ** Info: type 'end' if you don't want to continue

-> testes/Vargas_teste_options.txt

Criterion: TEAMC

NOERP CHINO INTERN.P ITO NLC
1.05.0 3.0 0.3333 9.0 7.0
0.21.0 0.2 0.1429 1.0 0.3333
0.33335.0 1.0 0.3333 7.0 3.0
3.07.0 3.0 1.0 5.0 5.0
0.11111.0 0.1429 0.2 1.0 0.3333

0.14293.0 0.3333 0.2 3.0 1.0

Criterion:ORGC

NOERP CHINO INTERN.P ITO NLC
1.03.0 0.1111 0.2 5.0 3.0
0.33331.0 0.1111 0.1429 1.0 0.3333
9.09.0 1.0 3.0 7.0 7.0
5.07.0 0.3333 1.0 9.0 7.0
0.21.0 0.1429 0.1111 1.0 0.3333
0.33333.0 0.1429 0.1429 3.0 1.0

Criterion:PROJMC

NOERP CHINO INTERN.P ITO NLC
1.07.0 0.3333 0.3333 5.0 3.0
0.14291.0 0.1111 0.1429 3.0 0.3333
3.09.0 1.0 1.0 7.0 7.0
3.07.0 1.0 1.0 7.0 9.0
0.20.3333 0.1429 0.1429 1.0 0.2
0.33333.0 0.1429 0.1111 5.0 1.0

Criterion:ROI

NOERP CHINO INTERN.P ITO NLC
1.0 0.3333 0.1429 0.1111 0.3333 0.3333
3.0 1.0 0.1111 0.1111 0.3333 0.3333
7.0 9.0 1.0 0.3333 7.0 5.0
9.0 9.0 3.0 1.0 7.0 5.0
3.0 3.0 0.1429 0.1429 1.0 0.3333
3.0 3.0 0.2 0.2 3.0 1.0

Criterion: PROFIT

NO ERP CHINO INTERN.P ITO NLC
1.0 1.0 0.1429 0.1111 0.2 0.3333
1.0 1.0 0.1429 0.1111 0.3333 0.2
7.0 7.0 1.0 0.3333 7.0 5.0
9.0 9.0 3.0 1.0 9.0 5.0
5.0 3.0 0.1429 0.1111 1.0 0.3333
3.0 5.0 0.2 0.2 3.0 1.0

Criterion: NPV

NOERP CHINO INTERN.P ITO NLC
1.00.3333 0.2 0.1429 0.3333 0.3333
3.01.0 0.2 0.1429 1.0 0.3333
5.05.0 1.0 0.3333 5.0 3.0
7.07.0 3.0 1.0 5.0 7.0
3.01.0 0.2 0.2 1.0 0.3333
3.03.0 0.3333 0.1429 3.0 1.0

Criterion:CIIM

NOERP CHINO INTERN.P ITO NLC
1.03.0 0.1111 0.1429 5.0 5.0
0.33331.0 0.1111 0.1111 0.3333 3.0
9.09.0 1.0 1.0 9.0 9.0
7.09.0 1.0 1.0 9.0 9.0
0.23.0 0.1111 0.1111 1.0 3.0
0.20.3333 0.1111 0.1111 0.3333 1.0

Criterion:IP

NOERP CHINO INTERN.P ITO NLC
1.00.2 3.0 5.0 1.0 7.0
5.01.0 7.0 7.0 1.0 7.0
0.33330.1429 1.0 1.0 0.1429 1.0
0.20.1429 1.0 1.0 0.1429 0.3333
1.01.0 7.0 7.0 1.0 7.0
0.14290.1429 1.0 3.0 0.1429 1.0

Criterion:REP

NOERP CHINO INTERN.P ITO NLC
1.00.3333 0.1429 0.2 3.0 0.1429
3.01.0 0.1111 0.2 5.0 0.1429
7.09.0 1.0 3.0 7.0 1.0
5.05.0 0.3333 1.0 7.0 0.3333
0.33330.2 0.1429 0.1429 1.0 0.1111
7.07.0 1.0 3.0 9.0 1.0

Criterion:LRO

NOERP CHINO INTERN.P ITO NLC
1.0 5.0 7.0 3.0 5.0 1.0
0.2 1.0 5.0 3.0 3.0 0.1429
0.1429 0.2 1.0 0.3333 0.3333 0.1111
0.3333 0.3333 3.0 1.0 5.0 0.1429
0.2 0.3333 3.0 0.2 1.0 0.1111
1.0 7.0 9.0 7.0 9.0 1.0

Criterion: URGE

NO ERP CHINO INTERN.P ITO NLC
1.0 0.3333 0.2 0.1429 3.0 1.0
3.0 1.0 0.1429 0.1111 3.0 3.0
5.0 7.0 1.0 0.3333 5.0 7.0
7.0 9.0 3.0 1.0 7.0 7.0
0.3333 0.3333 0.2 0.1429 1.0 0.3333
1.0 0.3333 0.1429 0.1429 3.0 1.0

Criterion: ITK

NO ERP CHINO INTERN.P ITO NLC
1.0 9.0 9.0 9.0 9.0 3.0

0.1111 1.0 0.3333 0.3333 0.2 0.1111
0.1111 3.0 1.0 3.0 1.0 0.1111
0.1111 3.0 0.3333 1.0 0.3333 0.1111
0.1111 5.0 1.0 3.0 1.0 0.1111
0.3333 9.0 9.0 9.0 9.0 1.0
###################### Solving Options ####################
Criterion:TEAMC
NOERP CHINO INTERN.P ITO NLC
0.20890.2273 0.3908 0.1509 0.3462 0.42
0.04180.0455 0.0261 0.0647 0.0385 0.02
0.06960.2273 0.1303 0.1509 0.2692 0.18
0.62670.3182 0.3908 0.4526 0.1923 0.3
0.02320.0455 0.0186 0.0905 0.0385 0.02
0.02980.1364 0.0434 0.0905 0.1154 0.06

NO:0.2907

ERP:0.0394

CHINO:0.1712

INTERN.P: 0.3801

ITO:0.0394

NLC:0.0793

Lambda:6.5028
CI:0.1006
CR:0.0811

Criterion:ORGC
NOERP CHINO INTERN.P ITO NLC
0.0630.125 0.0603 0.0435 0.1923 0.1607
0.0210.0417 0.0603 0.0311 0.0385 0.0179
0.56720.375 0.5431 0.6526 0.2692 0.375
0.31510.2917 0.181 0.2175 0.3462 0.375
0.01260.0417 0.0776 0.0242 0.0385 0.0179
0.0210.125 0.0776 0.0311 0.1154 0.0536

NO:0.1075

ERP:0.0351

CHINO:0.4637

INTERN.P: 0.2878

ITO:0.0354

NLC:0.0706

Lambda:6.5542
CI:0.1108
CR:0.0894

Criterion:PROJMC
NOERP CHINO INTERN.P ITO NLC
0.13030.2561 0.1221 0.1221 0.1786 0.1461
0.01860.0366 0.0407 0.0523 0.1071 0.0162
0.39080.3293 0.3663 0.3663 0.25 0.3409
0.39080.2561 0.3663 0.3663 0.25 0.4383
0.02610.0122 0.0523 0.0523 0.0357 0.0097
0.04340.1098 0.0523 0.0407 0.1786 0.0487

NO:0.1592

ERP:0.0453

CHINO:0.3406

INTERN.P: 0.3446

ITO:0.0314

NLC:0.0789

Lambda:6.5759
CI:0.1152
CR:0.0929

Criterion:ROI
NOERP CHINO INTERN.P ITO NLC
0.03850.0132 0.0311 0.0585 0.0179 0.0278
0.11540.0395 0.0242 0.0585 0.0179 0.0278
0.26920.3553 0.2175 0.1756 0.375 0.4167
0.34620.3553 0.6526 0.5268 0.375 0.4167
0.11540.1184 0.0311 0.0753 0.0536 0.0278
0.11540.1184 0.0435 0.1054 0.1607 0.0833

NO:0.0311

ERP:0.0472

CHINO:0.3015

INTERN.P: 0.4454

ITO:0.0702

NLC:0.1045

Lambda:6.5424
CI:0.1085
CR:0.0875

Criterion:PROFIT
NOERP CHINO INTERN.P ITO NLC
0.03850.0385 0.0309 0.0595 0.0097 0.0281
0.03850.0385 0.0309 0.0595 0.0162 0.0169
0.26920.2692 0.216 0.1786 0.3409 0.4213
0.34620.3462 0.6481 0.5357 0.4383 0.4213
0.19230.1154 0.0309 0.0595 0.0487 0.0281
0.11540.1923 0.0432 0.1071 0.1461 0.0843

NO:0.0342

ERP:0.0334

CHINO:0.2826

INTERN.P: 0.456

ITO:0.0791

NLC:0.1147

Lambda:6.5716
CI:0.1143
CR:0.0922

Criterion:NPV
NOERP CHINO INTERN.P ITO NLC
0.04550.0192 0.0405 0.0728 0.0217 0.0278
0.13640.0577 0.0405 0.0728 0.0652 0.0278
0.22730.2885 0.2027 0.1699 0.3261 0.25
0.31820.4038 0.6081 0.5097 0.3261 0.5833
0.13640.0577 0.0405 0.1019 0.0652 0.0278
0.13640.1731 0.0676 0.0728 0.1957 0.0833

NO:0.0379

ERP:0.0667

CHINO:0.2441

INTERN.P: 0.4582

ITO:0.0716

NLC:0.1215

Lambda:6.4278

CI:0.0856
CR:0.069

Criterion:CIIM
NOERP CHINO INTERN.P ITO NLC
0.05640.1184 0.0455 0.0577 0.2027 0.1667
0.01880.0395 0.0455 0.0449 0.0135 0.1
0.50750.3553 0.4091 0.4038 0.3649 0.3
0.39470.3553 0.4091 0.4038 0.3649 0.3
0.01130.1184 0.0455 0.0449 0.0405 0.1
0.01130.0132 0.0455 0.0449 0.0135 0.0333

NO:0.1079

ERP:0.0437

CHINO:0.3901

INTERN.P: 0.3713

ITO:0.0601

NLC:0.0269

Lambda:6.7396
CI:0.1479
CR:0.1193

Criterion:IP
NOERP CHINO INTERN.P ITO NLC
0.13030.0761 0.15 0.2083 0.2917 0.3
0.65140.3804 0.35 0.2917 0.2917 0.3
0.04340.0543 0.05 0.0417 0.0417 0.0429
0.02610.0543 0.05 0.0417 0.0417 0.0143
0.13030.3804 0.35 0.2917 0.2917 0.3
0.01860.0543 0.05 0.125 0.0417 0.0429

NO:0.1927

ERP:0.3775

CHINO:0.0457

INTERN.P: 0.038

ITO:0.2907

NLC:0.0554

Lambda:6.4466
CI:0.0893

CR:0.072

Criterion:REP
NOERP CHINO INTERN.P ITO NLC
0.04290.0148 0.0523 0.0265 0.0938 0.0523
0.12860.0444 0.0407 0.0265 0.1562 0.0523
0.30.3994 0.3663 0.3977 0.2188 0.3663
0.21430.2219 0.1221 0.1326 0.2188 0.1221
0.01430.0089 0.0523 0.0189 0.0312 0.0407
0.30.3107 0.3663 0.3977 0.2812 0.3663

NO:0.0471

ERP:0.0748

CHINO:0.3414

INTERN.P: 0.1719

ITO:0.0277

NLC:0.337

Lambda:6.5577
CI:0.1115
CR:0.0899

Criterion:LRO
NOERP CHINO INTERN.P ITO NLC
0.34770.3606 0.25 0.2064 0.2143 0.3987
0.06950.0721 0.1786 0.2064 0.1286 0.057
0.04970.0144 0.0357 0.0229 0.0143 0.0443
0.11590.024 0.1071 0.0688 0.2143 0.057
0.06950.024 0.1071 0.0138 0.0429 0.0443
0.34770.5048 0.3214 0.4817 0.3857 0.3987

NO:0.2963

ERP:0.1187

CHINO:0.0302

INTERN.P: 0.0979

ITO:0.0503

NLC:0.4067

Lambda:6.6745
CI:0.1349
CR:0.1088

Criterion:URGE
NOERP CHINO INTERN.P ITO NLC
0.05770.0185 0.0427 0.0763 0.1364 0.0517
0.17310.0556 0.0305 0.0593 0.1364 0.1552
0.28850.3889 0.2134 0.178 0.2273 0.3621
0.40380.5 0.6402 0.5339 0.3182 0.3621
0.01920.0185 0.0427 0.0763 0.0455 0.0172
0.05770.0185 0.0305 0.0763 0.1364 0.0517

NO:0.0639

ERP:0.1017

CHINO:0.2763

INTERN.P: 0.4597

ITO:0.0366

NLC:0.0618

Lambda:6.6671
CI:0.1334
CR:0.1076

Criterion:ITK
NOERP CHINO INTERN.P ITO NLC
0.56250.3 0.4355 0.3553 0.4383 0.675
0.06250.0333 0.0161 0.0132 0.0097 0.025
0.06250.1 0.0484 0.1184 0.0487 0.025
0.06250.1 0.0161 0.0395 0.0162 0.025
0.06250.1667 0.0484 0.1184 0.0487 0.025
0.18750.3 0.4355 0.3553 0.4383 0.225

NO:0.4611

ERP:0.0266

CHINO:0.0672

INTERN.P: 0.0432

ITO:0.0783

NLC:0.3236

Lambda:6.6525
CI:0.1305
CR:0.1052

```
###################### Final Result ###################
###############################
0.012503675045929726 |TEAMC
0.0049507419845175584 |ORGC
0.05185533626163584 |PROJMC
0.03587563002925276 |ROI
0.17937815014626382 |PROFIT
0.17937815014626382 |NPV
0.29411292977007447 |CIIM
0.03372791392622001 |IP
0.1292910866254859 |REP
0.02232282967085707 |LRO
0.005823313094345399 |URGE
0.05078024329915361 |ITK
###############################
TEAMC  ORGC  PROJMC  ROI  PROFIT  NPV  CIIM  IP  REP  LRO  URGE  ITK
0.2907|0.1075|0.1592|0.0311|0.0342|0.0379|0.1079|0.1927|0.0471|0.2963|0.0639|0.4611|  NO
0.0394|0.0351|0.0453|0.0472|0.0334|0.0667|0.0437|0.3775|0.0748|0.1187|0.1017|0.0266|  ERP
0.1712|0.4637|0.3406|0.3015|0.2826|0.2441|0.3901|0.0457|0.3414|0.0302|0.2763|0.0672|
CHINO
0.3801|0.2878|0.3446|0.4454|0.456|0.4582|0.3713|0.038|0.1719|0.0979|0.4597|0.0432|
INTERN.P
0.0394|0.0354|0.0314|0.0702|0.0791|0.0716|0.0601|0.2907|0.0277|0.0503|0.0366|0.0783|  ITO
0.0793|0.0706|0.0789|0.1045|0.1147|0.1215|0.0269|0.0554|0.337|0.4067|0.0618|0.3236|  NLC
NO :  0.1012
ERP :  0.0625
CHINO :  0.2935
INTERN.P :  0.3438
ITO :  0.0682
NLC :  0.1308
```

Centro de Ciências Naturais e Tecnologia
Curso de Graduação em Engenharia de Produção
Tv. Enéas Pinheiro, n° 2626 - Marco
CEP: 66095-100 Belém - PA
www.uepa.br