UNIVERSIDADE FEDERAL DA BAHIA

INSTITUTO DE MATEMÁTICA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Antonio de Almeida Souza Neto

Antares DSM: Visualização e Otimização de Dependências em Design Structures Matrices

Salvador

2008

Antonio de Almeida Souza Neto

Antares DSM: Visualização e Otimização de Dependências em Design Structures Matrices

Monografia apresentada ao Curso de graduação em Ciência da Computação, Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Profª . Christina von Flach Garcia Chavez

Salvador

2008

RESUMO

A compreensão e visualização de dependências em sistemas grandes e complexos, com muitos elementos e relacionamentos de diversos tipos entre eles, podem ser tarefas difíceis, compromentendo sua evolução e identificação de problemas. Design Structure Matrix (DSM) é um tipo de matriz de dependências que promove uma visualização compacta e intuitiva do sistema como um todo, além de permitir o uso de algoritmos existentes para otimização das dependências representadas com objetivos diversos. Este trabalho explora o uso de DSMs e propõe o algoritmo **Quick Triangular Matrix**, um algoritmo guloso para particionamento de DSMs e identificação de ciclos indesejáveis. Além disso, apresenta a ferramenta Antares DSM para visualização e otimização de DSMs com o algoritmo proposto. Um estudo de caso simples é usado para ilustrar os benefícios e limitações do algoritmo e ferramenta propostos.

Palavras-chave: Design Structure Matrix, DSM, matriz de dependências, grafos, Antares, Quick Triangular Matrix, particionamento, identificação de ciclos.

ABSTRACT

Visualizing and understanding dependencies in complex, large-scale systems are challenging tasks, that if not managed, may hinder evolution and system validation. The Design Structure Matrix (DSM) promotes compact and intuitive visualization of a complex system as a whole, and supports the use of existings algorithms to optimize dependencies among elements with different goals. This work exploits the use of DSMs and proposes the **Quick Triangular Matrix**, a greedy partition algorithm to identify undesirable cycles. The Antares DSM tool is introduced, to support the visualization of DSMs and optimization of system dependencies by using the proposed algorithm. A simple case study is presented to illustrate the contributions and limitations of the proposed algorithm and tool.

Keywords: Design Structure Matrix, DSM, graphs, Antares, Quick Triangular Matrix, partitioning, identification of cycles.

LISTA DE FIGURAS

1	Exemplo de grafo	11
2	Configurações de componentes em DSM	12
3	Matriz exemplo antes de aplicar o Quick Triangular Matrix	16
4	Matriz exemplo com as listas ordenadas de dependentes e fornecedores	17
5	Matriz exemplo após o término da parte I	17
6	Um exemplo sem troca e outro com troca de elementos	18
7	Matriz exemplo após aplicação do algoritmo	19
8	Matriz exemplo exibindo elementos pertencentes a ciclos	19
9	Antares DSM - Grafo e matriz de dependências	26
10	Antares DSM - Matriz particionada pelo algoritmo Quick Triangular Matrix	27
11	Antares DSM - Identificação e visualização de ciclos na DSM	28
12	Antares DSM - Criação de casos de teste	31
13	Tempo de execução de cada algoritmo	35
14	Porcentagem de resultados ótimos em cada algoritmo	35
15	Modelagem UML inicial do sistema de farmácia	38
16	Grafo do sistema de farmácia organizado como no diagrama de classes	40
17	DSM do sistema de farmácia	41
18	DSM do sistema de farmácia após particionamento e identificação dos ciclos	42
19	Grafo do sistema de farmácia reorganizado com destaque para os ciclos	43
20	Modelagem UML do sistema de farmácia, após segunda rodada de melhorias	44
21	Segundo particionamento com identificação de ciclos da DSM do sistema de farmácia	45

SUMÁRIO

1 Introdução **7**

1.1 Histórico 7

1.2 Objetivo 8

1.3 Justificativa 8

1.4 Organização do texto 10

2 Conceitos **11**

2.1 Grafos e DSMs 11

2.1.1 Manipulação de DSMs 13

2.2 Problemas NP-completos 13

3 Quick Triangular Matrix **15**

3.1 O problema 15

3.2 O algoritmo 15

4 Antares DSM **25**

4.1 Descrição da Ferramenta 25

4.1.1 Abrir DSM 26

4.1.2 Particionamento 26

4.1.3 Identificação de ciclos 27

4.1.4 Edição e salvamento de DSM 29

4.2 Arquivos de entrada e saída 29

4.3 Criação de casos de testes 30

5 Resultados experimentais **32**

6 Estudo de caso **37**

6.1 Objetivos 37

6.2 Escolha do caso 37

6.3 Análise do caso 39

6.3.1 Visualização de grafo e DSM 39

6.3.2 Aplicação dos métodos 41

6.3.3 Alterações no modelo 41

6.3.4 Verificações posteriores 45

7 Conclusões **46**

7.1 Dificuldades encontradas 47

7.2 Trabalhos futuros 47

7.3 Considerações finais 48

Referências **49**

1 INTRODUÇÃO

1.1 HISTÓRICO

A partir da década de 70, a quantidade de sistemas computacionais desenvolvidos cresceu significativamente. Junto com o crescimento da quantidade, houve também um aumento da complexidade e diversidade dos sistemas desenvolvidos, resultando na demanda pela aplicação de uma disciplina de "engenharia" ao software. A Engenharia de Software é a aplicação de uma abordagem sistemática, disciplinada, quantificável para o desenvolvimento, operação e manutenção de software (IEEE..., 1990), oferecendo métodos, técnicas e ferramentas para sua construção, evolução e reuso.

Uma das estratégias básicas para contornar a complexidade de qualquer sistema é *dividir-para-conquistar*. Um sistema de software grande e complexo pode ser dividido em partes menores, que se relacionam e trocam informações para um determinado fim; cada parte, por sua vez, pode ser decomposta até que uma solução ou implementação seja fornecida. Num sistema orientado a objetos, essas partes podem ser classes, métodos ou mesmo pacotes. Tais partes dependem umas das outras para, em conjunto, oferecer a funcionalidade desejada.

Modularidade de software é definida como o grau pelo qual um programa é composto por componentes discretos tal que uma mudança em um componente tem impacto mínimo em outros componentes (IEEE..., 1990). Idealmente, os módulos de um sistema devem ser relativamente independentes, de modo a diminuir impactos de mudanças, facilitar compreensão, reuso, e suporte ao desenvolvimento paralelo. Módulos devem ser fracamente acoplados e possuir alta coesão interna, de modo que a complexidade das partes e das dependências entre elas não supere a complexidade do todo.

Em um sistema grande e complexo, com centenas de classes e métodos e relacionamento entre eles, o controle e a visualização de dependências entre módulos não são tarefas triviais.

Devido a essas dificuldades, surgiu como alternativa a *Design Structure Matrix*, ou simplesmente DSM, que nada mais é do que uma matriz de dependências que representa elementos

que se relacionam. Recentemente, as DSMs têm ganhado espaço, pois mostram-se como um modelo enxuto, de fácil e rápida visualização das informações (DSMWEB, 2008).

DSMs são genéricas o suficiente para poderem ser utilizadas em projetos de diversas áreas. Além da área de Engenharia de Software, existem como exemplos estudos na área de construção de navios (PIERONI; NAVEIRO, 2005), construção civil (MANZIONE; B.MELHADO, 2007) e muitos outros.

O uso de DSMs envolve operações como particionamento e agrupamento que buscam obter melhorias específicas na modelagem de projetos com aplicação de DSM.

Smith e Eppinger (1994) refinaram o conceito de DSM. Browning (2001) revisou a aplicação da DSM para decomposição e integração de problemas, detalhou os tipos de DSM e apontou as novas direções desse modelo, destacando-se como uma das melhores contribuições encontradas na literatura. Banerjee, Carrillo e Paul (2007) fizeram uma análise da complexidade computacional envolvida nos algoritmos utilizados na manipulação da matriz (MORAES, 2007).

Recentemente, várias ferramentas surgiram para dar suporte à criação, manutenção e melhorias de DSMs, contribuindo para difundir a utilização do modelo.

1.2 OBJETIVO

O objetivo geral deste trabalho é fazer um estudo sobre as DSMs e formas de otimizá-las em relação à eliminação de ciclos indesejáveis. Em especial, tem como objetivos específicos: (1) o desenvolvimento de um algoritmo de particionamento "guloso", cuja principal finalidade é a identificação de tais ciclos, (2) sua incorporação em ferramenta de apoio para uso de DSMs, e (3) comparação do algoritmo proposto com algoritmos determinísticos existentes, apontando suas qualidades e defeitos.

1.3 JUSTIFICATIVA

Desde que foram criadas, as DSMs já se mostraram bastante eficazes, sendo utilizadas para diversos propósitos. Em planejamento e gerência de projetos, principalmente nas áreas de engenharia, é possível notar uma redução considerável no tempo total do projeto, otimizando as relações de dependência entre as atividades descritas na matriz, além de se obter uma fácil, rápida e intuitiva visualização do planejamento (PIERONI; NAVEIRO, 2005) e (MANZIONE;

B.MELHADO, 2007).

A utilização de DSMs no contexto de Engenharia de Software ainda é pequena. Porém, o uso consolidado do paradigma de objetos para o desenvolvimento de sistemas de software, em diversos domínios de aplicação, bem como a preocupação constante em controlar as dependências entre módulos, têm promovido o uso crescente de DSMs neste contexto: gerenciamento de dependências entre módulos.

Essa forma de modelagem parece ser uma das melhores formas de identificar relações de dependência entre sub-partes de um sistema, tais como: classes, pacotes, módulos ou subsistemas. As relações entre eles podem ser descritas na matriz de dependências, de modo escalável e com controle de visualização sobre implementação de estruturas mais complexas. Além de sua facilidade de visualização e compreensão do software, é possível aplicar algoritmos baseados em grafos na modelagem do software, para sugerir ao projetista modificações nas dependências entre os componentes, com o objetivo de melhorar características específicas desejáveis ao software em questão.

O particionamento de uma matriz, especificamente, tem como objetivo tornar a matriz o mais próximo possível da matriz triangular inferior, ou seja, com todos os elementos abaixo da diagonal principal. Assim, caso existam elementos acima da diagonal superior, esses elementos indicam a existência de ciclos, fazendo parte destes. O método de particionamento de DSMs será explicado com maiores detalhes no Capítulo 3.

O particionamento completo de uma matriz de dependências, com a identificação de todos os ciclos, de forma determinística, é um problema NP-completo. Isso significa que até o presente momento, todos os algoritmos propostos são exponenciais.

Os softwares modernos estão cada vez maiores e com mais partes que se relacionam. Com isso, num sistema muito grande, o tempo de particionamento de sua matriz de dependências correspondente pode se tornar inviável. O algoritmo "guloso" de particionamento proposto neste trabalho visa contribuir para a melhoria do sistema, particionando e identificando ciclos, mesmo que não seja por completo, com um algoritmo polinomial, significativamente mais rápido que os propostos até então para o caso geral.

Observando-se as vantagens comumente associadas ao uso de DSMs, descritas em estudos anteriores (DSMWEB, 2008; BROWNING, 2001; GEBALA; EPPINGER, 1991), é importante contornar as dificuldades existentes, de modo a estimular seu uso e adoção pela comunidade de Engenharia de Software. O estudo e desenvolvimento de um algoritmo de particionamento capaz de reestruturar com eficiência software orientado a objetos e outros tipos de sistemas de

outras áreas, melhorando sua modularidade, é o foco e principal contribuição desse trabalho.

1.4 ORGANIZAÇÃO DO TEXTO

O restante desta monografia está estruturada em 7 capítulos.

No Capítulo 2 são apresentados conceitos como Grafos, DSM e problemas NP-completos.

No Capítulo 3 é apresentado o algoritmo *Quick Triangular Matrix*, seu funcionamento e seu código-fonte em linguagem java.

No Capítulo 4 a ferramenta *Antares DSM* é apresentada, mostrando os passos necessários para sua utilização correta.

No Capítulo 5 são mostrados os resultados dos experimentos, comparando o algoritmo *Quick Triangular Matrix*, implementado dentro do *Antares DSM* com o algoritmo de particionamento de Gebala (GEBALA; EPPINGER, 1991).

No Capítulo 6 é apresentado um estudo de caso, que inclui a modelagem de um sistema simples de farmácia, a aplicação do algoritmo dentro da ferramenta, e a análise dos resultados obtidos.

No Capítulo 7 são apresentadas as dificuldades encontradas, apontando trabalhos futuros e fazendo as últimas considerações.

2 CONCEITOS

2.1 GRAFOS E DSMS

Um grafo é um modelo matemático que representa elementos com seus relacionamentos. Um grafo admite uma representação visual, Nesse caso os elementos são chamados de vértices e podem ser representados por pontos, letras ou componentes de livre escolha. Seus relacionamentos podem ser representados por linhas ou arcos e são chamados de arestas. A depender da aplicação, as arestas podem ou não ter direção. Se tiverem direção, suas arestas podem ser representadas por setas, que ligam um vértice a outro (DIESTEL, 2000).

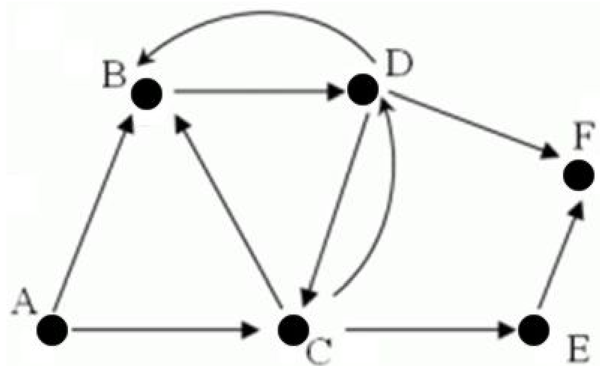


Figura 1: Exemplo de grafo

Grafos podem ser representados por matrizes. Suponha N vértices, descritos numa matriz quadrada M, NxN, onde cada vértice é representado por uma linha i, e também por uma coluna j. Diz-se que um vértice i depende de um vértice j, se há, na posição da matriz representada por ij (linha i, coluna j), algum valor marcado, numérico (indicando o grau de relacionamento) ou simplesmente um "X". Essa aplicação pode ser utilizada para descrever quaisquer tipos de componentes que se relacionem, em qualquer tipo de modelagem ou planejamento. Esse tipo de matriz foi chamada de “Design Structure Matrix” (DSM), ou matriz de estrutura de design ou, simplesmente, matriz de dependências.

Existem três tipos básicos de configuração de relacionamentos entre os elementos de um sistema em uma DSM: paralelo (ou concorrente), seqüencial (ou dependente) e acoplado (ou

interdependente) (DSMWEB, 2008) (Figura 2).

Três configurações de relacionamentos que caracterizam um sistema									
Relacionamento	Paralelo			Seqüencial			Acoplado		
Representação em grafo	A, B → C			→ A → B →			A ⇄ B		
Representação em DSM		A	B		A	B		A	B
	A			A		X	A		X
	B			B			B	X	

Figura 2: Configurações de componentes em DSM

Na configuração paralela, os elementos do sistema não interagem uns com os outros. Entender o comportamento de elementos individuais nos permite entender completamente o comportamento do sistema. Se o sistema é um projeto, então os elementos do sistema devem ser tarefas de projeto a serem executadas. Assim, a tarefa A é dita independente da tarefa B se nenhuma informação é trocada entre as duas atividades.

Na configuração seqüencial, um elemento influencia o comportamento de outro de maneira unidirecional. Isto é, os parâmetros de projeto de um elemento A são selecionados baseados nos parâmetros de projeto de um elemento B. Em termos de tarefas de projeto, a tarefa A não pode ser executada antes da tarefa B.

Finalmente, em um sistema acoplado, o fluxo de influência ou informação é atrelado: o elemento A influencia o elemento B e vice-versa. Isto deve ocorrer se um parâmetro de A, não pode ser determinado com certeza, sem primeiro conhecer um parâmetro de B e B não pode determiná-lo sem conhecer A. Esta dependência cíclica é chamada “circuitos” ou “ciclos de informação”. Neste exemplo foi exibido apenas um ciclo simples, com 2 vértices; no entanto, os ciclos podem ter qualquer quantidade inteira e maior que 1 de vértices.

No contexto de gerenciamento de projetos, o uso da DSM é válido devido a sua capacidade de representação de dependências cíclicas comuns em grande parte dos projetos, o que resulta em escalonamento de atividades melhorado e mais realista (DSMWEB, 2008).

DSMs podem ser classificadas segundo os tipos de dados que podem ser representados através delas (BROWNING, 2001):

1. *Baseada em Componentes ou Arquitetura*: Usada para modelar arquiteturas de sistemas baseadas em componentes, subsistemas e seus relacionamentos;

2. *Baseada em Equipes ou Organização*: Usada para modelar estruturas de organizações, baseadas em pessoas ou grupos e suas interações;

3. *Baseada em Atividade ou Escalonamento*: Usada para modelar processos e projetos baseados em atividades e seu fluxo de informação e outras dependências;

4. *Baseada em Parâmetros ou Escalonamento de Baixo Nível*: Usada para modelar relacionamentos de baixo nível entre decisões de projeto e parâmetros, equações de sistemas, sub-rotinas, troca de parâmetros etc.

2.1.1 MANIPULAÇÃO DE DSMS

A DSM não é somente um modelo de processos para gerência de projetos e modelagem, mas uma ferramenta para análise e melhoria da modelagem de sistemas ou processos. DSMs podem ser usadas para identificar ciclos num modelo ou identificar grupos, a fim de criar pacotes. Muitos algoritmos e métodos de análise podem ser encontrados. Gebala (1991) propôs dois algoritmos: “particionamento” e “agrupamento”. O algoritmo de particionamento reorganiza a seqüência dos itens da matriz para maximizar a disponibilidade e o fluxo de informações dos elementos. Após o particionamento, é mais fácil identificar ciclos no modelo. O algoritmo de agrupamento reorganiza a matriz formando blocos de elementos acoplados, que podem vir a se tornar pacotes, por exemplo. Kusiak (1993) propôs um método de análise qualitativa de processo de design para identificar padrões topológicos no processo (MORI et al., 1999).

Em 1999, Mori, Ishii, Kondo e Ohtomi (MORI et al., 1999) propuseram melhorias nos métodos de otimização de DSMs, baseados nos algoritmos de particionamento e agrupamento.

Além dos diversos algoritmos publicados, existem ferramentas no mercado como o `Lattix`[1] e o `Intellij Idea`[2], que aplicam alguns dos algoritmos existentes de forma a melhorar a modelagem do software.

2.2 PROBLEMAS NP-COMPLETOS

Na teoria da complexidade computacional, existe um conjunto de problemas chamado NP (Não Polinomial) e também há um conjunto de problemas chamado P (polinomiais) de modo que $P \subseteq NP$. Dentro do conjunto dos NP, existe um subconjunto de problemas chamados NP-

[1] www.lattix.com
[2] www.jetbrains.com/idea

completos. Pode-se dizer que os problemas NP-completos são os problemas mais difíceis de NP e muito provavelmente não fazem parte da classe de complexidade P.

Os problemas NP-completos são classificados como intratáveis, visto que os algoritmos sugeridos até então para sua solução possuem complexidade exponencial ou fatorial, que para entradas muito grandes tornaria sua execução inviável. Uma das estratégias para resolver alguns problemas desse tipo em tempo aceitável, é a técnica de algoritmos "gulosos", que procura resolver problemas baseados em escolhas locais, na esperança de que esta escolha leve até a solução ótima global. Esses algoritmos normalmente são simples e de fácil implementação, no entanto, a maioria desses algoritmos nem sempre conduzem o problema a uma solução ótima (CORMEN; LEISERSON; RIVEST, 1990).

A maioria dos algoritmos utilizados para resolver problemas em matrizes de dependências têm complexidade exponencial e conduzem a uma solução ótima. Para entradas de tamanho pequeno, seus resultados são aceitáveis, porém, numa matriz com muitos componentes, o analista pode ter que esperar horas apenas para ver o resultado do algoritmo aplicado à matriz.

Numa tentativa de melhorar esse tempo e desempenho associado a alguma ferramenta baseada em DSMs, foi criado um algoritmo polinomial guloso de particionamento, foco deste trabalho, e que será abordado no capítulo a seguir.

3 *QUICK TRIANGULAR MATRIX*

3.1 O PROBLEMA

O particionamento de uma DSM é o processo de manipulação, isto é, reordenação de suas linhas e colunas, de forma que a matriz resultante não contenha nenhuma marca de “feedback”, ou seja nenhum elemento que indique ciclo. O objetivo é transformar a DSM em uma matriz triangular inferior (todos os elementos acima da diagonal são nulos). Para sistemas complexos, é altamente improvável que a manipulação simples de linhas e colunas resulte em uma matriz triangular inferior (MORAES, 2007).

Ao perceber elementos acima da diagonal de uma matriz particionada, o analista pode perceber marcas de “feedback”, ou ciclos, que normalmente não são desejáveis e corrigí-los no projeto, se for possível.

A maioria dos algoritmos de particionamento utilizados atualmente têm complexidade exponencial, tornando sua execução muitas vezes inviável para grandes matrizes de dependências. Para contornar esse problema, foi criado o algoritmo **Quick Triangular Matrix**, com o objetivo principal de minimizar consideravelmente o tempo de execução do particionamento, sem prejuízos em sua análise.

Note que, por ser um algoritmo guloso, **Quick Triangular Matrix** nem sempre chegará a uma solução ótima (Em uma matriz particionada, nesse caso, podem ficar elementos que não pertencem a ciclos (sem marca de “feedback”) acima da diagonal superior); no entanto, a probabilidade de elementos acima da diagonal pertencerem a ciclos aumentará consideravelmente, e portanto, merecerem uma revisão, conforme mostram os resultados exibidos no Capítulo 5.

3.2 O ALGORITMO

O algoritmo de **Quick Triangular Matrix** é dividido em duas partes. Na primeira parte é feita uma ordenação geral dos somatórios dos graus de dependências das linhas e colunas da

matriz. Na segunda parte é feito um refinamento da ordem das linhas e colunas, comparando elementos aos pares, e trocando-os se necessário. A descrição do algoritmo em alto nível é apresentada a seguir, e seus passos são ilustrados por meio das matrizes obtidas a partir da matriz inicial mostrada na Figura 3.

Note que na matriz exemplo, para facilitar o entendimento, todas as dependências têm valor 1 e os ciclos existentes são simples, apesar de o algoritmo suportar dependências de maior grau, e também conseguir "isolar" acima da diagonal principal elementos que pertencem a ciclos com mais de 2 elementos.

	1	2	3	4	5	6	7
1		1		1	1		1
2					1		
3	1	1			1		1
4	1	1	1		1	1	
5	1	1	1				
6		1	1		1		1
7	1				1		

Figura 3: Matriz exemplo antes de aplicar o Quick Triangular Matrix

PARTE I

1. Montar 2 listas com os elementos, ordenadas (Figura 4) utilizando Quick Sort (HOARE, 1962) pelos critérios:

    1.1. Da soma dos valores dos graus de dependência no sentido dos fornecedores (colunas da matriz).

    1.2. Da soma dos valores dos graus de dependência no sentido dos dependentes (linhas da matriz).

2. Criar uma nova ordem para a matriz desempilhando alternadamente e em ordem, um elemento da lista 1.1 e colocando no final da nova ordem da matriz, e um elemento da lista 1.2 e colocando no início da nova ordem da matriz, até que não existam mais elementos. Cada elemento desempilhado de uma lista é também removido da outra lista.

3. Montar a matriz com essa ordem repetida nas linhas e colunas.

A Figura 5 mostra a matriz obtida ao final dos passos 1 a 3 da Parte I. A matriz nesse momento já tem a maior parte de suas dependências abaixo da diagonal principal.

	1	2	3	4	5	6	7	
1		1		1	1		1	4
2					1			1
3	1	1			1		1	4
4	1	1	1		1	1		5
5	1	1	1					3
6		1	1		1		1	4
7	1				1			2
	4	5	3	1	6	1	3	

Lista de fornecedores: 4, 6, 3, 7, 1, 2, 5
Lista de dependentes: 2, 7, 5, 1, 3, 6, 4

Figura 4: Matriz exemplo com as listas ordenadas de dependentes e fornecedores

	2	7	5	1	3	6	4
2			1				
7			1	1			
5	1			1	1		
1	1	1	1				1
3	1	1	1	1			
6	1	1	1		1		
4	1		1	1	1	1	

Figura 5: Matriz exemplo após o término da parte I

PARTE II

4. Percorrendo as colunas de trás para frente (ou de N para 1) e as linhas de baixo para cima (ou de N para 1), procurar dependências acima da diagonal (célula com valor maior que 0 na matriz).

5. Se achar dependência em (i,j), formar quadrado formado pelo conjunto das células entre (i,j) e (j,i).

    5.1. Somar bordas do triângulo superior (elementos (i,(i+1 até j)) e ((j+1 até j-1),j)) e do triângulo inferior (elementos ((i+1 até j),i) e (j,(i+1 até j-1))), onde i representa o elemento dependente encontrado no relacionamento (linha) e j o elemento fornecedor de informação do relacionamento (coluna)

    5.2. Se a soma da borda do triângulo superior for maior que a soma da borda do triângulo inferior, trocar(i,j) nas linhas e nas colunas.

A Figura 6 mostra um exemplo em que não será feita a troca entre os elementos 1 e 4 e será feita a troca entre os elementos 7 e 1.

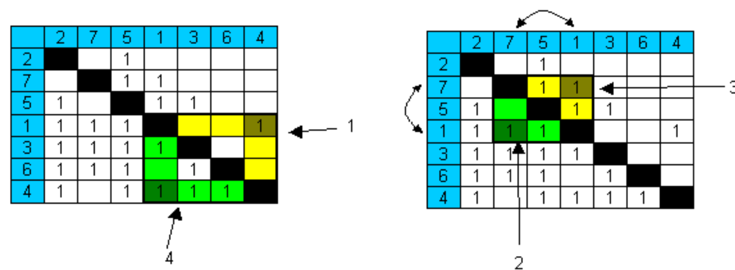


Figura 6: Um exemplo sem troca e outro com troca de elementos

Após a aplicação do algoritmo, note que a matriz de dependências, que tinha 10 elementos acima da diagonal, passou a ter apenas 5 (Figura 7).

É possível observar que todos os elementos que sobraram acima da diagonal da matriz, neste caso, formam ciclos simples de $X \rightarrow Y \rightarrow X$. Esses elementos devem ser revisados pelo analista, para saber se são realmente necessários em seu projeto (Figura 8). Em caso de modelagens que contêm dependências com grau maior que 1, o elemento de cada ciclo com maior probabilidade de ficar acima da diagonal principal é o elemento com menor valor, e portanto que será mais facil ser removido pelo analista. Note também que o algoritmo mantém as dependências originais, ele apenas modifica a ordem de seus elementos, para facilitar a identificação

	2	1	5	7	3	6	4
2			1				
1	1		1	1			1
5	1	1			1		
7		1	1				
3	1	1	1	1			
6	1		1	1	1		
4	1	1	1		1	1	

Figura 7: Matriz exemplo após aplicação do algoritmo

de ciclos. A necessidade do ciclo em cada projeto é interpretativa, e portanto, a decisão sobre sua retirada deve ser do analista, e não do algoritmo.

	2	1	5	7	3	6	4
2			1				
1	1		1	1			1
5	1	1			1		
7		1	1				
3	1	1	1	1			
6	1		1	1	1		
4	1	1	1		1	1	

Figura 8: Matriz exemplo exibindo elementos pertencentes a ciclos

Nas listagens abaixo são apresentados trechos da classe DSM, utilizada na ferramenta Antares DSM, com o algoritmo **Quick Triangular Matrix** em Java. No capítulo 5 o algoritmo terá o seu desempenho avaliado.

Listagem 3.1: Atributos da classe DSM

```
public class DSM {

    private int NdeVertices;
                //Número de vertices contando com o cabeçalho

    private ArrayList<String> listaVertices
                = new ArrayList<String>();

    private int matrizDependencias[][];
    private int matrizDependenciasCalculada[][];

...
...
}
```

Listagem 3.2: Parte I do algoritmo

```
public void QuickTriangularMatrix(){
    ArrayList<ElementoMatriz> listaFornecedores
                = new ArrayList<ElementoMatriz>();
    ArrayList<ElementoMatriz> listaDependentes
                = new ArrayList<ElementoMatriz>();
    ElementoMatriz elemento;
    int soma, somaInferior, somaSuperior;

    //--------------------------------------------------------------
    // Parte I do algoritmo:
    //
    // 1- Monta duas listas em ordem crescente, uma com a
    //      soma dos valores dos fornecedores (Colunas)
    //      e outra com a soma dos valores dos dependendentes
    //      (Linhas)
    // 2- Pega na ordem crescente, um item da lista de
    //      fornecedores e coloca no final e um item da lista
    //      de dependentes e coloca no início
    // 3- Monta matriz calculada com essa ordem
```

```
//------------------------------------------------------------------

//Monta as listas para serem ordenadas
//Lista de fornecedores
for(int i = 1; i < NdeVertices; i++){
    soma = 0;
    for (int j = 1; j < NdeVertices; j++) {
        soma = soma + matrizDependencias[j][i];
    }
    elemento = new ElementoMatriz(i,soma);
    listaFornecedores.add(elemento);
}

//Lista de dependentes
for(int i = 1; i < NdeVertices; i++){
    soma = 0;
    for (int j = 1; j < NdeVertices; j++) {
        soma = soma + matrizDependencias[i][j];
    }
    elemento = new ElementoMatriz(i,soma);
    listaDependentes.add(elemento);
}

listaFornecedores = OrdenaLista(listaFornecedores);
listaDependentes = OrdenaLista(listaDependentes);

matrizDependenciasCalculada
            = new int[NdeVertices][NdeVertices];

//Monta nova matriz, pegando na ordem um elemento da
//lista de fornecedores e colocando no final
//e um da lista de dependentes e colocando no
//início alternadamente
for(int i = 0; listaFornecedores.size() > 0; i++){
    int idAtual;
```

```
        idAtual = listaFornecedores.get(0).id;
        matrizDependenciasCalculada[NdeVertices - 1 - i][0]
                = idAtual;
        matrizDependenciasCalculada[0][NdeVertices - 1 - i]
                = idAtual;

        listaFornecedores.remove(0);
        //Procura mesmo elemento na outra lista e remove
        for(int j = 0; j < listaDependentes.size(); j++){
            if(listaDependentes.get(j).id == idAtual){
                listaDependentes.remove(j);
                break;
            }
        }



        if(listaDependentes.size() > 0 ){
            idAtual = listaDependentes.get(0).id;
            matrizDependenciasCalculada[i + 1][0] = idAtual;
            matrizDependenciasCalculada[0][i + 1] = idAtual;

            listaDependentes.remove(0);
            //Procura mesmo elemento na outra lista e remove
            for(int j = 0; j < listaFornecedores.size(); j++){
                if(listaFornecedores.get(j).id == idAtual){
                    listaFornecedores.remove(j);
                    break;
                }
            }
        }
    }

    atualizaMatriz();
...
```

Listagem 3.3: Parte II do algoritmo

...

```
//------------------------------------------------------------------
// Parte II do algoritmo:
//
// 1- Varrendo as colunas de trás para frente e as linhas
//       de baixo para cima, procurar dependências
//       - Se achar dependência em (i,j), formar quadrado
//           com (j,i) e somar bordas do triangulo
//           inferior e superior e comparar
//       - Se a soma do triângulo superior for maior que
//           a soma do triângulo inferior, trocar (i,j)
//------------------------------------------------------------------

for(int j = NdeVertices - 1; j > 0; j--){
    for (int i = j - 1; i > 0; i--) {
        if( matrizDependenciasCalculada[i][j]>0 ){
        //Tem dependência
            somaInferior = 0;
            somaSuperior = 0;

            for(int k = i; k <= j; k++){
                somaSuperior = somaSuperior
                    + matrizDependenciasCalculada[i][k];
                somaInferior = somaInferior
                    + matrizDependenciasCalculada[k][i];

                if(k != j && k != i){
                    somaSuperior = somaSuperior
                        + matrizDependenciasCalculada[k][j];
                    somaInferior = somaInferior
                        + matrizDependenciasCalculada[j][k];
                }
            }
```

```
                if(somaSuperior > somaInferior){
                    //Troca
                    matrizDependenciasCalculada[i][0]
                        = matrizDependenciasCalculada[0][j];
                    matrizDependenciasCalculada[j][0]
                        = matrizDependenciasCalculada[0][i];

                    matrizDependenciasCalculada[0][i]
                        = matrizDependenciasCalculada[i][0];
                    matrizDependenciasCalculada[0][j]
                        = matrizDependenciasCalculada[j][0];

                    atualizaMatriz();

                    j++; // Para analisar o novo
                         // elemento que ficou na posição

                    break;
                }
            }
        }
    }
}
```

4 ANTARES DSM

No cenário atual podemos encontrar algumas poucas ferramentas para manipular DSMs. Dois exemplos conhecidos são as ferramentas Lattix[1] e Intellij IDEA[2]. Essas ferramentas são proprietárias e de código fechado, além de utilizarem algoritmos de complexidade exponencial, que podem elevar muito o tempo de execução para entradas muito grandes, tornando o seu uso inviável.

Para suprir essa necessidade, foi criada a ferramenta **Antares DSM**, uma ferramenta livre sob licença LGPL (GNU Lesser General Public License[3]). Isso significa que ela pode ser copiada sem nenhum custo e pode ter seu código alterado por outros desenvolvedores. Além disso, **Antares DSM** utiliza como algoritmo de particionamento o **Quick Triangular Matrix**, explicado no capítulo 3, de melhor desempenho, como mostrado no capítulo 5.

Uma outra característica da ferramenta é ter seus textos escritos em português, podendo tornar-se uma referência de ferramenta no idioma, já que até o momento da publicação desse trabalho não foi encontrada ferramenta semelhante.

Neste capítulo são apresentadas as funcionalidades básicas da ferramenta **Antares DSM** (seção 4.1), o formato de representação dos arquivos de entrada e saída (seção 4.2) e o módulo de criação de casos de teste (seção 4.3).

4.1 DESCRIÇÃO DA FERRAMENTA

Nesta seção são descritas as funcionalidades básicas de AntaresDSM: abrir e salvar DSM, fazer particionamento e identificar ciclos. Exemplos de uso de AntaresDSM são mostrados no Capítulo 6.

---

[1] www.lattix.com

[2] www.jetbrains.com/idea

[3] Mais informações sobre a licença podem ser encontradas em http://www.gnu.org/copyleft/lgpl.html

4.1.1 ABRIR DSM

Um arquivo de DSM pode ser aberto através do item do menu *Arquivo* → *Abrir*. Após a escolha do arquivo desejado, o **Antares DSM** abrirá duas janelas, uma contendo o grafo que representa as dependências e outra contendo a matriz de dependências (Figura 9).

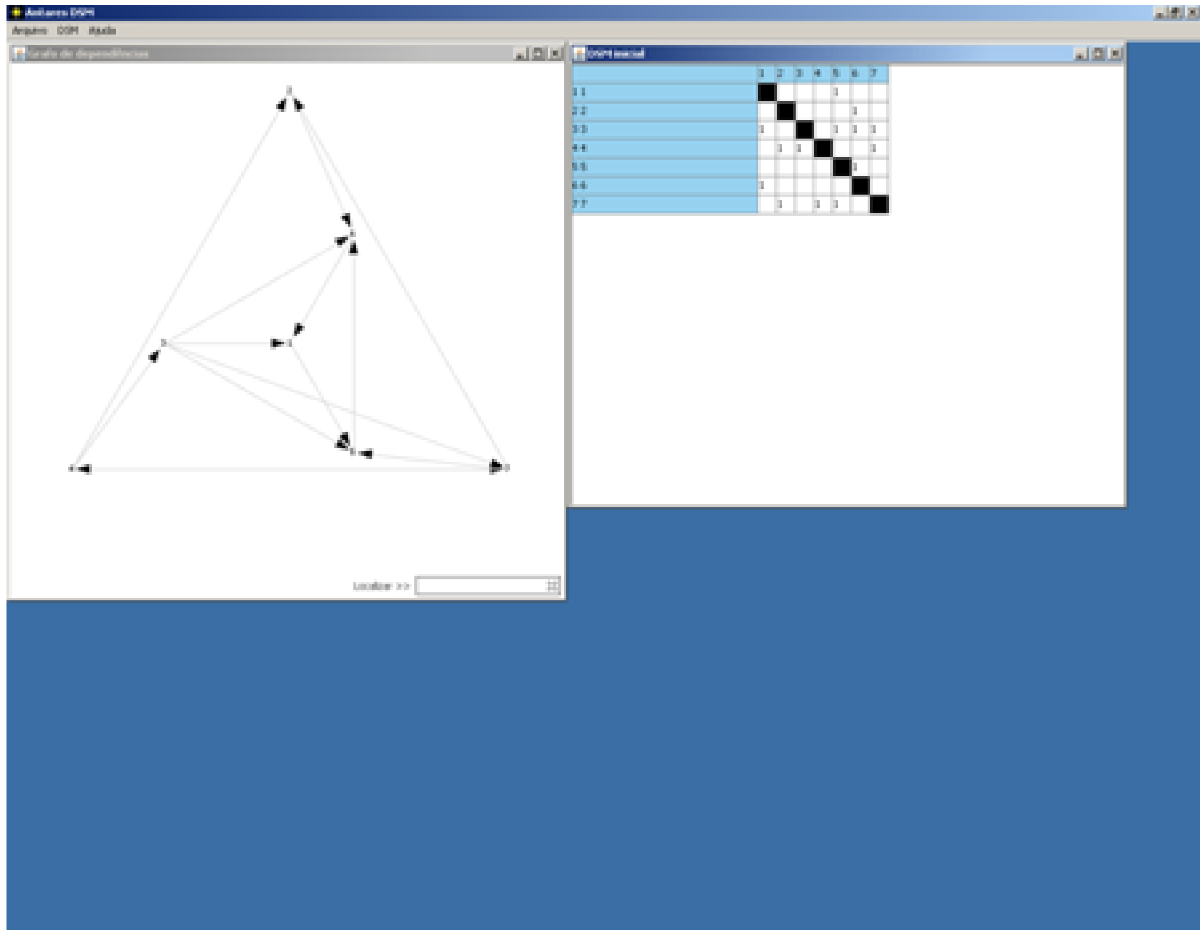


Figura 9: Antares DSM - Grafo e matriz de dependências

O grafo utilizado na primeira janela foi construído utilizando a ferramenta livre **Prefuse**[4], que permite visualizar, mover arestas e aplicar "zoom" ao grafo.

Na segunda janela é exibida a matriz de dependências, com os nomes dos componentes e as suas dependências.

4.1.2 PARTICIONAMENTO

O particionamento da matriz de dependências em AntaresDSM utiliza o algoritmo **Quick Triangular Matrix** através do item do menu *DSM* → *Particionar*. O **Antares DSM** abrirá uma

[4]Mais informações sobre a ferramenta em http://prefuse.org

nova janela (mantendo as anteriores) com a matriz particionada (Figura 10).

DSM calculada

	1	6	2	5	7	3	4
1 1				1			
6 6	1						
2 2		1					
5 5		1					
7 7			1	1			1
3 3	1	1		1	1		
4 4			1		1	1	

Figura 10: Antares DSM - Matriz particionada pelo algoritmo Quick Triangular Matrix

Note que se a matriz não contiver ciclos, ela deve ficar no formato triangular inferior. Caso existam ciclos, todos os elementos que se encontram acima da diagonal, devem pertencer a algum ciclo. Como o algoritmo é “guloso”, em alguns casos ele pode exibir acima da diagonal elementos que não pertencem a ciclo algum, já que ele não testa todas as possibilidades possíveis, embora esses casos sejam raros, conforme os experimentos exibidos no capítulo 5. Para garantir a identificação dos ciclos, AntaresDSM dá suporte a mais uma funcionalidade, apresentada a seguir.

4.1.3 IDENTIFICAÇÃO DE CICLOS

Com o particionamento concluído, é possível identificar os ciclos através do item do menu *DSM* → *Identificar ciclos*. Os resultados serão exibidos na mesma janela descrita no item anterior. No resultado dessa funcionalidade, as células acima da diagonal superior que pertencem a ciclos são pintadas de vermelho (Figura 11).

DSM calculada

	1	6	2	5	7	3	4
1 1				1			
6 6	1						
2 2		1					
5 5		1					
7 7			1	1			1
3 3	1	1		1	1		
4 4			1		1	1	

Figura 11: Antares DSM - Identificação e visualização de ciclos na DSM

As células em vermelho indicam ao analista que aquela dependência deve ser revisada e, se possível, removida. Em caso de dependências múltiplas, ou seja, com grau maior que 1, o item do ciclo que ficará acima da diagonal superior é o elo mais “fraco” do ciclo, ou seja, o que tem menor valor, e portanto, o que pode ser removido mais facilmente.

A identificação de ciclos apenas após o particionamento “guloso”, garante um desempenho melhor à execução com a finalidade de identificar ciclos, pois não é necessário testar os ciclos para todas as dependências da matriz, e sim, apenas para os que restaram acima da diagonal. Dessa forma, a identificação de ciclos complementa o particionamento “guloso”, conseguindo identificar ciclos de maneira eficiente. Juntos os dois algoritmos alcançam o resultado ótimo para o problema proposto, já que o algoritmo de identificação de ciclos verifica se os elementos acima da diagonal pertencem a ciclos percorrendo os caminhos de maneira recursiva, até ser encontrado algum ciclo, ou percorridos todos os caminhos possíveis a partir daquele elemento.

Essa funcionalidade também é útil para validar resultados do **Quick Triangular Matrix**, após sua execução. Caso existam elementos acima da diagonal superior na matriz particionada

que não foram pintados de vermelho, então o algoritmo **Quick Triangular Matrix** não obteve resultado ótimo.

4.1.4 EDIÇÃO E SALVAMENTO DE DSM

Na janela com a DSM calculada (particionada, com ou sem identificação de ciclos), e apenas nela, é possível editar os valores da DSM. Para isso, basta o analista clicar duas vezes na célula que deseja editar e alterar seu valor. Estas alterações devem ser as mesmas que o analista propôs para o seu modelo de projeto, e normalmente é interpretativa. Após as edições o analista pode salvar a nova DSM através do item do menu *Arquivo* → *Salvar*. A DSM salva poderá ser aberta e particionada novamente após suas alterações, se o analista assim desejar.

4.2 ARQUIVOS DE ENTRADA E SAÍDA

O **Antares DSM** utiliza um formato de arquivo de entrada texto, simples, que pode ser entendido facilmente por um profissional de qualquer área em que as DSMs possam ser aplicadas, tornando a utilização da ferramenta mais ampla. Uma outra vantagem desse formato de entrada, é a possibilidade de criação de módulos em outras ferramentas para exportação dos dados para o Antares DSM, como por exemplo, uma ferramenta de engenharia reversa de software, exportar as dependências entre as classes para o Antares DSM.

O arquivo de entrada é composto de duas partes.

A primeira é formada por uma lista com o número de cada componente (começando por 1, e em ordem) e a descrição dos componentes utilizados na DSM, separados por um espaço em branco. Cada elemento da lista é separado de outro por uma quebra de linha.

A primeira parte se encerra com uma linha contendo a palavra-chave “dep”.

A segunda parte do arquivo é formada pela lista de dependências. Cada elemento da lista de dependências é separado de outro por uma quebra de linha, e contém três valores em ordem e separados por espaço: **elemento dependente, elemento fornecedor, valor da dependência**, na forma *X Y VAL*, onde o elemento *X* depende do elemento *Y* e *VAL* é o valor da dependência. O valor da dependência é opcional e, se não for escrito, o valor será 1.

A listagem 4.1 mostra um exemplo de arquivo de entrada para o Antares DSM.

Listagem 4.1: Exemplo de arquivo de entrada

```
1 Classe1
2 Classe2
3 Classe3
4 Classe4
5 Classe5
6 Classe6
7 Classe7
8 Classe8
9 Classe9
dep
1 4
3 4
2 1
4 6
2 7 5
7 8 10
9 3
5 6 1
5 2
4 9
```

4.3 CRIAÇÃO DE CASOS DE TESTES

A ferramenta **Antares DSM** permite a criação de casos de testes para estudos dos processos que envolvem DSMs. Essa funcionalidade está disponível no item do menu *Arquivo* → *Novo* (Figura 12).

O **Antares DSM** permite ao analista criar uma nova DSM baseada em três parâmetros:

1. Quantidade de elementos da matriz a ser criada. A ferramenta também pode criar uma quantidade aleatória se o usuário assim desejar, digitando o valor 0 para esse parâmetro.
2. Grau de população dos elementos na matriz. Se a matriz criada for não vazia, então ela

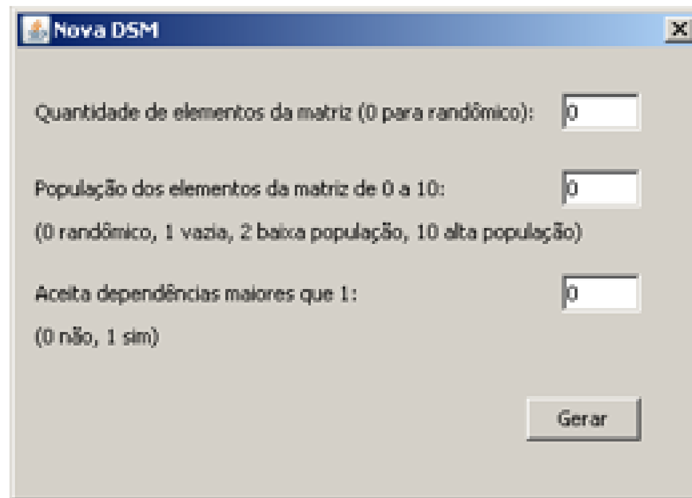


Figura 12: Antares DSM - Criação de casos de teste

será populada aleatoriamente. No entanto, é possível escolher a densidade dos elementos na matriz. A ferramenta varre cada célula da matriz aleatória, e gera um número real aleatório entre 0 e 1. Se esse número for menor do que o fator escolhido, a célula será populada com um valor de dependência. O fator utilizado para o cálculo pode ser aleatório (um número real entre 0 e 1), ou escolhido pelo analista, respeitando a fórmula:

$$Fator = (ValorEscolhido - 1) \times 0,1$$

Ou seja, escolhendo 2 (baixa população), o fator será 0,1, tendo uma população média em torno de 10% da matriz, e escolhendo 10 (alta população), o fator será 0,9, tendo uma população média em torno de 90% da matriz.

3. Valores das dependências. O analista pode escolher se a matriz gerada terá elementos com dependências múltiplas, ou seja, com valores maiores que 1. Na ferramenta, o analista deve entrar com o valor 0 (Só aceita dependências unitárias) ou 1 (aceita dependências múltiplas). Caso o analista tenha optado por dependências múltiplas, a matriz gerada terá elementos com valores aleatórios entre 1 e 100 com igual probabilidade, quando tiver uma célula preenchida de acordo com as regras do item anterior.

5 *RESULTADOS EXPERIMENTAIS*

Este capítulo tem como objetivo apresentar alguns resultados experimentais relacionados ao algoritmo **Quick Triangular Matrix**, implementado e utilizado pela ferramenta **Antares DSM**, e sua comparação com o algoritmo de particionamento de complexidade exponencial criado por *David A. Gebala* e *Steven D. Eppinger* em 1991(GEBALA; EPPINGER, 1991), implementado numa macro do *Microsoft Excel*.

Para cada algoritmo, foram criadas matrizes aleatórias de 10x10, 50x50 e 100x100, com pequena população (probabilidade de 10% de cada célula ser preenchida), média população (probabilidade de 50% de cada célula ser preenchida) e alta população (probabilidade de 90% de cada célula ser preenchida) cada uma, sendo 10 testes para cada combinação de *tamanho da matriz X tamanho da população*, com exceção das combinações de matrizes 100x100, onde foram feitos 5 testes para cada. Para os testes foram computados os tempos de execução de cada algoritmo dentro da sua ferramenta e se chegaram ou não ao resultado ótimo, e calculadas a média e desvio padrão para cada caso.

Os testes foram realizados num computador com 512MB de memória RAM e processador *Sempron* de 1,60GHz.

Os resultados dos testes são apresentados abaixo.

Matriz 10 elementos pouco populosa (0,1) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,3	0,3	0,2	0,1	0,3	0,2	0,3	0,1	0,2	0,23	0,08232726
Resultado ótimo (S N)	S	N	S	N	S	S	S	N	S	S	70%	

Matriz 10 elementos pop média (0,5) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,1	0,1	0,2	0,1	0,1	0,2	0,2	0,2	0,2	0,17	0,067494856
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 10 elementos populosa (0,9) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,2	0,2	0,2	0,1	0,1	0,2	0,1	0,2	0,1	0,1	0,15	0,052704628
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 10 elementos	90%	0,183333	0,074663998

Matriz 50 elementos pouco populosa (0,1) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,1	0,2	0,3	0,3	0,3	0,3	0,2	0,3	0,2	0,25	0,070710678
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos pop média (0,5) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,2	0,2	0,2	0,3	0,3	0,3	0,2	0,1	0,3	0,24	0,06992059
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos populosa (0,9) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,2	0,3	0,2	0,3	0,2	0,2	0,4	0,2	0,2	0,3	0,25	0,070710678
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 50 elementos	100%	0,246667	0,068144539

Matriz 100 elementos pouco populosa (0,1) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,3	0,3	0,3	0,2						0,28	0,04472136
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos pop média (0,5) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,5	0,3	0,4	0,2	0,3						0,34	0,114017543
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos populosa (0,9) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,5	0,4	0,3	0,3	0,4						0,38	0,083666003
Resultado ótimo (S N)	S	S	S	S	S						100%	

Média c/ 100 elementos	100%	0,333333	0,089973541

Média Quick Triangular Matrix - Antares DSM	96%	0,2	0,092842985

Matriz 10 elementos pouco populosa (0,1) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,4	0,2	0,3	0,4	0,4	0,2	0,2	0,2	0,3	0,2	0,28	0,091893658
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 10 elementos pop média (0,5) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,4	0,3	0,3	0,3	0,2	0,2	0,3	0,4	0,2	0,2	0,28	0,078881064
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 10 elementos populosa (0,9) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,3	0,3	0,3	0,3	0,2	0,2	0,3	0,3	0,3	0,28	0,042163702
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 10 elementos 100% | 0,28 | 0,071438423

Matriz 50 elementos pouco populosa (0,1) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	12,1	12,1	12	12	12,1	12,1	12,1	12,3	12,1	12,1	12,1	0,081649658
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos pop média (0,5) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	13,8	13,7	13,9	13,9	13,7	13,7	13,9	13,8	13,7	13,8	13,79	0,087559504
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos populosa (0,9) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	14,5	14,4	14,5	14,5	14,5	14,7	14,5	14,4	14,5	14,5	14,5	0,081649658
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 50 elementos 100% | 13,463 | 1,0270424

Matriz 100 elementos pouco populosa (0,1) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	185	185	184	185	185						184,8	0,447213595
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos pop média (0,5) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	210	210	209	210	209						209,6	0,547722558
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos populosa (0,9) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	220	219	220	220	219						219,6	0,547722558
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 100 elementos 100% | 204,67 | 15,15004322

Média Particionamento NP Completo 100% 46,4 80,14559551

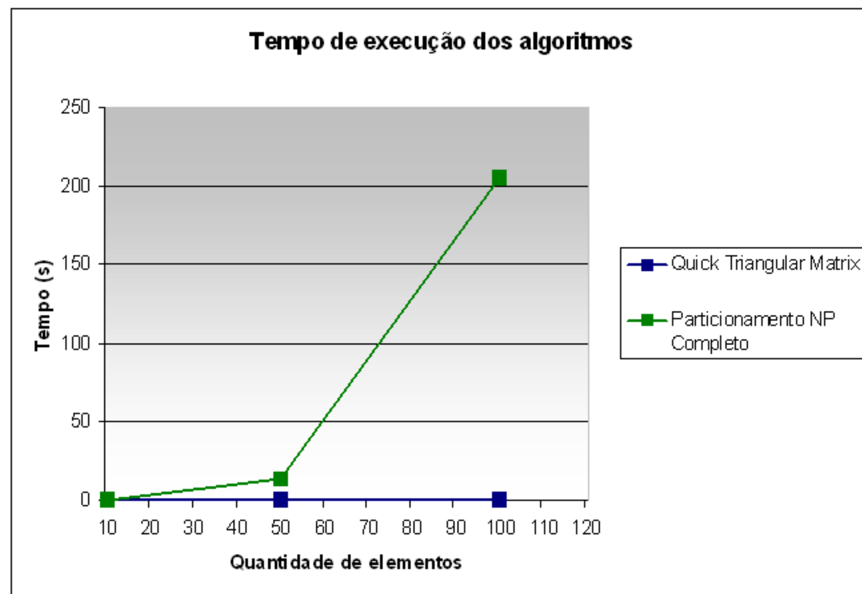


Figura 13: Tempo de execução de cada algoritmo

A Figura 13 apresenta uma interpretação gráfica dos resultados em relação ao tempo de execução de cada algoritmo. Observa-se que para matrizes com poucos elementos, a diferença do tempo de execução é praticamente irrelevante, dentro da margem de erro apontada pelo desvio padrão; no entanto, enquanto o tempo do algoritmo **Quick Triangular Matrix** cresce quase linearmente, o tempo do particionamento de *Gebala* e *Eppinger* cresce exponencialmente, tornando a diferença de tempo entre os dois algoritmos cada vez maior conforme a quantidade de elementos aumenta.

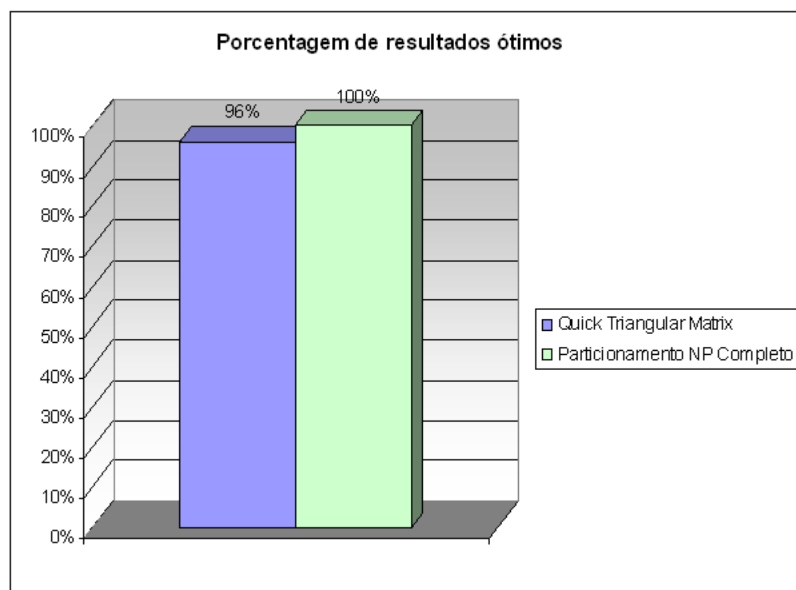


Figura 14: Porcentagem de resultados ótimos em cada algoritmo

A Figura 14 apresenta uma interpretação gráfica dos resultados em relação à porcentagem de resultados ótimos obtidos com a aplicação de cada algoritmo. Nesse quesito, o algoritmo de *Gebala* e *Eppinger* mostrou-se melhor do que o **Quick Triangular Matrix**, como era esperado, atingindo o resultado ótimo em 100% dos testes, contra 96% do **Quick Triangular Matrix**.

Enquanto o algoritmo *Gebala* e *Eppinger* testa todas as possibilidades para tornar a matriz triangular inferior, o **Quick Triangular Matrix** faz a ordenação baseada em heurísticas, na tentativa de encontrar o problema com o mínimo de trocas possíveis. As técnicas heurísticas não asseguram as melhores soluções, mas somente soluções válidas, aproximadas. No entanto, em geral, têm desempenhos melhores em relação a outras técnicas de complexidade exponencial.

6 ESTUDO DE CASO

6.1 OBJETIVOS

Este estudo de caso tem como objetivo realizar uma avaliação preliminar sobre os benefícios e as limitações do uso da DSM para visualização, compreensão e otimização de modelagens, assim como da ferramenta **Antares DSM** e do algoritmo de particionamento **Quick Triangular Matrix**.

A proposta do estudo de caso é utilizar uma modelagem em fase inicial de projeto, e tentar identificar problemas de ciclos indesejáveis nessa modelagem utilizando **Antares DSM**.

A ferramenta deve apontar ao analista os ciclos existentes no projeto, que, por sua vez, deve revisá-los e, se possível, modificar o projeto.

6.2 ESCOLHA DO CASO

Embora DSMs possam ser utilizadas em muitas áreas profissionais, foi escolhido um caso da área de *Engenharia de Software*, sub-área de *Ciência da Computação* para ser analisado.

A aplicação a ser estudada é um sistema de uma farmácia. Esse sistema foi escolhido por ser um sistema simples e didático. A Figura 15 apresenta uma visão estrutural do sistema por meio de um diagrama de classes em UML. A Linguagem de Modelagem Unificada (Unified Modeling Language - UML) (OMG, 2008) é uma linguagem de propósito geral para a especificação, visualização, construção e documentação de artefatos de sistemas de software. Basicamente, a UML permite que desenvolvedores visualizem os produtos de seu trabalho em diagramas padronizados(WIKIPÉDIA, 2008).

Figura 15: Modelagem UML inicial do sistema de farmácia

6.3 ANÁLISE DO CASO

Para a análise do caso, foi gerado um arquivo no formato de entrada padrão da ferramenta **Antares DSM** (Listagem 6.1), correspondendo ao diagrama de classes para o sistema de farmácia (Figura 15). Na seqüência, a matriz foi gerada e o fluxo padrão da ferramenta realizado (particionamento e identificação de ciclos).

Listagem 6.1: Arquivo de entrada do sistema de Farmácia

```
1 Farmacia
2 ItemVenda
3 Produto
4 Lote
5 ItemCompra
6 Fornecedor
7 Funcionario
8 ItemDeLote
9 Compra
10 Venda
11 Caixa
12 Cargo
13 ItemDeSalario
14 Salario
15 Loja
dep
3 2
3 4
4 8
8 15
10 2
5 3
9 5
6 9
7 6
1 6
1 9
1 7
10 1
1 15
11 7
12 7
7 14
15 7
14 13
13 12
11 10
```

6.3.1 VISUALIZAÇÃO DE GRAFO E DSM

As Figuras 16 e 17 mostram o grafo e a DSM correspondentes ao diagrama de classes do sistema da farmácia. Apesar da DSM promover uma visualização compacta e de fácil entendimento do sistema como um todo, não é trivial a localização de seus ciclos, mesmo sendo num sistema pequeno. Num sistema maior, isso poderia demandar um tempo grande do analista ao fazer essa atividade manualmente.

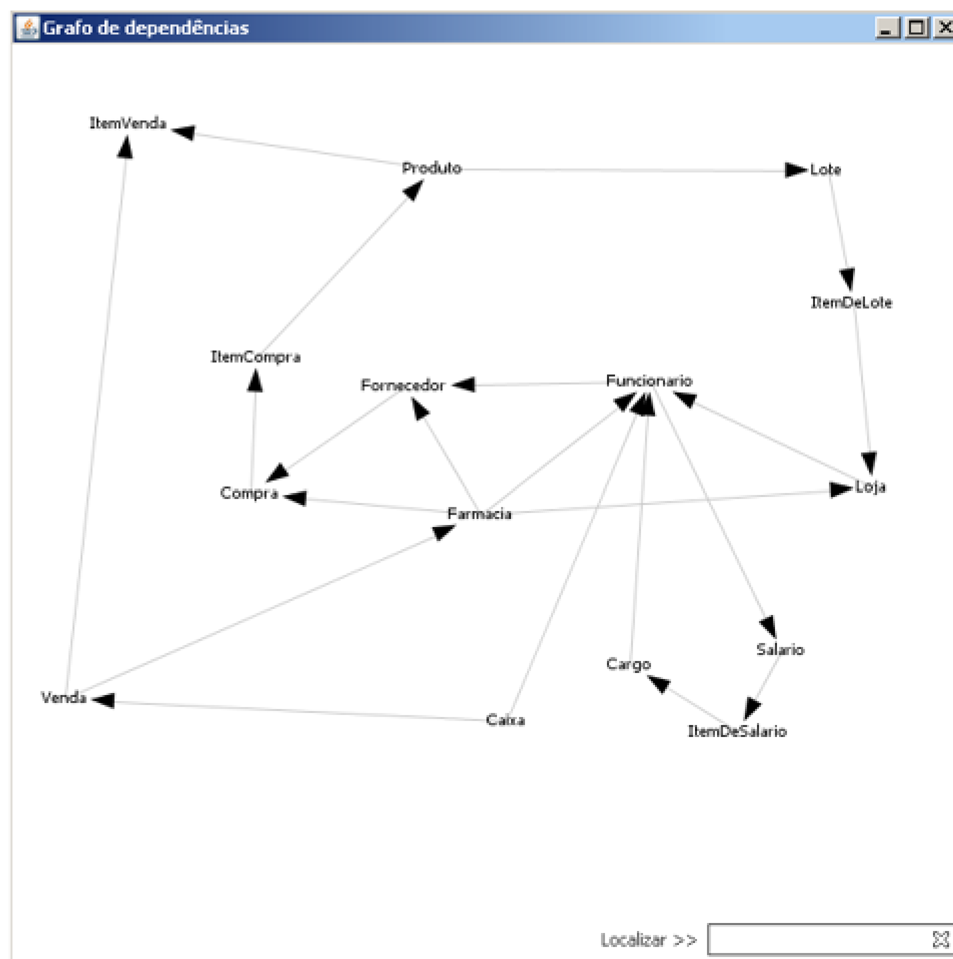


Figura 16: Grafo do sistema de farmácia organizado como no diagrama de classes

DSM inicial

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1 Farmacia						1	1		1						1
2 ItemVenda															
3 Produto		1		1											
4 Lote								1							
5 ItemCompra			1												
6 Fornecedor									1						
7 Funcionario						1								1	
8 ItemDeLote															1
9 Compra					1										
10 Venda	1	1													
11 Caixa							1			1					
12 Cargo							1								
13 ItemDeSalario												1			
14 Salario													1		
15 Loja							1								

Figura 17: DSM do sistema de farmácia

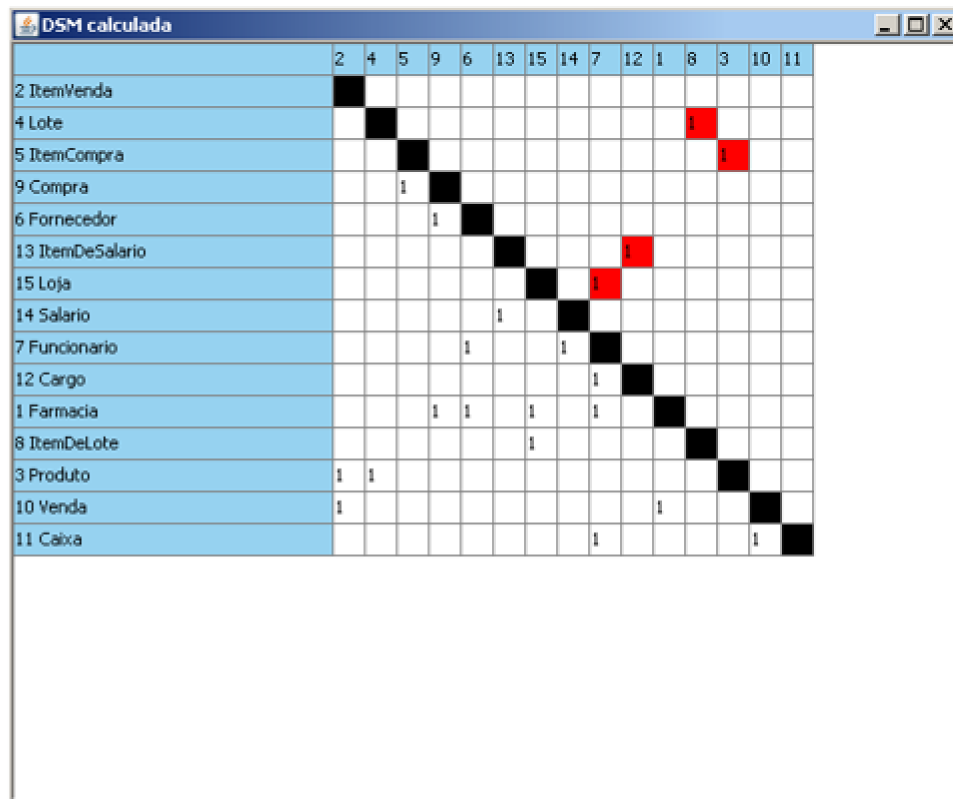
6.3.2 APLICAÇÃO DOS MÉTODOS

Após a visualização inicial, foram aplicados os métodos de particionamento e identificação de ciclos, respectivamente. O resultado da aplicação dos métodos é a matriz mostrado na Figura 18. Esses resultados mostram que as dependências 4 → 8, 5 → 3, 13 → 12 e 15 → 7, pintadas de vermelho, pertencem a ciclos que devem ser reavaliados. Rearrumando o grafo, é possível comprovar a existência desses ciclos, como mostra a Figura 19.

6.3.3 ALTERAÇÕES NO MODELO

Com base nos resultados anteriores, o analista deve reavaliar o modelo nos ciclos indicados. Após a reavaliação, foram realizadas as seguintes mudanças:

- Retirada do relacionamento entre as classes *Funcionario* e *Fornecedor*, e inserção do relacionamento entre *Compra* e *Funcionario*, já que o funcionário contata o fornecedor dentro do mundo real, mas no sistema o que é interessante, nesse caso, é fazer uma auditoria das compras feitas pelo funcionário.
- Inversão do sentido da dependência *Fornecedor* → *Compra* para *Compra* → *Fornecedor*,


DSM calculada

	2	4	5	9	6	13	15	14	7	12	1	8	3	10	11
2 ItemVenda	■														
4 Lote		■										1			
5 ItemCompra			■										1		
9 Compra			1	■											
6 Fornecedor				1	■										
13 ItemDeSalario						■				1					
15 Loja							■		1						
14 Salario						1		■							
7 Funcionario					1			1	■						
12 Cargo									1	■					
1 Farmacia				1	1		1		1		■				
8 ItemDeLote							1					■			
3 Produto	1	1											■		
10 Venda	1										1			■	
11 Caixa									1					1	■

Figura 18: DSM do sistema de farmácia após particionamento e identificação dos ciclos

partindo do pressuposto que o fornecedor não fornece a compra dentro da visão da farmácia, e sim, que a compra é feita de um fornecedor.

- Inversão do sentido da dependência $ItemDeLote \rightarrow Loja$ para $Loja \rightarrow ItemDeLote$, já que para a farmácia é melhor para controlar o estoque sabendo quais itens de lote estão em cada loja, do que qual loja pertence a cada item de lote.

- Inversão do sentido da dependência $Cargo \rightarrow Funcionario$ para $Funcionario \rightarrow Cargo$. Apesar de um cargo possuir um ou mais funcionários, a informação mais relevante nesse caso é saber o cargo de cada funcionário.

- Inversão do sentido da dependência $ItemDeSalario \rightarrow Cargo$ para $Cargo \rightarrow ItemDeSalario$. Nesse caso houve um erro grave de modelagem, já que um item de salário não possui um cargo, e sim um cargo que pode estar ligado a itens de salários, para definir o salário inicial de determinado funcionário baseado no seu cargo.

Após as alterações, foi gerado um novo diagrama de classes UML (Figura 20).

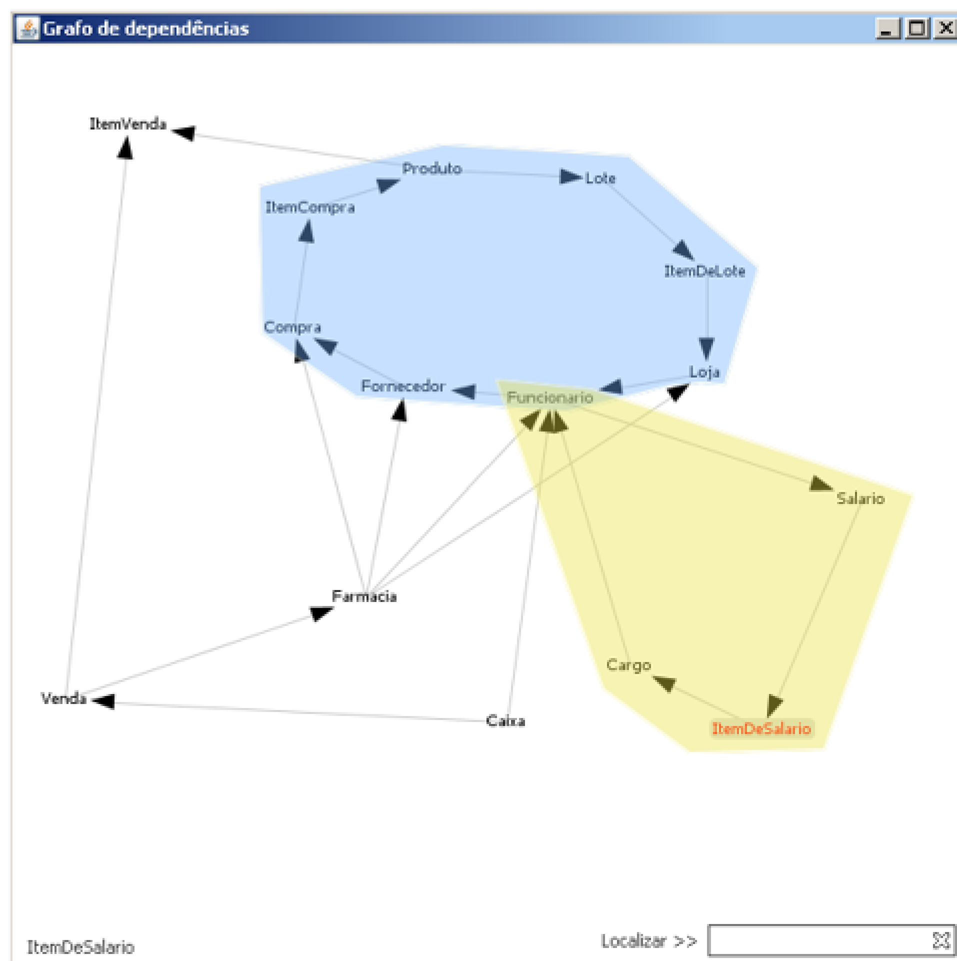


Figura 19: Grafo do sistema de farmácia reorganizado com destaque para os ciclos

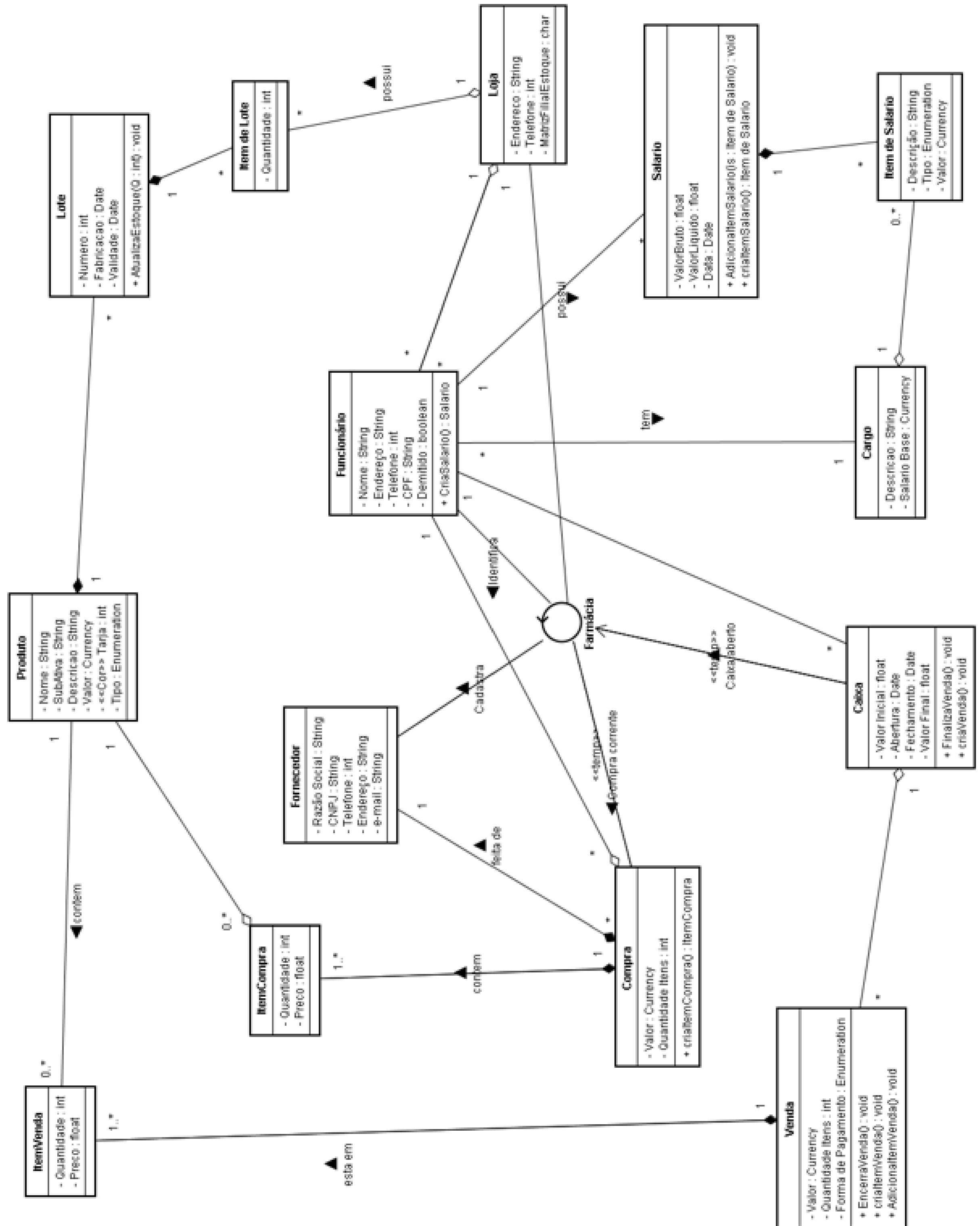


Figura 20: Modelagem UML do sistema de farmácia, após segunda rodada de melhorias

6.3.4 VERIFICAÇÕES POSTERIORES

Após as alterações descritas no item anterior, o analista deve atualizar a matriz, salvando-a logo em seguida. Como os processos de modelagem são iterativos, qualquer alteração na modelagem deve levar a uma alteração na DSM, que por sua vez, deve ser particionada novamente, e depois submetida ao processo de verificação de ciclos. Neste estudo de caso, após as alterações do modelo e da DSM, a matriz foi particionada novamente, incluindo a aplicação do método de identificação de ciclos, e o resultado obtido é exibido na Figura 21.

DSM calculada

	1	5	6	12	8	10	13	14	3	9	7	4	11	2	15
1 ItemVenda															
5 Fornecedor															
6 ItemDeSalario															
12 ItemDeLote															
8 Salario			1												
10 Cargo			1												
13 Produto	1													1	
14 Venda	1												1		
3 ItemCompra							1								
9 Funcionario					1	1									
7 Loja				1						1					
4 Compra		1							1	1					
11 Farmacia		1								1	1	1			
2 Lote				1											
15 Caixa								1		1					

Figura 21: Segundo particionamento com identificação de ciclos da DSM do sistema de farmácia

Note que, como o algoritmo **Quick Triangular Matrix** é guloso, nessa segunda iteração, ele não chegou ao resultado ótimo, deixando elementos que não pertencem a ciclos acima da diagonal, mas esse "problema" foi corrigido com o método de identificação de ciclos, que não pintou nenhuma das células de vermelho, demonstrando a inexistência de ciclos na modelagem atual e o bom resultado da ferramenta.

7 CONCLUSÕES

Este trabalho teve como ponto de partida um estudo sobre DSMs, seus conceitos e usos associados em diversas áreas do conhecimento. Em especial, DSMs foram estudadas em função de seu uso crescente na área de Engenharia de Software, onde DSMs podem ajudar os analistas a melhorar seus projetos a partir de uma melhor compreensão e gerência sobre as dependências entre artefatos e desenvolvedores. A partir desse estudo, alguns tópicos de interesse foram identificados e motivaram o desenvolvimento deste trabalho de conclusão de curso.

As principais contribuições desse trabalho são:

1. **Criação do algoritmo Quick Triangular Matrix** - O uso de DSMs para modelar sistemas de software com potencialmente milhares de classes e métodos, demanda algoritmos eficientes para manipular as matrizes. Conforme mostrou o comparativo entre os algoritmos apresentado no capítulo 5, para aplicações com muitos elementos, o tempo de espera por uma identificação de ciclos num algoritmo de complexidade exponencial pode tornar o uso de DSMs inviável. Para esses casos, o algoritmo criado pode ser uma alternativa interessante.

2. **Desenvolvimento da ferramenta Antares DSM** - A criação da ferramenta mostrou-se importante uma vez que o algoritmo Quick Triangular Matrix precisa ser incorporado em um ambiente de desenvolvimento para que analistas e programadores possam utilizá-lo, em conjunto com o desenho das DSMs. A ferramenta também é uma importante contribuição porque é livre e em língua portuguesa.

3. **Estudo sobre DSMs** - O estudo sobre as DSMs em si, é uma valiosa contribuição, por explicar os seus conceitos e como eles devem ser utilizados, com um estudo de caso simples, na área de Engenharia de Software, que pode motivar outras pessoas a explorar e se beneficiar de DSMs no dia-a-dia. Existem poucos estudos nessa área em língua portuguesa, principalmente no que diz respeito a algoritmos utilizados em DSMs, o que agrega maior valor ao trabalho, podendo este vir a se tornar uma referência em português na área para estudantes que não têm domínio sobre a língua inglesa.

7.1 DIFICULDADES ENCONTRADAS

No início do seu desenvolvimento, o planejamento do trabalho era para que ele fosse direcionado para aplicações na área de Engenharia de Software. No entanto, como as DSMs podem ser aplicadas a praticamente qualquer área, optou-se por tornar o trabalho mais genérico e abrangente, facilitando o uso de DSMs em outras áreas. Ainda assim, devido ao tempo curto e limite de páginas da monografia, não são mostrados exemplos de uso de DSMs em outras áreas. Espera-se que o leitor possa perceber, mesmo sem os exemplos, que as DSMs podem ser utilizadas em quaisquer modelagens ou projetos que possuam elementos e relações de dependências.

Outra questão refere-se à escolha da forma como seria feito o algoritmo. A maioria dos algoritmos existentes são de complexidade exponencial, então talvez não fosse interessante fazer um algoritmo equivalente aos que já existiam. Por isso optou-se por fazer um algoritmo "guloso". Após a criação do algoritmo, foram pensadas formas de melhorar a porcentagem de casos em que o algoritmo chegaria ao resultado ótimo, no entanto, essas alterações implicariam em mais passos ao algoritmo, que elevariam seu tempo de execução para cobrir uma pequena quantidade de casos. Escolher esse equilíbrio entre tempo e probabilidade de chegar ao resultado ótimo acabou sendo uma decisão um pouco complicada e difícil de ser tomada.

7.2 TRABALHOS FUTUROS

Este trabalho serve como ponto de partida para outros futuros, por exemplo:

1. Criar e validar algoritmos gulosos com outros objetivos para serem aplicados a DSMs, como por exemplo, o de agrupamento.
2. Implementar esses novos algoritmos na ferramenta **Antares DSM**.
3. Extensão da ferramenta para suprir outras necessidades de modelagens em DSMs, assim como módulos de integração com outras ferramentas, para montar a matriz de dependências de maneira mais automatizada.
4. Desenvolver novos estudos de casos em áreas como engenharia civil, engenharia mecânica, engenharia elétrica, administração de empresas, indústria, e tantas outras onde DSMs podem ser aplicadas.
5. Realizar mais experimentos, para obter dados mais precisos sobre os algoritmos.

7.3 CONSIDERAÇÕES FINAIS

A utilização de DSMs mostrou-se uma forma compacta e visual, de fácil compreensão e manipulação, que promove a identificação mais ágil de dependências entre elementos.

O algoritmo **Quick Triangular Matrix** (incorporado na ferramenta **Antares DSM**) mostrou-se bastante eficiente no que diz respeito a desempenho, podendo ser utilizado principalmente em modelagens com muitos elementos. No entanto, pelos estudos realizados observou-se que o algoritmo não chega ao resultado ótimo em cerca de 4% dos casos, de acordo com os experimentos, problema que pode ser contornado com a aplicação, em conjunto, do algoritmo de identificação de ciclos (também implementado na ferramenta **Antares DSM**).

Os estudos conduzidos se concentraram na melhoria de modelagens através da identificação de ciclos, porém, não são em todos os tipos de modelagens que os ciclos são indesejáveis. O analista deve antes observar a necessidade de remover ciclos em seu projeto. Para outras aplicações, existem outros algoritmos sobre os quais esse trabalho não trata, como o de agrupamento, que serve, dentre outras coisas, para definir submódulos ou equipes no projeto.

REFERÊNCIAS

BROWNING, T. R. Applying the design structure matrix to system decomposition and integration problems: A review and new directions. *IEEE Trans. Software Eng.*, n. 48, p. 292–306, 2001.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. *Introduction to Algorithms*. [S.l.]: MIT Press and McGraw-Hill, 1990.

DIESTEL, R. *GraphTheory*. [S.l.]: Springer, 2000.

DSMWEB. 2008. Último acesso em 19 de Junho de 2008. Disponível em: <http://www.dsmweb.org>.

GEBALA, D. A.; EPPINGER, S. D. Methods for analysing design procedures. In: *ASME Design Theory and Methodology Conference*. [S.l.]: DETC, 1991. p. 227–233.

HOARE, C. Quicksort. *Computer Journal*, n. Vol. 5, p. 10–15, 1962.

IEEE Standard Glossary of Software Engineering Terminology. New York: Institute of Electrical and Electronics Engineers, 1990.

MANZIONE, L.; B.MELHADO, S. Porque os projetos atrasam? uma análise crítica da ineficácia do planejamento de projetos adotada no mercado imobiliário de são paulo. In: *III Encontro de Tecnologia da Informação e Comunicação na Construção Civil*. Porto Alegre - Rio Grande do Sul: [s.n.], 2007.

MORAES, N. A. *Compreensão e Visualização de Projetos Orientados a Objetos com Matriz de Dependências*. 2007. Universidade Federal da Bahia.

MORI, T. et al. Task planning for product development by strategic schedulling of design reviews. In: *1999 ASME Design Engineering Technical Conferences*. Las Vegas NV: DETC, 1999.

OMG. 2008. Último acesso em 19 de Junho de 2008. Disponível em: <http://www.uml.org/>.

PIERONI, E.; NAVEIRO, R. M. A design structure matrix (dsm) aplicada ao projeto de navios. In: *Anais do V Congresso Brasileiro de Gestão e Desenvolvimento do Produto*. Curitiba - Paraná: [s.n.], 2005.

WIKIPÉDIA. 2008. Último acesso em 8 de Junho de 2008. Disponível em: <http://pt.wikipedia.org/wiki>.