

Universidade Estadual de Campinas  
Faculdade de Tecnologia

Marivaldo Felipe de Melo

Aprimoramento do algoritmo *PQR-Sort* para  
reordenação de matrizes binárias

Limeira, 2012



Universidade Estadual de Campinas

Faculdade de Tecnologia

Marivaldo Felipe de Melo

## Aprimoramento do algoritmo PQR-*Sort* para reordenação de matrizes binárias

Dissertação apresentada ao Curso de Mestrado em Tecnologia da Faculdade de Tecnologia da Universidade Estadual de Campinas, como requisito para a obtenção do título de Mestre em Tecnologia.

Área de Concentração: Tecnologia e Inovação

Orientador: Prof. Dr. Celmar Guimarães da Silva

Co-Orientador: Prof. Dr. Luiz Camolesi Júnior

Limeira, 2012

FICHA CATALOGRÁFICA ELABORADA POR SILVANA MOREIRA DA SILVA SOARES –  
CRB-8/3965  
BIBLIOTECA UNIFICADA FT/CTL  
UNICAMP

Melo, Marivaldo Felipe de, 1988-  
M491a Aprimoramento do algoritmo PQR-Sort para a  
reordenação de matrizes binárias / Marivaldo  
Felipe de Melo. – Limeira, SP : [s.n.],  
2012.

Orientador: Celmar Guimarães da Silva.  
Coorientador: Luiz Camolesi Júnior.  
Dissertação (mestrado) – Universidade Estadual de  
Campinas, Faculdade de Tecnologia.

1. Algoritmos de computador. 2. Matrizes.  
3. Visualização de informação. I. Silva, Celmar Guimarães da  
II. Camolesi Júnior, Luiz. III. Universidade Estadual  
de Campinas. Faculdade de Tecnologia. IV. Título.

Informações para Biblioteca Digital

Título em inglês: Improvement of PQR-Sort algorithm for binary matrices reordering

Palavras-chave em inglês (Keywords):

- 1- Computer algorithms
- 2- Matrices
- 3- Information visualization

Área de concentração: Tecnologia e Inovação

Titulação: Mestre em Tecnologia

Banca examinadora: Celmar Guimarães da Silva, André Franceschi de Angelis, Maria  
Cristina Ferreira de Oliveira

Data da Defesa: 26-06-2012

Programa de Pós-Graduação em Tecnologia

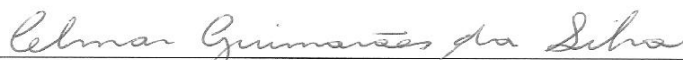


**DISSERTAÇÃO DE MESTRADO EM TECNOLOGIA**  
**ÁREA DE CONCENTRAÇÃO: TECNOLOGIA E INOVAÇÃO**

Aprimoramento do algoritmo PQR-Sort para reordenação de matrizes binárias

**Autor:** Marivaldo Felipe de Melo

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:



---

Presidente: Prof. Dr. Celmar Guimarães da Silva  
FT/UNICAMP



---

Prof. Dr. André Franceschi de Angelis  
FT/UNICAMP



---

Profa. Dra. Maria Cristina Ferreira de Oliveira  
ICMC/USP

## **Agradecimentos**

Ao Professor Celmar Guimarães da Silva, orientador do trabalho, por toda a orientação prestada.

Ao Professor João Meidanis, que gentilmente forneceu as classes para a construção de árvores PQR.

Aos meus pais, minhas irmãs e família pelo constante apoio apesar da distância geográfica.

## Resumo

Algoritmos de reordenação são importantes para análise de dados matriciais, pois encontram automaticamente permutações de linhas e colunas que agrupam valores semelhantes em uma matriz, visando facilitar a busca visual por padrões e tendências nos dados. Além disso, esses algoritmos tendem a reduzir a sobrecarga cognitiva do usuário, tendo em vista que, diferentemente das abordagens anteriores à seriação, o usuário não precisa permutar manualmente linhas e colunas para encontrar padrões.

Entre os algoritmos de reordenação de matrizes pesquisados, destaca-se o *PQR-Sort*, por sua natureza não-heurística e baixa complexidade assintótica de tempo de execução. Com base nesse algoritmo, este trabalho objetiva produzir versões aprimoradas do *PQR-Sort* visando melhorar a qualidade das matrizes por ele reordenadas (medida por funções de avaliação). Como principais resultados, foram criados dois novos algoritmos, *PQR-Sort with Sorted Restrictions* e *PQR-Sort + BC*, cujos resultados são melhores que os do algoritmo *PQR-Sort* de acordo com funções de avaliação de caráter local e global, respectivamente. O trabalho apresenta ainda um estudo de caso sobre a aplicação dos algoritmos propostos a um conjunto de dados real.

Palavras-Chave: *PQR-Sort*, reordenação de matrizes e Visualização de Informação

## Abstract

Reordering algorithms are important in matrix data analysis, because they automatically find row and column permutations that group similar data in a table, in order to ease finding patterns and trends in the data. Furthermore, these algorithms tend to reduce the user's cognitive overload, since, unlike previous reordering approaches, users don't need to swap rows and columns manually in order to find patterns.

Within the surveyed reordering algorithms, *PQR-Sort* stands out because of its non-heuristic nature and low asymptotic time complexity. Based on this algorithm, this work aims to produce enhanced versions of *PQR-Sort* in order to improve the quality of the reordered matrix (measured by evaluators). Among its main results is the creation of two new algorithms: *PQR-Sort* with sorted Restrictions and *PQR-Sort + BC*, which perform better than *PQR-Sort* according to local e global evaluators, respectively. This work also presents a study case about the proposed algorithms' application in a real dataset.

Keywords - *PQR-Sort*, matrix reordering, information visualization

## Sumário

Agradecimentos .....	iv
Resumo .....	v
Abstract.....	vi
Sumário.....	vii
Lista de Ilustrações .....	x
Lista de Quadros .....	xv
Lista de Fórmulas.....	xvi
Lista de Tabelas .....	xvii
1 Introdução.....	1
2 Reordenação de dados em matrizes e funções de avaliações .....	5
2.1 Introdução .....	5
2.2 Definições .....	6
2.2.1 Coeficientes de Similaridade e Dissimilaridade.....	6
2.2.2 Tipos de padrões em reordenações (globais e locais) .....	9
2.3 Algoritmos de Reordenação (Seriação) .....	10
2.3.1 PQR-Sort .....	10
2.3.2 2D Sort .....	18
2.3.3 Algoritmo de Sugiyama Modificado (Heurística Baricêntrica) .....	19
2.3.4 Algoritmo “Bond Energy” .....	22
2.3.5 Seriação pelo Problema do Caixeiro Viajante.....	23
2.3.6 Clusterização Hierárquica .....	24
2.3.7 Seriação Elíptica.....	25
2.3.8 Classificação de Métodos.....	28
2.4 Funções de avaliação de ordenação .....	29
2.4.1 Funções de Perda e Mérito.....	31

2.4.2 Anti-Robinson Loss Function (ARLS) .....	31
2.4.3 Minimal Span Loss Function (MSLF) .....	32
2.4.4 Critério de Inércia.....	33
2.4.5 Critério dos Quadrados Mínimos .....	33
2.4.6 Medida de Efetividade .....	33
2.4.7 Stress .....	34
2.5 Avaliação centrada no usuário .....	34
2.5.1 Experimento de Siirtola e Mäkinen (2005) .....	34
2.5.2 Experimento de Henry e Fekete (2006) .....	37
2.6 Ferramentas.....	39
2.6.1 MRA - Matrix Reordering Analyzer .....	40
2.7 Experimentos .....	42
2.8 Funções de Avaliação .....	43
2.9 Limitações da Ferramenta MRA.....	45
2.10 Considerações .....	48
3 Algoritmos desenvolvidos .....	49
3.1 Fases de algoritmos para melhorar o PQR- <i>Sort</i> .....	49
3.1.1 PQR- <i>Sort</i> + BC .....	50
3.1.2 PQR Smallest Restrictions .....	51
3.1.3 PQ- <i>Sort</i> .....	51
3.1.4 PQR- <i>Sort</i> with Sorted Restrictions .....	52
3.2 Conclusão.....	53
4 Ferramenta .....	55
4.1 Teste de Hipóteses .....	55
4.2 Sumarização de Relatórios.....	58
4.3 Considerações Finais .....	60

5 Testes.....	61
5.1 Algoritmos propostos.....	61
5.1.1 PQR-Sort + BC .....	62
5.1.2 PQR Smallest Restrictions .....	65
5.1.3 PQ-Sort.....	67
5.1.4 PQR-Sort with Sorted Restrictions .....	69
5.1.5 Resumo dos Resultados.....	71
5.1.6 Testes com base em tipos de padrões – Global e Local .....	72
5.1.7 Considerações sobre os testes com matrizes sintéticas .....	77
5.2 Conjunto de dados Íris .....	77
5.2.1 Classificação do Conjunto de Dados Íris .....	78
5.2.2 Reordenações do Conjunto de dados Íris .....	79
5.3 Considerações Finais .....	87
6 Conclusão .....	88
6.1 Trabalhos Futuros .....	90
Referências.....	91
Apêndice A - Versão completa dos relatórios de funções de avaliação .....	94
Anexo A – Versão Original do Conjunto de dados Íris .....	109
Anexo B – Distribuição T de Student.....	112

## Lista de Ilustrações

Figura 1 Composição de alimentos por 100 gramas de parte comestível: Centesimal, minerais, vitaminas e colesterol. É exibido nessa na figura apenas 1 das 34 páginas que compõem essa tabela. Figura extraída de Lima et al. (2006), p.30.....	1
Figura 2 Matriz binária exibindo a presença (marcada com a cor preta) de características em 16 cidades (A,B,C, ..., P). Figura extraída de Melo (2009), p.29). ....	4
Figura 3 Exemplo de uma matriz de dissimilaridade. Como em qualquer matriz de dissimilaridade, linhas e colunas contém o mesmo conjunto de objetos, letras de A a F. ....	7
Figura 4 Esquema da geração de uma matriz de similaridade das linhas de uma matriz de dados utilizando o coeficiente Jaccard.....	9
Figura 5 Em (a) uma matriz binária desordenada; em (b) a mesma matriz reordenada por um algoritmo que busca revelar padrões locais; e, em (c) a mesma matriz reordenada por um algoritmo que tenta revelar padrões globais. Nessa figura, células escuras indicam valores 1 e células brancas o valor 0. Figura adaptada de Liiv (2010).....	10
Figura 6 Em (a), uma matriz com a propriedade dos uns consecutivos. Em (b), o P1C resolvido. ....	11
Figura 8 Processo de construção de uma árvore PQR. Figura adaptada de Silva (2010), p.6 .	13
Figura 7 Exemplos de árvores PQR e permutações possíveis. Em (a), um nó P, em (b), um nó Q e em (c), um nó R. ....	12
Figura 9 Extração de restrições para linhas e colunas da matriz. Figura extraída de Silva (2010), p. 7. ....	14
Figura 10 Reordenação de uma matriz binária pelo PQR- <i>Sort</i> . Figura adaptada de Silva (2010), p.7. ....	14
Figura 11 Uma matriz binária 100x100 ordenada pelo PQR- <i>Sort</i> . Em (a) a matriz com uma ordenação aleatória e em (b) a matriz após a ordenação com o PQR- <i>Sort</i> .....	14
Figura 12 Refinamento de restrições com quatro passos; em (a) a situação inicial (matriz desordenada), em (b), (c), (d) e (e) os passos do refinamento.....	17
Figura 13 Reprodução da tabela da Figura 2 após aplicação do algoritmo 2D Sort. Figura extraída de Melo (2009), p.30. ....	19



Figura 14 Resultado ótimo para a reordenação de linhas e colunas da tabela da Figura 2. ....	19
Figura 15 Relação entre seriação e o problema da redução de cruzamentos de arestas em grafos bipartidos. Em (a), à esquerda grafo e à direita matriz de adjacência antes da reordenação; em (b), ambos após a reorganização para evitar cruzamentos. (Figura extraída de Mäkinen e Siirtola (2000), p. 464). .....	20
Figura 16 Ordenação da matriz da Figura 2 utilizando o algoritmo de Sugiyama Modificado.	21
Figura 17 Exemplo da Vizinhança 4-conectada. Marcado em vermelho os vizinhos da célula $i_{22}$ . .....	22
Figura 18 Esquema do Algoritmo BE para colunas da matriz. A Figura ilustra o posicionamento da coluna $c_3$ . .....	23
Figura 19 Em (A) uma matriz de dissimilaridade; em (B) um grafo representando essa matriz.	24
Figura 20 Esquema da clusterização hierárquica para reordenar linhas ou colunas de uma matriz. A partir de uma matriz de similaridade gera-se um dendrograma; e, partir do dendrograma é possível extrair boas permutações para linhas ou colunas. ....	25
Figura 21 Estruturas elípticas para algumas matrizes da sequência R. $\rho(n)$ indica o valor do <i>rank</i> . Figura extraída de Chen (2002), p. 2. ....	27
Figura 22 Esquema da reordenação de linhas através da seriação elíptica. Em (a) a estrutura elíptica de <i>rank</i> 2; em (b), um mapa de calor ordenado através das posições relativas da matriz. Cada quadrado sobre a elipse é um vetor de linhas da matriz $R^{(7)}$ . Figura adaptada de Chen (2002), p. 13. ....	28
Figura 23 Critério de linhas de uma matriz anti-Robinson aplicado a valores acima da diagonal principal. Os valores das células devem ser maiores a medida que se afastam da diagonal principal. ....	32
Figura 24 Interfaces dos protótipos de Siirtola e Mäkinen (2005) para reordenação de matrizes. Do lado direito a estrutura com seus valores – o preenchimento da célula é proporcional ao seu valor. Do lado esquerdo, uma matriz de correlação entre colunas. Figura extraída de Siirtola e Mäkinen (2005), p. 35. ....	36
Figura 25 Estrutura de <i>lattice</i> para os conjuntos {a,b,c,d} (a), {a,b} (b) e {a,b,c} (c). ....	37
Figura 26 Grupos de dados semelhantes identificados pelos usuários. Na esquerda, grupos em uma matriz ordenada alfabeticamente; no meio, matriz ordenada por clusterização hierárquica; e	

na direita, matriz ordenada pela a solução do PCV. Todas as tabelas contém os mesmos dados.	
Figura extraída de Henry e Fekete (2006), p. 4. ....	38
Figura 27 Esquema de uma comparação na ferramenta MRA (Silva, 2010). $F_1$ e $F_2$ indicam duas funções de avaliação. ....	41
Figura 28 Matrizes do tipo Rectnoise. Em (A) uma matriz sem ruído ( $d=0$ ) e em (B) uma um matriz com ruído. ....	43
Figura 29 Conversão de matrizes de dados em matrizes de similaridade. Cada matriz de dados gera duas matrizes de similaridade, uma para linhas e outra para colunas. Na figura, o coeficiente de similaridade empregado foi <i>Jaccard</i> ; e $F_1$ indica uma função de avaliação de matrizes de similaridade. ....	44
Figura 30 Esquema de um experimento após as modificações de Silva et al. (2011). A principal modificação em relação à versão anterior foi conversão de matrizes de dados em matrizes de similaridade (passo 3). Isso permitiu que os algoritmos de seriação fossem testados com funções de avaliação clássicas da área de seriação de matrizes. ....	45
Figura 31 Formatação de um relatório gerado pela ferramenta MRA. ....	47
Figura 32 Exemplo de um trecho de relatório gerado pela ferramenta MRA. ....	48
Figura 33 Esquema do algoritmo do PQR- <i>Sort</i> para uma dimensão (linha ou coluna) exibindo as entradas: domínio e restrições e as saídas: Permutações válidas ou Permutações com nós R. As três regiões da figura representam as categorias de abordagens para melhorar o PQR- <i>Sort</i> : Formação de Restrições, Construção da Árvore e Processamento das Permutações. ....	50
Figura 34 Trecho de um relatório de funções de avaliação gerado pela ferramenta MRA. A ausência de um melhor ou pior algoritmo é denotada por ‘?’ (caso em que a hipótese nula é aceita). ....	57
Figura 35 Relatório de funções de avaliação com detalhes dos testes de hipóteses. Empate entre algoritmos são denotados por ‘?’ ....	59
Figura 36 Relatório de funções de avaliação resumido. ....	59
Figura 37 Relatório de tempo de execução. ....	60
Figura 38 Comparação baseada em funções de avaliação ( <i>evaluators</i> ) entre PQR- <i>Sort</i> e PQR- <i>Sort</i> + BC para matrizes 10x10. Os casos em que o PQR- <i>Sort</i> + BC foi considerado melhor estão grifados em verde. ....	64

Figura 39 Comparação baseada em funções de avaliação ( <i>evaluators</i> ) entre PQR- <i>Sort</i> e PQR- <i>Sort</i> + BC para matrizes 100x100. Os casos em que o PQR- <i>Sort</i> + BC foi considerado melhor estão grifados em verde. ....	64
Figura 40 Relatório de tempo de execução dos algoritmos PQR- <i>Sort</i> e PQR- <i>Sort</i> + BC em matrizes 10x10.....	65
Figura 41 Relatório de tempo de execução dos algoritmos PQR- <i>Sort</i> e PQR- <i>Sort</i> + BC em matrizes 100x100.....	65
Figura 42 Comparação entre PQR- <i>Sort</i> e PQR- <i>Smallest Restrictions</i> (usando valor limite 0.5) com base em funções de avaliação em matrizes 10x10.....	66
Figura 43 Comparação entre PQR- <i>Sort</i> e PQR- <i>Smallest Restrictions</i> (usando valor limite 0.5) com base em funções de avaliação em matrizes 100x100. ....	66
Figura 44 Análise de tempo comparativa entre PQR- <i>Sort</i> e PQR- <i>Smallest Restrictions</i> (0.5) em matrizes 10x10.....	67
Figura 45 Análise de tempo comparativa entre PQR- <i>Sort</i> e PQR- <i>Smallest Restrictions</i> (0.5) em matrizes 100x100.....	67
Figura 46 Comparação entre PQR- <i>Sort</i> e PQ- <i>Sort</i> com base em funções de avaliação em matrizes 10x10. ....	68
Figura 47 Comparação entre PQR- <i>Sort</i> e PQ- <i>Sort</i> com base em funções de avaliação em matrizes 100x100. (A notação PQ Sort (1.0;1.0) indica que o algoritmo foi executado com pesos iguais para restrições menores e repetidas).....	68
Figura 48 Relatório de tempo de execução para matrizes 10x10.....	69
Figura 49 Relatório de tempo de execução para matrizes 100x100.....	69
Figura 50 Comparação entre PQR- <i>Sort</i> e PQR- <i>Sort with Sorted Restrictions</i> com base em funções de avaliação em matrizes 10x10.....	70
Figura 51 Comparação entre PQR- <i>Sort</i> e PQR- <i>Sort with Sorted Restrictions</i> com base em funções de avaliação em matrizes 100x100. O algoritmo PQR- <i>Sort with Sorted Restrictions</i> é referenciado na figura como PQR- <i>Sorted Restrictions</i> . ....	70
Figura 52 Relatório de tempo de execução para matrizes 10x10.....	71
Figura 53 Relatório de tempo de execução para matrizes 100x100.....	71

Figura 54 Comparação entre <i>PQR-Sort</i> , <i>PQR-Sort with Sorted Restrictions</i> e <i>PQ-Sort</i> com base em funções de avaliação em matrizes 10x10. O algoritmo <i>PQR-Sort with Sorted Restrictions</i> é referenciado na figura como <i>PQR-Sorted Restrictions</i> . .....	73
Figura 55 Comparação <i>PQR-Sort</i> , <i>PQR-Sort with Sorted Restrictions</i> e <i>PQ-Sort</i> com base em funções de avaliação em matrizes 100x100. ....	73
Figura 56 Análise de tempo comparativa entre <i>PQR-Sort</i> , <i>PQR-Sort with Sorted Restrictions</i> e <i>PQ-Sort</i> em matrizes 10x10.....	74
Figura 57 Análise de tempo comparativa entre <i>PQR-Sort</i> , <i>PQR-Sort with Sorted Restrictions</i> e <i>PQ-Sort</i> em matrizes 100x100.....	74
Figura 58 Comparação entre <i>PQR-Sort + BC</i> , <i>PQR-Sort with Sorted Restrictions</i> e Sugiyama com base em funções de avaliação em matrizes 10x10.....	75
Figura 59 Comparação entre <i>PQR-Sort + BC</i> , <i>PQR-Sort with Sorted Restrictions</i> e Sugiyama com base em funções de avaliação em matrizes 100x100.....	75
Figura 60 Análise de tempo comparativa <i>PQR-Sort+BC</i> , <i>PQR-Sort with Sorted Restrictions</i> e Sugiyama em matrizes 10x10.....	76
Figura 61 Análise de tempo comparativa <i>PQR-Sort+BC</i> , <i>PQR-Sort with Sorted Restrictions</i> e algoritmo de Sugiyama modificado (Heurística baricêntrica) em matrizes 100x100. ....	76
Figura 62 Reordenação aleatória para o íris dataset. ....	82
Figura 63 Íris dataset reordenado com o algoritmo <i>PQR-Sort</i> . ....	84
Figura 64 Íris dataset reordenado com o algoritmo <i>PQR-Sort with Sorted Restrictions</i> .....	84
Figura 65 Íris dataset reordenado com o algoritmo <i>PQR-Sort + BC</i> .....	85
Figura 66 Íris dataset reordenado com o algoritmo de Sugiyama Modificado .....	86
Figura 67 – Tabela da distribuição t de student. Na Figura, a primeira coluna, GL, indica o grau de liberdade. E, a primeira linha, C, indica o nível de confiança. Tabela extraída de Magalhaes & Lima (2002). ....	112

## **Lista de Quadros**

Quadro 1 Contadores utilizados no cálculo de Jaccard e Simple Matching .....	8
Quadro 2 Comparação entre algoritmos de reordenação de matrizes . .....	29
Quadro 3 Classificação dos atributos do íris dataset. ....	79

## Lista de Fórmulas

Fórmula 1 Fórmula para calcular o coeficiente Jaccard entre dois objetos.....	8
Fórmula 2 Fórmula para calcular o coeficiente Simple Matching entre dois objetos. ...	8
Fórmula 3 Correlação de Pearson.....	26

## **Lista de Tabelas**

Tabela 1 Vitórias, empates e derrotas para qualidade da reordenação. ....	71
Tabela 2 Vitórias, empates e derrotas para tempo de execução. ....	72





# 1 Introdução

Representações tabulares são uma das formas mais simples e comuns de apresentar dados. Pesquisas científicas frequentemente usam tabelas para classificar e comparar dados extraídos de experimentos. Apesar dessa simplicidade, essa estrutura possui limitações relativas ao entendimento das informações expressas visualmente. Uma tabela de dados com uma grande quantidade de informações pode dificultar a consulta visual de dados e a observação de padrões, devido à grande carga cognitiva necessária para essa análise. A Figura 1 ilustra essa situação. A tabela da Figura 1 foi extraída de Lima et al. (2006) e foi um dos resultados do projeto TACO (Tabela Brasileira de Composição de Alimentos), que busca estudar a composição dos principais alimentos consumidos no Brasil. A Figura 1 ilustra apenas umas das 34 páginas que possui essa tabela. Em uma tabela com esse tamanho é difícil extrair padrões visualmente; por exemplo, apenas com essa tabela seria trabalhoso responder a pergunta: “Quais são os alimentos que possuem baixa quantidade de lipídeos e alta quantidade de carboidratos?”.

**Tabela 1. Composição de alimentos por 100 gramas de parte comestível: Centesimal, minerais, vitaminas e colesterol**

Número do Alimento	Descrição dos alimentos	Umidade (%)	Energia		Proteína (g)	Lipídeos (g)	Colesterol (mg)	Carbo-Idrato (g)	Fibra Alimentar (g)	Cinzas (g)	Cálcio (mg)
			(kcal)	(kJ)							
156	Banana, da terra, crua	63,9	128	536	1,4	0,2	NA	33,7	1,5	0,8	*
157	Banana, figo, crua	70,1	105	440	1,1	0,1	NA	27,8	2,8	0,8	6
158	Banana, maçã, crua	75,2	87	363	1,8	0,1	NA	22,3	2,6	0,6	3
159	Banana, nanica, crua	73,8	92	383	1,4	0,1	NA	23,8	1,9	0,8	3
160	Banana, ouro, crua	68,2	112	470	1,5	0,2	NA	29,3	2,0	0,8	3
161	Banana, pacova, crua	77,7	78	326	1,2	0,1	NA	20,3	2,0	0,7	5
162	Banana, prata, crua	71,9	98	411	1,3	0,1	NA	26,0	2,0	0,8	8
163	Cacau, cru	79,2	74	311	1,0	0,1	NA	19,4	2,2	0,3	12
164	Cajá-Manga, cru	86,9	46	191	1,3	Tr	NA	11,4	2,6	0,4	13
165	Caju, cru	88,1	43	180	1,0	0,3	NA	10,3	1,7	0,3	1
166	Caju, polpa, congelada	89,8	37	153	0,5	0,2	NA	9,4	0,8	0,2	1
167	Caju, suco concentrado, envasado	88,4	45	189	0,4	0,2	NA	10,7	0,6	0,3	1
168	Caqui, chocolate, cru	79,7	71	299	0,4	0,1	NA	19,3	6,5	0,5	18
169	Carambola, crua	87,1	46	191	0,9	0,2	NA	11,5	2,0	0,4	5
170	Cinguela, crua	78,7	76	316	1,4	0,4	NA	18,9	3,9	0,7	27
171	Cupuaçu, cru	86,2	49	207	1,2	1,0	NA	10,4	3,1	1,2	13
172	Cupuaçu, polpa, congelada	86,6	49	204	0,8	0,6	NA	11,4	1,6	0,6	5
173	Figo, cru	88,2	41	173	1,0	0,2	NA	10,2	1,8	0,4	27
174	Figo, enlatado, em calda	48,8	184	771	0,6	0,2	NA	50,3	2,0	0,2	33
175	Fruta-pão, crua	80,9	67	281	1,1	0,2	NA	17,2	5,5	0,7	34
176	Goiaba, branca	85,7	52	216	0,9	0,5	NA	12,4	6,3	0,5	*
177	Goiaba, doce em pasta	24,8	269	1125	0,6	Tr	NA	74,1	3,7	0,5	10
178	Goiaba, vermelha	85,0	54	227	1,1	0,4	NA	13,0	6,2	0,5	*
179	Graviola, crua	82,2	62	258	0,8	0,2	NA	15,8	1,9	1,0	40
180	Graviola, polpa, congelada	89,2	38	160	0,6	0,1	NA	9,8	1,2	0,4	6
181	Jabuticaba, crua	83,6	58	243	0,6	0,1	NA	15,3	2,3	0,4	8
182	Jaca, crua	75,1	88	368	1,4	0,3	NA	22,5	2,4	0,8	11
183	Jambo, cru	92,1	27	113	0,9	0,1	NA	6,5	5,1	0,5	14

Figura 1 Composição de alimentos por 100 gramas de parte comestível: Centesimal, minerais, vitaminas e colesterol. É exibido nessa na figura apenas 1 das 34 páginas que compõem essa tabela. Figura extraída de Lima et al. (2006), p.30.

Em Visualização de Informação, umas das maneiras de lidar com esse problema é utilizar interatividade, aplicando o conceito de Matrizes Reordenáveis de Bertin (BERTIN, 2001).

Segundo Siirtola e Mäkinen (2005), a matriz reordenável de Bertin é uma visualização simples para explorar dados tabulares; a idéia básica é transformar dados multidimensionais em uma representação 2D interativa. Nesse tipo de matriz, as linhas e colunas podem ser permutadas e essa interatividade possibilita a busca por padrões. Além disso, as linhas podem ser ordenadas em ordem decrescente ou crescente.

A representação de dados em uma matriz de Bertin tem a vantagem de exibir a visão geral dos dados, sendo que em uma única matriz é possível exibir uma grande quantidade de dados. Por exemplo, as 34 páginas que compõem a tabela da Figura 1 poderiam ser exibidas em uma única matriz de Bertin.

Contudo, apesar do fato de a matriz de Bertin ser interativa, por permitir reordenação de linhas e colunas, certas análises não são triviais – especialmente em matrizes grandes, pois em uma matriz de Bertin existe um número fatorial de possíveis ordenações:

$$\text{número de ordenações possíveis} = (\text{número de Linhas})! \times (\text{número de Colunas})!$$

Mesmo em matrizes pequenas, o número de possíveis ordenações é grande. Na matriz da Figura 2 - que apresenta a presença de 9 características nas cidades A,B,C, ..., P – existem 9 linhas e 16 colunas; logo, existem aproximadamente  $7,59 \times 10^{18}$  possibilidades. Nessa situação, é difícil para os usuários reordenar manualmente linhas e colunas até encontrar padrões. Como alternativa, métodos de reordenação automática (também conhecidos como Seriação) foram desenvolvidos visando produzir permutações que agrupem valores na matriz automaticamente, sendo possível, se necessário, o usuário alterar posteriormente linhas e colunas manualmente para encontrar novos padrões. Conforme citam Siirtola e Mäkinen (2005), em uma matriz reordenável existem fases manuais (usuário) e automáticas (algoritmos de ordenação), e essas fases auxiliam uma a outra; dessa forma, o conhecimento humano guia o sistema, deixando as partes de processamento bruto para os algoritmos de seriação.

Entre os algoritmos de seriação, o PQR-*Sort* é um algoritmo que produz boas reordenações, especialmente reordenações locais. Em Silva et al. (2011), os algoritmos de seriação PQR-*Sort*, 2D-*Sort* e algoritmo de Sugiyama modificado foram comparados segundo tempo de execução e qualidade das reordenações (com o uso de funções de avaliação). Segundo consta em Silva et al. (2011), o PQR-*Sort* quando comparado com o 2D-*Sort* e Sugiyama-*Sort* foi superior em 100%

dos testes com matrizes aleatórias de dimensões 100x100 e 1000x1000 usando a função de avaliação *Minimal Span* (função que mede a presença de características globais na ordenação).

Apesar das vantagens, conforme os testes de Silva et al. (2011), o *PQR-Sort* apresentou os piores resultados quando comparado com *2D-Sort* e *Sugiyama-Sort* utilizando a função de avaliação *Anti-Robinson Loss Function* (que identifica boas ordenações globais).

Em relação ao tempo de execução, segundo Silva et al. (2011), para matrizes quadradas o algoritmo de Sugiyama modificado e *2D-Sort* possuem complexidade tempo  $O(n^2t)$ , em que  $n$  é o número de linhas (ou colunas) e  $t$  é o número de iterações; tratam-se de algoritmos heurísticos, nos quais, em geral, não se sabe antecipadamente quantas iterações serão executadas. Já o *PQR-Sort* possui complexidade de tempo  $O(n^2 \alpha(n^2))$ <sup>vide nota de rodapé<sup>1</sup></sup>. A título de comparação entre um algoritmo heurístico e o *PQR-Sort*, para reordenar uma matriz binária 1000x1000 aleatória de densidade 0,5, o algoritmo de Sugiyama modificado gasta 53 segundos, enquanto que, o *PQR-Sort* gastou 0,1 segundos (Silva et al. 2011).

Portanto, devido ao potencial do *PQR-Sort*, evidenciado por Silva et al. (2011), o objetivo do trabalho é provar a hipótese de que é possível melhorar o algoritmo *PQR-Sort*, com vistas aos resultados gerados por funções de avaliação de caráter global e local.

Para atender a esse objetivo, as novas versões produzidas do algoritmo *PQR-Sort* precisaram ser comparadas com outros algoritmos da área de seriação, com o objetivo de verificar se as versões produzidas geram boas reordenações. Essa tarefa envolve muitas matrizes de diferentes dimensões, por isso, é importante automatizá-la. Neste trabalho, para prover tal automatização, foi empregada a Ferramenta MRA (detalhada na Seção 2.6).

---

<sup>1</sup>  $\alpha$  é o inverso da função de Ackermann (CORMEN, 1990), é uma função de comportamento quase linear.



**Figura 2** Matriz binária exibindo a presença (marcada com a cor preta) de características em 16 cidades (A,B,C, ..., P).  
 Figura extraída de Melo (2009), p.29).

Além disso, neste trabalho, com o auxílio da ferramenta MRA (*Matrix Reordering Analyzer*), foram realizados experimentos com conjuntos de dados reais. Em nenhum dos trabalhos com o algoritmo PQR-*Sort* foram realizados até o momento experimentos desse tipo. O conjunto de dados real escolhido para testes foi o Íris (FISCHER, 1950), banco de dados amplamente conhecido na área de mineração de dados.

O resultado dos testes com a Ferramenta MRA com base em matrizes sintéticas e no conjunto de dados Íris mostrou que um dos algoritmos produzidos, o PQR-*Sort with Sorted Restrictions* gerou reordenações melhores que as produzidas pelo PQR-*Sort*.

O restante do texto está estruturado da seguinte forma:

- O **Capítulo 2** apresenta a revisão bibliográfica sobre algoritmos de seriação, funções de avaliação e ferramentas relacionadas à seriação de matrizes.
- O **Capítulo 3** descreve os algoritmos propostos para melhorar o PQR-*Sort*.
- O **Capítulo 4** apresenta as melhorias produzidas na Ferramenta MRA para possibilitar a comparação do PQR-*Sort* com os algoritmos produzidos.
- O **Capítulo 5** une os resultados dos Capítulos 3 e 4 e analisa os algoritmos produzidos com base em funções de avaliação e tempo de execução utilizando a ferramenta MRA.
- O **Capítulo 6** apresenta a conclusão do trabalho.

## 2 Reordenação de dados em matrizes e funções de avaliações

### 2.1 Introdução

Conforme apresentado no Capítulo 1, a análise de estruturas tabulares é uma tarefa que envolve uma grande sobrecarga cognitiva. Esse problema é potencializado em tabelas grandes, como a da Figura 1; é ainda mais difícil para o ser humano extrair padrões relevantes em tabelas que ocupam muitas páginas.

Uma das maneiras de lidar com a sobrecarga é utilizando matrizes de Bertin, pois, os dados são convertidos em cores para facilitar o processamento visual e linhas e colunas podem ser permutadas para agrupar cores semelhantes - segundo as Leis Gestalt (MAZZA,2009) marcas agrupadas são mais facilmente entendidos pela visão humana. Contudo, uma matriz de Bertin tem um número fatorial de possíveis ordenações de linhas e colunas. Encontrar manualmente permutações que agrupam linhas e colunas semelhantes é uma tarefa que aumenta a sobrecarga cognitiva e tira o usuário do seu foco principal: obter *insights* sobre os dados.

Assim, durante uma análise de dados em matrizes é importante prover uma reordenação automática, ou seja, uma seriação. Segundo Hahsler et al. (2010), seriação é o problema de organizar objetos (linhas e colunas) em uma ordem linear visando revelar informações estruturais.

Nesse trabalho, serão abordados os algoritmos de seriação: PQR-*Sort* (Seção 2.3.1), 2D *Sort* (Seção 2.3.2), Heurística Baricêntrica (Algoritmo de Sugiyama) (Seção 2.3.3), *Bond Energy* (Seção 2.3.4), Ordenação pelo Problema do Caixeiro Viajante (Seção 2.3.5), Clusterização Hierárquica (Seção 2.3.6) e Seriação Elíptica (Seção 2.3.7).

Uma preocupação posterior à ordenação de dados em matrizes é a avaliação de uma ordenação, ou seja, a aplicação de funções que calculam a qualidade de uma reordenação. Nesse trabalho foram abordadas as funções de avaliação: Medida de Efetividade (Seção 2.4.6), Medida de Inércia (Seção 2.4.4), *Anti-Robinson Loss Function* (Seção 2.4.2), *Minimal Span Loss Function* (Seção 2.4.3) e Stress (Seção 2.4.7).

Para automatizar o cálculo dos resultados de funções de avaliação pode-se usar ferramentas específicas; a Seção 2.6 trata de uma delas, a ferramenta MRA, que foi empregada para realizar os testes com funções de avaliação neste trabalho.

Além de algoritmos de ordenação e funções de avaliação, este capítulo descreve resultados de estudos centrados no usuário encontrados na literatura, com o objetivo de verificar se os algoritmos de reordenação e as funções de avaliação produzem bons resultados para usuários. A Seção 2.5.1 descreve os experimentos e resultados de Siirtola e Mäkinen (2005) e a Seção 2.5.2 descreve os resultados de Henry e Fekete (2006).

## 2.2 Definições

No decorrer deste trabalho será preciso ter conhecimento básico do conceito de matrizes de similaridade e dissimilaridade (ou distância) e sobre tipos de padrões em reordenações de matrizes (globais e locais), apresentados a seguir.

### 2.2.1 Coeficientes de Similaridade e Dissimilaridade

Conforme Choe et al. (2010), matrizes de similaridade e dissimilaridade têm muitas aplicações, como, por exemplo, em Biologia, Taxonomia, Segmentação de imagens, Arqueologia, Química etc.

Para este trabalho é importante discutir matrizes de similaridade e dissimilaridade porque algoritmos de seriação podem ser executados em **matrizes de similaridade**, em **matrizes de dissimilaridade** ou em **matrizes de dados**. Por exemplo, o algoritmo PQR-*Sort* é executado em uma matriz de dados; já o algoritmo de Clusterização Hierárquica pode ser executado em matrizes de similaridade ou dissimilaridade.

**Matrizes de dados** são matrizes com casos e variáveis representado objetos diferentes. Por isso, essas matrizes não são simétricas. A matriz da Figura 2 é um exemplo de matriz de dados; suas colunas representam um tipo de objeto (as cidades), e, suas linhas representam objetos de outro tipo (as características presentes ou ausentes nas cidades). Neste trabalho, matrizes de dados serão denotadas por  $M = \{m_{ij}\}$ .

**Matrizes de dissimilaridade (ou distância)** são matrizes simétricas que medem as diferenças entre elementos de um único conjunto de objetos (nessas matrizes, linhas e colunas representam o mesmo conjunto de objetos). Quanto maior um valor em uma matriz de dissimilaridade, mais diferentes são os elementos. Por exemplo, a Figura 3 exibe uma matriz de distância; o maior elemento dessa matriz é o valor 3,52 (Coluna A e Linha F); portanto, os dois elementos mais diferentes nessa matriz são A e F. A diagonal principal é sempre zero, pois não existe dissimilaridade quando se compara um elemento com ele mesmo. Neste trabalho, matrizes de dissimilaridade serão denotadas por  $D = \{d_{ij}\}$ .

$$D = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{pmatrix} 0,00 & -- & -- & -- & -- & -- \\ 0,95 & 0,00 & -- & -- & -- & -- \\ 1,99 & 1,05 & 0,00 & -- & -- & -- \\ 3,00 & 2,08 & 1,09 & 0,00 & -- & -- \\ 1,12 & 0,95 & 1,54 & 2,29 & 0,00 & -- \\ 3,52 & 2,60 & 1,60 & 0,52 & 2,77 & 0,00 \end{pmatrix} \end{matrix}$$

Figura 3 Exemplo de uma matriz de dissimilaridade. Como em qualquer matriz de dissimilaridade, linhas e colunas contém o mesmo conjunto de objetos, letras de A a F<sup>2</sup>.

**Matrizes de similaridade (ou proximidade)** são semelhantes às matrizes de distância. Ambas representam os mesmos objetos nas linhas e colunas, e, conseqüentemente ambas são simétricas. A diferença principal entre matrizes de similaridade e de distância é que as primeiras medem a semelhança entre objetos; ou seja, um valor alto em uma matriz de similaridade indica elementos semelhantes. Matrizes de similaridade serão denotadas neste trabalho por  $S = \{s_{ij}\}$ .

Para calcular matrizes de similaridade e dissimilaridade, basta utilizar os **coeficientes de dissimilaridade ou similaridade**. Neste trabalho, serão utilizados apenas dois coeficientes de similaridade: Jaccard e Simple Matching.

### 2.2.1.1 Jaccard e Simple Matching

Neste trabalho, os coeficientes Jaccard e Simple Matching serão empregados para gerar matrizes de similaridade a partir de matrizes de dados. Conforme será detalhado na Seção 2.4,

<sup>2</sup> Elementos acima da diagonal principal foram omitidos, pois, se trata de uma matriz simétrica.

algumas das funções empregadas para avaliar uma reordenação produzida por um algoritmo de seriação são aplicáveis apenas a matrizes de similaridade. Além disso, alguns dos algoritmos de reordenação de matrizes empregados neste trabalho, como *PQR-Sort* e Algoritmo de Sugiyama modificado, produzem matrizes de dados. Portanto, para aplicar funções que atuam em matrizes de similaridade em algoritmos que produzem matrizes de dados, é preciso calcular matrizes de similaridade a partir de matrizes de dados.

Para calcular uma matriz de similaridade a partir de uma matriz de dados, calcula-se a similaridade entre cada par de linhas da matriz de dados, gerando então a matriz de similaridade para as linhas; e calcula-se a similaridade entre cada par de colunas da matriz de dados, gerando assim a matriz de similaridade para as colunas.

No caso de Jaccard e Simple Matching, a similaridade entre dois objetos (entre duas linhas ou duas colunas) é calculada utilizando as Fórmulas 1 e 2, respectivamente. Nessas fórmulas  $a$ ,  $b$ ,  $c$ , e  $d$  representam contadores do número de casos em que as combinações de 0s e 1s exibidas no Quadro 1 ocorrem. O uso do Quadro 1 é exemplificado na Figura 4, em que uma matriz de dados é convertida em uma matriz de similaridade para linhas.

$$Jaccard(\text{Objeto } x, \text{Objeto } y) = \frac{a}{a + b + c}$$

**Fórmula 1** Fórmula para calcular o coeficiente Jaccard entre dois objetos.

$$Simple\ Matching(\text{Objeto } x, \text{Objeto } y) = \frac{a + d}{a + b + c + d}$$

**Fórmula 2** Fórmula para calcular o coeficiente Simple Matching entre dois objetos.

**Quadro 1** Contadores utilizados no cálculo de Jaccard e Simple Matching.

Contador	Objeto X	Objeto Y
a	1	1
b	1	0
c	0	1
d	0	0



A Figura 4 está dividida em três partes: Matriz de dados, Cálculo de similaridades entre objetos e Matriz de Similaridade para linhas. A partir da matriz de dados são calculadas as similaridades entre cada par de linhas, através da contagem dos casos a, b, c, e d e posterior aplicação da fórmula de Jaccard. Com os valores de Jaccard para cada par de linhas, constrói-se a matriz de similaridade para linhas.

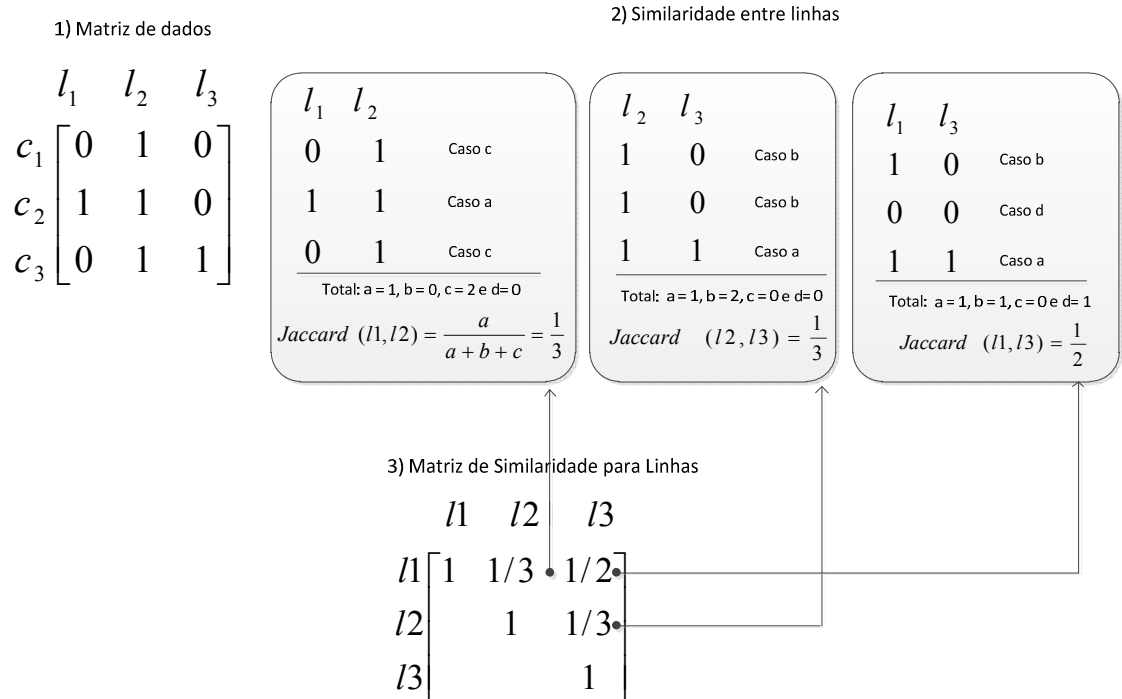


Figura 4 Esquema da geração de uma matriz de similaridade das linhas de uma matriz de dados utilizando o coeficiente Jaccard.

### 2.2.2 Tipos de padrões em reordenações (globais e locais)

Algoritmos de seriação podem produzir ao menos dois tipos de padrões ao reordenar matrizes: locais e globais. Os padrões globais expandem-se por toda a matriz. Por sua vez, os padrões locais podem ser observados apenas em algumas partes da matriz. A Figura 5 (a) ilustra uma matriz desordenada, e, as Figuras 5 (b) e (c) ilustram essa matriz reordenada por algoritmos que buscam revelar padrões locais e globais, respectivamente. Na matriz da Figura 5 (b) formaram-se diversos padrões (grupos de valores escuros), e na Figura 5 (c) existe um único e grande padrão de valores próximos à diagonal principal da matriz.

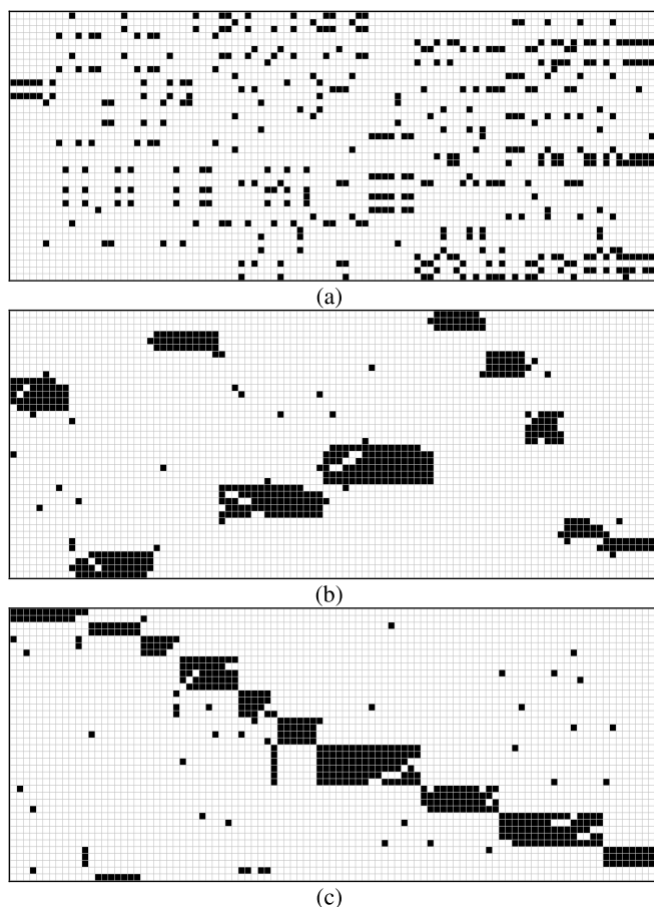


Figura 5 Em (a) uma matriz binária desordenada; em (b) a mesma matriz reordenada por um algoritmo que busca revelar padrões locais; e, em (c) a mesma matriz reordenada por um algoritmo que tenta revelar padrões globais. Nessa figura, células escuras indicam valores 1 e células brancas o valor 0. Figura adaptada de Liiv (2010).

## 2.3 Algoritmos de Reordenação (Seriação)

Esta seção apresenta alguns dos algoritmos de reordenação de dados em matrizes encontrados na bibliografia. Serão abordados os algoritmos *PQR-Sort* (Seção 2.3.1), *2D-Sort* (Seção 2.3.2), Algoritmo de Sugiyama modificado (Seção 2.3.3), Algoritmo “*Bond Energy*” (Seção 2.3.4), Seriação pelo problema do Caixeiro Viajante (Seção 2.3.5), Clusterização Hierárquica (Seção 2.3.6) e Seriação Elíptica (2.3.7).

### 2.3.1 PQR-Sort

*PQR-Sort* é um algoritmo de reordenação de dados em matrizes baseado em uma estrutura de dados chamada árvore PQR (TELLES e MEIDANIS, 2005). Essa estrutura de dados tem

como objetivo resolver o problema dos uns consecutivos (P1C) (MEIDANIS, 1998), ou seja, visa unir todos os valores não-nulos em todas as linhas e colunas de uma matriz binária. Essa característica faz com que uma árvore PQR seja uma estrutura com potencial para auxiliar a seriação de matrizes. Dois trabalhos exploraram esse potencial. Em Melo (2009) foram propostas maneiras de se utilizar árvores PQR para reordenar dados binários e não-binários. Já Silva (2010) fez um estudo comparativo entre o tempo de execução dos algoritmos de seriação *2D Sort*, Heurística Baricêntrica e *PQR-Sort*. As próximas seções apresentam o problema dos uns consecutivos e as árvores PQR.

### 2.3.1.1 Problema dos uns consecutivos

No Problema dos Uns Consecutivos (P1C) busca-se encontrar permutações de linhas e colunas de uma matriz binária que façam com que todos os valores não-nulos em todas as linhas e colunas estejam unidos. Diz-se que uma matriz que pode ser reorganizada dessa forma tem a propriedade dos uns consecutivos. A Figura 6 ilustra uma matriz com essa propriedade: em (a) a matriz original, e em (b) a solução do P1C, ou seja, uns consecutivos nas linhas e colunas. A ordem de linhas e colunas que resolve o problema dos uns consecutivos é chamada de **permutação válida**; portanto, as permutações “*fgh*” (para colunas) e “*cdaeb*” (para linhas) são consideradas permutações válidas.

$$\begin{array}{ccc}
 & f & g & h \\
 a & \left[ \begin{array}{ccc} 1 & 1 & 0 \end{array} \right] \\
 b & \left[ \begin{array}{ccc} 0 & 1 & 0 \end{array} \right] \\
 c & \left[ \begin{array}{ccc} 1 & 0 & 0 \end{array} \right] \\
 d & \left[ \begin{array}{ccc} 1 & 0 & 0 \end{array} \right] \\
 e & \left[ \begin{array}{ccc} 0 & 1 & 0 \end{array} \right] \\
 & (a) & & 
 \end{array}
 \qquad
 \begin{array}{ccc}
 & f & g & h \\
 c & \left[ \begin{array}{ccc} 1 & 0 & 0 \end{array} \right] \\
 d & \left[ \begin{array}{ccc} 1 & 0 & 0 \end{array} \right] \\
 a & \left[ \begin{array}{ccc} 1 & 1 & 0 \end{array} \right] \\
 e & \left[ \begin{array}{ccc} 0 & 1 & 0 \end{array} \right] \\
 b & \left[ \begin{array}{ccc} 0 & 1 & 0 \end{array} \right] \\
 & (b) & & 
 \end{array}$$

Figura 6 Em (a), uma matriz com a propriedade dos uns consecutivos. Em (b), o P1C resolvido.

### 2.3.1.2 Árvores PQR

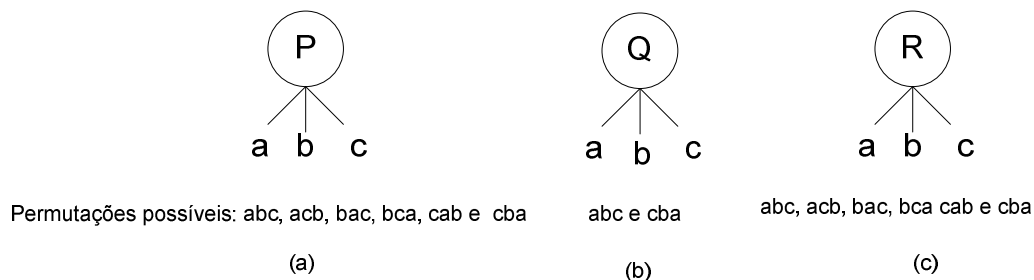
Uma árvore PQR é uma árvore capaz de representar permutações de um conjunto de objetos  $U$  (ou Universo) que respeitam um conjunto de restrições  $R$  (objetos que devem permanecer unidos). Por exemplo, considerando o domínio de objetos  $U=\{a,b,c\}$  e o conjunto de

restrições  $R=\{ac\}$ ; gerando uma árvore PQR com  $U$  e  $R$ , as permutações geradas pela árvore seriam: “ $acb$ ”, “ $bac$ ”, “ $cab$ ” e “ $bca$ ”.

Segundo Meidanis (1998), uma árvore PQR é uma árvore enraizada, com quatro tipos de nós: P, Q, R e folhas. Os nós internos (P, Q e R) possuem regras para representar permutações de seus filhos; os filhos de nós P e R podem ser permutados entre si; e, os filhos de nós Q apenas podem ser invertidos (lidos da esquerda para direita ou o contrário). A Figura 7 apresenta as permutações permitidas para os filhos dos nós internos.

A partir de uma árvore PQR, é possível extrair permutações que satisfazem as restrições (se existirem); basta obter sua fronteira, ou seja, ler as folhas da árvore da esquerda para direita. A Figura 8 esquematiza o procedimento de geração de uma árvore PQR e a obtenção de permutações válidas.

Em alguns casos, para um dado conjunto de objetos e suas restrições, podem não existir permutações que satisfaçam todas as restrições ao mesmo tempo. Algumas restrições podem ser conflitantes (por exemplo, quando  $U=\{a,b,c\}$  e  $R=\{ab, ac, bc\}$ ). Nesses casos, é impossível obter permutações válidas; o resultado da árvore PQR apenas conterá permutações inválidas. Apesar disso, esse resultado respeitará um subconjunto de restrições e indicará em um nó R elementos pertencentes às restrições que não puderam ser satisfeitas.



**Figura 7** Exemplos de árvores PQR e permutações possíveis. Em (a), um nó P, em (b), um nó Q e em (c), um nó R.

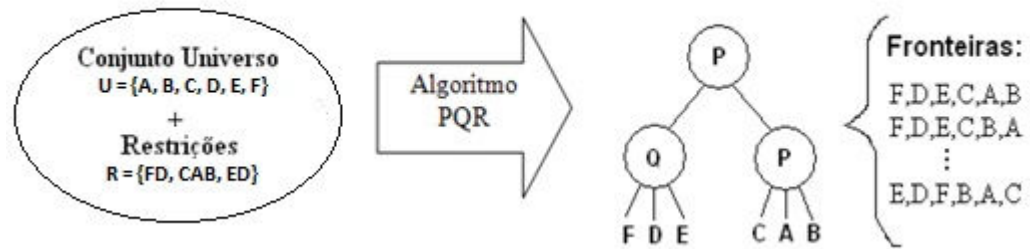


Figura 8 Processo de construção de uma árvore PQR. Figura adaptada de Silva (2010), p.6

### 2.3.1.3 Implementações do PQR-Sort

Em Silva (2010), convencionou-se chamar de PQR-Sort métodos de seriação que empregam árvores PQR; esta mesma nomenclatura será utilizada neste trabalho. Os métodos de seriação baseados em árvores PQR são aplicáveis a dados **binários** e **não-binários**. Para dados binários a seriação ocorre de maneira semelhante à solução de um problema dos uns consecutivos; ou seja, basta tratar a matriz a ser seriada como um problema dos uns consecutivos. Já para os casos não-binários existem duas abordagens: transformar um caso não-binário em binário e utilizar o refinamento de restrições (MELO, 2009). A seguir serão explicadas as abordagens de PQR-Sort para matrizes binárias e não-binárias.

### 2.3.1.4 PQR-Sort para matrizes binárias

Para serializar matrizes binárias utilizando o PQR-Sort, é preciso resolver o problema dos uns consecutivos utilizando árvores PQR. Para serializar uma matriz binária  $D$ , deve-se seguir os cinco passos a seguir:

1. Extrair restrições da matriz  $D$  para linhas, ou seja, indicar as colunas que possuem valores não nulos que devem permanecer unidos. Esse passo é ilustrado na Figura 9.
2. Construir uma árvore PQR para as restrições de linhas encontradas no passo anterior.
3. A partir da árvore PQR extrair a fronteira.
4. Repetir o procedimento de 1 a 3 para colunas.
5. Reordenar linhas e colunas conforme as fronteiras obtidas.

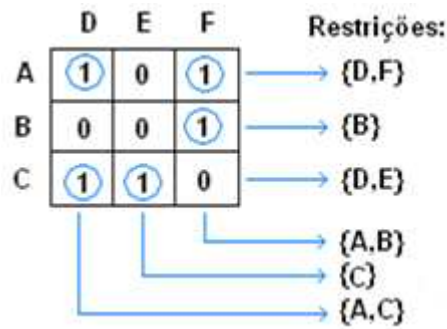


Figura 9 Extração de restrições para linhas e colunas da matriz. Figura extraída de Silva (2010), p. 7.

A Figura 10 apresenta um esquema ilustrando os 3 passos: a obtenção das restrições (a), geração das árvores PQR (b) e, por fim, extração das fronteiras e a ordenação pelas fronteiras obtidas (c). A Figura 11 exibe uma matriz 100x100 aleatória ordenada pelo PQR-Sort.

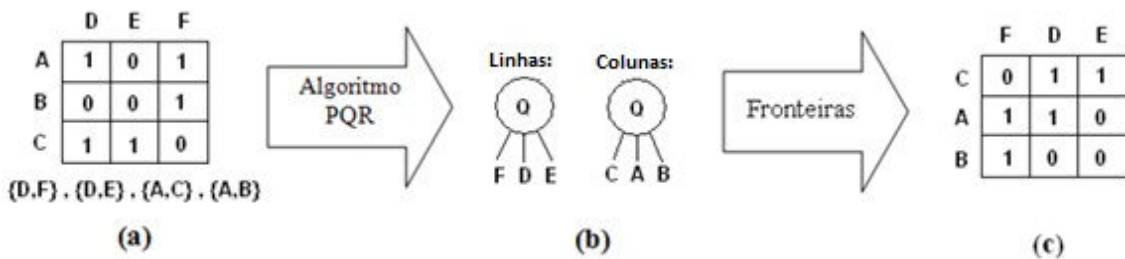


Figura 10 Reordenação de uma matriz binária pelo PQR-Sort. Figura adaptada de Silva (2010), p.7.

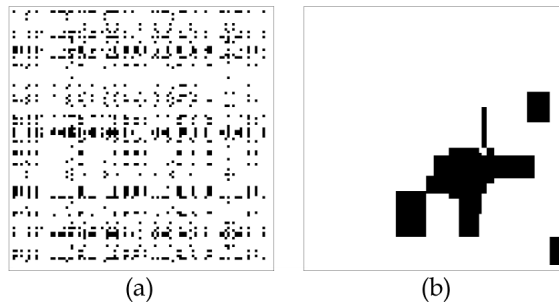


Figura 11 Uma matriz binária 100x100 ordenada pelo PQR-Sort. Em (a) a matriz com uma ordenação aleatória e em (b) a matriz após a ordenação com o PQR-Sort

### 2.3.1.5 PQR-Sort para matrizes não-binárias

Para aplicar árvores PQR para matrizes não-binárias algumas adaptações devem ser feitas; pois, originalmente como apresentada em Meidanis (2002), as árvores PQR resolvem o problema dos uns consecutivos e por isso podem reordenar apenas dados binários. A seguir serão

apresentadas duas abordagens para prover reordenação de dados não-binários com uso de árvores PQR: **transformar um caso não-binário em binário** e **refinamento de restrições**, ambas propostas por Melo (2009).

O método de **transformar um caso não-binário em binário** foi inspirado nas técnicas utilizadas por Mäkinen e Siirtola (2000) para casos não-binários no algoritmo de Sugiyama (Seção 2.3.3). Para transformar variáveis de saída não-binárias em binárias, basta oferecer ao usuário uma opção para selecionar um valor limite; valores superiores ou iguais ao valor limite são considerados uns, e valores inferiores são considerados zeros. Essa metodologia é recomendada apenas para tabela de dados  $N \times N \Rightarrow Q$  (ver nota<sup>3</sup>) – não funciona para variáveis de saída nominais, porque a princípio não foram definidos critérios apropriados para discretizar dados nominais em 0s e 1s.

O **refinamento de restrições** é um outro algoritmo para reordenar também tabelas de dados  $N \times N \Rightarrow Q$ . Quando se analisa uma tabela com dados quantitativos, muitas vezes não é interessante que apenas valores iguais fiquem unidos, pois pode haver casos em que nem existam valores iguais na tabela de dados. Por isso, em um caso quantitativo, uma boa solução (ordenação) é aquela que não só aproxima valores iguais, mas também valores numericamente próximos. Esse é o princípio utilizado nesse algoritmo.

No refinamento de restrições defini-se uma quantidade  $x$  de iterações e os valores da matriz são divididos em  $x$  intervalos de valores; a cada iteração é agrupado na matriz um intervalo de valores. A Figura 12 ilustra um exemplo de uma matriz com valores de 0 a 100 (matriz desordenada); foram utilizadas quatro iterações; em (b) primeiro passo do refinamento, agrupamento dos valores maiores que zero; em (c) agrupamento dos valores maiores que 25; em (d) agrupamento dos valores maiores que 50; e, em (e), o resultado final, agrupamento dos valores maiores que 75. Cada passo acrescenta restrições tentando não infringir as anteriores.

Com esse procedimento, as árvores PQR – que trabalham com restrições binárias – podem reordenar dados não-binários. Isso é possível, pois em cada um dos quatro passos do exemplo a matriz pode ser considerada binária. Por exemplo, no primeiro passo, para agrupar valores

---

<sup>3</sup> Notação para representação de tabelas utilizada em Melo (2009).  $N \times N \Rightarrow Q$  indica uma tabela em que os dados de entrada são Nominais (N) e as variáveis de saída são Quantitativas (Q).

maiores que 0 basta considerar todos os valores maiores que 0 como “1s”; já no segundo passo, são considerados 0 todos os valores menores que 25 e considerados com “1s” os demais valores; dessa forma, em cada passo se obtém uma matriz binária a partir da qual pode-se extrair restrições e gerar uma árvore PQR.



□ A	20	2	30	70	10	88
□ B	24	0	10	24	22	60
□ C	13	21	35	13	13	28
□ D	17	0	24	25	27	19
□ E	0	15	50	22	11	25
□ F	0	12	22	12	2	90
□ G	34	12	90	79	10	69
□ H	2	0	90	19	10	90
Origem	□ U	□ V	□ W	□ X	□ Y	□ Z
	Destino					

(A)

□ D	17	25	19	24	27	0
□ H	2	19	90	90	10	0
□ B	24	24	60	10	22	0
□ G	34	79	69	90	10	12
□ C	13	13	28	35	13	21
□ A	20	70	88	30	10	2
□ E	0	22	25	50	11	15
□ F	0	12	90	22	2	12
Origem	□ U	□ X	□ Z	□ W	□ Y	□ V
	Destino					

(B)

□ D	17	25	19	24	27	0
□ B	24	24	60	10	22	0
□ H	2	19	90	90	10	0
□ C	13	13	28	35	13	21
□ G	34	79	69	90	10	12
□ A	20	70	88	30	10	2
□ F	0	12	90	22	2	12
□ E	0	22	25	50	11	15
Origem	□ U	□ X	□ Z	□ W	□ Y	□ V
	Destino					

(C)

□ D	17	25	19	24	27	0
□ B	24	24	60	10	22	0
□ H	2	19	90	90	10	0
□ F	0	12	90	22	2	12
□ G	34	79	69	90	10	12
□ A	20	70	88	30	10	2
□ C	13	13	28	35	13	21
□ E	0	22	25	50	11	15
Origem	□ U	□ X	□ Z	□ W	□ Y	□ V
	Destino					

(D)

□ D	17	19	24	25	27	0
□ B	24	60	10	24	22	0
□ H	2	90	90	19	10	0
□ F	0	90	22	12	2	12
□ A	20	88	30	70	10	2
□ G	34	69	90	79	10	12
□ C	13	28	35	13	13	21
□ E	0	25	50	22	11	15
Origem	□ U	□ Z	□ W	□ X	□ Y	□ V
	Destino					

(E)

Figura 12 Refinamento de restrições com quatro passos; em (a) a situação inicial (matriz desordenada), em (b), (c), (d) e (e) os passos do refinamento.

### 2.3.2 2D Sort

Segundo Mäkinen e Siirtola (2000), *2D Sort* é um algoritmo heurístico simples, aplicável a matrizes com dados binários e não-binários, que tenta agrupar blocos de valores semelhantes no canto superior esquerdo e no canto inferior direito da matriz. Para isso, o algoritmo sucessivamente ordena as linhas e colunas conforme seus valores ponderados. O peso de cada termo em uma linha é o índice da coluna a que ele pertence. E o peso de cada termo em uma coluna é o índice da linha a que ele pertence. O Algoritmo 1, baseado nas informações apresentadas por Mäkinen e Siirtola (2000), ilustra o funcionamento do método *2D Sort*.

---

#### Algoritmo 1

---

```
enquanto (existir alterações nas linhas ou colunas e não for identificado um caso infinito)
{
    Ordenar linhas segundo valores ponderados;
    Ordenar colunas segundo valores ponderados;
}
```

---

Como apresentado no Algoritmo 1, o método *2D Sort*, com o uso de uma estrutura de iteração, repetidamente ordena linhas e colunas segundo seus valores ponderados. A cada laço de iteração deve-se verificar as últimas trocas de linhas ou colunas, pois existem casos de iteração infinita, chamados em Siirtola e Mäkinen (2005) de casos não-estáveis. Nessas situações, as mesmas linhas e colunas são permutadas indefinidamente. Por esse motivo, o algoritmo *2D Sort* deve identificar esses casos não-estáveis e parar a iteração.

A Figura 2 apresenta uma matriz com dados fictícios extraída de Melo (2009); essa tabela informa a presença de 9 características nas cidades *A, B, C, ... , P*. A Figura 13 mostra a mesma tabela da Figura 2 após a aplicação do algoritmo *2D Sort*.

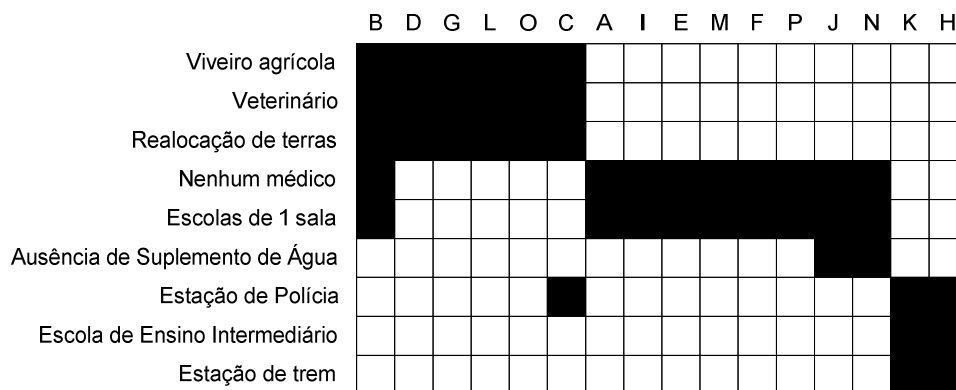


Figura 13 Reprodução da tabela da Figura 2 após aplicação do algoritmo 2D Sort. Figura extraída de Melo (2009), p.30.

A Figura 13, com as linhas e colunas reordenadas pelo *2D Sort*, já auxilia a análise de dados e a identificação de padrões, mesmo não sendo a melhor solução. Como se nota na Figura 13, existem três grupos de valores ao longo da diagonal principal; contudo, existe um valor que não pertence a nenhum grupo (C, estação de polícia). Além disso, o PIC não foi resolvido, pois a coluna C não possui todos os valores pretos consecutivos. Esses resultados não-ótimos são comuns em métodos heurísticos. Para lidar com essa situação, a interface que exibe essa solução poderia permitir que o usuário rearranjasse linhas e colunas manualmente. Assim, com algumas trocas de linhas seria possível obter a solução ótima exibida na Figura 14.



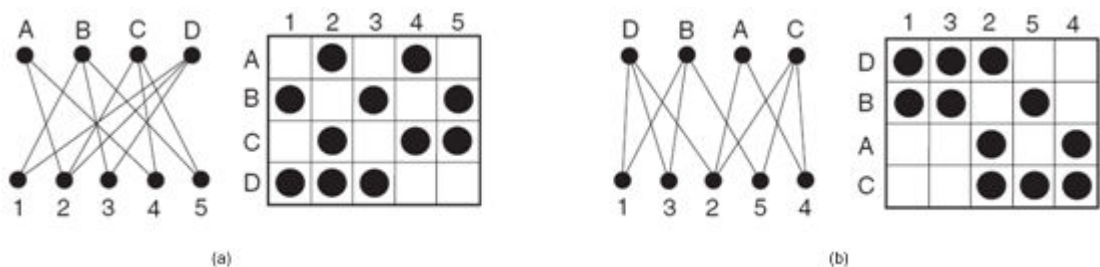
Figura 14 Resultado ótimo para a reordenação de linhas e colunas da tabela da Figura 2.

### 2.3.3 Algoritmo de Sugiyama Modificado (Heurística Baricêntrica)

O algoritmo de Sugiyama originalmente é utilizado para minimizar o cruzamento de arestas em um grafo bipartido. Um grafo bipartido possui dois grupos de vértices (dois níveis), e, cada

uma de suas arestas está ligada a apenas um vértice de cada grupo; ou seja, não há ligações entre vértices do mesmo grupo.

Como descrito em Mäkinen e Siirtola (2000), o problema de minimizar o cruzamento de arestas em um grafo bipartido pode auxiliar no agrupamento de valores em uma matriz. Pois, como exemplificado na Figura 15, ao reorganizar os vértices de um grafo para evitar cruzamento de arestas, a matriz de adjacência que o representa se organizará: tenderá a agrupar valores no canto superior esquerdo e inferior direito da matriz. E, logo, auxiliará a resolução de um PIC, podendo inclusive encontrar uma solução ideal. A aplicação do algoritmo de Sugiyama para reordenação de dados em matrizes é referenciada na literatura como Algoritmo de Sugiyama modificado (MÄKINEN e SIIRTOLA, 2000) ou como Heurística Baricêntrica (SIIRTOLA E MÄKINEN, 2005).



**Figura 15** Relação entre seriação e o problema da redução de cruzamentos de arestas em grafos bipartidos. Em (a), à esquerda grafo e à direita matriz de adjacência antes da reordenação; em (b), ambos após a reorganização para evitar cruzamentos. (Figura extraída de Mäkinen e Siirtola (2000), p. 464).

No algoritmo de Sugiyama, a posição de cada vértice é recalculada como sendo a média da posição de seus vizinhos. Na Figura 15a, por exemplo, o valor médio dos vizinhos de A é  $(2+4)/2 = 3$ , logo A deve passar a ocupar essa posição em sua partição. Esse mesmo cálculo é feito para todas as outras letras (B, C e D). Após isso, ordena-se as letras segundo esse valor médio. O mesmo procedimento é feito com o grupo inferior de vértices da Figura 15a. A ordenação dos grupos inferior e superior é repetida indefinidamente, até que não se note mais alterações na matriz. A Figura 15b mostra a situação final após o processamento do grafo da Figura 15a por esse algoritmo.

Portanto, para tentar agrupar valores numericamente próximos pelo algoritmo de Sugiyama modificado, basta considerar a matriz como sendo uma matriz de adjacência de um grafo, e

aplicar sobre esse grafo o algoritmo de Sugiyama. Assim, reordenando o grafo para reduzir cruzamentos, também se processará uma tentativa de formar grupos de valores no canto superior esquerdo e inferior direito da matriz. A Figura 16 apresenta o resultado da execução do algoritmo de Sugiyama Adaptado para o exemplo das cidades (Figura 2).

Para o algoritmo de Sugiyama lidar com casos em que os dados não são binários, Siirtola e Makinen (2005) apresentam duas metodologias:

a) **Transformar um caso não-binário em um caso binário:** Nessa abordagem um caso com dados não-binários é convertido em um caso binário. Isso é obtido através de um valor limite informado pelo usuário. Todos os valores superiores ou iguais ao valor limite são considerados 1 e valores inferiores são considerados 0. Assim, com essa matriz binária, aplica-se as metodologias para casos binários apresentadas anteriormente.

b) **Utilizar média ponderada:** Essa abordagem é semelhante ao caso binário. Mas, ao invés de utilizar média simples, utiliza-se média ponderada dos vizinhos de um nó. Os valores a serem ponderados são as entradas da tabela, e os pesos são os índices das linhas ou colunas. Assim como no caso binário, a ordenação é repetida indefinidamente até que não se perceba mais alterações. Exemplos dessa abordagem podem ser encontrados em Melo (2009).

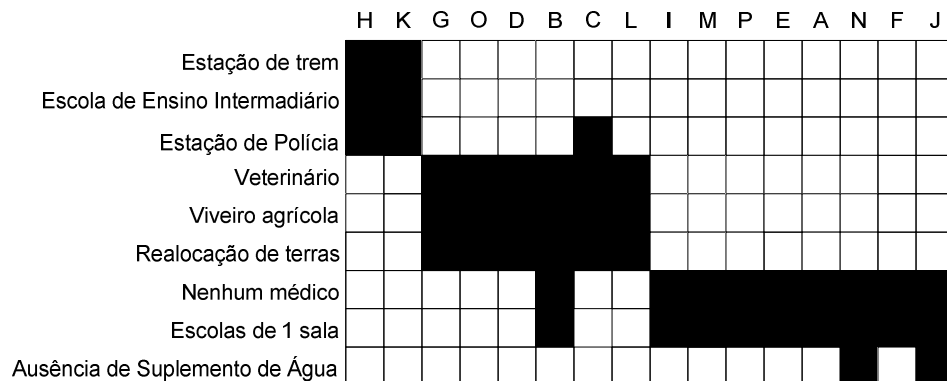


Figura 16 Ordenação da matriz da Figura 2 utilizando o algoritmo de Sugiyama Modificado.

### 2.3.4 Algoritmo “Bond Energy”

O Algoritmo Bond Energy - BE - (ARABIE e HULBERT, 1990) é um algoritmo heurístico para rearranjar linhas e colunas de matrizes visando agrupar valores próximos; é aplicável a matrizes *Two-way two-mode*, ou seja, matrizes com casos e variáveis representado objetos diferentes. O BE tenta ordenar linhas e colunas de forma que cada célula da matriz tenha o máximo possível de vizinhos (vizinhança 4-conectada) com valores próximos. A Figura 17 ilustra a vizinhança 4-conectada (marcada em vermelho), para a célula  $i_{22}$ . O BE possui três passos fundamentais:

$$\begin{bmatrix} \dot{i}_{11} & \dot{i}_{12} & \dot{i}_{13} & \dot{i}_{14} \\ \dot{i}_{21} & \dot{i}_{22} & \dot{i}_{23} & \dot{i}_{24} \\ \dot{i}_{31} & \dot{i}_{32} & \dot{i}_{33} & \dot{i}_{34} \\ \dot{i}_{41} & \dot{i}_{42} & \dot{i}_{43} & \dot{i}_{44} \end{bmatrix}$$

Figura 17 Exemplo da Vizinhança 4-conectada. Marcado em vermelho os vizinhos da célula  $i_{22}$ .

**Passo 1.** Da matriz original escolha uma coluna aleatória.

**Passo 2.** Tente posicionar a coluna escolhida no Passo anterior em uma posição que maximize o número de vizinhos com valores próximos. Uma das maneiras de verificar a melhor posição é calculando a Medida de Efetividade (ME - Seção 2.4.6) para cada posição possível. A ordenação que gerar a maior ME é escolhida. Enquanto todas as colunas não forem escolhidas e posicionadas, volte ao Passo 1.

**Passo 3.** Repita o procedimento para linhas.

A Figura 18 ilustra o funcionamento dos Passos 1 e 2. É apresentada a execução do algoritmo quando duas colunas já foram posicionadas,  $\{c1 \text{ e } c4\}$ . O algoritmo deve ser executado até que todas as colunas estejam posicionadas. Na Figura 18, a próxima coluna a ser posicionada é  $c3$ . O valor da ME é calculado para todos os posicionamentos possíveis de  $c3$ . Na Figura 18, foi

escolhida a ordenação {c1, c3 e c4} por gerar o maior valor para a função de efetividade; ou, em outras palavras, a configuração de colunas que gera mais vizinhos com valores próximos.

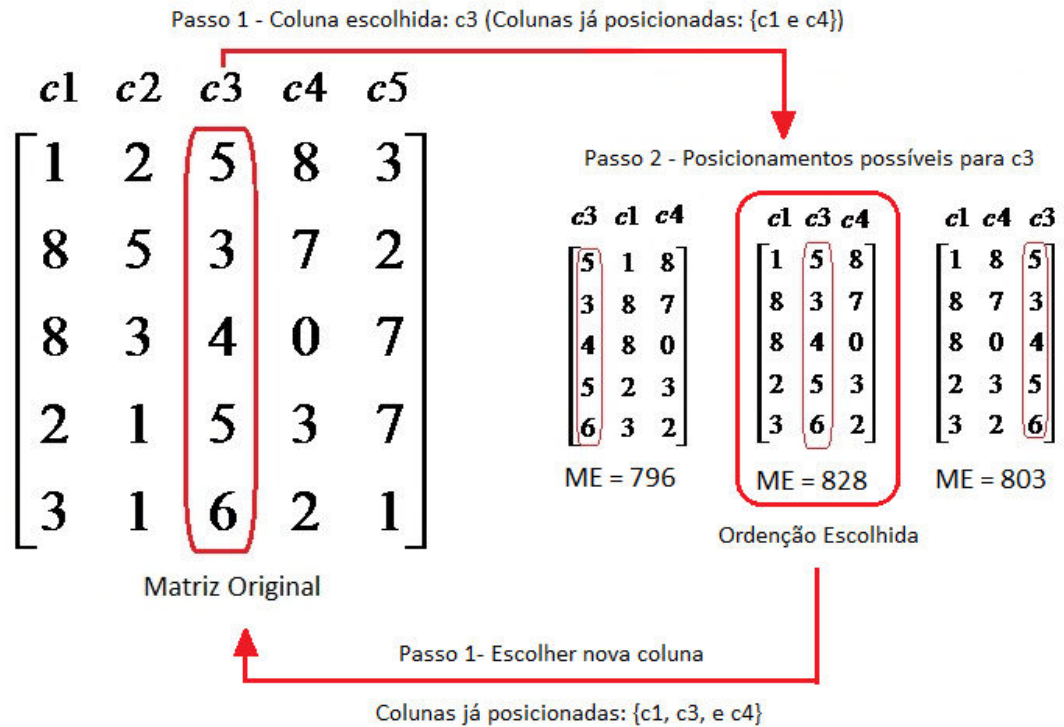


Figura 18 Esquema do Algoritmo BE para colunas da matriz. A Figura ilustra o posicionamento da coluna c3.

### 2.3.5 Seriação pelo Problema do Caixeiro Viajante

Realizar a seriação de uma matriz de dissimilaridade é equivalente a resolver o Problema do Caixeiro Viajante - PCV (HAHSLER ET AL., 2010). Isso é possível porque uma matriz de dissimilaridade pode ser representada como um grafo ponderado (Figura 19); e, além disso, a solução do PCV (sequência de vértices) é uma ordenação de linhas e colunas que agrupa valores semelhantes e, portanto, produz uma boa seriação.

Contudo, é necessária uma adaptação para seriar uma matriz através da solução do PCV. O Problema do Caixeiro Viajante consiste em encontrar o circuito Hamiltoniano de menor peso, ou seja, encontrar uma sequência de nós com o menor peso que passe por todo o grafo sem repetir vértices e volte ao nó de origem (WILSON, 1996). Dessa forma, a solução do problema é um circuito - começa e termina no mesmo vértice. Por exemplo, para o grafo da Figura 19b, um

possível resultado para o PCV seria {A, D, C, B, A}. Essa solução não ajuda na seriação de matrizes, pois possui repetições – não faz sentido uma matriz de dissimilaridade com repetição de uma coluna.

Uma das formas de adaptar o PCV para encontrar o caminho Hamiltoniano é acrescentar uma coluna fictícia na matriz, apenas com valores 0 (GARFINKEL, 1976 apud HAHLER ET AL., 2010). Então, resolve-se o PCV<sup>4</sup>; tendo a solução, basta eliminar a coluna fictícia para quebrar o circuito Hamiltoniano em um caminho. Esse caminho é a ordenação de linhas (ou colunas) da matriz que mantém valores próximos unidos.

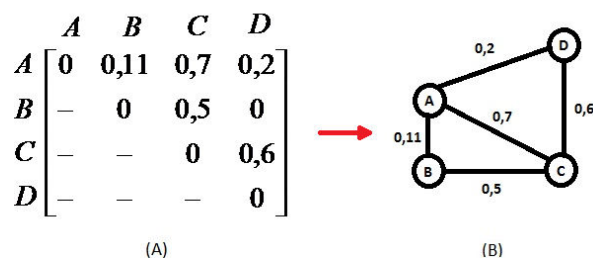


Figura 19 Em (A) uma matriz de dissimilaridade; em (B) um grafo representando essa matriz.

### 2.3.6 Clusterização Hierárquica

Clusterização hierárquica é uma maneira de agrupar elementos em uma árvore binária, de forma que elementos relacionados são mantidos em uma mesma subárvore. Esse conceito é aplicável a reordenação de matrizes; matrizes de similaridade ou dissimilaridade podem ser ordenadas através da clusterização hierárquica. O resultado de uma clusterização hierárquica pode ser visualizado em um dendrograma (Figura 17), ou seja, uma árvore em que cada nó interno divide-se em duas subárvores e possui um valor de similaridade (ou dissimilaridade) associado. Assim, como no *PQR-Sort*, a partir de uma árvore – nesse caso um dendrograma – é possível extrair permutações de linhas e colunas que mantêm valores próximos unidos. A Figura 20 esquematiza como são geradas permutações com clusterização hierárquica.

Um dendrograma gerado a partir de uma matriz de  $n$  elementos possui  $2^{n-1}$  nós internos; e, em cada nó interno as subárvores direita e esquerda (ou folhas) podem ser permutadas, resultando em  $2^{n-1}$  permutações de folhas possíveis em um dendrograma. Qualquer uma dessas permutações

<sup>4</sup> Algoritmos para resolver o PCV podem ser encontrados em Gutin e Punnen (2002).



gerará matrizes com valores próximos agrupados. O procedimento de usar diretamente a permutação gerada pelo dendrograma é chamado de clusterização hierárquica. Contudo, recentemente têm surgido algoritmos para extrair dentre as permutações possíveis do dendrograma uma permutação ótima (a que mantém mais valores próximos unidos). Dentre eles, é importante destacar o algoritmo apresentado por Bar-Joseph et al. (2001); nesse trabalho os autores apresentam um algoritmo baseado no Problema do Caixeiro Viajante para escolher a permutação de folhas ótima a partir do dendrograma. Esse algoritmo foi chamado de Ordenação Ótima de Folhas.

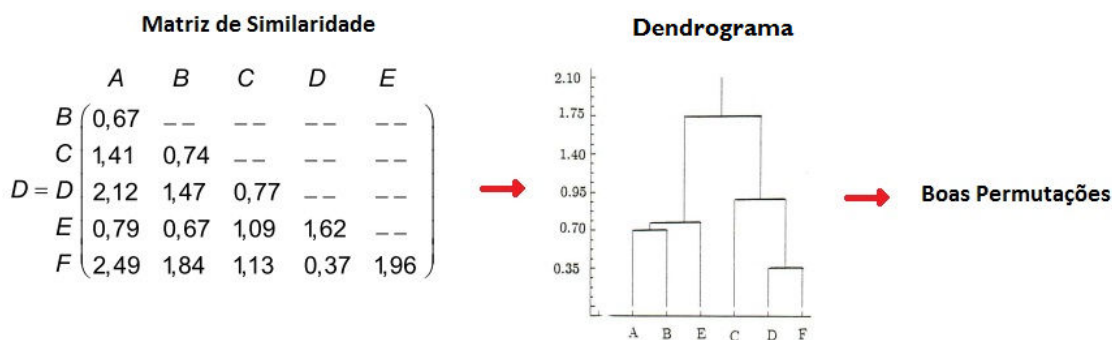


Figura 20 Esquema da clusterização hierárquica para reordenar linhas ou colunas de uma matriz. A partir de uma matriz de similaridade gera-se um dendrograma; e, partir do dendrograma é possível extrair boas permutações para linhas ou colunas.

### 2.3.7 Seriação Elíptica

Chen (2002) criou um algoritmo de ordenação de matrizes chamado Seriação Elíptica de Rank dois (do inglês: *Rank-Two Elliptical Seriation*). Na seriação elíptica utiliza-se a sequência de matrizes de correlação  $R = (R^{(1)}, R^{(2)}, \dots, R^{(n)})$  gerada a partir de uma matriz de proximidade  $S$ . Assim, para calcular o valor de  $R^{(1)}$ , basta aplicar coeficiente de correlação de Pearson (Fórmula 3) para cada célula  $s_{ij}$  da matriz  $S$ . Para calcular  $R^{(2)}$  basta aplicar a fórmula de Pearson para cada célula da matriz  $R^{(1)}$ ; e assim sucessivamente. A fórmula do coeficiente de correlação de Pearson é exibida na Fórmula 3. Nessa fórmula  $\bar{s}_i$  indica a média  $p^{-1} \sum_k s_{ik}$  ( $p$  indica a ordem da matriz).

$$\rho(i, j) = \frac{\sum_k (s_{ik} - \bar{s}_i)(s_{jk} - \bar{s}_j)}{\sqrt{\sum_k (s_{ik} - \bar{s}_i)^2} \sqrt{\sum_k (s_{jk} - \bar{s}_j)^2}}$$

**Fórmula 3 Correlação de Pearson.**

A sequência  $R$  é convergente, sendo  $R^{(\infty)}$  uma matriz que possui apenas valores -1 e 1. Contudo, para a seriação elíptica são importantes apenas os valores encontrados um pouco antes da convergência. Para cada valor da sequência  $R$  é calculado um valor chamado *rank*; o *rank* de uma matriz da sequência  $R$  é o número de valores maiores que  $e^{-13}$  (ver nota<sup>5</sup>). Quando a sequência  $R$  converge o valor do *rank* é 1; para a seriação elíptica são utilizados os valores da sequência  $R$  que possuem *rank* 2.

Os valores das matrizes da sequência  $R$  podem ser estudados em estruturas elípticas; os dois primeiros autovetores de cada matriz da sequência são utilizados para calcular a elipse; e os vetores de linhas ou colunas da matriz  $R^{(n)}$  são plotados dentro dessa elipse. A Figura 21 ilustra a evolução da sequência  $R$  e dos valores do *rank* -  $\rho^{(n)}$  - para dados de doenças mentais (CHEN, 2002)<sup>6</sup>. Na situação em que o *rank* é 2 são formados dois grupos de valores sobre a elipse e cada valor possui uma única posição relativa; com isso, é possível calcular uma permutação de linhas ou colunas. A Figura 22 ilustra a extração de permutações quando o *rank* é 2.

---

<sup>5</sup> Chen (2002) utilizou esse valor, mas, segundo o autor, podem ser usados outros valores.

<sup>6</sup> Para mais informações sobre o domínio de dados consultar Chen (2002), p.3.

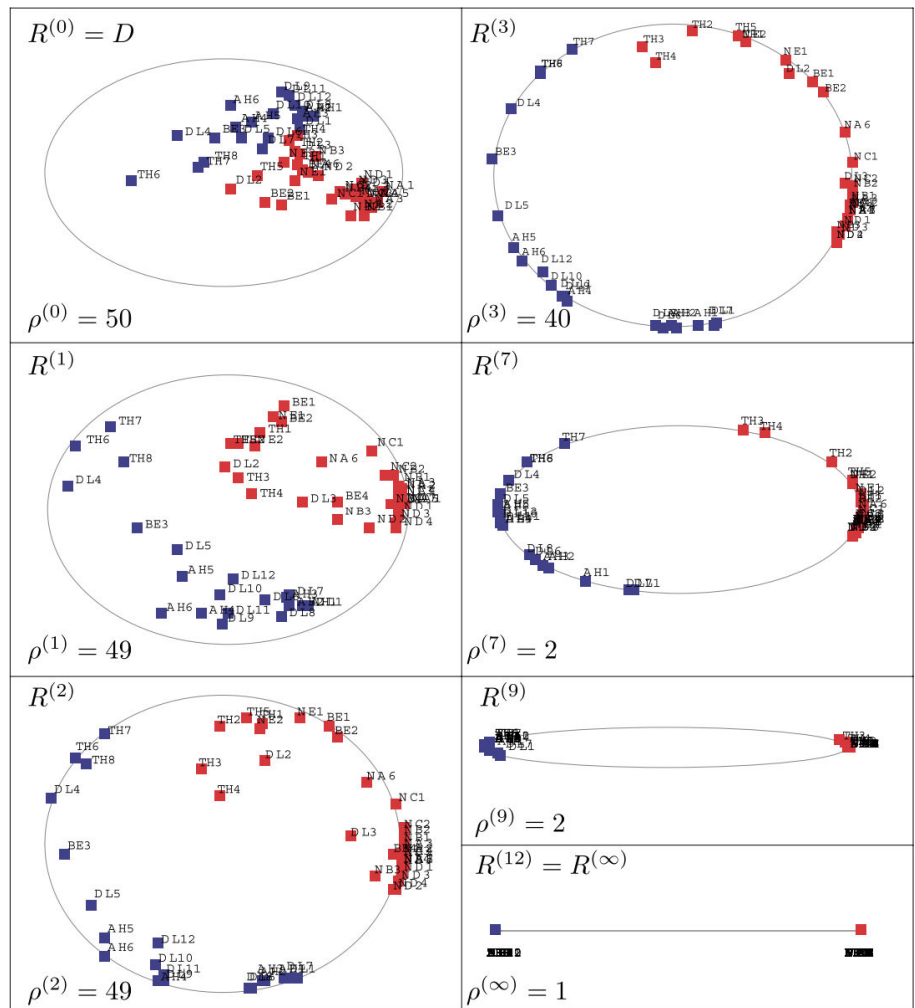


Figura 21 Estruturas elípticas para algumas matrizes da sequência  $R$ .  $\rho^{(n)}$  indica o valor do *rank*. Figura extraída de Chen (2002), p. 2.

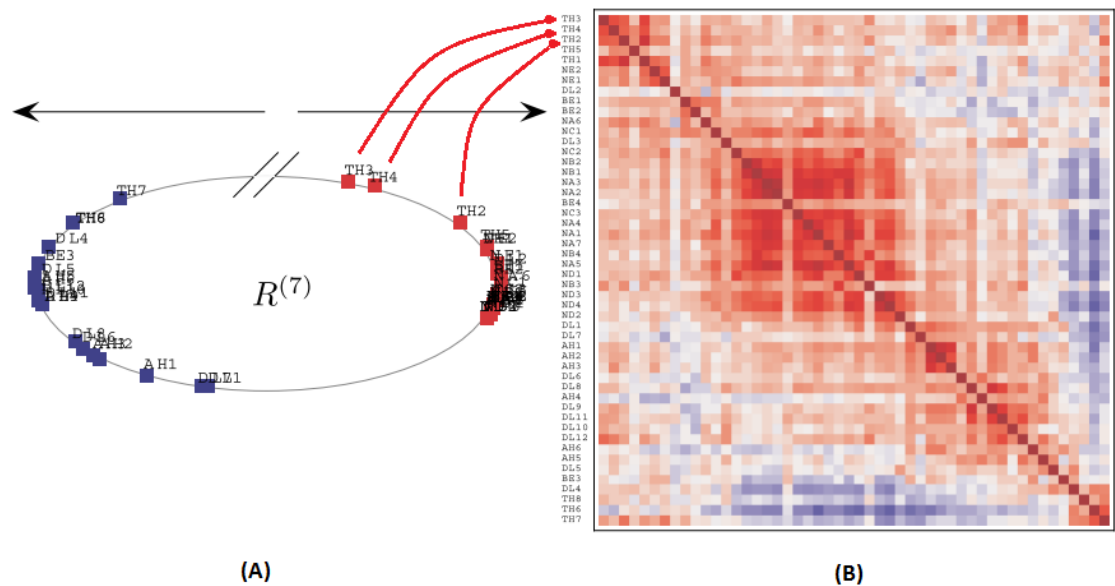


Figura 22 Esquema da reordenação de linhas através da seriação elíptica. Em (a) a estrutura elíptica de *rank* 2; em (b), um mapa de calor ordenado através das posições relativas da matriz. Cada quadrado sobre a elipse é um vetor de linhas da matriz  $R^{(7)}$ . Figura adaptada de Chen (2002), p. 13.

### 2.3.8 Classificação de Métodos

O Quadro 2 apresenta algumas características dos métodos de ordenação apresentados neste trabalho. Neste quadro, algumas células estão marcadas com interrogação, pois, algumas informações não estão explícitas nos artigos sobre os algoritmos de seriação.

Para o algoritmo BEA, uma consideração deve ser feita: McCormick et al. (1972) usam BEA para dados binários e não-binários. Arabie e Hubert (1990), por sua vez, afirmam que o BEA só deveria ser utilizado para dados binários.

Ainda no Quadro 1, nas complexidades dos algoritmos Sugiyama e 2D-*Sort*, o valor  $t$  indica o número de iterações executadas; por serem métodos heurísticos, é impossível determiná-lo previamente.

Em relação a clusterização Hierárquica, existem diversas implementações, com diferentes complexidades. A implementação citada no Quadro 2 foi a que possuía a menor complexidade de tempo.

Quadro 2 Comparação entre algoritmos de reordenação de matrizes .

Algoritmo	Heurístico	Dados Binários	Dados não-binários	Complexidade	Função de Avaliação que visa maximizar	Matriz de Entrada
<i>PQR-Sort</i>	NÃO	SIM	?	$O(n^2 \alpha(n^2))$	Sem função	Matriz de Dados
<i>2D Sort</i>	SIM	SIM	SIM	$O(n^2 t)$	Sem Função	Matriz de Dados
<b>Heurística Baricêntrica</b>	SIM	SIM	SIM	$O(n^2 t)$	Sem Função	Matriz de Dados
<b>BEA</b>	SIM	SIM	SIM	?	Medida de Efetividade	Matriz de Dados
<b>Seriação pelo PCV</b>	Depende da implementação	?	SIM	Depende da implementação	Medida do Caminho Hamiltoniano	Dissimilaridade
<b>Clusterização Hierárquica</b>	?	?	SIM	$O(n^3)$ (Eisen et al., 1998)	?	Dissimilaridade ou Similaridade
<b>Seriação Elíptica</b>	Não	?	SIM	?	?	Dissimilaridade

## 2.4 Funções de avaliação de ordenação

Na seção anterior foram apresentados algoritmos para reordenar dados em matrizes. Esta seção trata de uma preocupação posterior à reordenação de dados: a avaliação da qualidade de uma ordenação.

Segundo Hahsler et al. (2010), a Arqueologia foi uma das primeiras áreas a aplicar métodos de seriação como métodos formais; o objetivo principal era encontrar uma ordem cronológica para os túmulos descobertos na região do Nilo com base nos objetos encontrados (Petrie, 1899). Com uma matriz binária permutável com túmulos e objetos encontrados nos casos e variáveis (respectivamente), a ordem cronológica era encontrada. Arqueólogos permutavam manualmente linhas e colunas da matriz, procurando agrupar dados com valores não-nulos próximos à diagonal principal; assim, com essa organização, obtinha-se a ordem dos túmulos, indicando a ordem cronológica. Inicialmente a verificação da qualidade da ordenação era feita manualmente, e, em um momento posterior, o estatístico Robinson (1951) criou funções para avaliação dessa qualidade.

Esse exemplo histórico é importante para notar características relevantes das primeiras aplicações de ordenação: 1) as ordenações eram feitas manualmente; 2) conforme citam Hahsler et al. (2010), a avaliação da qualidade da ordenação era feita pelo arqueólogo. Atualmente, as características 1 e 2 deixaram de ser inteiramente manuais, e agora são mais automatizadas.<sup>7</sup> Existem métodos de reordenação automáticos para dados em estruturas matriciais e funções de avaliação para ordenação de matrizes, como Medida de Efetividade (Seção 2.4.6), Medida de Inércia (Seção 2.4.4), entre outras.

Entretanto, apesar da variedade de funções de avaliação, algumas considerações devem ser feitas em relação ao seu uso. As diferentes funções de avaliação medem a presença de características diferentes, pois têm origem em áreas de aplicação diferentes. Por exemplo, a função de Robinson, por ter origem na área de arqueologia e pelas especificidades das análises dessa área, busca verificar se os valores semelhantes (ou seja, de alta similaridade) estão próximos à diagonal principal. Já a função de Inércia é um pouco diferente: busca verificar se valores diferentes (com dissimilaridade alta) estão longe da diagonal principal. Em algumas situações, essas diferenças fazem com que uma função de avaliação só indique bons resultados em um tipo de algoritmo de ordenação; isso pode dificultar comparações entre diferentes algoritmos de ordenação.

Além disso, outra consideração a ser feita é em relação à qualidade de uma reordenação sob o ponto de vista do usuário. Um algoritmo de avaliação, como a Medida de Efetividade, por exemplo, pode verificar e quantificar a presença de determinada característica na ordenação da matriz; mas, ainda não se sabe se ordenações consideradas boas pela ME são também consideradas boas para os usuários que realizarão a análise. O mesmo vale para as outras funções. O que se pode inferir a princípio é que as funções de avaliação apresentadas nesse capítulo são boas para usuários porque agrupam valores próximos; pois, segundo as Leis Gestalt (MAZZA, 2009), valores agrupados são melhor percebidos pela visão humana.

Nesse contexto, dois trabalhos procuraram estudar o processo de análise dos dados em matrizes centrado no usuário. Siirtola e Mäkinen (2005) estudaram comparativamente como era a análise dados em matrizes com ordenação manual e com ordenação automática (Seção 2.5.1). Já

---

<sup>7</sup> Vale lembrar que o uso de algoritmos de ordenação não exclui a necessidade da participação humana na reordenação de linhas e colunas de uma estrutura matricial.

Henry e Fekete (2006) elaboraram um conjunto de tarefas (análises) para usuários realizarem em matrizes ordenadas por três diferentes algoritmos de seriação; os usuários realizaram as tarefas em papel digital, assim, todas as marcas feitas pelos usuários na matriz foram armazenadas (Seção 2.5.2). A seguir serão explicados os tipos de funções de avaliação apresentadas neste capítulo.

#### **2.4.1 Funções de Perda e Mérito**

Funções que avaliam ordenações de matrizes podem ser de dois tipos: funções de perda e funções de mérito. Funções de mérito, que neste trabalho são denotadas pela letra  $M$ , são aquelas que medem a proximidade de uma matriz com a situação ideal, ou seja, quanto maior o valor da função, melhor a avaliação da matriz. As funções de mérito abordadas neste trabalho são: Medida de Gradiente, Medida de Efetividade e Critério de Inércia.

Por outro lado, as funções de perda, denotadas por  $L$ , tem o significado oposto: seus valores indicam a distância em relação a uma solução ideal. Logo, valores baixos indicam proximidade com a reordenação ideal. As funções de perda abordadas neste trabalho foram: Medida do Caminho Hamiltoniano, Critério dos quadrados mínimos e *Stress*.

Em relação ao parâmetro de uma função de avaliação de ordenação, três possibilidades são consideradas neste trabalho: funções de avaliação podem ser calculadas em matrizes simétricas de dissimilaridade  $D=\{d_{ij}\}$ , em matrizes de similaridade  $S=\{s_{ij}\}$  e em matrizes de dados  $M=\{m_{ij}\}$ . A seguir serão apresentadas 7 funções de avaliação encontradas durante a revisão bibliográfica.

#### **2.4.2 Anti-Robinson Loss Function (ARLS)**

Uma matriz de dissimilaridade cujos valores das células aumentam à medida em que essas células se afastam da diagonal principal é chamada de matriz anti-Robinson perfeita (Robinson, 1951). Conforme Wu et al. (2008), uma matriz anti-Robinson perfeita é desejável por ser uma ordenação global que tem aspecto visual suave e agradável. A Figura 23 ilustra o critério de linhas de uma matriz anti-Robinson.

$$D = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} \\ d_{41} & d_{42} & d_{34} & d_{44} & d_{45} \\ d_{51} & d_{52} & d_{35} & d_{54} & d_{55} \end{pmatrix}$$

Figura 23 Critério de linhas de uma matriz anti-Robinson aplicado a valores acima da diagonal principal. Os valores das células devem ser maiores a medida que se afastam da diagonal principal.

Mais formalmente, uma matriz de dissimilaridade será uma matriz de anti-Robinson se  $d_{ij} \leq d_{ik}$  se  $i < k < j$  e  $d_{ij} \geq d_{ik}$  se  $i < j < k$ .

Hubert et al. (2001) propuseram a função a seguir para calcular diferença de uma matriz de dissimilaridade  $D$  em relação a uma matriz perfeita de anti-Robinson:

$$M(\mathbf{D}) = \sum_{i < k < j} f(d_{ik}, d_{ij}) + \sum_{i < k < j} f(d_{kj}, d_{ij})$$

Nessa função, a função  $f$  mede a violação do critério de uma matriz anti-Robinson perfeita (valores crescentes à medida que se afasta da diagonal principal). Ou seja, é uma função que pode ser adaptada conforme o problema. Neste trabalho, foi adotada uma função  $f$  que conta o número que eventos que não satisfazem o critério de anti-Robinson:

$$f(z, y) = \begin{cases} 0 & \text{se } z \leq y \\ 1 & \text{se } z > y \end{cases}$$

### 2.4.3 Minimal Span Loss Function (MSLF)

Como visto na Seção 2.3.5, uma matriz de dissimilaridade pode ser vista como um grafo ponderado. E, além disso, um caminho Hamiltoniano (WILSON, 1996) nesse grafo pode representar uma permutação de uma matriz de dissimilaridade. Entre todos os caminhos Hamiltonianos, o que produz uma boa seriação são os menores; segundo Hahsler et al. (2010), a minimização do Caminho Hamiltoniano produz uma seriação ótima em relação às dissimilaridades entre vizinhos. A função de perda baseada no caminho Hamiltoniano é:



$$L(\mathbf{D}) = \sum_{i=1}^{n-1} d_{i,i+1}$$

Onde  $n$  é a ordem da matriz de dissimilaridade  $D = \{d_{ij}\}$ . Conforme Chen et al. (2008), esta função de perda foca na otimização de estruturas locais, ou seja, pode ser empregada para encontrar reordenações que evidenciam padrões locais.

#### 2.4.4 Critério de Inércia

Outra forma de analisar o problema de seriação é focar em manter valores de alta dissimilaridade longe da diagonal principal, ao invés de focar em manter os valores de dissimilaridade baixa próximos à diagonal principal (HAHSLER ET AL., 2010). A função que mede a presença dessa característica é (CARAUX E PINLOCHE, 2005):

$$M(\mathbf{D}) = \sum_{i=1}^n \sum_{j=1}^n d_{ij} |i - j|^2$$

#### 2.4.5 Critério dos Quadrados Mínimos

O critério dos quadrados mínimos mede a diferença entre valores de dissimilaridade e distância entre elementos. Em Caraux & Pinloche (2005) é apresentada uma forma de quantificar essa diferença:

$$L(\mathbf{D}) = \sum_{i=1}^n \sum_{j=1}^n (d_{ij} - |i - j|)^2$$

#### 2.4.6 Medida de Efetividade

A Medida de efetividade (MCCORMICK ET AL., 1972) é uma função aplicável a matrizes de dissimilaridade e matrizes *two-way two-mode*, ou seja, matrizes não-simétricas com casos e variáveis representando conjuntos de objetos diferentes. Assim, para uma matriz não-simétrica  $X = \{x_{ij}\}$ , a função de mérito relativa a essa medida é calculada pela expressão

$$M(\mathbf{X}) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^m x_{ij} [x_{i,j+1} + x_{i,j-1} + x_{i+1,j} + x_{i-1,j}]$$

Nessa expressão convencionou-se que todos os valores fora dos limites da matriz - como  $x_{0,j}$  - são iguais a zero. Segundo Hahsler et al. (2010), essa função é maximizada quando cada elemento da matriz tem valor numericamente próximo aos valores de seus vizinhos.

### 2.4.7 Stress

*Stress* é uma função de avaliação para matrizes assimétricas que se baseia nos valores dos vizinhos de cada elemento da matriz. Em Niermann (2005) foram apresentadas duas fórmulas para calcular a vizinhança de um elemento: uma para a vizinhança de Moore (vizinhança de 8 elementos):

$$\sigma_{ij} = \sum_{k=\max(1,i-1)}^{\min(n,i+1)} \sum_{l=\max(1,j-1)}^{\min(m,j+1)} (x_{ij} - x_{kl})^2$$

E outra para vizinhança de Neumann (vizinhança de 4 elementos):

$$\sigma_{ij} = \sum_{k=\max(1,i-1)}^{\min(n,i+1)} (x_{ij} - x_{kj})^2 + \sum_{l=\max(1,j-1)}^{\min(m,j+1)} (x_{ij} - x_{il})^2$$

Ambas as funções podem ser usadas para construir uma função global de *Stress* para a matriz:

$$L(\mathbf{X}) = \sum_{i=1}^n \sum_{j=1}^m \sigma_{ij}$$

## 2.5 Avaliação centrada no usuário

Essa Seção descreve dois experimentos envolvendo usuários para entender questões relacionadas à maneira como usuários efetuam análises em dados matriciais e como a ordenação de dados afeta o entendimento dos usuários.

### 2.5.1 Experimento de Siirtola e Mäkinen (2005)

Siirtola e Mäkinen (2005) elaboraram protótipos de uma matriz reordenável de Bertin; estes protótipos foram submetidos a testes com usuários. O objetivo dos testes era comparar o tempo gasto para encontrar padrões e a qualidade dos *clusters* encontrados em três situações:

1) Usando uma matriz impressa em uma folha com auxílio de uma caneta para encontrar clusters e padrões nos dados;

2) Usando um protótipo que permitia reordenação de linhas e colunas manualmente - através de *Drag-and-Drop*.

3) Usando um protótipo que permitia ordenações automáticas de linhas e colunas.

A Figura 24 ilustra uma das interfaces utilizadas no experimento para a Situação 3. Para a Situação 2, bastaria desabilitar a opção de reordenação automática.

Cada usuário, além tentar encontrar *clusters* de dados em matrizes nas três situações, passou por uma entrevista e preencheu um formulário para avaliar a sua satisfação. Dentre os resultados do experimento conduzido, os principais resultados encontrados foram:

- Através das estatísticas do experimento ficou comprovada uma redução significativa do tempo necessário para encontrar *clusters* quando se utiliza a Situação 3. Além disso, na Situação 3, em menos tempo, usuários encontraram clusters mais próximos das soluções ideais.

- Usuários acreditam que a matriz de correlação ajuda na ordenação da dados.

- É útil substituir os valores de cada célula da matriz por formas geométricas proporcionais, como retângulos e círculos.

- O protótipo da matriz reordenável produz uma boa visão geral dados.

- Ordenar linhas e colunas manualmente não é um procedimento considerado fácil pelos usuários.

Ademais, a respeito do trabalho de Siirtola e Mäkinen (2005) é importante citar o método empregado para avaliar os *clusters* encontrados pelos usuários nas reordenações. Nos testes de Siirtola e Mäkinen (2005), após o usuário reordenar linhas e colunas de uma matriz, era requerida a digitação dos *clusters* encontrados; então, tais *clusters* foram avaliados com a estrutura *lattice* (Figura 25). Essa estrutura é capaz de representar combinações - partições - entre elementos de um conjunto. A Figura 25 mostra a estrutura de *lattice* para os conjuntos {a, b, c, d}, {a, b, c} e {a, b}.

Portanto, com essa estrutura, dada uma escolha de *clusters* feita pelo usuário  $P_u$  e a partição de *clusters* ideal  $P_i$ ; a precisão da partição encontrada pelo usuário é definida como o menor caminho para se chegar de  $P_u$  a  $P_i$ . Ou seja, basta contar o número de trocas entre elementos que é preciso para chegar de  $P_u$  a  $P_i$ . Por exemplo, após a reordenação de uma matriz, um usuário

encontrou a configuração de *clusters*  $P_u = \{c,d\}, \{a\}, \{b\}$ ; considerando como configuração ideal  $P_i = \{c\}, \{a,b,d\}$ , a qualidade da ordenação do usuário seria 3; são necessários 3 passos para se chegar de  $P_u$  a  $P_i$  (esse caminho está marcado em vermelho na Figura 25).

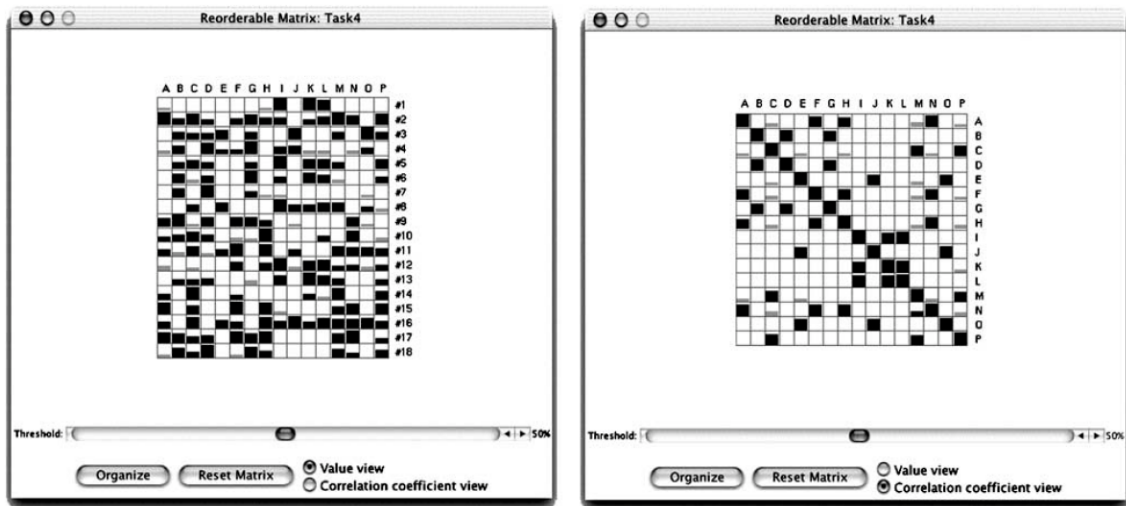


Figura 24 Interfaces dos protótipos de Siirtola e Mäkinen (2005) para reordenação de matrizes. Do lado direito a estrutura com seus valores – o preenchimento da célula é proporcional ao seu valor. Do lado esquerdo, uma matriz de correlação entre colunas. Figura extraída de Siirtola e Mäkinen (2005), p. 35.

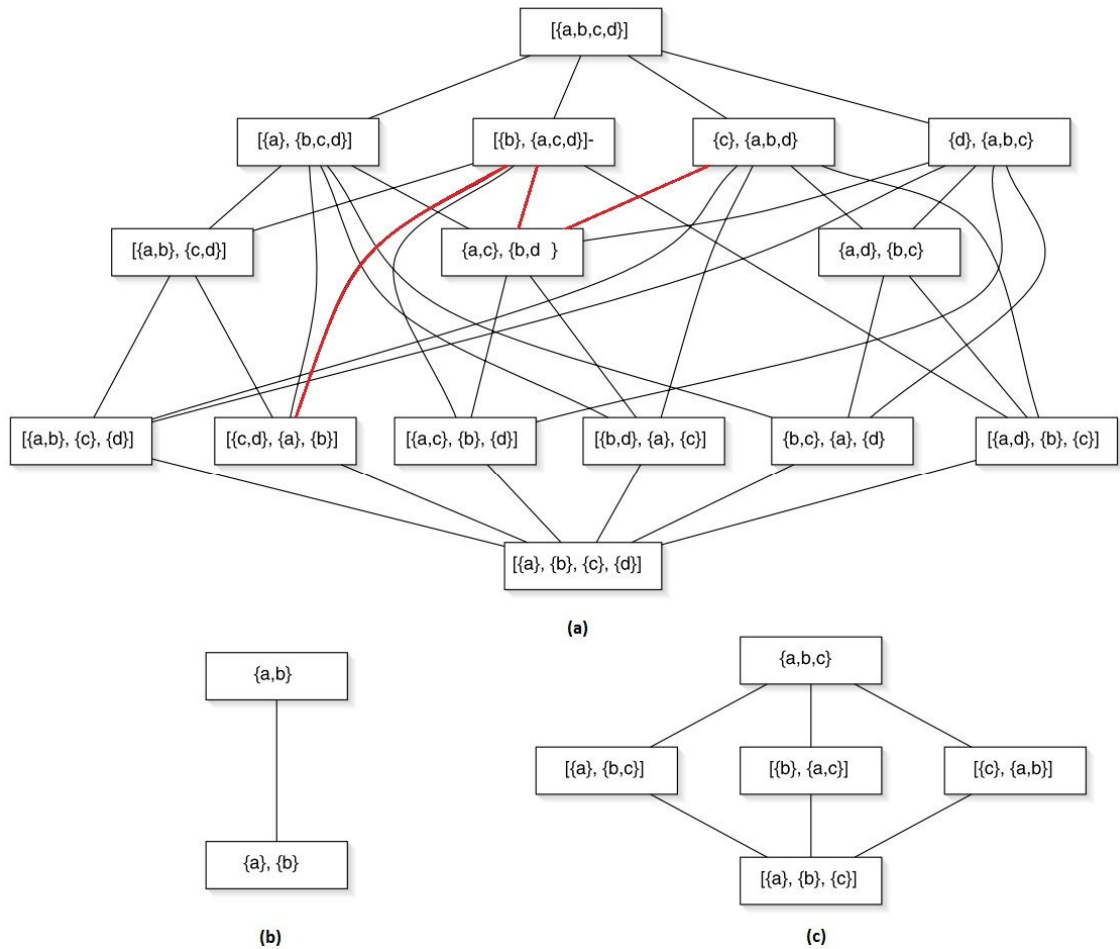


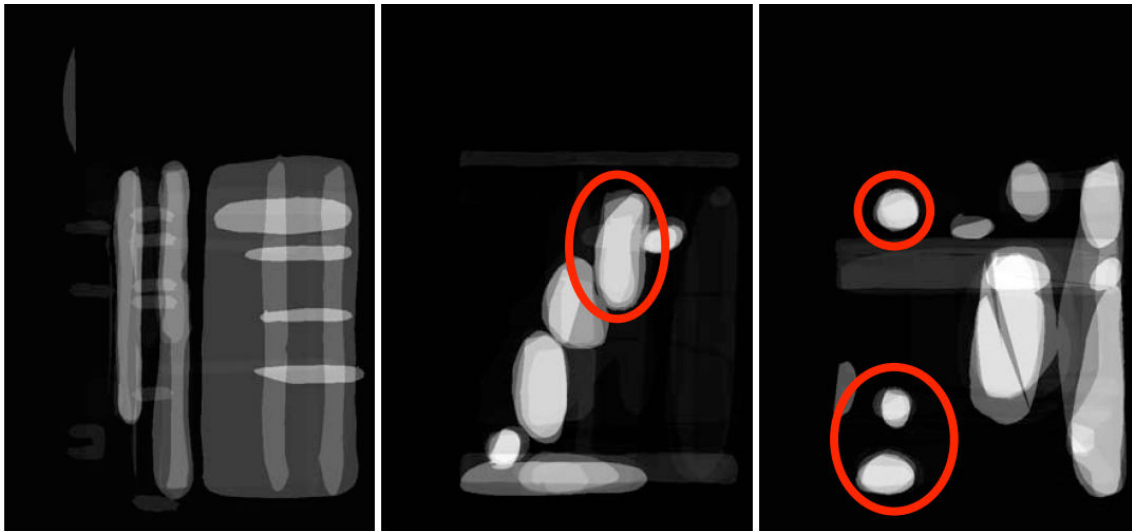
Figura 25 Estrutura de *lattice* para os conjuntos  $\{a,b,c,d\}$  (a),  $\{a,b\}$  (b) e  $\{a,b,c\}$  (c).

### 2.5.2 Experimento de Henry e Fekete (2006)

Henry e Fekete (2006) realizaram um experimento com usuários para entender como o *layout* de dados tabulares afeta o entendimento do usuário na exploração dos dados. Foram apresentados aos usuários três tabelas, cada uma com o mesmo conjunto de dados, mas ordenadas por três algoritmos diferentes: ordenação alfabética, ordenação por dendrograma (Clusterização Hierárquica) e ordenação baseada no PCV (Seção 2.3.5). As tabelas foram apresentadas aos usuários em papel digital<sup>8</sup>, assim, as anotações e respostas dos usuários foram facilmente capturadas.

<sup>8</sup> Usado em conjunto com uma caneta digital, com isso, todos os movimentos feitos com a caneta são enviados para um computador.

Os usuários deveriam responder um questionário de tarefas; algumas das tarefas requeriam que usuários circulassem algum *cluster*, *outlier* ou encontrassem tendências. Com isso, foi possível comparar as respostas dos usuários em matrizes ordenadas por diferentes algoritmos. A Figura 26 ilustra como eram os resultados de uma tarefa; com os dados capturados em um papel digital é possível observar todos os “círculos” desenhados pelos usuários na matriz (algumas tarefas pediam que usuários circulassem grupos de dados semelhantes).



**Figura 26** Grupos de dados semelhantes identificados pelos usuários. Na esquerda, grupos em uma matriz ordenada alfabeticamente; no meio, matriz ordenada por clusterização hierárquica; e na direita, matriz ordenada pela a solução do PCV. Todas as tabelas contém os mesmos dados. Figura extraída de Henry e Fekete (2006), p. 4.

Henry e Fekete (2006) citam algumas lições aprendidas com o experimento que podem ser úteis para avaliações dados em estruturas tabulares com usuários, a saber:

- Não use conjunto de dados reais. Henry e Fekete (2006) não geraram seus próprios dados conjuntos de dados; ao invés disso, utilizaram dados reais; e encontraram problemas com isso. Segundo os autores, com dados reais é difícil determinar a estrutura dos dados, “a verdade” a respeito dos dados. Uma abordagem sugerida é usar dados reais modificados para aumentar o número de padrões e tendências.
- Não faça muitas perguntas. Segundo os autores é importante manter o usuário envolvido com a análise dos dados; e um questionário longo com muitas perguntas não a ajuda atingir esse objetivo.

- Não exclua comentários livres e perguntas abertas. Apesar de comentários e perguntas abertas serem importantes, são difíceis de avaliar e classificar. A solução adotada foi realizar um experimento piloto com perguntas abertas. Baseado nas respostas das perguntas abertas, foram elaboradas alternativas para cada pergunta; assim, na execução do experimento de fato foram fornecidos aos usuários perguntas abertas (com espaço para escrever) e na sequência alternativas. Com isso, conforme cita Henry e Fekete (2006), a análise fica mais fácil e ainda fica disponível para consulta a pergunta aberta, caso seja necessária.

## 2.6 Ferramentas

Como visto na Seção 2.4, o processo de avaliação da qualidade de uma reordenação em uma matriz envolve a avaliações com usuários ou aplicação de funções de avaliação. Para este último, em matrizes grandes, como em matrizes 1000x1000, o tempo gasto com avaliações calculadas manualmente – sem o uso de alguma ferramenta – é impraticável. Além disso, quando se avalia a qualidade das reordenações de um algoritmo, uma função de avaliação não é aplicada a apenas uma matriz, mas a um conjunto de matrizes, com o objetivo de obter respostas com maior relevância estatística. Por isso, neste trabalho, em que algoritmos são comparados, foram realizados testes com conjuntos de matrizes.

Para definir que um algoritmo de reordenação de matrizes A é significativamente melhor que um algoritmo B, segundo uma função de avaliação  $F_I$ , por exemplo, é preciso analisar o resultado da aplicação da função  $F_I$  em N matrizes reordenadas pelo algoritmo A e N matrizes reordenadas pelo algoritmo B; e, com uso de recursos estatísticos, em especial, teste de hipóteses (MAGALHAES & LIMA, 2002), é possível obter respostas mais precisas sobre a comparação entre a qualidade de reordenação dos algoritmos A e B.

Assim, devido ao grande número de matrizes envolvidas no estudo comparativo entre algoritmos de seriação, foram empregadas ferramentas que automatizam a aplicação de funções de avaliação. Entretanto, a decisão de qual ferramenta utilizar foi baseada em mais outro requisito: a capacidade de reordenar matrizes. Conforme elucidado no parágrafo anterior, para

comparar dois algoritmos segundo funções de avaliação, é necessário ter matrizes reordenadas pelos dois algoritmos.

Portanto, resumidamente, os principais recursos que nortearam a escolha da ferramenta foram: a capacidade de reordenar matrizes com os algoritmos citados na revisão bibliográfica, e capacidade de aplicar funções de avaliação.

Entre as ferramentas pesquisadas, apenas duas atendiam os requisitos. A ferramenta *Matrix Reordering Analyzer (MRA)* e o Pacote R (HAHSLER ET AL., 2010). A primeira contém a implementação dos algoritmos: Heurística Baricêntrica, *2D-Sort* e *PQR-Sort*; enquanto a segunda contém a implementação dos algoritmos BEA e Clusterização Hierárquica.

Para este trabalho, a ferramenta MRA foi adotada, pois contém a implementação do algoritmo *PQR-Sort*, sendo que um dos objetivos deste trabalho é melhorá-lo. Além disso, essa ferramenta foi adotada por trabalhos anteriores do grupo de pesquisa sobre o algoritmo *PQR-Sort* (SILVA, 2010; SILVA ET AL., 2011). Essa ferramenta começou a ser desenvolvida por Silva (2010) e recebeu contribuições de Silva et al. (2011). A Seção 2.6.1 apresenta as principais características de cada versão.

### **2.6.1 MRA - Matrix Reordering Analyzer**

A Ferramenta MRA começou a ser desenvolvida por Silva (2010)<sup>9</sup>; nesse trabalho, o autor desenvolveu algumas variações do algoritmo *PQR-Sort* e as comparou com o *PQR-Sort* original e com os algoritmos *2D-Sort* (Seção 2.3.2) e Heurística Baricêntrica (Seção 2.3.3). Para automatizar a comparação entre algoritmos, a ferramenta MRA foi desenvolvida.

Inicialmente, a ferramenta continha a implementação dos algoritmos de seriação *PQR-Sort*, *2D-Sort* e Sugiyama modificado (Heurística Baricêntrica). Também possuía duas funções de avaliação, criadas por Silva (2010): Índice de Compactação de Dados e Método de Todos os Vizinhos.

A comparação de algoritmos é feita pela ferramenta em três passos principais (também ilustrados na Figura 27):

---

<sup>9</sup> O nome da ferramenta não havia sido definido por Silva (2010). Portanto, aproveita-se a oportunidade de trabalhar com esta ferramenta para poder nomeá-la.



Definição dos parâmetros da comparação: número  $N$  de matrizes binárias a serem geradas, densidade e tamanho.

Após a geração das  $N$  matrizes com as densidades especificadas, todas as matrizes são reordenadas pelos algoritmos implementados na ferramenta: PQR-Sort, 2D-Sort e Heurística Baricêntrica. Esse passo gera  $3N$  matrizes.

Por fim, todas as matrizes são avaliadas pelas funções de avaliação; representadas por  $F_1$  e  $F_2$  na Figura 27. Um relatório é gerado com dados da comparação; para cada algoritmo, o relatório exhibe a média do tempo de execução e média de valores de cada função de avaliação.

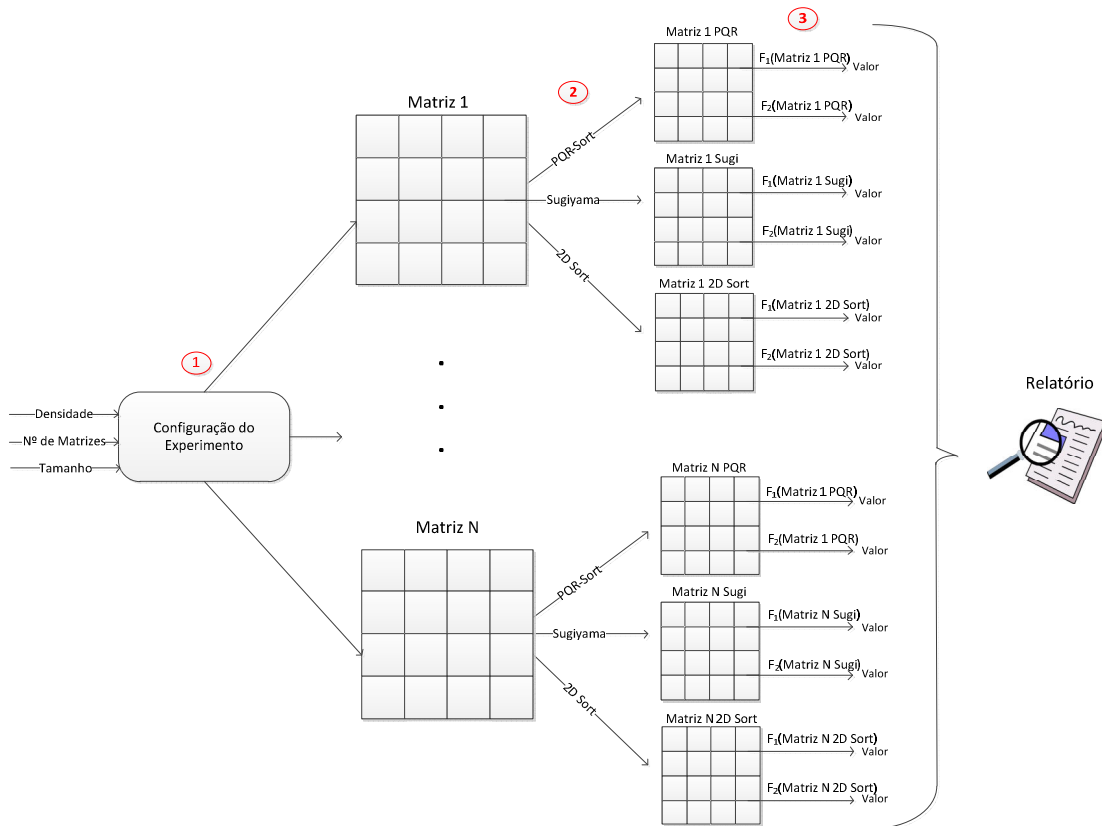


Figura 27 Esquema de uma comparação na ferramenta MRA (Silva, 2010).  $F_1$  e  $F_2$  indicam duas funções de avaliação.

Após o trabalho de Silva (2010), a ferramenta MRA recebeu contribuições de Silva et al. (2011). Foram implementadas novas funções de avaliação e o conceito de experimentos.

## 2.7 Experimentos

Na primeira versão da ferramenta, existia apenas uma maneira de comparar algoritmos: usando matrizes aleatórias. Definia-se o número de matrizes, o tamanho e a densidade, e em seguida, eram geradas matrizes aleatórias respeitando os parâmetros definidos.

Na versão de Silva et al. (2011) para a ferramenta MRA, foi criado o conceito de experimento. Em um experimento, é possível definir os mesmos parâmetros da versão anterior da ferramenta (número de matrizes, tamanho e densidade). Contudo, alguns parâmetros são definidos em conjuntos; define-se um conjunto  $T$  de tamanho de matrizes e um conjunto  $D$  de densidades. E, além disso, o experimento utiliza um novo parâmetro para especificar o algoritmo de geração da matriz, chamado de tipo de matriz. Em Silva (2010), as matrizes a serem reordenadas eram geradas aleatoriamente; em Silva et al. (2011), essa forma de gerar matrizes foi chamada de tipo de matriz *Random*. A criação de um novo tipo de parâmetro, o tipo de matriz, e a utilização de parâmetros em conjuntos aumentou a abrangência dos testes, visto que os algoritmos passaram a ser testados com matrizes de diferentes características.

Ademais, com as contribuições de Silva et al. (2011), foi criado um novo tipo de matriz, chamada de *Rectnoise*. O objetivo do novo tipo de matriz é gerar matrizes binárias com 10 blocos quadrangulares de tamanho aleatório, contendo valores 1. Esse tipo de matriz possui como parâmetro uma densidade ( $0 \leq d < 1$ ); ela define a porcentagem de valores da matriz que serão invertidos, ou seja, define uma porcentagem de ruído. Por exemplo, ao gerar uma matriz 10x10 do tipo *Rectnoise* com ruído 0.4, 40% das entradas da matriz terão seus valores invertidos (quando  $d=0$ , nenhum ruído é aplicado). A Figura 28 ilustra matrizes do tipo *Rectnoise*.

Com o *Rectnoise*, é possível testar os algoritmos de seriação com matrizes com características diferentes das geradas pelo tipo de matriz *Random*. Enquanto o *Random* gera matrizes aleatórias e sem garantias de que existirão similaridades altas entre linhas e colunas, o *Rectnoise*, por outro lado, gera matrizes com um nível maior de similaridade entre linhas e colunas.

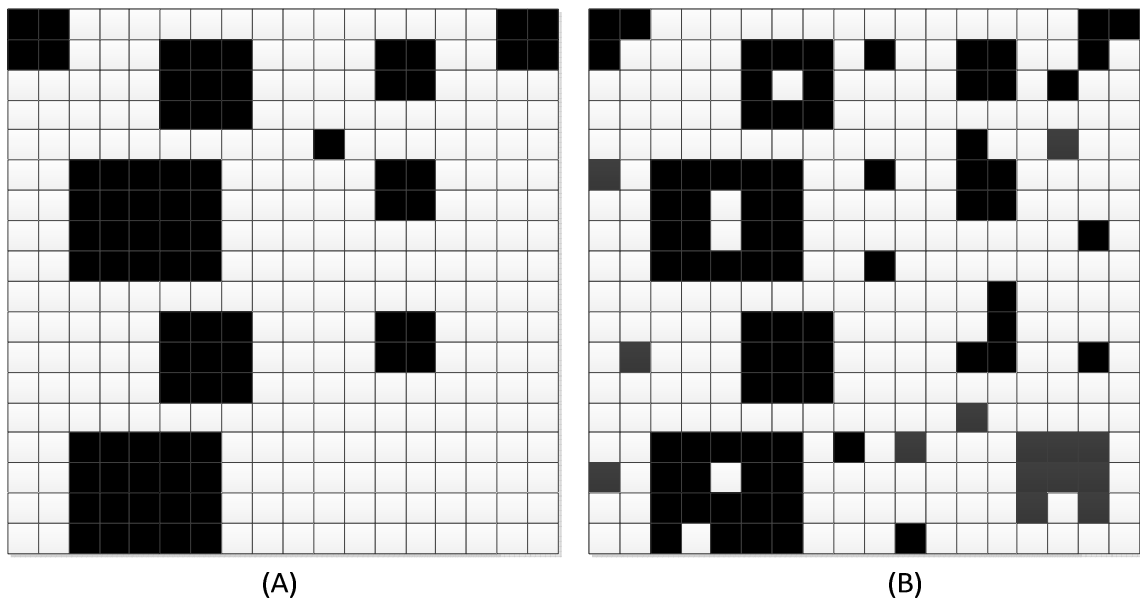


Figura 28 Matrizes do tipo Rectnoise. Em (A) uma matriz sem ruído ( $d=0$ ) e em (B) uma matriz com ruído.

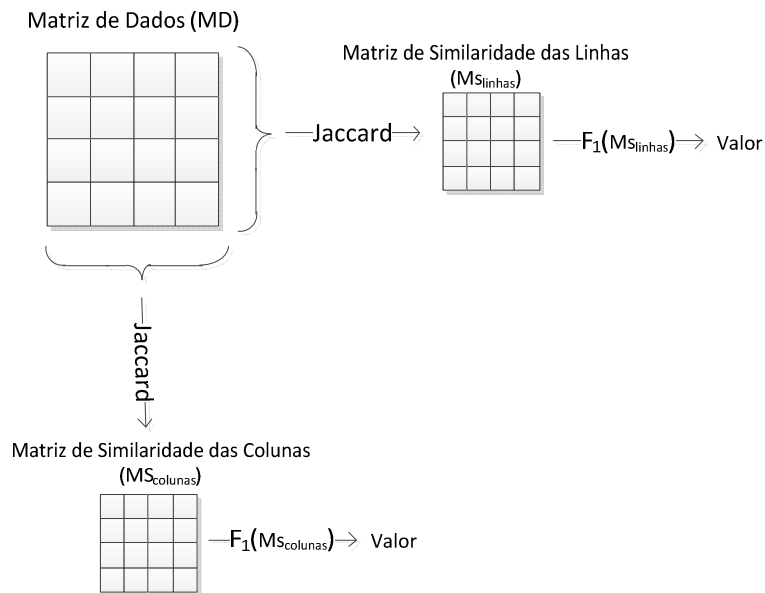
## 2.8 Funções de Avaliação

Além da criação de um novo tipo de matriz, em Silva et al. (2011) foram implementadas duas funções de avaliação indicadas na literatura: *Anti-Robinson Loss Function* (Seção 2.4.2) e *Minimal Span Loss Function* (Seção 2.4.3). Mas, como essas funções de avaliação são aplicadas a matrizes de similaridade ou dissimilaridade, e as matrizes geradas pelos algoritmos de seriação em Silva et al. (2011) – e também neste trabalho - são matrizes de dados, foi preciso gerar matrizes de similaridade a partir das matrizes de dados originais.

Para gerar matrizes de similaridade foram empregados dois coeficientes de similaridade, *Jaccard* e *Simple Matching* (COX & COX, 2008). Para cada matriz de dados foram geradas duas matrizes de similaridade, uma para linhas e outra para colunas. A Figura 29 ilustra a geração de matrizes de similaridade a partir de uma matriz de dados, e a posterior aplicação da função de avaliação  $F_1$  para matrizes de similaridade.

A geração de matrizes de similaridades ilustrada na Figura 29 também é exibida na Figura 30, que ilustra o esquema completo de um experimento após as modificações de Silva et al. (2011). Os passos 1 e 2 (ilustrados por círculos vermelhos na Figura 30) são iguais aos da versão anterior da ferramenta. A principal modificação foi a geração de matrizes de similaridade a partir

de matrizes de dados, com o objetivo de possibilitar a aplicação de funções de avaliação clássicas da área de seriação de matrizes. Essa modificação ocorre no passo 3, em que é utilizado um coeficiente de similaridade, e, no passo 4, em que as matrizes são avaliadas por  $f_2$ , que representa uma função de avaliação para matrizes de similaridade.



**Figura 29** Conversão de matrizes de dados em matrizes de similaridade. Cada matriz de dados gera duas matrizes de similaridade, uma para linhas e outra para colunas. Na figura, o coeficiente de similaridade empregado foi *Jaccard*; e  $F_1$  indica uma função de avaliação de matrizes de similaridade.

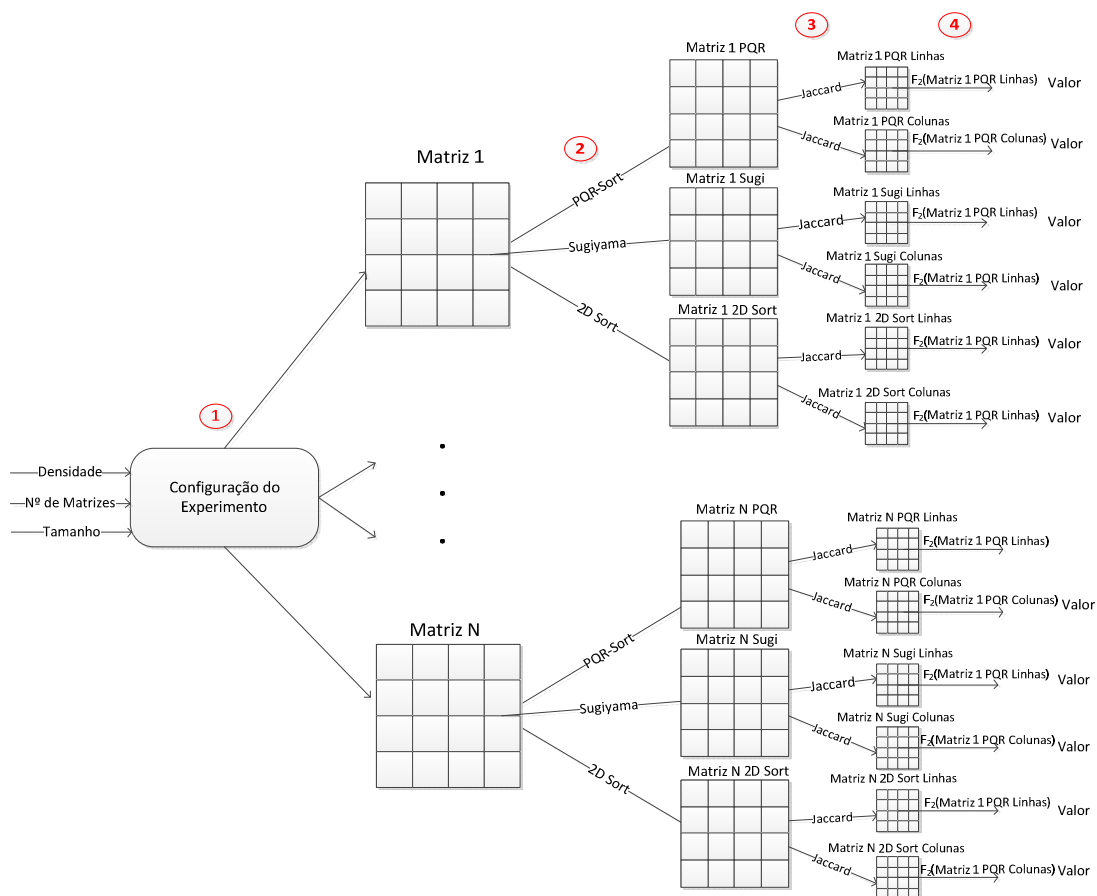


Figura 30 Esquema de um experimento após as modificações de Silva et al. (2011). A principal modificação em relação à versão anterior foi conversão de matrizes de dados em matrizes de similaridade (passo 3). Isso permitiu que os algoritmos de seriação fossem testados com funções de avaliação clássicas da área de seriação de matrizes.

## 2.9 Limitações da Ferramenta MRA

Nas versões da ferramenta MRA de Silva (2010) e Silva et al. (2011), as principais limitações detectadas são a quantidade de relatórios gerados em um experimento e a ausência de análises estatísticas que permitam conclusões.

Nessas versões, os experimentos envolviam muitos testes. De fato, ao comparar algoritmos de reordenação de matrizes, é importante testá-los com matrizes de diferentes densidades, visto que, um dos algoritmos pode gerar bons resultados apenas em matrizes densas ou apenas em matrizes esparsas. Por isso, ao executar um experimento, os algoritmos eram testados com um conjunto  $D$  de densidades. O mesmo é válido para tamanhos de matrizes.

Contudo, apesar da execução de experimentos com muitas densidades e tamanhos aumentar a abrangência dos testes, ao mesmo tempo aumenta a complexidade na análise dos relatórios, pois vários deles são gerados.

Neste trabalho, a configuração mais utilizada nos experimentos com tipo de matriz *Random* foi  $D = \{0.1, 0.3 \text{ e } 0.5\}$  e  $T = \{10 \times 10, 100 \times 100, 400 \times 400\}$ . Nesse caso, o total de experimentos realizados é igual a todas as combinações da tupla  $\{D, T\}$ , que é, nesse caso, 9.

Já para um experimento com tipo de matriz *Rectnoise*, a configuração mais utilizada foi  $D = \{0, 0.1, 0.3 \text{ e } 0.5\}$  e  $T = \{10 \times 10, 100 \times 100, 400 \times 400\}$ . Isso gerava um total de 12 experimentos.

Somando os experimentos dos tipos de matriz *Random* e *Rectnoise*, ao comparar algoritmos era preciso realizar 21 experimentos. E, como nas versões da ferramenta MRA é gerado um relatório para cada tupla  $\{D, T, M\}$  (sendo M o tipo de Matriz), o total é de 21 relatórios. Com isso, a análise do desempenho de um algoritmo envolvia a análise de muitos dados em diferentes relatórios. Para verificar se um algoritmo tem o melhor tempo de execução médio – considerando todas as densidades, tamanhos de matrizes e tipos de matrizes - era preciso abrir e comparar dados de 21 relatórios.

Além disso, em cada um dos relatórios existiam muitos dados, o que dificultava a extração de conhecimento. Cada relatório era dividido, primeiramente, em funções de avaliações. Caso o experimento utilizasse  $n$  funções de avaliação, o relatório era dividido em  $n$  seções. Cada uma das  $n$  seções eram divididas em seções de coeficientes de similaridade. Se o experimento utilizar  $m$  coeficientes, cada seção de função de avaliação era dividida em mais  $m$  subseções. E, finalmente, em cada subseção de coeficiente são listados os valores calculados para as funções de avaliação para cada algoritmo de seriação. A Figura 31 ilustra a estrutura do relatório e a Figura 32 exibe um relatório gerado pela ferramenta.

Além disso, outra limitação da ferramenta é a ausência de análises estatísticas que permitam conclusões; apenas as médias e desvios padrões eram calculados. Porém, no contexto de análise comparativa entre algoritmos de reordenação de matrizes, o fato de um algoritmo A ter média de tempo de execução inferior ao algoritmo B, quando comparados com um dado conjunto de densidades, tamanhos e tipos de matrizes, não significa que o algoritmo A é melhor em tempo de execução que o algoritmo B para esse conjunto de parâmetros. Em uma ferramenta de análise

comparativa entre algoritmos, um requisito importante é a possibilidade de empregar conceitos de estatística para obter respostas generalizadas e que permitam chegar a conclusões automaticamente.

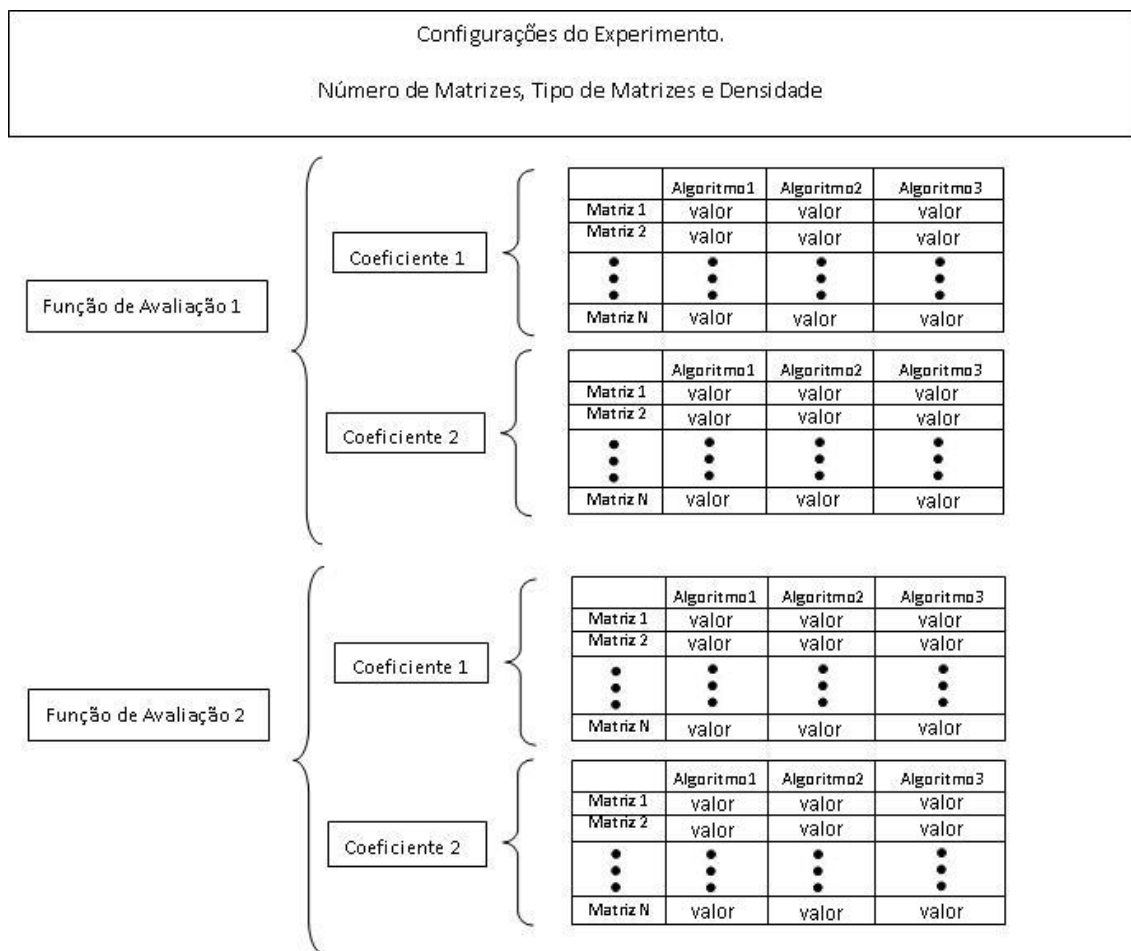


Figura 31 Formatação de um relatório gerado pela ferramenta MRA.

<b>Parameters:</b>		
Quantity:5 Dimensions: 100x100 Density: 0.1 Limit: 1 Maximum Value: 1 Binary: true		
<b>Tests Results:</b>		
Evaluation method:Minimal Span Loss Function		
Coefficient for similarity matrices:Jaccard Coefficient		
Matrix ID	PQR Sort (row)	PQR Sorted Restrictions (row)
m0	87.84582270493519	86.76502251757714
m1	88.25105858675067	87.17739487868133
m2	86.68345038146116	86.42149333250016
m3	87.9407343800693	87.40809500104886
m4	87.25349480672917	86.98663563069866
Mean:	87.5949121719891	86.95172827210124
Deviation:	0.624580979347229	0.379612535238266
Coefficient for similarity matrices:Simple Matching Coefficient		
Matrix ID	PQR Sort (row)	PQR Sorted Restrictions (row)
m0	15.990000000000002	15.649999999999997
m1	16.100000000000005	15.720000000000002
m2	15.670000000000005	15.500000000000005
m3	15.85	15.79
m4	15.759999999999994	15.73
Mean:		
Deviation:	0.17300288379192352	0.11122050136327744
Evaluation method:Anti-Robinson Loss Function		
Coefficient for similarity matrices:Jaccard Coefficient		
Matrix ID	PQR Sort (row)	PQR Sorted Restrictions (row)
m0	122752.0	113844.0
m1	128913.0	117562.0

Figura 32 Exemplo de um trecho de relatório gerado pela ferramenta MRA.

## 2.10 Considerações

Este capítulo apresentou algoritmos de seriação e funções para avaliar a qualidade de uma ordenação. Além disso, foram apresentados detalhes sobre a ferramenta a ser utilizada nesse trabalho: *Matrix Reordering Analyzer*. O próximo capítulo apresenta a teoria relacionada com os algoritmos criados para obter algoritmos superiores ao PQR-Sort em tempo de execução e/ou qualidade da reordenação.



### 3 Algoritmos desenvolvidos

Este capítulo descreve a teoria envolvida no processo de criação de uma nova versão do algoritmo *PQR-Sort*. O objetivo principal é gerar uma versão do *PQR-Sort* com menor tempo de execução e/ou com melhor resultado nas funções de avaliação.

A versão modificada que produziu melhores resultados em relação ao *PQR-Sort* Original foi o algoritmo chamado de *PQR-Sort with Sorted Restrictions* (resultados sobre a análise dos algoritmos criados estão descritos no Capítulo 5). Antes da obtenção dessa versão, outras tentativas foram feitas, ou seja, outros algoritmos foram criados. Apesar de algumas versões do algoritmo não atingirem o resultado esperado (ou seja, superar o *PQR-Sort*), elas foram descritas neste texto, pois cada algoritmo criado parte de suposições que foram testadas, e, estas podem indicar caminhos a serem seguidos ou não em trabalhos futuros. Ademais, alguns dos algoritmos produzidos podem não ter obtido bons resultados neste trabalho, que usou um conjunto de densidades, tamanhos, tipos de matrizes, coeficientes e funções da avaliação; mas, talvez, analisando esses algoritmos com diferentes configurações, tipos de matrizes e/ou coeficientes, por exemplo, o resultado pode ser diferente e os algoritmos podem ser considerados úteis para a área de seriação de matrizes.

A Seção 3.1 categoriza os algoritmos para melhorar o *PQR-Sort*, e, as subseções da Seção 3.1 descrevem os algoritmos propostos; por fim, a Seção 3.1.4 descreve a versão que supera o algoritmo *PQR-Sort*, o *PQR-Sort with Sorted Restrictions*.

#### 3.1 Fases de algoritmos para melhorar o *PQR-Sort*

No contexto deste trabalho, para melhorar o algoritmo *PQR-Sort*, foram consideradas três fases: Formação das Restrições, Construção da Árvore e Processamento das Restrições, ilustradas em vermelho na Figura 33 junto a um esquema do algoritmo *PQR-Sort*. Cada algoritmo desta seção está em pelo menos uma dessas categorias.

Os algoritmos incluídos na fase de **Formação das Restrições** atuam antes de restrições serem processadas pelo algoritmo de construção da árvore PQR. PQR *Smallest Resctictions* (Seção 3.1.2) é um exemplo de algoritmo que atua na fase de Formação de Restrições.

Na fase **Construção da Árvore** estão as abordagens que atuam diretamente com o algoritmo de construção das árvores PQR. Exemplos típicos de algoritmos dessa categoria são aqueles que atuam à medida que uma restrição é incorporada à árvore. O PQ-*Sort* (Seção 3.1.3), por exemplo, analisa a árvore PQR assim que cada restrição é incorporada a ela; se a restrição gerar um nó R, esta restrição é descartada.

A fase **Processamento das Permutações** agrupa algoritmos que atuam após a execução completa do PQR-*Sort*; são algoritmos que atuam nas permutações. Um exemplo de algoritmo nessa categoria é o PQR-*Sort* + BC (Seção 3.1.1), que aplica o algoritmo de Heurística Baricêntrica após a execução do PQR-*Sort*.

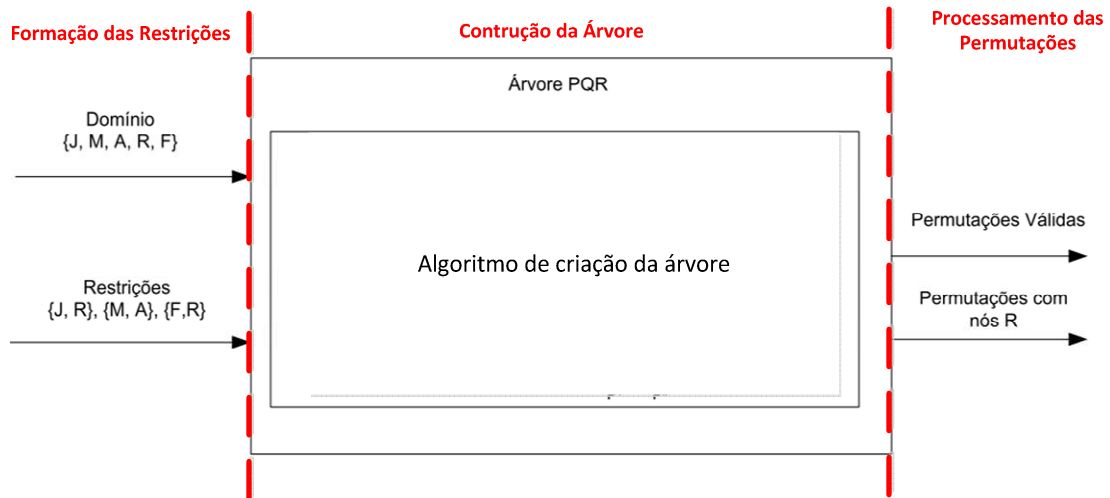


Figura 33 Esquema do algoritmo do PQR-*Sort* para uma dimensão (linha ou coluna) exibindo as entradas: domínio e restrições e as saídas: Permutações válidas ou Permutações com nós R. As três regiões da figuram representam as categorias de abordagens para melhorar o PQR-*Sort*: Formação de Restrições, Construção da Árvore e Processamento das Permutações.

### 3.1.1 PQR-*Sort* + BC

O PQR-*Sort* + BC atua na fase de Processamento das Permutações. Este algoritmo consiste em primeiramente reordenar uma dada matriz de dados pelo algoritmo PQR-*Sort*, e em seguida,

reordenar a matriz resultante por meio do algoritmo de Sugiyama Modificado (Heurística Baricêntrica). Um dos possíveis problemas com esse algoritmo é o tempo de execução, visto que dois algoritmos de reordenação de matrizes são utilizados integralmente. Resultados relativos a tempo de execução desse algoritmo e dos demais deste capítulo estão descritos no Capítulo 5.

### 3.1.2 PQR Smallest Restrictions

O algoritmo PQR *Smallest Restrictions* atua na Formação de Restrições. Esse algoritmo é idêntico ao PQR-*Sort*, exceto pelo acréscimo de um parâmetro: um tamanho limite  $x$  - em porcentagem - para as restrições, de tal modo que restrições maiores que esse limite sejam descartadas. Assim, dada uma matriz binária com  $m$  linhas e  $n$  colunas e considerando  $RL_m$  e  $RC_n$  como o conjunto de todas as restrições de linhas e colunas respectivamente, para cada restrição dos conjuntos  $RL_m$  e  $RC_n$ :

- a) Seja  $r_i \in RL_m, 0 \leq i \leq m$ . Se  $TAMANHO(r_i) < mx$  então descarte  $r_i$
- b) Seja  $r_j \in RC_n, 0 \leq j \leq n$ . Se  $TAMANHO(r_j) < nx$  então descarte  $r_j$

A hipótese considerada nesse algoritmo é que restrições grandes teriam maiores chances de formar nós R. Pois, por envolverem mais linhas e colunas, estatisticamente, teriam uma probabilidade maior de colidir com outras restrições e, portanto, gerar nós R.

### 3.1.3 PQ-Sort

O algoritmo PQ-*Sort* (SILVA, 2010) atua em duas fases do modelo proposto na Seção 3.1, Formação de Restrições e Construção da Árvore.

Na fase de Formação de Restrições, todas as restrições extraídas são reordenadas de forma crescente, segundo dois critérios: repetições de uma mesma restrição e tamanho das restrições; é possível atribuir pesos diferentes para cada critério, os quais influenciam na reordenação.

Com isso, restrições menores e com muitas repetições são processadas primeiramente. Quando as restrições menores têm prioridade, o efeito na árvore PQR é o retardamento do aparecimento dos nós R. Isso é uma vantagem porque todas as árvores em que ocorrem inserções após o aparecimento de nós R têm menores chances de gerar boas permutações. E, além disso, o

algoritmo também prioriza restrições repetidas; isso força que as restrições com repetições – que podem envolver muitas linhas ou colunas – sejam formadas antes dos nós R.

Após a fase de Formação de Restrições, o algoritmo *PQ-Sort* também atua na fase de Construção da Árvore. Sempre que uma restrição é incluída na árvore PQR, o algoritmo verifica se foram gerados nós R. Em caso afirmativo, a restrição é excluída, descarta-se a árvore PQR e o algoritmo continua com uma versão anterior da árvore (a versão anterior à inclusão da restrição que gerou o nó R). Com essa regra, que praticamente equivale a uma operação de “*Undo*” na inserção, a árvore final nunca terá nós R; por isso o nome *PQ-Sort*.

### **3.1.4 PQR-Sort with Sorted Restrictions**

O algoritmo *PQR-Sort with Sorted Restrictions* foi inspirado no algoritmo *PQ-Sort*. Contudo, ao invés de atuar na fase de Formação das Restrições e Construção da Árvore, atua apenas na primeira fase.

O algoritmo *PQR-Sort Restrictions* não descarta restrições que gerem nós R nas árvores PQR, e portanto não atua na fase de Construção da árvore.

Na fase de Formação das Restrições, o *PQR-Sort with Sorted Restrictions* reordena restrições com base apenas no critério de tamanho das restrições.

Com isso, o *PQR-Sort with Sorted Restrictions* baseia-se na mesma hipótese que o *PQ-Sort*: restrições grandes teriam maiores probabilidades de gerar nós R. Outro ponto importante é que o algoritmo não exclui restrições que produziram nós R. Os bons resultados do algoritmo *PQR-Sort with Sorted Restrictions* – apresentados na Seção 5.1.4- mostraram que, mesmo nós R sendo indesejados em uma árvore, é melhor ter uma restrição que gerou um nó R na árvore, do que descartá-la; visto que, a exclusão das restrições que geraram nós R, pode levar a exclusão de um grande número de restrições, ou seja, estariam sendo excluídas informações importantes de linhas e colunas que devem permanecer unidas.

## **3.2 Conclusão**

Este capítulo apresentou a teoria relacionada aos algoritmos de reordenação de matrizes desenvolvidos. O próximo capítulo apresentará as melhorias implementadas na Ferramenta MRA para possibilitar testes com os algoritmos desenvolvidos.



## 4 Ferramenta

Como visto na Seção 2.6, as versões existentes da ferramenta MRA têm duas limitações principais que a impediam de ser empregada neste trabalho em sua forma até então disponível: a ausência de análises estatísticas e o excesso de relatórios gerados. Este capítulo descreve como essas limitações foram superadas.

Para lidar com a ausência de testes estatísticos foram implementados na ferramenta testes de hipóteses (MAGALHAES & LIMA, 2002) para obter respostas mais precisas sobre o desempenho dos algoritmos, como descrito na Seção 4.1. Para lidar com o excesso de relatórios foi proposta uma sumarização dos relatórios, descrita na Seção 4.2.

### 4.1 Teste de Hipóteses

Para melhorar a comparação entre algoritmos de reordenação, foram utilizados testes de hipóteses. Com isso, em um experimento, dado um nível de confiança, é possível determinar se um algoritmo é significativamente melhor que outro.

Para o teste de hipóteses, os algoritmos são analisados em pares. Dados dois algoritmos A e B, definiu-se uma hipótese nula segundo a qual esses algoritmos, ao serem executados para reordenar a mesma matriz, geram matrizes reordenadas cujos resultados de uma dada função de avaliação não apresentam uma diferença significativa. Essa hipótese é definida como  $D = 0$ , onde  $D = X - Y$ , e  $X = F(M, A)$  e  $Y = F(M, B)$  são os valores da aplicação de uma função de avaliação  $F$  às matrizes resultantes da reordenação de uma matriz  $M$  pelos algoritmos A e B, respectivamente.

A hipótese alternativa é  $D \neq 0$ , o que implica que  $X \neq Y$ . Nesse caso, como neste trabalho foram empregadas apenas funções de avaliação de perda (nas quais quanto menor o resultado, melhor a reordenação), pode-se definir que o melhor algoritmo entre A e B é definido pelo sinal de D: se  $D < 0$ , portanto  $X < Y$ , e assim A é o melhor algoritmo; se  $D > 0$ , B é o melhor algoritmo.

Nos testes de hipóteses com tipos de matrizes *Random* e *Rectnoise*, a variância real dos tempos de execução e valores de funções de avaliação não é previamente conhecida. Ao realizar

um experimento na Ferramenta MRA tem-se apenas a variância amostral. Pois, em uma matriz binária 400x400, por exemplo, seria necessário gerar  $2^{(400 \times 400)}$  matrizes e aplicar funções de avaliação em cada uma delas, para então conhecer a variância real (da população) dos valores de funções de avaliação. Ademais, os experimentos envolvem densidades, tamanhos e tipos de matrizes diferentes, o que inviabilizaria calcular a variância real para cada tipo de matriz a ser testado.

Assim, segundo citam Magalhaes & Lima (2002), para testes em que a variância real não é conhecida, é possível empregar o teste  $t$  de *student* (que utiliza um estimador para a variância real). Dessa forma, com base em dados amostrais, média, desvio padrão e o parâmetro de um teste  $t$  de *student* - o grau de liberdade (denotado por  $t_{(n-1)}$ , em que  $n$  é o tamanho da amostra) - é possível obter os limites da região crítica da hipótese nula, ou seja, a região em que a hipótese alternativa é rejeitada. (Os valores de  $t$  podem ser encontrados no Anexo B).

Por exemplo, suponha um conjunto de 30 matrizes que foram avaliadas por uma função de avaliação com objetivo de comparar algoritmos, usando nível de confiança de 1%. A partir do Anexo B obtém-se que  $t_{(30-1)}=t_{29} = 2,756$ ; logo, a região crítica é  $RC = \{x \in \mathbb{R} : -2,756 < x < 2,756\}$ .

No caso de um teste  $t$  de *student*, para verificar se um par de algoritmos não está na região crítica, ou seja, verificar se eles produzem resultados significativamente diferentes para a função de avaliação, é preciso calcular  $T$ , um valor calculado com base nos dados amostrais, usando a fórmula:

$$T = \frac{(\bar{D} - \mu_D)\sqrt{n}}{S_D}$$

Em que  $D_i=X_i-Y_i$ , e  $X_i$  e  $Y_i$  são os valores da aplicação de uma função de avaliação  $F$  em matrizes geradas por dois algoritmos para a reordenação de uma matriz  $i$ . O termo  $n$  indica a quantidade de dados amostrais. Já o termo  $\mu_D$  representa a média da distribuição normal formada pelos valores de  $D$ . No contexto deste trabalho, as hipóteses são testadas a partir da hipótese nula; portanto, para todos os casos, inicialmente assume-se que  $\mu_D = 0$  para então verificar se essa hipótese é verdadeira. O termo  $S_D$  representa a variância dos valores de  $D$ . Assim, basta verificar se  $T$  pertence à RC; em caso afirmativo, os resultados da função de avaliação para os algoritmos



são considerados semelhantes comprovando portanto a hipótese nula; caso contrário, são considerados diferentes e o melhor algoritmo é determinado pelo sinal de  $T$ .

Como exemplo, Figura 34 a exibe o trecho de um relatório gerado pela Ferramenta MRA que usa testes de hipóteses para comparar dois algoritmos, *PQR-Sort* e *PQR-Sort with Sorted Restrictions* (esse algoritmo é referenciado na Figura 34 como *PQR-Sorted Restrictions*). Nesse teste, foram utilizadas 100 matrizes e nível de confiança de 1%. Para a primeira linha da Figura 34, não existe um melhor algoritmo, porque a região crítica calculada é  $RC = \{x \in \mathbb{R} : 2,756 < x < 2,756\}$ , e  $T=1,442$  está dentro da  $RC$ ; logo, a hipótese nula é aceita. Na segunda linha, existe um melhor algoritmo, pois o  $T=15,166$  está fora da  $RC$ ; o melhor algoritmo nesse caso é o *PQR-Sort with Sorted Restrictions*, porque o valor de  $T$  está fora do  $RC$  e é positivo, indicando que as matrizes reordenadas pelo algoritmo *PQR-Sort with Sorted Restrictions* geraram valores menores de função de avaliação.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR – PQR-Sorted Restrictions	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Média	0.2639999999999999	?	?
					St. Dev	1.8295967431169755		
					T	1.4429409157683448		
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Média	13481.92	PQR SortedRestrictions	PQR
					St. Dev	8889.478397260074		
					T	15.16615418532927		

Desvio Padrão Amostral ←

Média amostral →

T amostral ↓

Figura 34 Trecho de um relatório de funções de avaliação gerado pela ferramenta MRA. A ausência de um melhor ou pior algoritmo é denotada por ‘?’ (caso em que a hipótese nula é aceita).

Apesar do teste de hipóteses ser aplicado a pares de algoritmos, a ferramenta MRA permite a comparação entre vários algoritmos. Nesse caso, o melhor algoritmo é aquele que foi considerado o melhor em todas as comparações entre pares; caso não exista um algoritmo que satisfaça tal condição, a não existência de um melhor algoritmo é indicada por uma interrogação no relatório produzido pela ferramenta MRA.

## 4.2 Sumarização de Relatórios

Nas versões anteriores da ferramenta MRA, experimentos com tipo de matriz *Rectnoise* e *Random* com 4 densidades e 3 tamanhos de matriz geram um total de 24 relatórios. Para reduzir o número de relatórios foram propostos três relatórios para cada experimento.<sup>10</sup>

Dos três relatórios, dois exibem dados de funções de avaliação: relatório de funções de avaliação e relatório de funções de avaliação sumarizado. O primeiro contém mais detalhes estatísticos dos testes de hipóteses (média, desvio padrão e  $T$ ), enquanto o segundo contém menos detalhes e facilita análises rápidas dos resultados. O terceiro relatório gerado exibe informações sobre o tempo de execução dos algoritmos. As Figuras 35, 36 e 37 ilustram, respectivamente, os três relatórios gerados pela ferramenta.

Portanto, com o uso de teste de hipóteses e a sumarização de relatórios, facilitou-se a análise de algoritmos, visto que, em apenas um arquivo, é possível visualizar o melhor e o pior algoritmo de um experimento. Além disso, o teste de hipóteses permitiu a redução do número de relatórios, pois os dados passaram a ser mais processados: antes, eram exibidos os valores de funções de avaliação para cada uma das matrizes geradas para um experimento; com os novos relatórios, essa informação é substituída pela informação de qual o melhor e o pior algoritmo.

---

<sup>10</sup> Os relatórios existentes nas versões anteriores da ferramenta MRA foram mantidos, pois apesar de numerosos, são úteis em análises mais detalhadas.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	Sugi - PQR + BC	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	3.59E-001		
					St. Dev	6.95E-001		
					T	1.63E+000	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	2.55E-001		
					St. Dev	3.47E-001		
					T	2.32E+000	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	6.41E+000		
					St. Dev	1.13E+001		
					T	1.80E+000	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	7.97E+000		
					St. Dev	1.74E+001		
					T	1.45E+000	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	1.69E-001		
					St. Dev	5.48E-001		
					T	9.74E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	1.72E-001		
					St. Dev	2.63E-001		
					T	2.07E+000	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.45E+000		
					St. Dev	3.56E+000		
					T	2.17E+000	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.35E+001		
					St. Dev	1.61E+001		
					T	2.66E+000	PQR-Sort + BC Sugiyama	
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.39E-002		
					St. Dev	5.59E-001		
					T	-1.91E-001	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-6.66E-018		
					St. Dev	4.47E-001		
					T	-4.72E-017	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-2.49E+000		
					St. Dev	1.76E+001		
					T	-4.47E-001	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.49E+000		
					St. Dev	1.80E+001		
					T	2.61E-001	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.08E-001		
					St. Dev	6.61E-001		
					T	-5.15E-001	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-5.80E-002		
					St. Dev	4.62E-001		
					T	-3.97E-001	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	5.30E-001		
					St. Dev	1.83E+001		
					T	9.16E-002	?	?

Figura 35 Relatório de funções de avaliação com detalhes dos testes de hipóteses. Empate entre algoritmos são denotados por ‘?’.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 36 Relatório de funções de avaliação resumido.

<b>Matrix Type</b>	<b>Density</b>	<b>Dimensions</b>	<b>Best</b>	<b>Worst</b>
RECTNOISE	0.1	100x100	?	?
RANDOM_MATRIX	0.1	100x100	PQR Sort	PQR-Sorted Restrictions
RECTNOISE	0.3	100x100	?	?
RANDOM_MATRIX	0.3	100x100	?	?
RECTNOISE	0.5	100x100	?	?
RANDOM_MATRIX	0.5	100x100	?	?
RECTNOISE	0.0	100x100	?	?

Figura 37 Relatório de tempo de execução.

### 4.3 Considerações Finais

Este capítulo apresentou as melhorias implementadas na ferramenta MRA, sumarização de relatórios e testes de hipóteses. Tais melhorias permitiram que os testes com os algoritmos desenvolvidos - apresentados no próximo capítulo - produzissem relatórios com informações baseadas em testes estatísticos que facilitam a conclusão sobre o desempenho dos algoritmos.

## 5 Testes

Este capítulo apresenta testes feitos sobre os algoritmos propostos no Capítulo 3, com objetivo de produzir um algoritmo superior ao *PQR-Sort* em qualidade de reordenação e/ou tempo de execução. Por isso, dois tipos de testes foram executados: com matrizes sintéticas e com um conjunto de dados real.

Na Seção 5.1 os algoritmos propostos foram testados com matrizes sintéticas, ou seja, matrizes geradas pela ferramenta MRA (com as modificações descritas no Capítulo 4). Em especial, foram empregados dois tipos de matrizes, *Random* e *Rectnoise* (ver Seção 2.7). Neste capítulo, alguns dos algoritmos encontrados na revisão bibliográfica e os algoritmos propostos foram testados em relação ao algoritmo *PQR-Sort*, visto que um dos objetivos deste trabalho é gerar uma versão superior ao *PQR-Sort*, em tempo de execução e/ou qualidade da reordenação.

Durante os testes com matrizes sintéticas, descobriu-se que dois dos algoritmos propostos, *PQR-Sort + BC* e *PQR-Sort with Sorted Restrictions*, mostram-se com potencial para produzir ordenações que evidenciam padrões globais; enquanto que, *PQR-Sort*, *PQR-Sort with Sorted Restrictions* e *PQ-Sort* produziram ordenações que evidenciam padrões locais. Por isso, na Seção 5.1.6, foram conduzidos testes entre algoritmos com tendência a produzir boas ordenações locais e globais; este tipo de teste indicará bons algoritmos com base no tipo de padrão relevado pela reordenação, local e global.

Na Seção 5.2 os algoritmos propostos foram testados com um conjunto de dados real, o conjunto de dados Íris (FRANK & ASUNCION, 2010). Este conjunto de dados contém características de três espécies de flores: *iris-setosa*, *iris-versicolor* e *iris-virginica*. Com os algoritmos de seriação espera-se prover uma reordenação que auxilie a descoberta de conhecimento e extrações de padrões nesse conjunto de dados.

### 5.1 Algoritmos propostos

A seção descreve os resultados da comparação entre os algoritmos propostos por este trabalho no Capítulo 3 e o algoritmo *PQR-Sort*. A seguir, os algoritmos *PQR-Sort + BC*, *PQ-*

*Sort*, *PQR Smallest Restrictions* e *PQR-Sort with Sorted Restrictions* são comparados com o *PQR-Sort* com base em tempo de execução e funções de avaliação. Em todas as comparações desta seção foram padronizados os seguintes parâmetros:

- a) Nível de Confiança para testes de hipóteses: 1%;
- b) Número de Matrizes geradas: 100;
- c) Funções de avaliação: *Anti-Robinson Loss Function* e *Minimal Span Loss Function*;
- d) Coeficientes: *Jaccard* e *Simple Matching*
- e) Tipos de Matrizes: *Rectnoise* e *Random*
- f) Densidades: *Random* (0.1, 0.3 e 0.5) e *Rectnoise* (0, 0.1, 0.3 e 0.5)
- g) Tamanhos: 10x10 e 100x100.

Para os experimentos foram utilizadas as funções de avaliação: *Anti-Robinson Loss Function* e *Minimal Span Loss Function*. Elas foram selecionadas porque a primeira identifica a presença padrões globais; e, a segunda, faz o mesmo para padrões locais. Com isso, em todas as reordenações serão medidas a capacidade de evidenciar padrões e globais.

Para definir os coeficientes empregados no experimento, a classificação de simetria de variáveis binárias de Wu et. al. (2008) foi considerada. Segundo esses autores, uma variável binária é simétrica se seus estados são considerados equivalentes; ou seja, dadas duas possibilidades, nenhuma delas é mais relevante que a outra. Caso contrário, uma variável é considerada assimétrica e, portanto, um dos estados é mais raro e mais importante que o outro (em geral, isso ocorre em matrizes esparsas). E, conforme citam Wu et al. (2008), *Jaccard* é ideal para avaliar matrizes assimétricas e *Simple Matching* é ideal para avaliar matrizes simétricas; assim, tais coeficientes foram empregados para que os testes cubram os dois tipos de características.

### **5.1.1 PQR-Sort + BC**

O resultado das comparações entre *PQR-Sort* e *PQR-Sort + BC* está exibido nas Figuras 38 e 39; elas apresentam a comparação com base em funções de avaliação usando matrizes 10x10 e 100x100, respectivamente.

Nas matrizes 10x10, o PQR-*Sort* + BC foi significativamente melhor que o PQR-*Sort* em 7,14% dos casos. Os resultados dos algoritmos foram considerados semelhantes em 92,85% dos testes; o PQR-*Sort* não foi considerado o melhor em nenhum dos casos. Já para matrizes 100x100, o PQR-*Sort* + BC foi melhor em 46,43% dos casos, e o PQR-*Sort* foi melhor em 50% dos casos.

Um dos padrões verificados nos experimentos foi a tendência do PQR-*Sort* a gerar bons resultados para a função de avaliação *Minimal Span Loss Function*. Em matrizes 100x100, o PQR-*Sort* foi considerado o melhor em 100% dos casos com essa função. Por outro lado, o algoritmo PQR-*Sort* + BC tende a gerar bons resultados para a função de avaliação *Anti-Robinson Loss Function*; com essa função, nas matrizes 100x100, o algoritmo PQR-*Sort* + BC foi considerado o melhor em 92,85% dos casos. Esse resultado indica que o PQR-*Sort* é um algoritmo que tente a gerar boas reordenações para evidenciar padrões locais, e, o algoritmo PQR-*Sort* + BC, por outro lado, tente a gerar boas reordenações para evidenciar padrões globais.

Em relação a análises do tempo de execução do PQR-*Sort* e PQR-*Sort* + BC, para matrizes 100x100, o PQR-*Sort* foi melhor em todos os casos. Para matrizes 10x10, apenas houve empates. As Figuras 40 e 41 ilustram os relatórios de tempo de execução. Em matrizes 100x100, o fato de o PQR-*Sort* + BC ser o pior em todos os casos deve-se à utilização integral de dois algoritmos; após a aplicação do PQR-*Sort* é aplicada Heurística Baricêntrica (BC).

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
<b>RANDOM_MATRIX</b>	<b>0.1</b>	<b>10x10</b>	<b>Minimal Span Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
<b>RECTNOISE</b>	<b>0</b>	<b>10x10</b>	<b>Minimal Span Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 38 Comparação baseada em funções de avaliação (*evaluators*) entre PQR-Sort e PQR-Sort + BC para matrizes 10x10. Os casos em que o PQR-Sort + BC foi considerado melhor estão grifados em verde.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sort + BC
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sort + BC
<b>RECTNOISE</b>	<b>0.1</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
<b>RECTNOISE</b>	<b>0.1</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Simple Matching Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sort + BC
<b>RANDOM_MATRIX</b>	<b>0.1</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
<b>RANDOM_MATRIX</b>	<b>0.1</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Simple Matching Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sort + BC
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sort + BC
<b>RECTNOISE</b>	<b>0.3</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
<b>RECTNOISE</b>	<b>0.3</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Simple Matching Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sort + BC
<b>RANDOM_MATRIX</b>	<b>0.3</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
<b>RANDOM_MATRIX</b>	<b>0.3</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Simple Matching Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sort + BC
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sort + BC
<b>RECTNOISE</b>	<b>0.5</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
<b>RECTNOISE</b>	<b>0.5</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Simple Matching Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sort + BC
<b>RANDOM_MATRIX</b>	<b>0.5</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
<b>RANDOM_MATRIX</b>	<b>0.5</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Simple Matching Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sort + BC
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sort + BC
<b>RECTNOISE</b>	<b>0</b>	<b>100x100</b>	<b>Anti-Robinson Loss Function</b>	<b>Jaccard Coefficient</b>	<b>PQR-Sort + BC</b>	<b>PQR Sort</b>
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 39 Comparação baseada em funções de avaliação (*evaluators*) entre PQR-Sort e PQR-Sort + BC para matrizes 100x100. Os casos em que o PQR-Sort + BC foi considerado melhor estão grifados em verde.



Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	10x10	?	?
RANDOM_MATRIX	0.1	10x10	?	?
RECTNOISE	0.3	10x10	?	?
RANDOM_MATRIX	0.3	10x10	?	?
RECTNOISE	0.5	10x10	?	?
RANDOM_MATRIX	0.5	10x10	?	?
RECTNOISE	0.0	10x10	?	?

Figura 40 Relatório de tempo de execução dos algoritmos PQR-*Sort* e PQR-*Sort* + BC em matrizes 10x10.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	100x100	PQR Sort	PQR- <i>Sort</i> + BC
RANDOM_MATRIX	0.1	100x100	PQR Sort	PQR- <i>Sort</i> + BC
RECTNOISE	0.3	100x100	PQR Sort	PQR- <i>Sort</i> + BC
RANDOM_MATRIX	0.3	100x100	PQR Sort	PQR- <i>Sort</i> + BC
RECTNOISE	0.5	100x100	PQR Sort	PQR- <i>Sort</i> + BC
RANDOM_MATRIX	0.5	100x100	PQR Sort	PQR- <i>Sort</i> + BC
RECTNOISE	0.0	100x100	PQR Sort	PQR- <i>Sort</i> + BC

Figura 41 Relatório de tempo de execução dos algoritmos PQR-*Sort* e PQR-*Sort* + BC em matrizes 100x100.

### 5.1.2 PQR Smallest Restrictions

A comparação entre PQR-*Smallest Restrictions* (usando valor limite 0.5)<sup>11</sup> e PQR-*Sort* é exibida nas Figuras 42 e 43. O resultado dos experimentos provou que a hipótese em que se baseia o PQR *Smallest Restrictions* não é verdadeira. Segundo as análises de funções de avaliação, a abordagem de eliminar restrições grandes não foi útil – PQR *Smallest Restrictions* não foi o melhor em nenhum caso (na maioria dos casos ocorrem empates). Isso ocorre provavelmente porque, ao excluir uma restrição grande, além de reduzir o número de nós R, está sendo eliminada muita informação de linhas (ou colunas) que devem permanecer unidas. Em alguns casos, restrições grandes que não gerariam nós R foram excluídas. Em outras palavras, a abordagem não teve um bom resultado porque sua desvantagem (eliminar informações de linhas e colunas) é maior que sua vantagem (tentar reduzir o número de nós R).

Na análise de tempo de execução (Figuras 44 e 45), em matrizes 10x10, PQR-*Sort* e PQR-*Smallest Restrictions* foram considerados equivalentes. Para matrizes 100x100, o PQR- *Smallest Restrictions* foi considerado mais rápido que o PQR-*Sort* em dois casos, ambos em matrizes de

<sup>11</sup> Nesta seção foi adotado valor limite 0.5 devido aos bons resultados nos testes, quando comparado com outros valores limite.

densidade 0.5. Isso ocorre porque o PQR-Sort *Smallest Restrictions* leva uma pequena vantagem no tempo de execução em matrizes densas, por processar um conjunto menor de restrições.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	PQR Sort	PQR Smallest Restrictions (0.5)
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 42 Comparação entre PQR-Sort e PQR-Smallest Restrictions (usando valor limite 0.5) com base em funções de avaliação em matrizes 10x10.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR Smallest Restrictions (0.5)
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR Sort	PQR Smallest Restrictions (0.5)
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR Sort	PQR Smallest Restrictions (0.5)
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 43 Comparação entre PQR-Sort e PQR-Smallest Restrictions(usando valor limite 0.5) com base em funções de avaliação em matrizes 100x100.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	10x10	?	?
RANDOM_MATRIX	0.1	10x10	?	?
RECTNOISE	0.3	10x10	?	?
RANDOM_MATRIX	0.3	10x10	?	?
RECTNOISE	0.5	10x10	?	?
RANDOM_MATRIX	0.5	10x10	?	?
RECTNOISE	0.0	10x10	?	?

Figura 44 Análise de tempo comparativa entre PQR-Sort e PQR-Smallest Restrictions(0.5) em matrizes 10x10.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	100x100	?	?
RANDOM_MATRIX	0.1	100x100	?	?
RECTNOISE	0.3	100x100	?	?
RANDOM_MATRIX	0.3	100x100	?	?
RECTNOISE	0.5	100x100	PQR Smallest Restrictions (0.5)	PQR Sort
RANDOM_MATRIX	0.5	100x100	PQR Smallest Restrictions (0.5)	PQR Sort
RECTNOISE	0.0	100x100	?	?

Figura 45 Análise de tempo comparativa entre PQR-Sort e PQR-Smallest Restrictions(0.5) em matrizes 100x100.

### 5.1.3 PQ-Sort

Os resultados das comparações entre PQR-Sort e PQ-Sort com base em funções de avaliação são exibidos nas Figuras 46 (matrizes 10x10) e 47 (matrizes 100x100). Para matrizes 10x10, os algoritmos foram considerados semelhantes em todos os casos. Já para matrizes 100x100, observa-se uma tendência do PQR-Sort em gerar bons resultados para o *evaluator* (Função de avaliação) *Minimal Span Loss Function*; o PQR-Sort foi considerado o melhor algoritmo em 100% dos casos com esse *evaluator*. Para o *evaluator Anti-Robinson Loss Function* o PQ-Sort foi o melhor em 42,85% dos casos e houve 57,14% de empates; os empates ocorreram em densidades 0.5 e as vitórias em densidades baixas, 0.1 e 0.3.

Na análise de tempo de execução dos algoritmos, o PQR-Sort foi melhor. Nas matrizes 10x10, venceu em 57,14% dos casos; e, nas matrizes 100x100, o PQR-Sort foi melhor em 100% dos casos. O resultado ruim do PQ-Sort no tempo de execução provavelmente deve-se à reordenação das restrições realizada na fase de Formação das Restrições e ao tempo gasto para verificar se uma restrição gerou nós R. Os relatórios de tempo de execução são exibidos nas Figuras 48 e 49.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 46 Comparação entre PQR-Sort e PQ-Sort com base em funções de avaliação em matrizes 10x10.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 47 Comparação entre PQR-Sort e PQ-Sort com base em funções de avaliação em matrizes 100x100. (A notação PQ Sort (1.0;1.0) indica que o algoritmo foi executado com pesos iguais para restrições menores e repetidas).

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	10x10	?	?
RANDOM_MATRIX	0.1	10x10	?	?
RECTNOISE	0.3	10x10	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	10x10	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	10x10	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	10x10	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.0	10x10	?	?

Figura 48 Relatório de tempo de execução para matrizes 10x10.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	100x100	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.0	100x100	PQR Sort	PQ Sort(1.0;1.0)

Figura 49 Relatório de tempo de execução para matrizes 100x100.

#### 5.1.4 PQR-Sort with Sorted Restrictions

Os resultados da comparação - com base em funções de avaliação - do PQR-Sort com o *PQR-Sort with Sorted Restrictions* estão listados nas Figuras 50 (para matrizes 10x10) e 51 (para matrizes 100x100). Em matrizes 10x10, em todos os casos houve empates. Para matrizes 100x100, o *PQR-Sort with Sorted Restrictions* foi o melhor em 67,86% dos casos; além disso, foram 25% de empates e, em 7,15% dos casos o PQR-Sort foi considerado o melhor (sempre o PQR-Sort foi considerado melhor, o *evaluator* era o *Minimal span* e as matrizes não possuíam ruído).

O *PQR-Sort with Sorted Restrictions* mostrou-se superior ao PQR-Sort para o *evaluator Anti-Robinson Loss Function*, pois foi melhor em 92,86% dos casos. Já para o *evaluator Minimal Span*, o *PQR-Sort with Sorted Restrictions* foi melhor em 42,86% dos casos (PQR-Sort foi melhor em 14,29% e foram 42,85% de empates). Logo, visto que o *PQR-Sort with Sorted Restrictions* foi igual ou superior ao PQR-Sort considerando os dois *evaluators*, pode-se afirmar que, para uma matriz com características desconhecidas (quando não se conhece os tipos de padrões que podem ser revelados, globais ou locais), entre os algoritmos avaliados neste texto, o algoritmo mais recomendado para reordenar matrizes a fim de minimizar os valores dos



evaluators e, dessa forma, possivelmente facilitar a obtenção de padrões, é o *PQR-Sort with Sorted Restrictions*.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 50 Comparação entre *PQR-Sort* e *PQR-Sort with Sorted Restrictions* com base em funções de avaliação em matrizes 10x10.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.0	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQR-Sorted Restrictions
RECTNOISE	0.0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQR-Sorted Restrictions
RECTNOISE	0.0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 51 Comparação entre *PQR-Sort* e *PQR-Sort with Sorted Restrictions* com base em funções de avaliação em matrizes 100x100. O algoritmo *PQR-Sort with Sorted Restrictions* é referenciado na figura como *PQR-Sorted Restrictions*.

Nas análises de tempo de execução (Figuras 52, para matrizes 10x10, e 53 para matrizes 100x100), os algoritmos apresentaram tempos muito parecidos. Em matrizes 10x10, praticamente apenas houve empates; e, nas matrizes 100x100, houve apenas empates, exceto por um caso em que o *PQR-Sort* foi melhor.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	10x10	?	?
RANDOM_MATRIX	0.1	10x10	?	?
RECTNOISE	0.3	10x10	?	?
RANDOM_MATRIX	0.3	10x10	?	?
RECTNOISE	0.5	10x10	?	?
RANDOM_MATRIX	0.5	10x10	?	?
RECTNOISE	0.0	10x10	?	?

Figura 52 Relatório de tempo de execução para matrizes 10x10.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	100x100	PQR Sort	PQR-Sorted Restrictions
RANDOM_MATRIX	0.1	100x100	?	?
RECTNOISE	0.3	100x100	?	?
RANDOM_MATRIX	0.3	100x100	?	?
RECTNOISE	0.5	100x100	?	?
RANDOM_MATRIX	0.5	100x100	?	?
RECTNOISE	0	100x100	?	?

Figura 53 Relatório de tempo de execução para matrizes 100x100.

### 5.1.5 Resumo dos Resultados

As Tabelas 1 e 2 resumem os resultados da comparação entre os algoritmos da Seção 5.1 com o *PQR-Sort*.

Tabela 1 Vitórias, empates e derrotas para qualidade da reordenação.

Algoritmo – tamanho	Vitória(%)	Empate (%)	Derrota (%)
PQR Smallest Restrictions – 10x10	0	96,43	3,57
PQR Smallest Restrictions – 100x100	0	92,86	7,14
PQ Sort – 10x10	0	100	0
PQ Sort – 100x100	21,43	28,57	50
PQR-Sort + BC – 10x10	7,14	92,86	0
PQR-Sort + BC – 100x100	46,43	3,57	50
PQR-Sort with Sorted Restrictions – 10x10	0	100	0
PQR-Sort with Sorted Restrictions – 100x100	67,86	17,86	7,14

Tabela 2 Vitórias, empates e derrotas para tempo de execução.

Algoritmo – tamanho	Vitória(%)	Empate (%)	Derrota (%)
PQR Smallest Restrictions – 10x10	0	100	0
PQR Smallest Restrictions – 100x100	28,57	72,43	0
PQ Sort – 10x10	0	43,86	57,14
PQ Sort – 100x100	0	0	100
PQR-Sort + BC – 10x10	0	100	0
PQR-Sort + BC – 100x100	0	0	100
PQR-Sort with Sorted Restrictions – 10x10	0	100	0
PQR-Sort with Sorted Restrictions –100x100	0	85,71	14,29

### 5.1.6 Testes com base em tipos de padrões – Global e Local

Esta seção faz dois tipos de comparações: (1) entre os algoritmos com tendência a gerar bons resultados segundo o *evaluator Minimal Span Loss Function* (PQR-Sort, PQ-Sort e PQR-Sort with Sorted Restrictions); (2) entre algoritmos com tendência a gerar bons resultados para o *evaluator Anti-Robinson Loss Function* (PQR-Sort + BC, PQR-Sort with Sorted Restrictions e Sugiyama<sup>12</sup>).

Os resultados do primeiro experimento com base em funções de avaliação são exibidos nas Figuras 54 (matrizes 10x10) e 55 (matrizes 100x100). As Figuras 56 e 57 exibem os resultados com base em tempo de execução para matrizes 10x10 e 100x100, respectivamente.

<sup>12</sup> Conforme constatado em Silva et al. (2011), o Sugiyama gera resultados melhores que o PQR-Sort e 2D-Sort para o *evaluator Anti-Robinson Loss function*; por ser um algoritmo que produz boas reordenações globais, ele foi empregado neste teste.



Matrix Type	Density	Dimensions	Evaluator	Coef	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 54 Comparação entre PQR-Sort , PQR-Sort with Sorted Restrictions e PQ-Sort com base em funções de avaliação em matrizes 10x10. O algoritmo PQR-Sort with Sorted Restrictions é referenciado na figura como PQR-Sorted Restrictions.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	PQR Sort	?

Figura 55 Comparação PQR-Sort , PQR-Sort with Sorted Restrictions e PQ-Sort com base em funções de avaliação em matrizes 100x100.

Consideradas as funções de avaliação, para matrizes 10x10, os algoritmos foram considerados equivalentes em todos os casos. Para matrizes 100x100, o PQR-Sort with Sorted

*Restrictions* foi melhor em 53,57% dos casos, o *PQR-Sort* em 10,71% dos casos e *PQ-Sort* em 3,57% dos casos.

Para matrizes 100x100 e *evaluator Minimal Span Loss Function*, o algoritmo *PQ-Sort* não é recomendado, visto que foi considerado o pior em 100% dos casos. Nessas condições, o *PQR-Sort with Sorted Restrictions* foi melhor em 42,86% dos casos e o *PQR-Sort* foi melhor em 14,29% dos casos. É importante citar que o *PQR-Sort* foi considerado o melhor algoritmo apenas nas matrizes do tipo *Rectnoise* com ruído igual a 0 - matrizes que dificilmente ocorrem em conjuntos de dados reais.

Em relação aos tempos de execução do primeiro experimento, para matrizes 10x10, houve empates para melhor algoritmo em todos os casos. Nas matrizes 100x100, o *PQR-Sort* foi o melhor em 14,29% dos casos; no restante dos casos houve apenas empates.

Considerando os testes com tempo de execução e funções de avaliação, pode-se afirmar que, quando se deseja uma reordenação boa em revelar padrões locais e a matriz a ser reordenada trata-se de uma matriz 100x100 aleatória ou similar a uma matriz do tipo *Rectnoise*, o algoritmo *PQR-Sort with Sorted Restrictions* é o mais recomendado.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	10x10	?	?
RANDOM_MATRIX	0.1	10x10	?	?
RECTNOISE	0.3	10x10	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	10x10	?	PQ Sort(1.0;1.0)
RECTNOISE	0.5	10x10	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	10x10	?	PQ Sort(1.0;1.0)
RECTNOISE	0.0	10x10	?	?

Figura 56 Análise de tempo comparativa entre *PQR-Sort*, *PQR-Sort with Sorted Restrictions* e *PQ-Sort* em matrizes 10x10.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	100x100	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	?	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	?	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	?	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	?	PQ Sort(1.0;1.0)

Figura 57 Análise de tempo comparativa entre *PQR-Sort*, *PQR-Sort with Sorted Restrictions* e *PQ-Sort* em matrizes 100x100.

Os resultados do segundo teste, executado apenas com algoritmos com tendência a gerar bons resultados para o *evaluator Anti-Robinson Loss Function*, estão listados nas Figuras 58, 59, 60 e 61; as duas primeiras exibem resultados de testes baseados em funções de avaliação e as duas últimas, testes de tempo de execução.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	Sugiyama
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Jaccard Coefficient	?	PQR-Sorted Restrictions
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 58 Comparação entre PQR-Sort + BC, PQR-Sort with Sorted Restrictions e Sugiyama com base em funções de avaliação em matrizes 10x10.

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	PQR-Sorted Restrictions
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	PQR-Sorted Restrictions
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	PQR-Sorted Restrictions
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	PQR-Sorted Restrictions
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	PQR-Sorted Restrictions
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	PQR-Sorted Restrictions
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	PQR-Sorted Restrictions
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	PQR-Sorted Restrictions
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	PQR-Sorted Restrictions
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	PQR-Sorted Restrictions
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	PQR-Sorted Restrictions	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	?	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	?	?

Figura 59 Comparação entre PQR-Sort + BC, PQR-Sort with Sorted Restrictions e Sugiyama com base em funções de avaliação em matrizes 100x100.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	10x10	?	?
RANDOM_MATRIX	0.1	10x10	?	?
RECTNOISE	0.3	10x10	Sugiyama	?
RANDOM_MATRIX	0.3	10x10	?	?
RECTNOISE	0.5	10x10	?	?
RANDOM_MATRIX	0.5	10x10	Sugiyama	?
RECTNOISE	0	10x10	?	?

Figura 60 Análise de tempo comparativa PQR-Sort+BC, PQR-Sort with Sorted Restrictions e Sugiyama em matrizes 10x10.

Matrix Type	Density	Dimensions	Best	Worst
RECTNOISE	0.1	100x100	PQR-Sorted Restrictions	PQR-Sort + BC
RANDOM_MATRIX	0.1	100x100	PQR-Sorted Restrictions	?
RECTNOISE	0.3	100x100	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.3	100x100	PQR-Sorted Restrictions	?
RECTNOISE	0.5	100x100	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.5	100x100	PQR-Sorted Restrictions	?
RECTNOISE	0	100x100	PQR-Sorted Restrictions	PQR-Sort + BC

Figura 61 Análise de tempo comparativa PQR-Sort+BC, PQR-Sort with Sorted Restrictions e algoritmo de Sugiyama modificado (Heurística baricêntrica) em matrizes 100x100.

Para matrizes 10x10, considerando resultados de funções de avaliação, ocorreram empates em 100% dos casos para melhor algoritmo. E, segundo os testes de tempo de execução, o Sugiyama foi melhor em 28,57% dos casos.

Assim, considerando a predominância de empates nos testes com base em funções de avaliação e o resultado melhor do Sugiyama no tempo de execução, para uma matriz 10x10 em que se deseja obter padrões globais, conclui-se que o algoritmo Sugiyama é mais recomendado entre os algoritmos avaliados.

Para matrizes 100x100, considerando apenas o *evaluator Anti-Robinson Loss Function*, houve empates em 100% dos casos para melhor algoritmo. E, o *PQR-Sort with Sorted Restrictions* foi considerado o pior algoritmo em 71,43% dos casos. Portanto, pode-se afirmar que PQR-Sort + BC e algoritmo de Sugiyama modificado empataram para melhor algoritmo em 71,43% dos casos. Nos testes de tempo de execução, o *PQR-Sort with Sorted Restrictions* foi o melhor em 100% dos casos. E, o PQR-Sort + BC foi o pior algoritmo em 28,57% dos casos, ou seja, em 28,57% dos casos, o algoritmo de Sugiyama modificado foi melhor que o PQR-

*Sort+BC*, e nos demais casos empataram. Portanto, para uma matriz 100x100, quando é desejável uma ordenação que evidencie padrões globais, o algoritmo mais recomendado é o Sugiyama.

### 5.1.7 Considerações sobre os testes com matrizes sintéticas

Com base nos testes desta seção, pode-se afirmar que, quando se deseja revelar padrões locais, o algoritmo mais recomendado é o *PQR-Sort with Sorted Restrictions*. Além de produzir boas ordenações para evidenciar padrões locais, seu tempo de execução é equivalente ao do algoritmo *PQR-Sort*. Outra vantagem do *PQR-Sort with Sorted Restrictions* em relação ao *PQR-Sort* é a geração de padrões globais; o *PQR-Sort with Sorted Restrictions* mostrou-se melhor que o *PQR-Sort* em reordenações que evidenciam padrões globais.

Para padrões globais, o algoritmo mais recomendado é o Sugiyama, apesar do alto tempo de execução quando comparado ao *PQR-Sort with Sorted Restrictions* e ao *PQR-Sort+BC*.

## 5.2 Conjunto de dados Íris

Esta seção utiliza os algoritmos encontrados na bibliografia e os algoritmos propostos por este trabalho para reordenar a matriz do Íris dataset (FISCHER, 1950). Esse conjunto de dados foi extraído do *University of California, Irvine - Machine Learning Repository* (FRANK & ASSUNCION, 2010). É um dos conjuntos de dados mais populares do repositório, contando com mais de 250.000 acessos desde 2007. Segundo consta no site do *UCI Machine Learning Repository*, o conjunto de dados foi referenciado por mais de 100 trabalhos<sup>13</sup>. Em geral, grande parte dos trabalhos está relacionada com algoritmos de mineração de dados, como em Duda & Hart (1973); Gates (1972) e Dasarathy (1980).

O conjunto de dados contém informações sobre três espécies de flores do gênero Íris: *setosa*, *versicolor* e *virginica*; são 50 flores de cada espécie, totalizando 150 flores. Cada registro do conjunto de dados possui cinco informações: altura da sépala, largura da sépala, altura da pétala, largura da pétala e espécie da flor (o Anexo A contém uma tabela com esse conjunto de dados).

---

<sup>13</sup> Lista completa de trabalhos pode ser encontrada em <http://archive.ics.uci.edu/ml/datasets/iris>

Segundo o site da UCI, a tarefa associada a esse conjunto de dados é a classificação, isto é, o algoritmo a ser testado com esse conjunto de dados deve ser capaz de, com base nas medidas das flores, classificá-las em espécies. No contexto deste trabalho, o objetivo é diferente, visto que os algoritmos de seriação não classificam dados. Na seriação de matrizes, os algoritmos tentam revelar padrões nos dados, para que o usuário, interagindo com a matriz, possa extrair conhecimento. Por isso, nesta seção, será comparada a capacidade dos algoritmos de seriação propostos por este trabalho em prover ordenações que auxiliem usuários a extrair conhecimento.

Originalmente, o conjunto de dados Íris não contém dados binários. Nesta seção, esse conjunto de dados será classificado - e conseqüentemente convertido em um conjunto de dados binário - para possibilitar os testes. Isso é necessário pois alguns dos algoritmos a serem testados não suportam dados não-binários; é o caso dos algoritmos: *PQR-Sort with Sorted Restrictions*, *PQR-Sort*, *PQ-Sort* e *PQR-Sort + BC*. O algoritmo de Sugiyama modificado é o único dos algoritmos testados que suporta dados não-binários. Os detalhes deste processo estão detalhados na seção 5.2.1.

### **5.2.1 Classificação do Conjunto de Dados Íris**

Na classificação dos dados, quatro atributos do conjunto de dados foram utilizados: altura da sépala, largura da sépala, altura da pétala e largura da pétala. Para cada atributo foram criadas cinco classes de valores: muito pequeno (PP), pequeno (P), médio (M), grande (G) e muito grande (GG). Assim, cada valor de atributo é associado a uma dessas classes; a classe a que o valor do atributo pertence é marcada como valor 1, e as demais são marcadas com valor 0. O algoritmo a seguir resume o processo de classificação:

Para cada atributo do conjunto de dados:

1. Calcular o valor máximo e o mínimo para definir o intervalo de valores.
2. Dividir o intervalo de valores em cinco partes (convencionou-se utilizar 5 classes: PP, P, M, G e GG). Assim, conhece-se o intervalo de valores das 5 classes.
3. Por fim, todos os valores do atributo são analisados. Cada um deles é classificado como PP, P, M, G ou GG; a classe a que o valor de atributo pertence é marcada como valor 1, e as demais classes são marcadas com valor 0. Por exemplo, para

uma dada flor, caso a altura de sépala seja classificada como P, na matriz final, nos dados referentes à altura da sépala dessa flor, P estará marcado com o valor 1 e PP, M, G e GG estarão marcados com 0.

O Quadro 3 apresenta os intervalos produzidos após a classificação dos atributos do íris dataset.

**Quadro 3 Classificação dos atributos do íris dataset.**

Classificação \ Atributos	Sépala		Pétala	
	Largura	Comprimento	Largura	Comprimento
PP	[4,3;5,02)	[2,0; 2,48)	[0,1; 0,58)	[1,0; 2,18)
P	[5,02; 5,74)	[2,48; 2,96)	[0,58; 1,06)	[2,18; 3,36)
M	[5,74; 6,46)	[2,96; 3,44)	[1,06; 1,54)	[3,36; 4,54)
G	[6,46; 7,18)	[3,44; 3,92)	[1,54; 2,02)	[4,54; 5,72)
GG	[7,18; 7,9]	[3,92; 4,4]	[2,02; 2,5]	[5,72; 6,9]

Com essa classificação, como cada atributo do conjunto de dados original gerou 5 classes, o conjunto de dados classificado possui 20 atributos (4 atributos do conjunto de dados original x 5 classes). A Figura 62 ilustra o íris dataset classificado (com uma reordenação aleatória); as colunas representam os 20 atributos e as linhas representam as flores, numeradas de 1 a 150. A próxima seção apresenta o Íris dataset binário reordenado por algoritmos encontrados na bibliografia e por algoritmos propostos por este trabalho.

### 5.2.2 Reordenações do Conjunto de dados Íris

Esta seção apresenta reordenações do conjunto de dados da Íris binário utilizando algoritmos de seriação. A Figura 62 apresenta o conjunto de dados original com uma ordenação aleatória. As Figuras 63, 64, 65 e 66 apresentam reordenações com os algoritmos *PQR-Sort*, *PQR-Sort with Sorted Restrictions*, *PQR-Sort + BC*, e algoritmo de Sugiyama modificado, respectivamente. Em todas as figuras, as cores na primeira coluna indicam a espécie da flor: a cor azul representa a espécie *iris-setosa*, amarela representa a espécie *iris-versicolor* e verde representa a espécie *iris-virginica*.

Além disso, nas Figuras 63, 64, 65 e 66 foram destacados em vermelho alguns agrupamentos de flores similares, como forma de comparar os algoritmos. Estes agrupamentos

foram definidos empiricamente; não foi empregado nenhum algoritmo ou teste com usuário para defini-los (estes são alguns dos trabalhos futuros).

Comparando o dataset sem ordenação com as reordenações produzidas pelos algoritmos, pode-se observar que com os algoritmos de seriação é mais fácil extrair padrões e encontrar grupos de flores com características similares. Entre as versões reordenadas, também foi visível a diferença dos resultados produzidos por algoritmos que evidenciam padrões globais e aqueles que evidenciam padrões locais.

Os algoritmos *PQR-Sort + BC* e Sugiyama, que procuram evidenciar padrões globais, produziram resultados similares: as três espécies de flores praticamente permaneceram em linhas consecutivas (esse fato é evidenciado pelas cores das linhas). Com o algoritmo *PQR-Sort + BC*, por exemplo, a espécie *iris-setosa*, marcada em azul na matriz, teve os 50 casos agrupados consecutivamente nas linhas da matriz.

Por sua vez, os algoritmos que buscam evidenciar similaridades locais, *PQR-Sort* e *PQR-Sort with Sorted Restrictions*, somente formaram grupos de flores com alta similaridade. Isso faz com que, nas linhas, as espécies de flores não permaneçam consecutivas; por exemplo, no caso do algoritmo *PQR-Sort*, formaram-se mais de cinco grupos na matriz da espécie *iris-virginica* (marcada em verde nas matrizes).

Apesar dos quatro algoritmos testados produzirem bons resultados, entre os algoritmos que procuram evidenciar padrões globais, o algoritmo *PQR-Sort + BC* formou blocos maiores de flores similares quando comparado com o Sugiyama. Além disso, diferentemente do Sugiyama, o *PQR-Sort + BC* agrupou consecutivamente todas as flores da espécie *iris-setosa*.

E, entre os algoritmos que buscam evidenciar padrões locais, *PQR-Sort* e *PQR-Sort with Sorted Restrictions* tiveram resultados similares. Um das diferenças foi que o *PQR-Sort with Sorted Restrictions* agrupou consecutivamente as 50 flores da espécie *iris-setosa* (no resultado do *PQR-Sort*, a espécie *iris-setosa* está praticamente dividida em dois grandes grupos).

Portanto, ao utilizar um conjunto de dados real com os algoritmos de seriação produzidos por este trabalho, mostra-se que os algoritmos produzidos também podem ser úteis em matrizes não-sintéticas. Resultados mais conclusivos sobre o aspecto visual produzido por diferentes



algoritmos podem ser obtidos com testes com usuários, em que usuários podem tentar encontrar grupos de flores similares em diferentes reordenações e também avaliá-las subjetivamente.

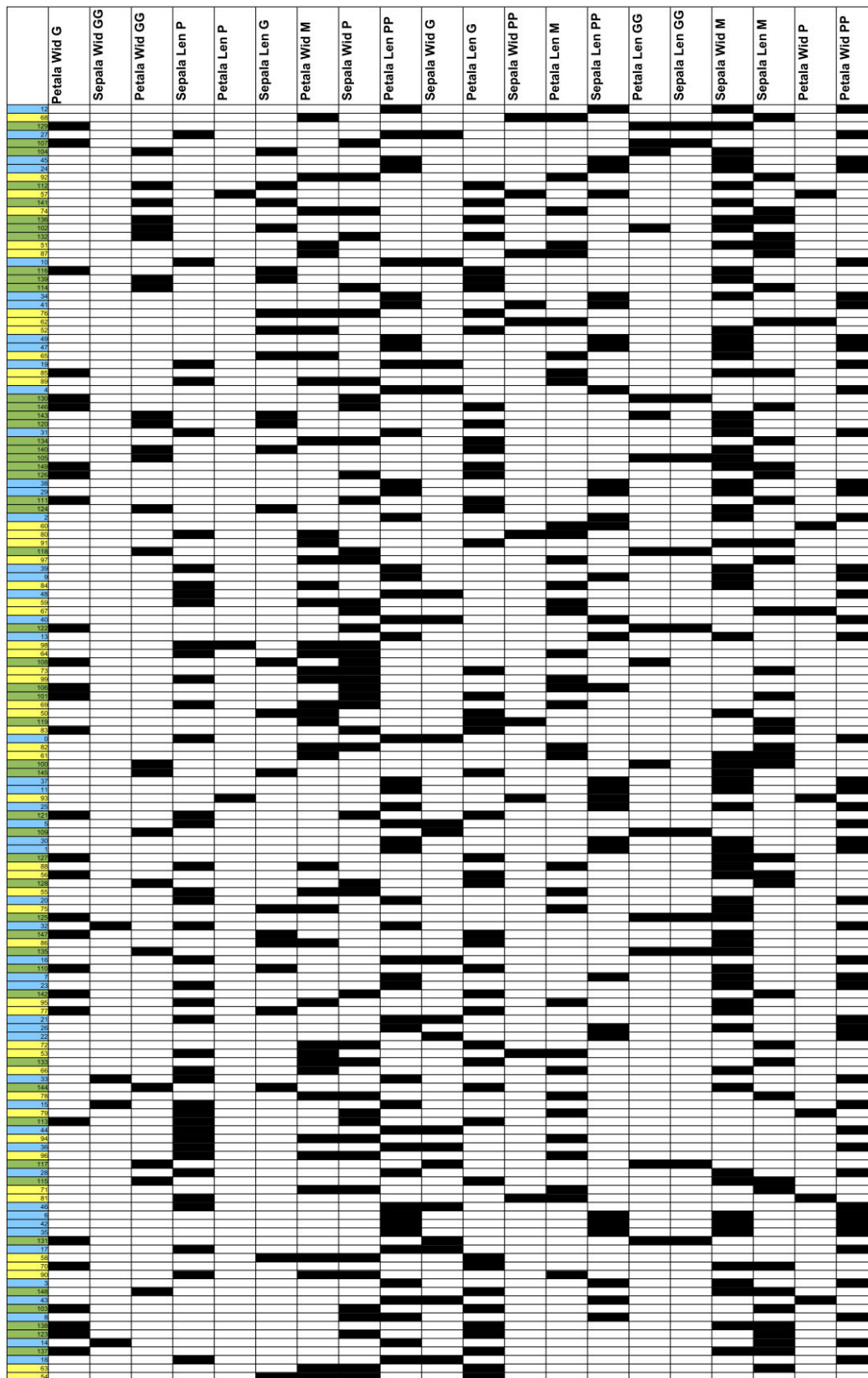


Figura 62 Reordenação aleatória para o iris dataset.



Figura 63 Íris dataset reordenado com o algoritmo PQR-Sort.

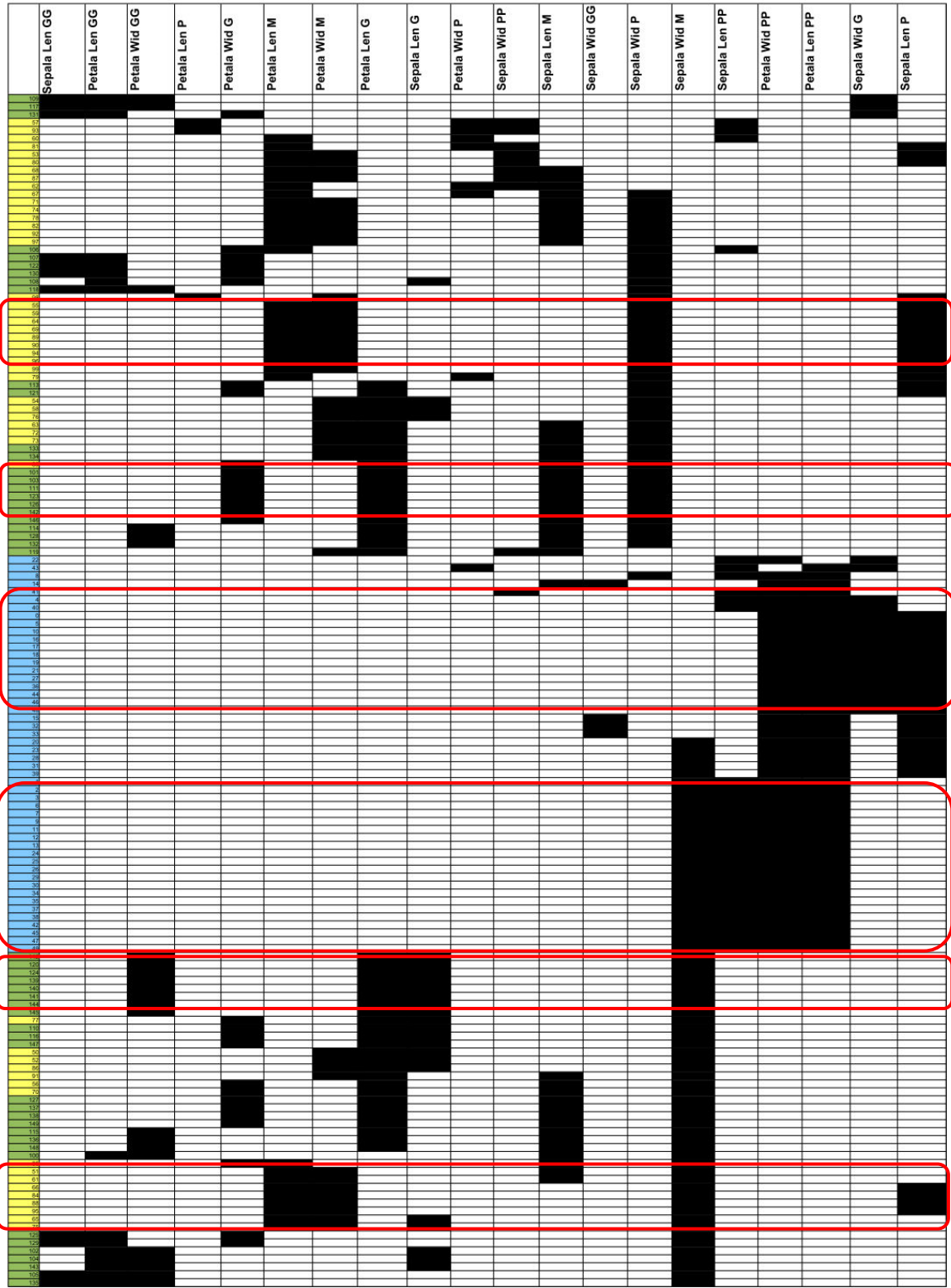


Figura 64 Íris dataset reordenado com o algoritmo PQR-Sort with Sorted Restrictions.

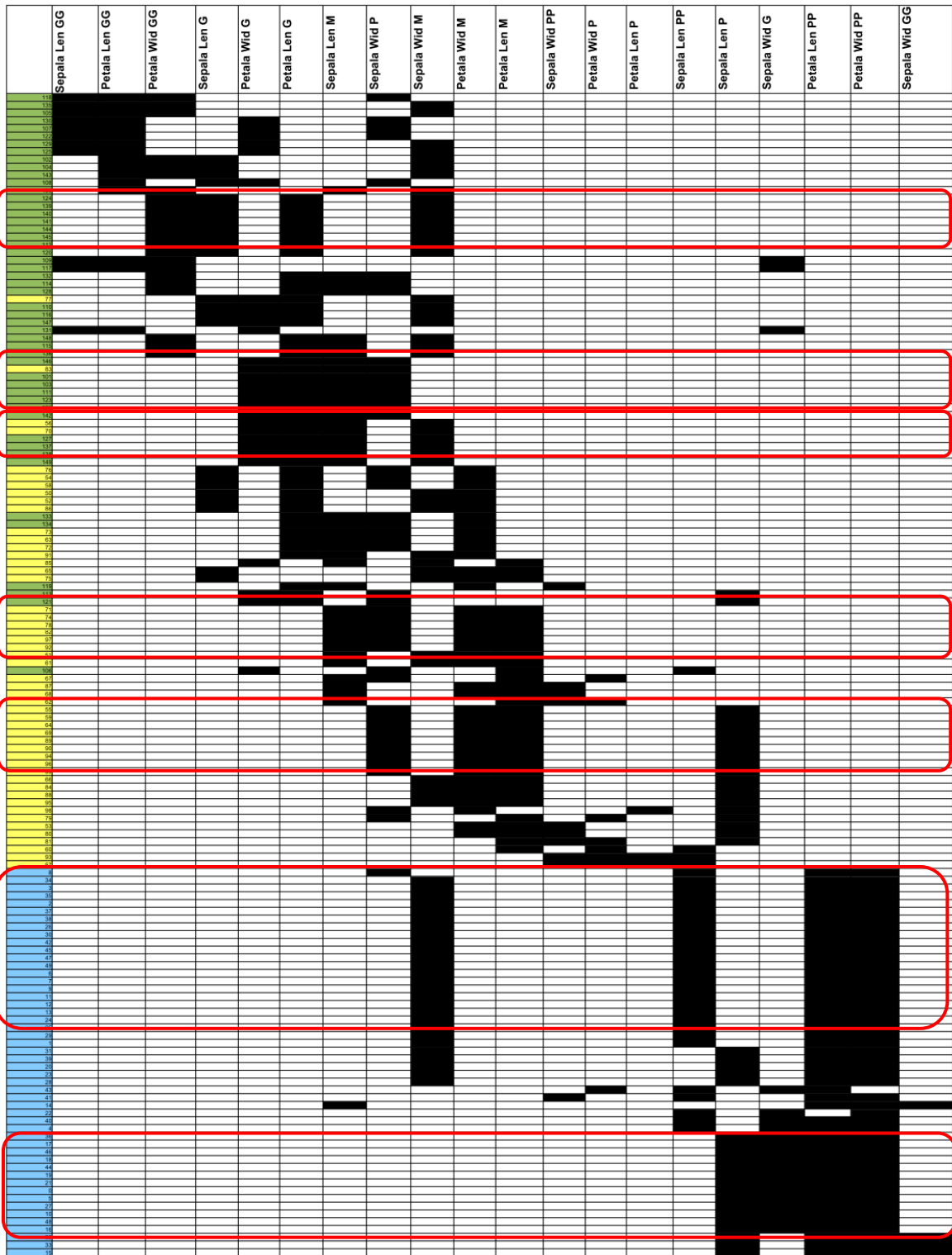


Figura 65 Íris dataset reordenado com o algoritmo PQR-Sort + BC

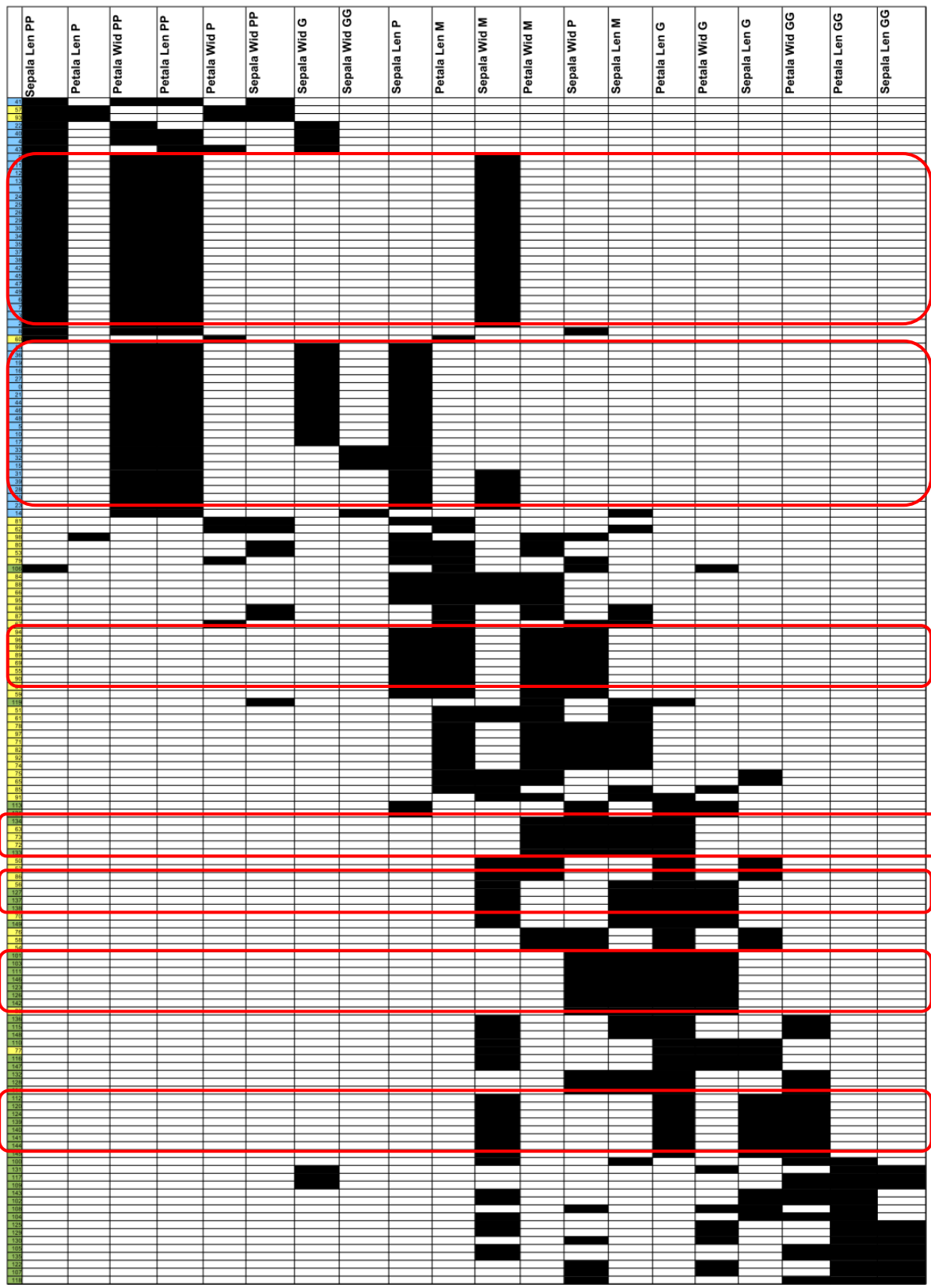


Figura 66 Íris dataset reordenado com o algoritmo de Sugiyama Modificado

### 5.3 Considerações Finais

Este capítulo apresentou os resultados dos testes com matrizes sintéticas e não sintéticas. Os testes com matrizes sintéticas revelaram que o algoritmo *PQR-Sort with Sorted Restrictions* é o algoritmo mais recomendado quando se deseja revelar padrões locais. Para revelar padrões globais o algoritmo mais recomendado é o algoritmo de Sugiyama modificado. Em relação a matrizes não sintéticas, foi realizado um estudo introdutório com o conjunto de dados Íris; este estudo mostrou que os algoritmos produzidos por este trabalho auxiliam na extração de conhecimento deste conjunto de dados.

## 6 Conclusão

Na revisão bibliográfica foi possível notar que boa parte dos métodos de seriação são heurísticos, como: BEA, *2D Sort*, Heurística Baricêntrica etc. No contexto de seriação, algoritmos heurísticos podem não produzir boas soluções. O melhor exemplo disso é o algoritmo *2D Sort*, pois a cada execução ele pode gerar uma solução diferente. Não existem garantias de que todas as soluções geradas serão boas. E, além disso, é difícil determinar a complexidade de um algoritmo heurístico, pois podem ocorrer situações de laços infinitos.

Os algoritmos de seriação não-heurísticos possuem características diferentes dos heurísticos. É possível determinar a complexidade dos algoritmos. E, portanto, é possível estimar o tempo de execução. *PQR-Sort*, Clusterização Hierárquica e Seriação Elíptica são exemplos de algoritmos de seriação não-heurísticos.

Por isso, neste trabalho buscou-se produzir uma nova versão para o algoritmo determinístico *PQR-Sort*. Os principais algoritmos produzidos foram o *PQR-Smallest Restrictions* e *PQR-Sort with Sorted Restrictions*.

Após a produção dos algoritmos foi importante testá-los com diferentes tipos de matrizes e, principalmente, compará-los com o algoritmo *PQR-Sort* original. Tal comparação foi possível devido à ferramenta MRA e às modificações propostas: sumarização de relatórios e testes de hipóteses.

As melhorias propostas para a ferramenta MRA reduziram significativamente o tempo de análise dos resultados de um experimento. Em versões anteriores, as informações de um experimento eram divididas em diversos relatórios; e, além disso, os diversos relatórios não continham testes estatísticos que produzissem respostas conclusivas sobre os testes.

Com os novos relatórios gerados pela ferramenta MRA, foi possível chegar a conclusões relevantes para a área de seriação de matrizes. Durante os testes, verificou-se que o *PQR-Sort*, *PQR-Sort with Sorted Restrictions*, *PQ-Sort* e *PQR-Smallest Restrictions* são algoritmos que produzem reordenações que buscam evidenciar padrões locais. E, por outro lado, Sugiyama e *PQR-Sort + BC* produzem reordenações que buscam evidenciar padrões globais.



Apesar de não ser um dos objetivos deste trabalho produzir um algoritmo que evidencie padrões globais, devido aos bons resultados do algoritmo *PQR-Sort + BC* considerando o *evaluator Anti-Robinson Loss Function*, foram realizados testes entre o *PQR-Sort + BC* e um algoritmo reconhecidamente bom para reordenações globais, o Sugiyama. Os resultados desses testes mostram que *PQR-Sort + BC* e Sugiyama são equivalentes considerando o *evaluator Anti-Robinson Loss Function*. E, nos testes de tempo de execução, o Sugiyama foi melhor que o *PQR-Sort + BC* em 71,43% dos casos.

Em relação aos algoritmos que buscam evidenciar padrões locais, a contribuição deste trabalho foi o algoritmo *PQR-Sort with Sorted Restrictions*. Este algoritmo, quando comparado com o *PQR-Sort* em matrizes sintéticas do tipo *Random* e *Rectnoise*, mostrou-se superior ou tão bom quanto em 92,85% dos casos; enquanto que, o *PQR-Sort* foi o melhor em apenas 7,15% dos casos. Ademais, ainda no mesmo teste, mas considerando apenas resultados do *evaluator Anti-Robinson Loss Function* – função que mede a presença de padrões globais – o *PQR-Sort with Sorted Restrictions* quando comparado com o *PQR-Sort* foi o melhor em 92,85% dos casos. E, considerando apenas o *evaluator Minimal Span Loss Function*, o *PQR-Sort with Sorted Restrictions* foi melhor ou tão bom quanto ao *PQR-Sort* em 85,72% dos casos. Esse resultado mostra que o *PQR-Sort with Sorted Restrictions* é superior ao ponto forte do *PQR-Sort*: os padrões locais; e, também é superior ao *PQR-Sort* em um dos seus pontos fracos, os padrões globais.

Em relação ao tempo de execução, *PQR-Sort* e *PQR-Sort with Sorted Restrictions* foram considerados similares. Nos testes realizados com matrizes 100x100, em 100% dos casos os algoritmos gastaram menos que 0,01 segundos nas reordenações. Esse baixo tempo de execução – quando comparado com outros algoritmos – faz com que o *PQR-Sort* possa ser utilizado em sistemas que interagem com usuários, visto que 0,1 segundos é o intervalo de tempo máximo em que um usuário percebe que uma ação do sistema foi resultado de sua última ação.

Outro ponto importante do trabalho foi a utilização dos algoritmos propostos com um conjunto de dados real e muito conhecido na área de mineração de dados. Esse estudo de caso mostra que os algoritmos propostos geraram boas reordenações para esse conjunto de dados. Contudo, respostas mais conclusivas sobre as reordenações produzidas para o conjunto de dados

Íris poderão ser obtidas com testes com usuários – um dos trabalhos futuros deste trabalho, elencado na Seção 6.1.

## 6.1 Trabalhos Futuros

Alguns pontos deste trabalho podem ser tema de trabalhos futuros, apresentados a seguir.

- **Comparar os algoritmos produzidos neste trabalho com mais algoritmos da área de Seriação.** Os algoritmos produzidos foram comparados apenas com o Sugiyama. Resultados interessantes poderiam ser obtidos da comparação do *PQR-Sort with Sorted Restrictions* com outros algoritmos clássicos da área de seriação de matrizes, como Clusterização Hierárquica.

- **Realizar testes com usuários para obter avaliações das reordenações do Íris dataset.** As reordenações geradas pelos algoritmos de seriação não foram submetidas a testes com usuários. Testes com usuários poderiam revelar se, de fato, as reordenações produzidas auxiliam na extração de conhecimento.

- **Testar os algoritmos produzidos com outros conjuntos de dados reais.** Neste trabalho, o único conjunto de dados real empregado foi o Íris dataset.

- **Adaptar os algoritmos produzidos para reordenar matrizes não-binárias.** Os algoritmos produzidos por este trabalho possuem uma limitação: reordenam apenas matrizes binárias. Melo (2009) apresenta variações do *PQR-Sort* para reordenar matrizes não-binárias. Trabalhos futuros, poderiam utilizar os algoritmos produzidos por Melo para permitir que o *PQR-Sort with Sorted Restrictions* reordene matrizes não-binárias.

- **Realizar testes com matrizes maiores.** Os algoritmos produzidos foram comparados apenas com matrizes 10x10 e 100x100. Trabalhos futuros podem utilizar os algoritmos produzidos em testes com matrizes de dimensões maiores, como 1000x1000.

## Referências

ARABIE, P.; HUBERT, L. J. The bond energy algorithm revisited. **IEEE transactions on systems, man and cybernetics**, Volume 20, 1990, p. 268- 274.

BAR-JOSEPH, Z.; GIFFORD, D. K.; JAAKKOLA, T. S. Fast optimal leaf ordering for hierarchical clustering. **Bioinformatics**, Vol. 17, 2001, p. 22-29.

BERTIN, J. Matrix theory of graphs. **Information Design Journal**, 2001, p. 5-19.

CARAUX, G.; PINLOCHE, S. Permutmatrix: A graphical environment to arrange gene expression profiles in optimal linear order. **Bioinformatics**, Volume 21, N°7, 2005, p.1280–1281.

CARD, S.K.; MACKINLAY, J.D.; SHNEIDERMAN, B. **Readings in Information Visualization: Using Vision to Think**. Morgan Kaufman Publishers, 1999.

CHEN, C. H. Generalized Association Plots: information visualization via iteratively generated correlation matrices. **Statistica Sinica** 12, 2002, p. 7-29.

CORMEN, T.; LEISERSON, C.;RIVEST, R. **Introduction to Algorithms**, McGraw-Hill, 1990.

COX, M.A.A.; COX, T.F. Multidimensional Scaling. In: Chen, C.; Härdle, W.; Unwin, A. (Eds.). **Handbook of Data Visualization**, Springer, 2008 , p. 315-348.

CHOE, S.; CHA, S.; TAPPERT, C. A Survey of Binary Similarity and Distance Measures. **Journal of Systemics, Cybernetics and Informatics** - Volume 8, 2010, p. 43-48.

DASARATHY, B.V. Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Volume PAMI-2, 1980, p. 67-71.

DUDA, R.O.; HART, P. E. **Pattern Classification and Scene Analysis**. John Wiley & Sons, 1973.

FRANK, A.; ASUNCION, A. **UCI Machine Learning Repository**. Irvine, CA: University of California, School of Information and Computer Science, 2010. Disponível em: <http://archive.ics.uci.edu/ml> (10/10/2011).

FISHER, R.A. The use of multiple measurements in taxonomic problems. **Annual Eugenics**, 7, 1936, p.179-188.

GATES, G.W. (1972) The Reduced Nearest Neighbor Rule. **IEEE Transactions on Information Theory**, 1972, p. 431-433.

GUTIN, G.; Punnen, A. P. **The Traveling Salesman Problem and Its Variations (Combinatorial Optimization)**, Kluwer, 2002.

HAHSLER, M.; HORNIK, K.; BUCHTA, C. **Getting things in order: An introduction to the R package seriation**, 2010. Disponível em <http://cran.r-project.org/web/packages/seriation/vignettes/seriation.pdf> (10/09/2010).

HENRY, N.; FEKETE, J. Evaluating visual table data understanding. **AVI Workshop on Beyond Time and Errors: Novel Evaluation Methods For information Visualization**, 2006, p. 1-5.

HUBERT, L.; ARABIE, P.; MEULMAN, J. **Combinatorial Data Analysis: Optimization by Dynamic Programming**. Society for Industrial Mathematics, 2001.

Liiv, I., Seriation and Matrix Reordering Methods: An Historical Overview. **Statistical Analysis and Data Mining** 3, 2010, p. 70-91.

LIMA, D. M.; COLUGNATI, F. A. B.; PADOVANI, R. M.; RODRIGUEZ-AMAYA, D. B.; SALAY, E.; GALEAZZI, M. A. M. **Tabela Brasileira de Composição de Alimentos – TACO**. Núcleo de Estudos e Pesquisas em Alimentação – NEPA, 2006. Disponível em: [http://www.unicamp.br/nepa/taco/contar/taco\\_versao2.pdf](http://www.unicamp.br/nepa/taco/contar/taco_versao2.pdf)(02/04/2012).

MAGALHAES, M. N.; LIMA, A. C. **Noções de Probabilidade e Estatística**, Edusp, 2002.

MAZZA, R. **Introduction to Information Visualization**. Springer, 2009.

MCCORMICK, W. T.; SCHWEITZER, P. J.; WHITE, T. W. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, Volume 20, N°5, 1972, p. 993-1009.

MEIDANIS, J.; PORTO, O.; TELLES, G.P. On the consecutive ones property. *Discrete Applied Mathematics*, Volume 8, 1998, p. 325-354.

MELO, M. F. **Reorganização de estruturas visuais matriciais utilizando árvores PQR.** Trabalho de Graduação Interdisciplinar. Faculdade de Tecnologia. Universidade Estadual de Campinas, 2009.

NIERMANN, S. Optimizing the Ordering of Tables With Evolutionary Computation. *The American Statistician*, Volume 59, N° 1, 2005, p. 41-46.

PETRIE, F. W. M. Sequences in prehistoric remains. *Journal of the Anthropological Institute*, 1899, p. 295-301.

SIIRTOLA, H.; MÄKINEN, E. Constructing and reconstructing the reorderable matrix. *Information Visualization*, 2005, p. 32-48.

MÄKINEN, E.; SIIRTOLA, H. Reordering the reorderable matrix as an algorithmic problem. *LNCS 1889*, 2000, p. 453-468,

SILVA, F. P. **Reorganização de estruturas visuais matriciais utilizando árvores PQR.** Relatório de Iniciação Científica - PIBIC. Faculdade de Tecnologia, Universidade Estadual de Campinas – UNICAMP, 2010.

SILVA, C. G.; MELO, M. F.; SILVA, F. P.; MEIDANIS, J.; **PQR-Sort - Using PQR-trees for binary matrix reorganization**, 2011. (Não publicado).

TELLES, G.P.; MEIDANIS, J. Building PQR trees in almost-linear time. *Electronic Notes in Discrete Mathematics*, Volume 19, 2005, p. 33-39.

WILSON, R. J. **Introduction to Graph Theory**, Prentice Hall, 1996.

WU, H.-M.; TZENG, S.; CHEN, C.-H. Matrix Visualization. In: Chen, C.-H.; Hardle, W.; Unwin, A., **Handbook of Data Visualization**, Springer-Verlag, 2008, p. 681-708.

## **Apêndice A - Versão completa dos relatórios de funções de avaliação**

Este apêndice apresenta a versão completa dos relatórios de funções de avaliação apresentados no Capítulo 5. Em tal capítulo, para maior legibilidade, foram exibidos os relatórios resumidos; a seguir, serão exibidas as versões completas.

## Comparação entre PQR-Sort e PQR-Sort + BC (matrizes 10x10)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQR + BC	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-5.83E-002		
					St. Dev	5.80E-001		
					T	-3.18E-001	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.32E-001		
					St. Dev	3.20E-001		
					T	-1.30E+000	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.03E+000		
					St. Dev	9.12E+000		
					T	7.04E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	2.10E-001		
					St. Dev	1.59E+001		
					T	4.18E-002	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	1.03E+000		
					St. Dev	7.44E-001		
					T	4.37E+000	PQR-Sort + BC	PQR Sort
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	1.04E-001		
					St. Dev	2.23E-001		
					T	1.48E+000	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.10E-001		
					St. Dev	1.23E+000		
					T	5.42E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-1.63E+000		
					St. Dev	1.92E+001		
					T	-2.69E-001	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.47E-001		
					St. Dev	6.07E-001		
					T	-1.81E+000	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-3.50E-001		
					St. Dev	4.24E-001		
					T	-2.61E+000	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.05E+001		
					St. Dev	1.76E+001		
					T	1.89E+000	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	7.91E+000		
					St. Dev	1.68E+001		
					T	1.49E+000	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.20E-001		
					St. Dev	6.19E-001		
					T	-1.64E+000	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-3.17E-001		
					St. Dev	4.49E-001		
					T	-2.23E+000	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.32E+001		
					St. Dev	1.69E+001		
					T	2.48E+000	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	6.83E+000		
					St. Dev	1.99E+001		
					T	1.08E+000	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.26E-001		
					St. Dev	4.96E-001		
					T	-2.08E+000	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-3.37E-001		
					St. Dev	4.55E-001		
					T	-2.34E+000	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	9.89E+000		
					St. Dev	2.15E+001		
					T	1.45E+000	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.22E+001		
					St. Dev	1.93E+001		
					T	2.00E+000	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.88E-001		
					St. Dev	5.69E-001		
					T	-1.04E+000	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-2.32E-001		
					St. Dev	5.13E-001		
					T	-1.43E+000	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.36E+001		
					St. Dev	2.11E+001		
					T	2.03E+000	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.47E+001		
					St. Dev	1.78E+001		
					T	2.61E+000	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	8.85E-001		
					St. Dev	8.06E-001		
					T	3.47E+000	PQR-Sort + BC	PQR Sort
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	7.20E-002		
					St. Dev	2.60E-001		
					T	8.75E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-9.00E-002		
					St. Dev	1.06E+000		
					T	-2.67E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-3.03E+000		
					St. Dev	1.78E+001		
					T	-5.39E-001	?	?
					St. Dev	2.22E+004		
					T	-1.54E-001	?	?

## Comparação entre PQR-*Sort* e PQR-*Sort* + BC (matrizes 100x100)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQR + BC	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-2.94E+000		
					St. Dev	2.05E+000		
					T	-1.44E+001	PQR Sort	PQR-Sort + BC
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.69E+000		
					St. Dev	1.61E+000		
					T	-1.05E+001	PQR Sort	PQR-Sort + BC
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.11E+004		
					St. Dev	1.28E+004		
					T	8.69E+000	PQR-Sort + BC	PQR Sort
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	4.40E+004		
					St. Dev	1.62E+004		
					T	2.72E+001	PQR-Sort + BC	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.55E+000		
					St. Dev	7.21E-001		
					T	-4.93E+001	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.26E+000		
					St. Dev	2.51E-001		
					T	-4.99E+001	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.35E+004		
					St. Dev	2.86E+003		
					T	4.71E+001	PQR-Sort + BC	PQR Sort
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	2.17E+003		
					St. Dev	6.20E+003		
					T	3.51E+000	PQR-Sort + BC	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.34E+000		
					St. Dev	7.64E-001		
					T	-1.76E+001	PQR Sort	PQR-Sort + BC
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.20E+000		
					St. Dev	7.26E-001		
					T	-1.66E+001	PQR Sort	PQR-Sort + BC
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.13E+004		
					St. Dev	6.11E+003		
					T	1.86E+001	PQR-Sort + BC	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.93E+004		
					St. Dev	6.77E+003		
					T	2.86E+001	PQR-Sort + BC	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.64E+000		
					St. Dev	5.70E-001		
					T	-2.87E+001	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.43E+000		
					St. Dev	4.75E-001		
					T	-3.01E+001	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.41E+004		
					St. Dev	3.73E+003		
					T	3.78E+001	PQR-Sort + BC	PQR Sort
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	9.20E+003		
					St. Dev	4.30E+003		
					T	2.14E+001	PQR-Sort + BC	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.17E+000		
					St. Dev	6.21E-001		
					T	-1.89E+001	PQR Sort	PQR-Sort + BC
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.33E+000		
					St. Dev	6.66E-001		
					T	-2.00E+001	PQR Sort	PQR-Sort + BC
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.46E+004		
					St. Dev	4.21E+003		
					T	3.45E+001	PQR-Sort + BC	PQR Sort
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.25E+004		
					St. Dev	1.73E+003		
					T	7.23E+001	PQR-Sort + BC	PQR Sort
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.12E+000		
					St. Dev	6.45E-001		
					T	-1.74E+001	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.28E+000		
					St. Dev	6.94E-001		
					T	-1.84E+001	PQR Sort	PQR-Sort + BC
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.44E+004		
					St. Dev	4.61E+003		
					T	3.13E+001	PQR-Sort + BC	PQR Sort
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.24E+004		
					St. Dev	1.90E+003		
					T	6.52E+001	PQR-Sort + BC	PQR Sort
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-5.90E-001		
					St. Dev	9.53E-001		
					T	-6.19E+000	PQR Sort	PQR-Sort + BC
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-4.71E-001		
					St. Dev	5.65E-001		
					T	-8.34E+000	PQR Sort	PQR-Sort + BC
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	5.69E+003		
					St. Dev	1.05E+004		
					T	5.41E+000	PQR-Sort + BC	PQR Sort
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-3.42E+002		
					St. Dev	2.22E+004		
					T	-1.54F-001	?	?



## Comparação entre PQR-*Sort* e PQR-*Smallest Restrictions* (matrizes 10x10)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQR-SR	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-8.94E-002		
					St. Dev	3.77E-001		
					T	-7.51E-001	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-2.40E-002		
					St. Dev	1.09E-001		
					T	-6.94E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-2.07E+000		
					St. Dev	9.08E+000		
					T	-7.21E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-2.02E+000		
					St. Dev	9.08E+000		
					T	-7.03E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-5.00E-002		
					St. Dev	3.59E-001		
					T	-4.41E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-9.00E-003		
					St. Dev	6.68E-002		
					T	-4.26E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-6.40E-001		
					St. Dev	4.54E+000		
					T	-4.46E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-6.90E-001		
					St. Dev	5.39E+000		
					T	-4.05E-001	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.19E-001		
					St. Dev	3.64E-001		
					T	-1.03E+000	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-7.20E-002		
					St. Dev	3.10E-001		
					T	-7.35E-001	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-1.01E+001		
					St. Dev	1.78E+001		
					T	-1.80E+000	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-7.30E+000		
					St. Dev	1.59E+001		
					T	-1.45E+000	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.01E-001		
					St. Dev	4.13E-001		
					T	-7.73E-001	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-7.00E-003		
					St. Dev	2.74E-001		
					T	-8.08E-002	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-8.36E+000		
					St. Dev	1.63E+001		
					T	-1.62E+000	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-3.73E+000		
					St. Dev	1.67E+001		
					T	-7.07E-001	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-2.21E-001		
					St. Dev	4.52E-001		
					T	-1.54E+000	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.71E-001		
					St. Dev	4.23E-001		
					T	-1.28E+000	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-1.31E+001		
					St. Dev	2.14E+001		
					T	-1.93E+000	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-9.01E+000		
					St. Dev	1.65E+001		
					T	-1.72E+000	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.16E-001		
					St. Dev	5.38E-001		
					T	-1.88E+000	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-2.06E-001		
					St. Dev	4.72E-001		
					T	-1.38E+000	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-1.71E+001		
					St. Dev	1.84E+001		
					T	-2.94E+000	PQR Sort	PQR Smallest Restrictions (0.5)
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-1.18E+001		
					St. Dev	1.56E+001		
					T	-2.37E+000	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?

## Comparação entre PQR-Sort e PQR-Smallest Restrictions (matrizes 100x100)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQR-SR	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.56E-001		
					St. Dev	5.87E-001		
					T	-2.66E+000	PQR Sort	PQR Smallest Restrictions (0.5)
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-7.89E-002		
					St. Dev	3.34E-001		
					T	-2.36E+000	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-6.54E+002		
					St. Dev	2.90E+003		
					T	-2.26E+000	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-2.51E+002		
					St. Dev	2.38E+003		
					T	-1.06E+000	?	?
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-4.72E-002		
					St. Dev	2.56E-001		
					T	-1.84E+000	?	?
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-4.13E-002		
					St. Dev	2.42E-001		
					T	-1.71E+000	?	?
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-4.59E+002		
					St. Dev	1.83E+003		
					T	-2.51E+000	?	?
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-1.43E+002		
					St. Dev	1.18E+003		
					T	-1.21E+000	?	?
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	0.00E+000		
					St. Dev	0.00E+000		
					T	NaN	?	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-4.66E-002		
					St. Dev	5.82E-001		
					T	-8.00E-001	?	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-4.51E-002		
					St. Dev	6.23E-001		
					T	-7.24E-001	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-6.32E+002		
					St. Dev	3.93E+003		
					T	-1.61E+000	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-1.01E+003		
					St. Dev	1.77E+003		
					T	-5.71E+000	PQR Sort	PQR Smallest Restrictions (0.5)
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	6.79E-003		
					St. Dev	5.23E-001		
					T	1.30E-001	?	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	3.10E-003		
					St. Dev	5.73E-001		
					T	5.41E-002	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-7.67E+002		
					St. Dev	3.35E+003		
					T	-2.29E+000	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-7.88E+002		
					St. Dev	1.81E+003		
					T	-4.35E+000	PQR Sort	PQR Smallest Restrictions (0.5)
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-5.30E-001		
					St. Dev	2.63E+000		
					T	-2.01E+000	?	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-5.09E-002		
					St. Dev	3.52E-001		
					T	-1.45E+000	?	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-4.10E+002		
					St. Dev	5.16E+003		
					T	-7.95E-001	?	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-8.70E+002		
					St. Dev	7.06E+003		
					T	-1.23E+000	?	?

## Comparação entre PQR-*Sort* e PQ-*Sort* (matrizes 10x10)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQ- <i>Sort</i>	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	7.88E-003		
					St. Dev	4.65E-001		
					T	5.36E-002	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	3.80E-002		
					St. Dev	2.53E-001		
					T	4.75E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.26E+000		
					St. Dev	8.79E+000		
					T	8.13E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.96E+000		
					St. Dev	1.81E+001		
					T	3.43E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	1.25E-002		
					St. Dev	1.56E-001		
					T	2.53E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	8.00E-003		
					St. Dev	7.06E-002		
					T	3.58E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	7.00E-002		
					St. Dev	6.07E-001		
					T	3.65E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	4.90E-001		
					St. Dev	6.80E+000		
					T	2.28E-001	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	8.27E-002		
					St. Dev	4.47E-001		
					T	5.85E-001	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	6.70E-002		
					St. Dev	3.61E-001		
					T	5.87E-001	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.35E+001		
					St. Dev	1.87E+001		
					T	2.28E+000	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.00E+001		
					St. Dev	1.83E+001		
					T	1.74E+000	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	1.45E-001		
					St. Dev	4.92E-001		
					T	9.30E-001	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	8.30E-002		
					St. Dev	3.81E-001		
					T	6.89E-001	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.32E+001		
					St. Dev	2.20E+001		
					T	1.89E+000	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	6.52E+000		
					St. Dev	2.04E+001		
					T	1.01E+000	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.14E-002		
					St. Dev	4.67E-001		
					T	-7.72E-002	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-5.90E-002		
					St. Dev	4.35E-001		
					T	-4.29E-001	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.11E+001		
					St. Dev	1.72E+001		
					T	2.05E+000	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	7.50E+000		
					St. Dev	1.73E+001		
					T	1.37E+000	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.88E-003		
					St. Dev	4.65E-001		
					T	-1.28E-002	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-2.20E-002		
					St. Dev	4.38E-001		
					T	-1.59E-001	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	9.63E+000		
					St. Dev	2.10E+001		
					T	1.45E+000	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	8.01E+000		
					St. Dev	2.01E+001		
					T	1.26E+000	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.50E-003		
					St. Dev	2.03E-001		
					T	-5.46E-002	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	6.00E-003		
					St. Dev	8.74E-002		
					T	2.17E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-1.10E-001		
					St. Dev	1.41E+000		
					T	-2.46E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	4.20E-001		
					St. Dev	7.31E+000		
					T	1.82E-001	?	?



## Comparação entre PQR-Sort e PQ-Sort (matrizes 100x100)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQ-Sort	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-6.77E+000		
					St. Dev	2.71E+000		
					T	-2.50E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-4.15E+000		
					St. Dev	1.93E+000		
					T	-2.15E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.93E+003		
					St. Dev	9.42E+003		
					T	3.11E+000	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	6.01E+003		
					St. Dev	1.23E+004		
					T	4.90E+000	PQ Sort(1.0;1.0)	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-2.16E+000		
					St. Dev	7.00E-001		
					T	-3.08E+001	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-6.57E-001		
					St. Dev	2.49E-001		
					T	-2.64E+001	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	6.68E+002		
					St. Dev	2.89E+003		
					T	2.31E+000	?	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	4.98E+003		
					St. Dev	7.30E+003		
					T	6.82E+000	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.83E+000		
					St. Dev	7.68E-001		
					T	-2.38E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.64E+000		
					St. Dev	7.35E-001		
					T	-2.24E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	4.18E+003		
					St. Dev	6.08E+003		
					T	6.88E+000	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.95E+003		
					St. Dev	4.37E+003		
					T	4.46E+000	PQ Sort(1.0;1.0)	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.95E+000		
					St. Dev	7.15E-001		
					T	-2.73E+001	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.59E+000		
					St. Dev	6.14E-001		
					T	-2.59E+001	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.85E+002		
					St. Dev	3.53E+003		
					T	8.08E-001	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	2.69E+003		
					St. Dev	4.02E+003		
					T	6.68E+000	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.65E+000		
					St. Dev	6.34E-001		
					T	-2.61E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.79E+000		
					St. Dev	6.95E-001		
					T	-2.57E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-3.46E+002		
					St. Dev	4.10E+003		
					T	-8.42E-001	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-8.41E+000		
					St. Dev	2.10E+003		
					T	-4.01E-002	?	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.52E+000		
					St. Dev	6.14E-001		
					T	-2.48E+001	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.65E+000		
					St. Dev	6.68E-001		
					T	-2.46E+001	PQR Sort	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	4.42E+001		
					St. Dev	4.31E+003		
					T	1.03E-001	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	2.87E+002		
					St. Dev	1.93E+003		
					T	1.48E+000	?	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean	-9.25E+000		
					St. Dev	8.46E+000		
					T	-1.09E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.67E+000		
					St. Dev	1.78E+000		
					T	-9.38E+000	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-2.50E+001		
					St. Dev	1.01E+004		
					T	-2.48E-002	?	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-4.65E+003		
					St. Dev	1.81E+004		
					T	-2.57E+000	?	?



## Comparação entre PQR-Sort e PQR-Sort with Sorted Restrictions (matrizes 10x10)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQR-SR	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-8.75E-002		
					St. Dev	3.47E-001		
					T	-7.97E-001	?	?
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-4.60E-002		
					St. Dev	2.18E-001		
					T	-6.68E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.27E+000		
					St. Dev	7.59E+000		
					T	9.45E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-9.90E-001		
					St. Dev	1.50E+001		
					T	-2.09E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-7.22E-002		
					St. Dev	1.99E-001		
					T	-1.15E+000	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-2.80E-002		
					St. Dev	9.33E-002		
					T	-9.49E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-1.40E-001		
					St. Dev	5.69E-001		
					T	-7.78E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-2.55E+000		
					St. Dev	7.98E+000		
					T	-1.01E+000	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	7.47E-002		
					St. Dev	4.22E-001		
					T	5.60E-001	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	8.00E-003		
					St. Dev	3.43E-001		
					T	7.39E-002	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.06E+001		
					St. Dev	1.87E+001		
					T	1.80E+000	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	6.35E+000		
					St. Dev	1.70E+001		
					T	1.18E+000	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	1.18E-002		
					St. Dev	5.03E-001		
					T	7.42E-002	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-5.20E-002		
					St. Dev	4.03E-001		
					T	-4.08E-001	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.21E+001		
					St. Dev	1.88E+001		
					T	2.05E+000	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	5.27E+000		
					St. Dev	1.93E+001		
					T	8.62E-001	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	3.87E-002		
					St. Dev	4.42E-001		
					T	2.77E-001	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-2.00E-003		
					St. Dev	5.01E-001		
					T	-1.26E-002	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	6.83E+000		
					St. Dev	1.81E+001		
					T	1.19E+000	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	4.34E+000		
					St. Dev	1.77E+001		
					T	7.76E-001	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	1.97E-003		
					St. Dev	4.31E-001		
					T	1.45E-002	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.70E-002		
					St. Dev	4.16E-001		
					T	-1.29E-001	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	7.14E+000		
					St. Dev	1.90E+001		
					T	1.19E+000	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	3.78E+000		
					St. Dev	1.85E+001		
					T	6.46E-001	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-1.87E-002		
					St. Dev	1.75E-001		
					T	-3.38E-001	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-1.00E-002		
					St. Dev	9.05E-002		
					T	-3.50E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.00E-002		
					St. Dev	4.49E-001		
					T	1.41E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-1.03E+000		
					St. Dev	9.10E+000		
					T	-3.58E-001	?	?

## Comparação entre PQR-Sort e PQR-Sort with Sorted Restrictions (matrizes 100x100)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQR-SR	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	3.27E-001 2.35E+000 1.39E+000		
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	1.26E-001 1.61E+000 7.86E-001	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	1.33E+004 9.13E+003 1.46E+001	?	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	9.64E+003 1.29E+004 7.46E+000	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	7.90E-001 7.59E-001 1.04E+001	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	2.64E-001 2.54E-001 1.04E+001	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	6.45E+003 3.12E+003 2.07E+001	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	3.06E+003 7.30E+003 4.19E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	3.21E-001 7.06E-001 4.55E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	2.95E-001 6.49E-001 4.55E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	6.72E+003 5.94E+003 1.13E+001	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	2.67E+003 4.18E+003 6.37E+000	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	2.18E-001 6.55E-001 3.33E+000	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	1.89E-001 5.38E-001 3.51E+000	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	3.68E+003 3.02E+003 1.22E+001	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	1.85E+003 3.90E+003 4.74E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	4.72E-002 5.74E-001 8.22E-001	?	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	4.53E-002 6.02E-001 7.52E-001	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	1.42E+003 4.66E+003 3.05E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	1.09E+003 2.20E+003 4.98E+000	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	5.05E-002 6.94E-001 7.28E-001	?	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	4.83E-002 7.44E-001 6.49E-001	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	1.62E+003 3.82E+003 4.25E+000	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	1.42E+003 2.01E+003 7.07E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-4.90E-001 7.48E-001 -6.56E+000	PQR Sort	PQR-Sorted Restrictions
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-2.59E-001 3.66E-001 -7.07E+000	PQR Sort	PQR-Sorted Restrictions
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	4.27E+003 9.80E+003 4.36E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-9.82E+003 2.35E+004 -4.18E+000	PQR Sort	PQR-Sorted Restrictions



## Comparação entre algoritmos que evidenciam padrões globais (matrizes 10x10)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	Sugi - PQR + BC	Sugi - PQR-SR	PQR + BC - PQR-SR
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	3.01E-001	2.17E-001	-8.40E-002
					St. Dev	7.42E-001	8.11E-001	6.52E-001
					T	1.28E+000	8.46E-001	-4.07E-001
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	2.09E-001	2.84E-001	7.50E-002
					St. Dev	4.14E-001	3.94E-001	3.07E-001
					T	1.60E+000	2.28E+000	7.73E-001
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	5.37E+000	5.17E+000	-2.00E-001
					St. Dev	1.05E+001	1.11E+001	9.12E+000
					T	1.61E+000	1.47E+000	-6.94E-002
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	6.78E+000	5.86E+000	-9.20E-001
					St. Dev	1.84E+001	2.25E+001	1.98E+001
					T	1.17E+000	8.25E-001	-1.47E-001
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	2.09E-001	-6.17E-001	-8.26E-001
					St. Dev	5.33E-001	9.54E-001	7.73E-001
					T	1.24E+000	-2.05E+000	-3.38E+000
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	1.78E-001	1.39E-001	-3.90E-002
					St. Dev	2.48E-001	2.33E-001	2.05E-001
					T	2.27E+000	1.89E+000	-6.02E-001
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	2.91E+000	2.62E+000	-2.90E-001
					St. Dev	3.52E+000	3.66E+000	9.98E-001
					T	2.81E+000	2.26E+000	-9.19E-001
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.31E+001	1.73E+001	4.21E+000
					St. Dev	1.52E+001	1.81E+001	1.73E+001
					T	2.73E+000	3.03E+000	7.69E-001
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	1.04E-002	3.08E-001	2.97E-001
					St. Dev	4.49E-001	5.21E-001	5.17E-001
					T	7.29E-002	1.87E+000	1.82E+000
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	1.60E-002	2.96E-001	2.80E-001
					St. Dev	4.02E-001	4.19E-001	4.28E-001
					T	1.26E-001	2.23E+000	2.07E+000
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.01E+000	5.00E-002	-9.60E-001
					St. Dev	1.63E+001	1.95E+001	1.74E+001
					T	1.96E-001	8.13E-003	-1.74E-001
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.80E-001	-2.89E+000	-3.07E+000
					St. Dev	1.68E+001	1.90E+001	1.85E+001
					T	3.39E-002	-4.62E-001	-5.24E-001
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-2.28E-002	3.46E-001	3.69E-001
					St. Dev	5.69E-001	5.41E-001	5.44E-001
					T	-1.27E-001	2.02E+000	2.14E+000
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	2.00E-003	3.09E-001	3.07E-001
					St. Dev	4.31E-001	4.35E-001	4.27E-001
					T	1.47E-002	2.24E+000	2.27E+000
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-6.40E-001	-7.50E-001	-1.10E-001
					St. Dev	1.29E+001	1.53E+001	1.55E+001
					T	-1.56E-001	-1.55E-001	-2.25E-002
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	2.59E+000	-1.26E+000	-3.85E+000
					St. Dev	1.71E+001	1.78E+001	1.74E+001
					T	4.79E-001	-2.24E-001	-6.99E-001
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-4.02E-002	2.59E-001	3.00E-001
					St. Dev	4.08E-001	4.71E-001	4.51E-001
					T	-3.11E-001	1.74E+000	2.10E+000
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-4.50E-002	2.41E-001	2.86E-001
					St. Dev	3.74E-001	4.36E-001	4.50E-001
					T	-3.80E-001	1.75E+000	2.01E+000
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-5.30E-001	-8.90E+000	-8.37E+000
					St. Dev	1.76E+001	2.04E+001	1.86E+001
					T	-9.52E-002	-1.38E+000	-1.42E+000
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	-1.35E+000	-1.14E+001	-1.01E+001
					St. Dev	1.45E+001	1.96E+001	1.82E+001
					T	-2.94E-001	-1.94E+000	-1.74E+000
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	-3.75E-002	2.51E-001	2.88E-001
					St. Dev	4.45E-001	4.72E-001	4.87E-001
					T	-2.66E-001	1.68E+000	1.87E+000
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	-2.20E-002	2.85E-001	3.07E-001
					St. Dev	4.35E-001	4.50E-001	4.60E-001
					T	-1.60E-001	2.00E+000	2.11E+000
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	-5.60E-001	-5.94E+000	-5.38E+000
					St. Dev	1.54E+001	2.08E+001	2.20E+001
					T	-1.15E-001	-9.04E-001	-7.74E-001
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.60E+000	-6.48E+000	-8.08E+000
					St. Dev	1.39E+001	1.82E+001	1.66E+001
					T	3.64E-001	-1.12E+000	-1.54E+000
RECTNOISE	0	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean	8.72E-002	-9.09E-001	-9.96E-001
					St. Dev	4.52E-001	9.04E-001	8.21E-001
					T	6.10E-001	-3.18E+000	-3.84E+000
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean	1.30E-001	9.70E-002	-3.30E-002
					St. Dev	2.18E-001	2.32E-001	2.34E-001
					T	1.88E+000	1.32E+000	-4.45E-001
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean	1.80E+000	1.66E+000	-1.40E-001
					St. Dev	2.36E+000	2.57E+000	1.03E+000
					T	2.41E+000	2.04E+000	-4.28E-001
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean	1.13E+001	1.75E+001	6.17E+000
					St. Dev	1.44E+001	2.03E+001	2.00E+001
					T	2.49E+000	2.73E+000	9.77E-001

## Comparação entre algoritmos que evidenciam padrões globais (100x100)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	Sugi - PQR + BC	Sugi - PQR-SR	PQR + BC - PQR-SR	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	1.82E-001 1.04E+000 1.76E+000	2.63E+000 2.20E+000 1.19E-001	2.44E+000 2.11E+000 1.16E+001		
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	1.03E-001 7.27E-001 1.42E+000	1.45E+000 1.62E+000 8.93E+000	1.35E+000 1.56E+000 8.63E+000	PQR-Sorted Restrictions	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	1.09E+003 4.88E+003 2.23E+000	5.92E+002 1.08E+004 5.48E-001	-4.95E+002 1.04E+004 -4.76E-001	PQR-Sorted Restrictions	?
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-5.08E+002 5.63E+003 -9.02E-001	-3.55E+004 1.76E+004 -2.02E+001	-3.50E+004 1.74E+004 -2.01E+001	?	?
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	1.76E-001 7.32E-001 2.40E+000	4.35E+000 7.76E-001 5.61E+001	4.17E+000 7.22E-001 5.78E+001		
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	6.52E-002 2.60E-001 2.50E+000	1.51E+000 2.62E-001 5.74E+001	1.44E+000 2.56E-001 5.64E+001	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	1.28E+001 1.62E+003 7.86E-002	-7.21E+003 2.70E+003 -2.67E+001	-7.22E+003 2.75E+003 -2.62E+001		
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-2.09E+001 4.90E+003 -4.27E-002	5.30E-001 7.11E+003 7.45E-004	2.15E+001 6.89E+003 3.12E-002	?	?
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-6.67E-002 6.78E-001 -9.84E-001	1.42E+000 7.54E-001 1.88E+001	1.49E+000 7.81E-001 1.90E+001		
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-7.11E-002 6.39E-001 -1.11E+000	1.31E+000 6.56E-001 1.99E+001	1.38E+000 7.20E-001 1.91E+001	PQR-Sorted Restrictions	?
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	-5.91E+001 1.93E+003 -3.00E-001	-5.98E+003 6.03E+003 -8.92E+000	-5.98E+003 6.15E+003 -8.66E+000	?	?
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-3.74E+002 1.81E+003 -2.07E+000	-1.74E+004 7.37E+003 -2.36E+001	-1.70E+004 7.30E+003 -2.35E+001	?	?
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-5.42E-002 6.27E-001 -8.64E-001	1.75E+000 6.74E-001 2.59E+001	1.80E+000 7.92E-001 2.28E+001		
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-2.40E-002 5.18E-001 -4.84E-001	1.53E+000 5.60E-001 2.73E+001	1.55E+000 5.63E-001 2.94E+001	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	2.24E+002 2.28E+003 9.83E-001	-1.06E+004 3.74E+003 -2.84E+001	-1.09E+004 3.76E+003 -2.89E+001	?	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-2.92E+002 2.22E+003 -1.32E+000	-6.94E+003 4.80E+003 -1.44E+001	-6.94E+003 4.73E+003 -1.41E+001	?	?
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	3.52E-002 5.71E-001 6.17E-001	1.16E+000 6.27E-001 1.85E+001	1.12E+000 6.78E-001 1.65E+001		
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	5.14E-002 6.43E-001 8.00E-001	1.33E+000 6.93E-001 1.91E+001	1.27E+000 7.52E-001 1.69E+001	PQR-Sorted Restrictions	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	3.50E+002 2.18E+003 1.61E+000	-1.22E+004 3.90E+003 -3.13E-001	-1.26E+004 3.89E+003 -3.24E-001	?	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-5.61E-001 8.40E+002 -6.68E-001	-1.11E+004 1.93E+003 -5.75E+001	-1.10E+004 1.93E+003 -5.71E+001		
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-6.49E-002 6.18E-001 -1.05E+000	1.13E+000 5.97E-001 1.89E+001	1.19E+000 6.93E-001 1.72E+001		
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-7.53E-002 6.82E-001 -1.10E+000	1.27E+000 6.56E-001 1.94E+001	1.35E+000 7.63E-001 1.76E+001	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	-2.98E+002 3.07E+003 -9.71E-001	-1.35E+004 4.30E+003 -3.14E+001	-1.32E+004 4.30E+003 -3.07E+001	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-5.40E+001 1.34E+003 -4.03E-001	-1.14E+004 2.20E+003 -5.16E+001	-1.13E+004 2.12E+003 -5.34E+001	?	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	7.26E-002 9.79E-001 7.41E-001	3.15E-001 9.70E-001 3.24E+000	2.42E-001 9.99E-001 2.42E+000		
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	2.98E-002 5.20E-001 5.73E-001	2.99E-001 4.94E-001 6.05E+000	2.69E-001 4.94E-001 5.44E+000	PQR-Sorted Restrictions	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	2.52E+003 6.85E+003 3.68E+000	1.89E+003 7.88E+003 2.15E+000	-8.29E+002 6.98E+003 -1.19E+000		
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-2.85E+003 1.99E+004 -1.43E+000	-7.09E+002 2.04E+004 -3.48E-001	2.14E+003 2.14E+004 9.99E-001	?	?

## Comparação entre algoritmos que evidenciam padrões locais (matrizes 10x10)

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQ-Sort	PQR - PQR-SR	PQ-Sort - PQR-SR	Best	Worst
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-6.93E-002 3.60E-001 -6.08E-001	-8.68E-002 3.02E-001 -9.09E-001	-1.74E-002 2.85E-001 -1.93E-001		
RECTNOISE	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-4.20E-002 1.92E-001 -6.92E-001	-6.60E-002 1.83E-001 -1.14E+000	-2.40E-002 1.36E-001 -5.59E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	1.15E+000 7.21E+000 5.05E-001	1.63E+000 6.14E+000 8.39E-001	4.80E-001 4.63E+000 3.28E-001	?	?
RECTNOISE	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-3.90E+000 1.63E+001 -7.58E-001	-4.33E+000 1.56E+001 -8.76E-001	-4.30E-001 1.04E+001 -1.31E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-8.08E-002 3.42E-001 -7.48E-001	-3.00E-002 2.11E-001 -4.49E-001	5.08E-002 2.81E-001 5.72E-001	?	?
RANDOM_MATRIX	0.1	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.60E-002 1.07E-001 -4.73E-001	-1.60E-002 1.12E-001 -4.53E-001	-1.11E-018 2.46E-002 -1.43E-016	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	-3.20E-001 1.51E+000 -6.70E-001	4.00E-002 4.00E-001 3.16E-001	3.60E-001 1.69E+000 6.73E-001	?	?
RANDOM_MATRIX	0.1	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.92E+000 1.12E+001 -5.40E-001	-2.08E+000 1.15E+001 -5.70E-001	-1.60E-001 1.86E+000 -2.72E-001	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	9.19E-002 5.16E-001 5.63E-001	8.01E-002 4.84E-001 5.23E-001	-1.18E-002 4.15E-001 -9.01E-002	?	?
RECTNOISE	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	6.30E-002 4.06E-001 4.90E-001	3.60E-002 3.78E-001 3.01E-001	-2.70E-002 3.27E-001 -2.61E-001	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	9.59E+000 2.12E+001 1.43E+000	7.88E+000 1.73E+001 1.41E+000	-1.91E+000 1.49E+001 -4.06E-001	?	?
RECTNOISE	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	6.05E+000 1.94E+001 9.86E-001	3.73E+000 1.76E+001 6.70E-001	-2.32E+000 1.39E+001 -5.27E-001	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.24E-002 5.06E-001 -7.72E-002	-1.13E-002 4.24E-001 8.44E-002	2.37E-002 4.29E-001 1.74E-001	?	?
RANDOM_MATRIX	0.3	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.00E-002 3.75E-001 -8.44E-002	-2.70E-002 3.71E-001 -2.30E-001	-1.70E-002 3.31E-001 -1.63E-001	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	9.30E+000 1.92E+001 1.53E+000	9.33E+000 1.89E+001 1.74E+000	3.00E-002 1.44E+001 6.58E-003	?	?
RANDOM_MATRIX	0.3	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	6.09E+000 1.98E+001 9.73E-001	4.10E+000 1.77E+001 7.34E-001	-1.99E+000 1.77E+001 -3.55E-001	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	6.45E-003 4.63E-001 4.40E-002	-2.12E-002 4.61E-001 -1.45E-001	-2.76E-002 4.16E-001 -2.10E-001	?	?
RECTNOISE	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-3.00E-003 4.65E-001 -2.04E-002	-3.20E-002 4.42E-001 -2.29E-001	-2.90E-002 4.11E-001 -2.23E-001	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	8.58E+000 2.02E+001 1.34E+000	7.85E+000 2.09E+001 1.19E+000	-7.30E-001 1.33E+001 -1.73E-001	?	?
RECTNOISE	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	9.00E+000 1.83E+001 1.55E+000	9.33E+000 1.86E+001 1.58E+000	3.30E-001 1.34E+001 7.81E-002	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-9.37E-004 5.24E-001 -5.65E-003	7.57E-002 4.62E-001 5.19E-001	7.66E-002 4.46E-001 5.43E-001	?	?
RANDOM_MATRIX	0.5	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-6.80E-002 5.02E-001 -4.28E-001	-2.00E-002 4.64E-001 -1.36E-001	4.80E-002 4.79E-001 3.17E-001	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	8.88E+000 1.74E+001 1.62E+000	9.11E+000 1.84E+001 1.57E+000	2.30E-001 1.36E+001 5.36E-002	?	?
RANDOM_MATRIX	0.5	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	6.01E+000 1.81E+001 1.18E+000	4.52E+000 1.81E+001 8.89E-001	-1.49E+000 1.23E+001 -3.84E-001	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-4.63E-002 2.13E-001 -6.89E-001	-4.63E-002 2.13E-001 -6.89E-001	0.00E+000 0.00E+000 NaN	?	?
RECTNOISE	0	10x10	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-8.00E-003 1.01E-001 -2.50E-001	-9.00E-003 1.01E-001 -2.83E-001	-1.00E-003 1.00E-002 -3.16E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.20E-001 8.68E-001 -4.37E-001	0.00E+000 3.48E-001 0.00E+000	1.20E-001 8.91E-001 4.26E-001	?	?
RECTNOISE	0	10x10	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-3.60E-001 8.72E+000 -1.31E-001	-4.60E-001 8.55E+000 -1.70E-001	-1.00E-001 1.22E+000 -2.60E-001	?	?

## Comparação entre algoritmos que evidenciam padrões locais (matrizes 100x100).

Matrix Type	Density	Dimensions	Evaluator	Coefficient	Measure	PQR - PQ-Sort	PQR - PQR-SR	PQ-Sort - PQR-SR	Best	Worst
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-6.72E+000 2.58E+000 -2.60E+001	1.93E-001 2.25E+000 8.56E-001	6.91E+000 2.16E+000 3.20E+001		? PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-4.03E+000 1.85E+000 -2.17E+001	6.69E-002 1.40E+000 6.19E-001	4.12E+000 1.55E+000 2.65E+001		? PQ Sort(1.0;1.0)
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	2.49E+003 8.43E+003 2.95E+000	1.20E+004 8.12E+003 1.48E+001	9.52E+003 5.70E+003 1.67E+001	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	7.20E+003 2.92E+001 7.22E+000	9.94E+003 2.78E+001 8.46E+000	2.74E+003 2.49E+001 2.90E+000	PQR-Sorted Restrictions	PQR Sort
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-2.05E+000 8.33E-001 -2.46E+001	8.17E-001 7.99E-001 1.02E+001	2.87E+000 7.22E-001 3.97E+001	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-5.08E-001 8.41E+000 -2.08E+001	2.51E-001 3.71E+000 9.07E+000	8.59E-001 -4.90E+000 3.45E+001	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	3.37E+002 3.17E+003 1.06E+000	5.93E+003 2.77E+003 2.14E+001	5.59E+003 2.60E+003 2.15E+001	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.1	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	6.17E+003 7.34E+003 8.41E+000	2.58E+003 6.94E+003 3.71E+000	-3.60E+003 7.35E+003 -4.90E+000	PQ Sort(1.0;1.0)	PQR Sort
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.84E+000 7.61E-001 -2.42E+001	3.25E-001 7.40E-001 4.39E+000	2.17E+000 6.47E-001 3.35E+001	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.63E+000 7.39E-001 -2.21E+001	3.03E-001 7.09E-001 4.27E+000	1.93E+000 6.25E-001 3.09E+001	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	3.62E+003 5.25E+003 6.89E+000	7.05E+003 5.30E+003 1.33E+001	3.43E+003 4.24E+003 8.10E+000	PQR-Sorted Restrictions	PQR Sort
RECTNOISE	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	2.55E+003 3.79E+003 6.73E+000	3.00E+003 4.06E+003 7.38E+000	4.43E+002 3.55E+003 1.25E+000	?	PQR Sort
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.75E+000 6.84E-001 -2.56E+001	2.67E-001 7.31E-001 3.65E+000	2.02E+000 6.22E-001 3.24E+001	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.40E+000 5.52E-001 -2.53E+001	2.20E-001 5.88E-001 3.74E+000	1.62E+000 5.08E-001 3.18E+001	PQR-Sorted Restrictions	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	4.20E+002 3.21E+003 1.33E+000	3.13E+003 3.22E+003 9.70E+000	2.70E+003 3.95E+003 6.84E+000	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.3	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	2.76E+003 4.41E+003 6.27E+000	2.41E+003 4.02E+003 6.00E+000	-3.49E+002 4.49E+003 -7.78E-001	?	PQR Sort
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.60E+000 6.34E-001 -2.53E+001	-4.99E-002 5.63E-001 -8.87E-001	1.55E+000 5.54E-001 2.80E+001	?	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.74E+000 6.92E-001 -2.52E+001	-6.50E-002 6.00E-001 -1.08E+000	1.88E+000 6.05E-001 2.77E+001	?	PQ Sort(1.0;1.0)
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	1.24E+002 4.75E+003 2.61E+001	1.39E+003 4.72E+003 2.94E+000	1.26E+003 3.29E+003 3.83E+000	PQR-Sorted Restrictions	?
RECTNOISE	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	2.60E+002 1.87E+003 1.39E+000	1.16E+003 1.86E+003 6.26E+000	9.04E+002 1.57E+003 5.77E+000	PQR-Sorted Restrictions	?
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.70E+000 6.39E-001 -2.05E+001	2.82E-002 6.26E-001 4.50E-001	1.72E+000 6.65E-001 2.63E+001	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.84E+000 7.00E-001 -2.62E+001	3.95E-002 6.67E-001 5.92E-001	1.88E+000 7.06E-001 2.66E+001	?	PQ Sort(1.0;1.0)
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	-5.38E+002 3.94E+003 -1.36E+000	9.85E+002 4.25E+003 2.32E+000	1.52E+003 3.42E+003 4.45E+000	?	?
RANDOM_MATRIX	0.5	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	4.41E+002 2.27E+003 1.94E+000	1.44E+003 2.16E+003 6.70E+000	1.00E+003 1.62E+003 6.20E+000	PQR-Sorted Restrictions	?
RECTNOISE	0	100x100	Minimal Span Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.10E+001 8.90E+000 -1.24E+001	-4.22E-001 7.62E-001 -5.53E+000	1.06E+001 8.76E+000 1.21E+001	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Minimal Span Loss Function	Simple Matching Coefficient	Mean St. Dev T	-1.88E+000 1.73E+000 -1.09E+001	-2.21E-001 3.73E-001 -5.93E+000	1.66E+000 1.69E+000 9.81E+000	PQR Sort	PQ Sort(1.0;1.0)
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Jaccard Coefficient	Mean St. Dev T	-1.34E+001 1.05E+004 -1.28E-002	3.58E+003 9.40E+003 3.81E+000	3.60E+003 5.01E+003 7.19E+000	PQR-Sorted Restrictions	?
RECTNOISE	0	100x100	Anti-Robinson Loss Function	Simple Matching Coefficient	Mean St. Dev T	-9.86E+003 1.93E+004 -5.12E+000	-7.10E+003 2.95E+004 -3.46E+000	2.76E+003 1.28E+004 2.15E+000	PQR Sort	?

## **Anexo A – Versão Original do Conjunto de dados Íris**

Este anexo exibe a versão original do Íris dataset (FRANK & ASSUNCION, 2010). As medidas de largura e comprimento estão em centímetro.

Altura de Sépala	Largura da Sépala	Altura da Pétala	Largura da Pétala	Espécie
5.10	3.50	1.40		0.20 Iris-setosa
4.90	3.00	1.40		0.20 Iris-setosa
4.70	3.20	1.30		0.20 Iris-setosa
4.60	3.10	1.50		0.20 Iris-setosa
5.00	3.60	1.40		0.20 Iris-setosa
5.40	3.90	1.70		0.40 Iris-setosa
4.60	3.40	1.40		0.30 Iris-setosa
5.00	3.40	1.50		0.20 Iris-setosa
4.40	2.90	1.40		0.20 Iris-setosa
4.90	3.10	1.50		0.10 Iris-setosa
5.40	3.70	1.50		0.20 Iris-setosa
4.80	3.40	1.60		0.20 Iris-setosa
4.80	3.00	1.40		0.10 Iris-setosa
4.30	3.00	1.10		0.10 Iris-setosa
5.80	4.00	1.20		0.20 Iris-setosa
5.70	4.40	1.50		0.40 Iris-setosa
5.40	3.90	1.30		0.40 Iris-setosa
5.10	3.50	1.40		0.30 Iris-setosa
5.70	3.80	1.70		0.30 Iris-setosa
5.10	3.80	1.50		0.30 Iris-setosa
5.40	3.40	1.70		0.20 Iris-setosa
5.10	3.70	1.50		0.40 Iris-setosa
4.60	3.60	1.00		0.20 Iris-setosa
5.10	3.30	1.70		0.50 Iris-setosa
4.80	3.40	1.90		0.20 Iris-setosa
5.00	3.00	1.60		0.20 Iris-setosa
5.00	3.40	1.60		0.40 Iris-setosa
5.20	3.50	1.50		0.20 Iris-setosa
5.20	3.40	1.40		0.20 Iris-setosa
4.70	3.20	1.60		0.20 Iris-setosa
4.80	3.10	1.60		0.20 Iris-setosa
5.40	3.40	1.50		0.40 Iris-setosa
5.20	4.10	1.50		0.10 Iris-setosa
5.50	4.20	1.40		0.20 Iris-setosa
4.90	3.10	1.50		0.10 Iris-setosa
5.00	3.20	1.20		0.20 Iris-setosa
5.50	3.50	1.30		0.20 Iris-setosa
4.90	3.10	1.50		0.10 Iris-setosa
4.40	3.00	1.30		0.20 Iris-setosa
5.10	3.40	1.50		0.20 Iris-setosa
5.00	3.50	1.30		0.30 Iris-setosa
4.50	2.30	1.30		0.30 Iris-setosa
4.40	3.20	1.30		0.20 Iris-setosa
5.00	3.50	1.60		0.60 Iris-setosa
5.10	3.80	1.90		0.40 Iris-setosa
4.80	3.00	1.40		0.30 Iris-setosa
5.10	3.80	1.60		0.20 Iris-setosa
4.60	3.20	1.40		0.20 Iris-setosa
5.30	3.70	1.50		0.20 Iris-setosa
5.00	3.30	1.40		0.20 Iris-setosa
7.00	3.20	4.70		1.40 Iris-versicolor

6.50	3.00	5.80	2.20 Iris-virginica
7.60	3.00	6.60	2.10 Iris-virginica
4.90	2.50	4.50	1.70 Iris-virginica
7.30	2.90	6.30	1.80 Iris-virginica
6.70	2.50	5.80	1.80 Iris-virginica
7.20	3.60	6.10	2.50 Iris-virginica
6.50	3.20	5.10	2.00 Iris-virginica
6.40	2.70	5.30	1.90 Iris-virginica
6.80	3.00	5.50	2.10 Iris-virginica
5.70	2.50	5.00	2.00 Iris-virginica
5.80	2.80	5.10	2.40 Iris-virginica
6.40	3.20	5.30	2.30 Iris-virginica
6.50	3.00	5.50	1.80 Iris-virginica
7.70	3.80	6.70	2.20 Iris-virginica
7.70	2.60	6.90	2.30 Iris-virginica
6.00	2.20	5.00	1.50 Iris-virginica
6.90	3.20	5.70	2.30 Iris-virginica
5.60	2.80	4.90	2.00 Iris-virginica
7.70	2.80	6.70	2.00 Iris-virginica
6.30	2.70	4.90	1.80 Iris-virginica
6.70	3.30	5.70	2.10 Iris-virginica
7.20	3.20	6.00	1.80 Iris-virginica
6.20	2.80	4.80	1.80 Iris-virginica
6.10	3.00	4.90	1.80 Iris-virginica
6.40	2.80	5.60	2.10 Iris-virginica
7.20	3.00	5.80	1.60 Iris-virginica
7.40	2.80	6.10	1.90 Iris-virginica
7.90	3.80	6.40	2.00 Iris-virginica
6.40	2.80	5.60	2.20 Iris-virginica
6.30	2.80	5.10	1.50 Iris-virginica
6.10	2.60	5.60	1.40 Iris-virginica
7.70	3.00	6.10	2.30 Iris-virginica
6.30	3.40	5.60	2.40 Iris-virginica
6.40	3.10	5.50	1.80 Iris-virginica
6.00	3.00	4.80	1.80 Iris-virginica
6.90	3.10	5.40	2.10 Iris-virginica
6.70	3.10	5.60	2.40 Iris-virginica
6.90	3.10	5.10	2.30 Iris-virginica
5.80	2.70	5.10	1.90 Iris-virginica
6.80	3.20	5.90	2.30 Iris-virginica
6.70	3.30	5.70	2.50 Iris-virginica
6.70	3.00	5.20	2.30 Iris-virginica
6.30	2.50	5.00	1.90 Iris-virginica
6.50	3.00	5.20	2.00 Iris-virginica
6.20	3.40	5.40	2.30 Iris-virginica
5.90	3.00	5.10	1.80 Iris-virginica

## Anexo B – Distribuição T de Student

<i>GL/c</i>	<b>90%</b>	<b>80%</b>	<b>70%</b>	<b>60%</b>	<b>50%</b>	<b>40%</b>	<b>30%</b>	<b>20%</b>	<b>10%</b>	<b>5%</b>	<b>2%</b>	<b>1%</b>	<b>0%</b>
1	0,158	0,325	0,51	0,727	1	1,376	1,963	3,078	6,314	12,706	31,821	63,657	636,619
2	0,142	0,289	0,445	0,617	0,816	1,061	1,386	1,886	2,92	4,303	6,965	9,925	31,598
3	0,137	0,277	0,424	0,584	0,765	0,978	1,25	1,638	2,353	3,182	4,541	5,541	12,924
4	0,134	0,271	0,414	0,569	0,741	0,941	1,19	1,533	2,132	2,776	3,747	4,604	8,61
5	0,132	0,267	0,408	0,559	0,727	0,92	1,156	1,476	2,015	2,571	3,365	4,032	6,869
6	0,131	0,265	0,404	0,553	0,718	0,906	1,134	1,44	1,943	2,447	3,143	3,707	5,959
7	0,13	0,263	0,402	0,549	0,711	0,896	1,119	1,415	1,895	2,365	2,365	3,499	5,408
8	0,13	0,262	0,399	0,546	0,706	0,889	1,108	1,397	1,86	2,306	2,896	3,355	5,041
9	0,129	0,261	0,398	0,543	0,703	0,883	1,1	1,383	1,833	2,262	2,821	3,25	4,781
10	0,129	0,26	0,397	0,542	0,7	0,879	1,093	1,372	1,812	2,228	2,764	3,169	4,587
11	0,129	0,26	0,396	0,54	0,697	0,876	1,088	1,363	1,796	2,201	2,718	3,106	4,437
12	0,128	0,259	0,395	0,539	0,695	0,873	1,083	1,356	1,782	2,179	2,681	3,055	4,318
13	0,128	0,259	0,394	0,538	0,694	0,87	1,079	1,35	1,771	2,16	2,65	3,012	4,221
14	0,128	0,258	0,393	0,537	0,692	0,868	1,076	1,345	1,761	2,145	2,624	2,977	4,14
15	0,128	0,258	0,393	0,536	0,691	0,866	1,074	1,341	1,753	2,131	2,602	2,947	4,073
16	0,128	0,258	0,392	0,535	0,69	0,865	1,071	1,337	1,746	2,12	2,583	2,921	4,015
17	0,128	0,257	0,392	0,534	0,689	0,863	1,069	1,333	1,74	2,11	2,567	2,898	3,965
18	0,127	0,257	0,392	0,534	0,688	0,862	1,067	1,33	1,734	2,101	2,552	2,878	3,922
19	0,127	0,257	0,391	0,533	0,688	0,861	1,066	1,328	1,729	2,093	2,539	2,861	3,883
20	0,127	0,257	0,391	0,533	0,687	0,86	1,064	1,325	1,725	2,086	2,528	2,845	3,85
21	0,127	0,257	0,391	0,532	0,686	0,859	1,063	1,323	1,721	2,08	2,518	2,831	3,819
22	0,127	0,256	0,39	0,532	0,686	0,858	1,061	1,321	1,717	2,074	2,508	2,819	3,792
23	0,127	0,256	0,39	0,532	0,685	0,858	1,06	1,319	1,714	2,069	2,5	2,807	3,767
24	0,127	0,256	0,39	0,531	0,685	0,857	1,059	1,318	1,711	2,064	2,492	2,797	3,745
25	0,127	0,256	0,39	0,531	0,684	0,856	1,058	1,316	1,708	2,06	2,485	2,787	3,726
26	0,127	0,256	0,39	0,531	0,684	0,856	1,058	1,315	1,706	2,056	2,479	2,779	3,707
27	0,127	0,256	0,389	0,531	0,684	0,856	1,057	1,314	1,703	2,052	2,473	2,771	3,69
28	0,127	0,256	0,389	0,53	0,683	0,856	1,056	1,313	1,701	2,048	2,467	2,763	3,674
29	0,127	0,256	0,389	0,53	0,683	0,854	1,055	1,311	1,699	2,045	2,462	2,756	3,659
30	0,127	0,256	0,389	0,53	0,683	0,854	1,055	1,31	1,697	2,042	2,457	2,75	3,646
40	0,126	0,255	0,388	0,529	0,681	0,851	1,05	1,303	1,684	2,021	2,423	2,704	3,551
60	0,126	0,254	0,387	0,527	0,679	0,848	1,046	1,296	1,671	2	2,39	2,66	3,46
120	0,126	0,254	0,386	0,526	0,677	0,845	1,041	1,289	1,658	1,98	2,358	2,617	3,373
∞	0,126	0,253	0,385	0,524	0,674	0,842	1,036	1,282	1,645	1,96	2,326	2,576	3,291

**Figura 67 – Tabela da distribuição t de student. Na Figura, a primeira coluna, GL, indica o grau de liberdade. E, a primeira linha, C, indica o nível de confiança. Tabela extraída de Magalhaes & Lima (2002).**