UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

Tese de Mestrado

Algoritmos para o Problema de Roteamento de Leituristas

Fábio Luiz Usberti

Campinas – São Paulo – Brasil
Junho de 2007

Fábio Luiz Usberti

Algoritmos para o Problema de Roteamento de Leituristas

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Automação.

Orientador: Prof. Dr. Paulo Morelato França

Campinas – São Paulo – Brasil
Junho de 2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Usberti, Fábio Luiz
Us17a Algoritmos para o problema de roteamento de leituristas / Fábio Luiz Usberti. --Campinas, SP: [s.n.], 2007.

Orientador: Paulo Morelato França
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Programação heurística. 2. Otimização combinatória. 3. Pesquisa operacional. 4. Teoria dos grafos. I. França, Paulo Morelato. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Algorithms for the routing meter readers problem
Palavras-chave em Inglês: Arc routing problem, Rural postman problem, Combinatorial optimization, Operational research
Área de concentração: Automação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Christiano Lyra Filho, Alysson Machado Costa
Data da defesa: 06/06/2007
Programa de Pós-Graduação: Engenharia Elétrica

"Pois como a criação do universo é a suprema perfeição e trabalho de um Criador de suprema sabedoria, nada no universo tem lugar sem que uma regra de máximo ou mínimo se apresente",

(Leonhard Euler, 1707 – 1783).

AGRADECIMENTOS

Ao Criador de suprema sabedoria, por ter possibilitado a mim um lugar nesse universo de suprema perfeição, agradeço com o máximo das minhas forças sujeito ao que minha fé permite.

Aos meus pais, Roberto Usberti e Maria Helena Cavallari Usberti, por todo o amor e apoio incondicionais e a quem devo cada fio do meu cabelo, meu muitíssimo obrigado!

Ao meu amigo e orientador, Paulo Morelato França, por quem me inspiro como exemplo de profissional, professor e ser humano, agradeço por todos os ensinamentos e pela confiança depositada em mim.

Ao meu grande amor, Fran, que com um único olhar consegue esvaecer todos meus problemas, e ao Gú, por tornar a vida mais alegre.

Aos meus irmãos, Ro e Bea, por todos os anos de convívio, brincadeiras, risadas, brigas e tudo mais que proporcionou a formação do meu caráter.

A queridíssima avó Eline, pelos carinhos e conselhos sempre edificantes e por ser um modelo de vida a ser seguido.

Aos amigos, prof. André Luiz Morelato França e prof. Vinícius Jacques Garcia, pela intensa troca de idéias que em muito facilitou esse trabalho.

Ao amigo, prof. Luiz Henrique Antunes Rodrigues, que me apresentou à Pesquisa Operacional e me encorajou a trilhar por essa fascinante área do conhecimento.

A prof[a] Anamaria Gomide, por me ensinar os primeiros passos de programação e a quem muito devo meu encanto pela computação.

Aos colegas do DENSIS, que compartilharam das minhas dificuldades, não tão difíceis quando enfrentadas pelo convívio mútuo.

A CAPES, pelo financiamento desse trabalho, o que possibilitou sua realização.

RESUMO

Esse trabalho se dedicou ao estudo dos algoritmos para roteamento de leituristas, incluindo propostas de alteração que resultem na melhoria da qualidade dos resultados. A motivação é proveniente da alta demanda por soluções computacionais para esse problema, ainda pouco estudado devido às peculiaridades que lhe são inerentes. Encontram-se na literatura duas heurísticas, de estratégias distintas e antagônicas para esse problema. Uma das heurísticas procura construir a rota ignorando a restrição de capacidade, para posterior particionamento dessa rota em subrotas, cada qual destinada a um leiturista ("route first, cluster second"). A outra heurística, em uma abordagem inversa, primeiramente subdivide a região de trabalho dos leituristas, para posterior roteamento dessas partições ("cluster first, route second"). Essas duas heurísticas foram testadas exaustivamente, tornando possível localizar aspectos sujeitos à melhoria, dando origem a duas novas heurísticas. Foi gerada uma base de testes contendo 144 instâncias que simulam as condições reais de trabalho dos leituristas, classificadas de acordo com o tamanho e dificuldade. A partir das soluções provenientes dos quatro algoritmos foi possível analisá-los comparativamente, avaliando o melhor em um âmbito geral (envolvendo todos os algoritmos) e específico (algoritmos de mesmo tipo, "route first cluster second" ou "cluster first route second"), segundo critérios de qualidade pré-definidos: número de rotas, tempo de percurso, violação da carga horária e tempo computacional. Os resultados revelam que os novos algoritmos foram melhores tanto na comparação específica quanto na comparação geral.

ABSTRACT

This work's main study object consists on algorithms for routing meter readers, from which proposals towards solution's improvement are made. The demand for computational results concerning this problem, added to literature little attention due to its inherited peculiarities, has been the outmost motivation. Two preexisting heuristics from literature, with distinct and antagonic strategies, are pointed out. One of these heuristics atempt to create a single route, dismissing the capacity restriction, and then partitionates this route into subroutes, each of them destinated to one meter reader (route first, cluster second). The other heuristic, in an inverse approach, first splits the meter reader's working area, and only then routes each of these partitions (cluster first, route second). The two heuristics were tested to exaustion, allowing enumeration of weak aspects subject to improvement. Therefore, two new heuristics were developed, based upon the originals, however adapted in order to outperform solution's quality. A testing base containing 144 instances was generated, simulating meter readers realistic labor's conditions, classified by size and difficulty. Through solutions provided by the four algorithms, comparison analyses have taken place, evaluating in a general (involving all algorithms) and specific manner (same kind algorithms, i.e., route first, cluster second or cluster first, route second), considering four predefined quality criteria: number of routes, deadheading time, violation of shiftwork time and computational time. Results revealed that the new algorithms achieved better solutions on specific and general comparisons.

LISTA DE FIGURAS

LISTA DE TABELAS

ÍNDICE

I. INTRODUÇÃO

O roteamento de leituristas consiste em um importante problema de natureza prática, envolvendo grandes empresas fornecedoras de energia elétrica, água e gás, nas quais trabalhadores são incumbidos de registrar o consumo de cada cliente, dispersos geograficamente nas ruas e avenidas de uma localidade. Esse problema, de difícil tratamento e elevado número de restrições, prevê a elaboração de um conjunto de rotas abertas, pelas quais os leituristas devem percorrer, atendendo a demanda por serviço de leitura. Uma solução ótima para esse problema envolve minimizar as seguintes variáveis:

- Número de leituristas necessários para atender toda demanda;
- Tempo percorrido sem leitura, ou seja, o tempo utilizado pelo leiturista para se deslocar de uma região a outra sem realizar qualquer leitura;
- Violação da carga horária do leiturista.

A motivação desse trabalho provém da demanda por soluções computacionais para esse problema de roteamento, sobretudo por parte das empresas que se utilizam desse serviço, em virtude de diversos fatores, como:

- Dificuldade de obter soluções empíricas (sem auxílio computacional ou pretensões otimizantes), o que é muito comum na prática;
- Baixa qualidade das soluções obtidas por métodos empíricos;
- Necessidade de novas soluções no curto prazo para atender à rápida mudança do cenário consumidor, sobretudo nas metrópoles, onde se verifica uma acentuada verticalização do espaço geográfico.

O roteamento de leituristas envolve um conjunto de particularidades inerentes, tais como: modo como o serviço de leitura se realiza (U ou Z, i.e., atravessando ou não a rua); presença de segmentos de rua que não requerem leitura; tolerâncias máxima e mínima de carga horária; diferença entre ler e percorrer um segmento de rua.

A natureza desse problema permite generalizá-lo como um problema de roteamento em arcos que, por sua vez, consiste em determinar um subconjunto ordenado de arestas E' sobre um grafo $G(V,E)$ ($E' \subseteq E$), atendendo a possíveis restrições (de demanda, capacidade, dentre outras) e minimizando determinada função objetivo. Ao problema de roteamento em arcos encontra-se ligado uma vasta árvore de subproblemas, dos quais o mais conhecido é o

Problema do Carteiro Chinês (CPP), que se caracteriza em determinar uma rota de peso mínimo, que passe ao menos uma vez por todas as arestas de uma rede, e que pode ser resolvido de maneira exata em tempo polinomial.

Um problema derivado do CPP consiste no Problema do Carteiro Rural (RPP) onde somente um subconjunto de suas arestas precisam ser visitadas. Esse problema pertence à classe NP-Completo, o que lhe caracteriza como complexo e de difícil tratamento até à otimalidade Adicionando ao RPP uma restrição de capacidade, torna-se ainda mais difícil resolvê-lo. Verifica-se que o problema de roteamento de leituristas é bastante similar ao RPP capacitado.

Foram constatadas duas heurísticas na literatura para roteamento de leituristas, que utilizam abordagens distintas, antagônicas inclusive, para tratar do problema. Esses algoritmos foram originalmente desenvolvidos para resolver os problemas locais nos quais estavam inseridos, logo são dotados de características específicas não compartilhadas. Nesse trabalho esses algoritmos foram denominados RFCS1 (“Route First, Cluster Second 1”) e CFRS1 (“Cluster First, Route Second 1”).

O estudo desses algoritmos possibilitou a detecção de pontos fracos que afetavam diretamente na qualidade das soluções. Nesse sentido, são propostas desse trabalho duas novas heurísticas, RFCS2 e CFRS2, que preservam a essência das originais, alterando, porém, aspectos passíveis de melhoria.

No intuito de avaliar o desempenho dos algoritmos, foi elaborado um conjunto diversificado de instâncias, que procuram simular as condições de trabalho dos leituristas.

Para realizar a análise comparativa dos algoritmos, foram estabelecidos quatro critérios de qualidade de solução: penalidade média, número de rotas, tempo de percurso e tempo de execução. Foram destacados os algoritmos que obtiveram o maior número de vitórias na comparação de médias dos critérios expostos, tanto na comparação geral (envolvendo os quatro algoritmos) quanto na comparação específica (para algoritmos de mesmo tipo, RFCS ou CFRS).

A contribuição desse trabalho consiste em fornecer uma estratégia eficiente para o rotemento de leituristas, a qual poderá ser utilizada pelas empresas (do setor elétrico, água e esgoto, gás ou dos correios) que se utilizam desse serviço e apresentam demanda por soluções de qualidade em um tempo computacional relativamente reduzido.

II. REVISÃO BIBLIOGRÁFICA

A revisão bibliográfica deste trabalho, dividida em cinco seções, pretende ser abrangente o suficiente para tornar-se autocontida, no sentido que um leitor com noções básicas de conjuntos possa compreender seu conteúdo sem grande dificuldade. A seção 1 faz uma introdução à teoria dos grafos, explicando os principais conceitos dessa importante estrutura matemática. A seção 2 aborda alguns problemas importantes relacionados aos grafos, assinalando as principais estratégias de resolução. A seção 3 menciona as estruturas de dados comumente utilizadas para representar os grafos computacionalmente. A seção 4 considera a questão dos problemas de roteamento, mencionando a classe de roteamento em nós e em seguida descrevendo com maiores detalhes a classe de roteamento em arcos, que é o objeto de estudo desse trabalho. Por fim, a seção 5 explora dois casos onde foram utilizados os conceitos de roteamento em arcos capacitado para tratar dois problemas reais de roteamento de leituristas.

1. Introdução à Teoria dos Grafos

A primeira seção será dedicada à definição de grafo, seus tipos, propriedades e operações. Serão abordados dois tipos de grafos de grande importância teórica e prática, sobretudo para os problemas de roteamento em nós e arcos: os grafos Euleriano e Hamiltoniano. Para maiores informações sobre o assunto tratado nessa seção, são sugeridas as seguintes referências [25][48][49][51], além daquelas citadas individualmente ao longo do capítulo.

1.1. Definição e Histórico

Um grafo simples G consiste em um par (V,E), onde V é um conjunto finito e não-vazio de nós ou vértices ($V = \{v_1, v_2, ..., v_n\}$) e E é um conjunto finito de pares não ordenados de V denominados arestas ($E = \{e_1, e_2, ..., e_m\}$), ou seja, $E \subseteq V^2$. Dois vértices v_i e v_j são considerados adjacentes quando existe uma aresta $e \in E$, tal que $e = (v_i,v_j)$. Nesse caso, a aresta e é considerada incidente aos vértices v_i e v_j.

Apesar de se tratar de uma estrutura matemática abstrata, os grafos são fáceis de ser representados graficamente. A representação mais usual consiste em desenhar pontos para

os vértices e linhas ligando os pontos para as arestas. Um exemplo dessa representação pode ser observado na Figura 1, à direita.

Em geral, os problemas que envolvem redes elétricas, mapas rodoviários, representação de substâncias químicas, planejamento de projetos, dentre muitos outros, são potenciais aplicação de grafos.

O primeiro problema tratado com uma representação de grafo é atribuído ao matemático Leonhard Euler, quando resolveu o famoso problema das "Sete Pontes de Königsberg". Essas sete pontes foram os objetos de estudo de Gribkovskaia, Halskau e Laporte [27] que fazem um belo relato histórico. No século XVIII, a cidade de Königsberg, Prússia, situava-se às margens do rio Pregel, possuindo duas grandes ilhas ligadas entre si e o continente por sete pontes, como mostra a Figura 1. Atualmente a cidade se chama Kaliningrad, Rússia, e algumas das pontes já não existem mais, por terem sido destruídas durante a 2ª Guerra Mundial.

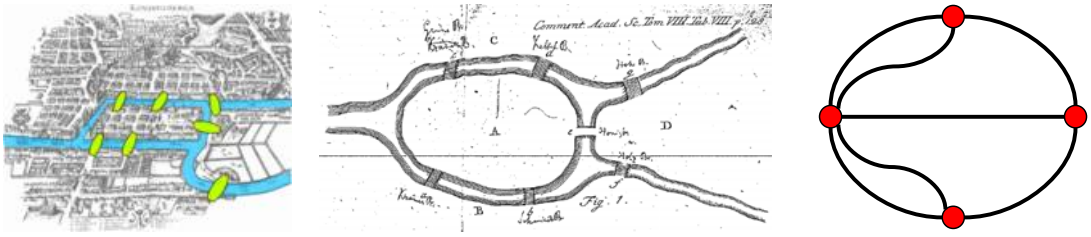


Figura 1. Mapa da cidade de Königsberg (esquerda); esboço original do problema por Leonhard Euler [17] (centro); grafo representativo do problema (direita).

O problema das "sete pontes de Königsberg" consistia em descobrir se era possível criar uma rota que, iniciando e terminando no mesmo ponto, atravessasse todas as pontes uma única vez. Em 1736, Euler [17], na primeira publicação abordando o conceito de grafos, provou que essa rota não existia e ainda revelou a condição necessária e suficiente para que tal rota exista em um grafo (vide seção 1.5).

1.2. Tipos de Grafo

1.2.a. Dígrafos, Grafos Mistos e Multigrafos

Um grafo direcionado, ou dígrafo, consiste em um grafo $G(V,A)$ onde A é um conjunto de pares ordenados de V, denominados arcos. Em um dígrafo, os vértices v_i e v_j, que representam um arco $a_u = \{v_i,v_j\}$, estão ordenados de modo que v_i é denominado vértice

inicial e v_j vértice terminal do arco, definindo assim um sentido pelo qual esses vértices são visitados. Essa propriedade do arco o diferencia da aresta, pois essa última não tem um sentido definido. Assim, duas arestas $e_u = \{v_i, v_j\}$ e $e_v = \{v_j, v_i\}$ são tais que $e_u \equiv e_v$, no entanto, dois arcos $a_u = \{v_i, v_j\}$ e $a_v = \{v_j, v_i\}$, são tais que $a_u \neq a_v$.

Existem casos onde um grafo possui tanto arcos quanto arestas, característica que o denomina grafo misto e, nesse caso, é denotado por $G(V,A,E)$.

Quando um grafo possui mais de uma aresta (ou arco) entre dois vértices ele passa a ser considerado um multigrafo.

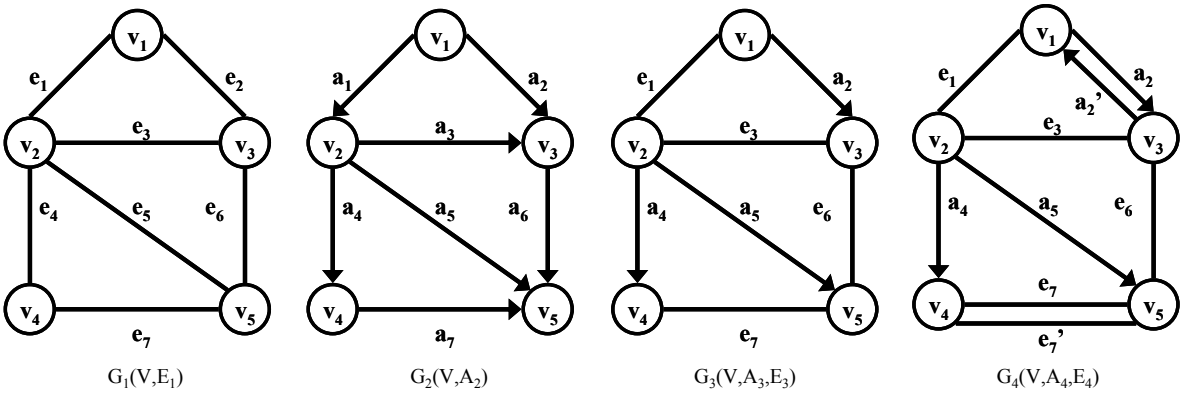


Figura 2. Grafo simples G_1, direcionado G_2, misto G_3 e multigrafo G_4.

Os conjuntos de vértices e arestas dos grafos G_1, G_2, G_3 e G_4 da Figura 2 podem estão descritos a seguir:

$V = \{v_1,v_2,v_3,v_4,v_5\}$;

$E_1 = \{e_1,e_2,e_3,e_4,e_5,e_6,e_7\}$; $A_2 = \{a_1,a_2,a_3,a_4,a_5,a_6,a_7\}$; $E_3 = \{e_1,e_3,e_6,e_7\}$;

$A_3 = \{a_2,a_4,a_5\}$; $E_4 = \{e_1,e_3,e_6,e_7,e_7'\}$; $A_4 = \{a_2,a_2',a_4,a_5\}$;

$e_1 = (v_1,v_2)$; $e_2 = (v_1,v_3)$; $e_3 = (v_2,v_3)$; $e_4 = (v_2,v_4)$;

$e_5 = (v_2,v_5)$; $e_6 = (v_3,v_5)$; $e_7 = (v_4,v_5)$; $e_7' = (v_4,v_5)$;

$a_1 = (v_1,v_2)$; $a_2 = (v_1,v_3)$; $a_2' = (v_3,v_1)$; $a_3 = (v_2,v_3)$;

$a_4 = (v_2,v_4)$; $a_5 = (v_2,v_5)$; $a_6 = (v_3,v_5)$; $a_7 = (v_4,v_5)$;

1.2.b. Grafos Completos

Um grafo completo consiste em um grafo onde todos os vértices são adjacentes entre si. A notação utilizada para um grafo completo de n vértices é $K_n = (V,V^2)$. Pode-se

verificar que o número de arestas de um grafo completo K_n é determinável pela equação $C_{n,2} = \frac{n(n-1)}{2}$. Na Figura 3 pode-se observar as representações dos grafos $K_1,...,K_8$.

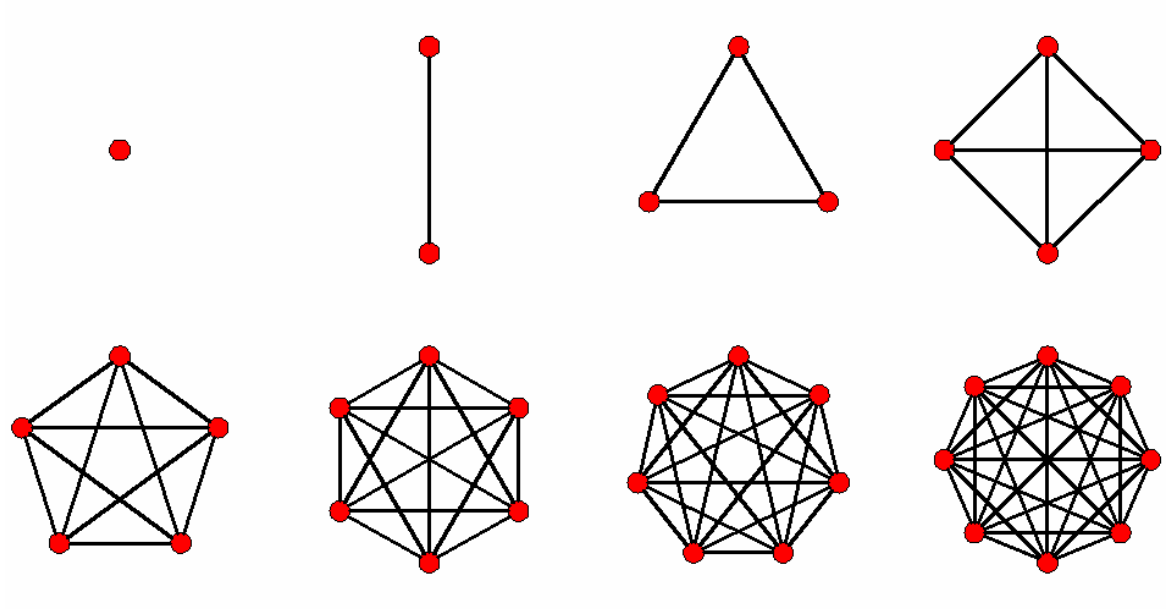


Figura 3. Grafos completos com 1 até 8 vértices (K_1 a K_8).

1.2.c. Grafos Complementares

Seja dado um grafo $G(V,E_1)$. Será definido o complemento de G, $\overline{G}(V,E_2)$, o grafo que possuir as seguintes propriedades:

- O conjunto V de G é igual ao de $\overline{G}$.
- Se o conjunto de V possuir n vértices, então $G \cup \overline{G} = K_n$ (vide seção 1.2.b).
- $G \cap \overline{G} = \varnothing$.
- $\overline{\overline{G}} = G$.

A Figura 4 torna mais claro o conceito de grafos complementares, mostrando o exemplo de dois grafos mutuamente complementares e a união dos dois, gerando um grafo completo K_6. Verifica-se que para obter o complementar de um grafo qualquer de n vértices, basta remover do grafo completo K_n as arestas do grafo original.

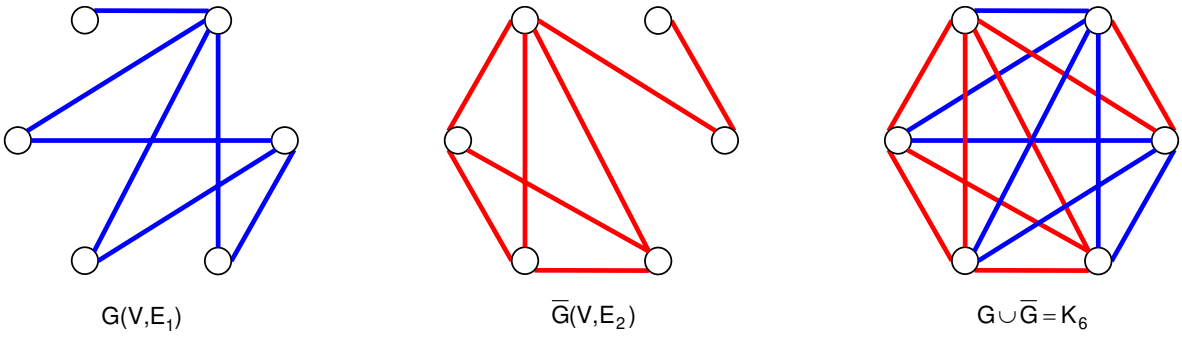


Figura 4. Exemplo de um grafo *G* de seis vértices, seu complementar $\overline{G}$ e a união dos dois, gerando um grafo completo K_6.

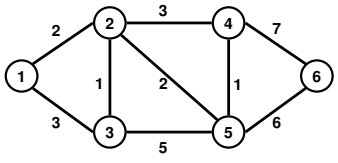
1.2.d. Redes

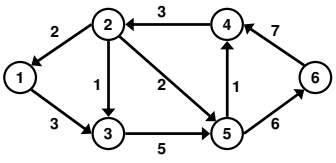
Algumas aplicações utilizam grafos com pesos (ou custos) associados em cada aresta. São os grafos ponderados, também denominados redes. Portanto, define-se uma rede como um grafo *G(V,E)* com uma função peso w associada a cada aresta, com imagem nos reais ($w: V^2 \rightarrow \mathbb{R}$). A imagem da função peso pode ser descrita como:

$$w(u,v)\begin{cases} \geq 0 \text{ para } (u,v) \in E \\ = 0 \text{ para } u = v \\ = \infty \text{ para } (u,v) \notin E \end{cases}$$

Dependendo da aplicação, as redes podem ser direcionadas ou não, e suas representações podem ser feitas de acordo com a Figura 5, onde os pesos são valores reais associados a cada aresta. A partir desse exemplo, podem ser construídas as matrizes de peso W_1 (rede não-direcionada) e W_2 (rede direcionada).

$$W_1 = \begin{bmatrix} 0 & 2 & 3 & \infty & \infty & \infty \\ 2 & 0 & 1 & 3 & 2 & \infty \\ 3 & 1 & 0 & \infty & 5 & \infty \\ \infty & 3 & \infty & 0 & 1 & 7 \\ \infty & 2 & 5 & 1 & 0 & 6 \\ \infty & \infty & \infty & 7 & 6 & 0 \end{bmatrix}$$





$$W_2 = \begin{bmatrix} 0 & \infty & 3 & \infty & \infty & \infty \\ 2 & 0 & 1 & \infty & 2 & \infty \\ \infty & \infty & 0 & \infty & 5 & \infty \\ \infty & 3 & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 1 & 0 & 6 \\ \infty & \infty & \infty & 7 & \infty & 0 \end{bmatrix}$$

Figura 5. Rede não-direcionada (esquerda) e direcionada (direita).

1.3. Propriedades e Operações em um Grafo

1.3.a. Grau ou valência de um vértice

O número de arestas incidentes a um vértice v_i é chamado grau ou valência do vértice e é denotado por $d(v_i)$. Um vértice com grau zero é denominado um vértice isolado, pois ele não é adjacente a nenhum outro vértice que não ele mesmo. Quando um grafo possui somente vértices de grau par/ímpar, ele é denominado um grafo par/ímpar. Para o grafo G_1 da Figura 2, os graus dos vértices são:

$d(v_1) = 2$ $d(v_2) = 4$ $d(v_3) = 3$ $d(v_4) = 2$ $d(v_5) = 3$

Em um dígrafo, distingue-se o grau de saída $d^+(v_i)$ (número de arestas que saem do vértice) e o grau de entrada $d^-(v_i)$ (número de arestas que entram no vértice). Um grafo é considerado simétrico quando os graus de entrada e saída são iguais para todo v_i. Quando um grafo (dígrafo ou grafo misto) for simultaneamente par e simétrico, ele é considerado um grafo balanceado. Para o dígrafo G_2 representado na Figura 2, os graus de entrada e de saída correspondem a:

$d^+(v_1) = 2$ $d^+(v_2) = 3$ $d^+(v_3) = 1$ $d^+(v_4) = 1$ $d^+(v_5) = 0$

$d^-(v_1) = 0$ $d^-(v_2) = 1$ $d^-(v_3) = 2$ $d^-(v_4) = 1$ $d^-(v_5) = 3$

Para um dígrafo, o grau do vértice consiste na soma do grau de entrada e de saída $d(v_i) = d^+(v_i) + d^-(v_i)$ e é equivalente ao grau do vértice considerando o grafo como não-direcionado.

1.3.b. Subgrafos

Considerando um grafo $G = (V,E)$, pode-se denominar $G' = (V',E')$ um subgrafo de G se V' for um subconjunto de V e E' um subconjunto de E, ou seja, $V' \subseteq V$ e $E' \subseteq E$. No caso onde todos os vértices de G estão presentes em G' ($V' = V$), pode-se dizer que G' consiste em um subgrafo gerador de G.

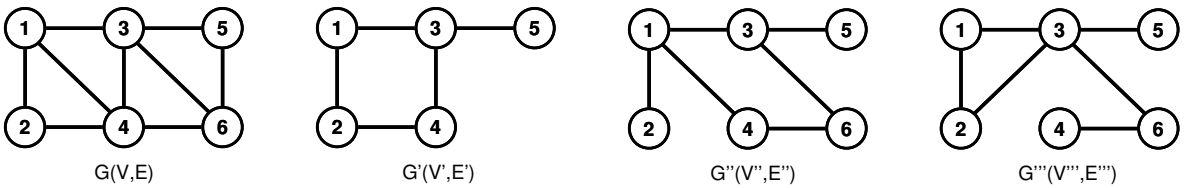


Figura 6. Grafo G, subgrafo G', subgrafo gerador G'', grafo G'''.

Na Figura 6 estão representados quatro grafos G, G', G'' e G'''. Pode-se observar que G' é um subgrafo de G, pois $V' \subset V$ e $E' \subset E$. Por sua vez, G'' pode ser considerado um subgrafo gerador de G pois $V'' = V$ e $E' \subset E$. E finalmente G''', que apesar de possuir todos os vértices de G, não pode ser considerado um subgrafo de G, tampouco um subgrafo gerador, devido à presença da aresta *(2,3)*, que não se encontra no grafo G original.

1.3.c. Caminhos, trilhas, ciclos e circuitos

Um caminho de um grafo $G = (V,E)$ consiste em uma seqüência finita e alternada de vértices e arestas $(v_0,e_1,v_1,e_2,...,v_{k-1},e_k,v_k)$ sempre começando e terminando por vértices de modo que v_{i-1} e v_i são os vértices terminais da aresta e_i, $1 \leq i \leq k$. Alguns casos particulares de caminhos devem ser mencionados:

Caminho simples: ocorre quando todos os vértices pertencentes ao caminho são distintos.

Trilha: ocorre quando todas as arestas pertencentes ao caminho são distintas.

Caminho simples fechado ou *Ciclo*: ocorre quando os nós terminais de um caminho simples consistem no mesmo nó.

Trilha fechada ou *Circuito*: ocorre quando os nós terminais de uma trilha consistem no mesmo nó.

Rota: termo utilizado para expressar tanto um ciclo quanto um circuito.

1.3.d. Particionamento

O particionamento $\{V_1,V_2,...,V_p\}$ de um grafo $G(V,A)$ consiste em dividir seu conjunto de vértices V, em p subconjuntos, tal que:

$V_1 \cup V_2 \cup ... \cup V_p = V$

$V_i \cap V_j = \varnothing$ ($\forall i,j,\ i \neq j$)

1.3.e. Conexidade

Dois vértices v_i e v_j são ditos conexos quando existe no grafo um caminho de v_i para v_j. Um grafo é dito conexo quando, para todos os pares de vértices i e j, existe um caminho unindo-os. Quando um grafo $G(V,A)$ é não-conexo podemos particionar o conjunto V em subconjuntos $V_1, V_2, ..., V_p$ de modo que cada subgrafo V_k ($k = 1, 2, ..., p$) é conexo e

nenhum vértice em V_s está conectado com qualquer vértice do subgrafo V_t, $s \neq t$. Nessas condições, os subgrafos V_i ($i = 1, 2, ..., p$) são denominados componentes conexos de G.

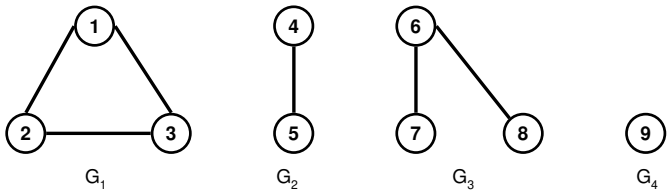


Figura 7. Grafo G com componentes conexos G_1, G_2, G_3 e G_4.

O grafo G da Figura 7 é um exemplo de grafo desconexo. Seus quatro componentes conexos G_1, G_2, G_3 e G_4 possuem conjuntos de vértices $V_1 = \{1,2,3\}$, $V_2 = \{4,5\}$, $V_3 = \{6,7,8\}$ e $V_4 = \{9\}$, respectivamente. O subgrafo G_4, apesar de possuir somente um vértice, também deve ser tratado como um componente conexo, pois por definição um nó sempre está conectado a si mesmo.

1.3.f. Isomorfismo

Dois grafos $G_1(V_1,E_1)$ e $G_2(V_2,E_2)$ são considerados isomórficos quando há uma correspondência vértice a vértice entre os dois grafos, de modo que se $u \in V_2$ é o correspondente de $v \in V_1$, então os vértices adjacentes de u são correspondentes aos adjacentes de v. Em outras palavras, o isomorfismo é uma propriedade que caracteriza dois grafos como equivalentes. A Figura 8 mostra cinco grafos aparentemente distintos que, no entanto, podem ser considerados isomórficos ou equivalentes ao grafo completo K_6.

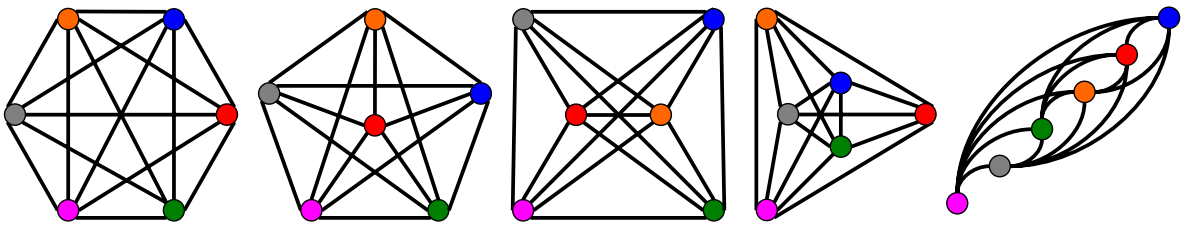


Figura 8. Cinco grafos isomorfos ao K_6.

1.3.g. União e Interseção de Grafos

As operações de união e interseção em grafos consistem na aplicação dessas operações sobre os conjuntos de vértices e arestas. Assim, dados dois grafos $G_1(V_1,E_1)$ e $G_2(V_2,E_2)$, a operação de união sobre eles $G_1 \cup G_2$ resulta em um grafo $G_\cup(V_\cup,E_\cup)$, tal que:

$V_\cup = V_1 \cup V_2$

$E_\cup = E_1 \cup E_2$

Considerando novamente G_1 e G_2, a operação de interseção $G_1 \cap G_2$ resulta em um grafo $G_\cap(V_\cap,E_\cap)$, tal que:

$V_\cap = V_1 \cap V_2$

$E_\cap = E_1 \cap E_2$

A Figura 9 a seguir mostra as operações de união e interseção sobre dois grafos $G_1(V_1,E_1)$ e $G_2(V_2,E_2)$. Pode-se perceber que a união dos dois grafos resultou em um grafo de cinco vértices e, pelo fato de possuírem arestas em comum, a interseção não resultou em vazio, gerando um grafo de dois componentes conexos (vide seção 1.3.e).

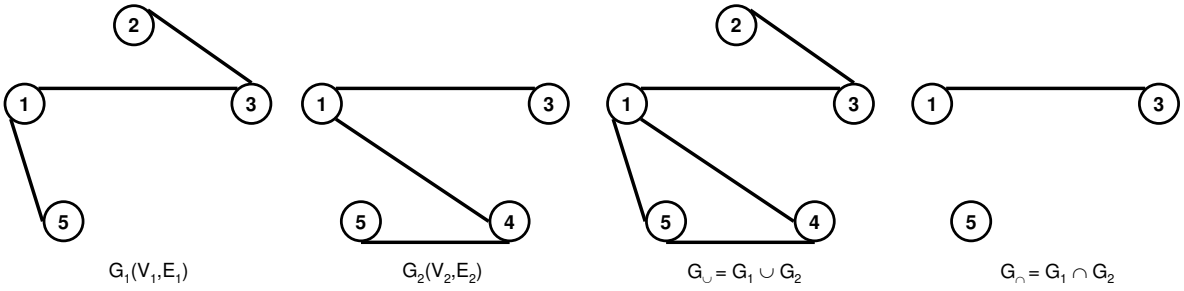


Figura 9. Operações de união e interseção sobre dois grafos $G_1(V_1,E_1)$ e $G_2(V_2,E_2)$.

1.3.h. Deleções e Contrações

A deleção de um vértice v_i de um grafo $G(V,E)$, denotado por $G - v_i$, consiste em remover v_i do seu conjunto de vértices V e remover as arestas (v_k,v_j) do conjunto E para todo $k = i$ ou $k = j$. Assim, a deleção de um vértice de G implica, além da sua remoção do conjunto de vértices, a remoção de todas as arestas incidentes ao mesmo.

A deleção de uma aresta $e_i = (v_j,v_k)$ do grafo G, denotado por $G - e_i$, consiste simplesmente na remoção dessa aresta do conjunto E do grafo.

A contração é uma operação realizada em uma aresta $e_i = (v_j,v_k)$ do grafo G, denotada por $G \setminus e_i$, consistindo na deleção dessa aresta seguido pelo ajuntamento dos vértices v_j e v_k aos quais e_i estava incidente. Assim, esses dois vértices passam a ser representados por um único vértice v_l, que passa a ser adjacente aos vértices antes adjacentes a v_i e v_k.

A Figura 10 exibe uma seqüência de três operações sobre um grafo completo $G_1 = K_5$. Primeiramente é feito a deleção do vértice {1}, seguido da deleção da aresta {(2,4)} e finalmente a contração da aresta {(3,4)}.

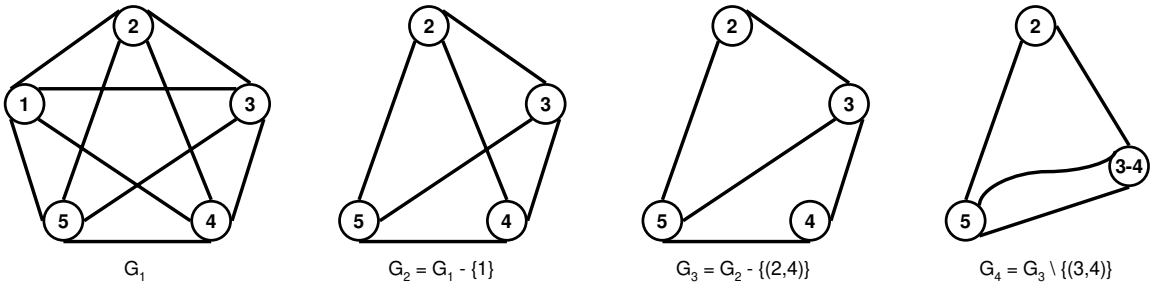


Figura 10. Operações de deleção e contração sobre um grafo.

1.4. Árvores Geradoras

As árvores podem ser definidas como grafos conexos (vide seção 1.3.e) que não possuem ciclos (vide seção 1.3.c). Uma floresta, por sua vez, consiste ou em uma árvore ou em um grafo desconexo acíclico, onde cada componente conexo corresponde a uma árvore. Seja *G(V,E)* um grafo com *n* vértices. As seguintes afirmações são equivalentes:

- *G* é uma árvore;
- *G* não possui circuitos e possui *n-1* arestas;
- *G* é conexo e possui *n-1* arestas;
- Todo arranjo de vértices tomados dois a dois de *G* está conexo por somente um caminho;
- *G* não contém nenhum circuito, mas a adição de qualquer nova aresta forma exatamente um circuito em *G*.

A partir dos nós e arestas de um grafo *G(V,E)* sempre é possível construir ao menos uma árvore, sendo essa um subgrafo de *G* (vide seção 1.3.b). Quando a árvore possui o mesmo conjunto *V* de *G*, então se define uma árvore geradora de *G*.

O grafo K_4 possui 16 árvores geradoras que podem ser observadas na Figura 11.

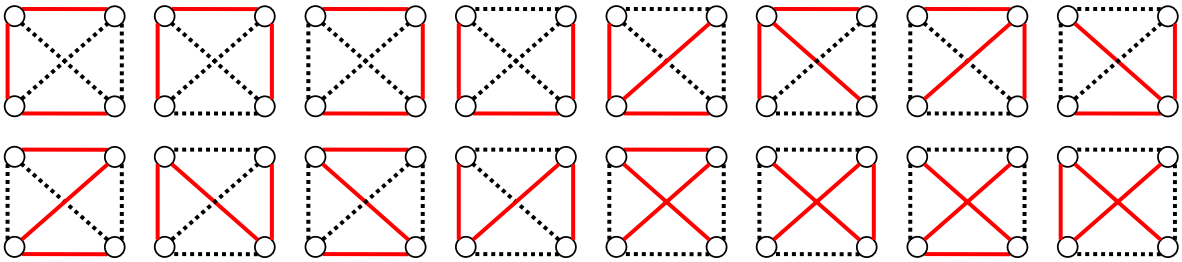


Figura 11. Todas as árvores geradoras (linha contínua) do grafo K_4.

1.5. Trilhas, Circuitos e Grafos Eulerianos

Uma trilha Euleriana em um grafo *G(V,E)* consiste em uma trilha (vide seção em 1.3.c) contendo todas as arestas de *G*. Um circuito Euleriano consiste em um circuito (vide seção em 1.3.c) contendo todas as arestas de *G*. Um grafo que possui um circuito Euleriano é denominado grafo Euleriano. Um grafo contendo uma trilha Euleriana é denominado grafo semi-Euleriano.

Em 1736, Euler [17] descreveu a condição necessária e suficiente para que um grafo *G(V,E)* seja Euleriano. A condição era que todos os vértices do grafo devem possuir grau par (vide seção 1.3.a). Posteriormente, foram descobertas as condições necessárias e suficientes para que grafos direcionados *G(V,A)* e mistos *G(V,A,E)* fossem também considerados Eulerianos. Eiselt, Gendreau e Laporte [15] descrevem essas condições do seguinte modo:

- Um grafo não-direcionado é Euleriano se e somente se ele for par ($d(v_i) = 2n$, $v_i \in V$, $n \in \mathbb{N}^*$).
- Um grafo direcionado é Euleriano se e somente se ele for simétrico ($d^+(v_i) = d^-(v_i)$, $v_i \in V$).

Um grafo misto é Euleriano se e somente se para cada subconjunto de vértices $S \subseteq V$, a diferença entre o número de arcos de S para $V \setminus S$ e o número de arcos de $V \setminus S$ para S deve ser menor ou igual ao número de arestas unindo S e $V \setminus S$.

No que diz respeito ao grafo semi-Euleriano, a condição necessária e suficiente para sua existência consiste nele possuir exatamente dois vértices com grau ímpar.

Voltando ao problema das sete pontes de Könisberg (vide seção 1.1), pode-se concluir que o problema efetivamente não possui solução, pois como mostra a Figura 1, o grafo representativo do problema era ímpar.

Na Figura 12 são observados três grafos G_{NE}, G_{SE} e G_E, cujos vértices de grau ímpar se encontram destacados. O grafo G_{SE}, possuindo dois vértices com grau ímpar (*3* e *4*), pode ser classificado como um grafo semi-Euleriano, o que garante a existência de uma trilha Euleriana. O grafo G_E, não possuindo nenhum nó de grau ímpar, consiste em um exemplo de grafo Euleriano, o que garante a existência de um circuito Euleriano. O grafo G_{NE}, por sua vez, possuindo um número de vértices com grau ímpar diferente de zero ou dois, não pode ser considerado Semi-Euleriano, tampouco Euleriano, restando-lhe por exclusão a classe de grafos não-Eulerianos.

Ainda com respeito à Figura 12, pode-se observar a construção de uma trilha Euleriana para o grafo G_{SE}, com a seguinte seqüência de nós: *{3,2,1,4,3,1,0,2,4}*. Para o grafo G_E foi construído um circuito Euleriano, onde a visitação dos nós se deu na seguinte ordem: *{5,3,2,1,4,3,1,0,2,4,5}*.

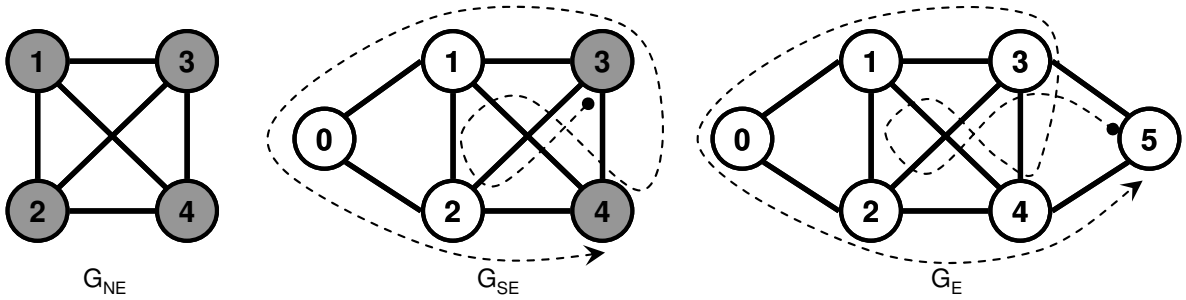


Figura 12. Grafo não-Euleriano G_{NE}, semi-Euleriano G_{SE} e Euleriano G_E (nós destacados com grau ímpar).

1.6. Caminhos, Ciclos e Grafos Hamiltonianos

Um caminho Hamiltoniano em um grafo *G(V,E)* consiste em um caminho (vide seção 1.3.c) que visita todos os nós de *G*. Um ciclo Hamiltoniano consiste em um ciclo (vide seção 1.3.c) que visita todos os nós de *G*. Um grafo contendo um ciclo Hamiltoniano é denominado um grafo Hamiltoniano. Um grafo contendo um caminho Hamiltoniano é denominado um grafo Semi-Hamiltoniano. Todo grafo Hamiltoniano é também semi-

Hamiltoniano, pois dado um ciclo Hamiltoniano, basta remover-lhe uma aresta qualquer para gerar um caminho Hamiltoniano.

Embora exista uma condição necessária e suficiente para a existência de um grafo Euleriano, o mesmo não ocorre para o grafo Hamiltoniano. A determinação de tal condição necessária e suficiente para a existência de grafos Hamiltonianos consiste em um dos maiores problemas abertos da teoria dos grafos. Até o momento existem somente teorias suficientes para a existência desse tipo de grafo, ou seja, condições que garantem que o grafo é Hamiltoniano, porém não garantem que todos os grafos Hamiltonianos respeitam essas condições.

O teorema de Dirac [48] consiste em uma condição suficiente para a existência de um grafo Hamiltoniano. Esse teorema diz que se os graus dos vértices de um grafo forem todos maiores ou iguais à metade do número de vértices, então o grafo é Hamiltoniano. Ou seja, dado *G(V,E)*, se para qualquer $v \in V$ ocorrer $d(v) \geq \frac{n}{2}$, então *G(V,E)* é Hamiltoniano.

Na Figura 13 constam três exemplos simples de grafos, G_{NH} (não-Hamiltoniano), G_{SH} (semi-Hamiltoniano) e G_H (Hamiltoniano). Pode-se observar um caminho Hamiltoniano *{3,1,2,4}* para o grafo G_{SH} e um ciclo Hamiltoniano *{3,1,2,4}* para o grafo G_H.

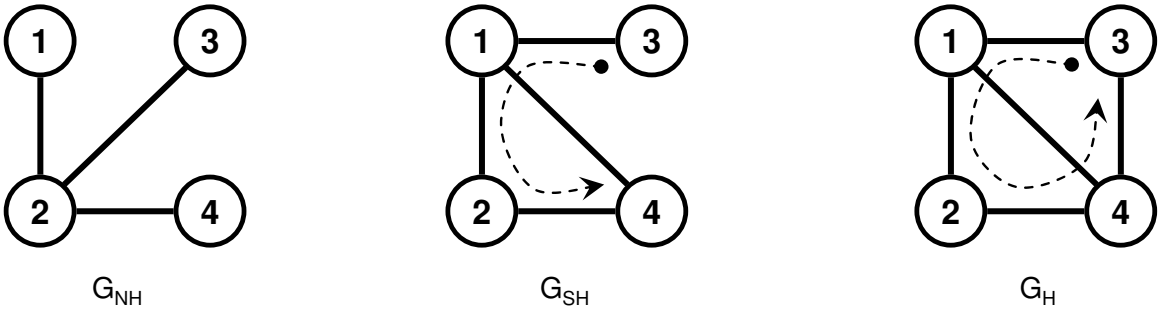


Figura 13. Grafo não-Hamiltoniano G_{NH}, semi-Hamiltoniano G_{SH} e Hamiltoniano G_H.

2. Algoritmos para Grafos

A seguir são apresentados algoritmos para problemas clássicos de grafos, envolvendo árvore geradora mínima, caminho mínimo e emparelhamento máximo de custo mínimo. Para cada um desses problemas são apresentados diferentes algoritmos, bastante

difundidos na literatura, dentre os quais um deles foi utilizado para o problema de roteamento de leituristas, segundo um critério de facilidade na implementação.

2.1. Árvore Geradora Mínima

Dada uma rede $G(V,E)$, com um peso $w(e) \in \mathbb{R}$ associado a cada aresta $e \in E$ ($w(e) = \infty$, para $e \notin E$), a árvore geradora mínima de G consiste em uma árvore $T(V,E_T) \subseteq G$, tal que a soma dos pesos de suas arestas seja mínimo. A Figura 11 ilustra uma rede com pesos associados às arestas, onde pode-se observar ao centro, em linha contínua, uma árvore geradora não-mínima de peso 53, e à direita uma a árvore geradora mínima, com peso 38.

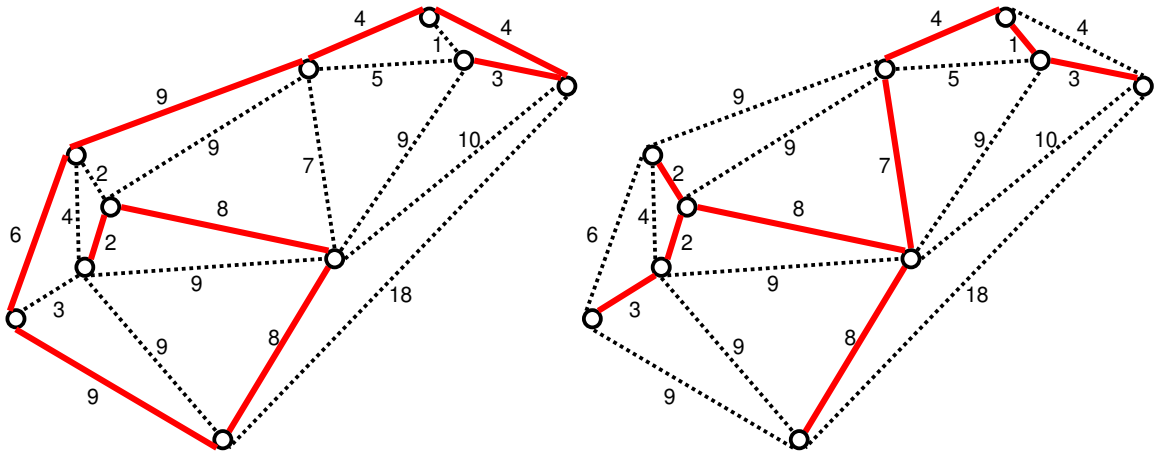


Figura 14. Árvore geradora em linha contínua (esquerda) e árvore geradora mínima em linha contínua (à direita).

Existem diversos algoritmos na literatura que tratam desse problema. Nesse trabalho serão comentados os três algoritmos que mais se destacam: o algoritmo de Prim [43], algoritmo de Kruskal [31] e algoritmo de Boruvka (descrito em [28]). Seus algoritmos serão transcritos a seguir:

2.1.a. Algoritmo de Prim $O(|V|^2)$

```
Passo 1. Adicione à arvore T a aresta de menor peso do grafo;
Passo 2. Repita até que a árvore T possua n−1 arestas:
      Passo 2.1. Adicione à T a aresta de menor peso que ligue um vértice
      de T com um vértice de G \ T.
```

2.1.b. Algoritmo de Kruskal *O(|E| log |V|)*

Passo 1. Crie uma lista ordenada *L* de forma crescente das arestas de *G* pelos seus pesos.

Passo 2. Repita até que a árvore *T* possua *n*−1 arestas:

Passo 2.1. Adicione à *T* a próxima aresta da lista *L* (a de menor peso).

Passo 2.2. Se ocorrer formação de ciclo em *T*, remova a última aresta inserida em *T*.

2.1.c. Algoritmo de Boruvka *O(|E| log |V|)*

Passo 1. Crie uma floresta *F* do grafo *G*, inicialmente vazia.

Passo 2. Repita para cada vértice *v* de *G*.

Passo 2.1. Se *e* for a aresta de menor peso incidente em *v*.

Passo 2.2. Faça a união de *T* com a aresta *e*.

Passo 3. Crie *G'* como sendo *G* com as arestas de *T* contraídas (1.3.h).

Passo 4. Recursivamente compute *T'* como sendo a árvore geradora mínima de *G'*.

Passo 5. Retorne *T* + *T'*.

2.2. Caminho Mínimo

Dada uma rede *G(V,E)*, com um peso *w(e)* associado a cada aresta $e \in E$ ($w(e) = \infty$, para $e \notin E$), o problema do caminho mínimo consiste em encontrar um caminho $P \subseteq E$ ligando dois vértices *u* e *v* do grafo tal que a soma dos pesos das arestas de *P* seja mínimo:

$$\min_P \left[w(P) = \sum_{e \in P} w(e) \right].$$

A resolução do problema de caminho mínimo pode ser aplicável diretamente em diversas situações reais, mas também pode ser útil como parte de problemas maiores, por exemplo, nas heurísticas de roteamento, onde os algoritmos de caminhos mínimos são utilizados rotineiramente. Em algumas ocasiões é desejável a obtenção do caminho mínimo de um nó para todos os demais, ou ainda de todos os nós para todos os nós.

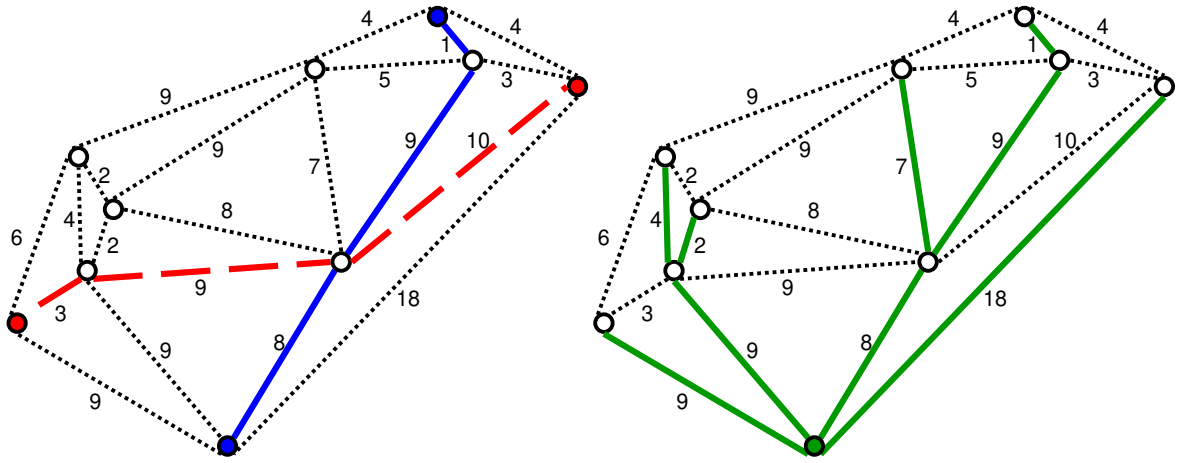


Figura 15. Caminhos mínimos entre dois pares de nós (destacados à esquerda) e os caminhos mínimos de um nó (destacado à direita) para todos os nós.

A Figura 15 mostra, à esquerda, os caminhos mínimos unindo dois pares de nós: um deles representado pela linha contínua, com peso total 18, e outro representado pela linha tracejada, com peso total 22. À direita, verificam-se os caminhos mínimos unindo um nó (destacado) a todos os outros nós.

Frente aos inúmeros algoritmos que tratam do problema do caminho mínimo, será feito uma descrição de três, que são muito utilizados: o algoritmo de Djikstra (de um nó para todos os nós, [12]), algoritmo de Floyd-Warshall (de todos os nós para todos os nós, [18]) e o algoritmo matricial revisado de Hu (de todos os nós para todos os nós, [29]). O algoritmo de Djikstra, apesar de obter o caminho mínimo de um nó para todos os demais, é facilmente adaptável para obter os caminhos mínimos de todos para todos os nós, bastando para isso rodar o algoritmo uma vez para cada nó. Esses três algoritmos possuem complexidade computacional $O(V^3)$ (quando se busca os caminhos mínimos de todos para todos os nós). A seguir serão feitas as apresentações dos três algoritmos, considerando para o caso do algoritmo de Djikstra que a busca está sendo feita de todos para todos os nós. Nos três algoritmos, os caminhos mínimos serão armazenados na matriz $Dist_{nxn}$, onde n é o número de nós do grafo.

2.2.a. Algoritmo de Djikstra

```
Passo 1. Para cada u ∈ V faça
      Passo 1.1. S = {u}
      Passo 1.2. Para cada v ∈ V faça
```

```
            Passo 1.2.1. Dist[u,v] = w(u,v)
      Passo 1.3. Enquanto S ≠ V faça
            Passo 1.3.1. Escolha w ∈ V \ S tal que
                  Dist[u,w] = MIN Dist[u,y] (∀y ∈ V \ S)
            Passo 1.3.2. S = S ∪ {w}
            Passo 1.3.3. Para cada y ∈ V \ S faça
                  Passo 1.3.3.1.
                        Dist[u,y] = MIN(Dist[u,y],Dist[u,w]+w[w,y])
```

2.2.b. Algoritmo de Floyd-Warshall

```
Passo 1. Para cada u ∈ V faça
      Passo 1.1. Para cada v ∈ V faça
      Passo 1.2. Dist[u,v] = w[u,v]
Passo 2. Para cada u ∈ V faça
      Passo 2.1. Para cada v ∈ V faça
      Passo 2.2. Se Dist[v,u] < ∞ faça
            Passo 2.2.1 Para cada w ∈ V faça
                  Passo 2.2.1.1. Se Dist[u,w] < ∞ faça
                        Passo 2.2.1.1.1. Soma = Dist[v,u] + Dist[u,w]
                        Passo 2.2.1.1.2. Se Soma < Dist[v,w] faça
                              Dist[v,w] = Soma
```

2.2.c. Algoritmo Matricial Revisado

O algoritmo matricial revisado consiste em uma técnica aperfeiçoada por Hu [29] que surgiu após os algoritmos de Djikstra [12] e Floyd [18]. Esse algoritmo obtém os caminhos mínimos de todos para todos os nós, utilizando uma operação especial de multiplicação de matrizes, definida por Hu da seguinte forma:

$$C = [c_{ij}] = A \otimes B \Rightarrow c_{ij} = \min_{k}\left(a_{ik} + b_{kj}\right)$$

Em palavras, o operador $\otimes$ pega uma linha da matriz A e uma coluna da matriz B e soma elemento a elemento, tomando somente o menor valor encontrado e inserindo-o no elemento da matriz C cuja posição é definida pela linha e coluna tomadas de A e B, respectivamente. É possível aplicar esse operador em uma matriz por ela mesma. No caso,

será considerada a matriz de distâncias (ou pesos) de uma rede, $D = [d_{ij}]$. Sendo assim, tem-se:

$$D^2 = [d_{ij}^2] = D \otimes D \Rightarrow d_{ij}^2 = \min_k \left(d_{ik} + d_{kj} \right)$$

Hu [29] mostrou que D^2 é a matriz de caminhos mínimos de todos para todos os nós utilizando no máximo dois arcos apenas. Assim, dois nós *u* e *v*, distantes três arcos um do outro, teriam $d_{uv}^2 = \infty$. Do mesmo modo, D^4 consiste na matriz de caminhos mínimos utilizando até quatro arcos. Seguindo essa lógica, para garantir a obtenção da matriz de caminhos mínimos, independente do número de arcos, deve-se continuar o processo de multiplicação por *k* vezes, levando a D^{2^k}. Como $2^k = |E| - 1$ (onde $|E|$ é o número de arestas do grafo), torna-se necessário $k = \log_2(n-1)$.

O processo descrito acima consiste no algoritmo matricial para a obtenção dos caminhos mínimos de todos para todos os nós. No entanto, Hu [29] revisou esse algoritmo e introduziu a seguinte modificação: quando uma entrada da matriz produto D^2 é calculada, imediatamente ela substitui os valores das duas matrizes que formam o produto antes que o próximo elemento da matriz produto seja calculado. Desse modo, torna-se importante a ordem na qual os elementos da matriz produto serão determinados. O autor estabelece duas ordens, o processo para frente $\otimes_F$ e o processo para trás $\otimes_B$.

No processo para frente, os elementos da matriz produto são determinados começando da esquerda para direita, de cima para baixo. Para representar esse novo processo matematicamente, define-se d_{ij}^F as entradas da matriz produto calculada pelo processo para frente e d_{ij}^0 as entradas da matriz original. O processo para frente pode então ser definido como:

$$D^F = D \otimes_F D$$
$$d_{ik}^F = \min_j \left(d_{ij}^p + d_{jk}^q \right)$$

onde:

$p = 0$ se $k \leq j$

$p = F$ se $k > j$

$q = 0$ se $i \leq j$

$q = F$ se $i > j$

No processo para trás, os elementos da matriz produto são determinados começando da direita para esquerda, de baixo para cima. A representação matemática desse processo, tendo-se d_{ij}^{B} como as entradas da matriz produto calculada pelo processo para trás, pode ser feita de maneira análoga ao processo para frente:

$$D^{B} = D \otimes_{B} D$$
$$d_{ik}^{B} = \min_{j}\left(d_{ij}^{p} + d_{jk}^{q}\right)$$

onde:

$p = 0 \quad \text{se} \quad j \le k$

$p = B \quad \text{se} \quad j > k$

$q = 0 \quad \text{se} \quad i \le j$

$q = B \quad \text{se} \quad i > j$

Por fim, Hu [29] prova que, aplicando um processo para frente seguido de um processo para trás na matriz de distâncias de uma rede, ter-se-á uma matriz *Dist* de caminhos mínimos (de todos para todos os nós). Portanto o algoritmo matricial de caminho mínimo se resume na seguinte linha:

$$Dist = D^{F} \otimes_{B} D^{F}$$

2.3. Emparelhamento Máximo de Peso Mínimo

Gabow [21] define um emparelhamento *M* como um subconjunto de arestas de um grafo *G(V,E)*, de modo que duas arestas de *M* não compartilham de um mesmo vértice. Um grafo emparelhado *(G,M)* consiste em um grafo contendo um emparelhamento. Um vértice é considerado emparelhado quando uma das arestas de *M* lhe é incidente. Um emparelhamento é considerado máximo M_{max} quando não existe outro emparelhamento com um número maior de arestas do que M_{max}. Por fim, um emparelhamento máximo de peso mínimo consiste em um emparelhamento máximo de uma rede *G(V,E)* (vide seção 1.2.d), de modo que a soma dos pesos das arestas do emparelhamento seja mínimo, ou seja, $\min_{M_{\max}}\left[w(M_{\max}) = \sum_{e \in M_{\max}} w(e) \right]$.

Para a obtenção de um emparelhamento máximo torna-se necessário definir os conceitos de “caminho alternante” e “caminho de aumento”. De acordo com Gabow [21],

um caminho alternante consiste em um caminho $P = \{v_0,e_1,v_1,...,e_n,v_n\}$ de um grafo emparelhado de modo que tomando duas arestas consecutivas quaisquer e_i e e_{i+1} ($1 \leq i \leq n-1$), somente uma delas pertencerá ao emparelhamento, enquanto a outra não. Um caminho de aumento P_A, por sua vez, consiste em um caminho alternante cujos nós terminais v_0 e v_n não estejam emparelhados.

Pettie & Sanders [42] explicam que a partir de um caminho alternante $P = \{vi,ei+1,vi+1,...,ej,vj\}$ de um grafo emparelhado (G,M), é possível trocar as arestas emparelhadas pelas não-emparelhadas. Quando essas trocas resultam em um novo emparelhamento M', esse processo passa a se denominar "operação de troca", cujo símbolo é $\oplus$, e pode ser formulado da seguinte maneira:

$$M' = P \oplus M = (P \setminus M) \cup (M \setminus P)$$

Pelo fato de haver troca de arestas do emparelhamento, o peso desse emparelhamento pode se alterar. É possível determinar o ganho de peso $g(M,P)$ decorrente da aplicação de uma operação de troca entre um caminho alternante P e um emparelhamento M a partir da equação:

$$w(M') - w(M) = g(M,P) = w(P \setminus M) - w(P \cap M)$$

Na condição do caminho alternante P ser um caminho de aumento P_A, Gabow [21] mostra que a operação de troca $M_A = P_A \oplus M$ resulta sempre em um novo emparelhamento M_A, que possui uma aresta a mais do que M. Nessa condição, a operação de troca passa a se chamar "operação de aumento". Uma operação de aumento, por sua vez, é sempre possível quando um grafo emparelhado possui um caminho de aumento, e decorre inevitavelmente no aumento de uma aresta do emparelhamento de G, o que justifica seu nome.

Em resumo, uma operação de troca é realizada a partir de um caminho alternante e possibilita alterar o peso do emparelhamento sem necessariamente alterar seu número de arestas; uma operação de aumento, realizada com um caminho de aumento, provoca o aumento de uma unidade no número de arestas do emparelhamento e possibilita também alterar seu peso.

Quando um caminho alternante ou de aumento é constituído de k arestas não-emparelhadas, ele é denominado caminho k-alternante (ou de k-aumento se for o caso). Define-se ciclo k-alternante como um caminho k-alternante que inicia e termina no mesmo

vértice. A operação de troca ou de aumento realizada com esses caminhos passa a se denominar operação de *k*-troca ou de *k*-aumento, respectivamente.

Berge [4] provou que um emparelhamento só é máximo quando ele não possui nenhum caminho de aumento. Nesse sentido, são comuns os algoritmos que encontram um emparelhamento máximo pela busca contínua de caminhos de aumentos, até que não seja possível mais qualquer processo de aumento.

A Figura 16 a seguir esclarece esses conceitos. À esquerda é dado um exemplo de emparelhamento que não é máximo, pois ainda é possível emparelhar dois de seus nós. Pelo teorema de Berge, como o emparelhamento não é máximo, deve haver algum caminho de aumento nesse grafo. De fato, há um caminho 2-aumento, como revela a Figura 16. Como relatado anteriormente, a partir de um caminho de aumento é possível realizar uma operação de aumento pela troca das arestas emparelhadas pelas não emparelhadas, obtendo-se assim um novo emparelhamento com uma aresta emparelhada a mais. A Figura 16 mostra a operação 2-aumento e como ela constrói o novo emparelhamento, que agora é máximo (todos os nós emparelhados).

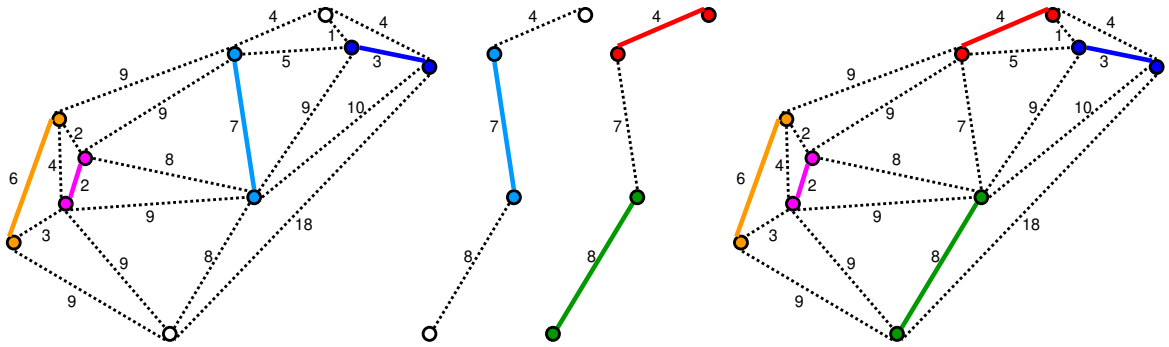


Figura 16. Sucessivamente, da esquerda para a direita: um emparelhamento (linhas contínuas); um caminho 2-aumento; uma operação 2-aumento; um emparelhamento máximo.

É importante ressaltar que o fato do emparelhamento ser máximo não implica que ele seja de mínimo custo. De fato, será mostrado posteriormente que o emparelhamento máximo da Figura 16, com peso total 23, não é de peso mínimo.

O primeiro algoritmo polinomial que encontra um emparelhamento máximo de custo mínimo foi desenvolvido por Edmonds [13], possuindo complexidade computacional $O(|V|^4)$, detalhadamente descrito em [14]. Gabow [21] conseguiu reduzir a complexidade computacional do algoritmo de Edmonds para $O(|V|^3)$ a partir da modificação da estrutura

de dados utilizada em seu algoritmo. Atualmente, o algoritmo de menor complexidade computacional, dada por $O(n(m+n\log(n)))$, para a resolução desse problema é atribuído a Gabow [22], que também a partir do algoritmo de Edmonds, obteve um melhor desempenho a partir do ajuste da estrutura de dados do algoritmo. Considerando que os pesos das arestas da rede são todos iguais, ou seja, quando o problema de emparelhamento máximo não é ponderado, pode-se atingir uma complexidade computacional $O(\sqrt{|v|}|E|)$ utilizando o algoritmo de Micali e Varizani [34].

De acordo com Drake e Hougardy [50], muitos problemas reais exigem grafos de tal ordem de grandeza que, na prática, mesmo os algoritmos de emparelhamento mais rápidos tornam-se ineficientes com relação ao tempo computacional. Além disso, existem diversas aplicações com grafos de médio porte onde o problema de emparelhamento precisa ser resolvido com muita freqüência. Nesse sentido, esses autores falam da necessidade de algoritmos aproximativos de emparelhamento que sejam rápidos, produzam soluções de boa qualidade (ainda que não ótimas) e que sejam de fácil implementação comparada com as implementações do algoritmo de Edmonds e Johnson [14].

Sejam M^{aprox} e M^* os emparelhamentos obtidos por algoritmos aproximativo e ótimo, respectivamente. Drake e Hougardy [50] explicam que a qualidade da solução de um algoritmo aproximativo é determinada a partir de um fator de eficiência $\delta \in \mathbb{R}^*$, tal que $w(M^*) \leq w(M^{aprox}) \leq \delta w(M^*)$ para o problema de minimização e $\frac{1}{\delta} w(M^*) \leq w(M^{aprox}) \leq w(M^*)$ para o problema de maximização.

Existe um algoritmo construtivo guloso para o emparelhamento máximo de mínimo custo bastante divulgado pela literatura e que funciona da seguinte maneira:

```
Passo 1. Ordene em um lista L as arestas do grafo G pelos seus pesos de
maneira crescente
Passo 2. Repita enquanto o emparelhamento M não for máximo:
      Passo 2.1. Selecione a próxima aresta da lista L (a de menor peso).
      Passo 2.2. Se a aresta escolhida for incidente a algum nó já
      emparelhado, descarte a aresta. Caso contrário, coloque-a no
      emparelhamento M.
```

Avis [1] provou que o fator de eficiência desse algoritmo guloso é $\delta = 2$, ou seja, a solução desse algoritmo fornecerá um peso até duas vezes maior do que o peso mínimo. A complexidade desse algoritmo é a complexidade de ordenar as arestas pelos seus pesos, ou seja, *O(|E|log|E|)*.

Drake e Hougardy [50] desenvolveram um algoritmo com fator de eficiência $\delta = \frac{3}{2} + \varepsilon$ e complexidade $O(\frac{|E|}{\varepsilon})$, onde ε é um valor arbitrário tal que, quanto menor for seu valor, melhor será a solução e pior o tempo computacional. Esse algoritmo possui uma estratégia de busca em vizinhança, de modo que a partir de uma solução inicial, obtida por exemplo pelo algoritmo guloso, são feitas trocas de arestas emparelhadas por arestas vizinhas não emparelhadas que resultem na melhoria da solução.

Pettie e Sanders [42] afirmam que o algoritmo de Drake-Hougardy não é claro e requer uma análise muito detalhada, além disso, a solução converge lentamente para $\left[\frac{3}{2} + \varepsilon\right] w(M^*)$. Nesse sentido, os autores desenvolveram um algoritmo aleatório com fator de eficiência $\delta = \frac{3}{2} + \varepsilon$ e complexidade $O(|E| \log \frac{1}{\varepsilon})$, portanto convergindo exponencialmente mais rápido que o algoritmo de Drake-Hougardy. Dado à simplicidade e eficiência de seu algoritmo aleatório, Pettie e Sanders consideram-no uma excelente opção na prática.

O algoritmo de Pettie e Sanders [42] é bastante simples e funciona basicamente a partir de sucessivas operações 2-troca em um grafo contendo um emparelhamento máximo (obtido, por exemplo, pelo algoritmo guloso). A seguir será apresentada uma descrição do algoritmo:

```
Passo 1. Repita por k iterações:
      Passo 1.1. Escolha aleatoriamente um nó do grafo.
      Passo 1.2. Busque todos os ciclos 2-alternantes adjacentes ao nó
      escolhido.
      Passo 1.3. Faça a operação 2-troca no ciclo que produzir a maior
      perda de peso de M, ou seja, o menor g(M,P).
```

A Figura 17 ilustra melhor o princípio de funcionamento do algoritmo de Pettie e Sanders [42]. Nela é possível encontrar (à esquerda), um emparelhamento máximo, o mesmo da Figura 16 (à direita). Resumidamente, o algoritmo vai procurar, por todo o grafo, ciclos 2-alternantes e verificar se a operação 2-troca resulta em uma redução do peso total do emparelhamento. A Figura 17 revela a presença de um ciclo com essa característica. Após a execução da operação 2-troca, percebe-se que as novas arestas emparelhadas possuem um peso três unidades abaixo das arestas anteriores, o que corresponde exatamente à redução pela qual o emparelhamento como um todo obteve. Portanto, o novo emparelhamento máximo possui agora um peso 20. Testes posteriores revelaram que esse novo emparelhamento máximo corresponde ao de mínimo custo.

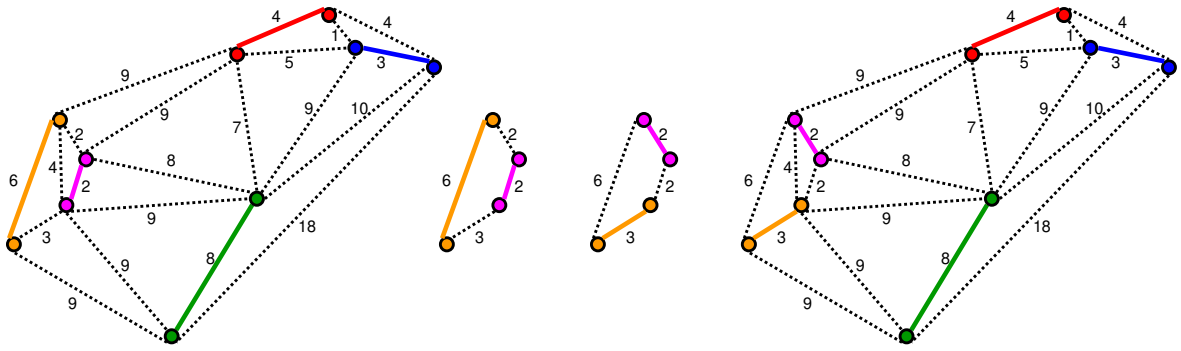


Figura 17. Sucessivamente, da esquerda para a direita: um emparelhamento máximo; um ciclo 2-alternante; uma operação 2-troca; um emparelhamento máximo de custo mínimo.

2.4. Determinação de um Circuito Euleriano em um Grafo Não-Direcionado

Dado um grafo *G* não-direcionado, onde todos os vértices possuem grau par, é possível mostrar a existência de um circuito (Euleriano) que, partindo de um nó inicial, passa por cada aresta do grafo uma única vez, retornando ao mesmo nó de partida (vide 1.5). Existem diversos algoritmos que tratam desse problema, alguns com complexidade computacional linear $O(|V|)$, cujas descrições podem ser encontradas em [14],[15]. Nesse trabalho foi utilizado um algoritmo atribuído a Edmonds e Johnson [14], denominado "next-node", o qual será descrito a seguir:

2.4.a. Algoritmo "next-node"

Passo 1. Seja r qualquer nó do grafo. Seja n_0 e n inicialmente iguais a r. Seja $k_i = 0$ para todos os nós i. Seja e qualquer aresta incidente ao nó n. Marque todas as arestas como não-utilizadas.

Passo 2. Seja m qualquer nó que não n, incidente à aresta e. Faça k_m mudar para k_{m+1} e para esse novo valor de k_m, faça $L_m(k_m) = n$. Marque essa aresta e como utilizada. Se o nó m possuir qualquer aresta não-utilizada incidente, vá para o passo 2. Caso contrário, o nó m deve ser igual ao nó n_0, então vá para o passo 3.

Passo 3. Faça n mudar para m e seja e uma aresta não-utilizada incidente a m. Vá para o passo 1.

Passo 4. Seja n_0 qualquer nó que possua ao menos uma aresta utilizada incidente e. Faça n mudar para n_0 e vá para o passo 1. Se nenhum nó n_0 existir, pare.

Esse algoritmo produz uma lista de nós $L_n(1)$, $L_n(2)$, ..., $L_n(k)$ que são visitados pelo circuito ao sair do nó n pela primeira até a k-ésima vez. Nesse algoritmo, a ordem de visitação das arestas deve ser no sentido contrário da visitação durante a execução algoritmo. Assim, para construir a rota Euleriana utilizando essa representação, a primeira vez que o nó n for visitado pelo circuito, ele deverá ser deixado para visitar o nó $L_n(k_n)$. Na próxima vez que o nó n for visitado, ele será deixado para visitar o nó $L_n(k_n)$. E assim sucessivamente, até a ultima visitação de n, que será deixado para visitar o nó $L_n(k_1)$.

3. Estruturas de Dados para Grafos

Existem diversas formas de representar um grafo, seja ele direcionado, não-direcionado, misto ou ainda uma rede. Quando se está tratando da aplicação de algoritmos para grafos, a escolha de uma boa estrutura de dados pode ser fundamental para um bom desempenho do algoritmo. A seguir serão apresentadas as estruturas de dados mais usuais para a representação computacional de um multigrafo $G(V,A,E)$ (vide Figura 18) com um conjunto de vértices $V = \{v_1,v_2,...,v_n\}$, um conjunto de arcos $A = \{a_1,a_2,...,a_p\}$, e um conjunto de arestas $E = \{e_1,e_2,...,e_q\}$.

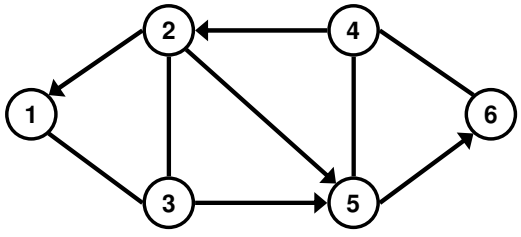


Figura 18. Um multigrafo $G(V,A,E)$, de seis nós, cinco arcos e quatro arestas.

3.1. Matriz de Adjacência

A matriz de adjacência $M = [m_{ij}]$ de G possui n x n elementos de modo que se o nó i for adjacente ao nó j, $m_{ij} = 1$, caso contrário $m_{ij} = 0$. A seguir será apresentada a definição formal dos elementos m_{ij} da matriz de adjacência.

$$m_{ij} = \begin{cases} 1 \text{ se } (v_i, v_j) \in A \\ 1 \text{ se } (v_i, v_j) \in E \text{ ou } (v_j, v_i) \in E \\ 0 \text{ caso contrário} \end{cases}$$

3.2. Lista de Adjacência

Na maioria dos casos, a matriz M consiste em uma matriz esparsa, onde grande parte de seus elementos são nulos, o que leva a um desperdício de memória. Nesse sentido, a lista de adjacência torna-se uma estrutura mais econômica, pois armazena somente os adjacentes de cada vértice. A lista de adjacência consiste em um vetor de tamanho n, cujos índices representam os vértices e o conteúdo de cada elemento do vetor aponta para uma lista contendo todos os adjacentes do vértice.

Tabela 1. Matriz e lista de adjacência para o multigrafo da Figura 18.

Matriz de Adjacência						
nó	*1*	*2*	*3*	*4*	*5*	*6*
1	0	0	1	0	0	0
2	1	0	1	0	1	0
3	1	1	0	0	1	0
4	0	1	0	0	1	1
5	0	0	0	1	0	1
6	0	0	0	1	0	0

Lista de Adjacência	
nó	*adjacentes*
1	{3}
2	{1,3,5}
3	{1,2,5}
4	{2,5,6}
5	{4,6}
6	{4}

3.3. Matriz de Incidência

A matriz de incidência $A = [a_{ij}]$ de G possui dimensão n x $(p+q)$, onde cada linha representa um nó e cada coluna representa um arco ou uma aresta. No intuito de simplificar a definição, será adotado que os arcos estarão posicionados nas colunas l para $1 \leq l \leq p$, e as arestas estão posicionadas nas colunas m tais que $p < m \leq p+q$. A seguir será apresentada a formalização dos elementos a_{ij} da matriz de incidência:

$$a_{ij} = \begin{cases} 1 \text{ se arco } a_j \text{ for incidente e estiver orientado para fora de } v_i \ (1 \le j \le p) \\ -1 \text{ se arco } a_j \text{ for incidente e estiver orientado para dentro de } v_i \ (1 \le j \le p) \\ 1 \text{ se aresta } e_j \text{ for incidente a } v_i \ (p+1 \le j \le p+q) \\ 0 \text{ caso contrário} \end{cases}$$

3.4. Lista de Incidência

A lista de incidência consiste em um vetor onde o índice de um elemento representa um vértice, e o conteúdo do elemento aponta para uma lista contendo todas as arestas e arcos incidentes a esse vértice.

Tabela 2. Matriz e lista de incidência para o multigrafo da Figura 18.

Matriz de Incidência									
	arcos					*arestas*			
nó	*(2,1)*	*(2,5)*	*(3,5)*	*(4,2)*	*(5,6)*	*(1,3)*	*(2,3)*	*(4,5)*	*(4,6)*
1	-1	0	0	0	0	1	0	0	0
2	1	1	0	-1	0	0	1	0	0
3	0	0	1	0	0	1	1	0	0
4	0	0	0	1	0	0	0	1	1
5	0	-1	-1	0	1	0	0	1	0
6	0	0	0	0	-1	0	0	0	1

Lista de Incidência		
nó	*arcos*	*arestas*
1	-	{(1,3)}
2	{(2,1),(2,5)}	{(2,3)}
3	{(3,5)}	{(3,1),(3,2)}
4	{(4,2)}	{(4,5),(4,6)}
5	{(5,6)}	{(5,4)}
6	-	{(4,6)}

4. Problemas de Roteamento

Os problemas de roteamento, que se inserem em uma grande família de problemas de otimização combinatória, podem ser subdivididos em duas grandes vertentes: roteamento nos nós e roteamento nos arcos (Figura 19). O interesse prático desses problemas é muito amplo, pois eles podem ser aplicados em contextos envolvendo entregas bancárias, coleta de lixo, serviços de disque-entrega, roteamento de ônibus escolares, inspeção de redes elétricas e gasodutos, entrega de correio e roteamento de leituristas, que é o objeto de estudo deste trabalho, dentre muitas outras aplicações. Bilhões de dólares são gastos todo ano nessas operações e o potencial de evitar desperdícios e economizar recursos através da otimização é gigantesco [15][16][45].

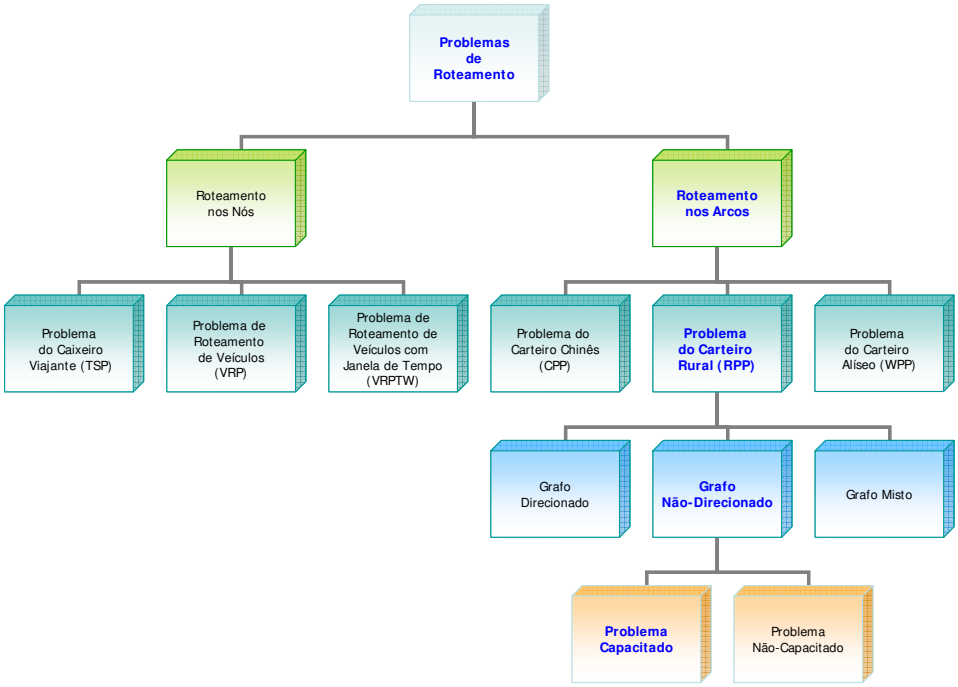


Figura 19. Árvore de problemas de roteamento.

4.1. Problema de Roteamento nos Nós (NRP)

Apesar de não se tratar do objeto de estudo deste trabalho, a seguir serão comentados três problemas clássicos de roteamento nos nós, consistindo nos problemas mais estudados de roteamento da literatura atual.

Lawler et al. [32] descreve o problema do caixeiro viajante (TSP) como a busca de um ciclo Hamiltoniano (vide seção 1.6) de mínimo custo de uma rede *G(V,E)* (vide seção 1.2.d). Em outras palavras, deseja-se encontrar um ciclo que visite todos os nós do grafo uma única vez e cuja soma dos custos das arestas desse ciclo seja a menor possível. De acordo com Sang-Ho, Young-Gun e Kyu-Maing [46], esse problema pode ser dividido em dois tipos: se o custo é simétrico para todos os nós *i* e *j* ($c_{ij} = c_{ji}$), tem-se então o problema do caixeiro viajante simétrico (STSP), caso contrário, o problema será assimétrico (ATSP). O TSP foi o primeiro problema de otimização que se comprovou pertencer à classe NP-Completo e portanto muitos métodos heurísticos foram pesquisados para resolvê-lo aproximadamente.

Solomon [45] descreve dois problemas derivados do TSP: O problema de roteamento de veículos (VRP) e o problema de roteamento de veículos com janela de

tempo (VRPTW). O VRP consiste na elaboração de um conjunto de rotas de mínimo custo, saindo de um nó central (depósito) e retornando ao mesmo nó, para uma frota de veículos com capacidade definida, servindo a um conjunto de clientes (nós) com demandas previamente conhecidas. Cada cliente é servido somente uma vez e todos os clientes devem ser submetidos a um veículo sem que esse exceda sua capacidade.

Por sua vez o VRPTW consiste em um VRP acrescentado de uma restrição: a janela de tempo, ou período permitido de serviço. Essa restrição representa o período imposto por cada cliente, caracterizado por um tempo inicial (o mais cedo) e um tempo final (o mais tarde) para ser atendido. Com a presença de janelas de tempo, os custos de roteamento incluem além da distância e do tempo, o custo de espera devido à chegada antecipada do veículo no local de um cliente. O VRPTW surgiu como uma importante área que permitiu o progresso da resolução das complicações e generalizações do modelo de roteamento básico. As janelas de tempo surgem naturalmente nos problemas enfrentados pelas organizações que trabalham baseadas em cronogramas [45].

Na Figura 20 são apresentadas duas soluções típicas de problemas de caixeiro viajante e roteamento de veículos (com ou sem janela de tempo). Pode-se observar que a solução do TSP consiste em uma única rota, pois o problema é considerado não-capacitado, o que não ocorre no VRP, onde se observam dez rotas de trabalho, certamente criadas para atender às restrições de demanda (e de tempo, caso o problema possua janelas).

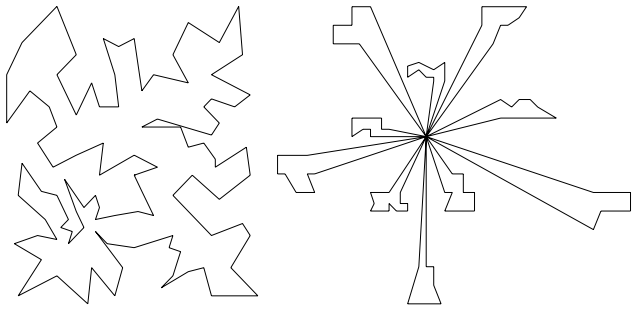


Figura 20. Soluções típicas de um problema TSP, à esquerda, e VRP(TW), à direita.

4.2. Problema de Roteamento nos Arcos (ARP)

Eiselt, Gendreau e Laporte [15] explicam que, dado um subconjunto de arestas de um grafo, o objetivo do roteamento em arcos (ARP) consiste na determinação da travessia de custo mínimo desse subconjunto, com ou sem restrições.

Três importantes problemas pertencentes à classe ARP podem ser destacados: o Problema do Carteiro Chinês ("Chinese Postman Problem" – CPP), o Problema do Carteiro Rural ("Rural Postman Problem" – RPP) e o Problema do Carteiro Alíseo ("Windy Postman Problem" - WPP).

De acordo com Eiselt, Gendreau e Laporte [16], a maioria dos problemas práticos não podem ser modelados como um problema ARP puro (CPP, RPP ou WPP) devido às características adicionais peculiares a cada caso, que devem ser incorporadas ao modelo de otimização. Na presença de tais problemas de ordem prática, os autores comentam sobre a decisão do grafo ser direcionado, não-direcionado ou misto, o que dependerá da topologia da malha viária e das questões operacionais envolvidas. No entanto, definem uma regra geral: pode-se definir uma rua de sentido único como um arco e uma rua de duplo sentido como uma aresta. Nos casos onde os dois lados da rua devem ser servidos separadamente, torna-se necessária a substituição da aresta por dois arcos de sentidos opostos.

4.3. Problema do Carteiro Chinês

O Problema do Carteiro Chinês (CPP) consiste em encontrar uma rota de mínimo custo sobre um grafo $G(V,E)$, que passe pelo menos uma vez em cada aresta $e \in E$. O CPP deve seu nome por ter sido introduzido pelo matemático chinês Mei-Ko [33] e por estar associado ao problema dos carteiros, que entregam as correspondências ao longo de cada aresta (rua) de um grafo, iniciando e terminando sua rota em um mesmo nó.

De acordo com Eiselt, Gendreau e Laporte [15], o algoritmo para o CPP pode ser dividido em duas fases distintas. A primeira consiste em determinar o aumento de mínimo custo do grafo, ou seja, um conjunto de arcos e arestas que quando duplicados tornam o grafo Euleriano (vide seção 1.5). A segunda fase prevê a construção de uma travessia no grafo aumentado, que pode ser executado em tempo polinomial.

Quando G é completamente direcionado ou não-direcionado, o aumento do grafo também é resolvido em tempo polinomial. De acordo com Edmonds e Johnson [14], o

aumento de um grafo não-direcionado, tornando-o Euleriano, consiste em resolver um problema de emparelhamento máximo de mínimo custo. Já para um grafo direcionado, o aumento de custo mínimo requer a resolução de um problema de fluxo em rede. Outros problemas derivados do CPP, no entanto não são tão fáceis. Por exemplo, quando *G* é um grafo misto, o problema de aumento de mínimo custo do grafo torna-se, de acordo com a demonstração feita por Papadimitriou [36], NP-completo.

4.3.a. Não-direcionado

Se existe um circuito Euleriano em um grafo *G(V,E)*, então ele resolve o CPP. Edmonds e Johnson [14] revelam que, no caso de grafos não-Eulerianos, para resolver o CPP, há necessidade de modificar o grafo de algum modo no intuito de torná-lo Euleriano. A idéia de Edmonds e Johnson foi criar uma ligação entre cada par de nós de grau ímpar do grafo. Esses autores perceberam que era possível determinar tais ligações com custo mínimo, resolvendo o problema de emparelhamento máximo de mínimo custo nesses nós. Feito isso, bastava inserir as arestas do emparelhamento no grafo para torná-lo Euleriano. A partir do grafo Euleriano, encontrar um circuito que inicia e termina no mesmo nó, passando por todas as arestas, passa a ser uma trivial (vide seção 2.4).

4.3.b. Direcionado

Eiselt, Gendreau e Laporte [15] mostram que um aumento de mínimo custo de um grafo direcionado, para que ele se torne Euleriano, é possível a partir da solução de um problema de fluxo em rede. No entanto, para que uma solução exista, é necessário que o grafo seja fortemente conexo, ou seja, deve haver um caminho direcionado entre cada par de nós, caso contrário, não há garantia de que esse método funcione. Construir um circuito Euleriano em um grafo Euleriano direcionado requer, de acordo com os autores, pequenas adaptações nos algoritmos para grafos não-direcionados (vide seção 2.4).

4.3.c. Misto

Pearn e Liu [40] consideram o problema CPP misto uma generalização interessante do clássico CPP, no qual uma rede é mista. Em uma rede mista, as arestas podem ser atravessadas nas duas direções enquanto os arcos só podem ser percorridos em um único sentido. Tal generalização reflete claramente situações práticas, como as ruas de uma

cidade, por exemplo, que só podem ser percorridas em um sentido, ora por serem estreitas ora pela simples necessidade de um maior controle do fluxo de tráfego. Pearn e Liu definem o CPP misto da seguinte maneira: dado um grafo misto $G = (V,A,E)$, cada aresta $(i,j) \in$ E possui duas distâncias (pesos) não negativas: d_{ij} para o sentido de i para j e d_{ji} para o sentido de j para i, com $d_{ji} = d_{ij}$. Por sua vez, um arco $(i,j) \in A$ possui uma distância não negativa d_{ij}, mas d_{ji} é definido como infinito. O CPP misto consiste em encontrar uma rota que, saindo de um nó inicial, atravesse todas as arestas e arcos $E \cup A$ ao menos uma vez, e retorne ao mesmo nó inicial com distância total mínima. Claramente, quando $A = \varnothing$ tem-se a redução do CPP misto para o CPP não-direcionado.

Como mencionado anteriormente, um grafo misto é Euleriano se e somente se ele for balanceado (1.3.a). Eiselt, Gendreau e Laporte [15] comentam que resolver um CPP em um G misto nem sempre é possível porque alguns grafos não podem ser feitos Eulerianos pela duplicação de arcos e arestas. Os algoritmos que resolvem o CPP misto normalmente se dividem em três fases: 1) direcionar algumas arestas de modo que o grafo se torne simétrico. 2) direcionar as arestas restantes de modo que o grau de cada vértice seja par. 3) Determinar uma travessia para G.

Frederickson [19] explica que o algoritmo de Edmonds e Johnson [14] para o CPP misto, denominado MIXED1, divide-se em duas fases. A primeira fase modifica o grafo, tornando o grau de cada vértice em par, pela transformação dos arcos em arestas. A segunda fase faz com que o grau de entrada de cada vértice seja igual ao grau de saída, preservando o grau par de cada vértice. Após a aplicação das duas fases, o grafo torna-se balanceado (vide seção 1.3.a), condição suficiente para existência de um circuito Euleriano. Frederickson ainda provou que o fator de eficiência desse algoritmo é $\delta = 2$, ou seja, a solução é no máximo duas vezes maior do que a solução ótima.

Um segundo algoritmo (MIXED2) foi proposto por Frederickson [19], cuja estratégia se baseia nas duas fases do MIXED1, porém de maneira inversa. Primeiramente os graus de saída e entrada de cada nó são igualados. Em seguida, o grau de cada nó é tornado par, preservando a igualdade dos graus de entrada e saída. Frederickson provou que esse algoritmo também possui um fator de eficiência de $\delta = 2$ (vide definição em 2.3).

Pela observação de algumas instâncias rodadas nos algoritmos MIXED1 e MIXED2, Frederickson [19] descobriu que o algoritmo MIXED1 produzia boas soluções

em instâncias cujo desempenho de MIXED2 não se mostrava tão eficiente. Por sua vez, também havia instâncias nas quais o algoritmo MIXED2 tratava melhor do que o MIXED1. Esse comportamento sugeria uma certa complementariedade entre os algoritmos, o que levou o autor a considerar uma combinação dos dois algoritmos, denominada MIXED12. Esse novo algoritmo executa o MIXED1 e MIXED2 separadamente, selecionando-se a melhor solução. Frederickson provou que tal estratégia atinge um fator de eficiência $\delta = \frac{5}{3}$, superando os fatores obtidos no emprego isolado de MIXED1 e MIXED2.

Pearn e Liu [40] sugerem uma modificação nos algoritmos MIXED1 e MIXED2. Com relação ao algoritmo MIXED1, os autores perceberam que certas vezes o algoritmo formava ciclos de arcos artificiais, cuja remoção poderia elevar a qualidade das soluções. Desse modo, sugeriram a inserção de uma fase ao MIXED1 que removia esses ciclos artificiais, sem alterar sua complexidade computacional. Quanto ao MIXED2, os autores perceberam que uma das etapas desse algoritmo consistia na resolução de um emparelhamento máximo de custo mínimo, aplicado a um subgrafo contendo somente arestas. Eles afirmam que se as distâncias dessas arestas forem relativamente grandes, MIXED2 pode não resultar em soluções de boa qualidade. Desse modo propõem uma nova abordagem envolvendo primeiramente a duplicação de alguns arcos e assinalando direções às arestas, no intuito de converter a rede em Euleriana. Essas modificações, propostas por Pearn e Liu, apesar de não melhorarem o fator de eficiência, resultaram em melhores soluções na prática. Além disso, o aumento do tempo computacional foi pouco significativo. Pearn e Chou [39] propuseram ainda outras modificações nos algoritmos MIXED1 e MIXED2, que também resultaram em melhores soluções, porém com a mesma complexidade computacional e fator de eficiência.

Raghavachari e Veerasamy [44] conseguiram formular um algoritmo denominado Modified MIXED12, também baseado no MIXED12 de Frederickson [19], resultando na melhoria do fator de eficiência para $\delta = \frac{3}{2}$. Para isso os autores propuseram uma modificação no algoritmo MIXED1, adicionando-lhe uma nova etapa.

Corberan, Marti e Sanchis [10], propuseram uma abordagem metaheurística baseada em GRASP ("Greedy Randomized Adaptive Search Procedures") para tratar o CPP misto. De acordo com os autores, as metaheurísticas têm tido sucesso na resolução de difíceis

problemas de otimização, por vezes encontrando soluções ótimas ou próximas da ótima. Em contrapartida, os autores comentam que o desenvolvimento de uma boa metaheurística continua sendo uma arte porque depende da habilidade e experiência do desenvolvedor, tanto nas técnicas de resolução como no conhecimento específico do problema em consideração.

O algoritmo GRASP que Corberan, Marti e Sanchis [10] propuseram é composto de duas fases: uma fase construtiva e uma fase de busca local. A fase construtiva consiste em obter uma solução inicial para o CPP misto, definindo-se uma orientação para cada aresta do grafo, tornando-o totalmente direcionado e passível de resolução por algoritmo polinomial (vide seção 4.3.b). A partir dessa solução inicial entra a segunda fase do algoritmo, onde ocorre uma busca na vizinhança da solução definida por uma operação que os autores denominaram translação ("move"), cujos operandos são dois nós. A fase de busca local procura por dois nós que estão ligados por um caminho (direcionado ou não) de arestas duplicadas. A operação translação então remove esse caminho e insere uma aresta com peso igual ao valor do caminho mínimo (vide seção 2.2) entre os dois nós. Os resultados comparativos foram feitos entre o GRASP, MIXED12 [19] e o Modified MIXED12 [44]. De acordo com Corberan, Marti e Sanchis o algoritmo baseado em GRASP, apesar de perder em termos de tempo computacional, apresentou resultados significativamente melhores.

4.4. Problema do Carteiro Rural

Enquanto no CPP deseja-se encontrar uma rota que passe por todas as arestas do grafo *G(V,E)*, existem situações onde somente algumas arestas $e \in R$ ($R \subseteq E$) requerem serviço, o que define o Problema do Carteiro Rural (RPP). De fato, Eiselt, Gendreau e Laporte [16] comentam que são poucas as situações reais onde todas as arestas (e arcos) exigem travessia. Eles explicam que o termo carteiro rural surgiu devido à associação desse problema com a entrega de correio em regiões rurais, onde há um número de vilas cujas ruas devem ser servidas pelo carteiro, enquanto as demais ruas servem apenas como elos, para transitar entre as vilas, não demandando qualquer serviço.

Eiselt, Gendreau e Laporte [16] comentam que o RPP para grafos direcionados e não-direcionados é comprovadamente NP-Completo. Um dos problemas mais difíceis de

roteamento em arcos consiste no RPP misto, também denominado "Stacker Crane Problem" (SCP). Os autores observam que, assim como no CPP, a estratégia algorítmica para tratar do RPP consiste em determinar um aumento de mínimo custo do grafo de modo a torná-lo Euleriano, para só assim descobrir um circuito Euleriano no grafo aumentado. Pelo fato do RPP ser NP-Completo, a fase de aumento do grafo normalmente é feita aplicando-se heurísticas que empregam emparelhamento de nós ou árvores geradoras.

4.4.a. Não-direcionado

Como foi comentado, para resolver um RPP em um grafo $G(V,E)$ onde $R = E$, basta aplicar o algoritmo CPP. Além disso, se o grafo reduzido $G'(V,R)$, contendo somente as arestas requeridas, for conexo, também nesse caso basta aplicar o algoritmo CPP sobre G'. O algoritmo mais conhecido para tratar do RPP não-direcionado é atribuído a Frederickson [19] e será explicado a seguir:

Passo 1. (Árvore Geradora Mínima) Construa uma árvore geradora mínima $T(V,E_T)$ sobre o grafo $G(V,E)$, onde E pode ser dividido em dois subconjuntos: das arestas requeridas (R) e não-requeridas ($E \setminus R$).

Passo 2. (Emparelhamento Máximo de Mínimo Custo) Seja G' o grafo induzido pela união das arestas R e E_T ($G' = (V, R \cup E_T)$). Verifique os nós de grau ímpar de G' e induza um grafo completo K_n a partir desses nós. Faça com que o peso de cada aresta de K_n seja igual à distância do caminho mínimo de seus nós incidentes. Faça o emparelhamento máximo de mínimo custo M sobre o grafo completo K_n.

Passo 3. (Circuito Euleriano) Seja G'' o grafo induzido pela união de G' e M ($G'' = (G' \cup M)$). Pode-se demonstrar que todos os nós de G'' possuem grau par, ou seja, G'' é Euleriano. Portanto, uma solução aproximada do RPP consiste no circuito Euleriano de G'', que pode ser determinado pelo algoritmo "next-node", por exemplo (vide seção 2.4.a).

4.4.b. Direcionado

O RPP direcionado é definido sobre um grafo direcionado $G(V,A)$. Também nesse caso, o problema se reduz ao CPP direcionado quando $G'(V,R)$ é conexo. A formulação matemática desse problema pode ser encontrada em [16]. Um algoritmo ótimo baseado em uma estratégia "branch and bound" foi proposto por Christofides et al. [7], que também elaboraram uma heurística que, testada em 23 instâncias, atingiu a solução ótima em dez, sendo o desvio médio igual a 1,3 %.

4.4.c. Misto

O RPP misto, definido sobre um grafo misto *G(V,A,E)*, é também denominado "Problema da Esteira" ("Stacker Crane Problem" – SCP) devido à associação dos arcos com o movimento de esteiras mecânicas em um sentido definido [16]. Trata-se de um dos problemas mais difíceis de roteamento em arcos e ainda não possui um algoritmo que o trate até a otimalidade [16], no entanto duas heurísticas chamadas LARGEARCS e SMALLARCS foram elaboradas por Frederickson, Hecht e Kim [20]. De acordo com os autores, a heurística LARGEARCS funciona melhor quando o custo total dos arcos é bem maior do que o custo total da rota ótima, e vice-versa. Para maiores informações sobre o RPP misto, sugere-se consultar as referências [9] e [11], onde são apresentadas outras heurísticas e um estudo da formulação do problema.

4.5. Problema do Carteiro Alíseo

Minieka [35] percebeu que, em situações práticas, é pouco provável que o custo de atravessar determinada aresta seja igual nos dois sentidos, pois um dos sentidos pode estar ladeira abaixo enquanto o outro ladeira acima, assim como um sentido pode estar a favor do vento e o outro contra. Desse modo, Minieka percebeu a necessidade de criar um novo problema, por ele denominado Problema do Carteiro Alíseo ("Windy Postman Problem" - WPP), onde o custo de travessia de uma aresta depende do sentido tomado.

Apesar do WPP ser NP-Completo, existem condições suficientes que tornam possível sua resolução em tempo polinomial. Uma condição diz que as duas orientações de todos os ciclos de *G* devem possuir o mesmo custo. A outra condição é *G* ser Euleriano.

É fácil observar que um problema CPP misto pode ser transformado em um WPP e vice-versa. Para isso, dado uma aresta do CPP misto, essa será uma aresta do WPP com custos de travessia iguais. No caso de um arco do CPP misto, esse será transformado em uma aresta com custo igual ao arco para seu sentido e custo infinito no sentido contrário. A semelhança entre esses dois problemas torna seus métodos de resolução também semelhantes e de acordo com Eiselt [15], as heurísticas que tratam desses dois problemas procuram encontrar um aumento de mínimo custo do grafo de modo a torná-lo Euleriano. Além disso, os algoritmos que resolvem esses problemas até a otimalidade utilizam formulação de programação linear inteira e métodos de "branch-and-cut".

Uma generalização do WPP consiste no Problema do Carteiro Rural Alíseo ("Windy Rural Postman Problem" – WRPP), um problema muito pouco estudado pela literatura que, porém, é de grande interesse devido a ser uma generalização de todos os problemas não-capacitados de roteamento em arcos (CPP, RPP e WPP). Benavent et al. [3] descrevem o WRPP do seguinte modo. Seja *G(V,E)* um grafo não-direcionado e conexo com dois custo não-negativos, c_{ij} e c_{ji}, associados a cada aresta *(i,j)* $\in E$, representando o custo de atravessar a aresta do nó *i* para o nó *j* e do nó *j* para o nó *i*, respectivamente. Seja *R* um subconjunto de arestas de *E*, denominado conjunto das arestas requeridas. Então, o WRPP consiste no problema de encontrar a rota de mínimo custo que atravesse cada aresta de *R* pelo menos uma vez. Esses autores descrevem quatro heurísticas construtivas, técnicas de melhoria da solução e uma metaheurística baseada em "Scatter Search". Mais três heurísticas construtivas, bem como uma estratégia de planos de corte ("cutting plane"), são propostos por Benavent et al. [2].

4.6. Problema de Roteamento em Arcos Capacitado

O problema de roteamento em arcos capacitado ("Capacitated Arc Routing Problem" - CARP) foi introduzido por Golden e Wong [24] que o definiram da seguinte forma: Dado uma rede não-direcionada *G(V,E)*, com uma função de custo c_{ij} e uma função de demanda q_{ij} ($c,q: E \rightarrow \mathbb{R}^+$) com domínio nas arestas e imagem nos reais não-negativos, cada aresta *(i,j)* deve ser servida por um único veículo com capacidade *W* ($W \geq \max(q_{ij})$) de uma frota. Torna-se importante distinguir os processos de servir e percorrer uma aresta, pois uma aresta pode ser percorrida sem ser servida, no entanto toda aresta servida é necessariamente percorrida. Devem-se então encontrar as rotas de cada veículo, passando por um nó central (depósito) e atendendo a demanda das arestas a um custo mínimo, sem ultrapassar a capacidade de cada veículo. Pearn [37] acrescenta que as rotas dos veículos devem ser construídas de modo que:

- Cada aresta com demanda positiva deve ser servida exatamente por um único veículo;
- Cada rota deve começar e terminar em um nó específico denominado depósito;
- A demanda total das arestas servidas por cada veículo não deve ultrapassar a capacidade *W* do veículo

- O custo total de todas as rotas deve ser minimizado.

Eiselt, Gendreau e Laporte [16] comentam que, pelo fato da maioria das aplicações práticas conterem restrições de capacidade, o CARP é provavelmente o problema mais importante na área de roteamento em arcos.

De acordo com Golden e Wong [24], quando existe demanda em todas as arestas do grafo ($q_{ij} > 0$), define-se então o Problema do Carteiro Chinês Capacitado (CCPP). Já quando algumas arestas *(i,j)* exigem serviço (*(i,j)* $\in R \subseteq E$) e outras não, tem-se o Problema do Carteiro Rural Capacitado (CRPP). Os autores examinam que o CARP também generaliza os problemas não-capacitados CPP e RPP vistos anteriormente. Para isso, basta supor que a frota dos problemas CCPP e CRPP consiste de apenas um único veículo com capacidade $W = \sum_i \sum_j q_{ij}$.

Golden e Wong [24] apresentam uma formulação matemática do CARP:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{p=1}^{K} c_{ij} x_{ij}^{p} \tag{1}$$

s.a.

$$\sum_{k=1}^{n} x_{ki}^{p} - \sum_{k=1}^{n} x_{ik}^{p} = 0 \qquad (i = 1,...,n);(p = 1,...,K) \tag{2}$$

$$\sum_{p=1}^{K} \left(l_{ij}^{p} + l_{ji}^{p} \right) = \left\lceil \frac{q_{ij}}{W} \right\rceil \qquad (i,j) \in E \tag{3}$$

$$x_{ki}^{p} \geq l_{ij}^{p} \qquad (i,j) \in E;(p = 1,...,K) \tag{4}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} l_{ij}^{p} q_{ij} \leq W \qquad (p = 1,...,K) \tag{5}$$

$$\begin{cases} \sum_{k=1}^{n} f_{ik}^{p} - \sum_{k=1}^{n} f_{ki}^{p} = \sum_{j=1}^{n} l_{ij}^{p} & (i = 2,...,n);(p = 1,...,K) \\ f_{ij}^{p} \leq n^2 x_{ij}^{p} & (i,j) \in E;(p = 1,...,K) \\ f_{ij}^{p} \geq 0 & (i,j) \in E;(p = 1,...,K) \end{cases} \tag{6}$$

$$x_{ij}^{p}, l_{ij}^{p} \in \{0,1\} \qquad (i = 1,...,n);(j = 1,...,n);(p = 1,...,K) \tag{7}$$

onde:

n = número de nós;

K = número de veículos;

q_{ij} = demanda da aresta *(i,j)*;

W = capacidade do veículo;

c_{ij} = custo da aresta *(i,j)*;

x_{ij}^{p} = 1 se aresta *(i,j)* for atravessada pelo veículo p, 0 caso contrário;

l_{ij}^{p} = 1 se aresta *(i,j)* for servida pelo veículo p, 0 caso contrário;

$\lceil z \rceil$ = teto de z: o menor inteiro maior ou igual a $z \in \mathbb{R}$.

A função objetivo (Equação 1) procura minimizar o custo total das rotas. A Equação 2 assegura a continuidade das rotas. A Equação 3 declara que cada aresta com demanda positiva deve ser servida exatamente uma vez. A Equação 4 garante que a aresta *(i,j)* deve ser servida pelo veículo p somente se ele atravessa o arco *(i,j)*. A Equação 5 impede ultrapassar a capacidade dos veículos. A Equação 6 garante a não formação de subrotas ilegais, ou seja, a formação de uma rota desconexa para um veículo. Para isso, a Equação 6 utiliza-se de variáveis de fluxo f_{ij}^{p} que podem assumir valores positivos somente quando x_{ij}^{p} = 1. Por fim, a Equação 7 consiste em uma restrição de integralidade.

Pode-se perceber que na formulação do problema não se procura minimizar o número de veículos usados, ou seja, o número de veículos não é uma variável de decisão, portanto supõe-se saber antecipadamente esse número. Normalmente a determinação desse número se dá dividindo-se a soma das demandas de todas as arestas pela capacidade do veículo, ou seja, $K = \dfrac{\sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}}{W}$, porém por se tratar de um limitante inferior, essa estratégia não garante que haja solução factível com esse número de veículos.

Golden e Wong [24] conseguiram demonstrar que não somente o CARP é NP-Completo, mas que 0,5-aproximado CARP, que consiste no problema de encontrar uma solução até 1,5 vezes maior que a solução ótima do CARP, também é. Nesse sentido, são propostas diversas heurísticas para tratar do CARP, cujas descrições estão bem elaboradas em [24], [37] e [38]. Greistorfer [26] propõe uma metaheurística híbrida (Tabu Scatter Search) para tratar do CARP. Seu algoritmo consiste de duas fases: a fase construtiva, onde uma solução inicial factível é obtida inicialmente tratando o problema como não capacitado

e em seguida formando ciclos que serão posteriormente unidos até formarem as rotas. Essa solução inicial serve de entrada para a segunda fase, que consiste em uma busca local guiada pelas soluções vizinhas da solução inicial. A vizinhança é definida por operadores de inserção e troca. Inicialmente há um período de intensificação, onde os operadores exploram a solução até não ser possível melhorá-la. Em seguida, ocorre a diversificação, que consiste na combinação de boas soluções (soluções de elite) obtidas durante o processo de intensificação. A diversificação produz novas soluções que são novamente exploradas por intensificação. Esse processo se repete até que um limitante inferior seja atingido ou quando o número de iterações máximo é ultrapassado.

5. Roteamento de Leituristas: Contribuições Anteriores

A seguir serão apresentados dois trabalhos onde foi aplicado o problema de roteamento em arcos em casos reais de roteamento de leituristas: o primeiro trabalho (Stern e Dror [47]) diz respeito ao roteamento de leituristas da companhia de energia elétrica da cidade de Beersheva, Israel; o segundo trabalho (Wunderlich et al. [52]) refere-se ao roteamento dos leituristas de uma companhia de gás de Los Angeles, Estados Unidos.

O algoritmo de Stern e Dror [47] trabalha com uma estratégia roteamento-agrupamento (“Route First, Cluster Second”) e trata o problema como não capacitado, a princípio, criando uma grande rota para todo o grafo. Em seguida são realizadas partições nessa rota, cada qual destinada a um leiturista. Por sua vez, o problema tratado por Wunderlich et al. [52] utiliza uma adaptação do algoritmo de “particionamento de arcos” desenvolvido por Bodin e Levy [5] e posteriormente aperfeiçoado pelos mesmos (Bodin e Levy [6]). Esse algoritmo de particionamento de arcos trabalha com uma estratégia inversa ao algoritmo de Stern e Dror, ou seja, uma estratégia agrupamento-roteamento (“Cluster First, Route Second”). Desse modo, primeiramente o grafo é particionado em regiões, cada qual destinada a um leiturista. Em seguida, são elaboradas as rotas para cada região.

Será feita uma explanação sobre os dois problemas práticos no tocante à motivação, o processo de formação do grafo, descrição geral do algoritmo e benefícios obtidos com o processo.

5.1. Caso 1: Leituristas de Beersheva, Israel

5.1.a. Motivação

A tarefa dos leiturista da companhia de energia elétrica de Beersheva, Israel, consiste em visitar periodicamente cada cliente, fazendo a leitura do seu consumo de energia elétrica. De acordo com Stern e Dror [47], o número de leituristas necessário para a tarefa pode ser reduzido otimizando-se o processo de geração das rotas de trabalho de cada leiturista, anteriormente feitas com base na intuição e experiência das pessoas envolvidas com as leituras.

O turno de trabalho máximo do leiturista é de cinco horas. Cada leiturista é transportado de automóvel da empresa até o endereço da primeira leitura a ser feita. Ao final do seu turno, ou quando não há mais leituras a serem feitas, o leiturista está livre para deixar a área e voltar de ônibus para sua casa.

5.1.b. Formação do Grafo

Stern e Dror [47] resolveram adotar um dos oito distritos da cidade de Beersheva como área de estudo. Pelo fato dos leituristas trabalharem a pé, eles podem percorrer as ruas em qualquer direção, portanto o grafo representativo da área de estudo é não-direcionado. Cada segmento de rua, entre duas interseções é representado por uma aresta. Os leituristas realizam a leitura em Z, ou seja, atravessam a rua de modo a realizar a leitura dos dois lados da rua simultaneamente. No entanto existem ruas de maior movimento que impedem o leiturista de atravessá-la com freqüência, logo são necessárias duas passagens por essa rua, uma de cada lado, em qualquer direção. Esse caso é representado por duas arestas paralelas não-direcionadas. As ruas que demandam leitura e que precisam ser servidas pelo leiturista serão as arestas requeridas ($R \subseteq E$). As ruas que não possuem consumidores não requerem leituras, portanto são as arestas não-requeridas ($E \setminus R$). Essas arestas podem no entanto ser utilizadas como atalho de uma região a outro do distrito pelo leiturista.

A cada aresta requerida $(i,j) \in R$ estão associados dois tempos: um tempo de leitura tl_{ij} e um tempo de percurso tp_{ij}. O tempo de leitura consiste no tempo necessário para o leiturista realizar o serviço naquela aresta. O tempo de percurso consiste no tempo do leiturista atravessar a aresta sem realizar qualquer serviço. É fácil perceber que o tempo de

leitura é sempre maior do que o tempo de percurso para uma aresta requerida. Já no caso das arestas não-requeridas $(i,j) \in E \setminus R$, só o tempo de percurso é positivo, pois o tempo de leitura é zero, não havendo leitura a ser feita. A estimativa dos tempos de leitura e percurso foi feita pela companhia elétrica de modo que esses tempos fossem devidamente ajustados para conter os intervalos de descanso e alimentação.

Podem ser citadas duas características importantes que diferenciam esse problema real do CARP:

- Não são necessárias trilhas fechadas (circuitos) (vide seção 1.3.c), porque a duração máxima da jornada de trabalho não inclui o tempo de saída e retorno do funcionário à empresa ou à sua casa.
- As trilhas podem iniciar e terminar em pontos intermediários de uma aresta.

5.1.c. Descrição do Algoritmo

Fase 1: Determinação de uma trilha Euleriana

Passo 1. Seja $G(V,E)$ o grafo original representativo do local de trabalho do leiturista, com um conjunto de arestas requeridas E. Remova todas as arestas não-requeridas de G, formando o grafo $G_1 = G - (E \setminus R)$.

Passo 2. Construir um grafo aumentado G^* a partir dos seguintes passos:

Passo 2.1. Se o grafo G_1 é Euleriano, faça $G^* = G_1$ e vá para o passo 2.6, caso contrário vá para o passo 2.2.

Passo 2.2. Identifique os nós de grau ímpar de G_1. Encontre os caminhos mínimos sobre o grafo original G, de todos os nós ímpares para todos os nós ímpares identificados. Crie o grafo completo G_2 contendo os nós ímpares identificados. Faça o peso de cada aresta de G_2 ser a distância do caminho mínimo de seus nós terminais.

Passo 2.3. Resolva o problema do emparelhamento máximo de mínimo custo sobre G_2, utilizando a heurística gulosa (vide seção 2.3).

Passo 2.4. Identifique o conjunto de arestas de G que representa o caminho mínimo entre os nós terminais de cada aresta emparelhada de G_2. Faça G^* ser a união dessas arestas com o grafo G_1.

Passo 2.5. Remova um dos caminhos mínimos de G_2 (uma sugestão consiste em retirar o maior) de modo que G^* contenha exatamente dois nós de grau ímpar, tornando-o semi-Euleriano.

Passo 2.6. Construa uma trilha Euleriana utilizando qualquer algoritmo de construção de rotas Eulerianas.

Fase 2: Particionamento da trilha Euleriana em jornadas de trabalho.

Passo 3. Coloque as arestas obtidas na Fase 1 em uma lista na ordem em que elas são visitadas.

Passo 4. Repita enquanto houver arestas na lista:

Passo 4.1. Repita até que a carga de trabalho W do leiturista seja excedida:

Passo 4.1.1. Remova uma aresta da lista e acumule seu tempo (de leitura ou de não-leitura, dependendo do caso).

Passo 4.2. Se a última aresta removida da lista foi uma aresta requerida, vá para o passo 4.3. Se for uma aresta não requerida, vá para o passo 4.4.

Passo 4.3. Anote a posição na aresta onde o tempo acumulado tenha sido exatamente igual a *W* e marque esse ponto como o final de uma jornada de trabalho. Inicie a próxima jornada de trabalho acumulando o tempo restante da aresta e volte para o passo 4.

Passo 4.4. Desfaça-se da aresta não-requerida e recorte a jornada de trabalho ao final da última aresta requerida que fora removida da lista.

Passo 4.5. Repita até que a próxima aresta da lista seja requerida ou até que a lista esteja vazia:

Passo 4.4.1. Descarte a próxima aresta da lista, que é não-requerida.

Passo 4.6. Retorne ao passo 4.

Passo 5. Retorne o número de jornadas de trabalho construídas e o tempo acumulado total de trabalho.

5.1.d. Benefícios

Stern e Dror [52] revelam que o número de rotas anteriormente necessárias para realizar a leitura da área de estudo eram 24, enquanto que o algoritmo que eles desenvolveram mostrou serem necessárias somente 15 rotas, o que corresponde a uma economia de aproximadamente 40 % da força de trabalho.

5.2. Caso 2: Leituristas de Los Angeles, Estados Unidos

5.2.a. Motivação

A SOCAL (Southern California Gas Company), iniciou em 1988 um estudo sobre os potenciais benefícios de um programa computacional de otimização para o roteamento de seus leituristas, motivada pelos seguintes motivos:

- O gasto anual da empresa chegava a 15 milhões de dólares.

- A empresa tinha um atraso de quatro anos na reestruturação de suas rotas, que já estavam muito sobrecarregadas.
- O desenvolvimento de novas rotas era uma tarefa árdua e lenta.

5.2.b. Área de Estudo

Para Wunderlich et al. [52], uma rota consiste no percurso que um leiturista consegue fazer ao longo da jornada de um dia de trabalho, que nesse caso corresponde a oito horas.

A empresa SOCAL testou e avaliou o software pra uma área de 2,5 % da área total de serviço. A área de estudo era constituída de aproximadamente 13.800 arestas.

5.2.c. Formação do Grafo

A formação do grafo ocorreu de modo muito semelhante ao Caso 1. A definição dos tempos de leitura tl_{ij} e percurso tp_{ij} de cada aresta *(i,j)*, por exemplo, são equivalentes nos dois casos. No entanto, há algumas diferenças no que diz respeito às arestas.

Wunderlich et al. [52] definem sua área de trabalho como uma rede conexa *G(V,E)* com um conjunto de arestas requeridas R e um conjunto de arestas não requeridas $E \setminus R$. Cada aresta $(i,j) \in R$ é não-direcionada, requer serviço e possui um tempo de leitura tl_{ij} e um tempo de percurso tp_{ij}. Já as arestas $(i,j) \in E \setminus R$ possuem somente tempo de percurso.

Devido à incompatibilidade da base de dados geográficos da SOCAL com outros dados geográficos, foi necessário traçar manualmente grande parte das arestas do grafo. Estimar o tempo de leitura foi uma tarefa difícil considerando que havia poucas informações no tocante a localização e distribuição dos consumidores. Para assegurar a conexidade do grafo da área de estudo, algumas arestas que não requeriam leitura foram consideradas como arestas requeridas.

Em toda área de estudo, havia sempre duas arestas incidentes aos mesmos nós, cada uma representando a leitura em um lado da rua. Essas duas arestas são denominadas pelo autor de “arestas contrapartidas” (“counterpart edges”), mas nesse trabalho elas serão denominadas “arestas paralelas”. Quando só um dos lados da rua exige leitura, cria-se uma aresta virtual com tempo de leitura zero e tempo de percurso igual à de sua aresta paralela. A presença de arestas paralelas entre dois nós adjacentes garante sempre um número par de arestas incidentes para qualquer nó, que consiste na condição necessária e suficiente para o

grafo ser Euleriano. Desse modo, o processo de encontrar um ciclo Euleriano nesse tipo de grafo torna-se bastante facilitado.

5.2.d. Descrição do Algoritmo

Como foi mencionado anteriormente, o algoritmo de particionamento de arcos de Bodin e Levy [6] procura dividir o grafo representativo da área de trabalho dos leituristas em um conjunto de subgrafos ou partições, onde cada partição será associada a um leiturista, que deverá cobrir as arestas da partição em uma jornada de trabalho. Antes de descrever o algoritmo, torna-se necessário explicar alguns conceitos.

Bodin e Levy [5] esclarecem que os leituristas possuem uma jornada de trabalho flexível, definida por uma carga horária mínima W_{INF} e máxima W_{SUP}, diferente do caso 1, onde se definia uma jornada de trabalho fixa. A duração W da jornada de trabalho de uma partição p é definida por Bodin e Levy [5] da seguinte forma:

$$W(p) = \sum_{(i,j)\in R} tl_{ij} + \sum_{(i,j)\in E\setminus R} tp_{ij}$$

Os autores também definem uma função de penalidade que engloba as N partições formadas, para o caso dos limitantes de alguma jornada de trabalho serem violados. Ela é definida da seguinte forma:

$$PEN = \alpha \times \sum_{p=1}^{N} \max\left[W(p) - W_{SUP}, 0\right] + \beta \times \sum_{p=1}^{N} \max\left[W_{INF} - W(p), 0\right]$$

Na equação acima α e β são parâmetros de ponderação do quão grave são as violações da jornada de trabalho mínima e máxima, respectivamente.

Quando a função de penalidade é positiva, isso significa que alguma partição possui uma duração da jornada de trabalho abaixo ou acima do desejado. Muitas vezes é possível reduzir ou até mesmo anular a penalidade pela simples realocação de um par de arestas paralelas entre partições vizinhas. Três operações dessa natureza são descritas por Bodin e Levy [5]: realocação de folha (“leaf swap”), realocação de ramo (“branch swap”) e realocação cíclica (“cycle swap”).

- A realocação de folha consiste em mover uma folha de uma partição para outra. Uma folha consiste em um par de arestas paralelas onde um de seus nós terminais não é adjacente à partição a que pertencem.

- A realocação de ramo, semelhante à de folha, consiste mover um ramo de uma partição para outra. Um ramo consiste em dois ou mais pares de arestas paralelas onde pelo menos um par é uma folha.
- A realocação cíclica consiste na realocação de um par de arestas paralelas que não seja uma folha, de uma partição para outra. Essa operação exige um maior cuidado porque pode tornar uma partição desconexa. Para que isso não ocorra, é necessário que o par de arestas paralelas a ser movido pertença a um ciclo.

A seguir será apresentado o algoritmo utilizado por Wunderlich et al. [52]:

Fase 1. Particionamento do grafo *G(V,E)*, onde cada partição conterá uma jornada de trabalho.

Passo 1. Estimar o número de partições (*NP*) a serem criadas, pegando-se o piso da divisão entre a soma dos tempos de leitura (tl_e) de todas as arestas $e \in E$ pela duração da jornada de trabalho do leiturista (*W*), ou seja, $NP = \left\lfloor \frac{\sum_{e \in E} tl_e}{W} \right\rfloor$.

Passo 2. Formar uma partição auxiliar que conterá o excesso da jornada de trabalho não coberta pelas partições *NP*. Assim, a duração da jornada de trabalho desejada para essa partição auxiliar (T_{AUX}) pode ser determinada como a diferença entre a soma dos tempos de leitura (tl_e) e a soma das durações das jornadas de trabalho das *NP* partições, ou seja, $T_{AUX} = \sum_{e \in E} tl_e - NP \times W$. Essa partição auxiliar é formada da seguinte maneira: escolhe-se aleatoriamente um dos nós fronteiriços do grafo. Em seguida, deve-se ir gerando o subgrafo a partir das arestas paralelas incidentes ao nó escolhido, até que a carga de leitura total dessa partição seja superior a T_{AUX}.

Passo 3. Particionar as arestas paralelas restantes de *G* em *NP* partições do seguinte modo:

Passo 3.1. Determinar *NP* nós sementes do grafo que irão compor cada partição. Primeiramente escolhe-se aleatoriamente o primeiro nó semente.

Passo 3.2. Repita até que haja *NP* nós sementes:

Passo 3.2.1. O próximo nó semente será aquele que maximizar a soma dos caminhos mínimos entre esse nó e os demais nós sementes.

Passo 3.3. Distribuição das arestas pelas partições. Repita até que todas as arestas tenham sido alocadas a alguma partição:

Passo 3.3.1. Verifique todas as arestas não alocadas incidentes a alguma partição, ou seja, que seja

incidente a qualquer nó pertencente a alguma partição. Dentre essas arestas, colete aquela de maior tempo de leitura, inserindo-a na menor das partições a que for incidente, ou seja, naquela cuja duração da jornada de trabalho for menor.

Passo 3.4. Balanceamento. Aplique as rotinas de realocação de folha, de ramo e cíclica enquanto for possível reduzir a penalidade.

Passo 4. Atualização dos nós sementes. Se após o balanceamento a função de penalidade resultar em um valor positivo, volte para o passo 3, caso contrário, pare.

Fase 2. Roteamento das partições.

Passo 5. Aplicar em cada partição de *G* (subgrafo Euleriano), um algoritmo que encontre um circuito Euleriano, por exemplo "next node" (vide seção 2.4).

5.2.e. Benefícios

A aplicação do algoritmo de roteamento em arcos resultou nos seguintes benefícios:

- Número de rotas de leituristas foi reduzido de 243 para 236;
- Rotas com durações mais próximas da jornada de trabalho de oito horas;
- Maior homogeneidade na duração das rotas;
- Tempo de percurso foi reduzido em 90 %.
- Benefício financeiro estimado em aproximadamente 870 mil dólares por ano.

III. MATERIAIS E MÉTODOS

Essa etapa do trabalho se dedica em mostrar o processo de desenvolvimento dos dois novos algoritmos de roteamento para leituristas: RFCS2 e CFRS2. Esses dois algoritmos foram baseados diretamente nos algoritmos de Stern e Dror [47] e Wünderlich et al. [52], que a partir daqui serão denominados RFCS1 (“Route First, Cluster Second 1”) e CFRS1 (“Cluster First, Route Second 1”).

Na seção 1 serão apresentadas as adaptações implementadas nos algoritmos RFCS1 e CFRS1 que deram origem aos novos algoritmos. A seção 2 revela a carga horária considerada para o leiturista e como a violação dessa carga horária penaliza a qualidade da solução. A seção 3 aborda o arquivo de entrada do algoritmo, mostrando as informações nele contidas. As seções 4 e 5 fazem as descrições dos algoritmos RFCS2 e CFRS2, com apresentação dos diagramas de blocos. A seção 6 exibe as instâncias utilizadas nos testes comparativos dos algoritmos, como elas estão organizadas e o modo como foram geradas. Finalmente a seção 7 introduz a abordagem utilizada na análise comparativa de desempenho dos algoritmos.

1. Adaptações dos Algoritmos de Roteamento

A criação de rotas de trabalho para leituristas consiste em um caso particular de problema de roteamento em arcos capacitado, ou mais especificamente, problema do carteiro rural capacitado, previamente abordado (vide II.4.6). Dois algoritmos existentes que tratam desse problema, denominados RFCS1 e CFRS1, foram abordados nas seções II.5.1 e II.5.2. No intuito de melhorar a qualidade das soluções desses dois algoritmos, alguns aspectos, que poderiam ser denotados como pontos fracos, foram aperfeiçoados, levando ao desenvolvimento de dois novos algoritmos, RFCS2 e CFRS2.

Considerando que os algoritmos originais RFCS1 e CFRS1 foram desenvolvidos no intuito de solucionar problemas reais de roteamento de leituristas, não é de se estranhar que os mesmos sofreram adaptações para abranger as condições particulares das condições locais nas quais estavam inseridos (vide Tabela 3).

Tabela 3. Particularidades dos algoritmos da literatura para roteamento de leituristas.

RFCS1	CFRS1
Rotas Abertas	Rotas Fechadas
Permitido parar no meio de uma aresta	Proibido parar no meio de uma aresta
Sem tolerância na carga horária do leiturista	Tolerâncias superior e inferior na carga horária do leiturista
Número de partições pós-determinado	Número de partições pré-determinado
Grafo torna-se Euleriano a partir de emparelhamento de nós	Grafo torna-se Euleriano a partir das arestas paralelas
Não há função de penalidade	Presença de função de penalidade
Não há rotina de melhoria da solução	Presença de rotina de melhoria da solução

Procurou-se estabelecer aos novos algoritmos, RFCS2 e CFRS2, uma aplicabilidade mais geral, mediante o tratamento dessas particularidades inerentes aos problemas para os quais RFCS1 e CFRS1 foram programados. Nesse sentido, foi adotado um conjunto de considerações válidas para RFCS2 e CFRS2, as quais estão enumeradas a seguir.

- As rotas são abertas, ou seja, os leituristas podem iniciar e terminar o serviço de leitura em nós distintos do grafo. Essa condição reflete a realidade pois os leituristas, de modo geral, não precisam necessariamente iniciar e terminar o serviço no mesmo ponto.
- Não é permitido ao leiturista iniciar ou terminar no meio de uma aresta. Essa condição não reflete a realidade, pois em muitos casos o leiturista pode terminar o serviço no meio de uma quadra, sobretudo em grandes avenidas. No entanto, essa restrição foi adotada tanto por simplificar os algoritmos, bem como por ser possível considerar essa questão na entrada de dados do algoritmo. Para isso, basta dividir uma aresta com alta demanda de serviço em duas ou mais arestas menores.
- São adotadas tolerâncias superiores e inferiores da carga horária. Essa relaxação torna o problema muito mais realista, pois é inviável a criação de rotas com exatidão nos tempos de serviço.
- Uma função de penalidade é utilizada nos dois algoritmos como um critério de qualidade da solução. Essa função indica o grau de violação das rotas, ou seja, o quanto elas ultrapassam as tolerâncias superior ou inferior da carga horária do leiturista. Nesse sentido, quanto menor o valor da função de penalidade, melhor será a solução associada.

2. Carga Horária e Função de Penalidade

Nesse trabalho, considera-se uma carga de trabalho com duração desejável de 5 horas, ou 300 minutos (W = 300), para o leiturista. Constatou-se que esse valor é utilizado na prática por algumas companhias brasileiras, no entanto, sua substituição não representa qualquer impedimento.Essa restrição de capacidade do leiturista não é apertada, possuindo uma tolerância superior e inferior de 15 minutos (W_{INF} = 285 e W_{SUP} = 315).

A função de penalidade, utilizada em CFRS1, passa a ser utilizada pelos dois algoritmos, RFCS2 e CFRS2, pois consiste em um indicador de qualidade da solução. Se a duração de todas as rotas de uma determinada solução estiver dentro das margens definidas pelas tolerâncias máxima e mínima (W_{INF} e W_{SUP}) de carga horária do leiturista, o valor da função é nulo. Caso contrário, o valor da função cresce com o aumento da violação das tolerâncias. Sejam *W(p)* a carga horária da partição *p*, *NP* o número de partições do grafo, α e β parâmetros de ponderação da gravidade da violação da carga superior com relação à inferior. A função de penalidade *PEN* utilizada nesse trabalho pode ser expressa como:

$$PEN = \sqrt{\alpha \times \sum_{p=1}^{NP} \max\left[W(p) - W_{SUP}, 0\right]^2 + \beta \times \sum_{p=1}^{NP} \max\left[W_{INF} - W(p), 0\right]^2}$$

Essa função, utilizada em RFCS2 e CFRS2, difere daquela utilizada em CFRS1 devido ao expoente quadrático da violação, que tende à elevação da penalidade de soluções cujas violações são muito heterogêneas (algumas rotas com grandes violações, outras com baixas violações), em detrimento de soluções com violações mais homogêneas, ainda que em termos absolutos as violações sejam iguais em ambos os casos.

3. Função Objetivo Hierárquica

Os algoritmos RFCS2 e CFRS2 utilizam a mesma função objetivo hierárquica para selecionar as melhores soluções. As hierarquias dessa função correspondem a:

- Minimizar função de penalidade (1ª hierarquia);
- Minimizar número de rotas (2ª hierarquia);
- Minimizar tempo de percurso (3ª hierarquia).

A solução com a menor função de penalidade será, desse modo, escolhida a melhor, ainda que outra solução possua, por exemplo, um número de rotas menor. A segunda hierarquia da função objetivo, minimizar o número de rotas, é utilizada somente quando

houver empate da primeira hierarquia (função de penalidade). Do mesmo modo, a terceira hierarquia é acionada mediante empate da primeira e segunda hierarquias.

A função de penalidade foi priorizada na função objetivo hierárquica pelo fato de que valores elevados de penalidade implicam no aumento do custo de uma solução, pois ora se está sub-aproveitando um leiturista, ora será necessário contratação de hora extra.

Justifica-se a presença do número de rotas na função objetivo também devido ao impacto provocado no custo da solução, considerando que o número de rotas equivale ao número de leituristas necessário para realizar o serviço.

Por sua vez, o tempo de percurso também se justifica na função objetivo, já que representa um tempo desperdiçado, onde o leiturista percorre determinada aresta sem realizar qualquer serviço.

4. Dados de Entrada

A base de dados de um problema de roteamento de leituristas envolve basicamente três informações: o grafo representativo da região de trabalho, os tempos de leitura e os tempos de percurso.

O grafo representativo da região de trabalho consiste em um grafo conexo cujos nós e arestas correspondem aos cruzamentos e segmentos de rua (respectivamente) da região onde há demanda por serviço de leitura. A formação do grafo, além das características geográficas locais, depende também do modo como o leiturista realiza seu serviço e para isso existem duas maneiras conhecidas:

- Leitura em Z: ocorre quando o serviço é realizado simultaneamente nos lados opostos de uma rua, a partir de travessias contínuas. Nesse caso, para cada segmento de rua basta uma aresta no grafo para representar o serviço de leitura.
- Leitura em U: ocorre quando o serviço é realizado em um lado da rua de cada vez. Nesse caso, os segmentos de rua devem ser representados por duas arestas, uma para cada face de quadra.

Estão associados a cada aresta do grafo dois tempos distintos: o tempo de leitura (*TL*) e o tempo de percurso (*TP*). Como o nome sugere, *TL* corresponde à duração média necessária para percorrer um segmento de rua realizando o serviço de leitura. Já *TP* corresponde à duração média para o leiturista percorrer determinado segmento de rua sem realizar qualquer leitura.

Existem segmentos de rua onde não há demanda por leitura. As arestas que representam esses segmentos são denominadas arestas não-requeridas e se identificam justamente pelo tempo de leitura nulo. Segundo essa terminologia, as demais arestas do grafo, que demandam serviço de leitura, são denominadas arestas requeridas e possuem tempos de leitura e percurso não-nulos.

5. Algoritmo RFCS2

A estratégia roteamento-agrupamento de Stern & Dror [47], denominado RFCS1 ("Route First, Cluster Second 1"), forma a base na qual se sustenta o novo algoritmo denominado RFCS2. O diagrama de blocos da Figura 21, contendo uma descrição geral de RFCS2, auxilia a compreensão de seu funcionamento. Nesse diagrama, alguns blocos estão destacados (preenchimento escuro) e representam as novas rotinas, inexistentes no algoritmo original (RFCS1).

5.1. Formação de G

A entrada do algoritmo RFCS2 consiste em um conjunto finito de arestas, dispostas em um arquivo texto na forma de uma lista de adjacências, ou seja, cada aresta é representada em uma linha do arquivo texto contendo nó inicial, nó final e os tempos de leitura e percurso. As arestas requeridas de um grafo possuem tempos de leitura e percurso não-nulos. As arestas não-requeridas, por sua vez, possuem somente tempo de percurso não-nulo.

Sendo *G* a estrutura de dados que armazena o grafo inicial do problema, para cada aresta requerida do arquivo de entrada, o algoritmo RFCS2 insere duas arestas em *G*: uma requerida, com peso igual ao tempo de leitura, e outra não-requerida, com peso igual ao tempo de percurso. Na ocorrência de uma aresta não-requerida no arquivo de entrada, somente uma aresta é inserida em *G*, também não-requerida e com peso correspondente ao tempo de percurso.

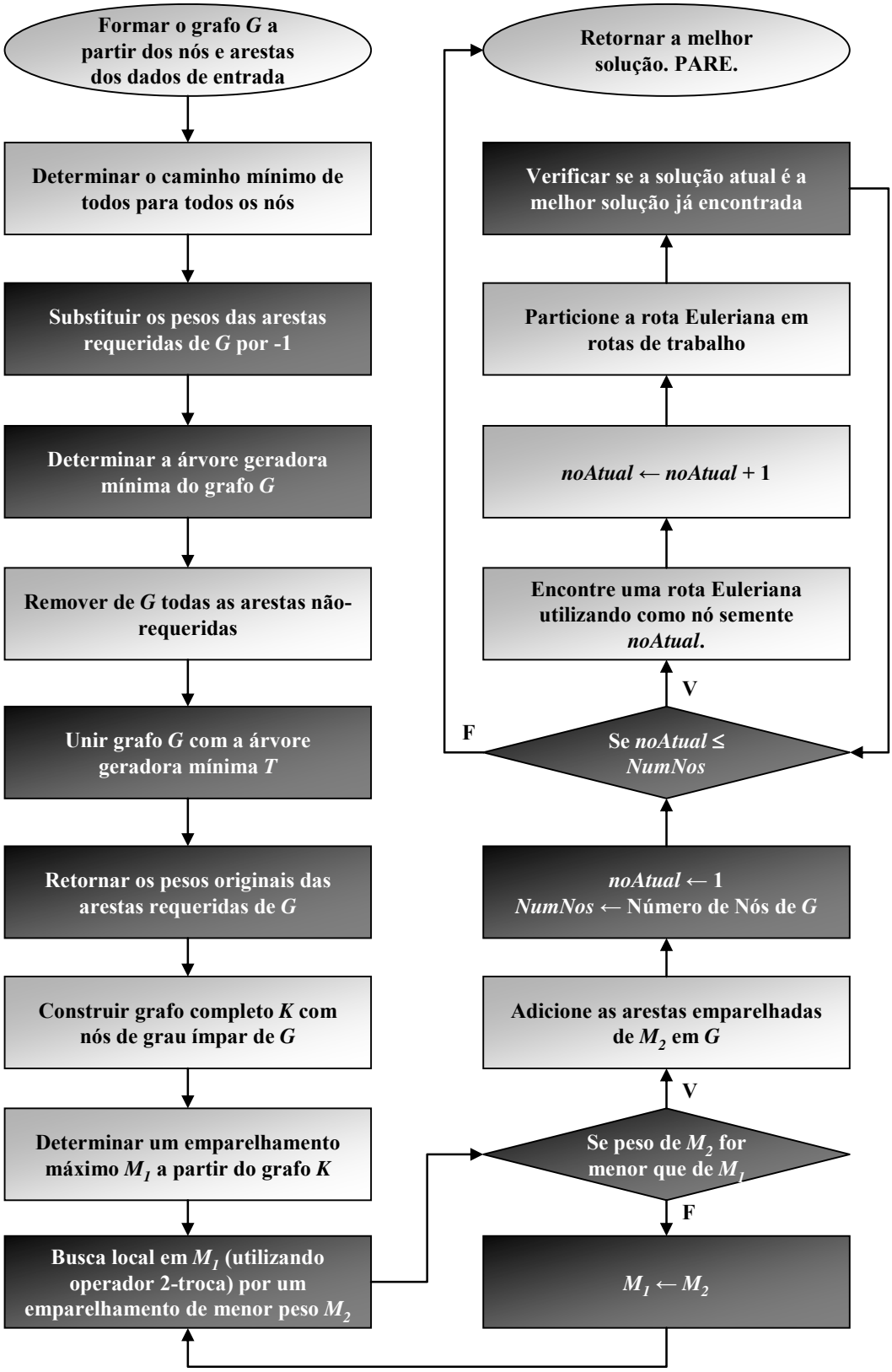


Figura 21. Diagrama de blocos do algoritmo RFCS2. Blocos destacados (preenchimento escuro) são inexistentes no algoritmo RFCS1.

5.2. Caminho Mínimo de Todos para Todos os Nós

Aplica-se no grafo *G* o algoritmo matricial de Hu [29], que obtém o caminho mínimo de todos para todos os nós do grafo. Nessa etapa, como pode existir mais de uma aresta entre dois nós adjacentes, considera-se somente aquela que possuir o menor peso.

5.3. Modificando o Peso das Arestas Requeridas

No intuito de priorizar a escolha das arestas requeridas na próxima etapa (árvore geradora mínima), atribui-se peso -1 a todas as arestas requeridas de *G*.

5.4. Árvore Geradora Mínima

Utiliza-se o algoritmo de Prim [43] para determinar a árvore geradora mínima *T* do grafo *G*, que consiste em um subgrafo conexo contendo todos os nós de *G* a um peso mínimo. Esse procedimento é fundamental quando a remoção das arestas não-requeridas de *G* torna-o desconexo. Nesse caso, o algoritmo procura um número suficiente de arestas não-requeridas para manter *G* conexo. Como os pesos das arestas requeridas estão negativos, essas recebem prioridade de seleção pelo algoritmo de Prim, que as coloca no maior número possível. Se a árvore *T* resultante possuir somente arestas requeridas, isso implica que *G* permaneceria conexo mesmo após a remoção de todas as arestas não-requeridas.

5.5. Retornando os Pesos Originais das Arestas Requeridas

Após a construção da árvore geradora mínima, os pesos das arestas requeridas de *G* voltam ao seu valor original.

5.6. Remoção das Arestas Não-Requeridas

Seja G_R o subgrafo resultante da remoção das arestas não-requeridas de *G*. Considerando que somente as arestas requeridas exigem serviço, o ideal seria construir as rotas para os leituristas utilizando somente essas arestas, no entanto, nem sempre isso é possível, ora por G_R ser desconexo, ora por não ser Euleriano. Os itens 5.7 a 5.11 a seguir são os procedimentos utilizados para aumentar minimamente G_R de modo a corrigir sua desconexidade e torná-lo Euleriano, quando necessário.

5.7. União de Grafos

Após a remoção das arestas não-requeridas de G, o grafo resultante G_R pode estar desconexo. A união do grafo G_R com a árvore geradora mínima T, acrescenta uma quantidade suficiente de arestas não-requeridas em G_R de modo que esse grafo se torne conexo. Como as arestas requeridas presentes em T já se encontram em G_R, a união G_U desses dois grafos consistirá no conjunto das arestas requeridas de G mais o conjunto das arestas não-requeridas de T.

5.8. Construção do Grafo Completo

Contando o número de arestas incidentes em cada nó de G_U, é possível determinar quais desses nós possuem grau ímpar, para finalmente utilizá-los na formação do grafo completo K. O peso de uma aresta de K consiste no valor do caminho mínimo, determinado em G, entre os nós terminais dessa mesma aresta.

5.9. Emparelhamento Máximo

Encontrar um emparelhamento máximo do grafo K a partir do algoritmo guloso descrito em II.2.3.

5.10. Busca Local para Emparelhamento Máximo

Aplicar a rotina de melhoria da solução de emparelhamento máximo de Pettie e Sanders [42], descrita em II.2.3. Essa busca local não se encontra no algoritmo original RFCS1. Mostrar-se-á que o desempenho dessa busca local reduz significativamente o tempo de percurso da solução, que consiste em um dos critérios da função objetivo hierárquica, comentada na seção III.1.

5.11. Obtenção do Grafo Euleriano

Construir um grafo Euleriano G_E, duplicando em G_U as arestas correspondentes ao caminho mínimo entre um nó e seu par, obtidos a partir do algoritmo de emparelhamento máximo. Esse processo acrescentará uma quantidade ímpar de arestas aos nós que possuem grau ímpar, tornando-os assim em grau par, o que caracteriza um grafo Euleriano G_E.

5.12. Obtenção da Rota Euleriana

Como G_E é Euleriano, torna-se possível traçar uma rota que percorra todas as arestas do grafo iniciando e terminando no mesmo nó. Para isso, utiliza-se o algoritmo "Next-Node", descrito na seção II.2.4.a.

5.13. Particionamento da Rota Euleriana

A última etapa do algoritmo consiste no particionamento da rota Euleriana em subrotas abertas, cada qual destinada a um leiturista. Esse método é similar ao algoritmo RFCS1, porém com uma diferença fundamental que considera proibido uma rota iniciar ou terminar em posição intermediária de uma aresta. Desse modo, a repartição da rota é feita iniciando de um nó qualquer da rota Euleriana, acumulando os pesos das arestas visitadas até que a carga horária W do leiturista se esgote. Assim que isso ocorrer, finaliza-se essa subrota na última aresta requerida visitada. A subrota seguinte começa na próxima aresta requerida. Esse processo continua até que todas as arestas requeridas se encontrem em alguma subrota.

Até a etapa de particionamento da rota Euleriana, o número de partições (ou rotas) NP, era um valor desconhecido. Nesse sentido, pode-se afirmar que o número de partições da solução de RFCS2 é pós-determinado.

6. Algoritmo CFRS2

O algoritmo CFRS2, cujo diagrama de blocos se encontra na Figura 22, tem como base a estratégia agrupamento-roteamento de Wünderlich et al. [52], denominada CFRS1. Uma das características herdadas por CFRS2 consiste no uso das arestas paralelas, ou seja, há sempre duas arestas incidindo os mesmos nós. Esse par de arestas paralelas é armazenado em CFRS2 como se fosse uma única aresta, definida requerida quando ao menos uma das arestas paralelas que lhe deram origem também for. De forma análoga ao item anterior, os blocos com preenchimento escuro da Figura 22 representam as novas rotinas, inexistentes no algoritmo original (CFRS1).

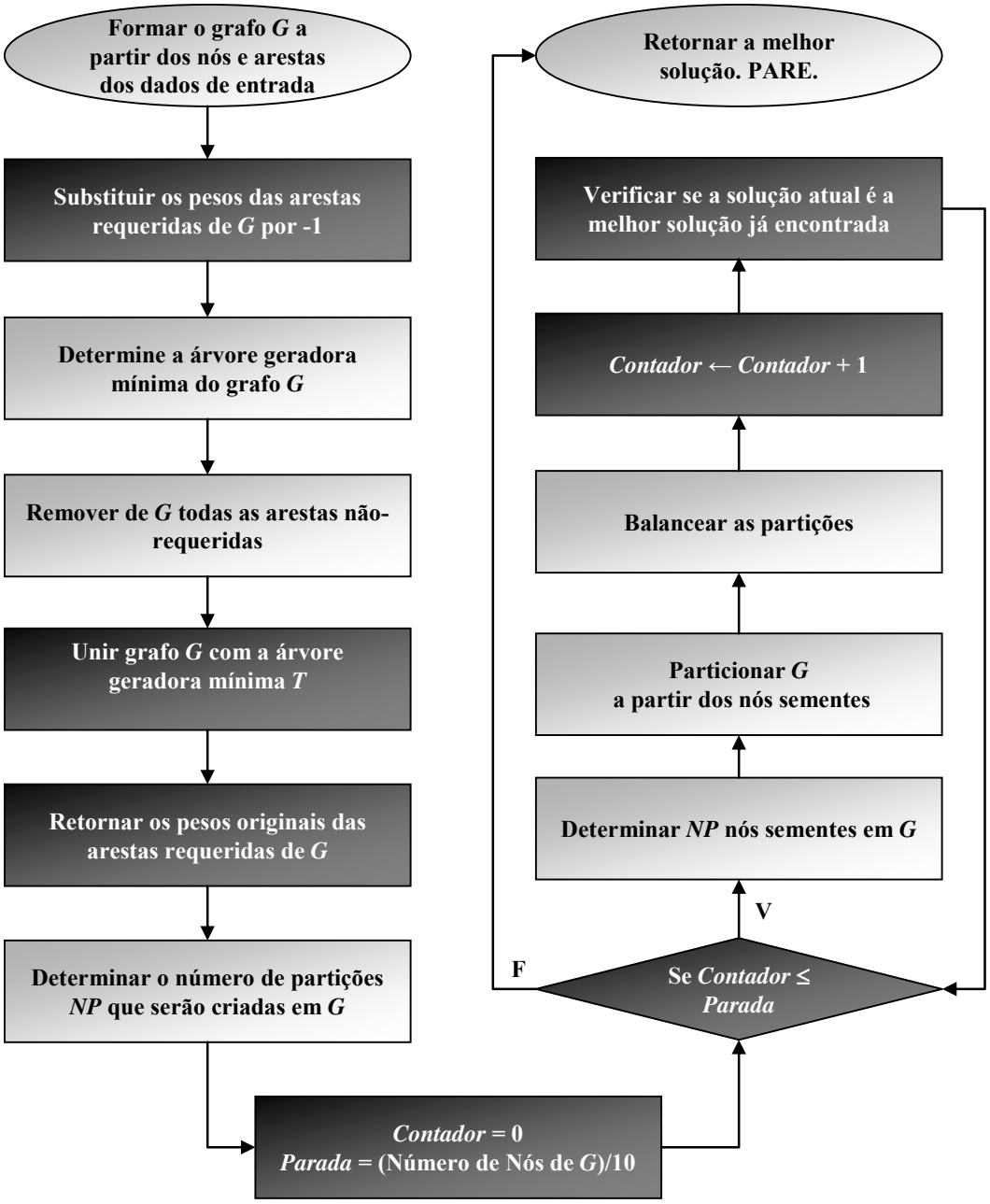


Figura 22. Diagrama de blocos do algoritmo CFRS2. Blocos destacados (preenchimento escuro) são inexistentes no algoritmo CFRS1.

6.1. Formação de G

Seja *G* a estrutura de dados de CFRS2 que representa o grafo inicial do problema. Considerando o conceito das arestas paralelas, deve-se garantir que sempre existam duas arestas entre dois nós adjacentes. Assim, quando se lê no arquivo de entrada uma aresta requerida, verifica-se se já existe em *G* uma aresta correspondente (com os mesmos nós

terminais). Caso ainda não exista, insere-se em *G* uma aresta requerida que representa um par de arestas paralelas. Desse par, uma aresta é requerida, com peso igual ao tempo de leitura, e a outra é não-requerida, com peso igual ao tempo de percurso. Considerando agora o caso de prévia existência em *G* de uma aresta correspondente, verifica-se o par de arestas paralelas que essa aresta representa. Sabe-se que uma aresta desse par será requerida, enquanto a outra não. Realiza-se então uma substituição da aresta paralela não-requerida pela aresta requerida do arquivo de entrada, com peso igual ao tempo de leitura. Ao fim dessa substituição, a aresta em *G* representará um par de arestas paralelas requeridas.

Quando se lê no arquivo de entrada uma aresta não-requerida, verifica-se também a prévia existência de uma aresta correspondente. Em caso positivo, nenhuma substituição é feita em *G* e a aresta do arquivo de entrada é descartada. Caso contrário, insere-se em *G* uma aresta não-requerida que representa um par de arestas paralelas não-requeridas.

A existência das arestas paralelas assegura a condição Euleriana do grafo, porque sempre haverá um número par de arestas incidindo qualquer nó. Desse modo, o algoritmo CFRS2 não requer a aplicação do algoritmo de emparelhamento máximo, na medida em que todas as operações em *G* preservam a condição das arestas paralelas.

6.2. Modificando o Peso das Arestas Requeridas

Equivalente ao item 5.3.

6.3. Árvore Geradora Mínima

Equivalente ao item 5.4.

6.4. Retornando os Pesos Originais das Arestas Requeridas

Equivalente ao item 5.5.

6.5. Remoção das Arestas Não-Requeridas

Equivalente ao item 5.6.

6.6. União de Grafos

Equivalente ao item 5.7.

6.7. Estimando o Número de Partições

O número de partições (ou rotas) *NP*, que devem ser criados no grafo para distribuir entre os leituristas consiste em um valor pré-determinado em CFRS2, obtido através da razão entre o peso total de G_U dividido pelo limitante superior da carga horária do leiturista W_{SUP}. No algoritmo original CFRS1, *NP* era determinado dividindo-se o peso de G_U pela carga horária desejada *W*, no entanto pode-se verificar que esse valor é superestimado já que desconsidera um tempo útil, corresponde à tolerância superior da carga horária.

6.8. Determinando os Nós Sementes

Os *NP* nós sementes, a partir de onde as partições se desenvolvem, são escolhidos aleatoriamente com distribuição uniforme entre os nós de G_U. Esse método é bem diferente daquele adotado pelo algoritmo CFRS1, onde o caráter aleatório da seleção se restringe à primeira semente, enquanto as seguintes são escolhidas pela maximização da soma das distâncias (de caminho mínimo) entre a próxima semente e aquelas previamente escolhidas.

O procedimento totalmente aleatório de seleção justifica-se pela má dispersão das sementes atestada em CFRS1 (o que será mostrado posteriormente), resultando em partições muito heterogêneas. Além disso, o procedimento de seleção das sementes pelas distâncias exige a aplicação do algoritmo de caminhos mínimos de todos para todos os nós, um procedimento de alto custo computacional ($O(V^3)$) para grandes instâncias. Portanto, com o novo procedimento de seleção, espera-se reduzir o tempo computacional, bem como aprimorar a qualidade das soluções a partir de uma melhor dispersão das sementes.

6.9. Distribuição das Arestas entre as Partições

A distribuição das arestas paralelas entre as partições ocorre do mesmo modo que no algoritmo CFRS1, ou seja, seleciona-se a partição de menor peso, nela inserindo o par de arestas paralelas, incidente a algum nó dessa partição, com o maior peso. Esse processo é repetido até que todas as arestas paralelas tenham sido alocadas em alguma partição.

6.10. Balanceamento

A última etapa do algoritmo consiste em redistribuir as arestas paralelas entre as partições com o objetivo exclusivo de reduzir a penalidade total da solução. São utilizadas as operações de “realocação de folha” e “realocação cíclica”, explicadas na seção II.5.2.d.

7. Estrutura de Dados

Os algoritmos RFCS2 e CFRS2 utilizaram em seus algoritmos a mesma estrutura de dados para representação de um grafo (arestas e nós). A Figura 23 a seguir revela como essa estrutura se organiza. Um grafo é armazenado na memória a partir de três estruturas simples: um vetor de nós, uma lista ligada de incidentes e uma lista ligada circular de arestas.

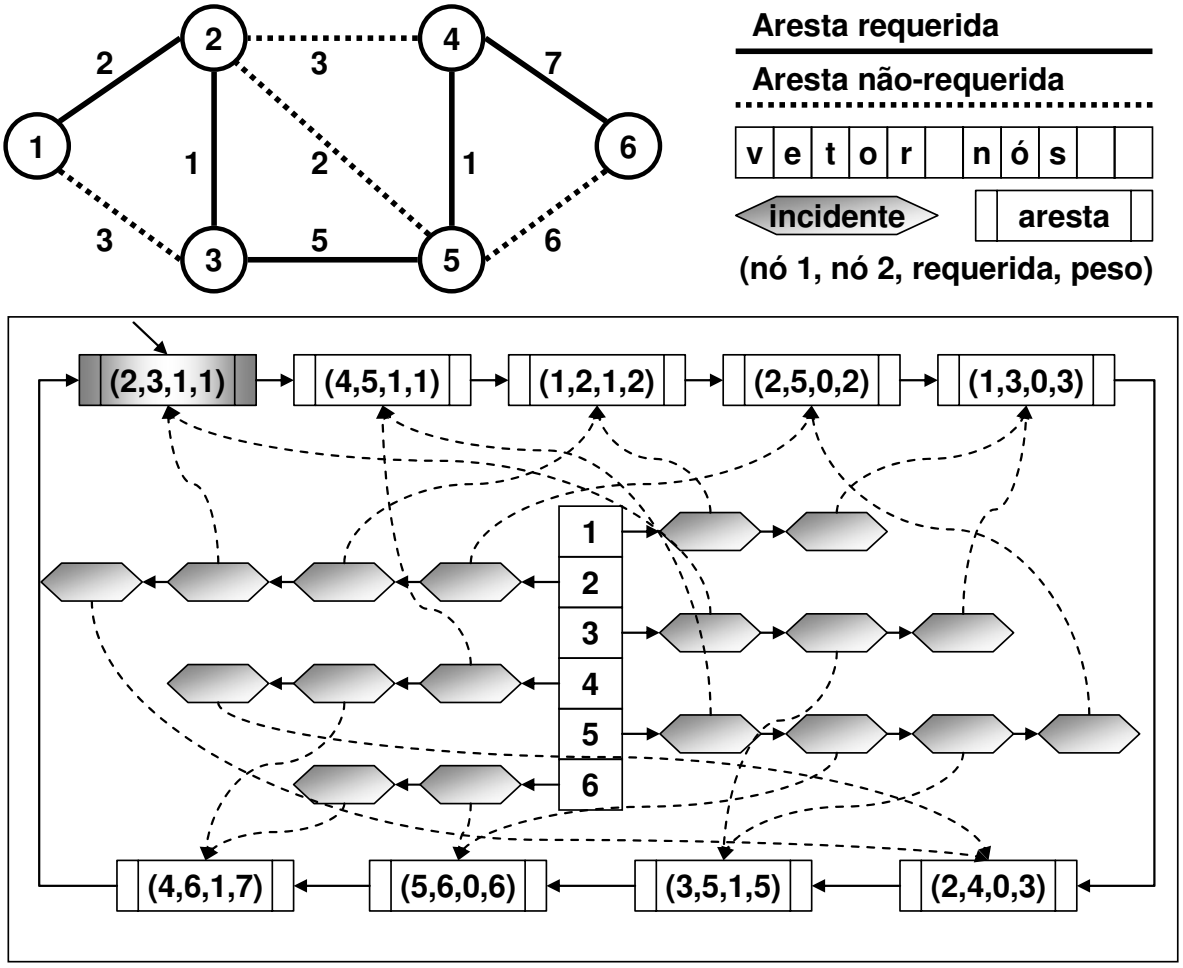


Figura 23. Representação da estrutura de dados utilizada.

O vetor de nós consiste em uma estrutura seqüencial simples, onde os índices do vetor representam os nós, enquanto que no conteúdo de cada campo do vetor se encontra um apontador para a cabeça da lista de arestas incidentes ao nó correspondente a esse campo.

Pode-se perceber que cada aresta armazena quatro informações: seus dois nós terminais; a condição requerida (valor um) ou não-requerida (valor zero); e o peso. A lista ligada circular de aresta está ordenada pelo peso de modo crescente, de modo que a cabeça da lista (destacada na Figura 23) possui o menor peso dentre as arestas.

O objeto "incidente" consiste em um vínculo entre um determinado nó e uma aresta a ele incidente. Desse modo, esse objeto facilita a coleta de informações das arestas incidentes de um determinado nó.

8. Testes Realizados

De modo a comparar a eficiência dos novos algoritmos RFCS2 e CFRS2 entre si e entre seus originais RFCS1 e CFRS1, foram utilizadas 144 instâncias classificadas por grupos (número de nós), famílias (número de componentes conexos) e classes (porcentagem de arestas não-requeridas no interior dos componentes conexos). A seguir, os grupos, famílias e classes das instâncias estão subdivididos de acordo com a característica do grafo que eles representam.

- Grupo 1 (64 nós, 48 instâncias):
  - Família A (conexo, 12 instâncias):
    - Classe I (nenhuma aresta não-requerida, 3 instâncias).
    - Classe II (25% de arestas não-requeridas, 3 instâncias).
    - Classe III (50% de arestas não-requeridas, 3 instâncias).
    - Classe IV (75% de arestas não-requeridas, 3 instâncias).
  - Família B (2 componentes conexos, 12 instâncias): Classes I, II, III e IV.
  - Família C (4 componentes conexos, 12 instâncias): Classes I, II, III e IV.
  - Família D (8 componentes conexos, 12 instâncias): Classes I, II, III e IV.
- Grupo 2 (256 nós, 48 instâncias): Famílias A, B, C e D: Classes I, II, III e IV.
- Grupo 3 (1024 nós, 48 instâncias): Famílias A, B, C e D: Classes I, II, III e IV.

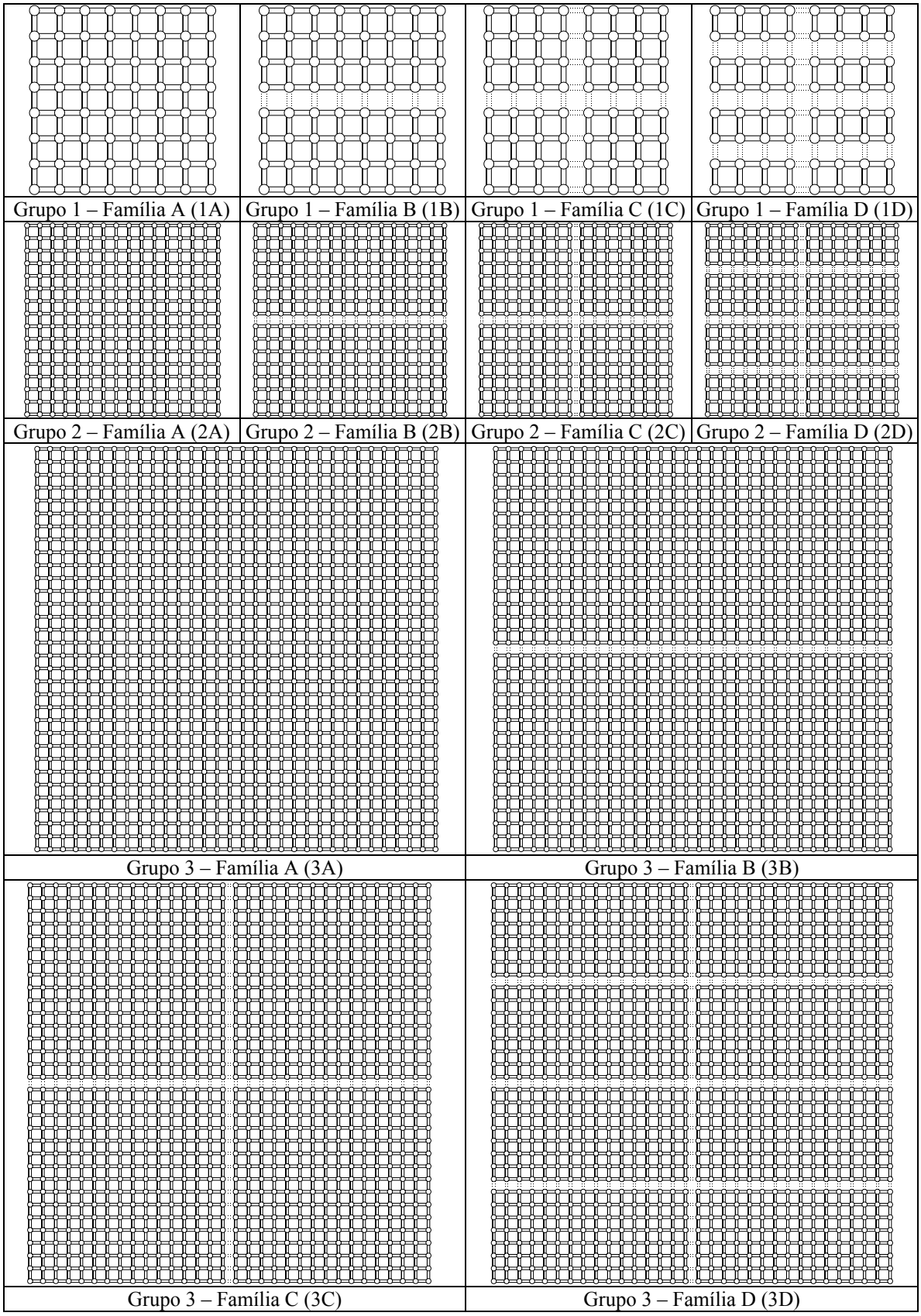


Figura 24. As possíveis combinações Grupo-Família das instâncias.

A Figura 24 ilustra todas as doze combinações Grupo-Família das instâncias, desconsiderada a classe. Pode-se notar que essas instâncias foram criadas de modo a simular as condições reais que os leituristas encontram nas regiões urbanas que demandam serviço de leitura. Pressupõe-se nessas instâncias que a leitura é do tipo U, ou seja, as faces de quadra são lidas separadamente. Desse modo, cada nó representa um cruzamento de ruas e cada aresta representa uma das faces de quadra (esquerda ou direita), o que explica a presença de duas arestas entre dois nós adjacentes. Como também existe a possibilidade de haver faces de quadra sem demanda por serviço de leitura, essa situação estaria representada por uma aresta não-requerida (arestas pontilhadas na Figura 24).

Desse ponto em diante, as instâncias serão referidas da seguinte forma:

<grupo><família><classe><(repetição)>

A instância 2D1(3), portanto, se refere à instância do Grupo 2, Família D, Classe 1, Repetição 3.

É importante salientar que os grafos das instâncias são todos originalmente conexos, ou seja, contém somente um componente conexo. O número de componentes conexos a que se refere cada Família decorre da desconexidade após a remoção das arestas não-requeridas. Desse modo, um grafo do tipo 3C, por exemplo, apresentará quatro componentes conexos após a remoção de todas as arestas não-requeridas.

As instâncias foram geradas automaticamente a partir de um algoritmo. De acordo com o Grupo, Família e Classe do grafo que se desejava criar, esse algoritmo gerava as arestas, inserindo-as linha a linha em um arquivo texto. As informações impressas no arquivo texto correspondiam ao nó inicial, no final, tempo de leitura e tempo de percurso. Os tempos de leitura (*TL*) e percurso (*TP*) foram gerados aleatoriamente com valores dentro dos seguintes intervalos:

$TL = 17{,}5 \pm 12{,}5 = [5{,}30]$ minutos

$TP = 2{,}5 \pm 2{,}5 = [0{,}5]$ minutos

A exceção ocorre para as arestas não-requeridas, cujo tempo de leitura é pré-determinado e igual a zero.

9. Geração das Saídas Gráficas

Devido à necessidade de visualizar as soluções dos algoritmos para validá-las, foi fundamental o desenvolvimento de um processo automático de geração das saídas gráficas a partir dos arquivos de saída dos algoritmos. Esse processo foi possível a partir da criação de um programa computacional que, lendo os arquivos contendo as soluções dos algoritmos, fazia interface com um pacote computacional de desenho técnico (AutoCad® 2006), gerando a representação do grafo e suas partições.

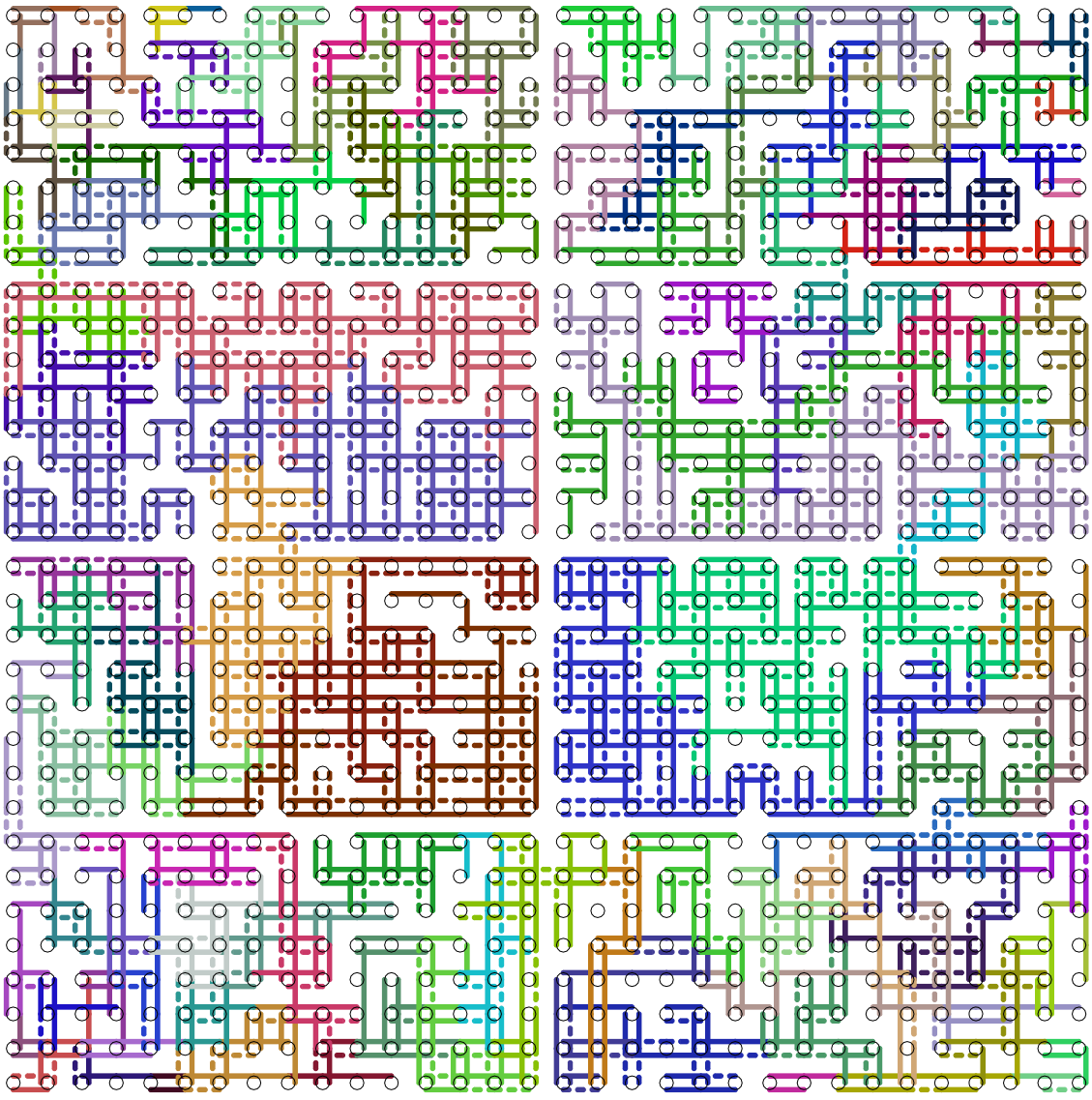


Figura 25. Solução do algoritmo CFRS1 para instância 3D3(1).

Na Figura 25 apresenta-se a saída gráfica para a instância 3D3(1) utilizando o algoritmo CFRS1. Nessa Figura, e nas demais representações gráficas adiante, as linhas contínuas representam arestas requeridas enquanto as linhas tracejadas representam arestas não-requeridas. Nem todas as arestas do grafo original são mostradas na representação gráfica, mas somente aquelas arestas efetivamente utilizadas na formação das partições. Essas partições, por sua vez, são representadas por cada conjunto contíguo de arestas com mesma coloração. No processo de geração da saída gráfica, a cor de cada partição era definida de maneira aleatória, tornando-se muito improvável que duas partições vizinhas possuam a mesma cor, considerando uma população superior a 16 milhões de cores.

Deve-se salientar que não foi possível exibir na saída gráfica a rota propriamente dita, ou seja, a seqüência de arestas que devem ser visitadas por cada leiturista. Não obstante, no arquivo de saída gerado pelos algoritmos, essa seqüência está presente. Por esse motivo, o termo partição, que é definido pelo conjunto não-ordenado de arestas, é mais adequado.

10. Análise dos Resultados

Os critérios de qualidade utilizados para comparar as soluções obtidas pelos quatro algoritmos foram os mesmos da função hierárquica (vide seção 1 desse capítulo), com a adição de um quarto critério, o tempo computacional. Desse modo, os quatro critérios utilizados para realizar a comparação entre os algoritmos são:

- Penalidade média (por rota);
- Número de partições;
- Tempo de percurso;
- Tempo computacional.

É possível afirmar que uma solução com reduzido valor de penalidade pode possuir um elevado número de partições, ou ainda, o tempo de percurso de uma solução é baixo, porém à custa de um elevado tempo de processamento. Essa afirmação implica que os critérios acima expostos são conflitantes. Nesse sentido, procurou-se saber qual algoritmo mostrou um melhor desempenho para cada um desses critérios, que dependendo da aplicação real, pode ser mais interessante que os demais critérios.

Como existem três repetições para cada combinação Grupo-Família-Classe de instâncias, a média de cada critério de qualidade dessas repetições foi determinada para posterior comparação.

Foram realizados dois tipos de comparação, uma envolvendo os quatro algoritmos, denominada comparação geral, e outra somente entre os algoritmos de mesmo tipo (RFCS ou CFRS), denominada comparação específica.

Uma comparação considera que um algoritmo foi vitorioso segundo algum critério de qualidade quando ele atinge a menor média para as três repetições de instâncias de um determinado Grupo-Classe-Família. Se mais de um algoritmo atingir a menor média, todos os algoritmos empatados são considerados vitoriosos. O algoritmo que apresentou o maior número de vitórias, segundo algum critério de qualidade, foi considerado o vencedor geral (dentre os quatro estudados) ou específico (algoritmos do mesmo tipo). Distinguem-se portanto os termos vitorioso e vencedor quando aplicados aos algoritmos.

IV. RESULTADOS E DISCUSSÃO

Esse capítulo se dedica à apresentação dos resultados obtidos com a aplicação dos quatro algoritmos de roteamento (CFRS1, CFRS2, RFCS1, RFCS2) nas 144 instâncias utilizadas como base de testes.

Na seção 1 serão apresentadas as tabelas e gráficos comparativos, a partir dos quais será realizada uma análise qualitativa dos algoritmos na seção 2. Em seguida, na seção 3, é feita uma discussão sobre a sensibilidade aos dados de entrada dos algoritmos. Na seção 4, são comentados os efeitos da busca local de emparelhamento máximo sobre os resultados do algoritmo RFCS2. Discute-se na seção 5 a modificação do critério de seleção de semente e seus efeitos sobre a solução do algoritmo CFRS2. A seção 6 mostra como a nova função de penalidade auxilia no processo de homogeneização das violações das rotas. Finalmente, na seção 7, são mostrados os diferentes aspectos visuais observados nas soluções de RFCS2 e CFRS2.

1. Apresentação dos Resultados

Devido ao elevado volume de informações contidas nos arquivos de entrada e saída dos algoritmos, bem como o número elevado de linhas de código fonte dos algoritmos, torna-se inviável expô-los nesse trabalho. No entanto, no CD-ROM anexo estão contidas todas as informações acima citadas, bem como as saídas gráficas das soluções obtidas por cada algoritmo e um arquivo de instruções para facilitar o entendimento e interpretação dos dados. Nesse trabalho, os resultados apresentados consistem em tabelas comparativas (Tabela 4 a Tabela 15) contendo informações pertinentes à análise comparativa dos algoritmos, bem como as saídas gráficas (vide Anexo I) das melhores soluções obtidas para cada instância (segundo o critério de penalidade), destacando o algoritmo que obteve esse resultado.

A seguir, as Tabelas 4 a 15 apresentam resultados coletados das soluções dos quatro algoritmos para as 144 instâncias utilizadas. Essas tabelas estão organizadas de modo que cada uma apresenta os resultados referentes à determinada combinação Grupo-Família de instâncias. Além disso, em cada tabela subdividem-se em suas linhas as Classes (I, II, III e IV), enquanto nas colunas subdividem-se os algoritmos analisados (RFCS1, RFCS2, CFRS1 e CFRS2) e as repetições de cada Classe. Foram analisados quatro critérios de

qualidade: penalidade média, número de rotas, tempo de percurso e tempo de execução. A média desses critérios de qualidade, referente às três repetições de cada combinação Grupo-Família-Classe, foi determinada para os quatro algoritmos. As células com preenchimento cinza escuro e texto branco fazem referência ao algoritmo (coluna) que obteve o melhor desempenho geral (dentre todos os algoritmos) referente a determinado critério de qualidade (linha) de uma instância. Por sua vez, as células com preenchimento cinza claro e texto preto fazem referência ao algoritmo (coluna) que obteve o melhor desempenho específico ao tipo de algoritmo a que pertence RFCS ou CFRS. Alguns exemplos de como interpretar as tabelas são fornecidos a seguir:

Na Tabela 4, por exemplo, pode-se verificar que o algoritmo RFCS2 obteve o melhor desempenho geral referente ao critério de Penalidade Média (0 minuto) para as instâncias da Classe I. Se forem considerados somente os algoritmos do tipo CFRS, pode-se afirmar que o algoritmo CFRS2 obteve o melhor desempenho específico (0,2 minutos).

Na Tabela 5, enquanto o algoritmo CFRS2 obteve o melhor desempenho geral para o critério de Tempo de Percurso (0,8 minutos) para as instâncias da Classe I, o algoritmo RFCS1 obteve o melhor desempenho específico (13,1 minutos).

Na Tabela 7, pode-se observar a ocorrência de um empate de desempenho entre os quatro algoritmos para as instâncias da Classe III, referente ao critério de Número de Rotas (5,7).

Tabela 4. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 1 – Família A.

Grupo 1 - Família A		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	3.3	1.1	2.1	2.2	0.0	0.0	0.0	**0.0**	1.9	0.7	0.9	1.2	0.0	0.3	0.1	**0.2**
	Número de Rotas	13.0	14.0	14.0	13.7	12.0	13.0	13.0	**12.7**	13.0	14.0	14.0	13.7	12.0	13.0	13.0	**12.7**
	Tempo de Percurso [min]	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**
	Tempo de Execução [s]	0.015	0.016	0.031	**0.021**	0.047	0.031	0.031	0.036	0.391	0.391	0.391	0.391	0.359	0.375	0.344	**0.359**
Classe II	Penalidade Média [min]	1.5	1.4	0.9	**1.3**	0.0	2.6	1.3	1.3	1.4	0.8	0.6	0.9	0.0	0.8	0.5	**0.5**
	Número de Rotas	11.0	10.0	10.0	10.3	10.0	10.0	10.0	**10.0**	11.0	10.0	10.0	10.3	10.0	10.0	10.0	**10.0**
	Tempo de Percurso [min]	61.4	48.6	35.3	48.4	33.3	41.4	39.9	**38.2**	52.1	68.0	93.8	71.3	43.8	68.3	95.4	**69.2**
	Tempo de Execução [s]	0.031	0.016	0.016	**0.021**	0.047	0.016	0.015	0.026	0.313	0.297	0.281	0.297	0.281	0.266	0.297	**0.281**
Classe III	Penalidade Média [min]	0.2	1.1	8.1	**3.1**	0.0	1.6	8.9	3.5	5.3	0.6	3.3	3.1	0.0	0.9	2.8	**1.2**
	Número de Rotas	7.0	8.0	7.0	**7.3**	7.0	8.0	7.0	**7.3**	8.0	8.0	7.0	7.7	7.0	8.0	7.0	**7.3**
	Tempo de Percurso [min]	73.0	72.3	92.2	79.2	83.1	69.4	83.3	**78.6**	117.8	129.8	102.6	**116.7**	118.1	130.1	111.3	119.8
	Tempo de Execução [s]	0.015	0.031	0.016	**0.021**	0.031	0.016	0.031	0.026	0.219	0.219	0.187	0.208	0.156	0.172	0.141	**0.156**
Classe IV	Penalidade Média [min]	12.4	11.9	0.0	**8.1**	13.7	13.6	0.0	9.1	5.5	8.3	11.4	**8.4**	5.9	8.3	23.5	12.6
	Número de Rotas	4.0	4.0	4.0	**4.0**	4.0	4.0	4.0	**4.0**	4.0	4.0	5.0	4.3	4.0	4.0	4.0	**4.0**
	Tempo de Percurso [min]	123.7	126.8	110.8	120.4	109.4	117.2	103.1	**109.9**	145.4	123.7	109.6	**126.2**	138.8	123.7	137.0	133.1
	Tempo de Execução [s]	0.016	0.015	0.016	**0.016**	0.031	0.031	0.016	0.026	0.094	0.094	0.109	0.099	0.078	0.062	0.062	**0.067**

Tabela 5. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 1 – Família B.

Grupo 1 - Família B		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.0	3.5	7.2	3.6	0.0	0.0	0.0	**0.0**	0.1	1.9	0.5	0.8	0.5	0.1	0.4	**0.4**
	Número de Rotas	12.0	13.0	13.0	12.7	12.0	12.0	12.0	**12.0**	12.0	13.0	12.0	12.3	12.0	12.0	12.0	**12.0**
	Tempo de Percurso [min]	13.7	5.7	19.9	**13.1**	17.6	4.4	20.6	14.2	1.3	1.3	1.3	1.3	1.3	0.0	1.3	**0.8**
	Tempo de Execução [s]	0.015	0.016	0.015	**0.015**	0.031	0.031	0.016	0.026	0.343	0.344	0.328	0.338	0.297	0.328	0.328	**0.318**
Classe II	Penalidade Média [min]	3.9	5.0	2.8	3.9	0.0	0.0	0.0	**0.0**	1.8	1.7	0.8	1.5	1.1	0.3	1.0	**0.8**
	Número de Rotas	10.0	10.0	10.0	10.0	9.0	9.0	9.0	**9.0**	10.0	10.0	10.0	10.0	9.0	9.0	10.0	**9.3**
	Tempo de Percurso [min]	72.1	66.0	48.0	62.0	51.5	60.1	48.4	**53.3**	75.1	74.1	89.1	79.4	74.8	78.4	69.6	**74.3**
	Tempo de Execução [s]	0.015	0.016	0.015	**0.015**	0.016	0.016	0.031	0.021	0.281	0.281	0.266	0.276	0.250	0.234	0.266	**0.250**
Classe III	Penalidade Média [min]	1.8	0.7	0.2	**0.9**	4.4	1.4	0.2	2.0	1.4	1.3	6.6	3.1	0.5	1.0	0.4	**0.6**
	Número de Rotas	6.0	7.0	7.0	**6.7**	6.0	7.0	7.0	**6.7**	6.0	7.0	8.0	7.0	6.0	7.0	7.0	**6.7**
	Tempo de Percurso [min]	84.1	90.6	55.9	76.9	66.0	70.5	67.2	**67.9**	109.2	97.1	108.5	**104.9**	122.7	111.1	135.7	123.2
	Tempo de Execução [s]	0.016	0.015	0.032	0.021	0.016	0.016	0.016	**0.016**	0.140	0.172	0.188	0.167	0.125	0.156	0.172	**0.151**
Classe IV	Penalidade Média [min]	0.0	10.9	23.2	11.4	0.7	12.2	0.0	**4.3**	21.3	6.9	15.2	14.5	0.8	8.2	12.6	**7.2**
	Número de Rotas	3.0	4.0	4.0	3.7	3.0	4.0	3.0	**3.3**	4.0	4.0	4.0	4.0	3.0	4.0	4.0	**3.7**
	Tempo de Percurso [min]	155.4	124.2	155.4	145.0	134.9	116.2	112.6	**121.2**	142.8	114.5	141.8	**133.0**	163.4	113.8	145.5	140.9
	Tempo de Execução [s]	0.016	0.015	0.016	**0.016**	0.016	0.031	0.016	0.021	0.094	0.078	0.079	0.084	0.078	0.062	0.063	**0.068**

Tabela 6. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 1 – Família C.

Grupo 1 - Família C		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	4.8	5.8	0.2	3.6	0.0	0.0	0.5	**0.2**	0.0	0.4	1.4	**0.6**	0.5	0.4	3.1	1.3
	Número de Rotas	12.0	12.0	12.0	12.0	11.0	11.0	12.0	**11.3**	11.0	11.0	12.0	11.3	11.0	11.0	11.0	**11.0**
	Tempo de Percurso [min]	28.2	24.1	35.5	**29.3**	27.7	30.8	32.6	30.4	5.6	4.1	2.9	**4.2**	5.6	4.1	4.6	4.8
	Tempo de Execução [s]	0.031	0.016	0.031	**0.026**	0.016	0.031	0.031	**0.026**	0.297	0.297	0.313	0.302	0.281	0.266	0.297	**0.281**
Classe II	Penalidade Média [min]	0.0	0.2	0.6	**0.3**	0.0	0.3	0.9	0.4	6.2	0.7	0.8	2.6	0.4	0.7	3.0	**1.4**
	Número de Rotas	9.0	9.0	10.0	**9.3**	9.0	9.0	10.0	**9.3**	10.0	9.0	10.0	9.7	9.0	9.0	10.0	**9.3**
	Tempo de Percurso [min]	62.6	61.5	63.2	62.4	50.7	67.2	49.8	**55.9**	53.6	73.1	58.9	61.9	64.6	64.7	54.4	**61.2**
	Tempo de Execução [s]	0.015	0.016	0.015	**0.015**	0.016	0.015	0.063	0.031	0.266	0.219	0.250	0.245	0.235	0.234	0.250	**0.240**
Classe III	Penalidade Média [min]	12.2	11.9	5.0	9.7	0.0	0.0	3.6	**1.2**	3.9	5.8	6.9	5.5	4.1	4.6	6.6	**5.1**
	Número de Rotas	7.0	7.0	7.0	7.0	6.0	6.0	7.0	**6.3**	7.0	7.0	7.0	**7.0**	7.0	7.0	7.0	**7.0**
	Tempo de Percurso [min]	65.5	77.4	72.0	**71.6**	69.9	78.8	87.9	78.9	114.1	119.6	80.4	104.7	114.2	116.4	78.2	**103.0**
	Tempo de Execução [s]	0.016	0.016	0.016	**0.016**	0.031	0.031	0.031	0.031	0.172	0.187	0.156	0.172	0.187	0.141	0.140	**0.156**
Classe IV	Penalidade Média [min]	27.9	13.6	19.3	20.3	0.0	16.1	21.1	**12.4**	17.4	13.6	22.2	17.8	15.1	13.5	11.7	**13.4**
	Número de Rotas	4.0	4.0	4.0	4.0	3.0	4.0	4.0	**3.7**	4.0	4.0	4.0	**4.0**	4.0	4.0	4.0	**4.0**
	Tempo de Percurso [min]	136.8	131.9	131.8	133.5	129.0	114.4	107.3	**116.9**	127.3	126.6	122.7	**125.6**	130.9	128.4	132.5	130.6
	Tempo de Execução [s]	0.015	0.016	0.016	**0.016**	0.032	0.016	0.015	0.021	0.078	0.079	0.093	0.083	0.078	0.062	0.078	**0.073**

Tabela 7. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 1 – Família D.

Grupo 1 - Família D		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.0	5.2	2.0	2.4	0.8	0.0	1.5	**0.7**	4.1	5.3	2.2	3.9	2.3	0.4	1.5	**1.4**
	Número de Rotas	10.0	10.0	10.0	10.0	10.0	9.0	10.0	**9.7**	10.0	10.0	10.0	10.0	10.0	9.0	9.0	**9.3**
	Tempo de Percurso [min]	40.2	62.7	73.5	**58.8**	67.0	54.9	69.2	63.7	11.7	13.6	12.5	**12.6**	12.4	15.5	11.4	13.1
	Tempo de Execução [s]	0.016	0.015	0.016	**0.016**	0.031	0.031	0.031	0.031	0.234	0.219	0.219	0.224	0.235	0.218	0.219	**0.224**
Classe II	Penalidade Média [min]	1.0	0.2	7.2	**2.8**	1.0	0.4	8.5	3.3	1.2	1.4	4.8	2.5	1.5	1.8	4.1	**2.5**
	Número de Rotas	8.0	8.0	7.0	**7.7**	8.0	8.0	7.0	**7.7**	8.0	8.0	7.0	**7.7**	8.0	8.0	7.0	**7.7**
	Tempo de Percurso [min]	72.1	92.3	95.2	86.5	62.5	82.0	70.9	**71.8**	55.9	79.2	81.7	**72.3**	59.2	77.0	81.6	72.6
	Tempo de Execução [s]	0.016	0.031	0.016	0.021	0.032	0.015	0.015	**0.021**	0.172	0.187	0.156	**0.172**	0.188	0.187	0.156	0.177
Classe III	Penalidade Média [min]	5.1	0.4	2.4	**2.6**	7.1	1.0	1.7	3.2	5.7	11.6	7.2	8.2	3.6	5.4	15.2	**8.1**
	Número de Rotas	6.0	6.0	5.0	**5.7**	6.0	6.0	5.0	**5.7**	6.0	6.0	5.0	**5.7**	6.0	6.0	5.0	**5.7**
	Tempo de Percurso [min]	93.7	106.4	100.6	100.2	75.1	84.2	93.6	**84.3**	110.7	103.8	104.0	106.1	103.2	109.9	103.9	**105.7**
	Tempo de Execução [s]	0.015	0.032	0.015	**0.021**	0.031	0.016	0.015	**0.021**	0.125	0.125	0.094	**0.115**	0.125	0.125	0.094	**0.115**
Classe IV	Penalidade Média [min]	24.5	25.6	26.5	25.5	0.0	0.0	0.0	**0.0**	21.6	16.5	17.9	**18.7**	24.9	15.5	16.6	19.0
	Número de Rotas	4.0	4.0	4.0	4.0	3.0	3.0	3.0	**3.0**	4.0	4.0	4.0	**4.0**	4.0	4.0	4.0	**4.0**
	Tempo de Percurso [min]	127.8	110.1	136.0	124.6	112.6	110.1	116.3	**113.0**	121.1	139.2	133.8	**131.4**	109.0	146.3	150.7	135.3
	Tempo de Execução [s]	0.015	0.016	0.015	**0.015**	0.032	0.015	0.032	0.026	0.094	0.094	0.078	0.089	0.062	0.078	0.078	**0.073**

Tabela 8. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 2 – Família A.

Grupo 2 - Família A		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	0.2	0.2	0.2	**0.2**	0.9	0.8	0.6	0.7
	Número de Rotas	57.0	57.0	58.0	57.3	54.0	55.0	55.0	**54.7**	56.0	56.0	57.0	56.3	53.0	53.0	54.0	**53.3**
	Tempo de Percurso [min]	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**
	Tempo de Execução [s]	0.672	0.687	0.688	**0.682**	0.828	0.813	0.813	0.818	14.078	13.859	13.625	13.854	12.063	11.688	11.828	**11.860**
Classe II	Penalidade Média [min]	0.1	0.1	0.2	0.1	0.0	0.0	0.0	**0.0**	0.7	1.6	0.6	1.0	0.4	0.5	0.7	**0.5**
	Número de Rotas	45.0	44.0	44.0	44.3	43.0	42.0	42.0	**42.3**	45.0	45.0	44.0	44.7	43.0	42.0	42.0	**42.3**
	Tempo de Percurso [min]	205.4	195.7	203.4	201.5	163.4	164.5	181.6	**169.8**	266.4	265.5	246.0	**259.3**	306.7	309.9	296.8	304.5
	Tempo de Execução [s]	1.219	1.515	2.218	**1.651**	1.687	2.422	3.141	2.417	11.859	11.484	11.531	11.625	10.735	10.828	10.203	**10.589**
Classe III	Penalidade Média [min]	0.2	0.3	0.9	**0.4**	1.8	0.0	0.0	0.6	9.9	6.0	0.7	5.5	0.8	0.3	0.3	**0.5**
	Número de Rotas	29.0	30.0	31.0	30.0	28.0	28.0	29.0	**28.3**	30.0	30.0	31.0	30.3	28.0	29.0	30.0	**29.0**
	Tempo de Percurso [min]	316.6	330.8	313.1	320.2	310.0	289.4	253.5	**284.3**	445.7	405.5	441.0	**430.7**	467.1	440.9	509.3	472.4
	Tempo de Execução [s]	0.797	1.485	1.922	**1.401**	1.672	2.204	2.453	2.110	7.063	7.531	8.234	7.609	7.016	7.000	7.766	**7.261**
Classe IV	Penalidade Média [min]	3.3	0.6	0.3	1.4	1.2	0.0	0.0	**0.4**	40.7	53.8	46.1	46.9	10.4	11.9	7.9	**10.1**
	Número de Rotas	16.0	16.0	17.0	16.3	15.0	15.0	16.0	**15.3**	16.0	16.0	18.0	16.7	15.0	16.0	17.0	**16.0**
	Tempo de Percurso [min]	507.0	499.7	545.4	517.4	489.9	473.5	525.2	**496.2**	514.1	504.1	487.5	501.9	500.5	471.5	514.0	**495.4**
	Tempo de Execução [s]	1.797	1.531	1.391	**1.573**	2.625	2.938	2.891	2.818	2.969	2.938	3.140	**3.016**	2.813	3.047	3.546	3.135

Tabela 9. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 2 – Família B.

Grupo 2 - Família B		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.2	0.1	0.3	0.2	0.0	0.0	0.0	**0.0**	0.2	0.2	1.2	**0.6**	1.0	1.2	1.2	1.2
	Número de Rotas	57.0	57.0	57.0	57.0	54.0	54.0	54.0	**54.0**	55.0	56.0	55.0	55.3	52.0	53.0	52.0	**52.3**
	Tempo de Percurso [min]	21.1	38.2	17.3	**25.5**	36.9	16.5	36.0	29.8	0.0	1.3	0.0	**0.4**	0.0	1.3	1.3	0.8
	Tempo de Execução [s]	0.687	0.704	0.671	**0.687**	0.828	0.828	0.828	0.828	12.859	12.937	12.906	12.901	11.172	11.391	11.172	**11.245**
Classe II	Penalidade Média [min]	0.2	0.2	0.2	0.2	0.5	0.0	0.0	**0.2**	0.5	0.8	0.5	0.6	0.6	0.4	0.6	**0.5**
	Número de Rotas	44.0	43.0	43.0	43.3	42.0	41.0	41.0	**41.3**	44.0	43.0	43.0	43.3	42.0	41.0	41.0	**41.3**
	Tempo de Percurso [min]	221.1	229.2	204.2	218.2	176.0	220.5	191.0	**195.8**	273.3	247.7	253.9	**258.3**	281.2	265.1	262.8	269.7
	Tempo de Execução [s]	1.860	1.875	1.406	**1.714**	2.531	2.984	1.985	2.500	11.032	10.906	10.969	10.969	10.203	9.922	10.172	**10.099**
Classe III	Penalidade Média [min]	0.2	0.7	0.4	0.4	0.0	0.0	0.0	**0.0**	2.8	4.9	2.5	3.4	0.6	0.5	0.6	**0.6**
	Número de Rotas	28.0	28.0	30.0	28.7	27.0	26.0	28.0	**27.0**	29.0	28.0	30.0	29.0	28.0	27.0	29.0	**28.0**
	Tempo de Percurso [min]	297.5	356.2	344.5	332.7	290.5	304.1	268.5	**287.7**	405.7	421.4	446.3	**424.5**	448.4	444.4	435.7	442.8
	Tempo de Execução [s]	0.984	1.890	1.219	**1.364**	2.407	2.781	2.234	2.474	7.359	7.000	7.766	7.375	6.563	6.453	7.078	**6.698**
Classe IV	Penalidade Média [min]	0.7	0.3	3.3	1.4	0.0	0.0	2.2	**0.7**	35.2	26.7	31.7	31.2	16.0	11.6	6.1	**11.2**
	Número de Rotas	16.0	16.0	16.0	16.0	15.0	15.0	15.0	**15.0**	16.0	17.0	16.0	16.3	16.0	16.0	16.0	**16.0**
	Tempo de Percurso [min]	535.5	526.9	477.3	513.2	446.6	464.9	456.3	**455.9**	473.2	485.7	540.2	**499.7**	506.3	505.3	557.6	523.1
	Tempo de Execução [s]	1.718	1.890	1.734	**1.781**	1.969	2.109	2.593	2.224	3.390	3.500	3.156	3.349	2.906	3.125	3.172	**3.068**

Tabela 10. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 2 – Família C.

Grupo 2 - Família C		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.1	0.2	0.2	0.2	0.0	0.0	0.0	**0.0**	0.3	0.2	0.3	**0.3**	1.7	0.9	1.7	1.4
	Número de Rotas	55.0	55.0	55.0	55.0	52.0	52.0	52.0	**52.0**	53.0	54.0	53.0	53.3	51.0	51.0	51.0	**51.0**
	Tempo de Percurso [min]	37.7	69.4	64.7	**57.3**	65.4	50.6	58.0	58.0	3.5	2.8	3.7	3.3	2.1	0.0	3.7	**1.9**
	Tempo de Execução [s]	0.687	0.719	0.671	**0.692**	0.812	0.844	0.844	0.833	12.437	12.859	12.828	12.708	11.281	11.156	11.110	**11.182**
Classe II	Penalidade Média [min]	0.2	0.2	0.2	0.2	0.0	0.0	0.0	**0.0**	2.0	0.9	0.9	1.3	0.5	0.6	0.5	**0.5**
	Número de Rotas	42.0	41.0	43.0	42.0	40.0	39.0	41.0	**40.0**	42.0	41.0	43.0	42.0	40.0	39.0	41.0	**40.0**
	Tempo de Percurso [min]	228.3	214.9	266.8	236.7	205.8	209.4	211.2	**208.8**	265.1	288.5	255.0	**269.5**	266.0	293.5	265.6	275.0
	Tempo de Execução [s]	1.469	1.422	1.422	**1.438**	1.938	2.266	2.015	2.073	10.375	10.062	10.734	10.390	9.438	9.688	9.843	**9.656**
Classe III	Penalidade Média [min]	0.3	0.4	0.1	0.3	0.0	0.0	0.2	**0.1**	7.0	1.2	7.0	5.1	1.9	2.0	1.4	**1.7**
	Número de Rotas	28.0	29.0	27.0	28.0	27.0	27.0	26.0	**26.7**	29.0	29.0	28.0	28.7	28.0	28.0	27.0	**27.7**
	Tempo de Percurso [min]	325.7	317.1	358.8	333.9	288.3	296.6	281.7	**288.9**	436.9	433.8	427.5	432.7	413.4	419.2	459.5	**430.7**
	Tempo de Execução [s]	1.438	1.844	1.188	**1.490**	2.641	2.203	1.750	2.198	7.172	7.469	6.937	7.193	6.656	6.828	6.203	**6.562**
Classe IV	Penalidade Média [min]	4.4	2.5	0.1	2.3	0.1	0.6	0.0	**0.2**	49.3	40.3	27.5	39.0	16.4	14.2	19.2	**16.6**
	Número de Rotas	16.0	16.0	14.0	15.3	15.0	15.0	13.0	**14.3**	16.0	16.0	15.0	15.7	15.0	15.0	14.0	**14.7**
	Tempo de Percurso [min]	535.9	543.5	602.3	560.5	487.6	498.5	525.9	**504.0**	562.6	476.9	543.2	527.6	505.3	529.1	527.8	**520.7**
	Tempo de Execução [s]	1.875	1.703	2.094	**1.891**	3.031	1.719	1.875	2.208	3.015	3.360	3.406	3.260	2.922	2.907	2.797	**2.875**

Tabela 11. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 2 – Família D.

Grupo 2 - Família D		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	8.8	14.2	9.4	10.8	1.6	1.6	1.4	**1.5**
	Número de Rotas	51.0	50.0	50.0	50.3	49.0	48.0	48.0	**48.3**	50.0	49.0	49.0	49.3	47.0	47.0	47.0	**47.0**
	Tempo de Percurso [min]	115.9	102.0	139.4	119.1	100.8	115.7	124.7	**113.7**	8.8	5.4	6.1	**6.7**	8.3	10.9	10.6	9.9
	Tempo de Execução [s]	0.688	0.672	0.672	**0.677**	0.844	0.844	0.828	0.839	10.891	9.985	11.235	**10.704**	11.891	10.391	10.453	10.912
Classe II	Penalidade Média [min]	0.3	0.2	0.1	0.2	0.0	0.0	0.0	**0.0**	7.7	6.8	11.5	8.7	1.3	1.6	1.9	**1.6**
	Número de Rotas	39.0	38.0	38.0	38.3	37.0	36.0	36.0	**36.3**	38.0	38.0	38.0	38.0	37.0	36.0	36.0	**36.3**
	Tempo de Percurso [min]	301.1	269.3	272.1	280.8	258.5	250.8	218.7	**242.7**	272.0	271.7	286.0	276.5	262.5	287.7	270.6	**273.6**
	Tempo de Execução [s]	1.656	1.390	1.672	**1.573**	1.954	2.953	1.781	2.229	9.375	9.265	9.438	9.359	8.922	9.109	9.265	**9.099**
Classe III	Penalidade Média [min]	0.4	0.3	0.3	**0.3**	0.0	1.8	0.0	0.6	20.3	19.9	21.2	20.5	1.7	3.2	4.2	**3.0**
	Número de Rotas	27.0	26.0	26.0	26.3	25.0	25.0	25.0	**25.0**	27.0	26.0	27.0	26.7	26.0	25.0	26.0	**25.7**
	Tempo de Percurso [min]	376.2	358.0	399.6	377.9	347.0	325.7	333.9	**335.5**	439.0	403.8	442.6	**428.5**	433.4	409.3	452.8	431.8
	Tempo de Execução [s]	1.563	1.735	1.500	**1.599**	2.688	1.891	2.156	2.245	6.359	5.843	6.016	6.073	6.250	5.656	5.812	**5.906**
Classe IV	Penalidade Média [min]	2.6	2.3	0.3	1.7	0.1	0.2	0.0	**0.1**	38.9	44.1	42.3	41.8	13.5	17.3	9.1	**13.3**
	Número de Rotas	15.0	16.0	15.0	15.3	14.0	15.0	14.0	**14.3**	15.0	16.0	15.0	15.3	14.0	15.0	15.0	**14.7**
	Tempo de Percurso [min]	527.2	538.0	609.9	558.3	538.1	552.8	493.7	**528.2**	517.3	497.0	531.4	515.2	518.0	490.4	536.6	**515.0**
	Tempo de Execução [s]	1.047	1.046	1.422	**1.172**	2.312	2.031	2.640	2.328	3.062	3.109	3.000	3.057	2.641	2.843	2.937	**2.807**

Tabela 12. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 3 – Família A.

Grupo 3 - Família A		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	2.9	2.9	2.7	2.8	0.6	0.5	0.6	**0.6**
	Número de Rotas	240.0	238.0	240.0	239.3	228.0	226.0	228.0	**227.3**	233.0	231.0	233.0	232.3	222.0	220.0	222.0	**221.3**
	Tempo de Percurso [min]	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	**0.0**
	Tempo de Execução [s]	128.766	129.250	129.328	**129.115**	143.938	145.094	145.109	144.714	1646.156	1665.640	1629.313	1647.036	499.297	501.391	516.718	**505.802**
Classe II	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	3.5	2.3	4.2	3.3	0.4	0.5	0.5	**0.4**
	Número de Rotas	186.0	181.0	182.0	183.0	177.0	172.0	172.0	**173.7**	185.0	181.0	181.0	182.3	177.0	172.0	173.0	**174.0**
	Tempo de Percurso [min]	785.7	860.3	817.5	821.2	688.7	670.6	702.7	**687.3**	1120.2	1217.3	1194.8	**1177.5**	1155.6	1306.5	1312.0	1258.0
	Tempo de Execução [s]	414.516	554.203	504.891	**491.203**	450.563	599.640	628.922	559.708	1176.282	1298.828	1217.890	1231.000	470.375	467.890	460.188	**466.151**
Classe III	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	14.5	12.4	14.4	13.7	0.6	0.4	0.5	**0.5**
	Número de Rotas	126.0	122.0	124.0	124.0	119.0	115.0	118.0	**117.3**	128.0	124.0	127.0	126.3	122.0	118.0	121.0	**120.3**
	Tempo de Percurso [min]	1215.0	1211.3	1190.8	1205.7	1148.5	1132.3	1151.8	**1144.2**	1750.8	1946.6	1917.4	**1871.6**	1832.6	1957.8	1951.7	1914.0
	Tempo de Execução [s]	479.203	541.266	455.500	**491.990**	666.625	620.859	491.953	593.146	597.890	737.047	670.531	668.489	335.875	330.890	343.797	**336.854**
Classe IV	Penalidade Média [min]	0.1	0.2	0.1	0.2	0.0	0.0	0.0	**0.0**	50.2	50.6	54.1	51.7	9.0	11.5	8.7	**9.7**
	Número de Rotas	70.0	71.0	68.0	69.7	66.0	67.0	65.0	**66.0**	71.0	72.0	70.0	71.0	68.0	68.0	66.0	**67.3**
	Tempo de Percurso [min]	2127.8	2055.0	1881.9	2021.6	1900.9	1902.1	1893.2	**1898.7**	2151.9	2176.5	2161.7	2163.4	2011.4	1919.6	2027.1	**1986.0**
	Tempo de Execução [s]	566.329	450.641	505.328	**507.433**	565.297	619.156	549.406	577.953	245.844	242.172	259.391	249.136	116.922	116.610	120.266	**117.933**

Tabela 13. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 3 – Família B.

Grupo 3 - Família B		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	3.4	2.5	3.4	3.1	0.6	0.6	0.7	**0.6**
	Número de Rotas	236.0	235.0	237.0	236.0	224.0	223.0	225.0	**224.0**	229.0	228.0	230.0	229.0	218.0	217.0	219.0	**218.0**
	Tempo de Percurso [min]	38.6	72.9	71.7	61.1	39.7	62.7	54.5	**52.3**	1.2	1.1	0.0	**0.8**	1.2	1.1	1.1	1.2
	Tempo de Execução [s]	128.297	127.953	127.406	**127.885**	144.453	144.532	146.204	145.063	1564.594	1575.516	1585.922	1575.344	509.421	509.640	510.016	**509.692**
Classe II	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	3.4	4.4	3.4	3.7	0.4	0.4	0.5	**0.4**
	Número de Rotas	181.0	180.0	179.0	180.0	172.0	171.0	170.0	**171.0**	180.0	179.0	178.0	179.0	171.0	171.0	170.0	**170.7**
	Tempo de Percurso [min]	826.4	898.7	853.4	859.5	744.8	728.0	777.9	**750.2**	1090.4	1140.5	1063.0	**1097.9**	1076.1	1145.3	1159.5	1126.9
	Tempo de Execução [s]	480.922	559.906	532.328	**524.385**	549.953	575.375	606.469	577.266	1189.766	1206.766	1203.047	1199.860	447.328	439.641	443.266	**443.412**
Classe III	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	12.7	13.4	12.7	12.9	0.7	0.6	0.6	**0.6**
	Número de Rotas	121.0	120.0	123.0	121.3	114.0	113.0	117.0	**114.7**	123.0	122.0	125.0	123.3	117.0	116.0	119.0	**117.3**
	Tempo de Percurso [min]	1239.6	1363.8	1262.1	1288.5	1125.1	1164.3	1127.6	**1139.0**	1831.6	1960.5	1863.1	1885.1	1860.4	1909.1	1836.5	**1868.7**
	Tempo de Execução [s]	464.625	444.531	486.204	**465.120**	634.250	520.812	549.281	568.114	767.766	659.594	626.594	684.651	331.687	323.907	323.438	**326.344**
Classe IV	Penalidade Média [min]	0.1	0.2	0.1	0.1	0.0	0.0	0.0	**0.0**	56.0	46.4	58.6	53.7	8.9	10.6	8.9	**9.5**
	Número de Rotas	68.0	67.0	63.0	66.0	64.0	64.0	60.0	**62.7**	69.0	68.0	64.0	67.0	66.0	65.0	61.0	**64.0**
	Tempo de Percurso [min]	2123.4	2055.0	2119.7	2099.4	1946.7	1974.5	2012.0	**1977.7**	2233.1	2113.0	2304.0	2216.7	2006.4	1953.9	2023.1	**1994.5**
	Tempo de Execução [s]	526.234	408.500	546.391	**493.708**	523.079	651.266	566.812	580.386	230.891	246.047	224.203	233.714	112.891	107.141	103.844	**107.959**

Tabela 14. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 3 – Família C.

Grupo 3 - Família C		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	3.2	3.2	3.2	3.2	0.6	0.7	0.7	**0.7**
	Número de Rotas	231.0	231.0	232.0	231.3	219.0	220.0	221.0	**220.0**	223.0	224.0	225.0	224.0	213.0	213.0	214.0	**213.3**
	Tempo de Percurso [min]	97.7	96.3	90.8	**94.9**	124.2	101.8	83.0	103.0	2.1	2.5	2.2	2.2	1.1	2.4	1.1	**1.5**
	Tempo de Execução [s]	127.438	127.546	127.594	**127.526**	144.969	144.344	145.593	144.969	1423.812	1494.735	1448.359	1455.635	509.922	504.156	501.750	**505.276**
Classe II	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.1	0.0	0.0	**0.0**	3.5	3.1	3.6	3.4	0.5	0.5	0.5	**0.5**
	Número de Rotas	174.0	177.0	174.0	175.0	165.0	168.0	165.0	**166.0**	173.0	176.0	173.0	174.0	165.0	168.0	165.0	**166.0**
	Tempo de Percurso [min]	882.5	908.3	924.4	905.1	779.5	799.6	796.9	**792.0**	1182.4	1141.4	1229.4	**1184.4**	1194.6	1219.0	1311.2	1241.6
	Tempo de Execução [s]	480.453	450.031	458.437	**462.974**	562.312	525.359	530.984	539.552	1151.735	1206.485	1147.828	1168.683	431.250	440.921	438.609	**436.927**
Classe III	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	10.9	13.8	11.7	12.1	0.9	0.7	0.6	**0.7**
	Número de Rotas	118.0	123.0	123.0	121.3	112.0	117.0	116.0	**115.0**	120.0	125.0	125.0	123.3	114.0	119.0	119.0	**117.3**
	Tempo de Percurso [min]	1346.0	1292.2	1299.0	1312.4	1180.7	1195.5	1085.5	**1153.9**	1728.7	1842.2	1814.7	**1795.2**	1821.6	1820.6	1859.9	1834.0
	Tempo de Execução [s]	439.000	435.625	552.110	**475.578**	456.485	454.015	558.172	489.557	715.625	623.188	722.843	687.219	330.906	322.687	343.094	**332.229**
Classe IV	Penalidade Média [min]	0.2	0.2	0.2	0.2	0.0	0.0	0.0	**0.0**	51.8	57.3	61.7	56.9	9.8	10.0	9.2	**9.7**
	Número de Rotas	66.0	69.0	63.0	66.0	63.0	65.0	59.0	**62.3**	68.0	70.0	64.0	67.3	64.0	67.0	61.0	**64.0**
	Tempo de Percurso [min]	2019.6	2080.4	2219.5	2106.5	1942.5	1934.0	2039.3	**1971.9**	2310.1	2200.2	2318.6	2276.3	2030.0	2042.0	2052.9	**2041.7**
	Tempo de Execução [s]	531.438	470.828	497.031	**499.766**	522.250	598.750	568.609	563.203	242.859	259.343	247.109	249.770	115.406	117.984	105.172	**112.854**

Tabela 15. Comparações de médias dos critérios de qualidade para as instâncias do Grupo 3 – Família D.

Grupo 3 - Família D		Algoritmo RFCS1				Algoritmo RFCS2				Algoritmo CFRS1				Algoritmo CFRS2			
		1	2	3	Média	1	2	3	Média	1	2	3	Média	1	2	3	Média
Classe I	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	11.8	10.3	10.7	10.9	0.7	0.8	0.9	**0.8**
	Número de Rotas	225.0	225.0	222.0	224.0	214.0	214.0	211.0	**213.0**	217.0	217.0	215.0	216.3	207.0	207.0	204.0	**206.0**
	Tempo de Percurso [min]	222.1	263.4	267.4	251.0	226.4	167.6	213.4	**202.5**	5.6	5.3	4.7	5.2	4.3	5.7	4.5	**4.8**
	Tempo de Execução [s]	129.016	129.156	129.860	**129.344**	148.250	147.469	151.296	149.005	891.813	1003.812	878.000	924.542	499.968	492.766	491.281	**494.672**
Classe II	Penalidade Média [min]	0.1	0.1	0.1	0.1	0.0	0.0	0.0	**0.0**	13.9	12.6	16.3	14.2	0.5	0.8	0.7	**0.7**
	Número de Rotas	167.0	169.0	170.0	168.7	159.0	160.0	161.0	**160.0**	167.0	168.0	169.0	168.0	159.0	160.0	161.0	**160.0**
	Tempo de Percurso [min]	936.2	908.6	947.8	930.9	806.5	852.4	808.7	**822.5**	1268.8	1237.2	1174.9	1227.0	1251.0	1219.5	1126.8	**1199.1**
	Tempo de Execução [s]	490.422	440.046	544.547	**491.672**	614.344	552.421	647.422	604.729	685.640	825.578	621.578	710.932	457.875	470.187	447.672	**458.578**
Classe III	Penalidade Média [min]	0.1	0.1	0.2	0.1	0.0	0.0	0.0	**0.0**	28.7	25.9	22.0	25.5	1.4	1.1	1.1	**1.2**
	Número de Rotas	113.0	116.0	118.0	115.7	107.0	110.0	111.0	**109.3**	114.0	118.0	119.0	117.0	109.0	112.0	114.0	**111.7**
	Tempo de Percurso [min]	1378.4	1352.4	1317.9	1349.6	1262.2	1197.0	1204.2	**1221.1**	1879.2	1917.1	1946.8	1914.4	1744.0	1776.9	1859.2	**1793.3**
	Tempo de Execução [s]	456.000	480.110	495.297	**477.136**	575.985	554.687	531.609	554.094	467.906	464.234	576.641	502.927	288.812	340.468	331.609	**320.296**
Classe IV	Penalidade Média [min]	0.2	0.2	0.2	0.2	0.0	0.0	0.0	**0.0**	76.7	43.5	61.2	60.5	9.7	11.0	10.8	**10.5**
	Número de Rotas	63.0	64.0	63.0	63.3	59.0	60.0	60.0	**59.7**	63.0	65.0	65.0	64.3	60.0	61.0	62.0	**61.0**
	Tempo de Percurso [min]	2265.6	2198.1	1984.9	2149.6	2029.7	2077.3	1977.0	**2028.0**	2287.1	2228.5	2345.3	2287.0	2030.3	2052.8	2097.6	**2060.2**
	Tempo de Execução [s]	569.062	530.296	515.187	**538.182**	650.812	593.922	534.766	593.167	220.109	245.359	244.515	236.661	103.469	106.406	110.375	**106.750**

2. Análise Comparativa dos Resultados

A partir da Tabela 4 até a Tabela 15, foi possível realizar uma contagem do número de vitórias obtidas por cada algoritmo a partir de uma comparação geral (envolvendo todos os algoritmos) e específica (somente algoritmos do mesmo tipo, CFRS ou RFCS), segundo cada critério de qualidade: penalidade média, número de rotas, tempo de percurso e tempo computacional. A Tabela 16 e Figura 26 revelam o resultado dessa contagem, apresentando a quantidade de vitórias (gerais ou específicas) atingidas por cada algoritmo, considerando cada critério de qualidade da solução.

No que diz respeito à comparação geral, pode-se perceber que o algoritmo RFCS2 foi o melhor sucedido (obteve o maior número de vitórias) nos critérios de Penalidade Média, Número de Rotas e Tempo de Percurso, só perdendo para RFCS1 no critério de tempo computacional. Era esperado que RFCS1 vencesse RFCS2 nesse quesito, devido à rotina de busca local presente no segundo algoritmo. No entanto, a diferença de tempo computacional entre os dois algoritmos é relativamente pequena, inferior a dois minutos mesmo nos piores casos. O algoritmo CFRS2 não foi vencedor em nenhum critério de qualidade, no entanto atingiu uma boa quantidade de vitórias no tocante ao número de rotas e tempo de percurso, ficando em segundo lugar nesses dois critérios. O CFRS1 teve um número de vitórias reduzido, somente 10 de 4×144 possíveis, mostrando-se relativamente pouco eficiente à sua finalidade.

Com relação às comparações específicas, o algoritmo RFCS2 venceu RFCS1 em todos os critérios de qualidade, exceto o tempo computacional, que consiste no critério de menor hierarquia dada à importância dos demais critérios. O algoritmo CFRS2 obteve um desempenho muito superior ao seu similar CFRS1, com exceção do tempo de percurso, onde o número de vitórias foi similar nos dois algoritmos (respectivamente 24 e 27). Foi possível reduzir o tempo computacional de CFRS2 devido à remoção da rotina de cálculo dos caminhos mínimos, necessária em CFRS1 devido à seleção das sementes ocorrer pelo critério de máximas distâncias.

Tabela 16. Contagem do número de vitórias (com ou sem empate) na comparação geral e específica.

Número de Vitórias (com ou sem empate)	Comparação Geral				Comparação Específica			
	RFCS1	RFCS2	CFRS1	CRFS2	RFCS1	RFCS2	CFRS1	CRFS2
Penalidade Média	5	39	0	4	9	39	6	42
Número de Rotas	6	37	2	26	6	48	5	48
Tempo de Percurso	4	36	8	9	10	41	27	24
Tempo Computacional	34	4	0	12	46	4	4	45

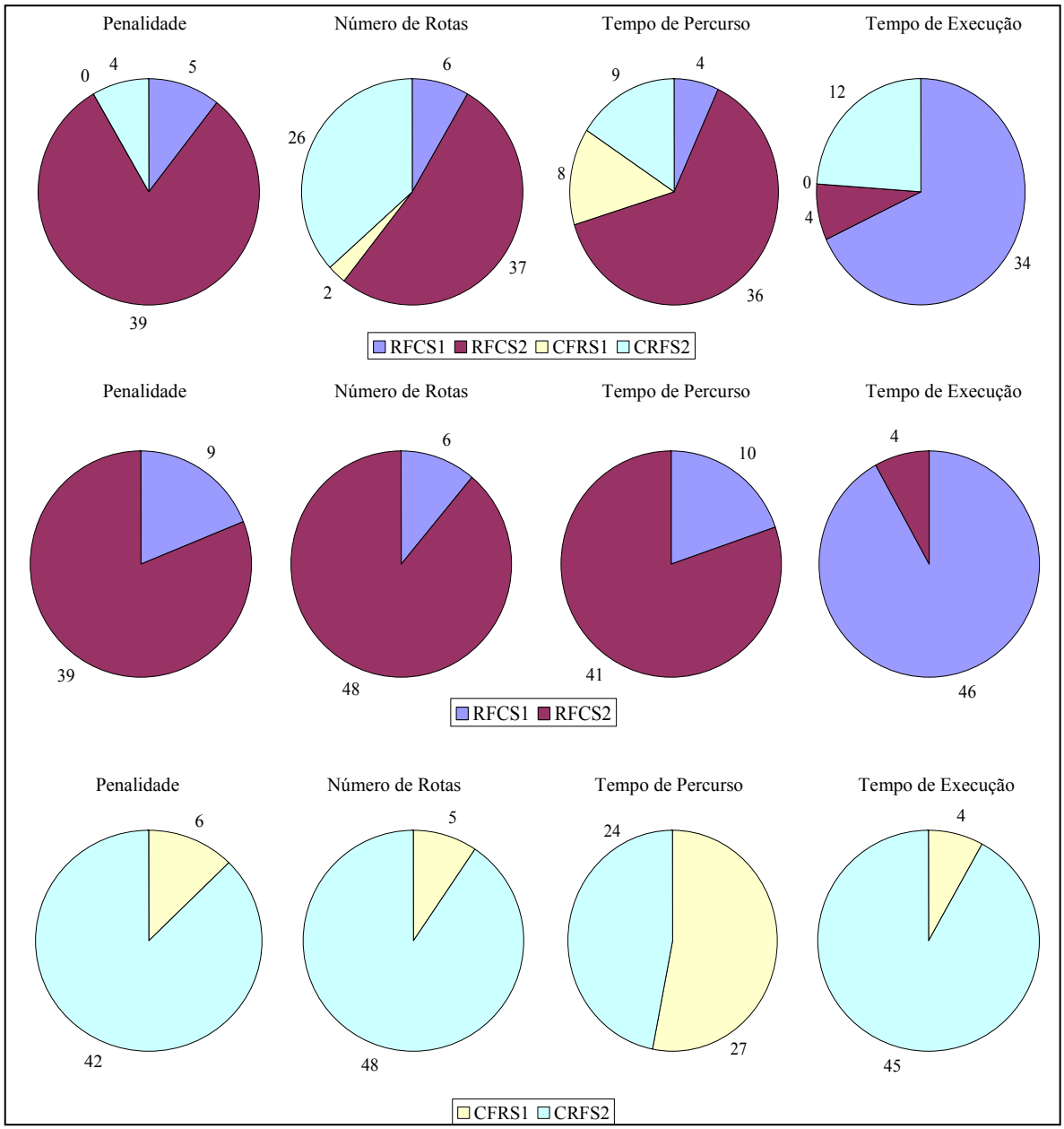


Figura 26. Contagem do número de vitórias (com ou sem empate) na comparação geral e específica.

3. Sensibilidade à Entrada

Foi possível observar uma sensibilidade à entrada do algoritmo CFRS2 não observada tão claramente em RFCS2. A Figura 27 à Figura 29 ilustram a relação do nível de penalidade da solução de CFRS2 com as diferentes combinações Grupo-Família-Classe das instâncias. O aspecto mais marcante consiste na deterioração das soluções de CFRS2 para instâncias de maior classe (% de arestas não-requeridas). Isso decorre da dificuldade de formar boas partições devido ao elevado número de arestas não-requeridas, pois torna-se necessário utilizá-las para conectar as arestas requeridas na rota.

Outra análise possível consiste na relação entre o Grupo (grandeza) e a Família (número de componentes) da instância e seus efeitos sobre a penalidade da solução. Nota-se que quanto maior o grafo, menor a sensibilidade de CFRS2 à variação do número de componentes conexos, ou seja, instâncias maiores são menos afetadas pelo número de componentes conexos.

Esses comportamentos foram observados de modo mais intenso no algoritmo CFRS1, observando-se grande sensibilidade à entrada, o que pode ser observado da Tabela 4 a Tabela 15. Isso sugere que as modificações implementadas em CFRS2 atenuaram a sensibilidade de CFRS2 à diversidade presente no conjunto de instâncias.

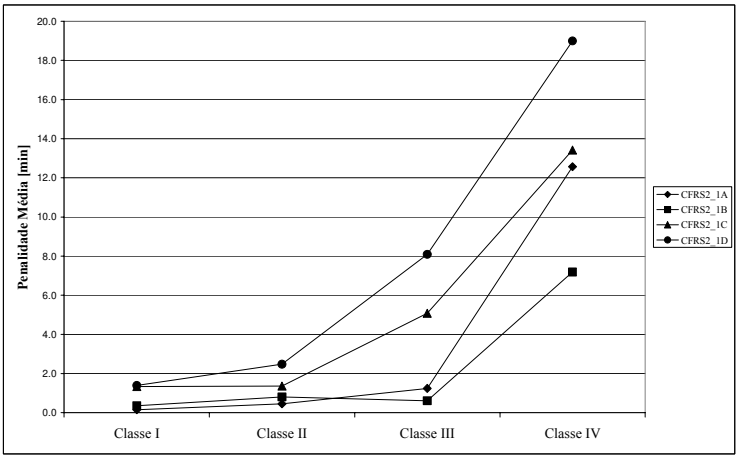


Figura 27. Sensibilidade do algoritmo CFRS2 às instâncias do Grupo 1.

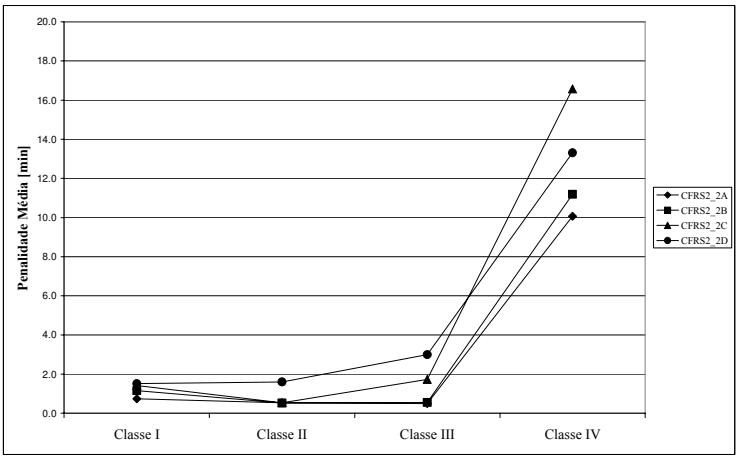


Figura 28. Sensibilidade do algoritmo CFRS2 às instâncias do Grupo 2.

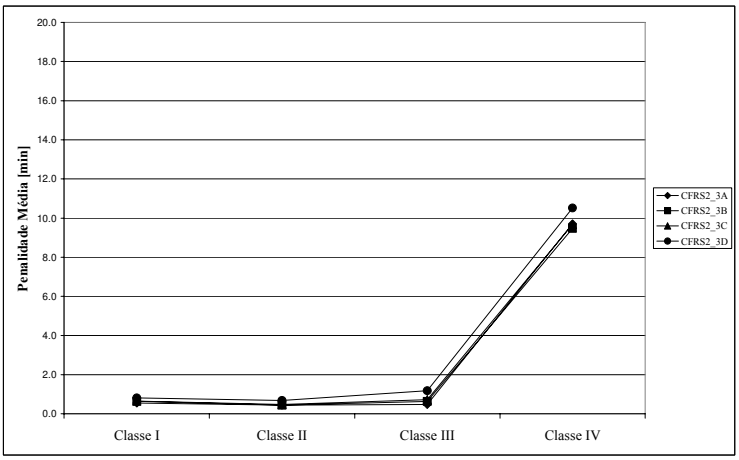


Figura 29. Sensibilidade do algoritmo CFRS2 às instâncias do Grupo 3.

4. Busca Local de Emparelhamento de Nós

A diferença mais significativa entre os algoritmos RFCS1 e RFCS2 consta da presença, no segundo algoritmo, da busca local de emparelhamento de nós. Essa rotina, explicada em III.5.9, troca os caminhos que ligam dois pares de nós ímpares, visando à redução do tempo de percurso do grafo. Uma visualização desse processo pode ser observada na Figura 30, onde se apresenta uma solução inicial de emparelhamento, em uma instância do Grupo 1, a partir do algoritmo guloso, seguido de 5 iterações da busca local. O efeito de redução do tempo de percurso afeta indiretamente o número de partições, pois dependendo da grandeza dessa redução, eleva-se a possibilidade de reduzir o número de partições necessário para atender todo o grafo. Da Figura 31 a Figura 33 fica clara a redução do tempo de percurso entre RFCS1 e RFCS2. Em 40 das 48 combinações Grupo-Família-Classe, o algoritmo RFCS2 atingiu tempos de percurso inferiores ao RFCS1. Se a diferença entre esses dois algoritmos se devesse unicamente à busca local, certamente RFCS2 seria vitorioso nas 48 combinações, empatando no pior caso. No entanto, outras modificações em RFCS2, priorizando a redução da penalidade, afetaram negativamente o tempo de percurso em algumas instâncias.

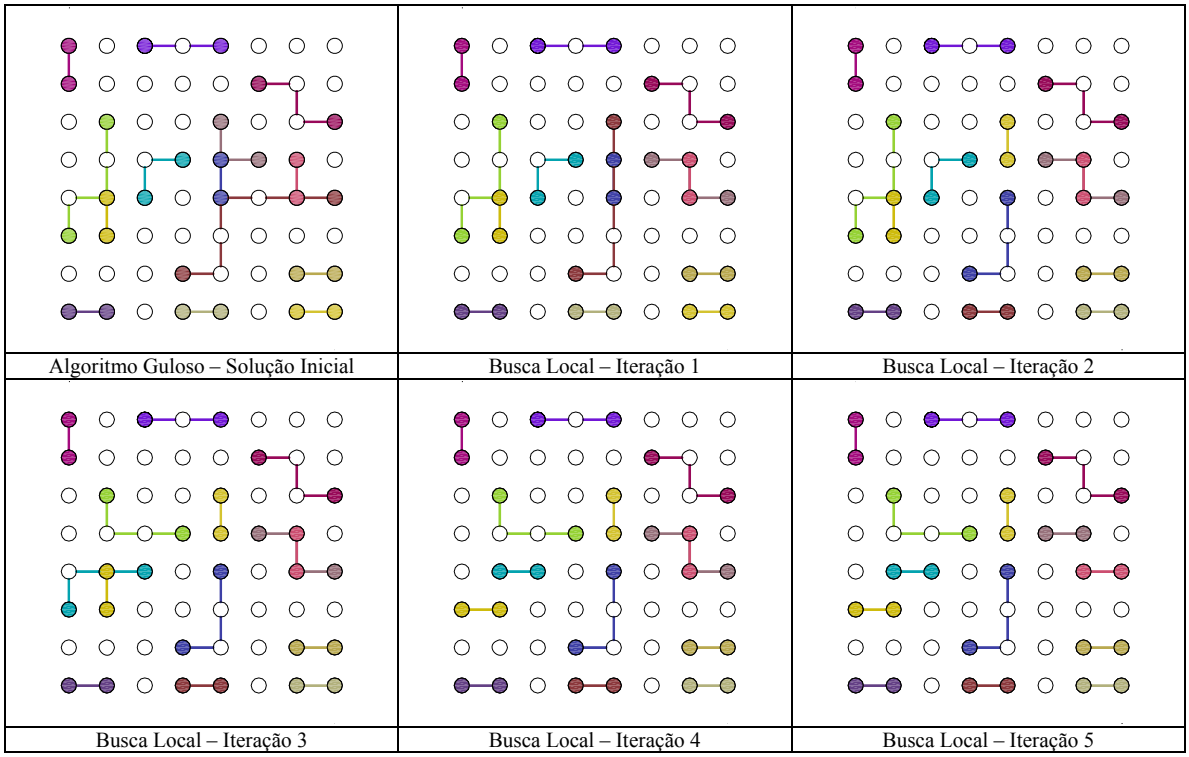


Figura 30. Visualização do princípio de funcionamento da busca local de emparelhamento.

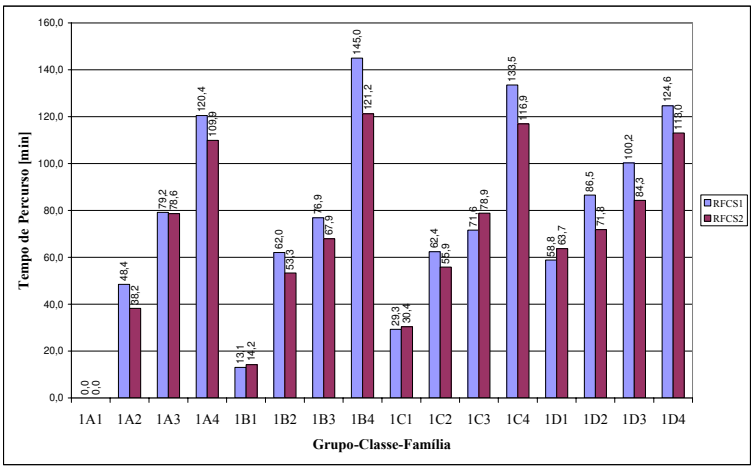


Figura 31. Comparação entre RFCS1 e RFCS2 quanto ao tempo de percurso das instâncias do Grupo 1.

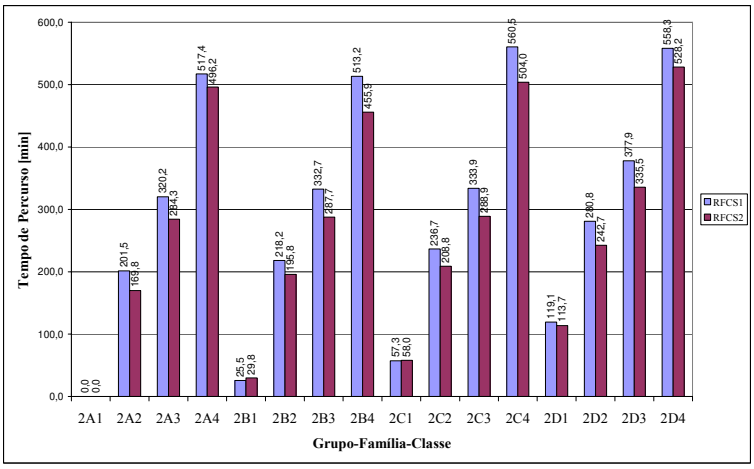


Figura 32. Comparação entre RFCS1 e RFCS2 quanto ao tempo de percurso das instâncias do Grupo 2.

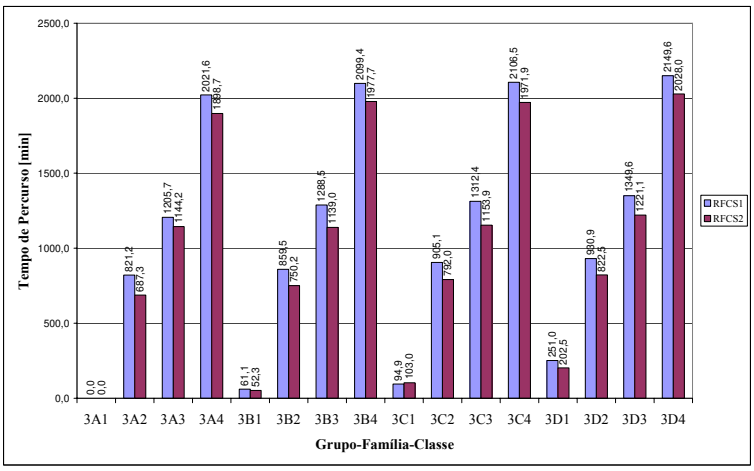


Figura 33. Comparação entre RFCS1 e RFCS2 quanto ao tempo de percurso das instâncias do Grupo 3.

5. Seleção dos Nós Sementes

Uma das etapas dos algoritmos CFRS1 e CFRS2 consiste na seleção dos nós sementes, a partir dos quais serão formadas as partições do grafo. A eficiência dessa etapa é importante para fornecer uma solução inicial de boa qualidade à etapa de balanceamento das partições. Quando os nós sementes se encontram pouco dispersos no grafo, torna-se mais difícil distribuir as arestas pelas partições visando maior homogeneidade no aspecto da carga horária necessária para realizar o serviço de leitura. Isso ficou comprovado com os resultados obtidos a partir dos dois métodos de seleção das sementes utilizados em CFRS1 e CFRS2. Enquanto no primeiro algoritmo, o critério de seleção se baseia na distância de uma semente com relação às demais, o segundo algoritmo distribui as sementes de maneira uniforme e aleatória.

A Figura 34 revela como os algoritmos CFRS1 e CFRS2 dispersaram as sementes nos grafos de duas instâncias. Torna-se evidente que o algoritmo CFRS1 é pouco eficiente na dispersão dos nós, pois se percebe claramente uma polarização dos mesmos nas fronteiras do grafo, podendo ser explicado pelo fato das regiões fronteiriças serem relativamente as mais distantes de quaisquer outras regiões. Por sua vez, o algoritmo CFRS2 foi mais bem sucedido nesse sentido, notando-se uma dispersão mais homogênea das sementes por todas as regiões do grafo.

O impacto da dispersão das sementes na qualidade da solução pode ser significativo, o que demonstra a Figura 34, onde as soluções finais dos algoritmos CFRS1 e CFRS2 para a instância 3D4(1) são apresentadas (valores dos critérios de qualidade se encontram na Tabela 15). Nota-se que o algoritmo CFRS1 concentrou as partições nas fronteiras, resultante da polarização dos nós nessa região. Isso resultou na presença de partições muito heterogêneas. Já o algoritmo CFRS2 conseguiu uma melhor distribuição e homogeneidade das partições ao longo do grafo. É importante esclarecer que essas soluções passaram pelo processo de balanceamento, que contribui efetivamente na melhoria da qualidade das duas soluções. Ainda assim é possível distinguir o efeito da dispersão dos nós sobre a solução.

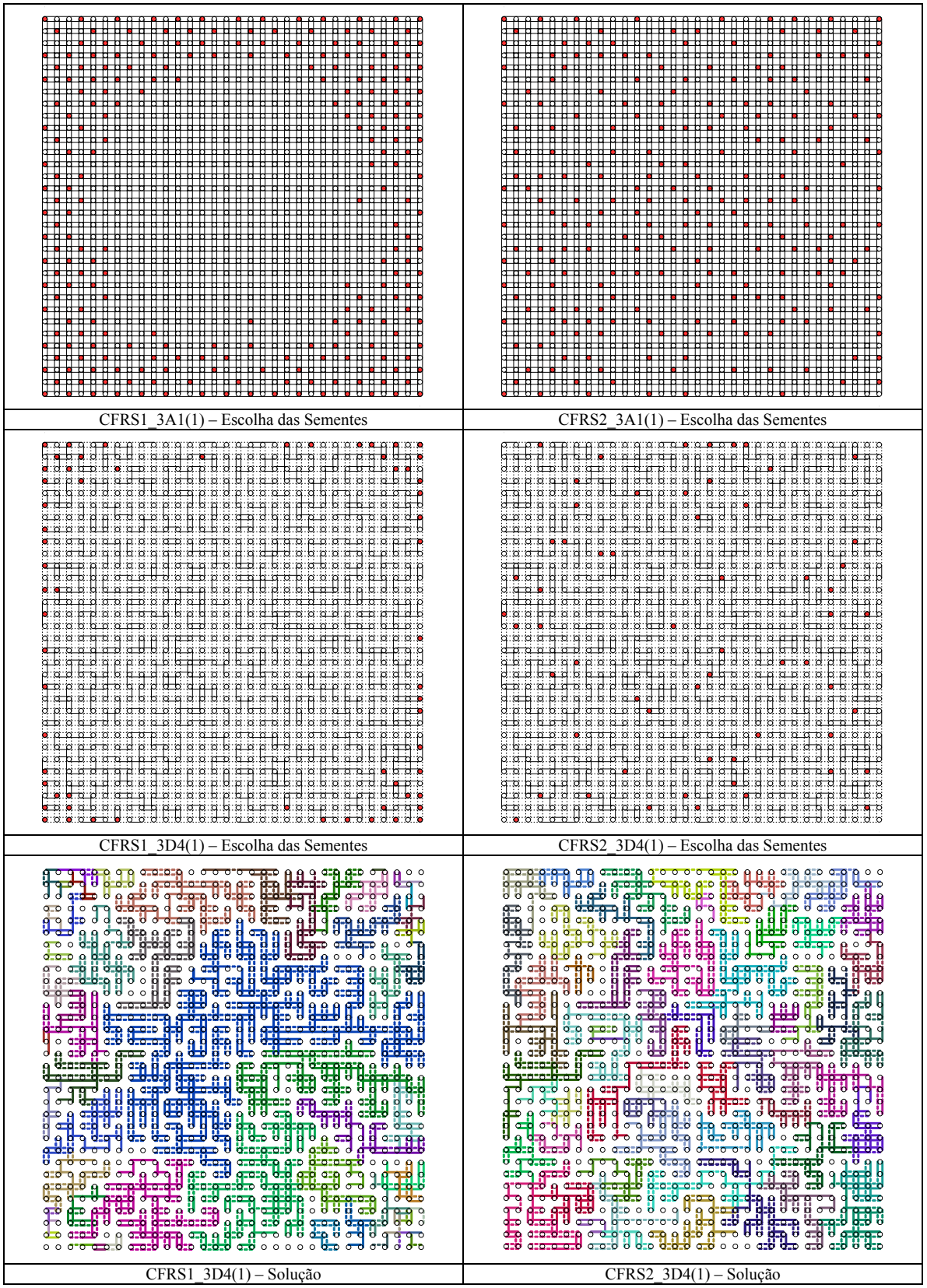


Figura 34. Resultado da escolha das sementes (nós em vermelho), seguido da solução obtida para instância 3D4(1) pelos algoritmos CFRS1 e CFRS2.

6. Balanceamento das Partições

A etapa mais importante dos algoritmos tipo CFRS consiste no balanceamento das partições. Essa busca local realiza realocações de arestas paralelas interpartições, visando à redução de penalidade, de modo que, no caso da distribuição inicial das arestas paralelas resultar em partições sem qualquer violação da carga horária, a etapa de balanceamento não se efetua. É possível implementar o balanceamento das partições visando outros objetivos que não a penalidade, por exemplo, redução do número de rotas. Essa estratégia, no entanto, deve garantir que a penalidade das partições não aumente, caso contrário as arestas seriam realocadas a ponto de formar uma única partição de elevada penalidade. Neste trabalho utilizou-se somente o critério de penalidade para o balanceamento, de modo que o número de partições, nos algoritmos CFRS1 e CFRS2, permanece constante.

A Figura 35 ilustra o princípio de funcionamento da rotina de balanceamento de partições aplicada à instância 1D2(1), utilizando o algoritmo CFRS2. Nessa figura, encontram-se destacadas as arestas paralelas que são realocadas de uma partição para outra. Além disso, também se menciona quando a realocação é de folha ou cíclica (vide II.5.2.d), as realocações de folha ocorrem quando as arestas paralelas realocadas se encontram em uma extremidade da partição. Já as realocações cíclicas ocorrem quando as arestas paralelas realocadas, apesar de não se encontrarem em uma extremidade, estão contidas em um ciclo da partição. A natureza desses dois tipos de realocação garante que as partições do grafo permanecerão conexas ao longo do balanceamento.

Em cada uma das 27 iterações da Figura 35, a rotina procura pela realocação (de folha ou cíclica) que mais reduz a função de penalidade. Assim, pode-se afirmar que após a 27ª iteração não foi encontrada nenhuma solução na vizinhança cuja penalidade fosse menor que a solução atual.

Pode-se imaginar que a solução final apresentada na Figura 35 corresponda à solução final do algoritmo CFRS2 para a instância 1D2(1), no entanto isso não é verdade em virtude da ocorrência de repetições dessa rotina, baseado em diversas seleções de sementes (vide diagrama de CFRS2 na Figura 22).

Figura 35. Processo de balanceamento de partições (arestas paralelas realocadas em destaque).

Em essência, as rotinas de balanceamento são as mesmas para CFRS1 e CFRS2, mudando apenas a função de penalidade, que coordena a prioridade das arestas paralelas que sofrerão a realocação. Comentou-se anteriormente a adição de um expoente quadrático na função de penalidade de CFRS2 (vide III.2), cujo objetivo consiste em elevar a penalidade nas situações de heterogeneidade entre partições, ou seja, grandes partições em detrimento de pequenas partições.

Pode-se perceber o efeito dessa nova função de penalidade sobre situações onde as partições são heterogêneas na Figura 36, que realça a diferença existente entre as funções de penalidade de CFRS1 (PEN1) e CFRS2 (PEN2). No eixo das abscissas observa-se a carga horária (em minutos) de duas partições distintas. São adotados os seguintes valores paramétricos: tolerância superior de carga horária W_{SUP} = 300 minutos, $\alpha = 1$, $\beta = \alpha / 4$. Procurou-se exemplificar as situações onde a soma das violações superiores das duas partições fossem sempre iguais a 200 minutos. Isso provocou um mapeamento constante da função de penalidade PEN1, que consiste na soma ponderada das violações das partições (vide seção II.5.2.d). A função de penalidade PEN2, por sua vez, produz um mapeamento quadrático para o mesmo domínio de violações. Percebe-se que o valor mínimo da função de penalidade PEN2 ocorre na situação de maior homogeneidade entre as duas partições, pois possuem as mesmas cargas horárias e, portanto, as mesmas violações.

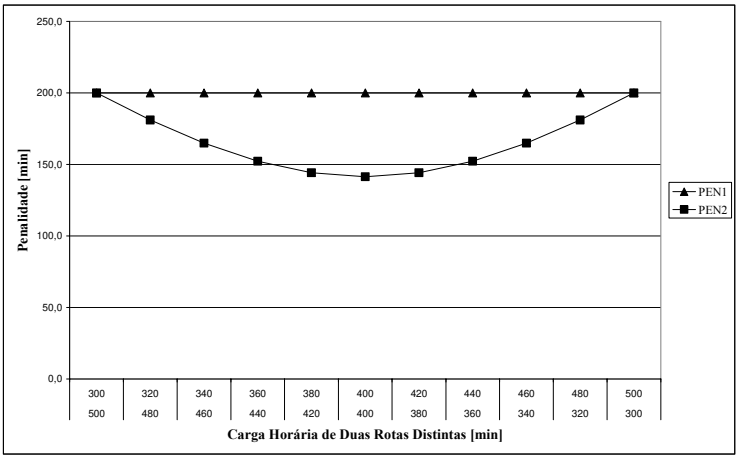


Figura 36. Relação entre a penalidade e homogeneidade interpartições (W_{SUP} = 300 min; $\alpha = 1$).

A função de penalidade PEN2 favorece à realocação de arestas paralelas entre partições que violam, ambas, a tolerância de carga horária (W_{INF} ou W_{SUP}), mas que diferem no grau de violação, possibilitando desse modo a redução dessa heterogeneidade. Em outras

palavras, quando não é mais possível reduzir a violação absoluta das partições, deve-se procurar torná-las menos discrepantes entre si, de modo que o leiturista de uma partição não fique demasiadamente sobrecarregado. A função PEN1 impossibilita esse processo devido à não diferenciação entre partições heterogêneas quanto à violação da carga horária.

7. Qualidade Visual das Soluções

Um critério não levado em consideração no processo de classificação das melhores soluções consistiu no aspecto visual. O motivo de não levar esse critério em consideração se deve à dificuldade de automatizar sua análise, pois se torna inviável classificar visualmente as soluções uma a uma. No entanto, esse critério não deixa de ser relevante, sobretudo no convencimento do pessoal técnico, responsável pelo desenvolvimento e manutenção de rotas de leituristas, de que as soluções são boas.

Notou-se claramente que os algoritmos do tipo RFCS, apesar de serem superiores nos demais critérios de qualidade, deixam a desejar no aspecto visual com relação aos algoritmos CFRS, como pode ser observado na Figura 37, onde são exibidas as saídas gráficas das instâncias 3A4(1), 3C4(2) e 3D4(3) geradas pelos algoritmos CFRS2 e RFCS2.

Pode-se perceber que o algoritmo CFRS2 setorializa o grafo original, de modo que as partições formadas tornam-se compactas e de contorno definido, facilitando muito a interpretação e distinção das áreas de trabalho de cada leiturista. Já as partições formadas pelo algoritmo RFCS2 são bem mais esparsas e de difícil contorno devido ao forte entrelaçamento. Em termos práticos, os leituristas, utilizando as soluções de CFRS2, seriam designados cada qual a um setor ou região e realizariam o serviço de leitura restritos a essa região designada, sem necessidade de transitar entre regiões muito distantes. No caso das soluções provenientes de RFCS2, os leituristas podem visitar pontos distantes entre si, inclusive invadindo as regiões de trabalho de outros leituristas, de maneira bastante enovelada.

Essa diferença se explica pelo processo de formação das partições em CFRS2, que parte de um nó semente e aos poucos vai agregando as arestas paralelas vizinhas. Esse processo favorece a formação de partições compactas. O algoritmo RFCS2, do modo como foi implementado, não se preocupa com o formato das partições, mas somente com a redução do tempo necessário para percorrer todas as arestas requeridas, indiferentemente da ordem na qual elas são visitadas.

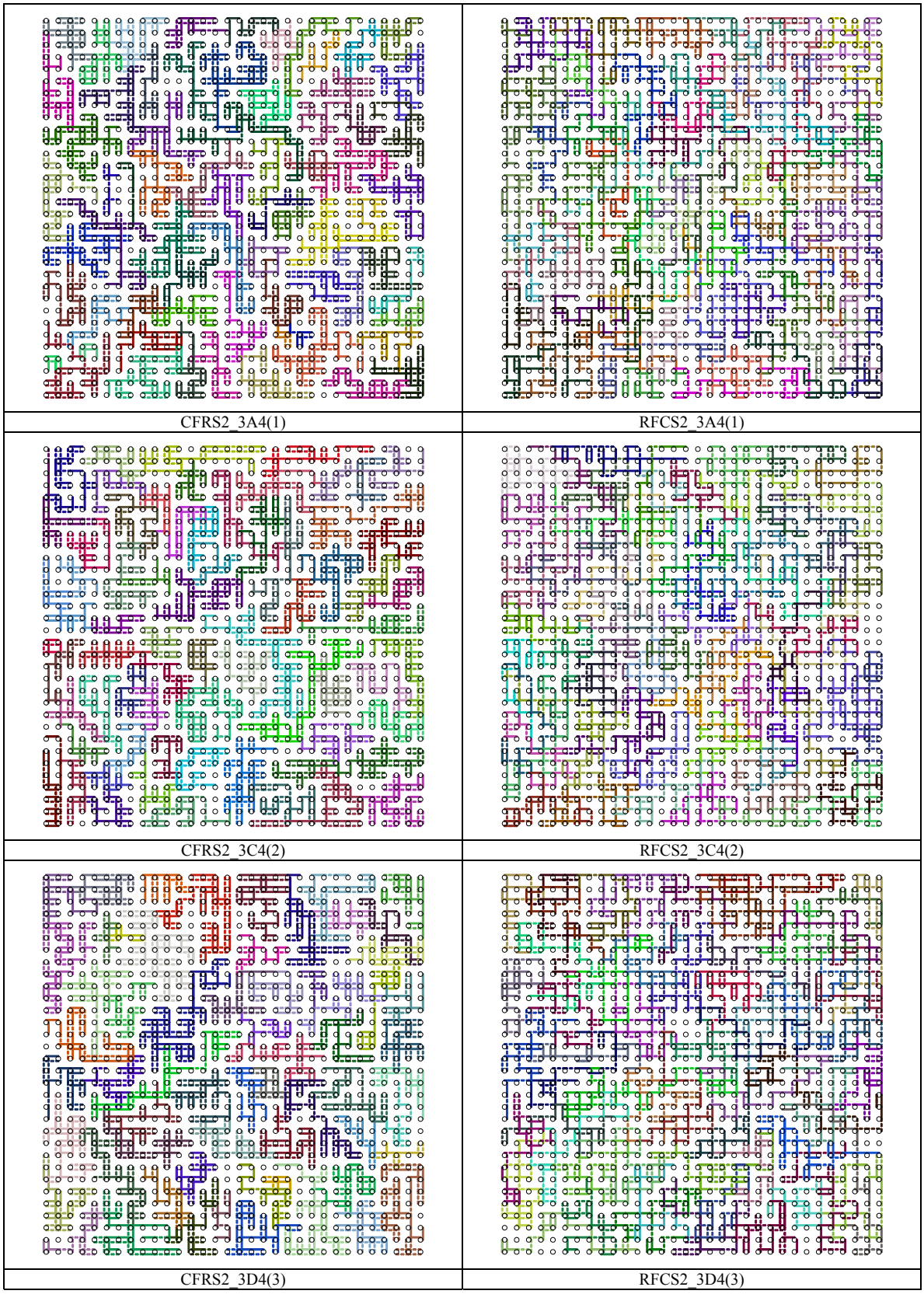


Figura 37. Comparação do aspecto visual entre soluções de CFRS2 e RFCS2.

V. CONCLUSÕES

Esse trabalho teve por objetivo fornecer estratégias computacionais otimizantes para o problema de roteamento de leituristas, onde se procura reduzir o número de rotas necessárias, o tempo de percurso e a violação da carga horária dos leituristas. Considerando que as soluções atualmente colocada em prática são normalmente elaboradas à partir de métodos empíricos, baseados na experiência de especialistas humanos, o ganho que se pode atingir aplicando métodos computacionais ao problema pode ser bastante significativo. Dois algoritmos (RFCS1 e CFRS1) da literatura ([47],[52]) foram testados, de modo que foi possível conhecer aspectos passíveis de melhoria. Nesse sentido, dois novos algoritmos (RFCS2 e CFRS2) baseados nos anteriores, são propostas desse trabalho, a partir dos quais foram realizados testes comparativos.

O algoritmo para roteamento de leituristas de maior destaque em termos de qualidade de solução, com base nos critérios de penalidade, número de rotas, tempo de percurso e tempo computacional, foi o RFCS2. Foram 116 vitórias gerais de um universo de 4x48 combinações Grupo-Família-Classe, o que representa 60,42%.

A inserção da busca local de emparelhamento de nós contribuiu para o bom desempenho de RFCS2. Isso ficou claro na comparação específica, onde RFCS2 atingiu 132 vitórias ou 68,75% do universo. A soma dos tempos de percurso para todas as soluções de RFCS1 e RFCS2 foram respectivamente 23.321,9 [min] contra 21.242,0 [min], o que representa uma redução de aproximadamente 8,92% ou quase 35 [h] economizadas.

O algoritmo CFRS2 mostrou-se superior ao seu antecessor, obtendo 159 vitórias na comparação específica (82,81%). Esse resultado foi possível com pequenas intervenções em CFRS1 e sugere que novas modificações podem levar a resultados ainda melhores.

A seleção aleatória das sementes em RFCS2 mostrou-se eficaz na dispersão desses nós, promovendo a formação de partições mais homogêneas. Verificou-se que essa etapa do algoritmo é fundamental na produção de uma boa solução inicial para que o processo posterior de balanceamento de partições tenha êxito. Desse modo, futuras implementações dessa heurística devem levar em consideração melhorias na seleção das sementes e como exemplo, sugere-se aplicar algoritmos para o problema das p-medianas capacitado.

A introdução de um expoente quadrático na função de penalidade colaborou no processo de homogeneização das soluções de RFCS2, decorrente da punição mais severa às soluções que violaram a carga horária do leiturista de modo heterogêneo.

Enquanto os tempos de processamento de RFCS2 foram superiores ao de RFCS1, devido à presença da rotina de busca local de emparelhamento de nós, CFRS2, ao contrário, apresentou uma importante redução no tempo de processamento devido à remoção da rotina de caminhos mínimos, antes utilizada em CFRS1.

Os algoritmos CFRS perdem para os algoritmos RFCS considerando os critérios de qualidade utilizados. No entanto, foi possível constatar que a qualidade visual das soluções fornecidas pelos primeiros são superiores, apresentando rotas compactas e de contornos bem definidos.

Torna-se fundamental, como sugestão de futuras pesquisas, elaborar um modelo matemático que descreva claramente o problema, facilitando a proposição de novas metodologias de resolução, incluindo estratégias que levem à otimalidade quando possível. Verificou-se que tal modelo ainda se encontra inexistente na literatura devido às particularidades inerentes do problema original como: ausência de um ponto central por onde partem os leituristas; não obrigatoriedade das rotas serem fechadas; tolerâncias na carga horária do leiturista; introdução de uma função de penalidade na ocasião da violação das tolerâncias de carga horária.

A dificuldade de tratar o problema do carteiro rural capacitado até a otimalidade sugere que o problema de roteamento de leituristas, similar ao primeiro, é também de difícil tratamento. Nesse sentido, abordagens metaheurísticas como Busca Tabu, GRASP, Simulated Annealing, Colônias de Formigas, Algoritmo Genético, dentre outras, são apontadas como meios alternativos, bem sucedidos em diversos problemas de otimização combinatória, para tratar do problema de roteamento de leituristas.

Os algoritmos estudados e implementados nesse trabalho, especialmente o RFCS2, que mostrou o melhor desempenho relativo nos critérios de qualidade de solução, podem ser utilizados pelas empresas que realizam a leitura de consumo dos clientes, visando à redução do número necessário de rotas de leitura, tempo de percurso total e violação da carga horária dos leituristas.

VI. REFERÊNCIAS BIBLIOGRÁFICAS

1. Avis, D. A survey of heuristics for the weighted matching problem. **Networks**, 13 (4): 475-493, 1983.
2. Benavent, E., Carrotta A., Corberan A., Sanchis, J. M., Vigo, D. Lower bounds and heuristics for the windy rural postman problem. **European Journal of Operational Research**, 176 (2): 855-869, 2007.
3. Benavent, E., Corberan A., Piñana, E., Plana I., Sanchis, J. M. New heuristic algorithms for the windy rural postman problem. **Computers and Operations Research**, 32 (12): 3111-3128, 2005.
4. Berge, C. Two theorems in graph theory. **Proceedings of the National Academy of Sciences of the United States of America**, 43: 842-844, 1957.
5. Bodin, L., Levy, L. The arc oriented location routing problem. **INFOR**, 27 (1): 74-94, 1989.
6. Bodin, L., Levy, L. The arc partitioning problem. **European Journal of Operational Research**, 53 (3): 393-401, 1991.
7. Christofides, N., Campos, V., Corberan, A., Mota, E. An algorithm for the rural postman problem on a directed graph. **Mathematical Programming Study**, 26: 155-166, 1986.
8. Cook, S. A. The complexity of theorem-proving procedures. **Proc. 3rd Ann. ACM Symp. On Theory of Computing**, Association for Computing Machinery, New York, 151-158 (1971).
9. Corberan A., Marti, R., Romero, A. Heuristics for the mixed rural postman. **Computers and Operations Research**, 27 (2): 183-203, 2000.
10. Corberan, A., Marti, R., Sanchis, J. M. A GRASP heuristic for the mixed Chinese postman problem. **European Journal of Operational Research**, 142 (1): 70-80, 2002.
11. Corberan, A., Mota, E., Sanchis, J. M. A comparison of two different formulations for arc routing problems on mixed graphs. **Computers and Operations Research**, 33 (12): 3384-3402, 2006.

12. Djikstra, E. W. A note on two problems in connection with graphs. **Numerische Mathematik**, 1: 269-271, 1959.
13. Edmonds, J. Maximum matching and a polyhedron with 0,1-vertices. **Journal of Research of the National Bureau of Standards – B. Mathematics and Mathematical Physics,** 69B (1-2): 125-130, 1965.
14. Edmonds, J., Johnson, E. L. Matching, Euler tours and the Chinese postman. **Mathematical Programming,** 5: 88-124, 1973.
15. Eiselt, H. A., Gendreau, M., Laporte, G. Arc Routing Problems, Part I: The Chinese postman problem. **Operations Research**, 43 (2): 231-242, 1995.
16. Eiselt, H. A., Gendreau, M., Laporte, G. Arc Routing Problems, Part II: The rural postman problem. **Operations Research,** 43 (3): 399-414, 1995.
17. Euler, L. Solutio problematic ad geometriam situs pertinentis. **Comment. Academiae Sci. I. Petropolitanae**, 8: 128-40, 1736.
18. Floyd, R. W. Algorithm 97: shortest path. **Communications of the ACM**, 5 (6): 345, 1962.
19. Frederickson, G. N. Approximation algorithms for some postman problems. **Journal of the Association for Computing Machinery**, 26 (3): 538-554, 1979.
20. Frederickson, G. N., Hetch, M. S., Kim, C. E. Approximation algorithms for some routing problems. **SIAM Journal on Computing**, 7 (2): 178-193, 1978.
21. Gabow, H. N. An efficient implementation of Edmond's algorithm for maximum matching on graphs. **Journal of the Association for Computing Machinery**, 23 (2): 221-234, 1976.
22. Gabow, H. N. Data structures for weighted matching and nearest common ancestors with linking. **Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms**, ACM, 434-443, 1990.
23. Garey, M. R., Johnson D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W. H. Freeman, 1979.
24. Golden, B. L., Wong, R. T. Capacitated arc routing problems. **Networks**, 11 (3): 305-315, 1981.
25. Gondran, M., Minoux M. **Graphs and Algorithms**. John Wiley & Sons, 1984.

26. Greistorfer, P. A tabu scatter search metaheuristic for the arc routing problem. **Computers and Industrial Engineering**, 44 (2): 249-266, 2003.
27. Gribkovskaia, I., Halskau, ∅., Laporte, G. The bridges of Königsberg – A historical perspective. **Networks**, 49 (3): 199-203, 2007.
28. Graham, R., Hell, P. On the history of the minimum spanning tree problem. **Annals of the History of Computing**, 7: 43-57, 1985.
29. Hu, T. C. Revised matrix algorithms for shortest paths. **SIAM Journal on Applied Mathematics**, 15 (1): 207-218, 1967.
30. Klee, V. & Minty, G.J. How good is the simplex algorithm? In O. Sshisha (ed.), **Inequalities III**, Academic Press, New York, 159-175 (1972).
31. Kruskal, J. B. On the shortest spanning subtree and the traveling salesman problem. **Proceedings of the American Mathematical Society**, 7: 48-50, 1956.
32. Lawler, E. L. Lenstra, J. K., Rinnooy Kan A. H. G., Shmoys D. B. **The Traveling Salesman Problem**. Wiley & Sons, New York, 1985.
33. Mei-Ko, K. Graphic programming using odd or even points. **Chinese Mathematics**, 1: 273-277, 1962.
34. Micali, S., Vazirani, V. V. An O($\sqrt{|v|}\,|E|$) Algorithm for finding maximum matching in general graphs. **Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science**, IEEE, New York, 17-27, 1980.
35. Minieka, E. The Chinese postman problem for mixed networks. **Management Science**, 25 (7): 643-648, 1979.
36. Papadimitriou, C. H. On The complexity of edge traversing. **Journal of the Association for Computing Machinery**, 23 (3): 544-554, 1976,.
37. Pearn, W. L. Approximate solutions for the capacitated arc routing problem. **Computers and Operations Research**, 16 (6): 589-600, 1989.
38. Pearn, W. L. Augment-insert algorithms for the capacitated arc routing problem. **Computers and Operations Research**, 18 (2): 189-198, 1991.
39. Pearn, W. L., Chou. J.B. Improved solutions for the Chinese postman problem on mixed networks. **Computers and Operations Research**, 26 (8): 819-827, 1999.
40. Pearn, W. L., Liu, C. M. Algorithms for the Chinese postman problem on mixed networks. **Computers and Operations Research**, 22 (5): 479-489, 1995.

41. Pearn, W. L., Wu, T. C. Algorithms for the rural postman problem. **Computers and Operations Research**, 22 (8): 819-828, 1995.
42. Pettie, S., Sanders P. A simpler linear time 2/3 – ε approximation for maximum weight matching. **Information Processing Letters**, 91 (6): 271-276, 2004.
43. Prim, R. C. Shortest connection networks and some generalizations. **Bell System Technical Journal**, 36: 1389-1401, 1957.
44. Raghavachari, B., Veerasamy J. A 3/2-approximation algorithm for the mixed postman problem. **SIAM Journal on Discrete Mathematics**, 12 (4): 425-433, 1999.
45. Solomon, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. **Operations Research Society of America**, 35 (2): 254-265, 1987.
46. Sang-Ho, K., Young-Gun G., Kyu-Maing, K. Application of the out-of-kilter algorithm to the asymmetric traveling salesman problem. **Journal of the Operational Research Society**, 54 (10): 1085-1092, 2003.
47. Stern, H. I., Dror, M. Routing Electric Meter Readers. **Computers and Operations Research**, 6 (4): 209-223, 1979.
48. Swamy M. S. S., Thulasiraman K. **Graphs, Networks, and Algorithms**. John Wiley & Sons, 3-71, 1981.
49. Tucker, A. **Applied Combinatorics**. John Wiley & Sons, 2nd ed., 1-162, 1984.
50. Drake, D. E., Hougardy, S. A linear-time approximation algorithm for weighted matching in graphs. **ACM Transactions on Algorithms**, 1 (1): 107-122, 2005.
51. Wilson, R.J. **Introduction to Graph Theory**. Longman Scientific & Technical, 3rd ed., 1-58, 1985.
52. Wunderlich, J., Collette M., Levy, L. Bodin, L. Scheduling meter readers for Southern California gas company. **Interfaces**, 22 (3): 22-30, 1992.

VII. APENDICE I: COMPLEXIDADE COMPUTACIONAL

Nesse capítulo será abordada a função de complexidade computacional de um algoritmo, que expressa, para cada tamanho de entrada possível, o tempo necessário no pior caso para resolver instâncias de um problema computacional. De certo, essa função não é bem definida enquanto não for fixada a estrutura de dados, o esquema da implementação do algoritmo e o computador que foi utilizado. No entanto, escolhas particulares como essa terá pouco efeito nas distinções mais amplas feitas na teoria da NP-completude, também abordada nesse capítulo. Para maiores informações, sugere-se a referência [23], que abrange esses tópicos em profundidade.

1. Notação Assintótica

A ordem de crescimento do tempo de execução de um algoritmo fornece uma caracterização simples da eficiência do algoritmo e também permite comparar o desempenho relativo de algoritmos alternativos.

Quando se observa tamanhos de entrada grandes o suficiente para tornar relevante apenas a ordem de crescimento do tempo de execução, está se estudando a eficiência assintótica dos algoritmos. Em geral, um algoritmo que é assintoticamente mais eficiente será a melhor escolha para todas as entradas, exceto as menores.

A notação assintótica possui duas áreas principais de aplicação: na Matemática, onde é utilizada para caracterizar um termo residual de uma série infinita truncada; e na Ciência da Computação, onde é utilizada para análise de complexidade de algoritmos.

2. Definição da notação $O(g(x))$

Suponha f(x) e g(x) duas funções definidas em algum subconjunto dos números reais positivos. A notação assintótica $O(g(x))$ pode ser definida como a seguir.

$$f(x) \in O(g(x)) \Leftrightarrow \exists\ x_0, \exists\ c > 0 : x > x_0 \Rightarrow f(x) \leq cg(x)$$

Nesse sentido, quando $f(x) \in O(g(x))$, a partir de um determinado valor de x a função $f(x)$ (que pode representar o tempo computacional de pior caso de um algoritmo, por exemplo) é limitada superiormente pela função $g(x)$ multiplicada por uma constante positiva.

3. Problemas, Algoritmos e Complexidade

Define-se problema como uma questão geral a ser respondida, usualmente possuindo diversos parâmetros, ou variáveis livres, cujos valores não estão instanciados. Um problema é descrito pelo fornecimento de: (1) uma descrição geral sobre todos seus parâmetros e (2) uma declaração das propriedades da resposta ou solução, que devem ser satisfeitas. Uma instância de um problema é obtida especificando valores particulares para todos os parâmetros do problema. Como exemplo, pode-se considerar o problema do caixeiro viajante. Os parâmetros desse problema consistem em um conjunto finito $C = \{c_1,c_2,...,c_m\}$ de cidades e, para cada par de cidades c_i, c_j em C, a distância $d(c_i,c_j)$ entre elas. A solução é um subconjunto ordenado $<c(1),c(2),...,c(\mathrm{m})>$ das cidades que minimizam

$$Min\ \left[D = d\left(c(1),c(2)\right) + d\left(c(2),c(3)\right) + \ldots + d\left(c\left(m-1\right),c\left(m\right)\right) + d\left(c\left(m\right),c(1)\right)\right]$$

Essa expressão fornece a distância da rota que se inicia em $c(1)$, visita todas as cidades em seqüência, e retorna diretamente a $c(1)$ após a última cidade $c(m)$.

Algoritmos são procedimentos passo a passo bem definidos, utilizados para resolver problemas. É dito que um algoritmo resolve um problema PI se ele pode ser aplicado a qualquer instância I de PI e sempre produzir uma solução.

Em geral, há interesse em encontrar o algoritmo mais "eficiente" para resolver determinado problema. De modo geral, a eficiência de um algoritmo se relaciona com os recursos computacionais utilizados para executar o algoritmo. No entanto, considera-se um algoritmo como o mais eficiente quando ele é mais rápido na obtenção da solução de um mesmo problema, quando comparado a outros algoritmos.

A função de complexidade temporal de um algoritmo expressa o tempo necessário, no pior caso, para resolver uma instância de um determinado problema.

4. Algoritmos de Tempo Polinomial e Problemas Intratáveis

Diferentes algoritmos possuem uma variedade muito grande de funções de complexidade computacional e a caracterização de quais delas são "boas o suficiente" e quais são "muito ineficientes" sempre vai depender da situação. No entanto, cientistas de computação reconhecem uma simples distinção que oferece considerável conhecimento dessa questão. Essa distinção se refere aos algoritmos de tempo polinomial e exponencial.

Um algoritmo de complexidade polinomial é definido como sendo um que possui função de complexidade de tempo seja $O(p(n))$ para alguma função polinomial p, onde n se refere ao tamanho da entrada. Qualquer algoritmo cuja função de complexidade de tempo não puder ser limitado dessa forma é denominado algoritmo de tempo exponencial (embora, funções não polinomiais, de complexidade $n\log(n)$, não sejam consideradas funções de complexidade exponencial).

A distinção entre esse dois tipos de algoritmos tem uma importância particular quando consideramos soluções de problemas de instâncias grandes. A Tabela 17 a seguir revela a diferença de crescimento das funções de complexidades distintas. É possível notar o crescimento explosivo das funções de complexidade exponencial (em itálico), comparadas às demais.

Tabela 17. Crescimento de funções polinomiais e exponenciais (em itálico).

	n			
f(n)	**10**	**20**	**40**	**80**
log n	1	1,30	1,60	1,90
$n^{1/2}$	5	10	20	40
n	10	20	40	80
n log n	10	26,02	64,08	152,25
n^2	100	400	1600	6400
2^n	*1024*	*1,05E+06*	*1,10E+12*	*1,21E+24*
n!	*3,63E+06*	*2,43E+18*	*8,16E+47*	*7,16E+118*

Essa tabela indica a razão pela qual os algoritmos de tempo polinomial são geralmente considerados mais desejáveis do que os exponenciais. Essa visão e a distinção entre os dois tipos de algoritmos é a noção central da intratabilidade inerente de determinados problemas e de toda teoria da NP-completude.

A maioria dos algoritmos exponenciais são meras variações da busca exaustiva de um problema, ou seja, verificar todas as soluções possíveis para decidir qual a melhor, ou que atende as restrições do problema. Já os algoritmos de complexidade polinomial geralmente são desenvolvidos a partir do estudo da estrutura do problema. Há um entendimento geral de que um problema ainda não foi "bem-resolvido" enquanto um algoritmo polinomial que o resolva não for descoberto. Portanto, um problema é considerado intratável quando ele é tão difícil de ser resolvido que nenhum algoritmo polinomial consegue resolvê-lo.

Existem alguns algoritmos exponenciais que podem ser muito úteis na prática. A complexidade do algoritmo é geralmente definida no pior caso, logo o fato de um algoritmo possuir complexidade $O(2^n)$ implica somente que existe pelo menos uma instância de tamanho n que requer tempo exponencial. A maioria das instâncias dos problemas de complexidade exponencial podem apresentar um tempo computacional muito menor do que esse, existindo diversos exemplos que corroboram com essa afirmação. O algoritmo simplex para programação linear foi comprovado ser de complexidade exponencial [Klee and Minty, 1972], mas na prática ele funciona muito rápido para diversos problemas. Do mesmo modo, o algoritmo branch-and-bound, para alguns problemas de otimização linear inteira, tem sido tão bem sucedido que alguns consideram esses problemas como bem resolvidos. Apesar disso, esses algoritmos também são de complexidade exponencial. No entanto, poucos algoritmos dessa complexidade são reconhecidamente úteis na prática como esses apresentados.

A definição de intratável, aplicada a problemas computacionais, fornece uma base teórica de grande poder e generalidade. Um problema intratável mostra-se essencialmente independente de qualquer codificação e do computador utilizado para determinar sua complexidade.

5. Problemas Comprovadamente Intratáveis

Torna-se útil distinguir duas possíveis causas da intratabilidade de problemas. A primeira, já discutida, diz respeito àqueles problemas que são de tal ordem difíceis, que não há qualquer algoritmo polinomial conhecido que os resolva à otimalidade. A segunda causa ocorre quando a solução requerida mostra-se tão extensa que não pode ser descrita com uma expressão possuindo um comprimento limitado por uma função polinomial do tamanho da entrada.

A segunda causa ocorre, por exemplo, em uma variante do problema do caixeiro viajante que inclui um número B como parâmetro adicional e pergunta por todas as rotas de comprimento B ou menor que B. É fácil construir uma instância desse problema na qual o número de rotas possíveis torna-se exponencial, de modo que nenhum algoritmo polinomial poderia listar todas elas.

6. Problemas NP-Completos

Pesquisadores continuam a investigar métodos para provar problemas intratáveis. Além disso, estudam maneiras pelas quais diversos problemas estão relacionados com respeito às suas dificuldades. O descobrimento de tal relação entre esses problemas normalmente podem fornecer informações úteis para os desenvolvedores de algoritmos.

A principal técnica utilizada para demonstrar que dois problemas se relacionam é "reduzindo-os" uns nos outros, a partir de uma transformação que mapeia qualquer instância do primeiro problema em uma instância equivalente do segundo. Tal transformação fornece os meios para converter qualquer algoritmo que resolve o segundo problema no algoritmo correspondente ao primeiro problema.

Os fundamentos da teoria da NP-completude foram expressas por Cook (1971), que realizou importantes feitos:

1. Enfatizou o significado de "polinomialmente redutível", ou seja, reduções que podem ser efetuadas por algum algoritmo de complexidade polinomial. Se uma redução polinomial de um problema para outro existir, isso assegura que qualquer algoritmo de tempo polinomial descoberto para o segundo problema, pode ser convertido em um algoritmo polinomial correspondente ao primeiro problema.

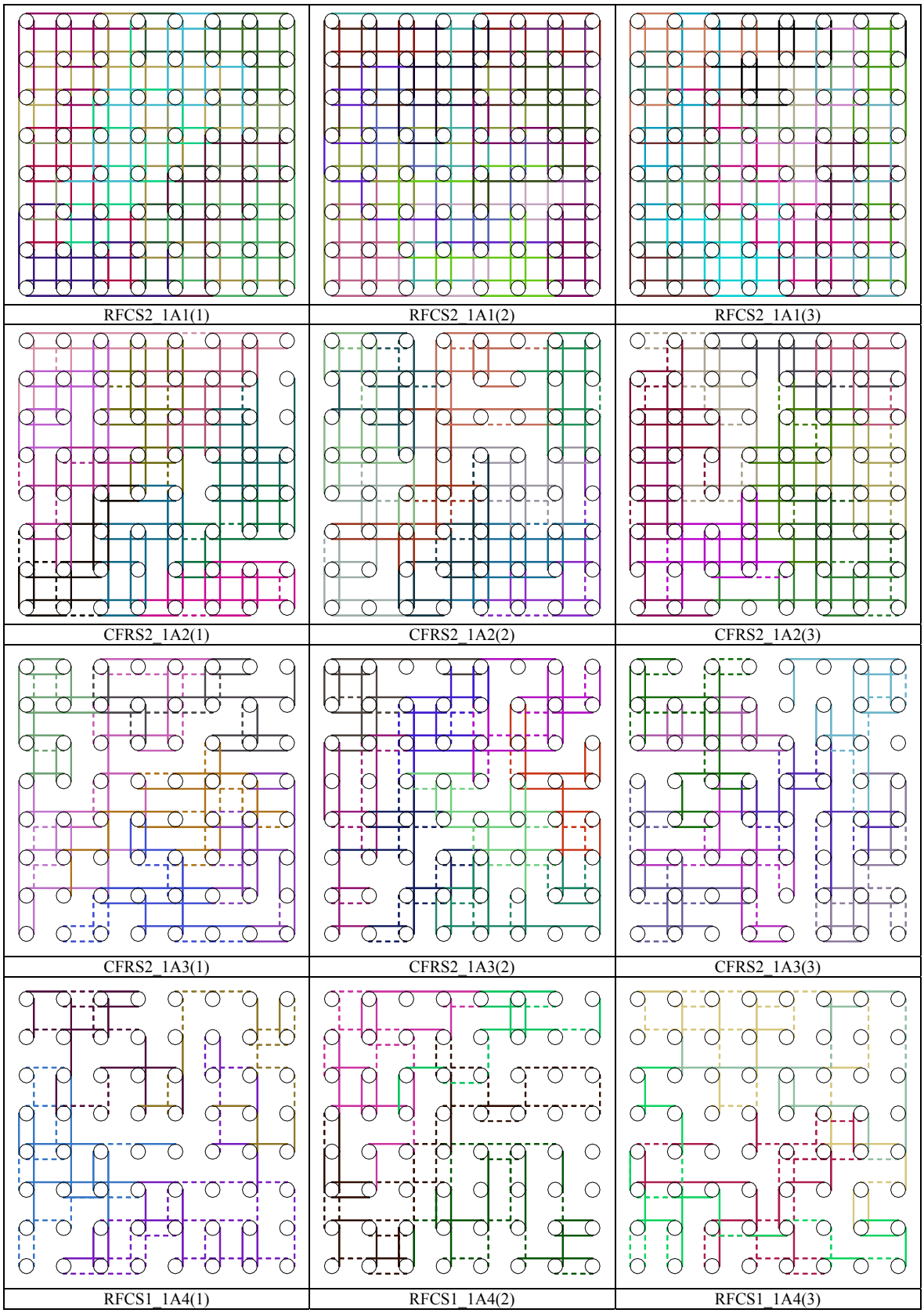
2. Intensificou sua atenção aos problemas de decisão NP, ou seja, que podem ser resolvidos polinomialmente por um computador não-determinístico. (Um problema de decisão é aquele cuja solução é "sim" ou "não").

3. Provou que um problema particular NP, denominado "problema de satisfação", possui a propriedade que qualquer outro problema em NP pode ser polinomialmente reduzido a ele. Se o problema de satisfação pode ser resolvido com um algoritmo de tempo polinomial, então qualquer problema em NP também poderá, e se qualquer problema em NP for intratável, então assim também será o problema de satisfação.

Finalmente, Cook sugeriu que outros problemas NP podem compartilhar com o problema de satisfação a propriedade de ser o mais "difícil" membro de NP. A partir disso, foi provado que muitos problemas, incluindo o problema do Caixeiro Viajante, são tão difíceis quanto o problema de satisfação, o que deu origem a uma classe de problemas denominados NP-Completos.

Se os problemas NP-Completos são intratáveis ou não, corresponde a um dos questionamentos em aberto mais importantes da matemática contemporânea e da ciência da computação. Embora a persistência dos pesquisadores em conjeturar que os problemas NP-Completos são intratáveis, pouco progresso ocorreu no sentido de estabelecer uma prova ou contra-prova dessa conjectura. No entanto, mesmo sem uma prova que a NP-Completude implica intratabilidade, um avanço muito grande será necessário para resolvê-los com um algoritmo de complexidade polinomial.

VIII. APENDICE II: SAÍDAS GRÁFICAS DAS MELHORES SOLUÇÕES

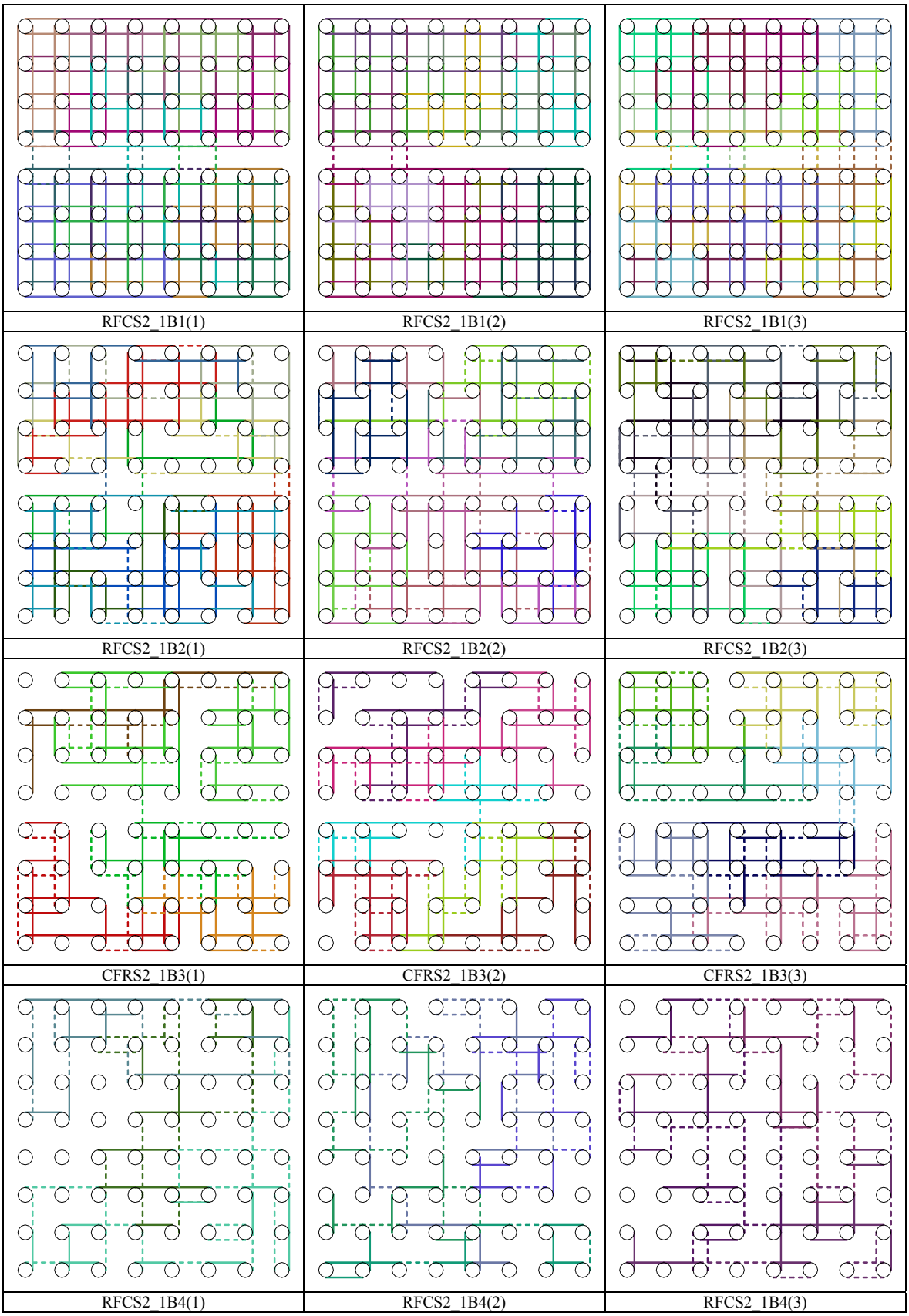


RFCS2_1A1(1) | RFCS2_1A1(2) | RFCS2_1A1(3)

CFRS2_1A2(1) | CFRS2_1A2(2) | CFRS2_1A2(3)

CFRS2_1A3(1) | CFRS2_1A3(2) | CFRS2_1A3(3)

RFCS1_1A4(1) | RFCS1_1A4(2) | RFCS1_1A4(3)

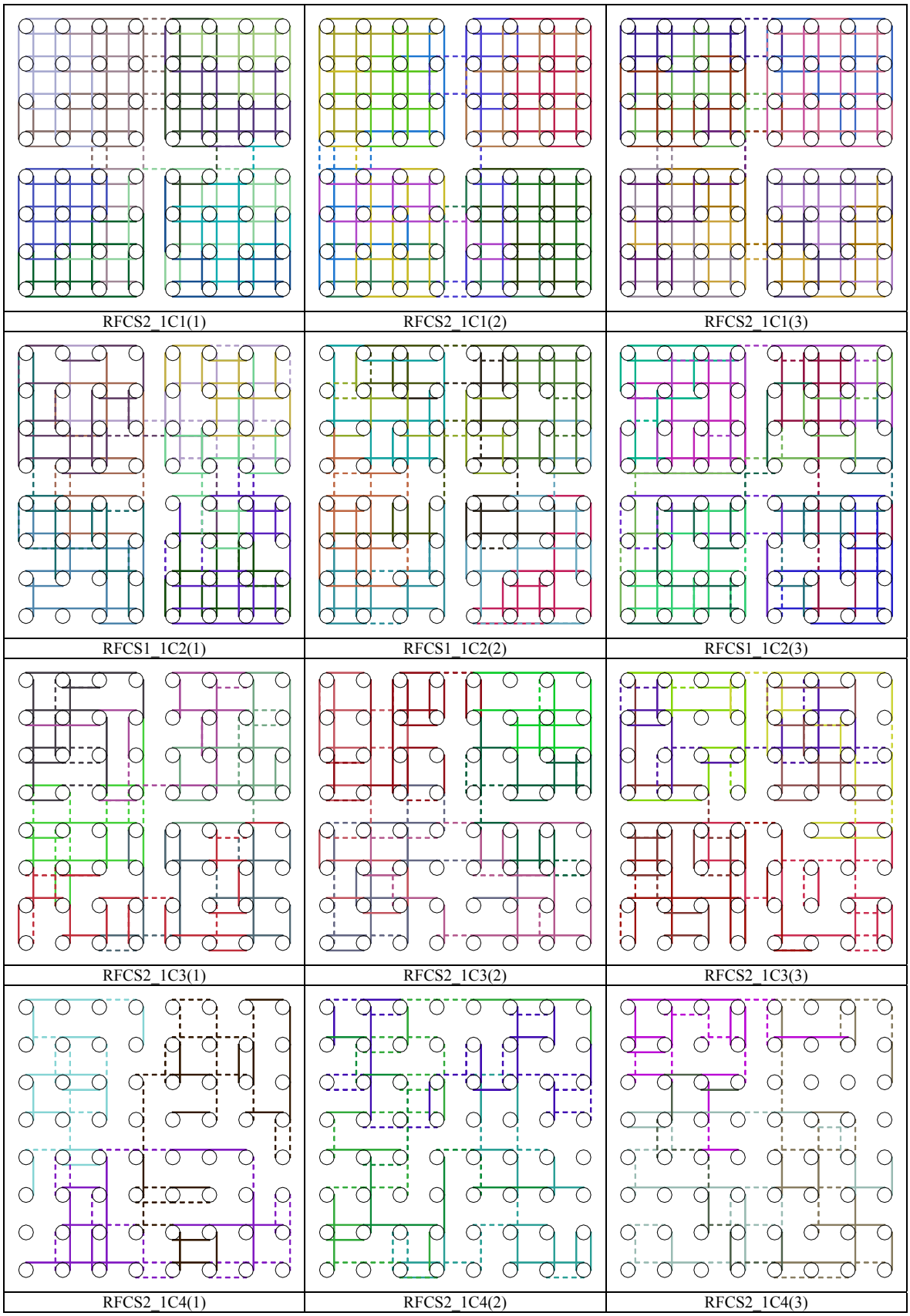


RFCS2_1B1(1) | RFCS2_1B1(2) | RFCS2_1B1(3)

RFCS2_1B2(1) | RFCS2_1B2(2) | RFCS2_1B2(3)

CFRS2_1B3(1) | CFRS2_1B3(2) | CFRS2_1B3(3)

RFCS2_1B4(1) | RFCS2_1B4(2) | RFCS2_1B4(3)

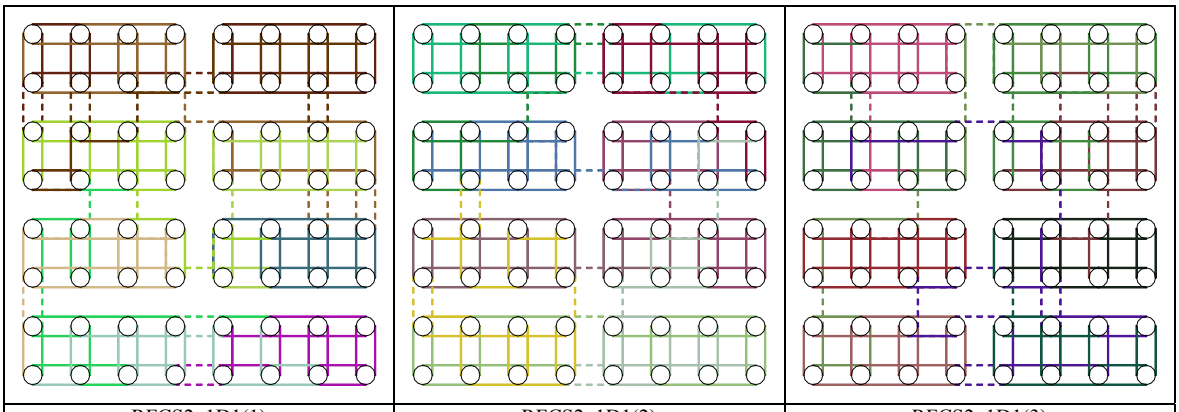


RFCS2_1C1(1) | RFCS2_1C1(2) | RFCS2_1C1(3)

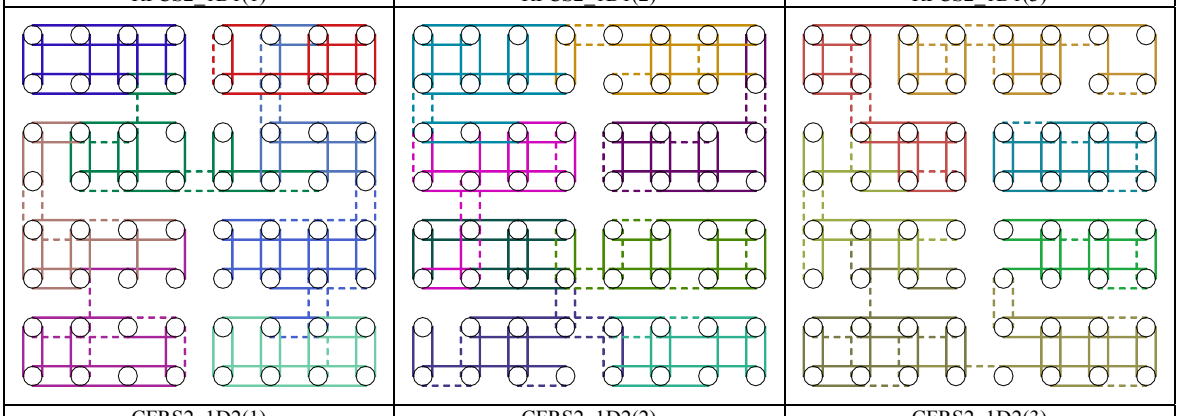
RFCS1_1C2(1) | RFCS1_1C2(2) | RFCS1_1C2(3)

RFCS2_1C3(1) | RFCS2_1C3(2) | RFCS2_1C3(3)

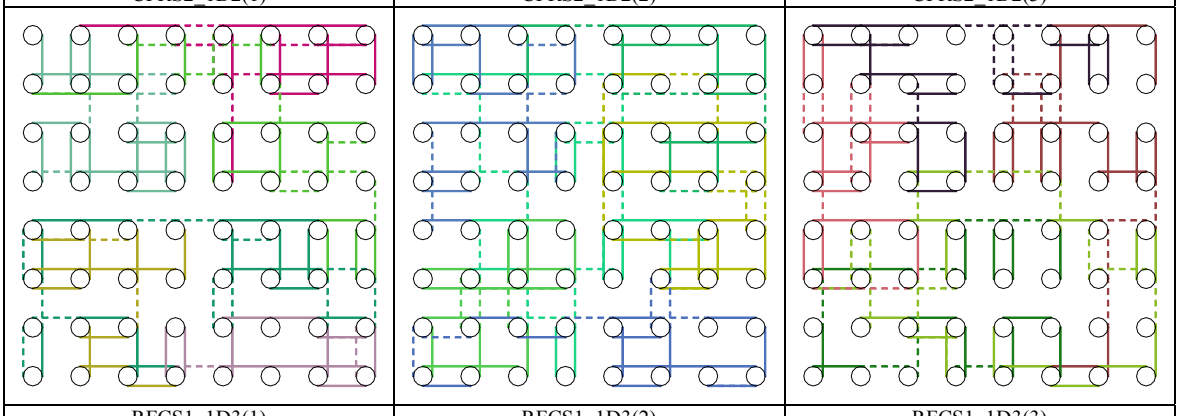
RFCS2_1C4(1) | RFCS2_1C4(2) | RFCS2_1C4(3)



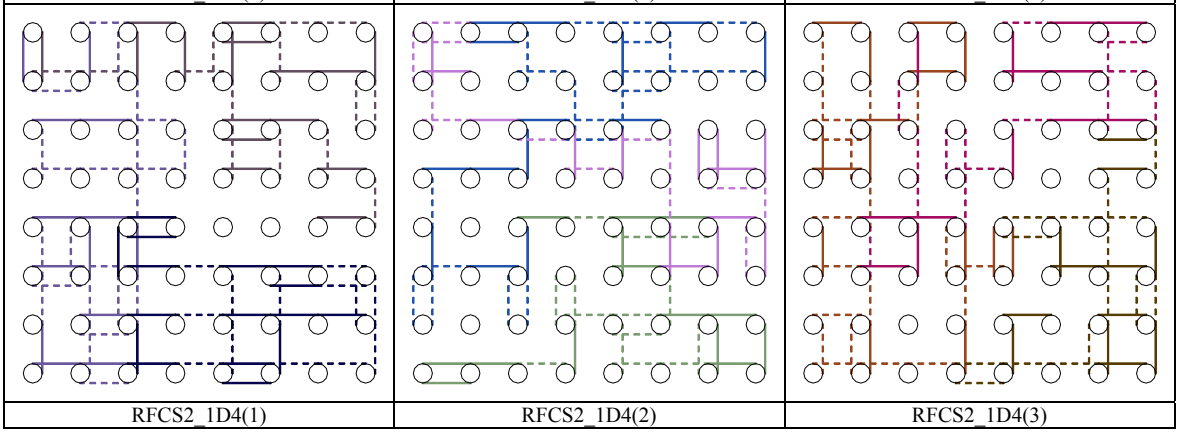
RFCS2_1D1(1) | RFCS2_1D1(2) | RFCS2_1D1(3)



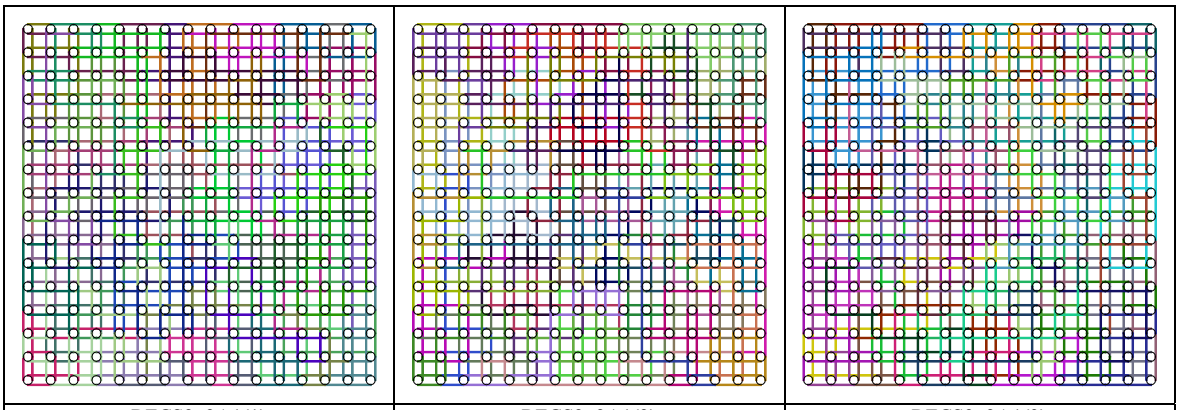
CFRS2_1D2(1) | CFRS2_1D2(2) | CFRS2_1D2(3)



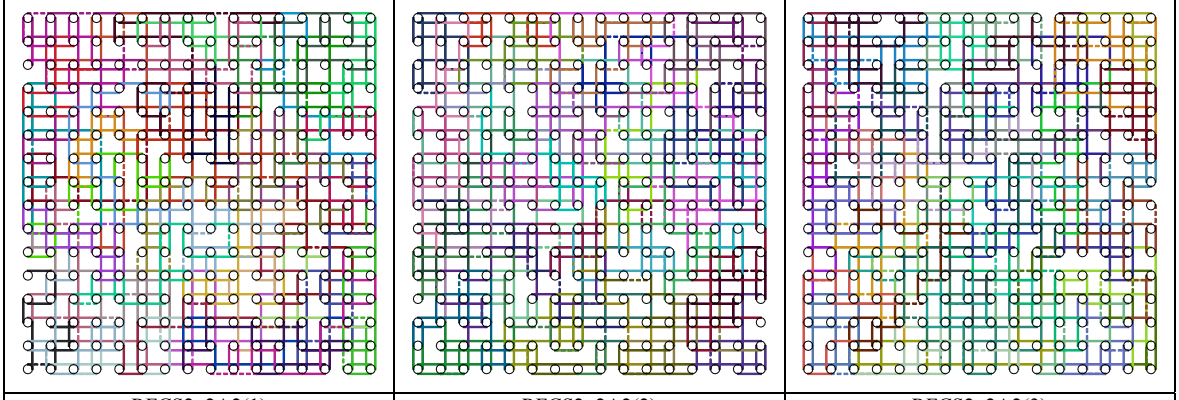
RFCS1_1D3(1) | RFCS1_1D3(2) | RFCS1_1D3(3)



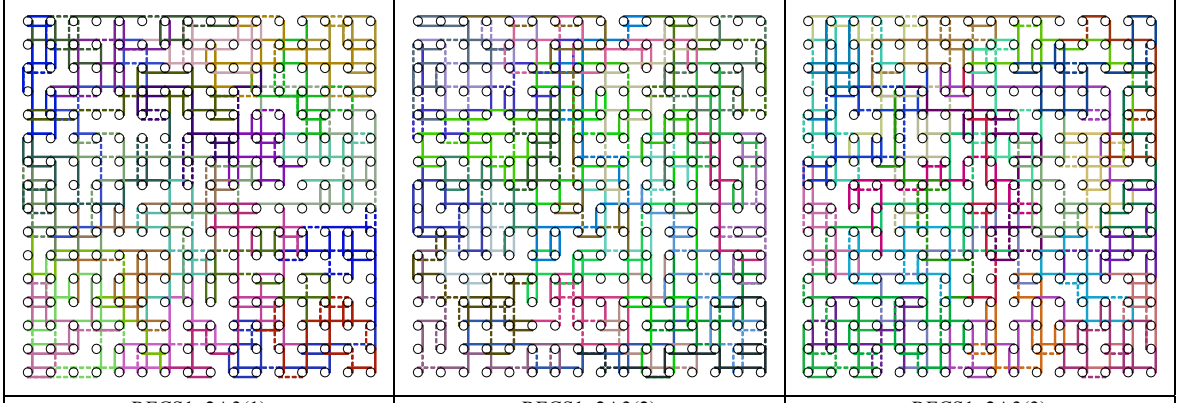
RFCS2_1D4(1) | RFCS2_1D4(2) | RFCS2_1D4(3)



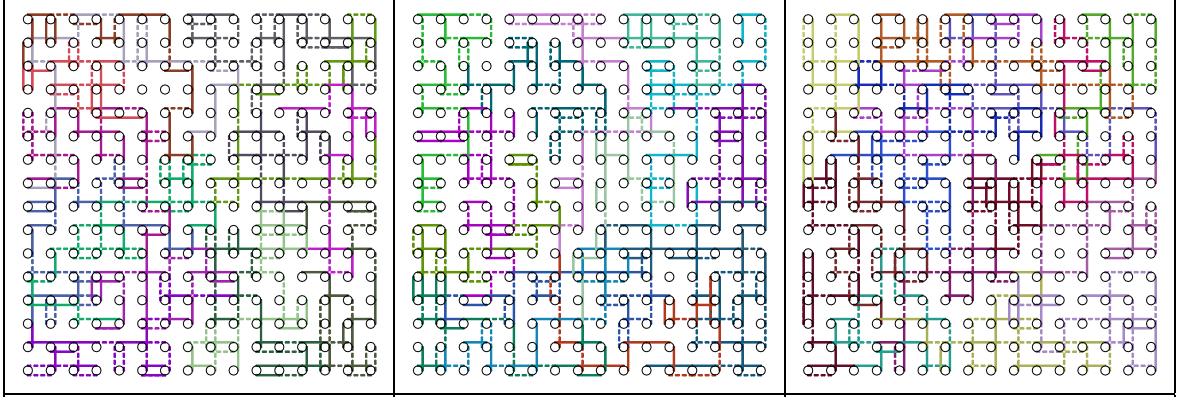
RFCS2_2A1(1) | RFCS2_2A1(2) | RFCS2_2A1(3)



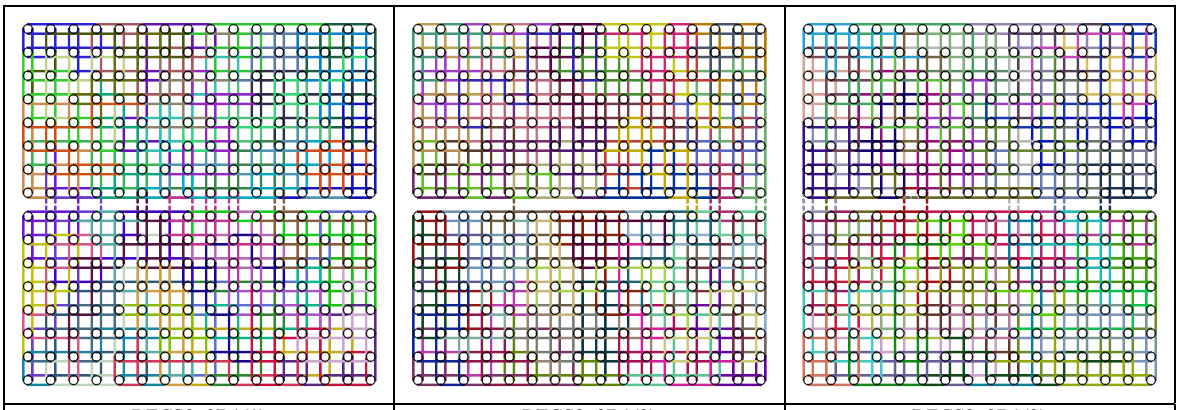
RFCS2_2A2(1) | RFCS2_2A2(2) | RFCS2_2A2(3)



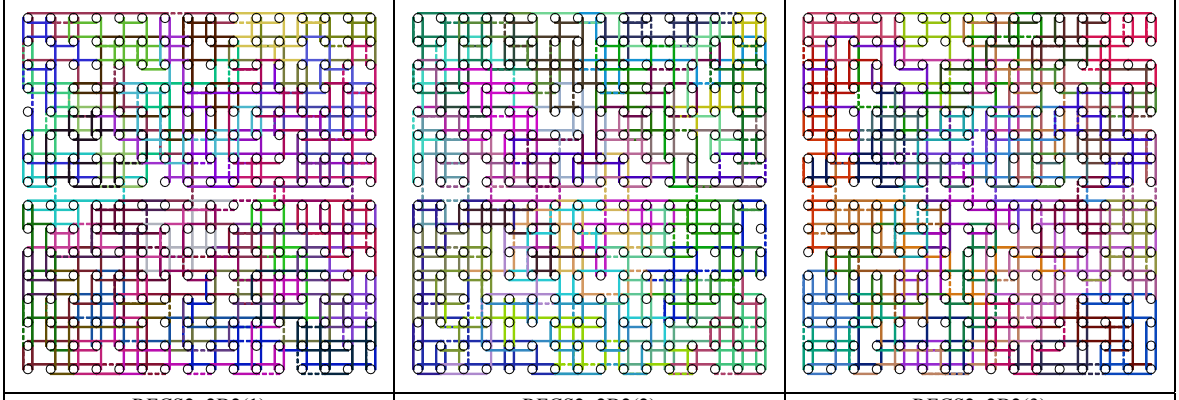
RFCS1_2A3(1) | RFCS1_2A3(2) | RFCS1_2A3(3)



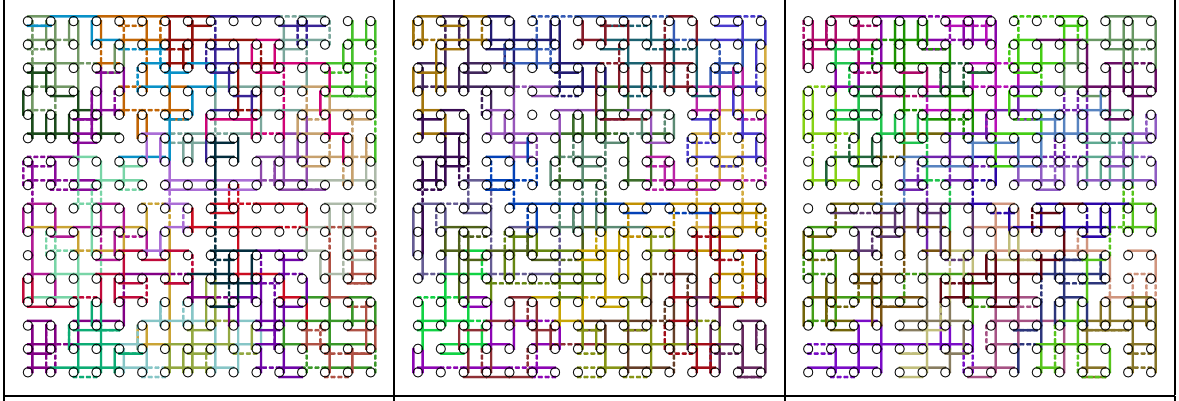
RFCS2_2A4(1) | RFCS2_2A4(2) | RFCS2_2A4(3)



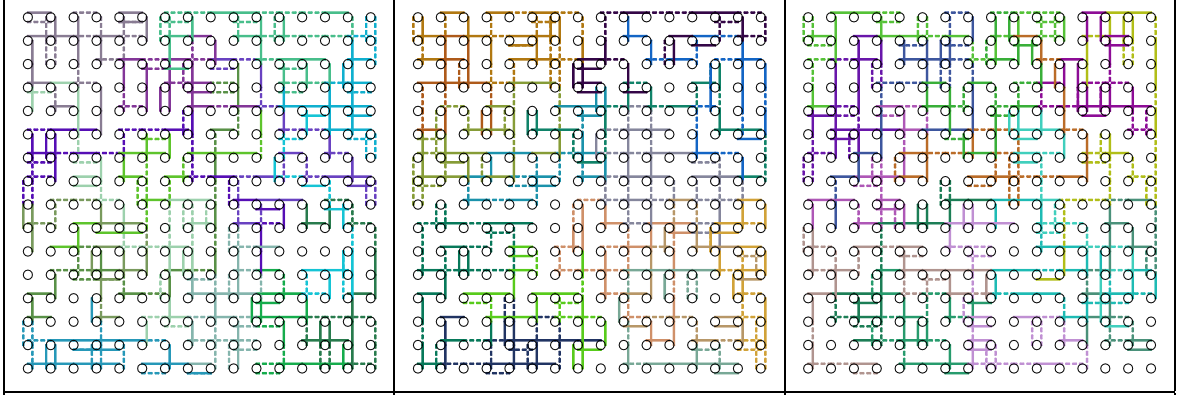
RFCS2_2B1(1) | RFCS2_2B1(2) | RFCS2_2B1(3)



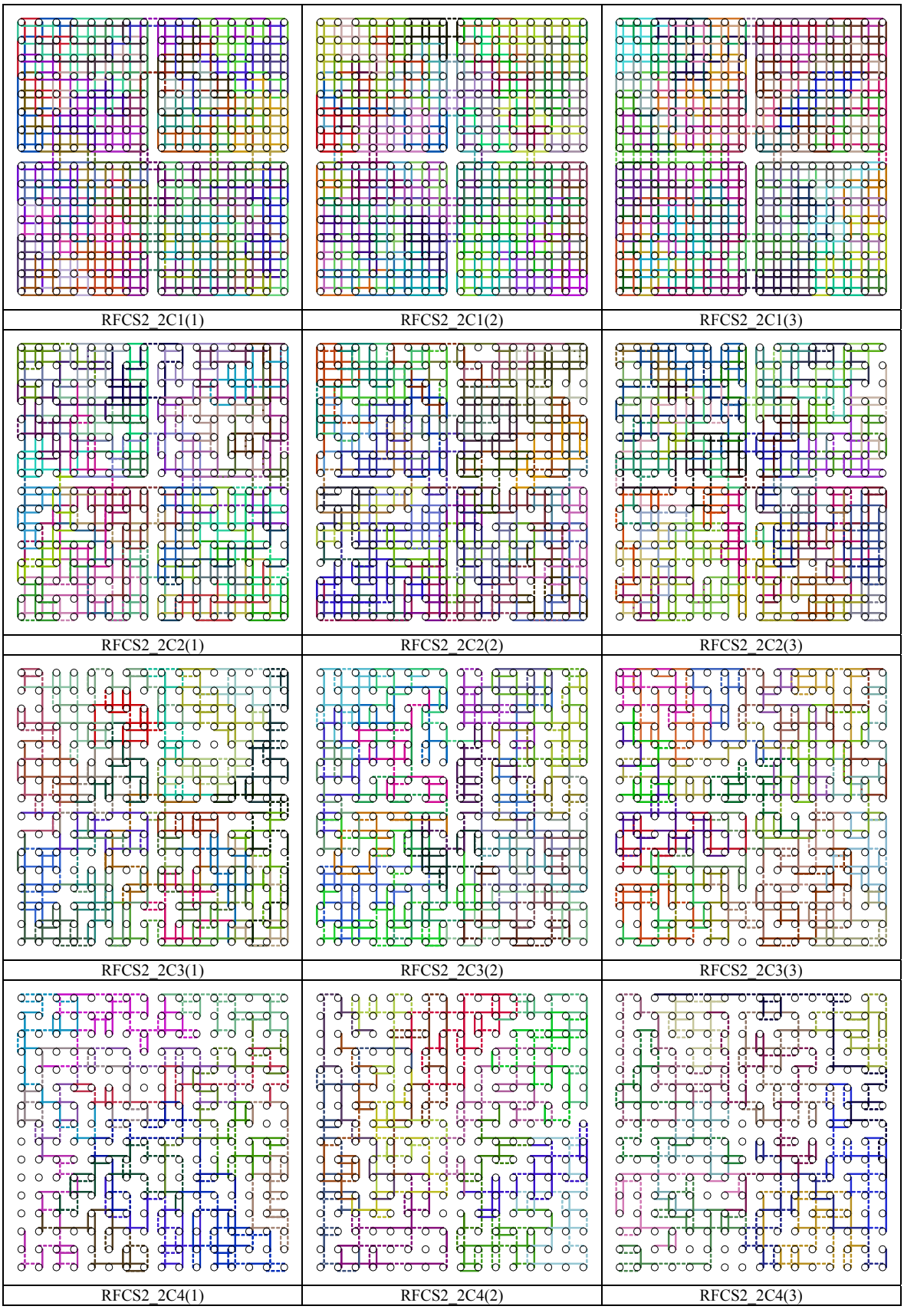
RFCS2_2B2(1) | RFCS2_2B2(2) | RFCS2_2B2(3)

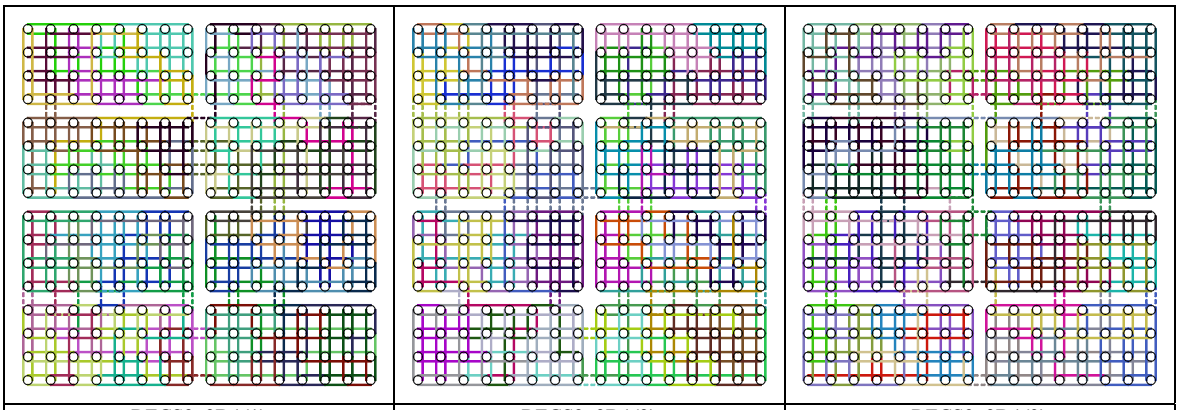


RFCS2_2B3(1) | RFCS2_2B3(2) | RFCS2_2B3(3)

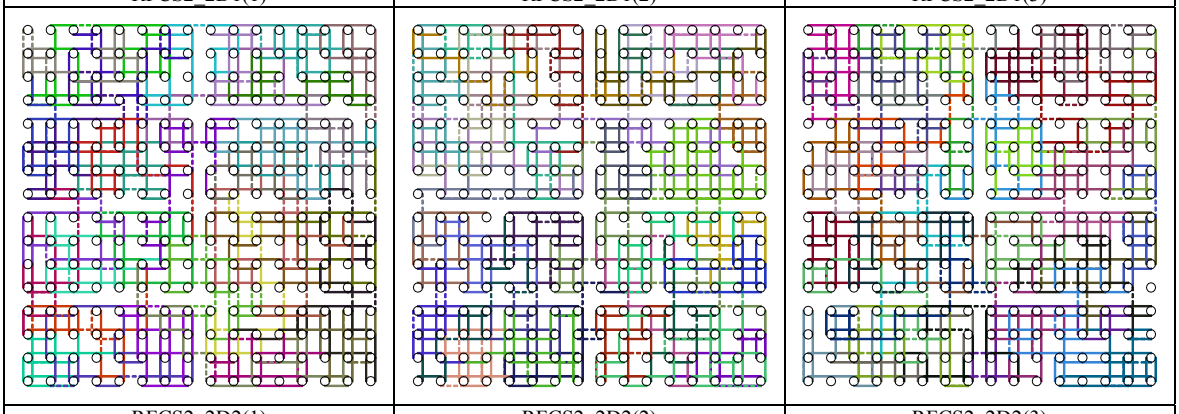


RFCS2_2B4(1) | RFCS2_2B4(2) | RFCS2_2B4(3)

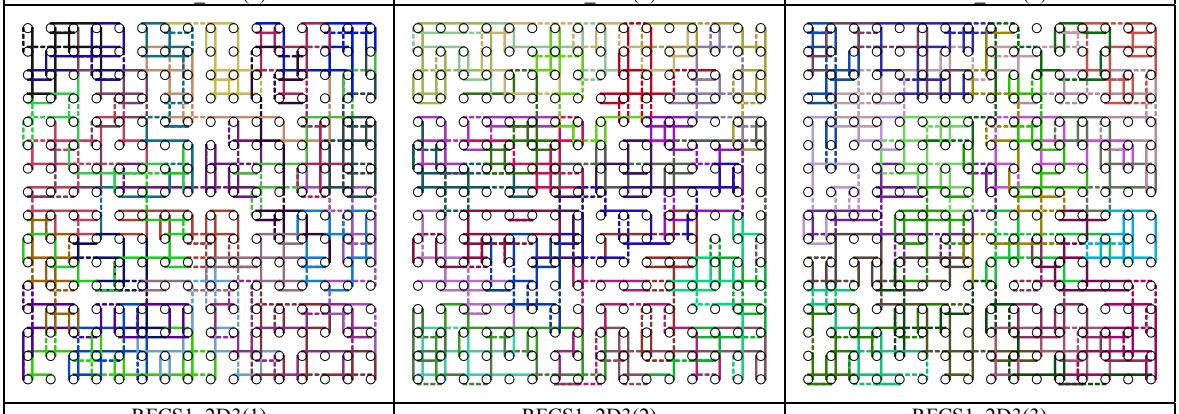

RFCS2_2C1(1)
RFCS2_2C1(2)
RFCS2_2C1(3)
RFCS2_2C2(1)
RFCS2_2C2(2)
RFCS2_2C2(3)
RFCS2_2C3(1)
RFCS2_2C3(2)
RFCS2_2C3(3)
RFCS2_2C4(1)
RFCS2_2C4(2)
RFCS2_2C4(3)



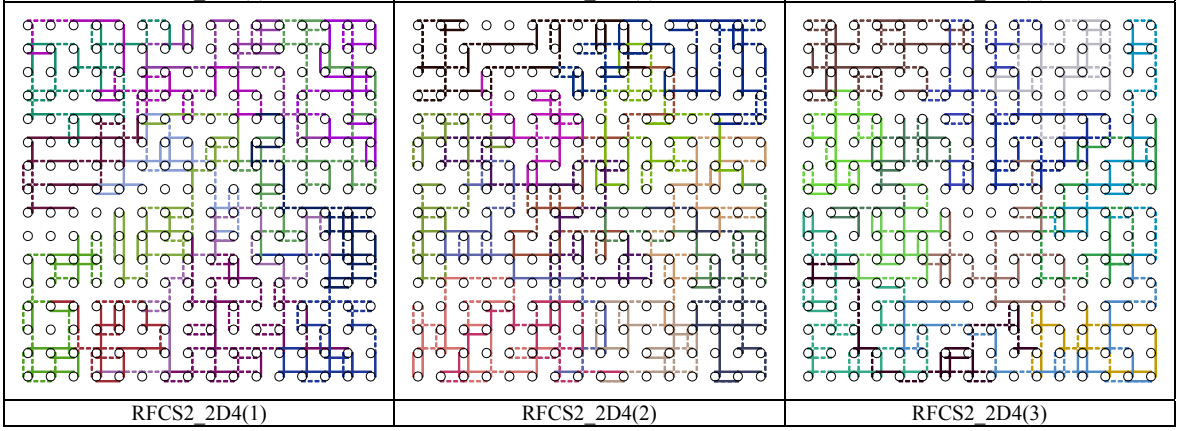
RFCS2_2D1(1) | RFCS2_2D1(2) | RFCS2_2D1(3)



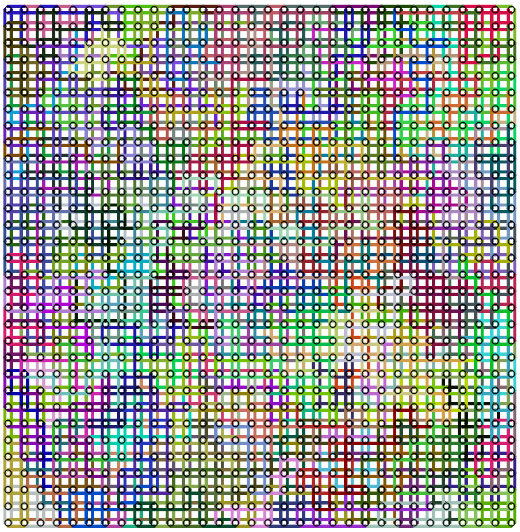
RFCS2_2D2(1) | RFCS2_2D2(2) | RFCS2_2D2(3)



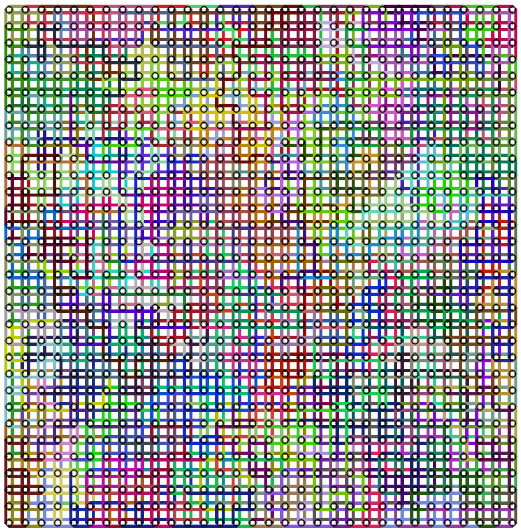
RFCS1_2D3(1) | RFCS1_2D3(2) | RFCS1_2D3(3)



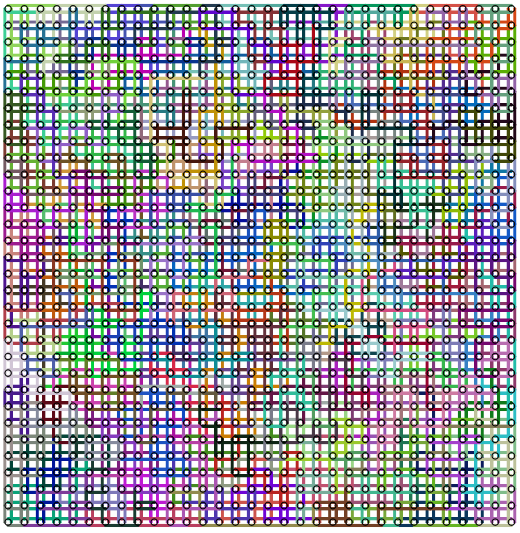
RFCS2_2D4(1) | RFCS2_2D4(2) | RFCS2_2D4(3)



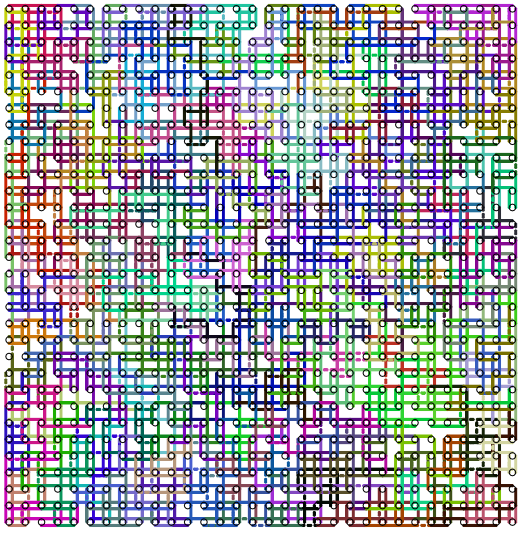
RFCS2_3A1(1)



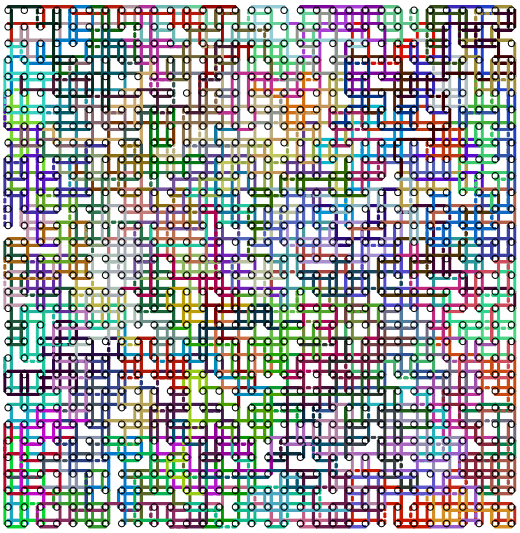
RFCS2_3A1(2)



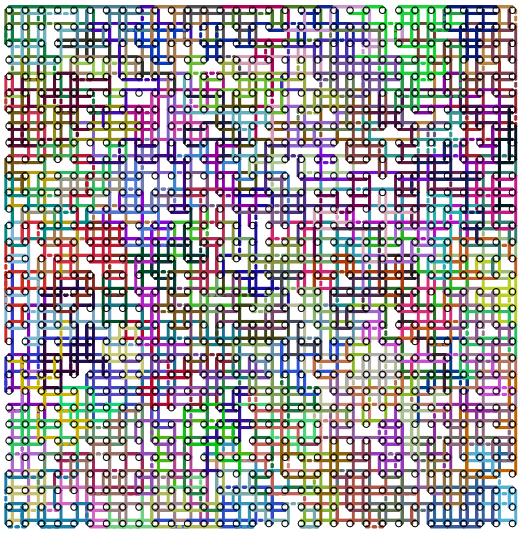
RFCS2_3A1(3)



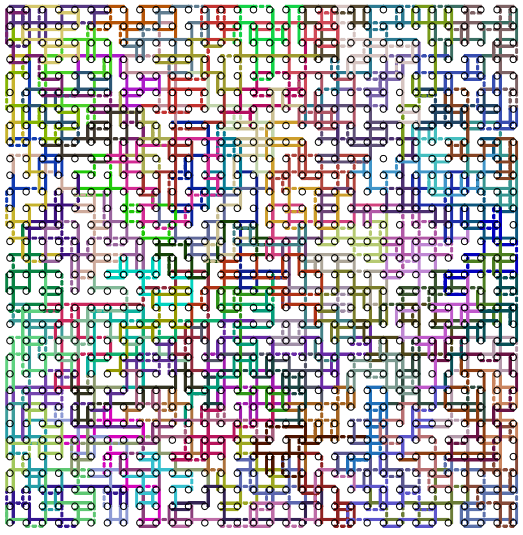
RFCS2_3A2(1)



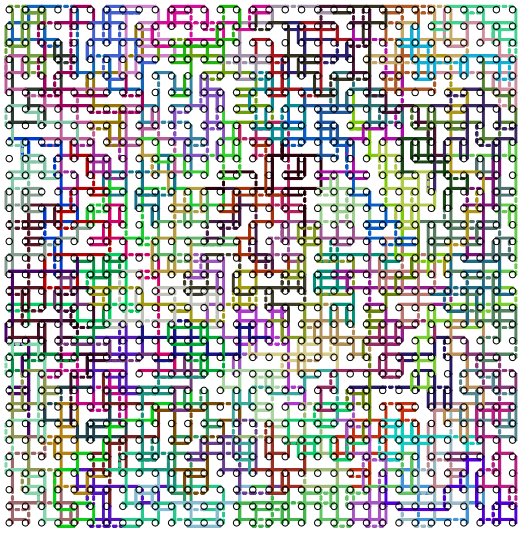
RFCS2_3A2(2)



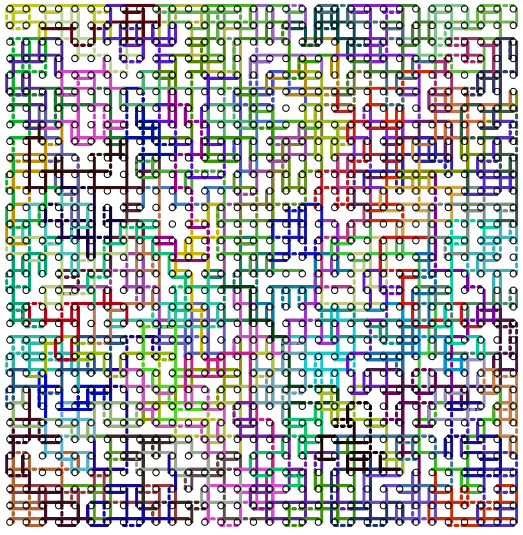
RFCS2_3A2(3)



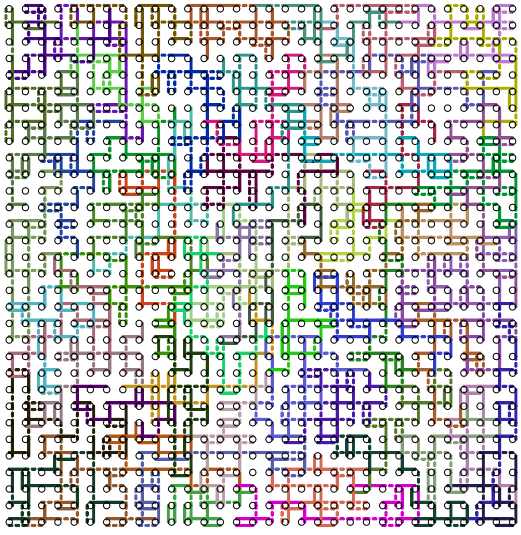
RFCS2_3A3(1)



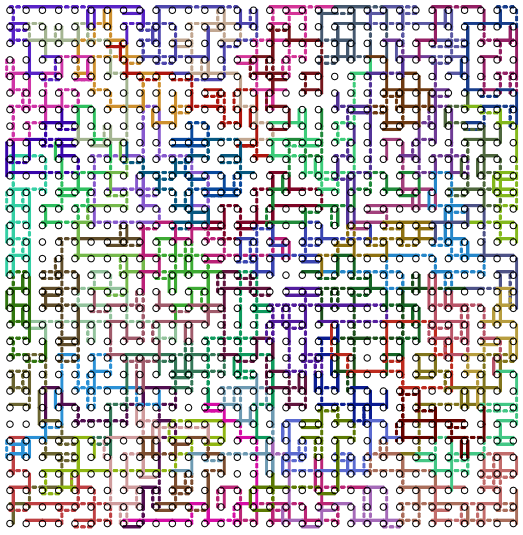
RFCS2_3A3(2)



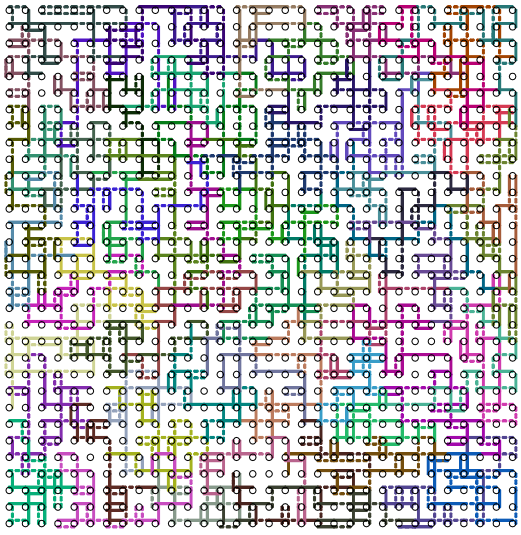
RFCS2_3A3(3)



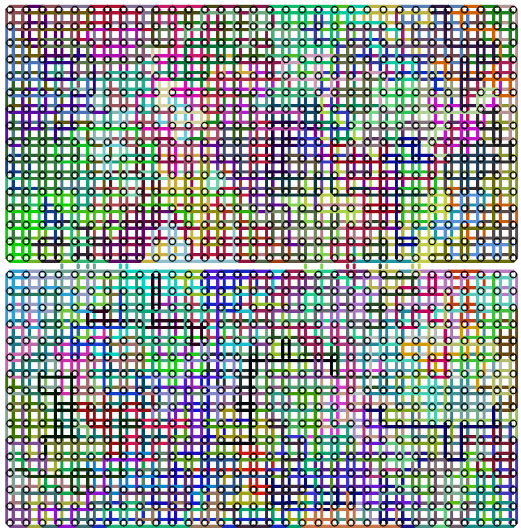
RFCS2_3A4(1)



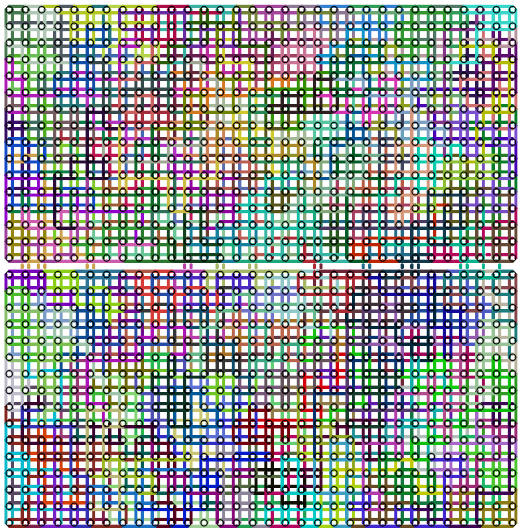
RFCS2_3A4(2)



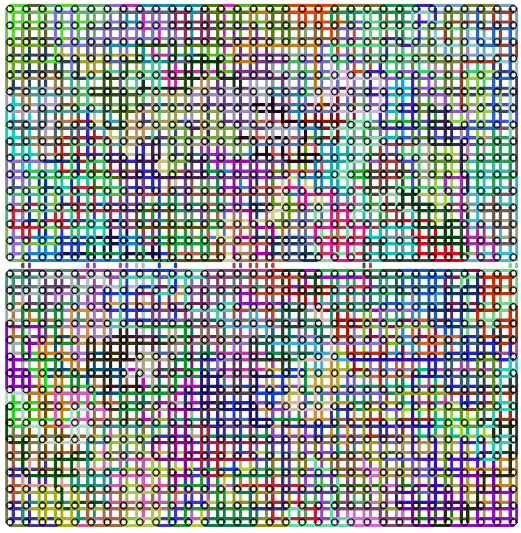
RFCS2_3A4(3)



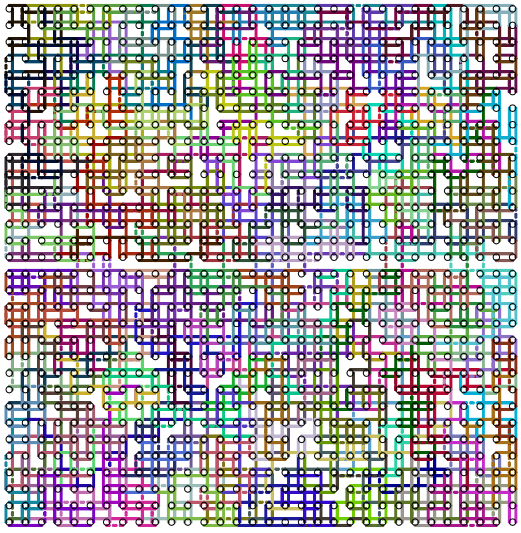
RFCS2_3B1(1)



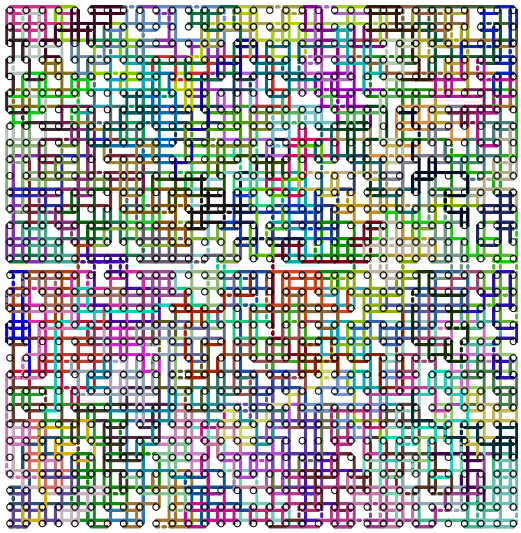
RFCS2_3B1(2)



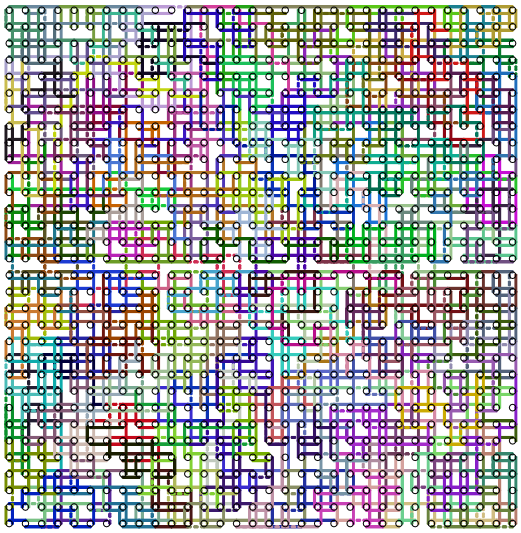
RFCS2_3B1(3)



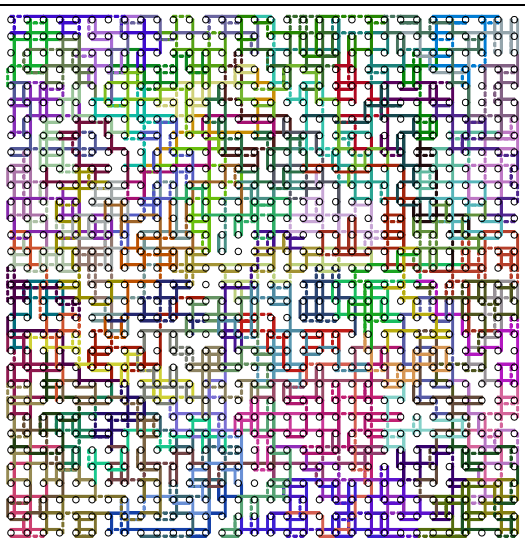
RFCS2_3B2(1)



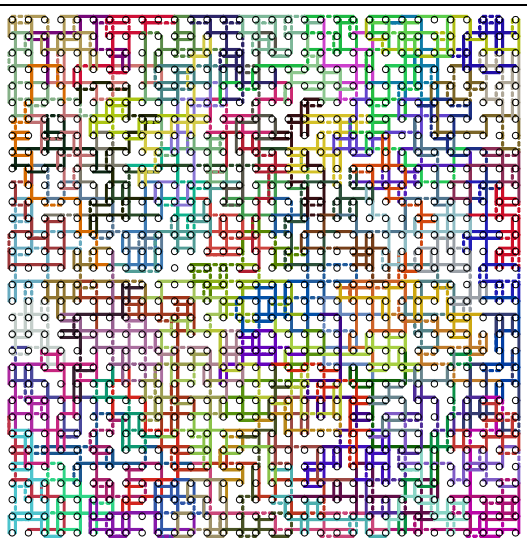
RFCS2_3B2(2)



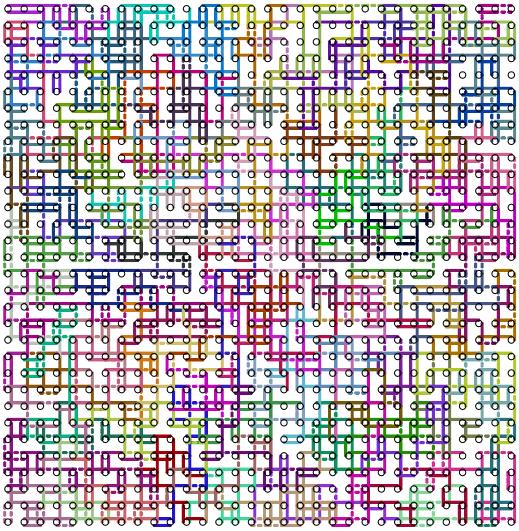
RFCS2_3B2(3)



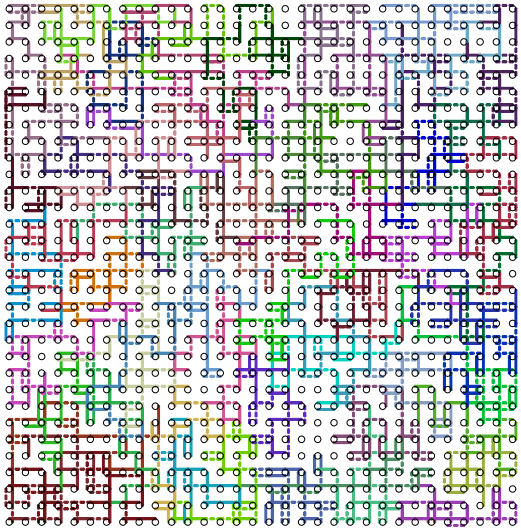
RFCS2_3B3(1)



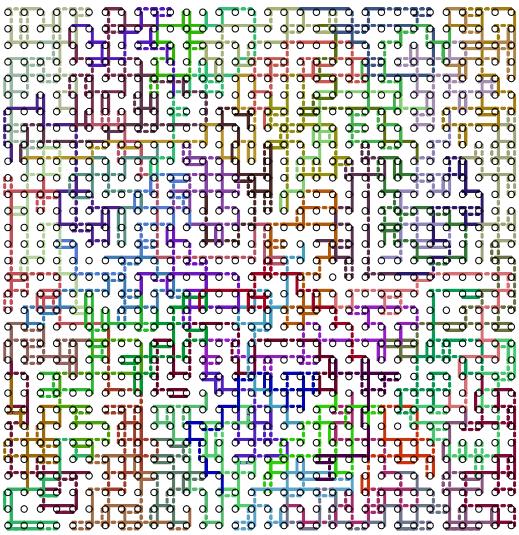
RFCS2_3B3(2)



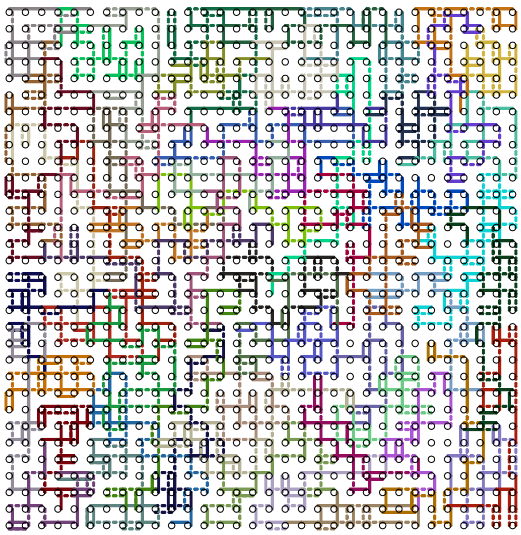
RFCS2_3B3(3)



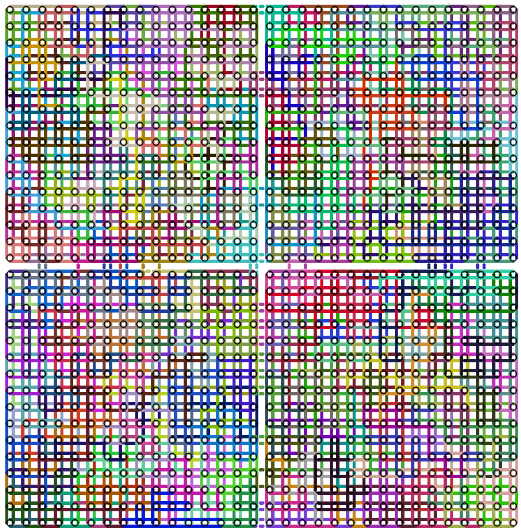
RFCS2_3B4(1)



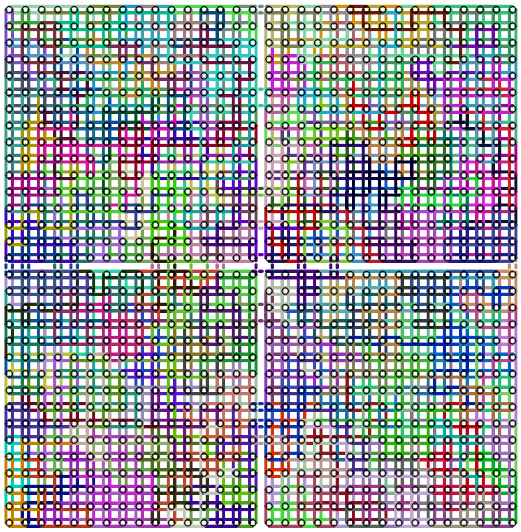
RFCS2_3B4(2)



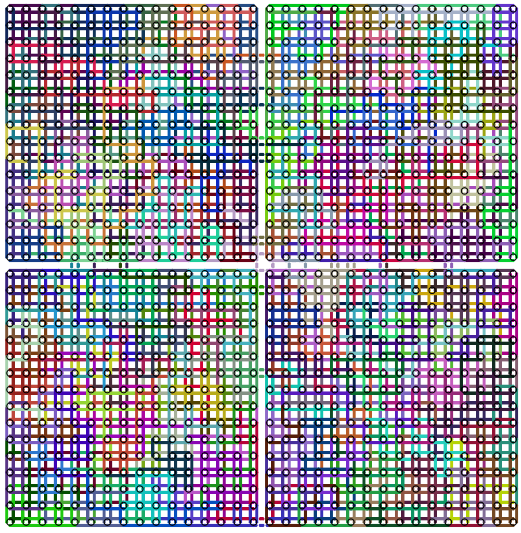
RFCS2_3B4(3)



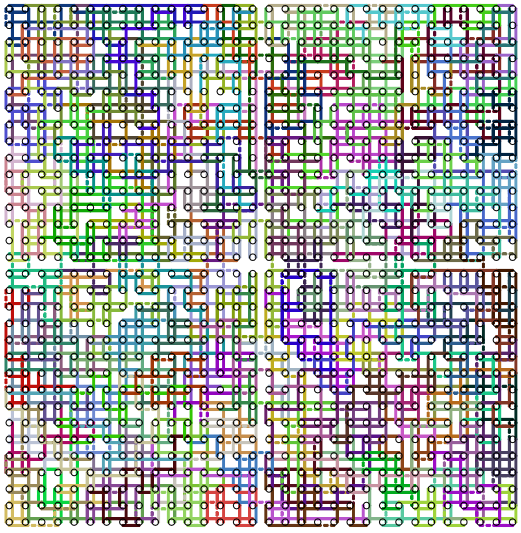
RFCS2_3C1(1)



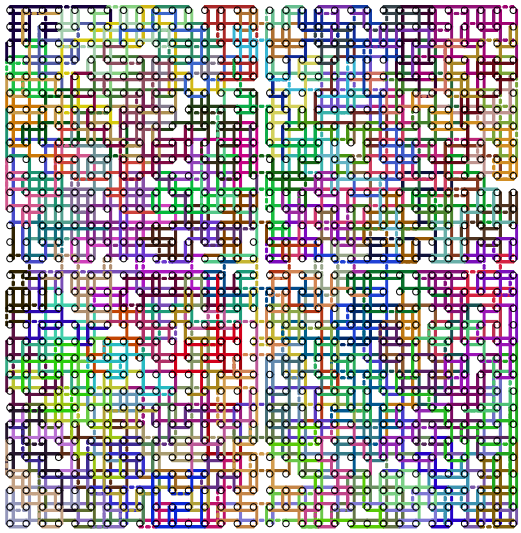
RFCS2_3C1(2)



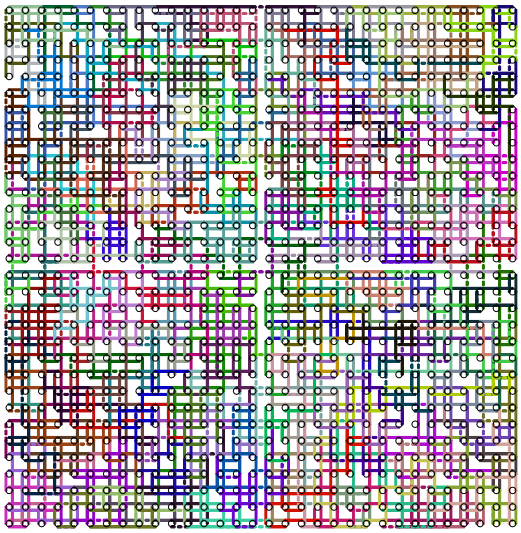
RFCS2_3C1(3)



RFCS2_3C2(1)



RFCS2_3C2(2)



RFCS2_3C2(3)

RFCS2_3C3(1)

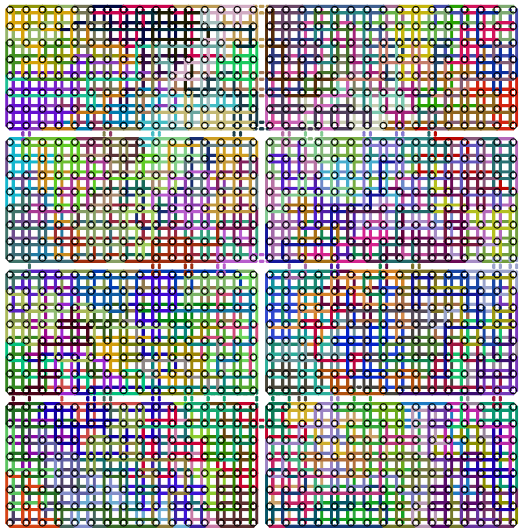
RFCS2_3C3(2)

RFCS2_3C3(3)

RFCS2_3C4(1)

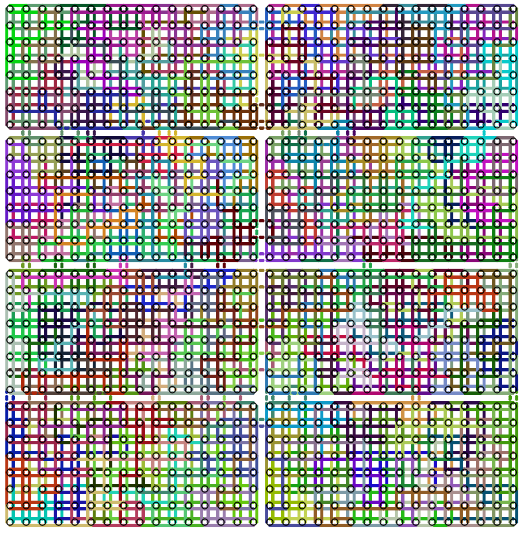
RFCS2_3C4(2)

RFCS2_3C4(3)

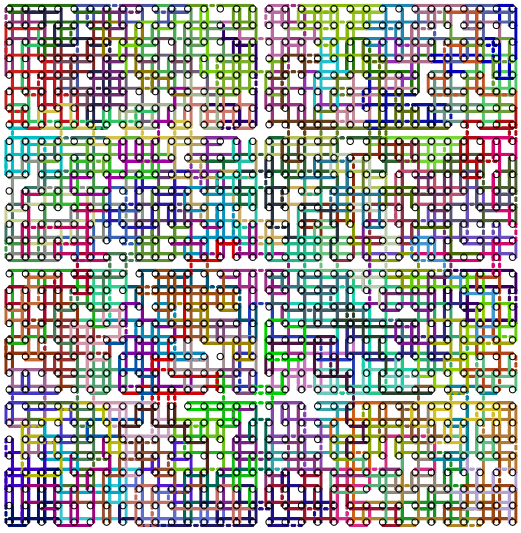


RFCS2_3D1(1)

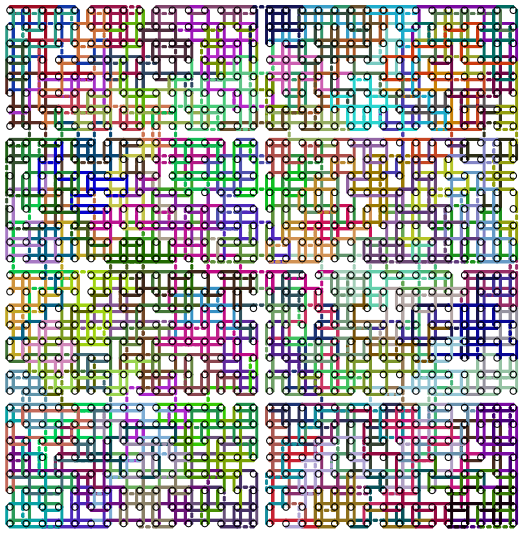
RFCS2_3D1(2)



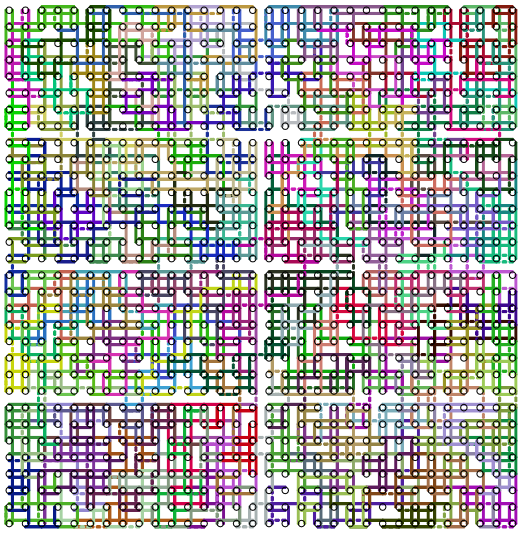
RFCS2_3D1(3)



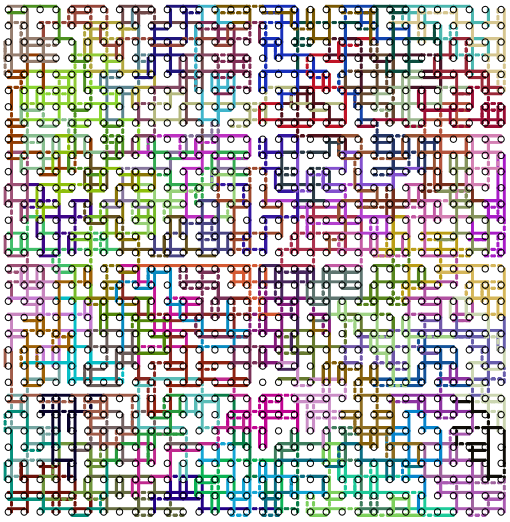
RFCS2_3D2(1)



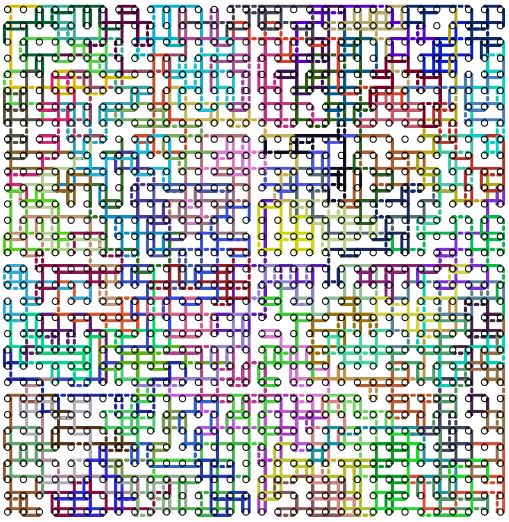
RFCS2_3D2(2)



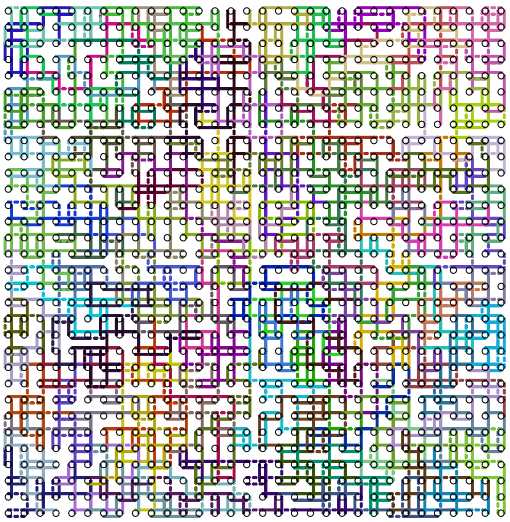
RFCS2_3D2(3)



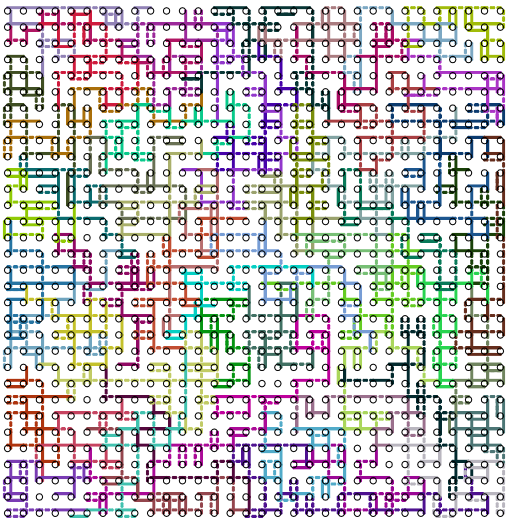
RFCS2_3D3(1)



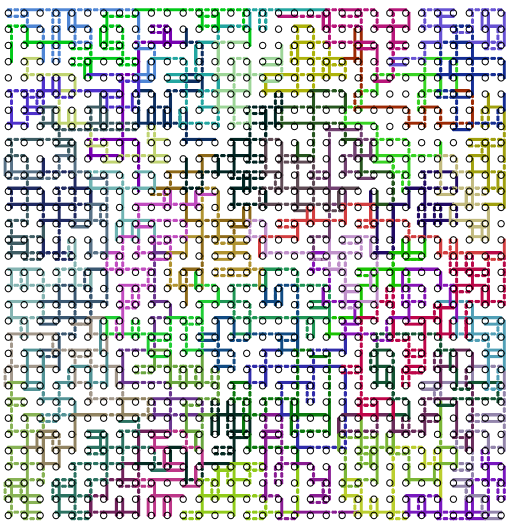
RFCS2_3D3(2)



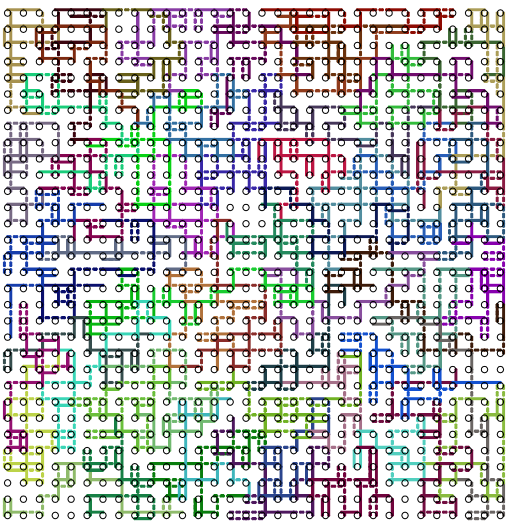
RFCS2_3D3(3)



RFCS2_3D4(1)



RFCS2_3D4(2)



RFCS2_3D4(3)