UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
DEMAC
Ciências da Computação

André Luis Dias Nogueira
Felipe Melchior de Britto
Rafael Daiki Kaneko
Ryan Hideki Tadeo Guimarães
Vitor Marchini Rolisola

Relatório sobre Grafos Hamiltonianos

Relatório Acadêmico

Rio Claro, 2024

André Luis Dias Nogueira
Felipe Melchior de Britto
Rafael Daiki Kaneko
Ryan Hideki Tadeo Guimarães
Vitor Marchini Rolisola

Relatório sobre Grafos Hamiltonianos

Este relatório apresenta a implementação de testes para verificar se grafos aleatórios satisfazem os teoremas hamiltonianos de Dirac, Ore e Bondy-Chvátal, utilizando modelos de grafos com ciclo inicial e arestas adicionadas com probabilidade p. A análise foi realizada para diferentes valores de N e p, com resultados apresentados em gráficos e tabelas.

UNIVERSIDADE ESTADUAL PAULISTA

Orientador: Prof. Emílio Bergamim Júnior

Rio Claro
2024

RESUMO

Este trabalho tem como objetivo de realizar a análise de grafos hamiltonianos por meio da implementação e avaliação dos teoremas de Dirac, Ore e Bondy-Chvátal. Foi aplicado esses critérios em grafos aleatórios gerados a partir de dois modelos principais: o modelo Cíclico-Aleatório, que inicia com um ciclo hamiltoniano e incrementa conexões com uma probabilidade p, e o modelo de Erdos-Renyi, onde cada par de vértices recebe uma aresta com probabilidade fixa. Foram gerados grafos aleatórios para diferentes combinações de N (número de vértices) e p, e aplicados testes para determinar a conformidade com os teoremas citados. Também foi desenvolvido um programa para realizar a automação desses testes e gerar os resultados apresentados neste trabalho. Os resultados, apresentados em gráficos e tabelas, demonstram a eficácia de cada teorema e modelo na identificação de grafos hamiltonianos, oferecendo uma análise empírica sobre a aplicabilidade desses critérios em redes complexas.

Palavras-chaves: grafos hamiltonianos, teorema de Dirac, teorema de Ore, teorema de Bondy-Chvátal, modelo de Erdos-Renyi, teoria dos grafos.

SUMÁRIO

1 INTRODUÇÃO

A Teoria dos Grafos é uma área essencial da matemática discreta, com aplicações em diversos campos, incluindo ciência da computação, logística, redes de comunicação, biologia computacional e pesquisa operacional. Um conceito particularmente relevante nessa teoria é o de ciclos hamiltonianos, onde se busca um percurso cíclico que passa por todos os vértices de um grafo exatamente uma vez. Grafos que contêm tais ciclos são denominados **grafos hamiltonianos** e têm implicações práticas em problemas de otimização, como o problema do caixeiro viajante e o roteamento de redes, onde se procura uma rota eficiente que minimize o custo de deslocamento.

1.1 JUSTIFICATIVAS E RELEVÂNCIA

O estudo de grafos hamiltonianos ganha importância na medida em que muitos problemas complexos podem ser simplificados pela verificação de hamiltonianidade em suas representações gráficas. Contudo, a determinação exata da presença de ciclos hamiltonianos é um problema computacionalmente difícil (NP-completo). Para contornar essa dificuldade, a teoria propõe critérios suficientes de hamiltonianidade, que, embora não garantam uma solução exata para todos os grafos, oferecem maneiras eficientes de inferir a presença de ciclos hamiltonianos em grafos que satisfaçam certas condições. Os teoremas de **Dirac**, **Ore** e **Bondy-Chvátal** são três desses critérios, cada um propondo condições suficientes que, quando satisfeitas, garantem a hamiltonianidade do grafo. A relevância desses teoremas está no potencial de reduzir significativamente a complexidade do problema da hamiltonianidade, o que tem implicações diretas em áreas como o planejamento urbano e a configuração de redes, onde rotas e conexões precisam ser eficientes e bem estruturadas.

Explorar e comparar os modelos de grafos que satisfaçam esses teoremas em condições variáveis de conexão oferece uma base empírica valiosa para avaliar a aplicabilidade e a robustez de cada critério. Esse estudo também contribui para uma melhor compreensão dos modelos aleatórios de grafos, que frequentemente são usados para simular redes reais, onde a distribuição de conexões segue padrões probabilísticos.

1.2 METODOLOGIA

A metodologia proposta para este estudo envolve a implementação de um conjunto de testes para verificar se um grafo dado satisfaz os critérios de hamiltonianidade estabele-

cidos pelos teoremas de Dirac, Ore e Bondy-Chvátal. Para isso, serão aplicados algoritmos específicos para cada teorema:

1. **Teste de Dirac**: Será verificado se todos os vértices de um grafo possuem grau $\delta \geq \frac{n}{2}$, sendo n o número de vértices do grafo.
2. **Teste de Ore**: Para cada par de vértices não adjacentes u e v, será avaliado se a soma dos graus $d(u) + d(v) \geq n$.
3. **Teste de Bondy-Chvátal**: Utilizando o método de fechamento do grafo, serão inseridas arestas entre vértices não adjacentes sempre que a soma de seus graus seja pelo menos n, e em seguida, será avaliado se o grafo resultante é hamiltoniano.

Os testes serão aplicados a grafos gerados aleatoriamente de acordo com dois modelos:

- **Modelo Cíclico-Aleatório**: Um grafo inicialmente configurado como um ciclo simples de N vértices (o que garante que ele seja hamiltoniano) e, em seguida, arestas adicionais são inseridas entre pares de vértices com uma probabilidade p.
- **Modelo de Erdos-Renyi**: Cada par de vértices recebe uma aresta com uma probabilidade fixa p, sem uma configuração inicial de ciclo, resultando em grafos com conectividade aleatória.

Para cada combinação de N (número de vértices) e p (probabilidade de conexão), serão gerados dez grafos aleatórios. Cada grafo será submetido aos três testes, e os resultados serão organizados em tabelas e gráficos, comparando a frequência com que cada teorema é satisfeito em cada modelo.

1.2.1 TEOREMA DE DIRAC

O Teorema de Dirac, um dos primeiros critérios suficientes para a hamiltonianidade, postula que um grafo simples com $n \geq 3$ vértices é hamiltoniano se todos os seus vértices possuem grau $d(v) \geq n/2$. Esse teorema baseia-se na premissa de que, quando cada vértice possui um número mínimo de conexões, o grafo torna-se suficientemente "denso"para conter um ciclo hamiltoniano. O Teorema de Dirac é relevante por sua simplicidade e pela garantia que fornece em grafos densos, mas aplica-se apenas a grafos onde o grau de cada vértice atinge um limite mínimo específico.

1.2.2 TEOREMA DE ORE

O Teorema de Ore amplia a condição de Dirac ao considerar pares de vértices não adjacentes. Segundo esse teorema, se em um grafo simples com $n \geq 3$ vértices a soma dos

graus de cada par de vértices não adjacentes u e v satisfaz $d(u) + d(v) \geq n$, então o grafo é hamiltoniano. Ao incluir pares de vértices não conectados diretamente, o Teorema de Ore apresenta uma condição menos restritiva, aplicando-se a uma gama mais ampla de grafos e oferecendo uma abordagem mais geral para verificar a hamiltonianidade.

1.2.3 TEOREMA DE BONDY-CHVÁTAL

O Teorema de Bondy-Chvátal propõe uma abordagem iterativa para verificar a hamiltonianidade, conhecida como operação de fechamento do grafo. Esse teorema afirma que um grafo G com n vértices é hamiltoniano se e somente se seu fechamento G^* for hamiltoniano, onde G^* é obtido ao adicionar arestas entre pares de vértices não adjacentes u e v sempre que $d(u) + d(v) \geq n$. Essa condição permite construir um grafo equivalente em termos de hamiltonianidade ao adicionar conexões entre pares de vértices conforme necessário, simplificando o problema ao permitir uma verificação gradativa.

1.2.4 MODELO CÍCLICO-ALEATÓRIO

No modelo cíclico-aleatório, inicia-se com um ciclo simples de N vértices, o que garante que o grafo possui um ciclo hamiltoniano desde o início. Em seguida, são adicionadas arestas aleatórias entre pares de vértices não adjacentes com uma probabilidade p. Esse modelo permite que o grafo mantenha um ciclo básico enquanto aumenta gradualmente a conectividade, possibilitando a análise da transição de grafos com hamiltonianidade garantida para grafos mais complexos e densamente conectados.

1.2.5 MODELO DE ERDOS-RENYI

O modelo de Erdos-Renyi, proposto por Paul Erdős e Alfréd Rényi, é um dos modelos mais tradicionais para a geração de grafos aleatórios. Nesse modelo, cada par de vértices em um grafo recebe uma aresta com uma probabilidade fixa p, resultando em grafos com distribuição de conexões aleatória e sem uma estrutura cíclica inicial. Esse modelo é amplamente utilizado para estudar propriedades estatísticas de grafos e para modelar redes complexas onde as conexões entre vértices ocorrem de maneira independente e sem padrões definidos.

1.3 OBJETIVOS

Este estudo possui os seguintes objetivos principais:

1. Explorar a Aplicabilidade dos Teoremas de Dirac, Ore e Bondy-Chvátal: Aprofundar a compreensão dos critérios de hamiltonianidade em grafos aleatórios, identificando em que circunstâncias cada teorema é aplicável.

2. Desenvolver Testes Computacionais para Verificação da Hamiltonianidade: Implementar algoritmos que verifiquem a conformidade de grafos com os três teoremas, de modo a avaliar a eficiência de cada critério como indicador de hamiltonianidade.

3. Comparar Modelos de Grafos Aleatórios: Examinar a eficácia dos modelos cíclico-aleatório e Erdos-Renyi na produção de grafos que satisfaçam os critérios de hamiltonianidade, e comparar as taxas de grafos hamiltonianos produzidos por cada modelo para diferentes valores de N e p.

Este estudo pretende fornecer uma visão prática e teórica sobre os critérios hamiltonianos, contribuindo para o entendimento de sua aplicabilidade e oferecendo uma base empírica para o uso desses critérios na análise e simulação de redes complexas.

2 IMPLEMENTAÇÃO

A implementação deste projeto consiste na criação de um algoritmo para verificar se um grafo satisfaz os três teoremas hamiltonianos: Dirac, Ore e Bondy-Chvátal. Além disso, foi desenvolvido um script com o Jupyter Notebook (Python) para automatizar a geração e análise dos resultados.

2.1 CÓDIGO PRINCIPAL

O código principal, escrito em linguagem C, está organizado em várias seções, cada uma dedicada a funcionalidades específicas relacionadas à geração, manipulação e análise de grafos, com relação aos ciclos hamiltonianos.

2.1.1 PRIMEIRA REGIÃO (VETORES)

```c
int *criar_vetor(int n) {
    int *vetor = (int *)calloc(n, sizeof(int));
    return vetor;
}

void troca_lugares(int *vetor, int num1, int num2) {
    int aux = vetor[num1];
    vetor[num1] = vetor[num2];
    vetor[num2] = aux;
}

void liberar_vetor(int *vetor) {
    free(vetor);
}
```

A primeira região, **vetores**, contém funções para gerenciamento de array dinâmico (*vetor*). Inclui funções para criar um vetor (*criar_vetor*), trocar elementos dentro de um vetor (*troca_lugares*) e liberar a memória alocada para um vetor (*liberar_vetor*).

2.1.2 SEGUNDA REGIÃO (FILAS)

```c
typedef struct dado {
    int dado;
    struct dado *proximo;
} DADO;

typedef struct {
    DADO *entrada;
    DADO *saida;
} Fila;

typedef Fila *p_fila;

p_fila criar_fila() {
    p_fila f = malloc(sizeof(Fila));
    f->entrada = NULL;
    f->saida = NULL;
    return f;
}

int fila_vazia(p_fila f) {
    return (f->saida == NULL);
}

void esvaziar_fila(p_fila f) {
    DADO *aux;
    while (!fila_vazia(f)) {
        aux = f->saida;
        f->saida = f->saida->proximo;
        free(aux);
    }
    f->entrada = NULL;
}

void liberar_fila(p_fila f) {
    DADO *aux;
    while (!fila_vazia(f)) {
        aux = f->saida;
        f->saida = f->saida->proximo;
```

```
39          free(aux);
40      }
41      free(f);
42  }
43
44  void enfileirar(p_fila f, int k) {
45      DADO *aux = malloc(sizeof(DADO));
46      aux->dado = k;
47      aux->proximo = NULL;
48      if (!fila_vazia(f)) {
49          f->entrada->proximo = aux;
50          f->entrada = aux;
51      } else {
52          f->entrada = aux;
53          f->saida = aux;
54      }
55  }
56
57  int desenfileirar(p_fila f) {
58      DADO *aux;
59      int i;
60      if (!fila_vazia(f)) {
61          aux = f->saida;
62          if (f->entrada != f->saida) {
63              f->saida = f->saida->proximo;
64          } else {
65              f->entrada = NULL;
66              f->saida = NULL;
67          }
68          i = aux->dado;
69          free(aux);
70          return i;
71      }
72      return INT_MIN;
73  }
74
75  bool remover_item(p_fila f, int k) {
76      DADO *atual = f->saida;
```

```
77      DADO *anterior = NULL;
78      while (atual != NULL) {
79          if (atual->dado == k) {
80              if (anterior != NULL) {
81                  anterior->proximo = atual->proximo;
82                  free(atual);
83              } else {
84                  desenfileirar(f);
85              }
86              return true;
87          }
88          anterior = atual;
89          atual = atual->proximo;
90      }
91      return false;
92  }
```

A segunda região, **filas**, define e gerencia uma estrutura de dados de fila $(FIFO)$. Inclui a definição da estrutura $(DADO)$, que representa um elemento na fila, e a estrutura Fila, que representa a própria fila. As funções nesta região incluem criar uma fila $(criar\_fila)$, verificar se uma fila está vazia $(fila\_vazia)$, esvaziar uma fila $(esvaziar\_fila)$, liberar a fila $(liberar\_fila)$, enfileirar $(enfileirar)$, desenfileirar $(desenfileirar)$ e remover um item específico da fila $(remover\_item)$.

2.1.3 TERCEIRA REGIÃO (MATRIZES)

```
1   typedef struct matricial {
2       int n; // linhas
3       int **matriz; // ponteiro para matriz
4       int *grau; // ponteiro para vetor com o grau dos vértices
5       p_fila *lista_n_adjascencia; // ponteiro para vetor da lista de
        ↪ não adjascência
6   } Matriz;
7
8   Matriz *inicializar_matriz(int qtd_vertices) {
9       Matriz *matricial = (Matriz *)malloc(sizeof(Matriz));
10      int **matriz_adjascencia = (int **)malloc(qtd_vertices *
        ↪ sizeof(int *));
11      for (int i = 0; i < qtd_vertices; i++) {
```

```
12          matriz_adjascencia[i] = (int *)malloc(qtd_vertices *
            ↪  sizeof(int));
13      }
14      matricial->n = qtd_vertices;
15      matricial->matriz = matriz_adjascencia;
16      matricial->lista_n_adjascencia = malloc(sizeof(p_fila) *
        ↪  qtd_vertices);
17      for (int i = 0; i < qtd_vertices; i++) {
18          matricial->lista_n_adjascencia[i] = criar_fila();
19      }
20      matricial->grau = criar_vetor(qtd_vertices);
21      return matricial;
22  }
23
24  Matriz *copiar_matriz(Matriz *matricial) {
25      Matriz *copia = inicializar_matriz(matricial->n);
26      DADO *aux;
27      for (int i = 0; i < copia->n; i++) {
28          copia->grau[i] = matricial->grau[i];
29          aux = matricial->lista_n_adjascencia[i]->saida;
30          while (aux != NULL) {
31              enfileirar(copia->lista_n_adjascencia[i], aux->dado);
32              aux = aux->proximo;
33          }
34          for (int j = 0; j < copia->n; j++) {
35              copia->matriz[i][j] = matricial->matriz[i][j];
36          }
37      }
38      return copia;
39  }
40
41  void liberar_matriz(Matriz *matricial) {
42      for (int i = 0; i < matricial->n; i++) {
43          free(matricial->matriz[i]);
44          liberar_fila(matricial->lista_n_adjascencia[i]);
45      }
46      liberar_vetor(matricial->grau);
47      free(matricial->lista_n_adjascencia);
```

```
    free(matricial->matriz);
    free(matricial);
}
```

A terceira região, **matrizes**, lida com operações de matriz, especificamente para representar grafos. Ela define a estrutura Matriz, que inclui a matriz de adjacência, o grau de vértices e uma lista de vértices não adjacentes. As funções nesta região incluem inicializar uma matriz (*inicializar_matriz*), copiar uma matriz (*copiar_matriz*) e liberar a memória alocada para uma matriz (*liberar_matriz*).

2.1.4 QUARTA REGIÃO (GRAFOS)

```
typedef Matriz *Grafo;

void gerar_grafo(Grafo grafo, bool orientado, float probabilidade) {
    int porcentagem = (int)(100 * probabilidade);
    if (!orientado) { // garante espelhamento
        for (int i = 0; i < grafo->n; i++) {
            for (int j = i; j < grafo->n; j++) {
                if (i != j) { // evitar ligacoes proprias
                    grafo->matriz[i][j] = (rand() % 100 <
                    ↪  porcentagem) ? 1 : 0; // pesos entre 1 e 0
                    ↪  (tem ou nao tem)
                    grafo->matriz[j][i] = grafo->matriz[i][j];
                    if (grafo->matriz[i][j]) {
                        grafo->grau[i]++;
                        grafo->grau[j]++;
                    } else {
                        enfileirar(grafo->lista_n_adjascencia[i], j);
                        enfileirar(grafo->lista_n_adjascencia[j], i);
                    }
                } else {
                    grafo->matriz[i][j] = 0; // falso para quando for
                    ↪  a diagonal principal
                }
            }
        }
    } else {
        for (int i = 0; i < grafo->n; i++) {
```

```
25              for (int j = 0; j < grafo->n; j++) {
26                  if (i != j) { // evitar ligacoes proprias
27                      grafo->matriz[i][j] = (rand() % 100 <
                         ↪  porcentagem) ? 1 : 0; // pesos entre 1 e 0
                         ↪  (tem ou nao tem)
28                      if (grafo->matriz[i][j]) {
29                          grafo->grau[i]++;
30                      } else {
31                          enfileirar(grafo->lista_n_adjascencia[i], j);
32                      }
33                  } else {
34                      grafo->matriz[i][j] = 0; // falso para quando for
                         ↪  a diagonal principal
35                  }
36              }
37          }
38      }
39  }
40
41  void gerar_grafo_hamiltoniano(Grafo grafo, bool orientado, float
     ↪  probabilidade) {
42      gerar_grafo(grafo, orientado, probabilidade);
43      int *ciclo = criar_vetor(grafo->n);
44      for (int i = 0; i < grafo->n; i++) {
45          ciclo[i] = i;
46      }
47      if (!orientado) {
48          troca_lugares(ciclo, 0, rand() % (grafo->n));
49          for (int i = 1; i < grafo->n; i++) {
50              troca_lugares(ciclo, i, rand() % (grafo->n - i) + i);
51              if (!(grafo->matriz[ciclo[i - 1]][ciclo[i]])) {
52                  grafo->matriz[ciclo[i - 1]][ciclo[i]] = 1;
53                  grafo->matriz[ciclo[i]][ciclo[i - 1]] =
                     ↪  grafo->matriz[ciclo[i - 1]][ciclo[i]];
54                  grafo->grau[ciclo[i - 1]]++;
55                  grafo->grau[ciclo[i]]++;
56                  remover_item(grafo->lista_n_adjascencia[ciclo[i - 1]],
                     ↪  ciclo[i]);
```

```
57                  remover_item(grafo->lista_n_adjascencia[ciclo[i]],
                        ↪  ciclo[i - 1]);
58              }
59          }
60          if (!(grafo->matriz[ciclo[grafo->n - 1]][ciclo[0]])) {
61              grafo->matriz[ciclo[grafo->n - 1]][ciclo[0]] = 1;
62              grafo->matriz[ciclo[0]][ciclo[grafo->n - 1]] =
                    ↪  grafo->matriz[ciclo[grafo->n - 1]][ciclo[0]];
63              grafo->grau[ciclo[grafo->n - 1]]++;
64              grafo->grau[ciclo[0]]++;
65              remover_item(grafo->lista_n_adjascencia[ciclo[grafo->n -
                    ↪  1]], ciclo[0]);
66              remover_item(grafo->lista_n_adjascencia[ciclo[0]],
                    ↪  ciclo[grafo->n - 1]);
67          }
68      }
69      liberar_vetor(ciclo);
70  }
71
72  bool dirac(Grafo grafo) {
73
74      for (int i = 0; i < grafo->n; i++) {
75          if (grafo->grau[i] < grafo->n / 2)
76              return false;
77      }
78      return true;
79  }
80
81  bool ore(Grafo grafo) {
82      DADO *aux;
83      for (int i = 0; i < grafo->n; i++) {
84          aux = grafo->lista_n_adjascencia[i]->saida;
85          while (aux != NULL) {
86              if (grafo->grau[i] + grafo->grau[aux->dado] < grafo->n)
87                  return false;
88              aux = aux->proximo;
89          }
90      }
```

```
91      return true;
92  }
93
94  Grafo fecho_hamiltoniano(Grafo grafo) {
95      Grafo fecho_hamiltoniano = copiar_matriz(grafo);
96      int aux;
97      for (int i = 0; i < fecho_hamiltoniano->n; i++) {
98          while (fecho_hamiltoniano->lista_n_adjascencia[i]->saida !=
            ↪  NULL) {
99              if (ore(fecho_hamiltoniano))
100                 return fecho_hamiltoniano;
101             aux = desenfileirar(fecho_hamiltoniano->
                ↪  lista_n_adjascencia[i]);
102             fecho_hamiltoniano->matriz[i][aux] = 1;
103             fecho_hamiltoniano->matriz[aux][i] =
                ↪  fecho_hamiltoniano->matriz[i][aux];
104             fecho_hamiltoniano->grau[i]++;
105             fecho_hamiltoniano->grau[aux]++;
106         }
107     }
108     liberar_matriz(fecho_hamiltoniano);
109     return NULL;
110 }
111
112 bool bondy_chvatal(Grafo fecho) {
113     if (fecho == NULL)
114         return false;
115     for (int i = 0; i < fecho->n; i++) {
116         for (int j = i + 1; j < fecho->n; i++) {
117
118             if (!(fecho->matriz[i][j])) {
119                 return false;
120             }
121         }
122     }
123     return true;
124 }
```

A quarta região, **grafo**, foca em operações específicas de grafos. Inclui fun-

ções para gerar um gráfico aleatório ($gerar\_grafo$), gerar um gráfico hamiltoniano ($gerar\_grafo\_hamiltoniano$), verificar se um gráfico satisfaz o teorema de Dirac ($dirac$), verificar se um gráfico satisfaz o teorema de Ore (ore), gerar um fechamento hamiltoniano de um gráfico ($fecho\_hamiltoniano$) e verificar se um gráfico satisfaz o teorema de Bondy-Chvátal ($bondy\_chvatal$). Além disso, inclui funções para imprimir o gráfico em um arquivo ($imprimir\_grafo\_arquivo$) e visualizar o gráfico e suas informações ($visualizar\_grafo\_e\_informacoes$).

2.1.5 REGIÃO FINAL (MAIN)

```c
int main() {
    int opcao, n, salvar;
    Grafo grafo=NULL, grafo=NULL;
    float probabilidade;
    static int orientado = false;
    while (true) {
        srand(time(NULL)); // garantir boa aleatorizacao
        printf("+-------------------Menu---------------------+\n");
        printf("Escolha uma opcao:\n");
        printf("1. Gerar Grafo\n");
        printf("2. Gerar Grafo Hamiltoniano\n");
        printf("3. Verificar Teorema de Dirac\n");
        printf("4. Verificar Teorema de Ore\n");
        printf("5. Verificar Bondy-Chvatal\n");
        printf("6. Gerar Fecho Hamiltoniano\n");
        printf("7. Visualizar Grafo\n");
        printf("8. Visualizar o Fecho Hamiltoniano\n");
        printf("0. Sair\n");
        printf("+--------------------------------------------+\n");
        scanf("%d", &opcao);
        switch (opcao) {
        case 1:
            if (grafo) {
                liberar_matriz(grafo);
                printf("Grafo anterior existente foi excluido!\n");
            }
            printf("Digite o numero de vertices: ");
            scanf("%d", &n);
            printf("Digite a probabilidade de aresta (0 a 1): ");
```

```
30              scanf("%f", &probabilidade);
31              grafo = inicializar_matriz(n);
32              gerar_grafo(grafo, orientado, probabilidade);
33              printf("Grafo gerado com sucesso!\n");
34              break;
35          case 2:
36              ...
37              printf("Digite o numero de vertices: ");
38              scanf("%d", &n);
39              printf("Digite a probabilidade de aresta (0 a 1): ");
40              scanf("%f", &probabilidade);
41              grafo = inicializar_matriz(n);
42              gerar_grafo_hamiltoniano(grafo, orientado, probabilidade);
43              printf("Grafo Hamiltoniano gerado com sucesso!\n");
44              break;
45          case 3:
46              ...
47              if (dirac(grafo)) {
48                  printf("O grafo satisfaz o Teorema de Dirac.\n");
49              } else {
50                  printf("O grafo NAO satisfaz o Teorema de Dirac.\n");
51              }
52              break;
53          case 4:
54              ...
55              if (ore(grafo)) {
56                  printf("O grafo satisfaz o Teorema de Ore.\n");
57              } else {
58                  printf("O grafo NAO satisfaz o Teorema de Ore.\n");
59              }
60              break;
61          case 5:
62              ...
63              if (ore(grafo)) {
64                  fecho = grafo;
65              }
66              if (bondy_chvatal(fecho)) {
```

```
67                  printf("O fecho hamiltoniano satisfaz o Teorema de
                     ↪  Bondy-Chvatal.\n");
68              } else {
69                  printf("O fecho hamiltoniano NAO satisfaz o Teorema de
                     ↪  Bondy-Chvatal.\n");
70              }
71              break;
72          case 6:
73              ...
74              if (ore(grafo)) {
75                  printf("O grafo já possui um fecho hamiltoniano.\n");
76              } else {
77                  fecho = fecho_hamiltoniano(grafo);
78                  if (fecho != NULL) {
79                      printf("Um fecho hamiltoniano para o grafo foi
                         ↪  gerado!\n");
80                  } else {
81                      printf("Não foi possível modificar o grafo.\n");
82                  }
83              }
84              break;
85          case 7:
86              ...
87          case 8:
88              if (ore(grafo)) {
89                  printf("O próprio grafo já é um fecho
                     ↪  hamiltoniano.\n");
90                  break;
91              }
92              ...
93          ...
94          }
95      }
96  }
```

A região final, **main**, contém a função (*main*) que fornece uma interface orientada a menu para o usuário interagir com o programa. O usuário pode gerar grafos, verificar vários teoremas, gerar fechamentos hamiltonianos e visualizar os grafos. A função principal manipula a entrada do usuário, chama as funções apropriadas com base na escolha do

usuário e garante o gerenciamento de memória adequado ao liberar recursos alocados antes de sair.

2.2 AUTOMAÇÃO PARA TESTES

Foi desenvolvido um Jupyter Notebook para apoiar a análise exploratória e visualização de dados, utilizando as bibliotecas `pandas`, `matplotlib` e `seaborn`, amplamente empregadas em pesquisa acadêmica e ciência de dados. Este documento descreve cada etapa metodológica, desde a importação e tratamento dos dados até a criação e análise de gráficos, garantindo um fluxo estruturado e intuitivo para o usuário.

2.2.1 IMPORTAÇÃO DE BIBLIOTECAS

As bibliotecas fundamentais para a manipulação e visualização de dados foram importadas na seção inicial do notebook. Dentre elas:

- **pandas**: utilizado para a manipulação de dados tabulares e preparação do dataset.
- **matplotlib**: responsável pela criação de gráficos básicos de forma altamente configurável.
- **seaborn**: empregado para visualizações mais estilizadas e estatísticas, complementando o `matplotlib` com gráficos mais detalhados e coloridos.

2.2.2 CARREGAMENTO E ESTRUTURAÇÃO DOS DADOS

A etapa de carregamento de dados é realizada através da função $(read\_csv)$ da biblioteca `pandas`, que permite a leitura de arquivos CSV, principal formato dos conjuntos de dados analisados. O notebook fornece opções para importação tanto local quanto de fontes online, garantindo flexibilidade no processo de obtenção de dados. Após a importação, são apresentados procedimentos para verificar a consistência estrutural do dataset (ex.: tipos de dados e colunas).

2.2.3 TRATAMENTO E LIMPEZA DE DADOS

Para assegurar a qualidade dos dados analisados, o notebook inclui etapas de pré-processamento, onde são tratados valores ausentes e duplicados. A estrutura dos dados é ajustada, quando necessário, para manter a coerência e precisão das informações a serem visualizadas. Este tratamento inclui filtragens por critérios específicos e transformações que permitem a análise das variáveis relevantes.

2.2.4 CONSTRUÇÃO E CONFIGURAÇÃO DOS GRÁFICOS

A seção de criação de gráficos aborda a implementação de diferentes tipos de visualizações, ajustados para evidenciar relações específicas entre as variáveis. Os gráficos principais utilizados no notebook incluem:

- **Gráficos de Linha**: aplicados na visualização de séries temporais, permitindo a análise de tendências e variações ao longo do tempo.
- **Gráficos de Dispersão**: utilizados para investigar correlações entre duas variáveis contínuas, sendo particularmente úteis na observação de padrões de associação.
- **Gráficos de Barras**: empregados para comparações de valores entre categorias discretas, facilitando a visualização de frequências ou magnitudes. Cada gráfico é configurado com elementos visuais essenciais (ex.: cores, rótulos de eixos e legendas) para garantir uma interpretação clara e objetiva dos dados. A integração entre `matplotlib` e `seaborn` permite personalizar e aprimorar a apresentação visual, tornando os gráficos mais intuitivos.

2.2.5 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Para cada gráfico gerado, o notebook inclui uma seção interpretativa, onde os resultados são discutidos de forma a identificar padrões e observações relevantes. As análises destacam possíveis tendências, sazonalidades e relações entre variáveis, oferecendo subsídios importantes para a validação de hipóteses iniciais e possíveis estudos subsequentes.

3 RESULTADOS E DISCUSSÃO

Nesta seção, apresentamos os resultados obtidos a partir da análise dos três modelos de grafos gerados: o **grafo de fecho hamiltoniano**, o **grafo de Erdos-Renyi** e o **grafo hamiltoniano inicial**. Para cada grafo, foram aplicados os testes dos teoremas de Dirac, Ore e Bondy-Chvátal. A análise foi conduzida para diferentes valores de (N) (número de vértices) e (p) (probabilidade de conexão), com os resultados sintetizados em gráficos e tabelas.

3.1 ANÁLISE DOS DADOS

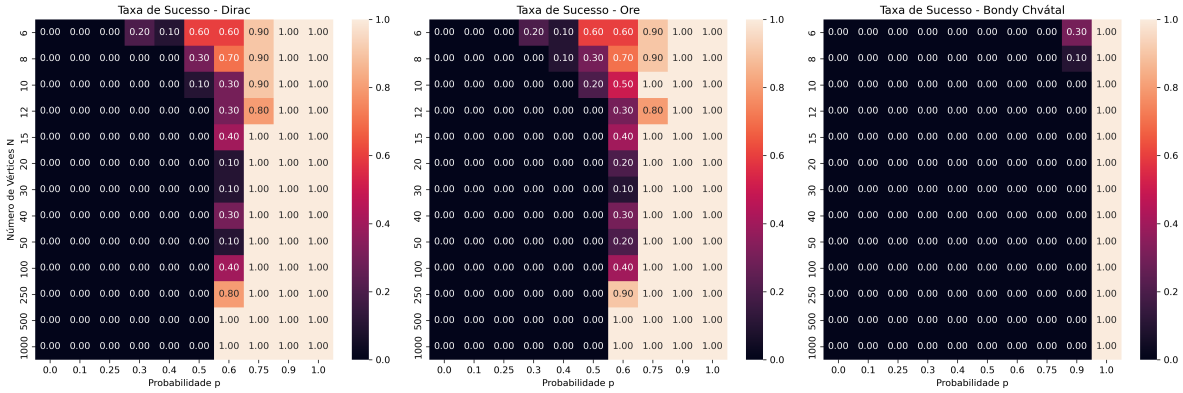

Figura 1 – Heatmap do grafo de fecho

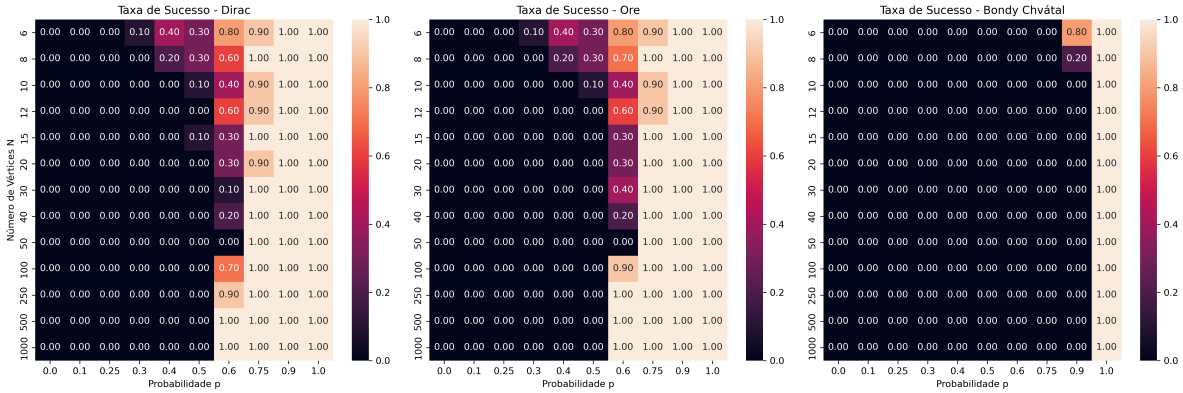

Figura 2 – Heatmap do grafo hamiltoniano

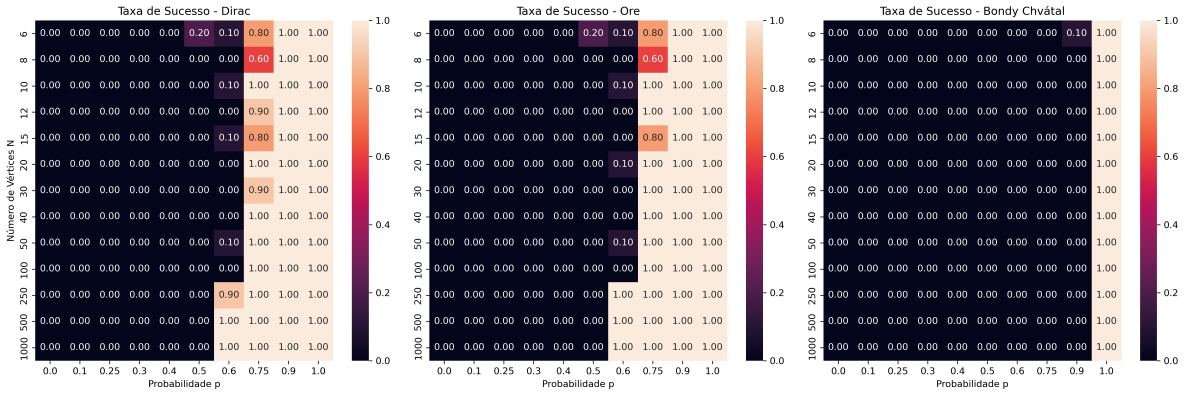

Figura 3 – Heatmap do grafo normal (Erdos-Renyi)

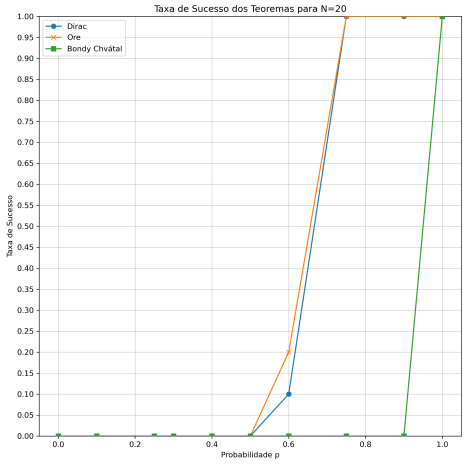

Figura 4 – Taxa de sucesso no grafo de fecho, variando N

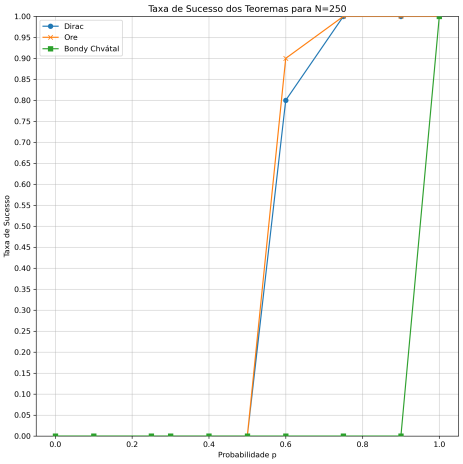

Figura 5 – Taxa de sucesso no grafo de fecho, variando N

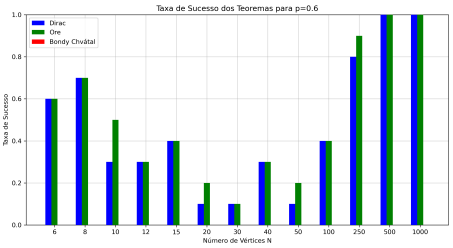

Figura 6 – Taxa de sucesso no grafo de fecho, variando p

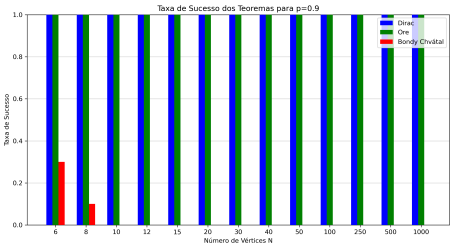

Figura 7 – Taxa de sucesso no grafo de fecho, variando p

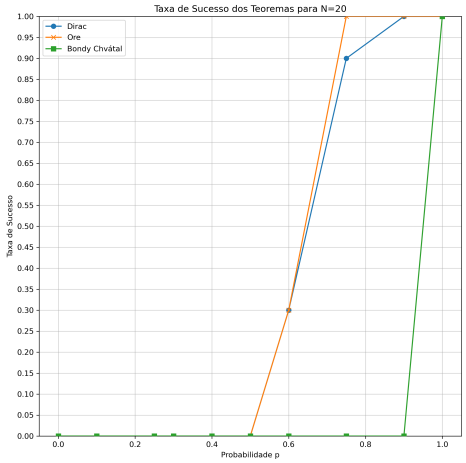

Figura 8 – Taxa de sucesso no grafo hamiltoniano, variando N

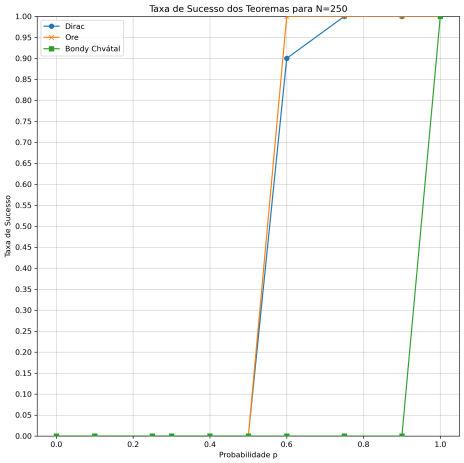

Figura 9 – Taxa de sucesso no grafo hamiltoniano, variando N

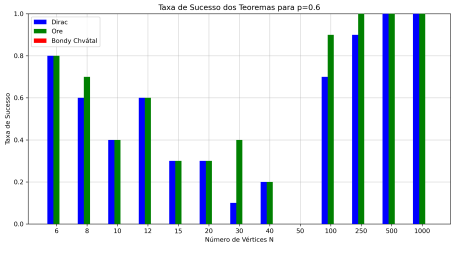

Figura 10 – Taxa de sucesso no grafo hamiltoniano, variando p

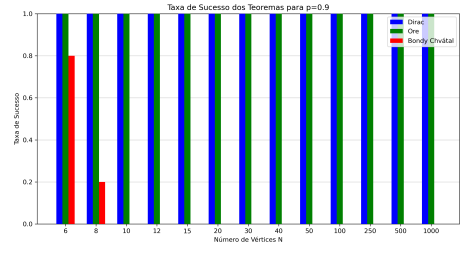

Figura 11 – Taxa de sucesso no grafo hamiltoniano, variando p

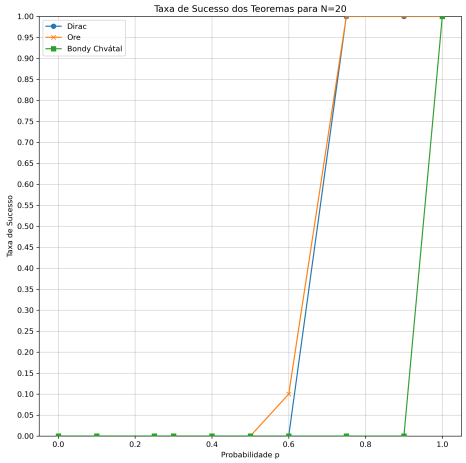

Figura 12 – Taxa de sucesso no grafo normal, variando N

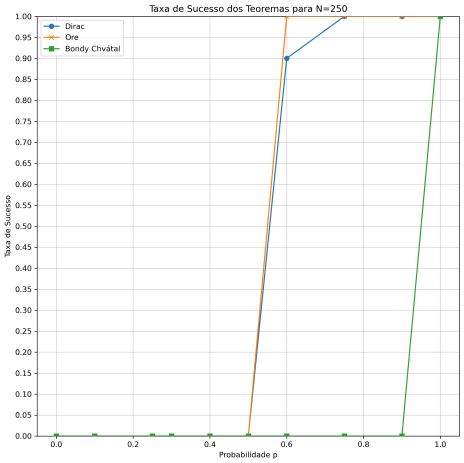

Figura 13 – Taxa de sucesso no grafo normal, variando N

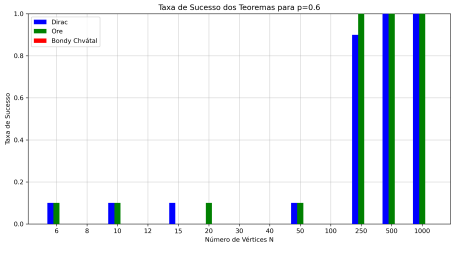

Figura 14 – Taxa de sucesso no grafo normal, variando p

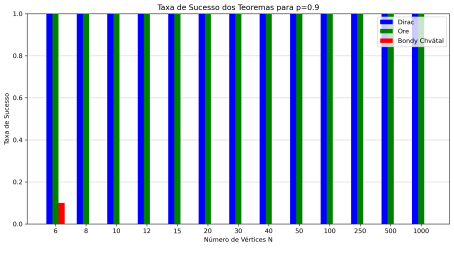

Figura 15 – Taxa de sucesso no grafo normal, variando p

3.2 DISCUSSÃO DOS RESULTADOS

A análise dos resultados obtidos a partir dos testes aplicados aos diferentes modelos de grafos (fecho hamiltoniano, Erdos-Renyi e hamiltoniano inicial) permitiu identificar padrões e tendências importantes, que foram complementadas pelas visualizações gráficas.

Os gráficos de linha indicaram uma clara correlação positiva entre a probabilidade de conexão (p) e a taxa de conformidade aos teoremas de Dirac e Ore. À medida que p aumenta, observa-se um aumento consistente na taxa de sucesso, especialmente para o modelo de fecho hamiltoniano. Este comportamento era esperado, uma vez que o aumento da densidade de arestas tende a aumentar a conectividade do grafo, facilitando a formação de ciclos hamiltonianos.

Os heatmaps fornecem uma representação visual da densidade de conexões para cada modelo. O grafo hamiltoniano apresenta uma alta densidade, especialmente nas regiões centrais, refletindo sua estrutura inicial que garante a existência de um ciclo hamiltoniano. Em contraste, o grafo de Erdos-Renyi, com sua distribuição de arestas puramente aleatória, exibe uma densidade menos uniforme, o que explica a menor conformidade observada nos testes. O grafo de fecho hamiltoniano, por sua vez, mostra um aumento na densidade nas regiões de fechamento, indicando a eficácia da operação de fechamento em aumentar a conectividade.

O grafo hamiltoniano inicial apresentou a maior taxa de conformidade aos teoremas, especialmente aos de Dirac e Ore. Esta superioridade pode ser atribuída à sua construção, que garante um ciclo desde o início, criando uma estrutura propícia para satisfazer os critérios de hamiltonianidade. No entanto, é importante notar que este modelo serve mais como um ”caso ideal”e menos como uma representação realista de grafos aleatórios.

O modelo de Erdos-Renyi, apesar de ser amplamente utilizado para simular redes complexas, teve o pior desempenho em termos de conformidade. A ausência de uma estrutura inicial de ciclo e a distribuição aleatória de arestas resultam em grafos menos densos e conectados, o que dificulta a formação de ciclos hamiltonianos, especialmente para valores baixos de p. Estes resultados indicam que o modelo de Erdos-Renyi pode não ser o mais adequado para a geração de grafos hamiltonianos em estudos onde a conectividade é um fator crítico.

O grafo de fecho hamiltoniano, que aplica uma operação de fechamento baseada no Teorema de Ore, apresentou um desempenho intermediário. A operação de fechamento aumentou a conectividade e melhorou a taxa de conformidade, especialmente para valores moderados de p. No entanto, a conformidade com o Teorema de Bondy-Chvátal permaneceu baixa, sugerindo que, embora a operação de fechamento seja eficaz, ela não é suficiente para garantir hamiltonianidade em grafos menos densos.

Os resultados indicam que os Teoremas de Dirac e Ore são menos rigorosos e mais facilmente satisfeitos em grafos com maior densidade de conexões. O Teorema de Bondy-Chvátal, que requer uma análise mais aprofundada da estrutura do grafo, apresentou a menor taxa de conformidade, especialmente para o modelo de Erdos-Renyi. Este comportamento pode ser explicado pelo fato de que este teorema depende de operações iterativas de fechamento, que exigem uma alta conectividade para serem eficazes.

O Teorema de Dirac mostrou-se particularmente eficaz para grafos com alta densidade, como o grafo hamiltoniano inicial, onde todos os vértices possuem grau mínimo elevado. O Teorema de Ore, ao considerar pares de vértices não adjacentes, foi mais abrangente e apresentou melhores resultados em grafos de fecho, onde as arestas adicionais aumentam a conectividade.

Os achados deste estudo sugerem que a escolha do modelo de geração de grafos tem um impacto significativo na conformidade aos critérios de hamiltonianidade. O modelo cíclico-aleatório e o grafo de fecho são opções promissoras para gerar grafos com alta probabilidade de satisfazer os teoremas de Dirac e Ore, especialmente quando a probabilidade de conexão é alta. No entanto, a baixa conformidade ao Teorema de Bondy-Chvátal indica que modelos adicionais ou modificações nos algoritmos de fechamento podem ser necessários para aumentar a probabilidade de satisfazer este critério.

Para estudos futuros, sugere-se a investigação de outros modelos de geração de grafos, como o modelo de Watts-Strogatz ou o modelo Barabási-Albert, que incorporam propriedades de redes reais, como a formação de "pequenos mundos"e conectividade preferencial. Além disso, o desenvolvimento de algoritmos de fechamento mais sofisticados pode contribuir para melhorar a conformidade ao Teorema de Bondy-Chvátal, especialmente em grafos com baixa densidade.

Em resumo, os gráficos e análises apresentadas fornecem uma visão abrangente da conformidade dos diferentes modelos aos teoremas hamiltonianos. O estudo destaca a importância da densidade de conexões e da estrutura inicial do grafo na formação de ciclos hamiltonianos, oferecendo insights valiosos para a modelagem e análise de redes complexas.

4 CONCLUSÃO

Este trabalho realizou uma análise abrangente sobre a hamiltonianidade de grafos aleatórios, aplicando três critérios clássicos: os teoremas de Dirac, Ore e Bondy-Chvátal. Através da implementação computacional e testes empíricos em diferentes modelos de grafos, exploramos como cada teorema se comporta frente a estruturas com variadas densidades e conectividades.

Os resultados mostraram que o **Teorema de Dirac** apresentou maior eficácia em grafos com alta densidade de arestas, especialmente no modelo cíclico-aleatório, onde todos os vértices possuem, desde o início, um grau relativamente elevado. Este teorema, ao exigir que o grau de cada vértice seja no mínimo a metade do número total de vértices, oferece um critério forte e direto para identificar a hamiltonianidade, mas é limitado a grafos densos e não é adequado para grafos com menor conectividade.

O **Teorema de Ore**, por sua vez, ampliou a aplicabilidade ao considerar a soma dos graus de pares de vértices não adjacentes. Esta abordagem se mostrou particularmente eficaz no modelo de fecho hamiltoniano, onde a operação de fechamento incrementa a conectividade do grafo, aproximando-o das condições necessárias para satisfazer o teorema. No entanto, em grafos gerados pelo modelo de Erdos-Renyi, a aleatoriedade da distribuição de arestas resultou em uma taxa de conformidade mais baixa, indicindicando que a estrutura inicial do grafo é crucial para satisfazer o critério proposto por Ore. No modelo de Erdos-Renyi, a ausência de uma estrutura cíclica e a distribuição uniforme de arestas tornam menos provável que pares de vértices não adjacentes tenham graus suficientemente altos para cumprir a condição $d(u) + d(v) \geq n$. Isso destaca a importância de uma maior densidade e conectividade para alcançar conformidade com o Teorema de Ore, especialmente em grafos aleatórios, onde as arestas não seguem um padrão estruturado.

O **Teorema de Bondy-Chvátal** foi o critério mais rigoroso entre os três analisados. Ao exigir uma sequência iterativa de fechamento até que todos os pares de vértices não adjacentes satisfaçam a condição de soma dos graus, este teorema apresentou a menor taxa de conformidade, especialmente em grafos esparsos. Este resultado reflete a complexidade intrínseca do critério, que depende fortemente da densidade e da conectividade global do grafo. Assim, este teorema se mostrou mais adequado para grafos densos e bem conectados, onde a adição de arestas no processo de fechamento é mais eficaz em criar o ciclo hamiltoniano desejado. Em grafos esparsos, a falta de conectividade suficiente torna o fechamento ineficaz, limitando a aplicabilidade prática deste teorema em contextos onde a densidade de arestas é baixa.

4.1 CONSIDERAÇÕES FINAIS

Em síntese, este trabalho contribuiu para o entendimento dos critérios de hamiltonianidade em grafos aleatórios, demonstrando como diferentes modelos e teoremas se comportam frente a variações na densidade e estrutura dos grafos. Embora existam limitações inerentes ao uso de modelos aleatórios e critérios clássicos, os resultados obtidos oferecem uma base sólida para futuras investigações e aplicações em redes complexas. Ao explorar novos modelos, algoritmos e aplicações práticas, a pesquisa nesta área pode evoluir significativamente, permitindo a identificação eficiente de ciclos hamiltonianos em grafos de grande escala e alta complexidade, o que é crucial para diversas áreas, como otimização de redes de transporte, análise de redes sociais e resolução de problemas logísticos complexos. Essas direções apontam para um campo fértil de estudo, onde avanços teóricos e práticos podem convergir para desenvolver métodos robustos e escaláveis, aprimorando a compreensão e a aplicabilidade dos conceitos de hamiltonianidade em cenários reais.

Este estudo fornece um ponto de partida valioso para aqueles que buscam compreender a complexa dinâmica da hamiltonianidade em grafos, abrindo caminho para novas descobertas e avanços na teoria dos grafos e suas aplicações.

REFERÊNCIAS

BONDY, J. A.; MURTY, U. S. R. **Graph Theory**. 1. ed. Berlin, Heidelberg: Springer, 2008. ISBN 978-1-84628-970-5. Disponível em: https://doi.org/10.1007/978-1-84628-970-5.

DIRAC, G. A. Some Theorems on Abstract Graphs. *Proceedings of the London Mathematical Society*, v. 3, n. 1, p. 69-81, 1952. DOI: https://doi.org/10.1112/plms/s3-2.1.69.

ORE, Oystein. Note on Hamilton Circuits. *The American Mathematical Monthly*, v. 67, n. 1, p. 55-58, 1960. DOI: https://doi.org/10.2307/2308931.

BOLLOBÁS, Béla. **Random Graphs**. 2. ed. Cambridge: Cambridge University Press, 2001. ISBN 978-0521797221. Disponível em: https://doi.org/10.1017/CBO9780511814068.

WATTS, Duncan J.; STROGATZ, Steven H. Collective Dynamics of 'Small-World' Networks. *Nature*, v. 393, p. 440-442, 1998. DOI: https://doi.org/10.1038/30918.

BARABÁSI, Albert-László; ALBERT, Réka. Emergence of Scaling in Random Networks. *Science*, v. 286, p. 509-512, 1999. DOI: https://doi.org/10.1126/science.286.5439.509.

WEST, Douglas B. **Introduction to Graph Theory**. 2. ed. Prentice Hall, 2001. ISBN 978-0130144003.

BONDY, J. A.; CHVÁTAL, V. A Method in Graph Theory. *Discrete Mathematics*, v. 15, p. 111-135, 1976. DOI: https://doi.org/10.1016/0012-365X(76)90030-2.