



# UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

2010.1

# OTIMIZAÇÃO EM GRAFOS:
# ÁRVORES GERADORAS COM RESTRIÇÕES

## PROPOSTA DE TRABALHO DE GRADUAÇÃO


| | | |
|---|---|---|
| **Aluno** | Artur Ribeiro de Aquino | {ara@cin.ufpe.br} |
| **Orientador** | Liliane Rose Benning Salgado | {liliane@cin.ufpe.br} |
| **Co-orientador** | Anjolina Grisi de Oliveira | {ago@cin.ufpe.br} |


09 de Julho de 2010


UNIVERSIDADE FEDERAL DE PERNAMBUCO (UFPE)

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CENTRO DE INFORMÁTICA (CIn)

Artur Ribeiro de Aquino

# *OTIMIZAÇÃO EM GRAFOS: ÁRVORES GERADORAS COM RESTRIÇÕES*

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como requisito para a conclusão do curso de Ciência da Computação.

Orientador: Liliane Rose Benning Salgado
Co-orientador: Anjolina Grisi de Oliveira

Recife, 09 de Julho de 2010

“E sem saber que era impossível, ele foi lá e fez.”

-- Jean Cocteau

# Agradecimentos

Agradeço, primeiramente a minha família que foi e continua sendo decisiva em toda a minha formação como indivíduo.

Ao meu primo, Ernesto, por ter me ajudado em algumas correções.

Aos orientadores e avaliadores, Liliane, Anjolina e Kátia, que tornaram este trabalho possível.

A todos que contribuíram direta ou indiretamente para a construção deste trabalho.

# Assinaturas

Este Trabalho de Graduação é resultado dos esforços do aluno Artur Ribeiro de Aquino, orientado pela professora Liliane Rose Benning Salgado, com o título “Otimização em Grafos: Árvores Geradoras com Restrições”. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

Co-orientador

_____

Professora Anjolina Grisi de Oliveira

Aluno

_____

Artur Ribeiro de Aquino

# Resumo

Grafos são muito usados para modelar problemas do mundo real. Porém, muitas vezes os grafos devem ser transformados em estruturas mais próximas do problema.

Árvores são grafos acíclicos e conexos. Elas podem ser obtidas através da simples remoção de arestas de um grafo conexo. Um tipo de árvore importante é a MST (Minimum Spanning Tree) que conecta todos os vértices do grafo sem ciclos. Embora ela seja fácil de ser encontrada pode-se adicionar restrições às essas MSTs tornando seu processo de obtenção bastante complicado. Esse tipo de problema é tão difícil que se torna mais viável a obtenção de uma aproximação do que da solução real.

Várias técnicas foram desenvolvidas buscando uma boa forma de se obter um resultado aproximado de MSTs com restrição. Neste trabalho, são apresentadas as abordagens VNS, EA e ACO como formas de buscar essa aproximação. Posteriormente elas são avaliadas comparativamente.

**Palavras-chave:** Grafos, Árvores, MST, Restrição, Aproximação, VNS, ACO, EA

# Abstract

Graphs are much used as an abstract model for real-world problems. Although, they often need to be transformed to another structure that is more close to an abstraction of the problem.

Trees are connected acyclic graphs. They can be obtained through edge removal. One important type of tree is MST (Minimum Spanning Tree) it connects all nodes of the graph with no cycles. Even though it is easy to get sometimes restriction are added to MSTs turning much more difficult for them to be obtained. This kind of problem is so hard that it is more feasible to get an approximation than the real solution.

Many techniques were developed trying to get a good way to obtain a good approximated result of constrained MSTs. At this work VNS, EA and ACO approaches are presented as a way to seek this approximation. Afterword they are comparatively evaluated.



**Keywords:** Graphs, Trees, MST, Restriction, Approximation, VNS, ACO, EA

# Abreviações

Ao longo deste trabalho são utilizadas várias abreviações. Para uma maior facilidade na leitura todas elas são listadas aqui. As abreviações são apresentadas juntamente com a expressão completa.

- MST(s): Minimum Spanning Tree(s)
- MCST: Minimum Cost Spanning Tree
- STP: Spanning Tree Protocol
- AGPM: Árvore Geradora de Peso Mínimo
- GRASP: Greedy Randomized Adaptative Search Procedure
- EA: Evolutionary Algorithm
- VNS: Variable Neighbourhood Search
- ACO: Ant Colony Optimization
- BDMST: Bounded Diameter Minimum Spanning Tree
- RTC: Randomized Tree Construction

# Conteúdo

# Introdução

Otimização Combinatória é a área da matemática que estuda métodos para encontrar pontos ótimos (máximo ou mínimo) de uma função definida sob um certo domínio. Nos problemas desta área o domínio é finito, e os pontos podem ser enumerados. Entretanto, o número de pontos do domínio pode ser muito grande, inviabilizando uma abordagem que enumerasse todas as possibilidades.

Diversos problemas práticos podem ser modelados como problemas de Otimização Combinatória. Tais aplicações práticas motivam o estudo de abordagens aproximadas para sua resolução, objeto principal de estudo neste projeto. A área de grafos tem sido tradicionalmente uma fonte de problemas interessantes de otimização combinatória, particularmente serão abordados os problemas de árvores geradoras com restrições.

O problema de encontrar a MCST (*Minimal Cost Spanning Tree)*, ou simplesmente MST (Minimal Spanning Tree), pode ser resolvido eficientemente utilizando algoritmos bem conhecidos na literatura, como o de Kruskal, Prim ou Boruvka. Porém, quando restrições adicionais (na sua altura, diâmetro, capacidade, grau dos vértices, entre outros) são exigidas nestas árvores, o problema pode dar origem a variantes NP-difíceis.

## 1.1 Motivação

Encontrar árvores geradoras com restrições é um problema bastante importante, e, por isso, vem sendo muito estudado. Seu uso se enquadra em vários problemas do mundo real, principalmente o campo está relacionado com a Internet ou com redes em geral.

Algumas possíveis aplicações diretas são em redes para realização de um *broadcast* não crítico (KAO, 2007). Dentre esses tipos de algoritmo distribuído, o mais famoso é o *Spanning Tree Protocol* (STP), que é um protocolo de rede da camada de enlace. Ele é usado para criar uma árvore geradora tendo os links existentes como o grafo de entrada e visa evitar excesso de *broadcasts*. Os links que não estiverem na MST são desabilitados deixando um único caminho ativo entre quaisquer dois nós da rede.

O entendimento e o estudo para melhorar a comunicação em redes são cada vez mais importantes. Com o aumento do número de computadores o gargalo do fluxo de comunicação passa a ser a forma pela qual os dados transitam. E essa forma que o trânsito dos dados toma sempre pode cair numa MST já que pode ser modelado através de minimização em árvores, onde cada computador seria um nó.

## 1.2 Objetivos

O objetivo deste trabalho de graduação é estudar, na área de otimização combinatória, árvores geradoras com restrições e as abordagens já existentes na literatura para a solução desse problema. Uma comparação entre algumas abordagens também será mostrada.

## 1.3 Organização do Documento

O documento está organizado em quatro seções principais. Esta serve como uma introdução do trabalho realizado, apresentando suas motivações e objetivos. A seção seguinte fala do problema das árvores geradoras. Porém, antes disso, passa por explicações necessárias para que o problema seja de fato entendido. Desse modo, são apresentados grafos e conceitos mais básicos e práticos sobre árvores geradoras. Após isso, é apresentada a classificação dos problemas segundo sua complexidade.

Seguindo para a próxima seção é mostrado como a adição de restrições às árvores geradoras é afetada segundo esse ponto de vista de complexidade. Nessa parte, também, estão alguns problemas de árvores geradoras com restrições e uma descrição breve de cada um deles. São descritas algumas aplicações para árvores geradoras com restrição.

Na última parte deste documento, encontram-se as abordagens conhecidas mais famosas para lidar com o problema em questão. E, por último, se encontra um capítulo com uma comparação de três dessas abordagens.

# 2. Contexto

Nesta seção, serão apresentados os conceitos básicos utilizados ao longo deste trabalho. Além disso, serão apresentadas alguns problemas mais básicos no que diz respeito a árvores geradoras e suas soluções.

Tendo explicado essa parte mais básica, tem-se início a parte da complexidade dos problemas e a implicação da adição de restrições às MSTs sob essa ótica da complexidade.

## 2.1 Grafos

O foco deste documento está nas MSTs, mas como elas sempre estão ligadas a um grafo é necessário saber também o que é um grafo para entender essa relação de dependência.

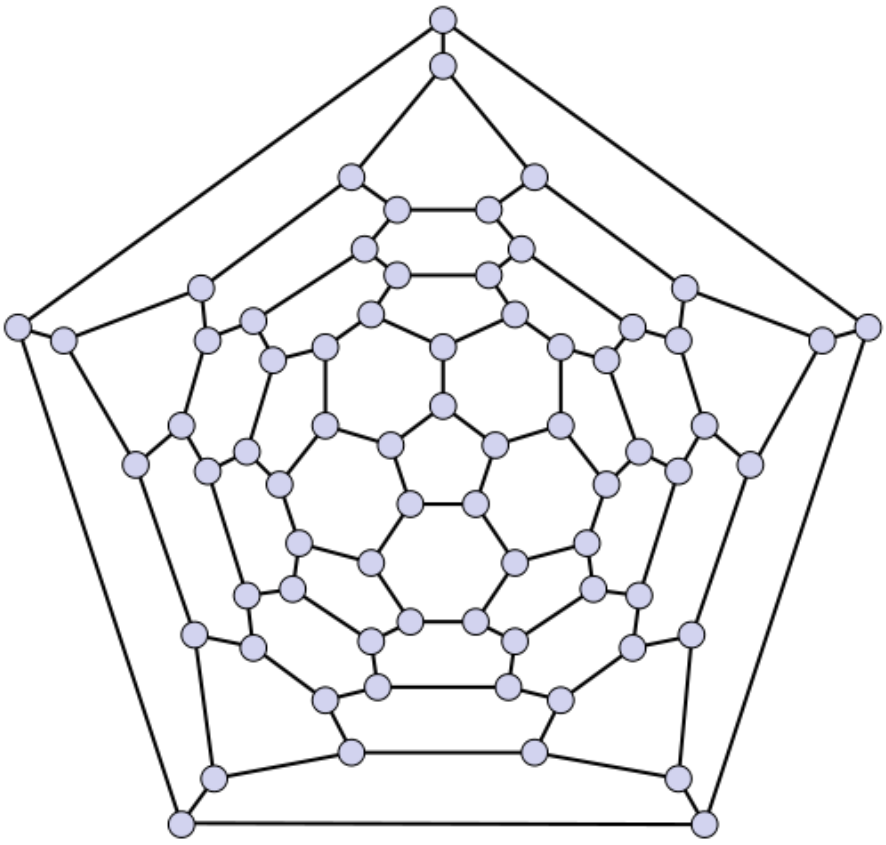


**Figura 1 - Grafo não-direcionado (Wikimedia, 2008)**

Um grafo não passa de uma representação abstrata em que se tem um conjunto de objetos e que alguns pares desses objetos estão conectados, como se pode observar na Figura 1. Esses objetos são chamados de vértices

ou nós e essas ligações entre dois vértices são chamadas de arestas. Uma definição mais formal de grafo, representado pela letra G, seria a de um par G = (V,E) em que V é o conjunto de nós e E é um conjunto de pares dos elementos de V, ou seja, o conjunto das arestas (CORMEN, LEISERSON, *et al.*, 2002). E pode ser constituído de pares ordenados, nesse caso o grafo é dito orientado ou direcionado, como o grafo da Figura 2. No entanto, caso E seja formado de pares não ordenados tem-se um grafo não-orientado ou não-direcionado, como o representado pela Figura 1. É válido dizer que em um grafo um nó pode ter uma ligação com ele mesmo.

Muitas definições para grafos orientados e não-orientados são idênticas, mas às vezes os termos têm apenas um significado ligeiramente diferente. Se (u,v) é uma aresta de um grafo direcionado, diz-se que ela é incidente do vértice u (sai de u) e incidente no vértice v (vai para ou entra em v), ou seja, é uma aresta que vai de u para v. Em um grafo não-direcionado diz-se que (u,v) é incidente nos vértices u e v (CORMEN, LEISERSON, *et al.*, 2002). Essa diferenciação ocorre porque o par (u,v), em um grafo não-direcionado, não é ordenado, representando, portanto, a mesma aresta do par (v,u). Logo, a aresta tanto vai de u para v quando de v para u.

Em uma festa, à medida que as pessoas vão aparecendo é natural que se cumprimentem. Um cumprimento muito comum é o aperto de mão. Ao modelar com grafos tem-se, por exemplo, um vértice para cada pessoa e uma aresta entre os vértices que representam as pessoas que apertaram as mãos. Não é possível apertar a mão de outro convidado sem que este aperte também a sua mão. Logo, para esse exemplo, não faz sentido um grafo direcionado, pois o aperto de mão é um ato recíproco. Assim sendo, sempre que dois nós tivessem uma relação seriam necessárias duas arestas direcionadas, que poderiam simplesmente serem substituídas por uma não-direcionada. Agora considere que ao invés do aperto de mão a relação a ser representada pelas arestas do grafo é se uma pessoa sabe o nome da outra. Um grafo não-direcionado desta vez não torna possível a modelagem, pois a relação não é recíproca. Uma pessoa saber o nome de outra não obriga essa outra a saber o nome dela, basta que tenha um ator famoso na festa que essa reciprocidade provavelmente não acontecerá. Então, o grafo a ser usado nesse caso deve ser o orientado.

Quando os vértices estão interconectados é possível chegar ao nó B, partindo do nó A, não apenas quando existe uma aresta ligando os dois diretamente. Também pode existir um caminho que passa por outros nós e que vai de A para B sem utilizar a aresta que vai de A para B de forma direta. Um grafo não-orientado é dito conexo se todo par de vértices está conectado por um caminho (CORMEN, LEISERSON, *et al.*, 2002). Para os grafos orientados tem-se um conceito similar: grafo fracamente conexo. Desconsiderar o sentido de uma aresta é fazer com que a aresta (u,v) seja igual a aresta (v,u), exatamente como em um grafo não-orientado. Assim, um grafo orientado é fracamente conexo quando, desconsiderando o sentido de suas arestas, cada um de seus vértices é acessível a partir de outro.

As arestas de um grafo podem guardar consigo uma propriedade. Ela tem como objetivo indicar quanto aquela aresta é favorável ou não. Essa

propriedade é chamada de peso ou custo e é simbolizada por um valor numérico, como pode ser visualizado através da Figura 2. Como foi dito, é possível percorrer um grafo através de suas arestas, mas ao sair de um vértice até outro pode-se ter vários caminhos. Considerando, no grafo representado na Figura 2, que se queira ir do vértice 8 para o vértice 4, por exemplo. Uma alternativa seria ir pelo vértice 1 (8 -> 1 -> 4) e outra seria ir pelo vértice 2 (8 -> 2 -> 4). A aresta (8,1), que é a que vai do vértice 8 para o vértice 1 (utilizando o conceito de par ordenado da definição formal), tem peso 5, mas a aresta (1,4) tem peso 20. Somando esses dois valores tem-se que esse caminho custa 25 no total. Pelo outro caminho percorre-se as arestas (8,2) com peso 20 e a aresta (2,4) com peso 10, ou seja, o custo total é de 30. Dessa forma o caminho com o menor custo, e provavelmente mais favorável já que geralmente busca-se a diminuição do custo total, é o primeiro.

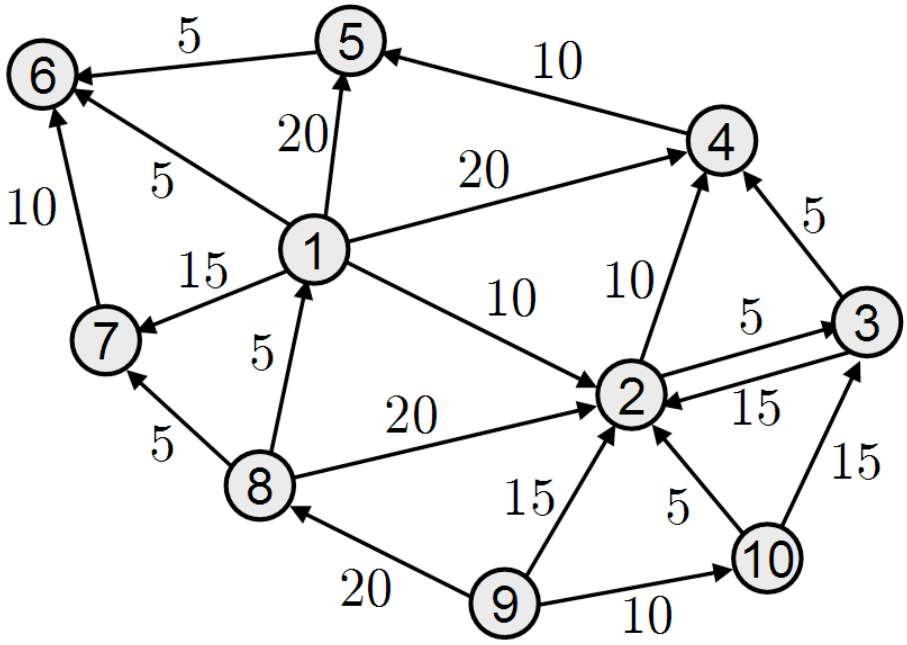



**Figura 2 - Grafo direcionado e com pesos (CARVALHO, 2009)**

Além dessa propriedade das arestas, os vértices dos grafos também apresentam propriedades ou classificações de acordo com suas características, como o grau. Um vértice pode ter várias arestas o relacionando com outros vértices. O grau de um vértice em um grafo não-orientado é o número de arestas incidentes nele. Em um grafo direcionado, o grau de saída é o número de arestas que saem dele, e grau de entrada é o número de arestas que entram nele. O grau de um vértice em um grafo direcionado é a soma do seu grau de entrada com seu grau de saída (CORMEN, LEISERSON, *et al.*, 2002).

Um exemplo muito comum que ilustra o uso de grafos é com relação a viagens de avião. Há vários aeroportos espalhados pelo mundo e vários aviões que viajam de um aeroporto para outro. Porém, nem sempre se consegue chegar diretamente onde se deseja, às vezes são necessárias paradas em outros aeroportos para fazer conexões. A analogia com grafos é simples de ser enxergada nesse caso, os aeroportos seriam os nós e os vôos as arestas. A analogia se estende ao ponto de ser possível de ir de um aeroporto (nó) a outro utilizando-se de mais de um vôo (aresta). Mas, os vôos não são iguais uns aos outros. Uma viagem pode ser mais longa ou mais curta e geralmente a distância influencia o preço. A partir de fatores como esses podem ser atribuídos pesos às arestas do grafo. É importante não congestionar muito um aeroporto. Um avião pode demorar mais a pousar por conta disso. O congestionamento de um aeroporto está relacionado com o grau do nó que o representa. Quanto mais arestas para um determinado nó maior a chance do aeroporto representado por ele se sobrecarregar.

## 2.2 Árvores Geradoras

Para entender bem o que é uma árvore geradora é necessário saber algumas coisas sobre grafos e árvores. Um caminho em um grafo é uma sequência de vértices, tal que, para todos os vértices, com exceção do último, existe uma aresta que sai dele e vai para o vértice seguinte da sequência. Quando um caminho tem o primeiro vértice igual ao último, pelo menos uma aresta e, com exceção do último, todos os vértices são distintos diz-se que esse caminho forma um ciclo simples. Uma árvore é um grafo acíclico, não-orientado e conexo (CORMEN, LEISERSON, *et al.*, 2002). Ao conjunto de árvores se dá o nome de floresta, termo geralmente usado quando o grafo não é conexo. Uma árvore enraizada é uma árvore em que um nó se distingue dos outros, esse nó é chamado de raiz. Se a última aresta no caminho da raiz até um nó x é (y,x) então y é pai de x e x é filho de y. Um nó sem filhos chama-se folha. A altura de uma árvore enraizada é a quantidade de arestas no caminho mais longo de sua raiz até uma folha.

Uma árvore geradora de um determinado grafo é uma árvore que conecta todos os vértices desse grafo através de um subconjunto de suas arestas. Como as arestas do grafo podem ter pesos (ou custos) tem-se o custo da árvore geradora como sendo a soma dos custos de todas as suas arestas. Quando o custo da árvore geradora é sempre menor ou igual ao de qualquer outra ela é chamada de árvore geradora de custo (ou peso) mínimo – AGPM. Em inglês é chamada de MCST (*Minimum Cost Spanning Tree*) ou simplesmente MST (*Minimum Spanning Tree*), que será o adotado neste documento pelo fato de o assunto abordado estar mais presente na língua inglesa. Caso o grafo não seja conexo, obviamente não existirá nenhuma árvore que conecte todos os seus vértices. Nesse caso, pode-se ter uma floresta com uma MST para cada componente conexo.

A importância das árvores geradoras se deve ao fato de que sempre é possível, a partir de um determinado nó, atingir qualquer outro. Além disso elas

tem menos arestas que o grafo original e não tem ciclos. Essas propriedades permitem a modelagem de problemas que envolvem diversos tipos de redes, como de computador ou de transmissão de energia. Ela também pode servir como abstração de tipos de comunicação de rodovias, de ferrovias, até de aviões.

Para uma idéia mais clara basta imaginar uma companhia de TV a cabo que esteja expandindo para um determinado local. Ela tem que distribuir o serviço em vários pontos dessa nova cidade. Essa distribuição pode ser feita de várias maneiras, inclusive de forma redundante, ou seja, com mais de um caminho para que o sinal da TV chegue em determinados pontos. Porém o ideal para a companhia é economizar nos custos. Quanto mais cabo for utilizado mais gasto terá a companhia. Com isso, percebe-se que uma árvore geradora pode resolver o problema, já que com ela todos os pontos estariam conectados e sem redundância (ciclos no grafo). Além disso, ao fazer a árvore geradora mínima, tem-se o menor custo para cabear a cidade. Sendo suficiente para que o sinal da TV chegue em todos os pontos de interesse.

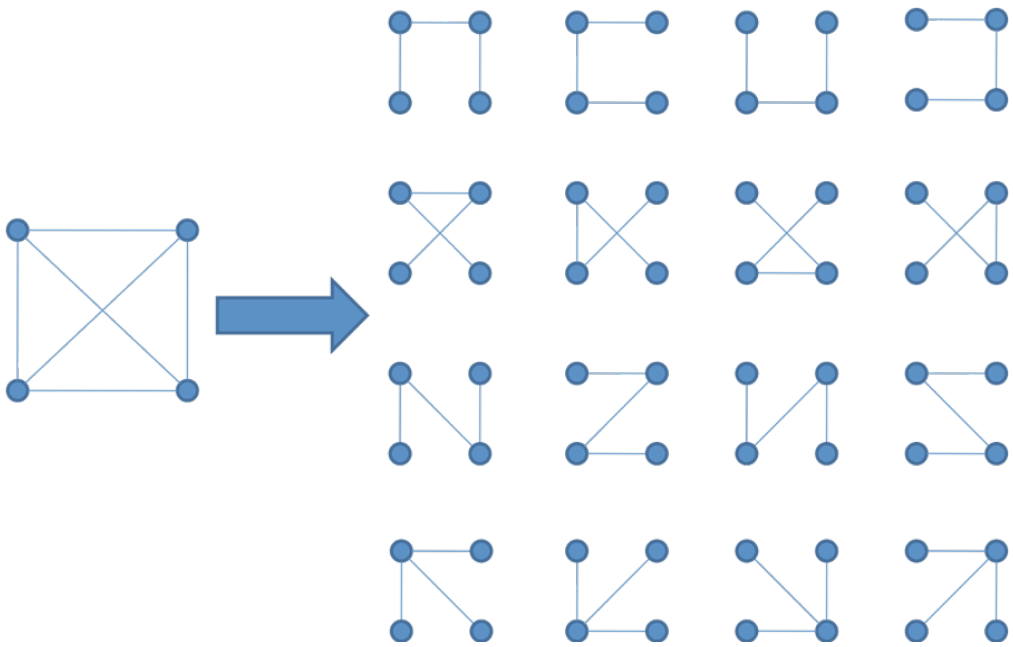


**Figura 3 - Conjunto de árvores geradoras de um grafo**

Como visto na figura acima um único grafo pode possuir muitas árvores geradoras, mesmo sendo um grafo de apenas 4 nós. Embora seja um caso tão simples, ainda é necessário elaborar uma estratégia bem definida para obter uma árvore geradora. A necessidade dessa estratégia pode ser percebida considerando a quantidade de grafos existentes, com o mesmo conjunto de vértices, e um subconjunto das arestas, do grafo original. No exemplo da Figura 3 o grafo tem 6 arestas, logo existem $2^6$ grafos que podem ser obtidos através da remoção de 0 ou mais arestas. Como são 16 árvores geradoras possíveis, só ¼ do total de grafos, que pode ser obtido pela remoção de arestas, corresponde à árvores geradoras. Além disso, como já foi explicado,

MSTs podem servir para representar e modelar problemas mais específicos. Portanto, dentre todas as árvores geradoras, geralmente deseja-se obter uma MST. No entanto, não é garantida a unicidade, na verdade é até bem comum que grafos apresentem mais de uma MST.

Felizmente o Problema da MST, que visa encontrar uma MST de um determinado grafo, já está satisfatoriamente resolvido por diversas técnicas. Mesmo ainda podendo existir mais de uma, o simples fato de ser uma MST geralmente já é suficiente. O próximo tópico discorre sobre as técnicas para a solução do Problema da MST por meio dos algoritmos de Boruvka, Kruskal e Prim.

## 2.3 Algoritmos para MSTs

Como já foi dito, existem soluções satisfatórias para o Problema da MST. Os 3 algoritmos mais comuns são o Boruvka, o Kruskal e o Prim. Todos os 3 são algoritmos gulosos que rodam em tempo polinomial.

O primeiro algoritmo para encontrar a MST foi desenvolvido pelo cientista Tcheco Otakar Boruvka em 1926, data de sua primeira publicação. Entretanto, o mesmo algoritmo foi também redescoberto várias vezes por pessoas diferentes entre os anos de 1938 e 1960. Algumas vezes, o algoritmo é dito de Sollin por ele ter sido o único cientista da computação do ocidente.

O algoritmo de Boruvka funciona através da adição de arestas a uma floresta de sub árvores geradoras do grafo. Esse procedimento é feito em estágios. Em cada estágio, a aresta de menor peso que conecta cada sub árvore geradora a outra é encontrada. Depois, essas arestas são todas adicionadas à MST final.

Uma explicação mais detalhada do algoritmo pode ser vista na Figura 4 e nos passos a seguir:

1. Criar uma floresta F com todos os vértices do grafo como sendo árvores separadas.
2. Adicione todas as arestas de menor custo que conectam duas árvores à MST.
3. Se F tiver mais de uma árvore volte para 2.

A figura abaixo ilustra o algoritmo descrito. Como pode ser observado, no início todos os vértices estão separados, há apenas uma indicação de quais são as arestas de menor custo. Essa indicação de menor custo é feita pelas setas, por exemplo, do vértice 0, a aresta de menor custo é a que vai para 2, e, no vértice 4, a que vai para o vértice 3 tem o menor custo. Estas são

selecionadas para fazerem parte da MST no passo seguinte. Essa etapa é repetida até que a floresta contenha apenas uma árvore. No caso da figura s;o foi necessária mais uma iteração.

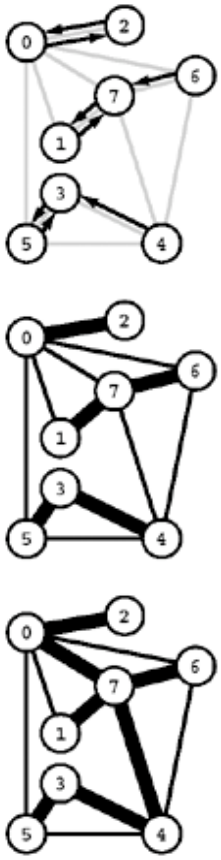



**Figura 4 - Passos do algoritmo de Boruvka (WESLEY)**

O algoritmo de Kruskal checa pontos de ótimo locais. A idéia dele é a menor aresta que não forma um ciclo. Nele a MST de um grafo conexo é obtida através dos passos:

1. Criar uma floresta F com todos os vértices do grafo como sendo árvores separadas.
2. Criar um conjunto S contendo todas as arestas do grafo.
3. Enquanto S não estiver vazio e F ainda tem elementos desconexos.
    a. Remover uma aresta com peso mínimo de S.
    b. Se ela conecta duas árvores de F, então adicione ela em F.

Em F encontra-se o MST do grafo. Sendo E o número de arestas o tempo do algoritmo de Kruskal é O(E log E). Sua execução em um grafo se dá como mostrado na figura abaixo.

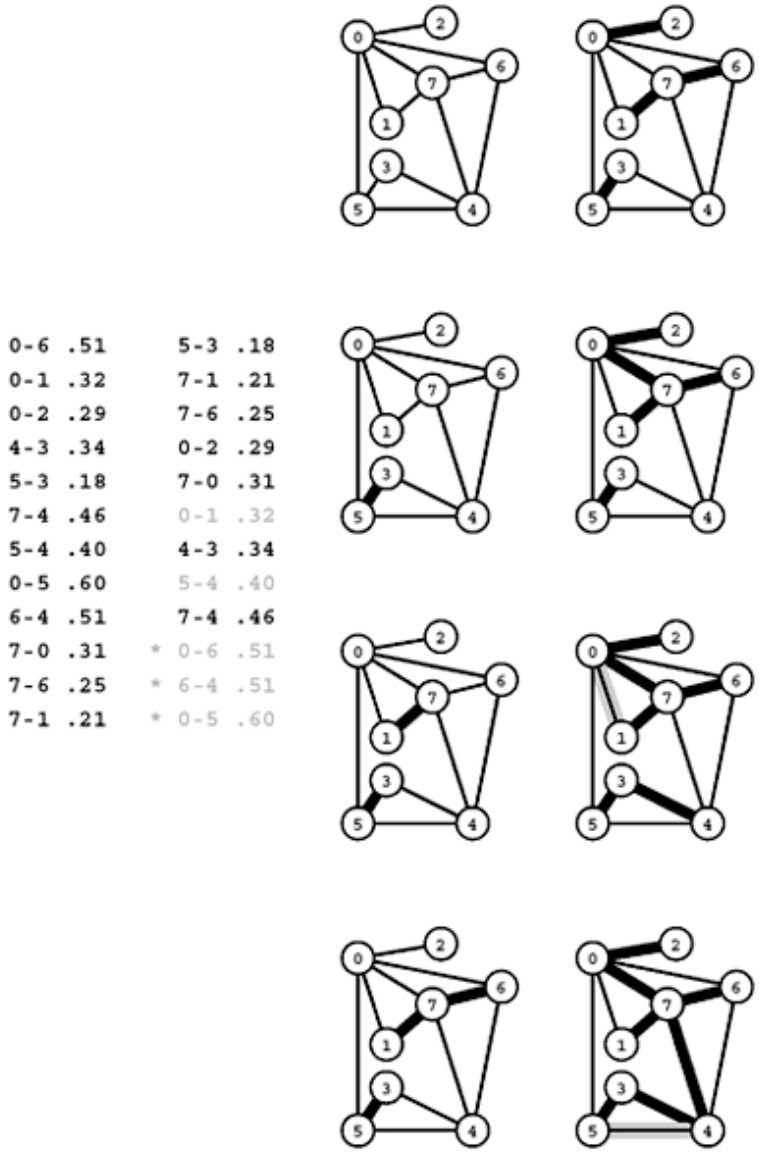



**Figura 5 - Passos do algoritmo de Kruskal (WESLEY)**

Na Figura 5 tem-se os passos do algoritmo de Kruskal. Na esquerda estão indicados os pesos das arestas e na direta cada aresta sendo adicionada em F.

O algoritmo de Prim roda em $O(V^2)$, quando usada uma matriz de adjacência para representar o grafo, sendo V o número de vértices do grafo. Com heaps e listas de adjacência consegue-se um tempo de O(E logV) ou O(V logV). Ele constrói a MST com um vértice por vez, funciona assim:

1. Escolha um vértice de partida e adicione a um conjunto Sv.
2. Crie um conjunto de arestas vazio Se.

3. Repita enquanto Sv ainda não for igual a V.
    a. Escolha uma aresta em que apenas um dos nós está em Sv e que tenha peso mínimo.
    b. Adicione o vértice que falta à Sv e a aresta à Se.

Todos eles têm complexidade de tempo e espaço polinomiais e, nos dois casos, as complexidades são próximas. Mas a maneira que constroem a MST é bem diferente, como pode ser visto na figura abaixo o Prim.

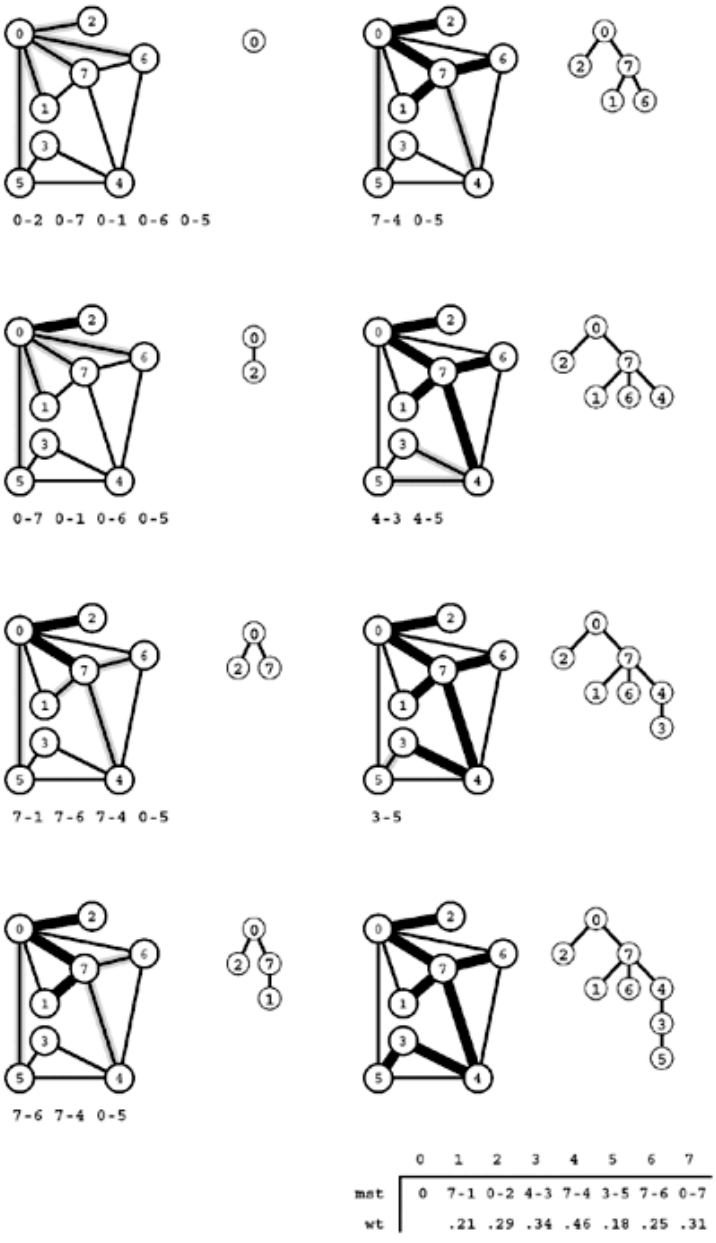



**Figura 6 - Passos do algoritmo de Prim (WESLEY)**

Na figura acima, o passo a passo do algoritmo de Prim é executado. Embaixo de cada grafo tem uma fila indicando a ordem em que as arestas serão escolhidas de acordo com os nós visitados e com os pesos.

## 2.4 Complexidade dos Problemas

A teoria de complexidade computacional é um segmento da teoria computacional que foca na classificação de problemas computacionais de acordo com sua dificuldade inerente. Para essa classificação foram criadas classes de complexidade para os problemas. As classes de complexidade são formadas por um conjunto de problemas com complexidade relacionadas. As principais são P, NP, NP-Completo e NP-Difícil. Os dois recursos mais comuns para avaliação de um problema são o tempo que é necessário para resolvê-lo e a memória, ou espaço, que ele necessita. Para a análise da complexidade basta levar em consideração o tempo.

Os problemas pertencentes à P são os problemas considerados fáceis. Esta classe consiste dos problemas que podem ser resolvidos por uma máquina seqüencial e determinística em tempo polinomial de acordo com o tamanho da entrada. Isso quer dizer que a solução dos problemas em P demora segundo uma função polinomial relativa ao tamanho da entrada rodando numa máquina como um computador. Já os problemas em NP são aqueles que podem ser verificados em um tempo polinomial com relação ao tamanho da entrada. Entretanto isso não quer dizer que eles tenham uma solução polinomial também. Informalmente, NP-Completos são os problemas mais difíceis dentre os NP. Mas, além de estarem em NP, eles são redutíveis em tempo polinomial a todos os outros problemas NP-Completos. Ser redutível a outro problema é poder ser transformado neste outro problema. Por fim, tem-se os NP-Difíceis que são pelo menos tão difíceis quanto os NP-Completos. Esses problemas podem ser tão difíceis que nem sequer seja possível a verificação em tempo polinomial.

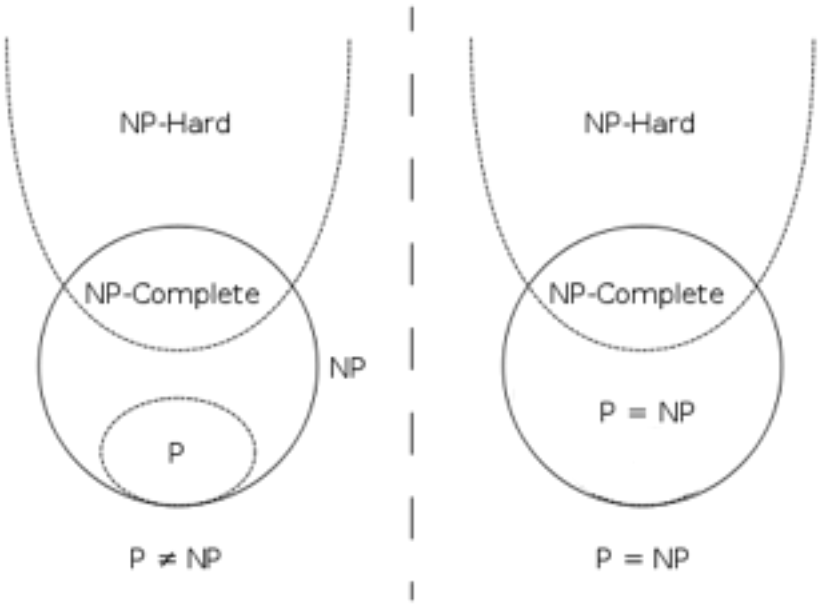



**Figura 7 - Diagrama de Euler para complexidades dos problemas**

A figura acima representa o diagrama de Euler, utilizado para visualização das classes de complexidades citadas. Como pode-se perceber há uma separação de dois diagramas na figura. Isso ocorre porque não se provou se P e NP são o mesmo conjunto ou não, se for, o diagrama da direita está correto, caso contrário deve-se usar o da esquerda. Como para todos os efeitos práticos P é diferente de NP, pode-se considerar o da esquerda. O diagrama está organizado de modo que os problemas mais difíceis estão mais em cima do outros. Temos basicamente o conjunto NP com P e NP-Completo como subconjuntos e o NP-Difícil que os problemas mais difíceis representados no diagrama com NP-Completo também como seu subconjunto.

# 3. MSTs com Restrições

Os algoritmos vistos aqui para a solução da MST dão o resultado exato e consomem poucos recursos. Infelizmente, também há muitos outros problemas que não admitem soluções tão eficientes assim. Algumas vezes, o que separa os problemas “fáceis” dos “difíceis” é uma linha aparentemente bem tênue. O que parece um detalhe à primeira vista pode tornar um problema bem mais complicado. Encontrar o caminho com o menor custo total (menor caminho) de um vértice para outro em um grafo com pesos é um problema com uma solução eficiente. Entretanto, ao pedir pelo maior caminho (sem ciclos) entre dois vértices tem-se um problema em que não se sabe uma solução substancialmente melhor do que checar todas os caminhos possíveis. A variação do problema passa a ter uma solução com tempo exponencial.

Muitas vezes não se precisa exatamente da MST, mas de uma variação da mesma. Para diversas aplicações do mundo real, considerar simplesmente a soma dos pesos como mínima não é suficiente. Esse tipos de aplicações necessitam de modificações nas MSTs para se aproximarem mais dos problemas encontrados. É através dessa necessidade de uma nova e mais específica modelagem que restrições são adicionadas.

## 3.1 Adição de Restrições nas MSTs

Existem vários tipos de restrições que podem ser aplicadas em MSTs com o objetivo de uma melhor aproximação à problemas reais. Pode-se restringir o grau, ou seja, a MST deve ter todos seus vértices com no máximo um determinado grau. Limitar a capacidade de uma MST significa dizer que nenhuma sub-árvore diretamente ligadas à raiz têm a soma do peso de suas arestas maior do que um valor fixo. A MST com restrição de capacidade é muito útil no projeto de redes. Mas, para um melhor entendimento da necessidade dessa adição de restrições às MSTs, alguns casos em que essas restrições ajudam na modelagem dos problemas são explicados.

*Switches* em uma rede de comunicação atual têm um número limitado de conexões disponíveis. Desse modo, os sistemas de transporte de dados devem estabelecer um limite ao número de caminhos que se encontram de um determinado lugar (que seria um nó da rede). Assim, para os *switches*, o grau dos nós da rede deve ser limitado. Além disso, esse limite do grau de cada nó também diminui o impacto potencial caso um nó falhe.

Um outro exemplo, relacionado à administração de redes de computadores, pode ser visualizado inicialmente tomando a raiz como sendo um conjunto de roteadores. Os *links* geralmente são limitados, basicamente

para prevenir que o uso excessivo de uns prejudique outros. Admitindo que cada *slot* tem um limite de capacidade de Q MB/s temos que a soma da demanda de todos os clientes da árvore conectada a cada *slot* (sub-árvores enraizadas nesse conjunto de roteadores) também será limitada, da mesma forma e pela mesma velocidade. Assim sendo, a capacidade de tráfego máxima do *slot* será de Q MB/s. Esse tipo de implementação também permite baixar o custo da implementação da rede.

Há vários outros tipos conhecidos de restrições que podem ser aplicadas nas MSTs. Um deles, bastante encontrado na literatura é a limitação do diâmetro. O diâmetro de uma árvore é a maior distância entre qualquer par de seus vértices. Ele encontra importantes aplicações na indústria. Porém, ao adicionar restrições a essas árvores, o problema pode se tornar bem mais complicado. Sabe-se que essa restrição pode tornar o problema NP-difícil. Inclusive, se limitarmos o diâmetro a 4 ou mais, o problema fica tão difícil de aproximar quanto o Problema da Cobertura de Conjuntos (para o qual sabemos que não pode existir uma aproximação de razão log n, onde n é o número de elementos no conjunto, a menos que P = NP). Bar-Ilan, Korsarz e Peleg mostram em (ILAN, KORTSARZ e PELEG, 2001) uma aproximação de razão logarítmica para os casos em que o diâmetro é limitado a 4 e a 5. Existe uma estreita relação entre este problema e o Problema da Localização de Facilidades, para o qual são conhecidas aproximações O(log n) (HOCHBAUM, 1982) e (JAIN, MAHDIAN, *et al.*, 2003).

Uma variante deste, consiste em, dado um grafo conexo, encontrar uma árvore geradora de diâmetro mínimo. Hassin e Tamir mostraram em (HASSIN e TAMIR, 1995) que esse problema pode ser resolvido em tempo polinomial mesmo quando cada aresta tem um comprimento dado e o diâmetro leva em conta esses comprimentos. A variante em que se busca uma árvore geradora de diâmetro mínimo (com comprimento nas arestas) e grau máximo limitado também é de interesse e se mostra bem mais difícil. Könemann, Levin e Sinha apresentaram uma aproximação $O(\log_B n)$ em (KÖNEMANN, SINHA e LEVIN, 2004) – para essa variante, onde B denota a limitação no grau máximo da árvore.

O problema da árvore geradora com graus limitados também não possui uma solução com tempo polinomial, pertencendo portanto, a classe NP-Difícil. Ele foi estudado por Fürer e Raghavachari em (FÜRER e RAGHAVACHARI, 1990) e (FÜRER e RAGHAVACHARI, 1992) e por Könemann e Ravi em (KÖNEMANN e RAVI, 2002) e (KÖNEMANN e RAVI, 2005) e são conhecidas boas aproximações para diferentes variantes dele. No entanto, nem sempre os próprios autores estão certos sobre a complexidade das aproximações que eles propõem.

Além da variante citada no parágrafo anterior, árvore geradora de diâmetro mínimo e grau máximo, há uma outra variante importante. O objetivo dessa outra variante é encontrar a MST tal que o grau máximo seja o mínimo possível. Esse problema é NP-Difícil e generaliza o problema dos caminhos hamiltonianos. Em (FÜRER e RAGHAVACHARI, 1992) são mostrados algoritmos iterativos, baseados em técnicas combinatoriais, para aproximar o problema da árvore geradora com graus limitados e problemas relacionados,

incluindo o problema da MST com o grau máximo sendo o mínimo possível. Esse processo se dá na tentativa de diminuir o grau de um nó da MST obtida na iteração anterior.

Também existem resultados de inaproximabilidade, ou seja, de impossibilidade da aproximação em alguns casos, geralmente casos com complexidade de tempo curto. Um estudo feito em (LUCA, 2001) prova alguns resultados de não-aproximabilidades para restrições em problemas de otimização combinatória para instâncias de restrição de grau. Nesse trabalho, foi observado que o problema do conjunto independente em grafos, com o grau no máximo de B, era difícil a aproximação por um fator $B/2^{O(\mathrm{sqrt}(\log B))}$, a menos que P seja igual a NP.

# 4. Abordagens Conhecidas

No momento, sabe-se que os problemas NP-Completos requerem um tempo muito grande e não se sabe se existem algoritmos mais rápidos. No entanto, as soluções para esses problemas são muito importantes. Então, foram desenvolvidas e são aplicadas diversas técnicas para a obtenção dessas soluções ou do mais próximo possível.

As técnicas a seguir podem ser aplicadas para resolver problemas computacionais em geral e frequentemente dão origem a novos e mais rápidos algoritmos:

- **Aproximação**: Busca, ao invés de uma solução ótima, uma solução “quase” ótima.
- **Randomização**: Utiliza a aleatoriedade para conseguir um tempo médio mais rápido e permite que o algoritmo falhe, mas com uma probabilidade pequena.
- **Restrição**: Restringe a entrada para um subconjunto da mesma, geralmente de acordo com a estrutura.
- **Parametrização**: Limita a entrada de acordo com determinados parâmetros, fixando-os, para que algoritmos mais rápidos possam ser aplicados.
- **Heurística**: Um algoritmo que funciona razoavelmente bem, mas que não há provas que sempre tem um bom resultado ou que é sempre rápido.

Como já mencionando o foco deste trabalho está nos algoritmos de aproximação. Nesta seção, serão descritos algumas abordagens já existentes para a solução dos principais problemas de árvores geradoras com restrições. Existem muitas abordagens possíveis e muitas vezes elas aparecem combinadas em um algoritmo para uma melhor obtenção do resultado. Como exemplo de abordagens mais vistas na literatura temos:

i. Branch and Bound
ii. Greedy Randomized Adaptative Search Procedure (GRASP)
iii. Evolutionary / Genetic Algorithms (EA)
iv. Variable Neighbourhood Search (VNS)
v. Ant-based algorithms / Ant Colony Optimization (ACO)

O Branch and Bound ou Branch and Cut é uma técnica que consiste em uma enumeração sistemática de todas as soluções candidatas. Nessa enumeração, grandes quantidades de candidatos malsucedidos à solução são

descartados em massa pelo uso de limites superior e inferior do que está sendo otimizado. Para algumas etapas dele, como o particionamento do espaço de soluções, podem ser usados os algoritmos para MSTs, como o Kruskal.

GRASP é uma metaheurística iterativa. Assim como diversos métodos construtivos consiste primeiramente em criar uma solução inicial, no caso, como o próprio nome diz, através de algoritmos gulosos e aleatórios. Depois efetuar uma busca local para melhorar a solução já que geralmente não é obtido um ótimo local de primeira. Após o número desejado de iterações, a melhor solução de todas é adotada como o resultado final.

As três últimas abordagens citadas serão melhor explicadas nas seções que seguem devido a sua importância e maior uso nos problemas de MST com restrição, principalmente quando a restrição é no grau ou no diâmetro.

## 4.1 Evolutionary / Genetic Algorithm

Algoritmos evolucionários são baseados na real evolução biológica. E com tal se utiliza de certos mecanismos inspiradas pela evolução biologia: reprodução, mutação, recombinação e seleção. Soluções candidatas são tidas, para o algoritmo, como indivíduos em uma população. A *fitness function* é uma função que decide quais desses indivíduos vão “viver”. O algoritmo genético é o tipo mais popular de EA em que a solução é buscada na forma de string de números e são utilizados operadores genéticos melhor descritos em breve.

Esse algoritmo funciona da seguinte forma:

1. Inicialização
2. Seleção
3. Reprodução
4. Término

Na inicialização muitas soluções individuais são geradas aleatoriamente para formar a população inicial. Procura-se fazer com que esse população inicial cubra bem todo o intervalo de possíveis soluções, ou seja, que os indivíduos dessa população estejam bem distribuídos.

A seleção é um processo que busca a cada nova geração criada indivíduos para gerar uma nova geração. Para isso são utilizadas funções que buscam avaliar quais indivíduos estão mais próximos da solução, sendo eles, mais provavelmente selecionados.

No próximo passo se dá a reprodução dos indivíduos. Aqui é onde os indivíduos selecionados vão originar novos indivíduos a partir de “operações genéticas”. Cada indivíduo é codificado para ser utilizado pelo algoritmo

genético, ao resultado dessa codificação dá-se o nome de genótipo, que geralmente é um array de bits. Antes de codificar, ou depois de decodificar o genótipo temos o fenótipo (nomes dados também de acordo com os nomes reais para as situações análogas da biologia) que é a expressão do genótipo, ou seja, um possível resposta ao problema original. Essas operações genéticas acontecem no nível dos genótipos (analogia com cromossomos) e são basicamente uma recombinação ou uma mutação. Na recombinação dois cromossomos trocam informações para originar dois outros cromossomos. Ele é realizado pela escolha de um gene aleatório ao longo dos cromossomos e a troca de todos os genes (bits) a partir desse ponto. Desse modo o início de cada um dos cromossomos é o mesmo, porém o final deles fica trocado. A mutação dá a capacidade de mudança de um cromossomo sem interferência externa. Nela estabelece-se uma chance, muito pequena, para a inversão do valor de um bit.

Como o processo é cíclico deve se estabelecer um critério de parada. Geralmente se escolhe um número máximo de gerações a serem geradas ou então um patamar mínimo para a solução, dentre outros. Não é necessário que se escolha só um critério, pode-se ter os dois critérios citados como de parada, até porque dependendo do patamar escolhido pode ser que o algoritmo demore um tempo “infinito” para parar.

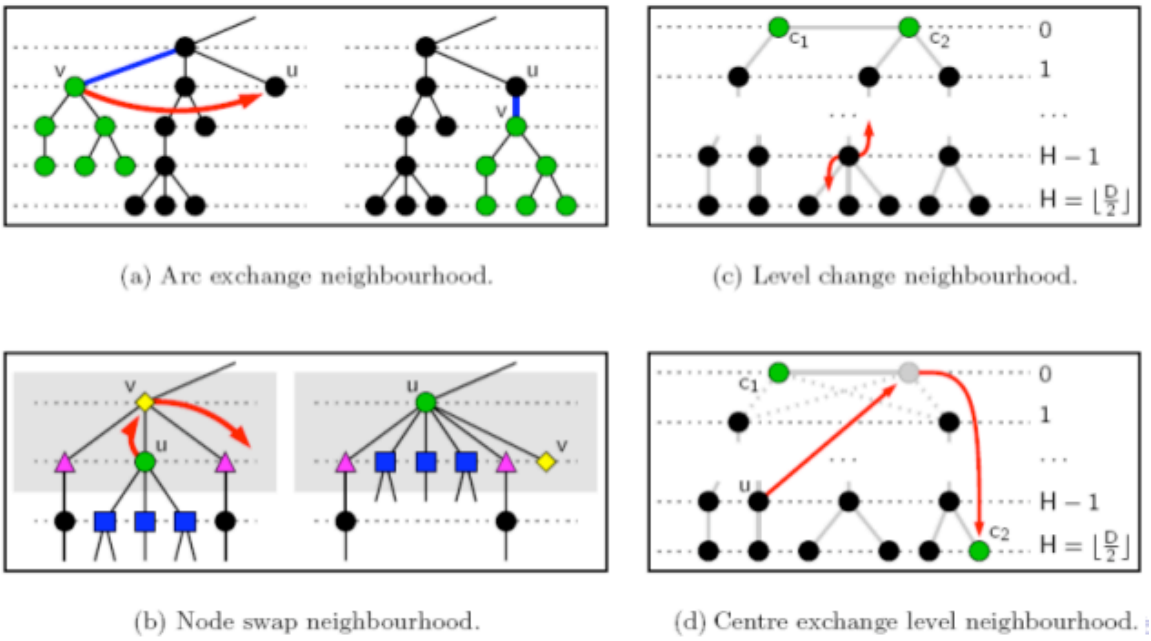
## 4.2 Variable Neighbourhood Search

Essa abordagem é uma abstração espacial do problema. Como é comum nas buscas ou algoritmos que têm a vizinhança como foco, o VNS tem como premissas:

- Mínimos locais não dependem da estrutura da vizinhança.
- O mínimo global é o mínimo local com respeitos a todas as vizinhanças.
- Os mínimos locais geralmente estão próximos uns dos outros.

A mudança sistemática da vizinhança com uma possível busca local aleatória leva a uma simples e efetiva metaheurística para otimização combinatória.

Dito isso, o VNS busca atingir o mínimo global através de mudanças na estrutura, desse modo consegue sair de um provável mínimo local para outro. No caso de MSTs a figura abaixo mostra operações que podem ser feitas na árvores para se ter essas mudanças na estrutura. Dentre essas operações estão mudança de nível de nós na árvore e mudança de arco.




(a) Arc exchange neighbourhood.

(c) Level change neighbourhood.

(b) Node swap neighbourhood.

(d) Centre exchange level neighbourhood.

**Figura 8 - Operações para VNS (GRUBER, HEMERT e RAIDL, 2006)**

Como pôde ser observado na figura há operações bem simples a serem feitas em árvores para uma mudança em sua estrutura (GRUBER, HEMERT e RAIDL, 2006). Na primeira tem-se uma troca de arco. Pode ser interpretada como desconectar uma sub árvore e conectá-la em um outro lugar conveniente. Logo abaixo é representada a troca de nó que foca na relação entre os nós e seu conjunto de sucessores diretos. Eles trocam de lugar e o sucessor mantém os filhos. Na figura 8-c uma solução adjacente é encontrada pelo incremento ou decremento do nível de exatamente um nó. Por último tem-se a troca do nó do centro por outro nó que não seja central (nós centrais são os que estão no nível 0).

## 4.3 Ant Colony Optimization

Ant Colony Optimization é uma técnica probabilística para resolver problemas computacionais que podem ser reduzidos a encontrar bons caminhos em grafos. O algoritmo é baseado nas formigas e em sua movimentação fora do formigueiro.

As formigas liberam uma substância, chamada feromônio, à medida que caminham, tornando esse caminho por onde passam mais atrativo para todas as formigas. Assim, quando estão levando comida para o formigueiro elas não começam por um bom caminho necessariamente. As concentrações de feromônio vão sendo alteradas ao longo do caminho com o passar do tempo. Logo, o caminho vai se modificando a medida que elas transitam entre sua fonte de alimento e o formigueiro. A figura 9 representa justamente esse

trânsito das formigas e o ajuste do caminho de acordo com a concentração de feromônio.

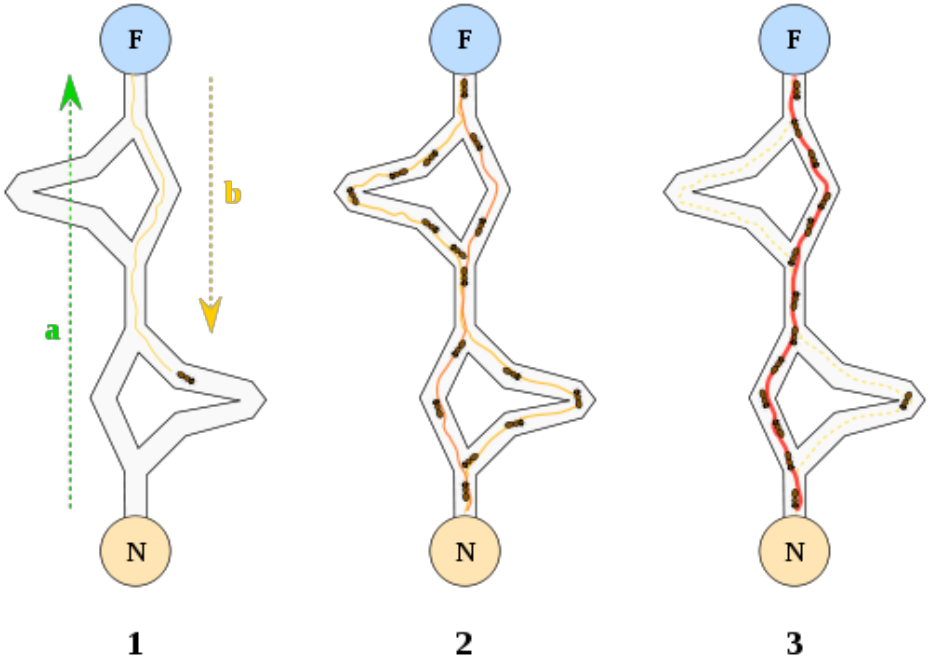



**Figura 9 - Escolha do caminho por formigas**

Pela figura se percebe que o caminho mais curto foi favorecido. Isso acontece porque, com o tempo, o feromônio vai evaporando, reduzindo a sua força atrativa. Quanto maior o tempo que um formiga leva para ir e voltar por um determinado caminho, maior o tempo que o feromônio tem para evaporar. Portanto, como os caminhos menores são percorridos mais rapidamente, fazendo com que a densidade de feromônio ao longo deles fique mais elevada do que em caminhos maiores.

O algoritmo, para o caso das MSTs, se dá de forma análoga. Primeiro as formigas passeiam no grafo sem muitas restrições e isso dá origem à uma árvore temporária (lembrando de respeitar a restrição em questão). Depois se atualiza a melhor MST obtida caso essa nova tenha um custo menor, como na primeira vez essa é a melhor árvore que se tem ela é atualizada da mesma forma. Depois se atualiza o nível de feromônio dos caminhos aumentando por onde passou mais formigas e evaporando um pouco em todos os lugares. Caso a melhor MST obtida tenha sido atualizada recentemente  repita o processo.

# 5. Comparação

De acordo com o exposto pode-se perceber que existem várias técnicas com diversas abordagens para a construção de um algoritmo capaz de dar um resultado aproximado para MSTs com restrição. Embora existam tantas possibilidades muitas vezes é necessária apenas uma implementação. Porém, dentre todas essas possibilidades qual deve ser a escolhida?

Para ajudar a responder essa pergunta geralmente quando os algoritmos são desenvolvidos eles são também avaliados teoricamente. Nessa avaliação teórica projeta-se, principalmente, o quanto de espaço e tempo é necessário para o algoritmo. Entretanto, essa análise prévia muitas vezes não dá uma noção muito boa da alocação de recursos real. Esta distância entre a análise teórica e a prática foi estudada em (GOMES, MENESES, *et al.*, 2006) para os problemas da cobertura de vértices e cobertura de conjuntos.

Não sendo possível escolher qual das possibilidades é melhor usar com certeza através de uma análise teórica resta escolher de acordo com experimentos práticos. Embora a escolha esteja intimamente ligada ao tipo de problema, estudos que avaliam na prática as abordagens a serem utilizadas são bastante úteis. Um estudo comparativo entre as abordagens descritas aqui: EA, ACO e VNS foi feita em (GRUBER, HEMERT e RAIDL, 2006) e será apresentada a seguir.

## 5.1 Metodologia

A comparação entre as três abordagens EA, VNS e ACO será feita através do BDMST (*Bounded Diameter Minimum Spanning Tree* – MST com restrição no diâmetro). Foram implementados três algoritmos para o BDMST, cada um em uma abordagem, para posteriormente serem comparados os resultados obtidos.

Inicialmente tem-se uma árvore gerada aleatoriamente (RTC – *Randomized Tree Construction*). Para que depois os resultados sejam melhorados pelas abordagens utilizadas. Foram utilizados grafos completos (todo nó ligado diretamente a todos os outros) de tamanhos de 100, 250, 500 e 1000 vértices. Essas instâncias foram retiradas da *Beasley's OR-Library* (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/esteininfo.html), elas são representadas por pontos e a distância euclidiana entre qualquer par é o peso da aresta. Os diâmetros foram limitados a 10, 15, 20 e 25, para as instâncias

de 100, 250, 500 e 1000 respectivamente. A altura da árvore é mantida menor do que a metade da restrição do diâmetro, porque, desse modo, o diâmetro é mantido dentro do seu limite.

Todos os teste foram feitos em um PC, com o mesmo processador (Pentium 4 de 2,8GHz) e mesmo sistema operacional (Linux). Para o EA uma população de 100 indivíduos foi escolhida. O número de formigas artificiais no ACO foi 25 e o valor de decaimento do feromônio foi dependente da instância: 0,003, 0,005, 0,006 e 0,008.

Com respeito a condição de término foram feitas duas séries de experimentos: uma longa e uma curta. Os experimentos com tempo maior foram dados tempos 2000, 3000 e 4000 segundos para os grafos com 100, 250 e 500 vértices respectivamente. Este era relativo à entrada de menor tamanho, quanto maior a entrada maior era o tempo dado. Entretanto nem sempre era necessário esperar tanto, podendo a solução convergir antes tornando melhoras bastante improváveis.

## 5.2 Resultados

Para a comparação dos resultados foi dado o foco principal na qualidade da MST gerada. Além da qualidade da árvore também foi medido o tempo de cada abordagem. A qualidade da MST foi medida através do *objective value* dela. O *objective value* é um número obtido através de uma função chamada de *objective function* que mede o quão boa é uma aproximação. Neste trabalho quanto menor o *objective value* tem-se uma menor distância entre a aproximação e a solução exata.

Para mostrar os dados dos experimentos realizados foram feitas duas tabelas (abaixo). Estão listados os *object values* melhor (best) e médio (mean), a correspondente variação padrão (stdev) e os tempos médios em segundos (sec) de identificação das melhores soluções para cada instância. Os melhores resultados das três abordagens utilizadas estão escrito em negrito nas duas tabelas.

| Instance | | | VNS | | | | level-encoded EA | | | | ACO | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $D$ | nr | best | mean | stdev | sec. | best | mean | stdev | sec. | best | mean | stdev | sec. |
| 100 | 10 | 1 | **7.759** | 7.819 | 0.03 | 37.35 | 7.760 | 7.785 | 0.03 | 678.70 | **7.759** | **7.768** | 0.02 | 27.78 |
| | | 2 | 7.852 | 7.891 | 0.03 | 41.52 | **7.849** | **7.860** | 0.02 | 734.65 | 7.850 | 7.864 | 0.01 | 25.10 |
| | | 3 | **7.904** | 7.962 | 0.04 | 38.66 | **7.904** | 7.964 | 0.04 | 897.58 | 7.907 | **7.943** | 0.04 | 28.48 |
| | | 4 | 7.979 | 8.046 | 0.03 | 34.27 | **7.977** | 8.008 | 0.03 | 732.83 | 7.979 | **8.000** | 0.01 | 38.24 |
| | | 5 | 8.165 | 8.203 | 0.03 | 39.31 | **8.164** | 8.176 | 0.03 | 410.17 | **8.164** | **8.170** | 0.00 | 25.45 |
| 250 | 15 | 1 | 12.301 | 12.430 | 0.05 | 1584.31 | 12.280 | 12.377 | 0.05 | 1992.70 | **12.231** | **12.280** | 0.02 | 174.17 |
| | | 2 | 12.024 | 12.171 | 0.06 | 1678.90 | 12.054 | 12.156 | 0.06 | 1969.42 | **12.016** | **12.038** | 0.01 | 156.71 |
| | | 3 | 12.041 | 12.112 | 0.04 | 1309.21 | 12.026 | 12.095 | 0.04 | 1897.87 | **12.004** | **12.021** | 0.01 | 145.29 |
| | | 4 | 12.507 | 12.615 | 0.06 | 1572.39 | 12.487 | 12.594 | 0.05 | 1742.48 | **12.462** | **12.486** | 0.01 | 159.41 |
| | | 5 | 12.281 | 12.423 | 0.07 | 1525.39 | 12.319 | 12.423 | 0.06 | 1712.16 | **12.233** | **12.288** | 0.04 | 211.11 |
| 500 | 20 | 1 | 16.974 | 17.129 | 0.07 | 3718.54 | 16.866 | 16.967 | 0.06 | 2609.28 | **16.778** | **16.850** | 0.03 | 906.17 |
| | | 2 | 16.879 | 17.052 | 0.07 | 3762.02 | 16.764 | 16.858 | 0.05 | 2472.59 | **16.626** | **16.699** | 0.03 | 1012.91 |
| | | 3 | 16.975 | 17.148 | 0.07 | 3849.42 | 16.856 | 16.977 | 0.05 | 2808.15 | **16.792** | **16.844** | 0.03 | 1069.84 |
| | | 4 | 16.992 | 17.166 | 0.06 | 3687.97 | 16.943 | 17.040 | 0.06 | 2837.81 | **16.796** | **16.923** | 0.04 | 1010.91 |
| | | 5 | 16.572 | 16.786 | 0.07 | 3693.13 | 16.501 | 16.590 | 0.05 | 2294.43 | **16.421** | **16.456** | 0.02 | 947.26 |

**Tabela 1 - *Objective values* para o experimento longo (GRUBER, HEMERT e RAIDL, 2006)**

| Instance | | | limit | VNS | | | | level-encoded EA | | | | ACO | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $D$ | nr | sec. | best | mean | stdev | sec. | best | mean | stdev | sec. | best | mean | stdev | sec. |
| 500 | 20 | 1 | 50 | 17.753 | 18.108 | 0.12 | 46.41 | **16.573** | **16.760** | 0.16 | 37.94 | 17.594 | 17.751 | 0.06 | 41.29 |
| | | 2 | | 17.688 | 17.966 | 0.10 | 44.70 | **16.826** | **17.014** | 0.11 | 41.06 | 17.403 | 17.583 | 0.05 | 40.33 |
| | | 3 | | 17.799 | 18.114 | 0.10 | 46.23 | **16.947** | **17.192** | 0.13 | 43.15 | 17.653 | 17.756 | 0.05 | 39.66 |
| | | 4 | | 17.930 | 18.161 | 0.11 | 45.38 | **16.957** | **17.085** | 0.08 | 39.18 | 17.647 | 17.793 | 0.05 | 41.41 |
| | | 5 | | 17.464 | 17.863 | 0.12 | 45.94 | **17.055** | **17.245** | 0.13 | 39.54 | 17.331 | 17.438 | 0.05 | 40.95 |
| 500 | 20 | 1 | 500 | 17.290 | 17.460 | 0.08 | 476.22 | **16.534** | **16.641** | 0.07 | 340.34 | 17.017 | 17.150 | 0.07 | 485.57 |
| | | 2 | | 17.215 | 17.373 | 0.08 | 480.87 | **16.808** | **16.902** | 0.05 | 320.84 | 16.864 | 17.072 | 0.08 | 478.47 |
| | | 3 | | 17.252 | 17.464 | 0.05 | 476.33 | **16.886** | **17.017** | 0.06 | 319.09 | 17.094 | 17.259 | 0.07 | 479.17 |
| | | 4 | | 17.318 | 17.514 | 0.07 | 476.80 | **16.923** | **17.036** | 0.06 | 316.33 | 17.070 | 17.277 | 0.08 | 472.57 |
| | | 5 | | 16.932 | 17.139 | 0.09 | 473.82 | 17.007 | 17.105 | 0.06 | 288.66 | **16.613** | **16.791** | 0.08 | 479.93 |
| 1000 | 25 | 1 | 100 | 25.850 | 26.188 | 0.13 | 75.40 | **24.831** | **25.019** | 0.10 | 92.06 | 25.246 | 25.437 | 0.07 | 81.42 |
| | | 2 | | 25.501 | 25.981 | 0.17 | 68.30 | **24.890** | **25.159** | 0.10 | 89.29 | 25.092 | 25.239 | 0.07 | 80.17 |
| | | 3 | | 25.340 | 25.705 | 0.09 | 62.33 | 25.021 | 25.338 | 0.14 | 92.27 | **24.870** | **25.007** | 0.06 | 73.96 |
| | | 4 | | 25.562 | 26.128 | 0.17 | 73.89 | **25.133** | 25.524 | 0.12 | 92.17 | 25.329 | **25.450** | 0.06 | 76.56 |
| | | 5 | | 25.504 | 25.826 | 0.15 | 74.75 | 25.493 | 25.675 | 0.08 | 89.18 | **24.884** | **25.153** | 0.07 | 79.90 |
| 1000 | 25 | 1 | 1000 | 25.177 | 25.572 | 0.14 | 905.50 | **23.434** | **23.573** | 0.08 | 565.38 | 24.842 | 25.033 | 0.07 | 812.78 |
| | | 2 | | 25.015 | 25.342 | 0.14 | 930.04 | **23.464** | **23.668** | 0.08 | 561.49 | 24.634 | 24.834 | 0.06 | 847.79 |
| | | 3 | | 24.816 | 25.086 | 0.11 | 956.06 | **23.635** | **23.793** | 0.08 | 524.21 | 24.498 | 24.619 | 0.06 | 838.68 |
| | | 4 | | 25.289 | 25.572 | 0.11 | 928.97 | **23.787** | **23.962** | 0.09 | 602.30 | 24.993 | 25.091 | 0.06 | 793.41 |
| | | 5 | | 25.026 | 25.254 | 0.12 | 935.85 | **23.837** | **23.982** | 0.10 | 516.74 | 24.571 | 24.732 | 0.06 | 844.67 |

**Tabela 2 - *Objective values* para o experimento curto (GRUBER, HEMERT e RAIDL, 2006)**

Como pode ser percebido pela primeira tabela acima, as instâncias com a quantidade de vértices (n) igual a 100 parecem muito pequenas para prover uma boa comparação, já que os valores ficaram muito próximos. Portanto a avaliação dos resultados serão feitas sem levar esse caso em consideração.

Para o experimento com tempo maior (Tabela 1) o ACO se mostrou muito melhor que os outros. Ele tanto apresenta *objective values* menores que os outros dois sempre quanto tem um desvio padrão menor também. O EA também foi melhor do que o VNS, apresentando os valores médios menores ou iguais sempre. Além disso tudo ao olhar para os tempos médios para encontrar as melhores soluções o ACO também foi, em praticamente todos os casos melhor do que os outros.

O experimento com tempo menor (Tabela 2) torna os resultados um pouco diferentes dos anteriores. O EA e o ACO continuam melhores do que o VNS em quase todos os casos, porém a média do EA está melhor do que a do ACO.

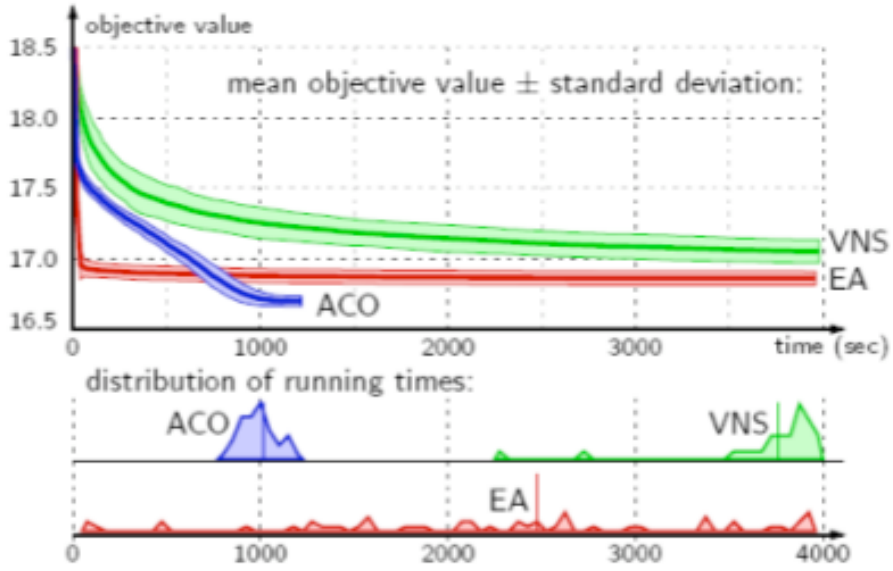



**Figura 10 - *Objective value* pelo tempo e distribuição do tempo de execução (GRUBER, HEMERT e RAIDL, 2006)**

A Figura 10 mostra o *objective value* médio pelo tempo para múltiplas execuções do VNS, EA e ACO com 500 nós e 20 de diâmetro máximo. A parte de baixo exibe a distribuição dos tempo de execução requeridos para identificar a melhor solução de uma execução. Os tempos de execução médios são indicados por uma linha vertical.

Dessa forma fica mais fácil a visualização dos dados da tabela, conseguindo extrair a essência da comparação: o EA se estabiliza mais rápido e chega a um bom resultado, mas se tiver tempo disponível suficiente o ACO atinge melhores resultados. O VNS demora mais e não chega a uma solução tão boa.

# Conclusões e Trabalhos Futuros

MSTs com restrição são problemas difíceis de serem resolvidos, mas, devido a sua importância, muitas vezes se busca sua solução por uma aproximação satisfatória. Há diversas técnicas diferentes para que esse objetivo possa ser alcançado.

Devido à grande quantidade de abordagens diferentes a serem escolhidas para resolver algum problema é necessário uma comparação entre elas. Tomando o problema BDMST como alvo tem-se que o EA apesar de ser um algoritmo bem geral ele consegue bons resultados e de maneira muito rápida, com uma boa distribuição de tempo. O VNS se mostrou o pior dentre os comparados demorando para convergir para soluções mais próximas da solução exata e não chegou a aproximações tão boas. O ACO obteve as melhores soluções. No entanto, ele necessita de mais tempo para obter aproximações realmente boas, apesar de não ser muito lento o processo até a estabilização das aproximações.

As comparações são muito específicas dependendo muito da implementação utilizada e como o problema foi modelado. Portanto não se pode afirmar que os resultados quando aplicados a um problema de uma MST com restrição de grau sejam os mesmos.

Houve uma grande dificuldade na definição clara do conteúdo deste documento. Propor uma solução nova seria um trabalho com uma dimensão muito superior à proposta e agregaria um risco muito alto. Com isso, foi estabelecido que o trabalho ficaria em torno de soluções já propostas. No entanto, muitos algoritmos propostos não eram bem explicados fazendo com que uma implementação deles pudesse não refletir exatamente a solução de seu autor. Como alternativa foi feito um estudo de caso com explicações sobre árvores geradoras, técnicas empregadas e comparações de algumas técnicas mais importantes.

Como trabalhos futuros, pode-se fazer uma avaliação de outros algoritmos ou abordagens. Também é possível que, ao estudar novos algoritmos, pontos de melhoramentos sejam encontradas e então sugeridos. Pode-se se criar uma nova frente para esse estudo avaliando as melhoras provocadas em algoritmos híbridos em relação aos puros. Uma outra possível evolução do trabalho é buscar o desenvolvimento, e posteriormente, até uma implementação de um novo algoritmo.

# Referências Bibliográficas


CARVALHO, R. Optimal Account Balancing II, 28 Julho 2009. Disponivel em: <http://stochastix.wordpress.com/2009/07/>.

CORMEN, T. et al. **Algoritmos:** Teoria e Prática. Tradução de Vandenberg Souza. 2. ed. Rio de Janeiro: Elsevier, v. 1, 2002.

FÜRER, M.; RAGHAVACHARI, B. **An NC Approximation Algorithm for the Minimum Degree Spanning Tree Problem**. Allerton Conference on Communication, Control and Computing. Philadelphia: [s.n.]. 1990. p. 274-281.

FÜRER, M.; RAGHAVACHARI, B. **Approximating the Minimum Degree Spanning Tree to Within One From the Optimal Degree**. ACM-SIAM Symposium on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics. 1992. p. 317-324.

GOMES, F. et al. Experimental analysis of approximation algorithms for the vertex cover and set covering problems. **Computers and Operations Research**, Oxford, 33, n. 12, Dezembro 2006. 3520-3534.

GRUBER, M.; HEMERT, J.; RAIDL, G. **Neighbourhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO**. Genetic And Evolutionary Computation Conference. New York: ACM. 2006. p. 1187-1194.

HASSIN, R.; TAMIR, A. On the Minimum Diameter Spanning Tree Problem. **Information Processing Letters**, Amsterdam, v. 53, n. 2, p. 109-111, January 1995.

HOCHBAUM, D. Approximation Algorithms for the Set Covering and Vertex Cover Problems. **SIAM Journal on Computing**, v. 11, n. 3, p. 555-556, 1982.

ILAN, J. B.; KORTSARZ, G.; PELEG, D. **Generalized Submodular Cover Problems and Applications**. Theoretical Computer Science. Essex: Elsevier Science Publishers Ltd. 2001. p. 179-200.

JAIN, K. et al. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. **Journal of the ACM**, New York, v. 50, n. 6, p. 795-824, November 2003.

KÖNEMANN, J.; RAVI, R. **A Matter of Degree:** Improved Approximation Algorithms for Degree Bounded Minimum Spanning Trees. SIAM Journal on

Computing. Philadelphia: Society for Industrial and Applied Mathematics. 2002. p. 1783-1793.

KÖNEMANN, J.; RAVI, R. **Primal-Dual Meets Local Search:** Approximating MSTs With Nonuniform Degree Bounds. SIAM Journal on Computing. Philadelphia: Society for Industrial and Applied Mathematics. 2005. p. 763-773.

KÖNEMANN, J.; SINHA, A.; LEVIN, A. Approximating the Degree-Bounded Minimum Diameter Spanning Tree Problem. **Algorithmica**, v. 41, n. 2, p. 117-129, November 2004.

KAO, M. Y. **Encyclopedia of Algorithms**. New York: Springer-Verlag, v. 1, 2007. 231-233 p.

SEDGEWICK, R. **Algorithms**. 1. ed. London: Addison-Wesley, v. 1, 1983.

TANG, S.-L. Minimum Spanning Trees and Clustering, 20 Abril 2010. Disponivel em: <http://www.cs.sjsu.edu/faculty/lee/cs157b/Minimum%20Spanning%20Tree%20and%20Clustering%20-%2004202010.ppt>. Acesso em: 12 Junho 2010.

TREVISAN, L. **Non-approximability Results for Optimization Problems on Bounded Degree Instances**. Proceedings of the thirty-third annual ACM symposium on Theory of computing. Nova Iorque: ACM. 2001. p. 453-461.

WESLEY, A. **java-alg**. Disponivel em: <http://java-alg.info/Addison.Wesley-Algorithms.in.Java.Third.Edition.Part.5-Graph.Algorithms/tindex.htm>. Acesso em: 26 Junho 2010.

WIKIMEDIA , 3 Julho 2008. Disponivel em: <http://commons.wikimedia.org/wiki/File:Graph_of_70-fullerene_w-nodes.svg>.