António Galrão Ramos
Pedro Rocha

Logística

Modelos, Algoritmos e Aplicações

CPLOT

António Galrão Ramos
Pedro Rocha

Logística

Modelos, Algoritmos e Aplicações

Center for Production and Logistics Optimization Technologies
Instituto Superior de Engenharia do Porto

Published by
Center for Production and Logistics Optimization Technologies
Instituto Superior de Engenharia do Porto

www.cplot.isep.ipp.pt

ISBN 978-989-35251-0-4

Agradecimentos

Gostaríamos de expressar a nossa mais profunda gratidão ao Doutor Fábio Neves-Moreira pela sua inestimável assistência ao longo da jornada de criação deste livro. Seu apoio inabalável, dedicação e experiência foram fundamentais para moldar esta obra.

Conteúdo

Capítulo 1
Conceitos Elementares de Grafos e Redes

Um **grafo** é uma representação visual de um sistema através de vértices e arestas.

A notação usada para representar um grafo é $G(V, E)$, em que V representa o conjunto de vértices e E o conjunto de arestas (edges).

Um grafo $G = (V, E)$ é uma estrutura discreta formada por um conjunto finito de vértices $V = \{1, 2, ..., n\}$ e um conjunto finito de arestas $E = \{(i, j)|i, j \in V\}$ que representam ligações entre estes vértices.

Uma aresta formada por um par **não ordenado** de vértices distintos ($i \neq j$), ($\{i, j\} = \{j, i\}$), diz-se **não orientada**. Se a aresta for representada por um par ordenado de vértices, diz-se **orientada**.

Um vértice i diz-se **adjacente** ao vértice j se existe uma aresta $(i, j) \in E$.

Um **caminho** é uma sequência de vértices e arestas.

Um caminho forma um **circuito**, se liga um vértice a si mesmo. Um **circuito directo** é um circuito em que todos os arcos estão orientados na mesma direcção.

Um grafo diz-se **conexo** se todos os pares de vértices estão ligados por um circuito.

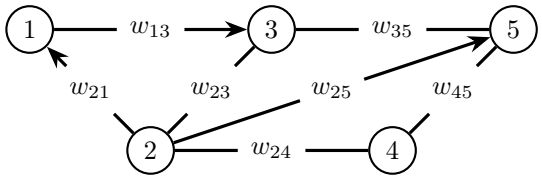
Uma **árvore** é um grafo com um e um só caminho entre quaisquer dois vértices. Uma **árvore de suporte** de um grafo $G(V, E)$ é uma árvore formada por arestas de E e que contenha todos os vértices V.

Uma **rede** é um grafo cujos vértices e/ou arestas têm associado atributos (por exemplo distâncias, custos, capacidades e/ou oferta e procura). Nas redes utiliza-se a designação nós e arcos, em vez de vértices e arestas, respectivamente. Uma rede $R = (V, A, C)$ em que (V, A) corresponde a um grafo e

C aos atributos dos nós e/ou arcos.

Como ilustração, a rede R na figura seguinte é descrita como:

- $V = 1, 2, 3, 4, 5$
- $A = (1, 3), (1, 2), (2, 3), (2, 4), (2, 5), (3, 5), (4, 5)$



Os arcos $(1, 3), (1, 2)$ e $(2, 5)$ são arcos orientados. Os arcos $(2, 3), (2, 4), (3, 5)$ e $(4, 5)$ são arcos não orientados.

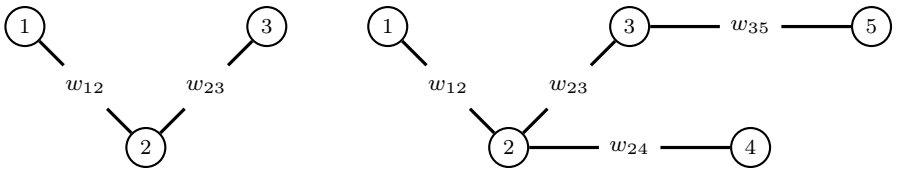


Figura 1.1 Árvore (Esquerda) vs Árvore Suporte (Direita)

Numa árvore com n nós, o número de arcos é $n - 1$.

Nos proximos capítulos serão apresentados alguns algorithmos que utilizam estes conceitos (nós, arestas, grafos, etc) para resolver problemas distintos. Um algoritmo é definido como um processo, ou conjunto de regras que devem ser seguidas na realização de calculos, ou outras operações que visam resolver problemas, normalmente utilizando um computador.

Os algoritmos abordados nos próximos capitulos são os seguintes:

- Árvore de Suporte de Custo Mínimo
- Caminho Mais Curto
- Problema do Fluxo Máximo
- Escalonamento e Planeamento com Janelas Temporais
- Localização de Instalações
- Caixeiro-Viajante
- Roteamento de Veículos

Capítulo 2
Árvore de Suporte do Custo Mínimo

2.1 Descrição

O problema da árvore de suporte do custo mínimo, trata a minimização do comprimento de uma árvore de suporte. O comprimento é igual ao valor da soma de todos os arcos da árvore de suporte. Uma vez que para cada rede existe mais do que uma árvore de suporte, a que apresenta o comprimento menor é a árvore de suporte do custo mínimo.

2.2 Formulação Matemática

Dado um grafo $G = (V, E)$ com n vértices, pesado com custos c_ij em cada aresta $(i, j) \in E$, e seja S um subconjunto de V.

Variáveis de Decisão

$$x_{ij} : \begin{cases} 1, & \text{se o arco } (i,j) \text{ fizer parte do caminho mais curto} \\ 0, & \text{caso-contrário} \end{cases} \tag{2.1}$$

Função Objectivo

$$min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{i,j} x_{i,j} \tag{2.2}$$

Restrições

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{i,j} = n - 1 \tag{2.3}$$

$$\sum_{i \in S}^{n-1} \sum_{j \in S, j>1}^{n} x_{i,j} \leq |S| - 1, \qquad \forall S \subseteq V\{1\}, |S| \geq 2 \tag{2.4}$$

$$x_{i,j} = \{0, 1\}, \forall (i, j) \in E \tag{2.5}$$

A variável de decisão especificada em (2.1) é binária, assumindo o valor de 1 caso o arco (i, j) seja utilizado, e 0 caso contrário. A função objectivo (2.2), agrega os custos $c_{i,j}$ de cada arco (i, j) seleccionado para proceder à sua minimização. A restrição (2.3) garante que o número mínimo de arcos seleccionados é $n - 1$. As restrições (2.4) garantem que não existem ciclos, sendo as restrições de eliminação de sub-circuitos (*subtours*). As restrições (2.5), garantem que as variáveis de decisão $x_{i,j}$ assumem apenas o valor de 0 ou 1.

2.3 Algoritmos

Para a solução do problema de árvore de suporte de custo mínimo são apresentados os seguintes algoritmos:

- Algoritmo de Prim Prim, 1957
- Algoritmo de Kruskal Kruskal, 1956

2.3.1 Algoritmo de Prim

2.3.1.1 Descrição

O algoritmo de Prim envolve os seguintes passos:

Passo 1 Definir para a rede $V = 1, 2, ..., n$ dois conjuntos:

- P_k - Conjunto de nós ligados permanentemente
- T_k - Conjunto de nós ligados temporariamente
- E - Arcos da Árvore de Suporte do Custo Mínimo

e afectar a $P_0 = \emptyset$ e a $T_0 = V$
Para $k = 1$ (sendo k o indicador da iteração actual) seleccionar arbitrariamente um nó, i, do conjunto de nós temporários (T_k) e fazer:

- $P_k = \{i\}$
- $T_k = V - \{i\}$

Passo 2 Incrementar k (fazendo $k = k + 1$), seleccionar do conjunto de nós temporários T_{k-1} um nó j que liga a um nó i do conjunto de nós perma-

nentes P_{k-1}, tendo o menor custo de ligação. O nó j passa a ser incluído em P_{k-1} sendo removido de T_{k-1}.

- $P_k = P_{k-1} + \{j\}$
- $T_k = T_{k-1} - \{j\}$
- $E = \{(i, j)\}$

Se o conjunto de nós temporários estiver vazio ($T_k = \emptyset$) então terminar o algoritmo. Enquanto $T_k \neq \emptyset$, repetir o 2º passo.

2.3.1.2 Pseudo-Código

O Algoritmo de Prim (Algoritmo 1) é um algoritmo que se baseia em seleccionar, a cada etapa, um vértice não pertencente à árvore com o menor custo de ligação (entre o vértice e um dos vértices pertencentes à árvore actual).

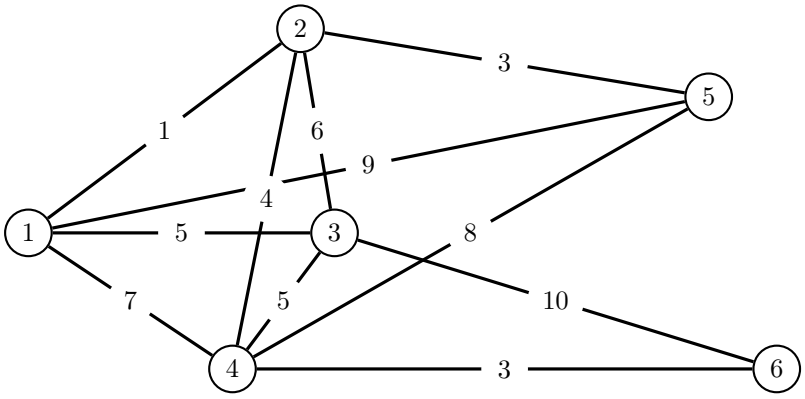
Algoritmo 1: Algoritmo de Prim

```
Entrada: Grafo G
Saída: Arestas da Árvore Suporte Custo Mínimo
início
    cria grupo de vértices Q com vértices de G(V)
    para cada vértice u ∈ Q faça
        prioridade[u] = ∞                ▷ distancia de vértice u é ∞
        parente[u] = ∞   ▷ Nodo anterior a vértice u não definido.
    fim
    prioridade[s] = 0                    ▷ Prioridade de vértice s é 0
    parente[s] = ∅           ▷ Nodo anterior a vértice s não existe.
    enquanto Q ≠ ∅ faça
        u = MinDistance(Q)   ▷ Selecciona o vértice mais próximo u
        do conjunto Q
        remove u de Q
        para Cada vizinho V de U faça
            se v ∈ Q e distancia(u, v) < prioridade[v] então
                parente[v] = u          ▷ Elimina algum vértice contido
                prioridade[v] = distancia(u, v)     ▷ Elimina arcos ligados
            fim
        fim
    fim
    Retorna parente[ ], prioridade[ ]
fim
```

2.3.1.3 Ilustração

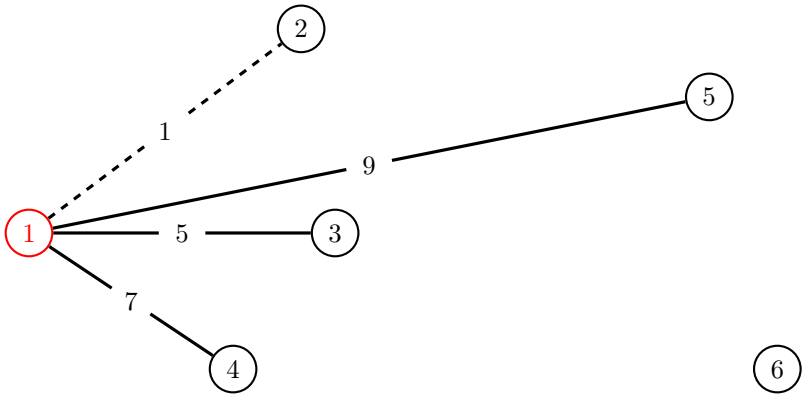
Considere-se o seguinte exemplo para ilustração do algoritmo:

Uma companhia de TV por Cabo está em processo de fornecer serviço de cabo a cinco novas áreas habitacionais. A figura seguinte, mostra as ligações potenciais entre as cinco áreas. As distâncias de cabos em km são indicadas em cada arco. Quais as ligações a realizar de forma a minimizar o comprimento de cabo a utilizar?



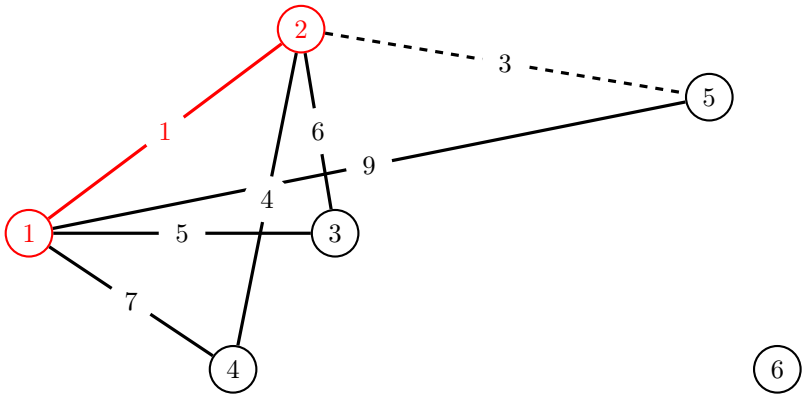
Passo 1 Iniciar $K = 1$. Iniciar o algoritmo no nó 1 (pode ser outro qualquer), o que dá:
$P_1 = \{1\}$
$T_1 = \{2, 3, 4, 5, 6\}$

Passo 2 Fazer $k = 2$. O menor arco, que liga P_1 a T_1 é o arco (1, 2).
$P_2 = 1, 2$
$T_2 = \{3, 4, 5, 6\}$
$E = \{(1, 2)\}$
Como $T_1 \neq \emptyset$ repetir o $2^{\underline{o}}$ passo.



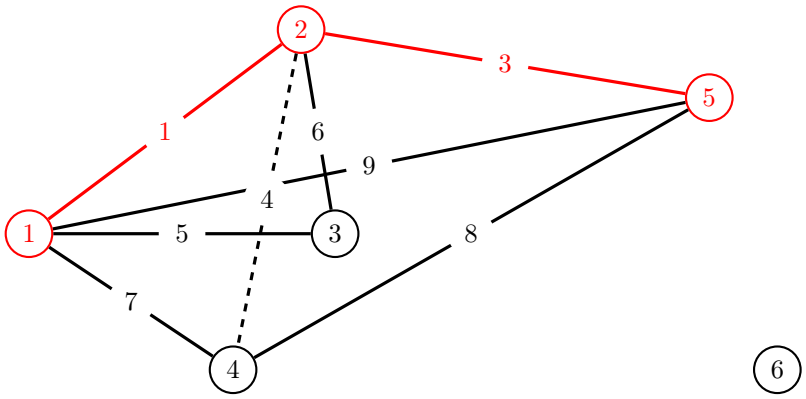
Passo 2 Fazer $k = 3$. O menor arco, que liga P_2 a T_2 é o arco (2, 3).
$P_3 = 1, 2, 3$
$T_3 = \{4, 5, 6\}$
$E = \{(1, 2), (2, 3)\}$

Como $T_3 \neq \emptyset$ repetir o 2º passo.

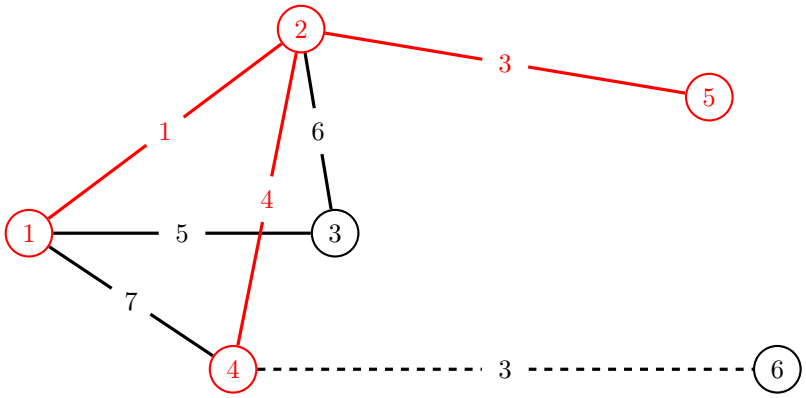


Passo 2 Fazer $k = 4$. O menor arco, que liga P_3 a T_3 é o arco (2, 4).
$P_4 = 1, 2, 3, 4$
$T_4 = \{5, 6\}$
$E = \{(1, 2), (2, 3), (2, 4)\}$

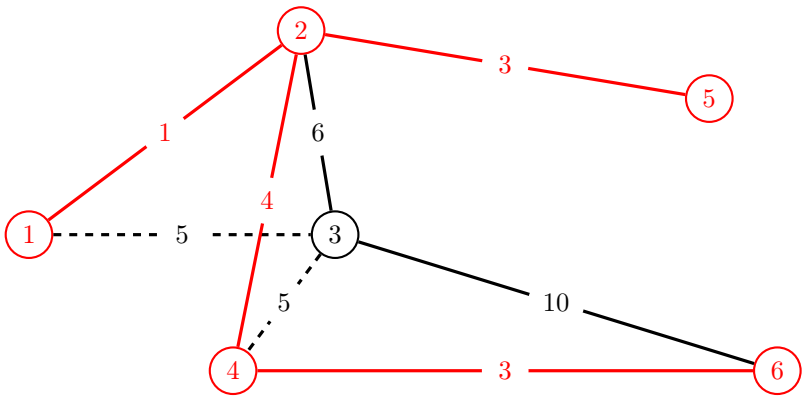
Como $T_4 \neq \emptyset$ repetir o passo 2.



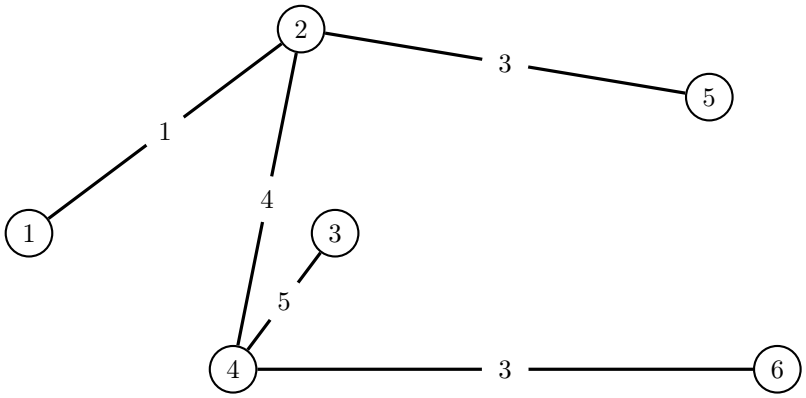
Passo 2 Fazer $k = 5$. O menor arco, que liga P_4 a T_4 é o arco (4, 6).
$P_5 = \{1, 2, 3, 4, 6\}$
$T_5 = \{6\}$
$E = \{(1, 2), (2, 3), (2, 4), (4, 6)\}$
Como $P_5 \neq \emptyset$ repetir o passo 2.



Passo 2 Fazer $k = 6$. A ligação de P_5 a T_5 com o menor arco pode ser o arco (1, 3) ou o arco (4, 3). A escolha é arbitrária.
$P_6 = \{1, 2, 3, 4, 5, 6\}$
$T_6 = \emptyset$
$E = \{(1, 2), (2, 3), (2, 4), (4, 6), (3, 4)\}$
Como $T_6 = \emptyset$ a solução é óptima.



A figura seguinte ilustra a Árvore de Suporte do Custo Mínimo.



O número mínimo de Km necessários é de $1 + 3 + 4 + 3 + 5 = 16$ Km.

2.3.2 Algoritmo de Kruskal

2.3.2.1 Descrição

O algoritmo de Kruskal envolve os seguintes passos:

Iteração 1 Definir para o grafo G composto pelos vértices $V = 1, 2, ..., n$ os conjuntos:

- E - Arestas da Árvore de Suporte do Custo Mínimo
- Q - Conjunto de arestas ordenado
- A - Conjunto de grupos de vértices

e inicializar $E = \emptyset$ e $A = (1), (...), (n)$.
Seleccionar arestas (u_n, v_n) na iteração $k = 1$ com menor custo do conjunto Q e faça:

- $E = \{(u_1, v_1)\}$
- $A = \{(u_1, v_1)\}$

Iteração 2 Faça $k = k + 1$. Seleccionar a aresta (u, v) na iteração $k = 2$ com menor custo do conjunto Q e faça:

- $E = \{(u_1, v_1), (u_2, v_2)\}$
- Sendo $v_1 = u_2$ adicionar v_2 a $A = \{(u_1, v_1, v_2)\}$

Se todos os vértices estiverem no mesmo grupo o algoritmo termina.

Iteração n Faça $k = k + 1$
Seleccionar a aresta (u, v) na iteração $k = m$ com menor custo do conjunto Q e fazer:

- $E = \{(u_1, v_1), (u_2, v_2), ..., (u_m, v_m)\}$

- Unir os grupos dos vértices u e v: $Arvore(u) \cup Arvore(v)$

O algoritmo termina quando todos os vértices estiverem no mesmo conjunto, e devolve as arestas em E.

2.3.2.2 Pseudo-Código

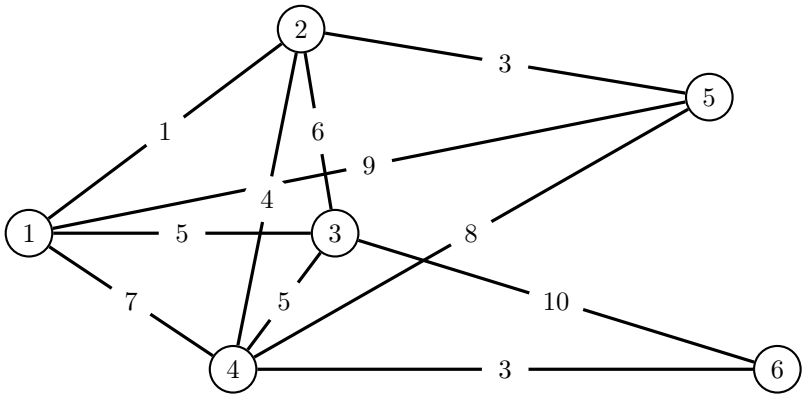
O Algoritmo de Kruskal (Algoritmo 2) baseia-se em seleccionar, a cada etapa, a aresta com menor custo salvaguardando a condição que a estrutura se mantém acíclica.

Algoritmo 2: Algoritmo de Kruskal

Entrada: Grafo G
Saída: Arestas da Árvore Suporte Custo Mínimo
início
  $E = \emptyset$ ▷ Cria conjuntos de arestas seleccionadas
  conjunto Q = arestas de G ordenadas por ordem não decrescente de custo
  para *cada aresta* $u, v \in Q$ **faça**
    se $Arvore(u) \neq Arvore(v)$ **então**
      ▷ Se v e u pertencerem a diferentes grupos
      $E = E \cup (u, v)$ ▷ Adiciona aresta (u, v) ao conjunto E
      $Arvore(v) \cup Arvore(u)$ ▷ Junta os grupos de u e v
    fim
  fim
  Retorna $E[\ ]$,
fim

2.3.2.3 Ilustração

Inicialização Inicia $k = 1$. O grafo original corresponde ao seguinte:



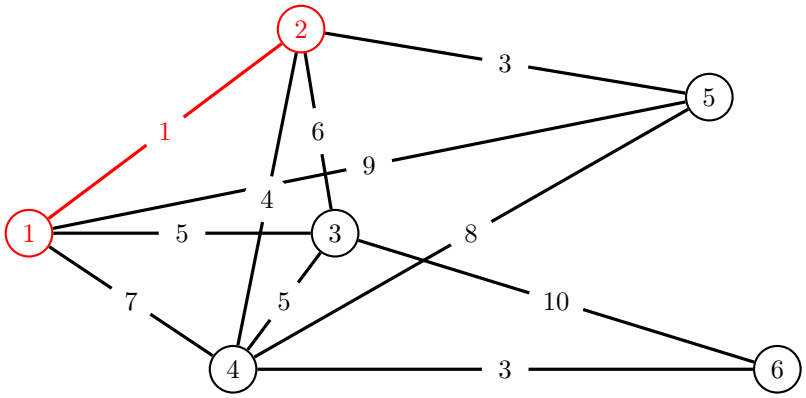
A sequência de arestas ordenadas por custo estão apresentado na Tabela 2.1.

Aresta	Custo	Conjunto	Selecção
1,2	1	-	-
2,5	3	-	-
4,6	3	-	-
2,4	4	-	-
1,3	5	-	-
3,4	5	-	-
2,3	6	-	-
1,4	7	-	-
1,5	9	-	-
3,6	10	-	-

Tabela 2.1 Arestas seleccionadas - Inicialização

Os valores dos conjuntos são:
$E = \{\emptyset\}$
$A = \{(1), (2), (3), (4), (5), (6)\}$

Iteração 1 A primeira aresta seleccionada é a aresta com menor custo $(1, 2)$.
Com a selecção dessa aresta, os seus vértices são juntos no conjunto A.

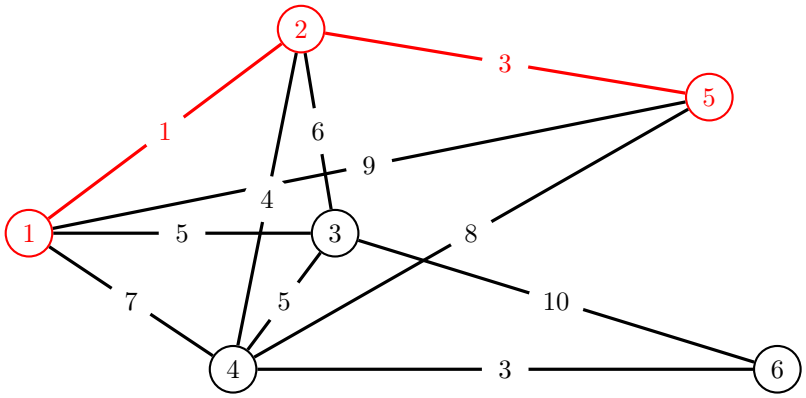


Aresta	Custo	Conjunto	Selecção
1,2	1	(1,2)	Sim
2,5	3	-	-
4,6	3	-	-
2,4	4	-	-
1,3	5	-	-
3,4	5	-	-
2,3	6	-	-
1,4	7	-	-
1,5	9	-	-
3,6	10	-	-

Tabela 2.2 Arestas seleccionadas - Iteração 1

Conjuntos:
$E = \{(1,2)\}$
$A = \{(1,2),(3),(4),(5),(6)\}$

Iteração 2 A próxima aresta seleccionada é a segunda aresta com menor custo $(2,5)$. Essa aresta tem os dois vértices em dois grupos diferentes de (A), logo o novo vértice é junto a $(1,2)$.

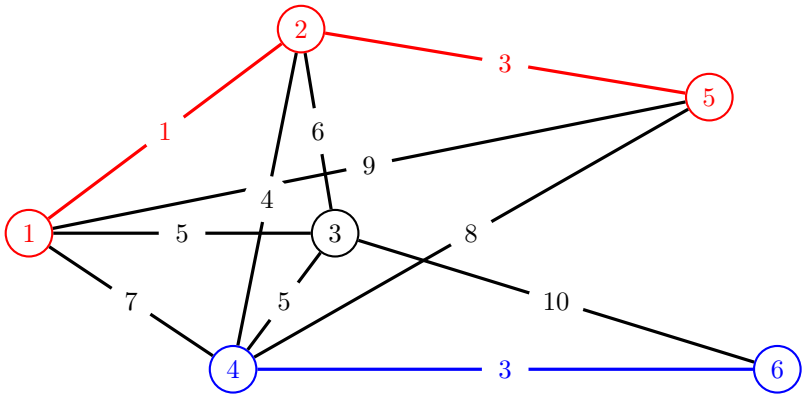


Aresta	Custo	Conjunto	Selecção
1,2	1	(1,2)	Sim
2,5	3	(1, 2, 5)	Sim
4,6	3	-	-
2,4	4	-	-
1,3	5	-	-
3,4	5	-	-
2,3	6	-	-
1,4	7	-	-
1,5	9	-	-
3,6	10	-	-

Tabela 2.3 Arestas seleccionadas - Iteração 2

Conjuntos:
$E = \{(1,2),(2,5)\}$
$A = \{(1,2,5),(3),(4),(6)\}$

Iteração 3 A próxima aresta seleccionada é $(4,6)$. Os vértices pertencem a diferentes grupos.

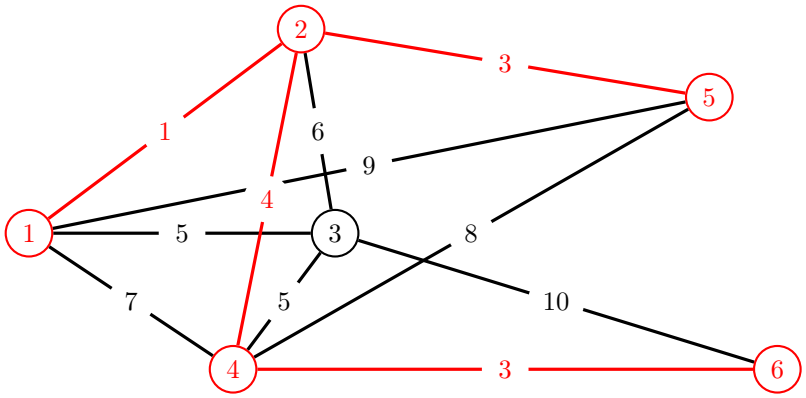


Aresta	Custo	Conjunto	Selecção
1,2	1	(1,2)	Sim
2,5	3	(1,2,5)	Sim
4,6	3	(4,6)	Sim
2,4	4	-	-
1,3	5	-	-
3,4	5	-	-
2,3	6	-	-
1,4	7	-	-
1,5	9	-	-
3,6	10	-	-

Tabela 2.4 Arestas seleccionadas - Iteração 3

Conjuntos:
$E = \{(1,2),(2,5),(4,6)\}$
$A = \{(1,2,5),(3),(4,6)\}$

Iteração 4 A próxima aresta seleccionada é $(2,4)$. Os vértices desta aresta pertencem a grupos diferentes, $(1,2,5)$ e $(4,6)$. Como tal, os vértices de ambos os grupos são juntos num único grupo.

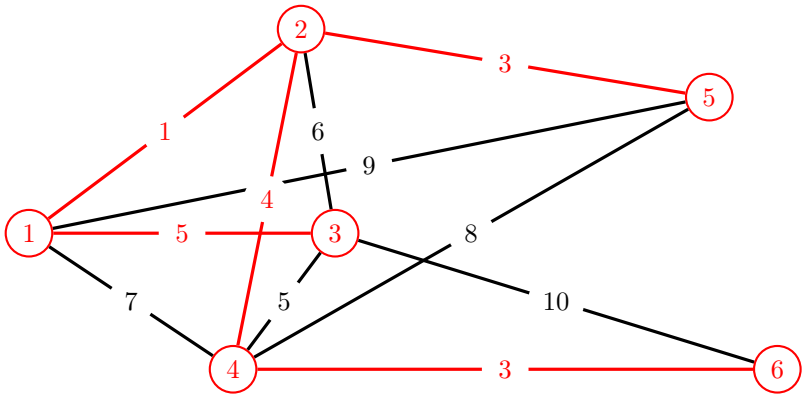


Aresta	Custo	Conjunto	Selecção
1,2	1	(1,2)	Sim
2,5	3	(1,2,5)	Sim
4,6	3	(4,6)	Sim
2,4	4	(1,2,4,5,6)	Sim
1,3	5	-	-
3,4	5	-	-
2,3	6	-	-
1,4	7	-	-
1,5	9	-	-
3,6	10	-	-

Tabela 2.5 Arestas seleccionadas - Iteração 4

Conjuntos:
$E = \{(1,2),(2,5),(4,6),(2,4)\}$
$A = \{(1,2,4,5,6),(3)\}$

Iteração 5 A próxima aresta seleccionada é $(1,3)$ terminando o algoritmo por não haver mais vértices por ligar. O vértice é adicionado ao grupo (1,2,4,5,6), as arestas seleccionadas até à iteração actual são devolvidas como resultado sendo todas as restantes descartadas. O resultado final está apresentado abaixo, com o valor final para a árvore de suporte de custo mínimo de $1+3+3+4+5=16$.



Aresta	Custo	Conjunto	Selecção
1,2	1	(1,2)	Sim
2,5	3	(1,2,5)	Sim
4,6	3	(4,6)	Sim
2,4	4	(1,2,4,5,6)	Sim
1,3	5	(1,2,3,4,5,6)	Sim
3,4	5	-	-
2,3	6	-	-
1,4	7	-	-
1,5	9	-	-
3,6	10	-	-

Tabela 2.6 Arestas seleccionadas - Iteração 5

Conjuntos:
$E = \{(1,2),(2,5),(4,6),(2,4),(1,3)\}$
$A = \{(1,2,3,4,5,6)\}$

Capítulo 3
Caminho Mais Curto

3.1 Descrição

O problema do caminho mais curto determina o caminho mais curto entre um nó de origem (s) e um nó de destino (t) numa rede $R = V, A, C$.

3.2 Formulação

Variáveis de Decisão

$$x_{ij} : \begin{cases} 1, & \text{se o arco (i,j) fizer parte do caminho mais curto} \\ 0, & \text{caso-contrário} \end{cases} \tag{3.1}$$

Função Objectivo

$$min \sum_{(i,j)\in A}^{t} c_{i,j} x_{i,j} \tag{3.2}$$

Restrições

$$-\sum_{(i,j)\in A}^{t} x_{i,j} + \sum_{(j,k)\in A}^{t} x_{j,k} = \begin{cases} 1, & \text{se j=s} \\ 0, & \text{se j} \neq \text{s,t} \\ -1, & \text{se j=t} \end{cases} \tag{3.3}$$

$$X_{i,j} = \{0, 1\}, \forall (i, j) \in A \tag{3.4}$$

A variável de decisão especificada em (3.1) é binária, assumindo o valor de 1 caso o arco (i, j) seja utilizado, e 0 caso contrário. A função objectivo, apresentada em (3.2), agrega os custos de $c_{i,j}$ de cada arco (i, j) seleccionado para proceder à sua minimização. As restrições (3.3) garantem que cada nó

tem obrigatoriamente um arco de entrada, e um arco de saída, excepto o nó inicial (que só tem um arco de saída) e o nó final (que só tem um arco de entrada). As restrições (3.4), garantem que a variável de decisão $x_{i,j}$ assume apenas o valor de 0 ou 1 (variável binária).

3.3 Algoritmos

Para a solução do problema do caminho mais curto são apresentados os seguintes algoritmos:

- Algoritmo de Dijkstra Dijkstra, 1959
- Algoritmo de Floyd-Warshall Floyd, 1962
- Algoritmo de Bellman-Ford Bellman, 1958

Estes algoritmos são frequentemente utilizados para lidar com problemas de minimização.

3.3.1 Algoritmo de Dijkstra

3.3.1.1 Descrição

O algoritmo de Dijkstra permite determinar o caminho mais curto entre um nó de origem e outro qualquer nó da rede.

O algoritmo associa um rótulo a cada nó, que corresponde à distância mais curta entre o nó e o nó de origem.

A computação do algoritmo avança desde o nó i até ao imediatamente seguinte j utilizando um procedimento de rotulagem.

Assumindo u_i como sendo a distância mais curta do nó 1 até ao nó i e $d_{ij}(\geq 0)$ definido como comprimento do arco (i, j), o rótulo para o nó j é definido da forma seguinte:

$$[u_j, i] = [u_i + d_{ij}, i], d_{ij} \geq 0 \tag{3.5}$$

Os rótulos no algoritmo de Dijkstra são de dois tipos:

- T:temporários - representam o limite superior da distancia mais curta até ao nó.
- P:permanentes. - Representa a distância mais curta até ao nó.

Sempre que uma rota mais curta é identificada até um determinado nó, o rótulo temporário é modificado. Quando nenhuma rota mais curta puder ser encontrada, o rótulo muda de tipo temporário para tipo permanente.

Os passos principais do algoritmo de Dijkstra são os seguintes:

Passo 0 Rotular o nó de origem (nó 1) com o rótulo permanente [0,-] e definir $i = 1$.

Passo 1 Calcular os rótulos $[u_i + d_{ij}, i]$ para cada nó j temporário que pode ser acedido a partir do nó i. Se o nó j já está rotulado com $[u_j, k]$ através de um outro nó k, e se $u_i + d_{ij} < u_j$ então substituir $[u_j, k]$ por $[u_i + d_{ij}, i]$.

Passo 2 Seleccione o rótulo $[u_r, s]$ com a distância mais curta $(= u_r)$ a partir dos nós temporários disponíveis. Se todos os nós tiverem rótulos permanentes terminou o algoritmo senão fazer $i = r$ e regressar ao **Passo 1**.

3.3.1.2 Pseudo-código

O pseudo-código é apresentado no Algoritmo 3.

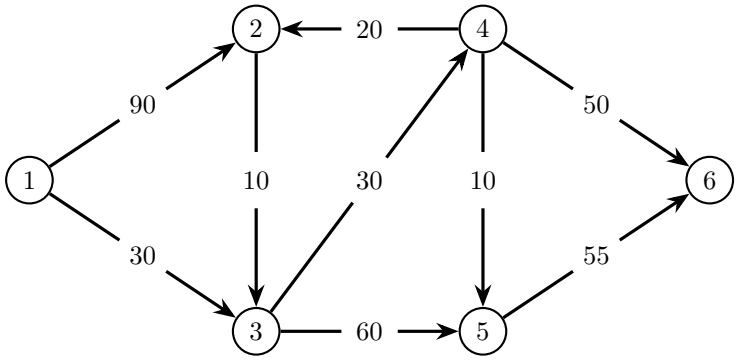
Algoritmo 3: Algoritmo de Dijkstra

```
Entrada: Grafo, Origem
Saída: Número esperado de nós atingidos
início
    cria grupo de vértices Q
    para cada vértice v no Grafo faça
        distancia[v] = ∞   ▷ Distancia inicial de Origem a v é ∞
        anterior[v] = indefinido   ▷ Nodo anterior em caminho óptimo da Origem
        adiciona v a Q
    fim
    distancia[Origem] = 0   ▷ Distancia de Origem a Origem
    enquanto Q não está vazio faça
        u = nó em Q com menor distancia[u]   ▷ Nó com menor distancia seleccionado primeiro
        remove u de Q
        para Cada vizinho v de u faça
            alternativa = distancia[u] + distancia_entre(u, v)
            se alternativa < distancia[v] então
                distancia[v] = alternativa   ▷ Elimina algum nó contido
                anterior[v] = u   ▷ Elimina arcos ligados
            fim
                ▷ Se caminho mais curto foi descoberto
        fim
                                ▷ Em que v ainda está em Q
    fim
    Retorna distancia[], anterior[]
fim
```

3.3.1.3 Ilustração

O seguinte exemplo ilustra a aplicação do Algoritmo de Dijkstra.

A rede da figura seguinte apresenta os itinerários possíveis e as suas distâncias em km entre a cidade 1 (nó 1) e quatro outras cidades (nós 2 a 6). O objectivo é determinar a distância mais curta desde a cidade 1 a cada uma das restantes cidades.



Iteração 0 Atribuir o rótulo permanente $[0, -]$ ao nó 1.

Iteração 1 A partir do nó 1 (o ultimo nó rotulado de permanente) podem ser acedidos os nós 2 e 3. A lista dos nós rotulados (permanentes e temporários) fica:

Nó	Rótulo	Estado
1	**[0,-]**	**Permanente**
2	[0+90,1]=[90,1]	Temporário
3	[0+30,1]=[30,1]	Temporário
4	[∞,1]	Temporário
5	[∞,1]	Temporário
6	[∞,1]	Temporário

Para os dois rótulos $[90, 1]$ e $[30, 1]$, o nó 3 retorna a menor distancia ($u_3 = 30$). Assim, o estado do nó 3 é alterado para permanente. Uma vez que os nós 2, 4, 5 e 6 são temporários, coloca-se $i = 3$.

Iteração 2 Dado que os nós 4 e 5 podem ser alcançados a partir do nó 3 (o último a ser rotulado como permanente), a lista de nós rotulados fica:

Nó	Rótulo	Estado
1	**[0,-]**	**Permanente**
2	[90,1]	Temporário
3	**[30,1]**	**Permanente**
4	[30+30,3]=[60,3]	Temporário
5	[30+60,3]=[90,3]	Temporário
6	[∞,1]	Temporário

O estado do rótulo temporário no nó 4 é alterado para permanente ($u_4 = 60$), ficando o nó 4 com o estado permanente. Com os nós 2, 4 e 5 a temporários coloca-se $i = 4$.

Iteração 3 Os nós 2, 5 e 6 podem ser alcançados a partir do nó 4. A lista de nós rotulados fica:

Nó	Rótulo	Estado
1	**[0,-]**	**Permanente**
2	[60+20,4]=[80,4]	Temporário
3	**[30,1]**	**Permanente**
4	**[60,3]**	**Permanente**
5	[60+10,4]=[70,4]	Temporário
6	[60+50,4]=[110,4]	Temporário

O rótulo temporário com distância mais curta é o nó 5 ($u_5 = 70$), sendo este seleccionado e passado para permanente. Sendo os nós 2 e 6 temporários, $i = 5$.

Iteração 4 A partir do nó 5 pode-se aceder ao nó 6. O rótulo não é actualizado pois a distância não é inferior à actualmente presente. A lista dos nós rotulados fica:

Nó	Rótulo	Estado
1	**[0,-]**	**Permanente**
2	[80,4]	Temporário
3	**[30,1]**	**Permanente**
4	**[60,3]**	**Permanente**
5	**[70,4]**	**Permanente**
6	[110,4]	Temporário

O rótulo temporário com distância mais curta é o nó 2 ($u_2 = 80$), ficando o nó 2 a permanente.Sendo o nó 6 temporário, fica $i = 5$.

Iteração 5 O nó 2 não permite aceder a mais nenhum nó. Como tal a lista de nós rotulados fica:

Nó	Rótulo	Estado
1	**[0,-]**	**Permanente**
2	**[80,4]**	**Permanente**
3	**[30,1]**	**Permanente**
4	**[60,3]**	**Permanente**
5	**[70,4]**	**Permanente**
6	[110,4]	Temporário

O rótulo temporário com distância mais curta é o nó 6 ($u_6 = 110$). O estado do nó 6 passa a permanente e sem mais nós temporários para seleccionar o algoritmo termina.

O caminho mais curto entre nós 1 e qualquer outro nó é determinado começando no nó de destino e verificando em sentido reverso através dos nós usando a informação dos nós permanentes da tabela. Considerando o exemplo anterior, o caminho mais curto do nó 1 para o nó 2 é:

$$(2) \rightarrow [80, 4] \rightarrow (4) \rightarrow [60, 3] \rightarrow (3) \rightarrow [30, 1] \tag{3.6}$$

Logo, a rota desejada é $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$, com um comprimento total de 80 km.

3.3.1.4 Resolução Tabular

As iterações principais do algoritmo de Dijkstra estão presentes na Tabela 3.1.

Na primeira iteração, o conjunto S ainda não tem nós. Esse conjunto (S) indica os nós já visitados que foram seleccionados com custo mínimo ao nó inicial (1) sendo marcados como permanentes. O nó inicial é marcado como permanente sendo seleccionado como ponto de actualização dos valores dos outros nós conectados a ele.

A cada iteração, os valores de distância ao nó actual (ultimo marcado como permanente) são actualizados (ficando esses nós com a última distância mínima conhecida para o nó de origem (1). Caso a distância seja igual ou superior, então corta-se o valor indicando não actualizado. Após a actualização das distancias, é seleccionado um novo nó permanente escolhendo o que tem a menor distância à origem (entre os não permanentes), e adicionado ao conjunto S. Repete iterativamente este passo até todos os nós serem marcados como permanentes.

Iteração	S	L(1)	L(2)	L(3)	L(4)	L(5)	L(6)
1	$\emptyset$	0	∞	∞	∞	∞	∞
2	$\{1\}$	–	90	30	∞	∞	∞
3	$\{1, 3\}$	–	–	–	60	90	∞
4	$\{1, 3, 4\}$	–	80	–	–	70	110
5	$\{1, 3, 4, 5\}$	–	–	–	–	–	~~110~~
6	$\{1, 3, 4, 5, 2\}$	–	–	~~90~~	–	–	–
7	$\{1, 3, 4, 5, 2, 6\}$	–	–	–	–	–	–

Tabela 3.1 Método Tabular

A distância de cada nó à origem é feita através de *backtracking*, seleccionando na tabela o nó de destino desejado e verificando qual o nó a partir do qual a sua distância mínima foi calculada. Como exemplo, na Tabela 3.1, se escolher o nó 6, pode verificar que o seu valor mínimo foi encontrado na iteração 4, a partir do nó 4. O primeiro ramo de *backtrack* fica com $6 - 4$. Verificando o nó 4, verifica-se que o mínimo foi encontrado na iteração 3, vindo do nó 3. O caminho de *backtrack* fica com $6 - 4 - 3$. O nó que deu origem ao caminho mínimo para o nó 3 foi o nó 1, logo o caminho de *backtrack* fica com $6 - 4 - 3 - 1$. Este procedimento pode ser utilizado para verificar a distância de qualquer nó ao nó de origem. Com este algoritmo podem ser também determinados caminhos mais curtos entre pares de nós desde que

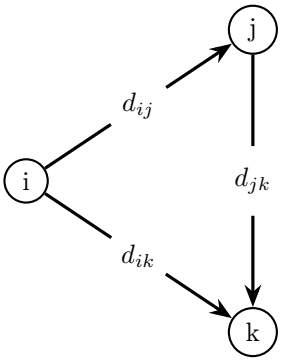
pertençam a qualquer um dos caminhos feitos por *backtracking* até ao nó de origem.

3.3.2 Algoritmo de Floyd–Warshall

3.3.2.1 Descrição

O Algoritmo de Floyd é mais geral que o Algoritmo de Dijkstra, uma vez que determina o caminho mais curto entre dois qualquer nós da rede. O algoritmo representa uma rede com n nós como uma matriz quadrada com n linhas e n colunas. A entrada $d_{i,j}$ indica a distância do nó i ao nó j, que é finito se i está ligado directamente a j e infinito caso não esteja. Dados os três nós i, j e k da figura seguinte, com as distâncias entre eles indicadas nos três arcos, é mais curto aceder a k de i passando por j se:

$$d_{ij} + d_{jk} < d_{ik}$$



Neste caso, é óptimo substituir o caminho directo i, k pelo caminho indirecto i, j, k. Esta operação tripla é aplicada sistematicamente a toda a rede usando os passos seguintes:

Passo 1 Fazer $k = 0$. Definir a matriz de distâncias inicial D_0 e a matriz de sequência de nós S_0 como indicado na tabela 3.2 e tabela 3.3. Os elementos da diagonal principal estão marcados com $(-)$ para indicar que estão bloqueados.

Passo 2 Fazer $k = k + 1$. Definir a linha k e a coluna k como linha e coluna pivot. Aplicar a operação tripla a cada elemento d_{ij} em D_{k-1}, para todos os i e j. Se a condição

$$d_{ik} + d_{kj} < d_{ij}, (i \neq k, j \neq k, ei \neq j) \tag{3.7}$$

$D_0 =$

	1	2	...	j	...	n
1	-	d_{12}	...	d_{1j}	...	d_{1n}
2	d_{21}	-	...	d_{2j}	...	d_{2n}
.	...	...	.	.	.	.
i	d_{i1}	d_{i2}	...	d_{ij}	...	d_{in}
.	...	...	.	.	.	.
n	d_{n1}	d_{n1}	...	d_{nj}	...	-

Tabela 3.2 Matriz distância inicial

$S_0 =$

	1	2	...	j	...	n
1	-	2	...	j	...	n
2	1	-	...	j	...	n
.	...	...	.	.	.	.
i	1	2	...	j	...	n
.	...	...	.	.	.	.
n	1	2	...	j	...	-

Tabela 3.3 Matriz sequencia nós

for satisfeita, modificar a matriz de distancias e a matriz de sequencia da forma seguinte:

- Criar D_k substituindo d_{ij} em D_{k-1} por $d_{ik} + d_{kj}$.
- Criar S_k substituindo s_{ij} em S_{k-1} por k.
- Se $k = n$ terminar o algoritmo, senão repetir o 2º passo.

O algoritmo pode ser melhor compreendido pela análise da tabela 3.4 ;

	Coluna j	Coluna Pivot k	Coluna q
Linha i	d_{ij}	d_{jk}	d_{iq}
.	...	...	...
Linha Pivot k	d_{kj}	...	d_{kq}
.	...	...	...
Linha p	d_{pj}	d_{pk}	d_{pq}

Tabela 3.4 Contrução matriz distâncias

A distância mais curta entre os nós i e j, é obtida por d_{ij} na matriz D_n. Para determinar o caminho mais curto entre dois nós i e j verificar se $s_{ij} = i$. Se $s_{ij} = i$ existe um ligação directa entre i e j. Senão, i e j estão ligados através de pelo menos outro nó e o caminho é obtido retrocedendo a partir de j.

3.3.2.2 Pseudo-código

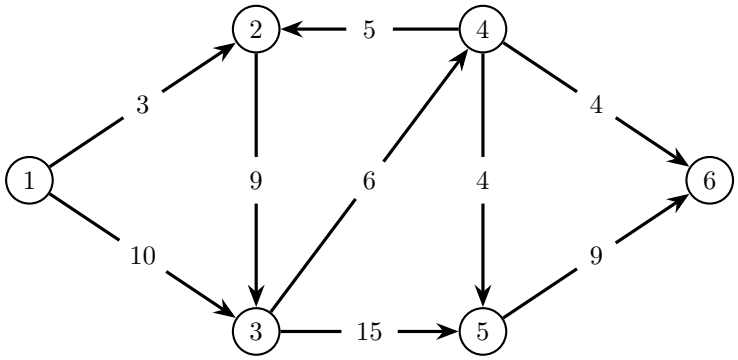
O pseudo-código pode ser visto no Algoritmo 4.

Algoritmo 4: Algoritmo de Floyd–Warshall

```
Entrada: Rede
Saída: Distâncias mínimas entre vértices v
início
    criar matriz de distâncias entre nós v * v inicializado a ∞
    para cada nó v faça
        distancia[v][v] = 0
    fim
    para cada arco u, v faça
        distancia[u][v] = W(u, v)        ▷ Distancia inicial de
        Origem a v é ∞
    fim
    para k de 1 até v faça
        para i de 1 até v faça
            para j de 1 até v faça
                se distancia[i][j] > distancia[i][k] + distancia[k][j]
                 então
                    distancia[i][j] = distancia[i][k] + distancia[k][j]
                fim
            fim
        fim
    fim
    Retorna distancia[]
fim
```

3.3.2.3 Ilustração

O seguinte exemplo ilustra o Algoritmo de Floyd–Warshall. Para a rede da figura 3.3.2.3 encontrar o caminho mais curto entra dois quaisquer nós. As distâncias (em Km) estão indicadas nos arcos. O arco (4,6) permite tráfico nas ambas as direcções. Todos os outros arcos permitem tráfico apenas numa direcção.



Passo 1 Fazer $k = 0$.

As matrizes D_0 e S_0 dão a representação inicial da rede.

D_0	1	2	3	4	5	6
1	-	3	10	∞	∞	∞
2	∞	-	9	∞	∞	∞
3	∞	∞	-	6	15	∞
4	∞	∞	∞	-	4	4
5	∞	∞	∞	∞	-	9
6	∞	∞	∞	4	∞	-

S_0	1	2	3	4	5	6
1	-	2	3	4	5	6
2	1	-	3	4	5	6
3	1	2	-	4	5	6
4	1	2	3	-	5	6
5	1	2	3	4	-	6
6	1	2	3	4	5	-

Tabela 3.5 Representação inicial rede (distâncias e sequências)

Passo 2 Fazer $k = 1$. A linha pivot e a coluna pivot são indicadas pela linha e coluna sombreadas indicadas na matriz D_0. Não existem elementos que podem ser melhorados pela operação tripla.

D_1	1	2	3	4	5	6
1	-	3	10	∞	∞	∞
2	∞	-	9	∞	∞	∞
3	∞	∞	-	6	15	∞
4	∞	∞	∞	-	4	4
5	∞	∞	∞	∞	-	9
6	∞	∞	∞	4	∞	-

S_1	1	2	3	4	5	6
1	-	2	3	4	5	6
2	1	-	3	4	5	6
3	1	2	-	4	5	6
4	1	2	3	-	5	6
5	1	2	3	4	-	6
6	1	2	3	4	5	-

Tabela 3.6 Representação rede $k = 1$ (distâncias e sequências)

Como $(k = 1) \neq (n = 6)$ avançar para o 2º passo.

Passo 2 Fazer $k = 2$. Não existem elementos que podem ser melhorados pela operação tripla.

Passo 2 Fazer $k = 3$. Os elementos d_{14}, d_{15}, d_{24} e d_{25}, são os que podem ser melhorados pela operação tripla. Criar D_3 e S_3 a partir de D_2 e S_2.

1. Substituir d_{14} por $d_{13} + d_{34} = 10 + 6 = 16$ e fazer $s_{24} = 3$.

D_2	1	2	3	4	5	6
1	-	3	10	∞	∞	∞
2	∞	-	9	∞	∞	∞
3	∞	∞	-	6	15	∞
4	∞	∞	∞	-	4	4
5	∞	∞	∞	∞	-	9
6	∞	∞	∞	4	∞	-

S_2	1	2	3	4	5	6
1	-	2	3	4	5	6
2	1	-	3	4	5	6
3	1	2	-	4	5	6
4	1	2	3	-	5	6
5	1	2	3	4	-	6
6	1	2	3	4	5	-

Tabela 3.7 Representação rede $k = 2$ (distâncias e sequências)

2. Substituir d_{15} por $d_{13} + d_{35} = 10 + 15 = 25$ e fazer $s_{25} = 3$.
3. Substituir d_{24} por $d_{23} + d_{34} = 9 + 6 = 15$ e fazer $s_{24} = 3$.
4. Substituir d_{25} por $d_{23} + d_{35} = 9 + 15 = 24$ e fazer $s_{25} = 3$.

D_3	1	2	3	4	5	6
1	-	3	10	**16**	**25**	∞
2	∞	-	9	**15**	**24**	∞
3	∞	∞	-	6	15	∞
4	∞	∞	∞	-	4	4
5	∞	∞	∞	∞	-	9
6	∞	∞	∞	4	∞	-

S_3	1	2	3	4	5	6
1	-	2	3	**3**	**3**	6
2	1	-	3	**3**	**3**	6
3	1	2	-	4	5	6
4	1	2	3	-	5	6
5	1	2	3	4	-	6
6	1	2	3	4	5	-

Tabela 3.8 Representação rede $k = 3$ (distâncias e sequências)

Como $(k = 3) \neq (n = 6)$ avançar para o 2º passo.

Passo 2 Fazer $k = 4$. Os elementos $d_{15}, d_{16}, d_{25}, d_{26}, d_{35}$ e d_{36}, são os que podem ser melhorados pela operação tripla. Criar D_4 e S_4 a partir de D_3 e S_3.

- Substituir d_{15} por $d_{14} + d_{45} = 16 + 4 = 20$ e fazer $s_{15} = 4$.
- Substituir d_{16} por $d_{14} + d_{46} = 16 + 4 = 20$ e fazer $s_{16} = 4$.
- Substituir d_{25} por $d_{24} + d_{45} = 15 + 4 = 19$ e fazer $s_{25} = 4$.
- Substituir d_{26} por $d_{24} + d_{46} = 15 + 4 = 19$ e fazer $s_{26} = 4$.
- Substituir d_{35} por $d_{34} + d_{45} = 6 + 4 = 10$ e fazer $s_{35} = 4$.
- Substituir d_{36} por $d_{34} + d_{46} = 6 + 4 = 10$ e fazer $s_{36} = 4$.
- Substituir d_{65} por $d_{64} + d_{45} = 4 + 4 = 8$ e fazer $s_{65} = 4$.

Como $(k = 4) \neq (n = 6)$ avançar para o 2º passo.

Passo 2 Fazer $k = 5$. Não são possíveis mais melhoramentos.

Como $(k = 5) \neq (n = 6)$ avançar para o 2º passo.

Passo 2 Fazer $k = 6$. O elemento d_{54} é o que pode ser melhorado pela operação tripla. Criar D_6 e S_6 a partir de D_5 e S_5. Substituir d_{54} por $d_{56} + d_{64} = 16 + 4 = 20$ e fazer $s_{54} = 6$.

Como $(k = 6) = (n = 6)$ terminar o algoritmo. As matrizes finais D_6 e S_6 contêm toda a informação para determinar o caminho mais curto entre

D_4	1	2	3	4	5	6
1	-	3	10	16	**20**	**20**
2	∞	-	9	15	**19**	**19**
3	∞	∞	-	6	**10**	**10**
4	∞	∞	∞	-	4	4
5	∞	∞	∞	∞	-	9
6	∞	∞	∞	4	**8**	-

S_4	1	2	3	4	5	6
1	-	2	3	3	**4**	**4**
2	1	-	3	3	**4**	**4**
3	1	2	-	4	**4**	**4**
4	1	2	3	-	5	6
5	1	2	3	4	-	6
6	1	2	3	4	**4**	-

Tabela 3.9 Representação rede $k = 4$ (distâncias e sequências)

D_5	1	2	3	4	5	6
1	-	3	10	∞	∞	∞
2	∞	-	9	∞	∞	∞
3	∞	∞	-	6	15	∞
4	∞	∞	∞	-	4	4
5	∞	∞	∞	∞	-	9
6	∞	∞	∞	4	∞	-

S_5	1	2	3	4	5	6
1	-	2	3	4	5	6
2	1	-	3	4	5	6
3	1	2	-	4	5	6
4	1	2	3	-	5	6
5	1	2	3	4	-	6
6	1	2	3	4	5	-

Tabela 3.10 Representação rede $k = 5$ (distâncias e sequências)

D_6	1	2	3	4	5	6
1	-	3	10	16	20	20
2	∞	-	9	15	19	19
3	∞	∞	-	6	10	10
4	∞	∞	∞	-	4	4
5	∞	∞	∞	**12**	-	8
6	∞	∞	∞	4	8	-

S_6	1	2	3	4	5	6
1	-	2	3	3	4	4
2	1	-	3	3	4	4
3	1	2	-	4	4	4
4	1	2	3	-	5	6
5	1	2	3	**6**	-	6
6	1	2	3	4	4	-

Tabela 3.11 Representação rede $k = 6$ (distâncias e sequências)

dois quaisquer nós na rede. Por exemplo a distância mais curta entre o nó 1 e o nó 5 é $d_{15} = 20$ km pelo caminho $1 \to 3 \to 4 \to 5$.

3.3.3 Algoritmo de Bellman-Ford

3.3.3.1 Descrição

O algoritmo de Bellman-Ford permite determinar o caminho mais curto de um único nó para todos os outros nós de uma rede. É menos eficiente que o algoritmo de Dijkstra mas permite lidar com alguns arcos com custos negativos. No entanto, se existirem ciclos com custos negativos numa rede, então não existe solução possível, pois o algoritmo continuará a percorrer o ciclo indefinidamente reduzindo o custo. Esta propriedade é muitas vezes utilizada para permitir detectar ciclos em grafos.

Inicialização Considerando uma rede $R(V, A, C)$ composto por nós V e arcos A, definen-se dois vectores (*distancia* e *anterior*) com tamanho de V, sendo inicializados, respectivamente, a ∞ e $\emptyset$. A distância do nó *Origem* é colocada a zero.

Passo 1 Em todos os arcos (u, v) existentes em E, verificar se a distância v é maior que a distância u somada à distância (custo) do ramo entre (u, v), e caso positivo, fazer:

- $distancia[v] = distancia[u] + custo[u, v]$
- $anterior[v] = u$

Passo 2 Em todos os arcos (u, v) existentes em E, verificar se a distância v é maior que a distância u somada à distância (custo) do ramo entre (u, v), e caso positivo, para o algoritmo indicando que existem ciclos de custo negativo.

3.3.3.2 Pseudo-código

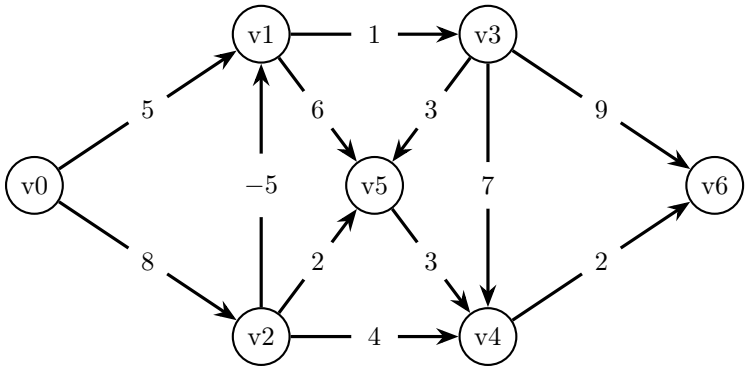
O pseudo-código do algoritmo de Bellman-Ford é apresentado em Algoritmo 5.

Algoritmo 5: Algoritmo de Bellman-Ford

Entrada: $Rede(V, A, C), Origem$
Saída: Distâncias mínimas entre nós v
início
  para *cada nó* v *em* V **faça**
    distancia[v] $= \infty$ ▷ Inicializa distâncias para o nó v como ∞
    anterior[v] $= \emptyset$ ▷ Inicializa anterior a v como nulo
  fim
  distancia[$Origem$] $= 0$ ▷ Inicializa distâncias do nó $origem$ como 0
  para *cada nó* v *em* V **faça**
    para *cada arco* u, v *em* A **faça**
      w=$distancia(u, v)$ ▷ Atribui a w distância entre (u, v)
      se $distancia[u] + w < distancia[v]$ **então**
        distancia[v] = distancia[u] $+ w$ anterior[v] $= u$
      fim
    fim
  fim
  para *cada arco* u, v *em* A **faça**
    w=$distancia(u, v)$ ▷ Atribui a w distância entre (u, v)
    se $distancia[u] + w < distancia[v]$ **então**
      Grafo com ciclos negativos
    fim
  fim
  Retorna distancia[], anterior[]
fim

3.3.3.3 Ilustração

Considerando a rede com arcos negativos apresentada de seguida, o algoritmo de Bellman-Ford pode ser resolvido de acordo com as iterações apresentadas, e ilustrado nas tabelas respectivas.



Iteração 1 Colocar todos os rótulos a uma distância infinita, e o rótulo do nó de origem a $[0, -]$. Selecciona-se o nó actual como sendo o nó $v0$.

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	*
v1	**[∞,-]**	-
v2	**[∞,-]**	-
v3	**[∞,-]**	-
v4	**[∞,-]**	-
v5	**[∞,-]**	-
v6	**[∞,-]**	-

Tabela 3.12 Iteração 1

Iteração 2 A partir do nó $v0$ podem ser acedidos os nós 1 e 2, com custo de 5 e 8, respectivamente. O conjunto de nós actualizados passa a incluir $v1, v2$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	-
v1	**[5,v0]**	*
v2	**[8,v0]**	*
v3	**[∞,-]**	-
v4	**[∞,-]**	-
v5	**[∞,-]**	-
v6	**[∞,-]**	-

Tabela 3.13 Iteração 2

O próximo nó seleccionado é o nó $v1$, sendo o primeiro nó disponível da lista de nós recentemente actualizados, sendo também removido dela.

Iteração 3 A partir do nó $v1$ podem ser acedidos os nós $v3$ e $v5$, com custo de $5+1$ e $5+6$, respectivamente. O conjunto de nós actualizados $v2$ passa a incluir $v3, v5$ ficando com $v2, v3, v5$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[5,v0]**	**-**
v2	**[8,v0]**	*****
v3	**[5 + 1,v1]**	*****
v4	**[∞,-]**	**-**
v5	**[5 + 6,v1]**	*****
v6	**[∞,-]**	**-**

Tabela 3.14 Iteração 3

O próximo nó seleccionado é o nó $v2$, sendo o primeiro nó disponível da lista de nós recentemente actualizados, sendo também removido dela.

Iteração 4 A partir do nó $v2$ podem ser acedidos os nós $v1, v4$ e $v5$, com custo de $8-5, 8+4$ e $8+2$, respectivamente. O conjunto de nós actualizados $v3, v5$ passa a incluir $v1, v4$ ficando com $v3, v5, v1, v4$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[8 − 5,v2]**	*****
v2	**[8,v0]**	**-**
v3	**[6,v1]**	*****
v4	**[8 + 4,v2]**	*****
v5	**[8 + 2,v2]**	*****
v6	**[∞,-]**	**-**

Tabela 3.15 Iteração 4

O próximo nó seleccionado é o nó $v3$.

Iteração 5 A partir do nó $v3$ podem ser acedidos os nós $v5$ e $v6$, com custo de $6+3$ e $6+9$, respectivamente. O conjunto de nós actualizados $v5, v1, v4$ passa a incluir $v6$ ficando com $v5, v1, v4, v6$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[3,v2]**	*****
v2	**[8,v0]**	**-**
v3	**[6,v1]**	**-**
v4	**[12,v2]**	*****
v5	**[6 + 3,v3]**	*****
v6	**[6 + 9,v3]**	*****

Tabela 3.16 Iteração 5

O próximo nó seleccionado é o nó $v5$.

Iteração 6 A partir do nó $v5$ não podem ser acedidos os nós com custo inferior ao existente. O conjunto de nós actualizados fica com $v1, v4, v6$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[3,v2]**	*****
v2	**[8,v0]**	**-**
v3	**[6,v1]**	**-**
v4	**[12,v2]**	*****
v5	**[9,v3]**	**-**
v6	**[15,v3]**	*****

Tabela 3.17 Iteração 6

O próximo nó seleccionado é o nó $v1$.

Iteração 7 A partir do nó $v1$ pode ser acedido o nó $v3$, com custo de $3 + 1$. O conjunto de nós actualizados $v4, v6$ passa a incluir $v3$ ficando com $v4, v6, v3$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[3,v2]**	**-**
v2	**[8,v0]**	**-**
v3	**[3 + 1,v1]**	*****
v4	**[12,v2]**	*****
v5	**[9,v3]**	**-**
v6	**[15,v3]**	*****

Tabela 3.18 Iteração 7

O próximo nó seleccionado é o nó $v4$.

Iteração 8 A partir do nó $v4$ pode ser acedido o nó $v6$, com custo de $12 + 2$. O conjunto de nós actualizados $v6, v3$ continua com $v6, v3$. A lista dos nós rotulados fica:

O próximo nó seleccionado é o nó $v6$.

Iteração 9 A partir do nó $v6$ não pode ser acedido nenhum nó com custo inferior ao actual. O conjunto de nós actualizados $v3$ não é alterado. A lista dos nós rotulados fica:

O próximo nó seleccionado é o nó $v3$.

Iteração 10 A partir do nó $v3$ podem ser acedidos os nós $v4, v5$ e $v6$, com custo de $4 + 7, 4 + 3$ e $4 + 9$, respectivamente. O conjunto de nós actualizados (vazio) fica com $v4, v5$ e $v6$. A lista dos nós rotulados fica:

O próximo nó seleccionado é o nó $v4$.

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[**3,**v2]**	**-**
v2	**[**8,**v0]**	**-**
v3	**[**4,**v1]**	*****
v4	**[**12,**v2]**	**-**
v5	**[**9,**v3]**	**-**
v6	**[**12 + 2,**v4]**	*****

Tabela 3.19 Iteração 8

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[**3,**v2]**	**-**
v2	**[**8,**v0]**	**-**
v3	**[**4,**v1]**	*****
v4	**[**12,**v2]**	**-**
v5	**[**9,**v3]**	**-**
v6	**[**14,**v4]**	**-**

Tabela 3.20 Iteração 9

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[**3,**v2]**	**-**
v2	**[**8,**v0]**	**-**
v3	**[**4,**v1]**	**-**
v4	**[**4 + 7,**v3]**	*****
v5	**[**4 + 3,**v3]**	*****
v6	**[**4 + 9,**v3]**	*****

Tabela 3.21 Iteração 10

Iteração 11 A partir do nó $v4$ não podem ser acedidos nós com custo menor ao actual. O conjunto de nós actualizados actualiza-se para $v5$ e $v6$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[**3,**v2]**	**-**
v2	**[**8,**v0]**	**-**
v3	**[**4,**v1]**	**-**
v4	**[**11,**v3]**	**-**
v5	**[**7,**v3]**	*****
v6	**[**13,**v3]**	*****

Tabela 3.22 Iteração 11

O próximo nó seleccionado é o nó $v5$.

Iteração 12 A partir do nó $v5$ pode ser acedido o nó $v4$ com custo de $7+3$. O conjunto de nós actualizados fica com $v6, v4$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[3,v2]**	**-**
v2	**[8,v0]**	**-**
v3	**[4,v1]**	**-**
v4	**[7 + 3,v5]**	*****
v5	**[7,v3]**	**-**
v6	**[13,v3]**	*****

Tabela 3.23 Iteração 12

O próximo nó seleccionado é o nó $v6$.

Iteração 13 A partir do nó $v6$ não podem ser acedidos nós de custo inferior ao actual. O conjunto de nós actualizados fica com $v4$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[3,v2]**	**-**
v2	**[8,v0]**	**-**
v3	**[4,v1]**	**-**
v4	**[10,v5]**	*****
v5	**[7,v3]**	**-**
v6	**[13,v3]**	**-**

Tabela 3.24 Iteração 13

O próximo nó seleccionado é o nó $v4$.

Iteração 14 A partir do nó $v4$ pode ser acedido o nó $v6$ com custo de $10+2$. O conjunto de nós actualizados fica com $v6$. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[3,v2]**	**-**
v2	**[8,v0]**	**-**
v3	**[4,v1]**	**-**
v4	**[10,v5]**	**-**
v5	**[7,v3]**	**-**
v6	**[10 + 2,v4]**	*****

Tabela 3.25 Iteração 14

O próximo nó seleccionado é o nó $v6$.

Iteração 15 A partir do nó $v6$ não podem ser acedidos nós de custo inferior ao actual. O conjunto de nós actualizados fica vazio. A lista dos nós rotulados fica:

Nó	Distancia/Anterior	Actualizado
v0	**[0,-]**	**-**
v1	**[3,v2]**	**-**
v2	**[8,v0]**	**-**
v3	**[4,v1]**	**-**
v4	**[10,v5]**	**-**
v5	**[7,v3]**	**-**
v6	**[12,v4]**	**-**

Tabela 3.26 Iteração 15

O algoritmo termina dado não haver mais nós para verificar que tenham sido recentemente actualizados. O caminho a partir de qualquer nó pode ser traçado verificando o custo, e o seu nó anterior. Como exemplo, o nó $v6$ tem uma distância da origem de 12 sendo o nó anterior o $v4$. O nó $v4$ tem uma distância à origem de 10 com o nó anterior sendo $v5$. Fazendo esta operação consecutivamente, encontra-se o caminho de qualquer nó à origem. No caso de $v6$, o caminho reverso é $v6, v4, v5, v3, v1, v2, v0$ com custo 12.

3.3.3.4 Resolução Tabular

A resolução tabular segue os mesmos principios da resolução iterativa com rótulos, apenas agregando a informação de cada iteração a cada linha. O conjunto S identifica os últimos vértices que sofreram actualizações, e que necessitam ser verificados nas proximas iterações para actualizar a distância a outros vértices conectados. Ao serem verificados, são removedos do conjunto S. A distância à origem de um vértice escolhido (através da coluna respectiva) pode ser traçada considerando o valor mínimo da coluna e seleccionando como vértice anterior o que lhe deu origem nessa linha (coluna Anterior). Como exemplo, o caminho reverso do vértice $v6$ é $v6, v4, v5, v3, v1, v2, v0$ com custo 12. Na coluna do vértice $v6$ o valor mínimo é 12 que foi obtido através do vértice anterior $v4$. Ao verificar o vértice $v4$ usando o mesmo procedimento verifica-se que o valor mínimo é 10 e foi obtido através do vértice anterior $v5$. Ao realizar este procedimento iterativamente encontra-se o caminho até à origem.

Anterior	L(v0)	L(v1)	L(v2)	L(v3)	L(v4)	L(v5)	L(v6)	(S)
-	0	∞	∞	∞	∞	∞	∞	{}
$v0$	0	5	8	–	–	–	–	$\{v1, v2\}$
v1	–	–	–	6	–	11	–	$\{v2, v3, v5\}$
$v2$	–	3	–	–	12	10	–	$\{v3, v5, v1, v4\}$
v3	–	–	–	–	*	9	15	$\{v5, v1, v4, v6\}$
v5	–	–	–	–	*	–	–	$\{v1, v4, v6\}$
$v1$	–	–	–	4	–	*	–	$\{v4, v6, v3\}$
v4	–	–	–	–	–	–	14	$\{v6, v3\}$
v6	–	–	–	–	–	–	–	$\{v3\}$
$v3$	–	–	–	–	11	7	13	$\{v4, v5, v6\}$
v4	–	–	–	–	–	–	*	$\{v5, v6\}$
$v5$	–	–	–	–	10	–	–	$\{v6, v4\}$
v6	–	–	–	–	–	–	–	$\{v4\}$
$v4$	–	–	–	–	–	–	12	$\{v6\}$
$v6$	–	–	–	–	–	–	–	{}

Tabela 3.27 Tabela com iterações de Bellman-Ford.

Capítulo 4
Problema de Fluxo Máximo

4.1 Descrição

O Problema de Fluxo Máximo consiste em encontrar o fluxo que maximiza a transferência entre os nós de origem (e) e de destino (s) numa rede $R = (V, A, C)$, considerando que existem capacidades C associadas a cada arco.

4.2 Formulação

Variáveis de Decisão

$$x_{i,j} : \text{Capacidade de fluxo no arco } (i,j), \text{ de } i \text{ para } j \tag{4.1}$$

$$c_{i,j} : \text{Capacidade do arco } (i,j) \tag{4.2}$$

Função Objectivo

$$max \sum_{k \in V} x_{e,k} = \sum_{k \in V} x_{k,s} \tag{4.3}$$

Restrições

$$\sum_{j \in V} x_{k,j} = \sum_{i \in V} x_{i,k}, \forall k \in V \tag{4.4}$$

$$x_{i,j} \leq c_{i,j}, \forall (i,j) \in A \tag{4.5}$$

$$x_{i,j} \geq 0 \text{ e inteira}, \forall (i,j) \tag{4.6}$$

A variável de decisão especificada em (4.1) é inteira, indicando o valor de fluxo utilizado no arco (i,j). A função objectivo, apresentada em (4.3), maximiza a quantidade de fluxo a partir da entrada ou da saída da rede. As restrições (4.4) asseguram que o fluxo total de saída de um nó é idêntico

ao fluxo total de entrada no mesmo nó. As restrições (4.5) limitam o fluxo em cada arco ao valor máximo especificado na capacidade $c_{i,j}$ e a última restrição, em (4.6), garante o fluxo em cada arco $x_{i,j}$ assume apenas valores não-negativos e inteiros.

4.3 Algoritmo de Ford-Fulkerson

4.3.0.1 Descrição

O algoritmo de Ford-Fulkerson (Ford e Fulkerson, 1956) apresenta uma solução com fluxos válidos, que termina quando o sistema atingir o seu fluxo máximo. Os pontos principais são do algoritmo estão descritos de seguida:

Passo 1 Inicializar o nó de entrada com um fluxo nulo, i.e. 0
Passo 2 Capacidades iniciais dos ramos = menor capacidade disponível no caminho
Passo 3 Determinar caminho não saturado (em que capacidade disponível é diferente de zero) entre nó de entrada e nó de saída:

- Se não existir: Solução óptima
- Caso exista: Selecciona caminho

Passo 4 Somar ao fluxo de entrada um fluxo igual à capacidade do fluxo seleccionado
Passo 5 Alterar capacidades dos arcos do caminho seleccionado diminuindo a capacidade disponível
Passo 6 Adicionar arcos ao grafo residual com sentido oposto e capacidade igual à capacidade adicionada na iteração actual[1]
Passo 7 Retornar ao passo 3

Verificação de Optimalidade

A verificação de fluxo máximo de um sistema é realizado através de um método de cortes mínimos que consiste em separar a rede em duas partes (A e B) em que uma contém o ponto de entrada (e) e outra contém o ponto de saída (s). O sistema está em fluxo máximo quando a entrada é separada da saída por um único corte que atravessa ramos saturados para a saída, e ramos de fluxo nulo para a entrada (também conhecido por corte mínimo). A capacidade de um corte é obtida somando as capacidades dos arcos do corte dirigidos da entrada para a saída.

Fluxos Negativos

Quando o sistema não têm um corte mínimo (ou fluxo máximo) por existir um caminho não saturado, a sua identificação é feita considerando a existência de fluxos contabilizados como negativos, que são fluxos que atravessam

[1] Passo apenas considerado quando a resolução do algoritmo utiliza grafos residuais

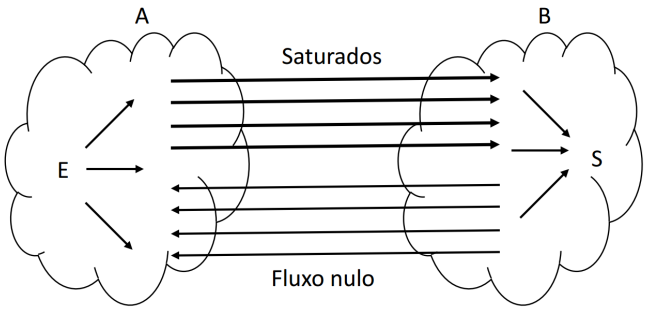


Figura 4.1 Exemplo de Corte Mínimo

os ramos no sentido contrário à sua orientação. A cada iteração, um fluxo considerado negativo consiste numa redução do fluxo máximo que passa em sentido oposto.

4.3.0.2 Pseudo-Código

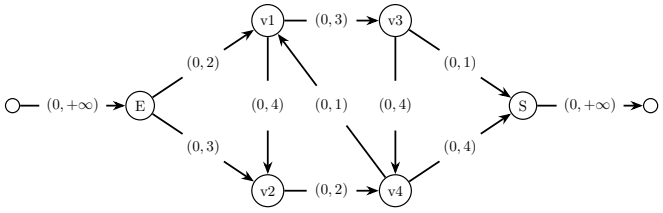
O pseudo-código do algoritmo de Ford-Fulkerson pode ser visto no Algoritmo 6.

Algoritmo 6: Algoritmo de Ford-Fulkerson

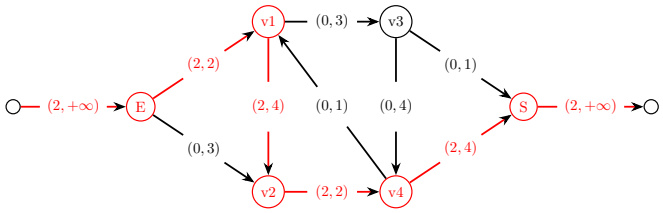
Entrada: $Rede, Entrada, Saida$
Saída: Fluxo máximo entre nó de entrada E e saída S
início
  1 : ▷ Marca 1
  $R(s)$= (-,∞) ▷ Rótulos são inicializados v como ∞
  X= {s}
  repita
    seleccionar $i \in X$
    para *cada vértice j não rotulado:* $x_{ij} < l_{ij}$ **faça**
      R$[j] = +i, \delta_j$, sendo $\delta_j = \min\{\delta_i, l_{ij} - x_{ij}\}$
      $X = X \cup \{j\}$
    fim
    para *cada vértice j não rotulado:* $x_{ij} > 0$ **faça**
      R$[j] = -i, \delta_j$, sendo $\delta_j = \min\{\delta_i, x_{ij}\}$
      $X = X \cup \{j\}$
    fim
    $X = X - \{i\}$
  até $X = \emptyset$ *ou* $S = rotulado$
  se $S = rotulado$ **então**
    Aumentar fluxo saltar para 1:
  fim
  retorna distancia[], anterior[]
fim

4.3.0.3 Ilustração utilizando Grafos Residuais

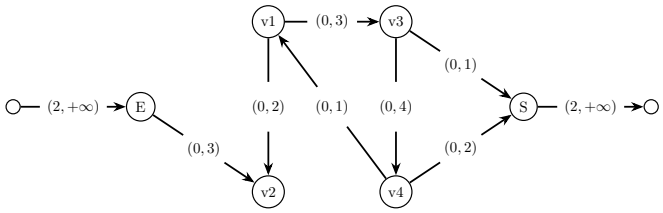
Considerando uma rede com o tipo de estrutura ilustrado de seguida, procede-se à sua resolução (iterativa) de forma a obter a capacidade de cada arco que maximiza o fluxo do nó de entrada E para o nó de saída S.



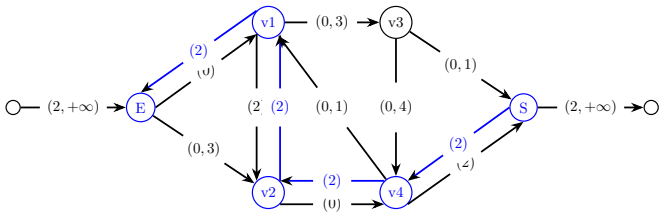
Iteração 1 Encontra-se uma rota viável, determinando capacidade (igual a mínimo de $2, 4, 2, 4$) e incrementando o fluxo actual total.



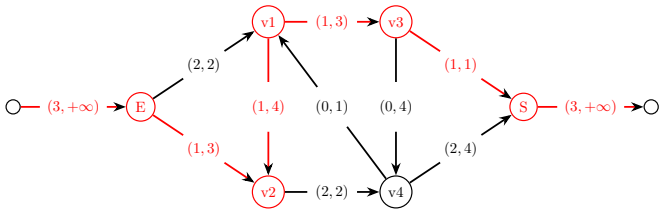
Subtraindo a rota escolhida e eliminando os ramos esgotados obtém-se o seguinte sistema.



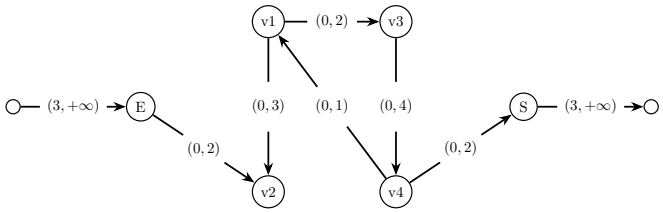
Dado que não existe rota viável, o algoritmo termina. No entanto, não é possível encontrar um corte mínimo, e verifica-se também a existência de vários ramos com capacidade não utilizada. A capacidade residual (não utilizada) pode ser acedida utilizando fluxos negativos (através da capacidade já alocada). O sistema considerando os fluxos já alocados (em sentido negativo) é o seguinte:



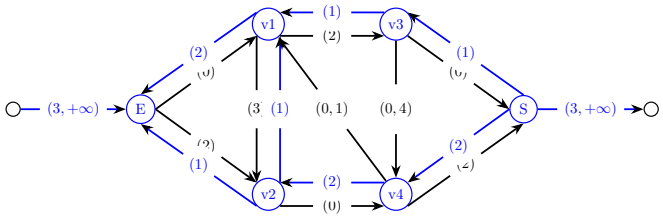
Iteração 2 Escolhendo outra rota considerando os fluxos negativos e considerando capacidade (igual a mínimo de $3, (4-2), 3, 1$):



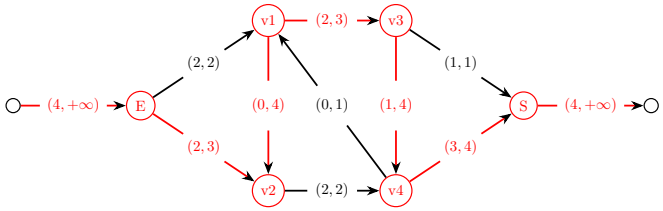
Subtraindo as rotas escolhidas e eliminando os ramos esgotados obtém-se o sistema apresentado de seguida:



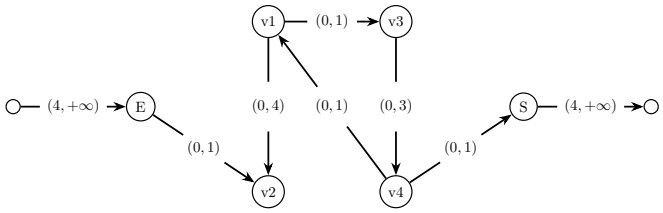
Pode-se verificar que ainda não é possivel obter um corte mínimo. A capacidade residual pode ser acedida para gerar caminhos alternativos. Os fluxos negativos disponiveis estão identificados no sistema seguinte:



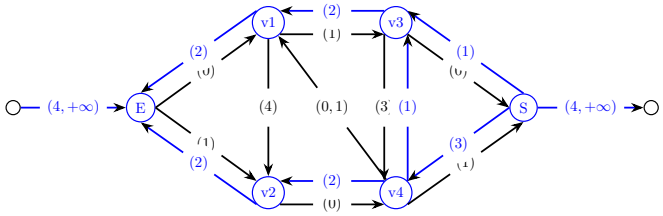
Iteração 3 Escolhendo a proxima rota viavel, e considerando capacidade (igual a mínimo de $(3-1), (4-1), (3-1), 4, (4-2)$):



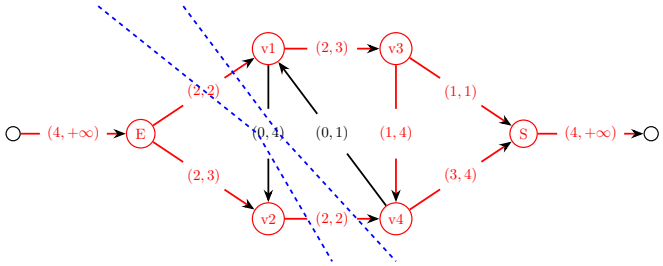
Fazendo a subtracção das rotas escolhidas ao sistema original consegue-se verificar que não existem rotas que possam ser escolhidas directamente.



Os fluxos negativos também não permitem escolher mais nenhuma rota possivel através da inversão dos fluxos já seleccionados.



Iteração 4 O algoritmo termina com os seguintes fluxos no sistema. Pode-se notar também o corte mínimo identificado pelas linhas que separam o sistema em duas regiões distintas, e na separação verificam-se fluxos saturados em direcção à saída, e fluxos nulos em direcção à entrada.



4.3.0.4 Descrição utilizando Rótulos

O algoritmo de Ford-Fulkerson envolve os seguintes passos:

Passo 1 Inicializar com fluxo nulo no nó de entrada

Passo 2 Capacidades iniciais dos ramos = menor capacidade disponivel no caminho

Passo 3 Determinar caminho não saturado mais "acima"(em que capacidade disponivel diferente de zero) entre nó de entrada e nó de saída:

- Se não existir: Solução óptima
- Caso exista: Selecciona caminho

Passo 4 Somar ao fluxo de entrada um fluxo igual à capacidade do fluxo seleccionado

Passo 5 Alterar capacidades dos ramos do caminho seleccionado, diminuindo a capacidade disponivel

Passo 6 Voltar ao Passo 3

4.3.0.5 Pseudo-Código utilizando Rótulos

O pseudo-código do algoritmo de Ford-Fulkerson é apresentado em Algoritmo 7.

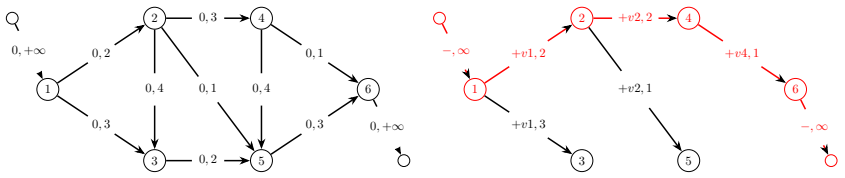
Algoritmo 7: Algoritmo de Ford-Fulkerson

Entrada: $Rede, Entrada, Saida$
Saída: Fluxo máximo entre nó entrada E e saída S
início
  1: ▷ Marca 1
  $R(s)$= (-,∞) ▷ Rótulos são inicializados v como ∞
  X= {s}
  repita
    seleccionar $i \in X$
    Examinar (i)
    $X = X - \{i\}$
  até $X = \emptyset$ *ou* $S = rotulado$
  se $S = rotulado$ **então**
    Aumentar fluxo
    saltar para 1:
  fim
  retorna distancia[], anterior[]
fim

4.3.0.6 Ilustração utilizando Rótulos

O algoritmo de Ford-Fulkerson pode ser resolvido alternativamente utilizando um método de rotulagem, tal como descrito nas proximas iterações.

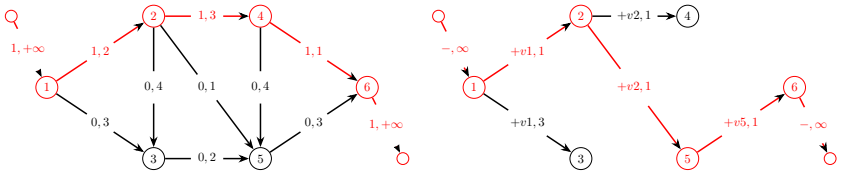
Iteração 1 Inicialmente escolhe-se um caminho do nó entrada $v1$ até ao nó de saída $v6$, através de ramos com capacidade disponível. Os rótulos para cada vértice ficam definidos a partir do vértice anterior i(indicando sentido positivo/negativo dependendo da direcção do ramo), e mínimo de (capacidade do vértice anterior, capacidade do ramo anterior) sendo capacidade definida como j, com o formato $\pm v_i, j$. Deve-se assinalar os caminhos necessários até conectar todos os vértices, além do caminho principal definido. Os ramos com capacidade esgotada não são considerados para efeitos de ligação a nós ainda não conectados.



Valor de fluxo actual de $v1$ para $v6$ calculado com valor de 1.

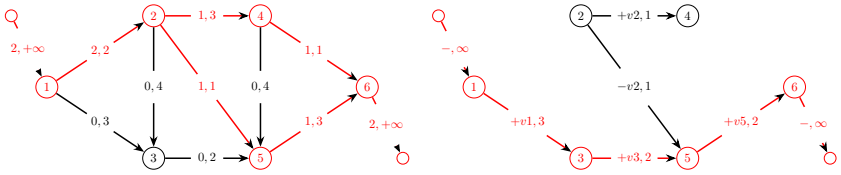
Iteração 2 Estando alguns ramos com capacidade residual definida, continua-se a definir caminhos desde a entrada $v1$ até à saída $v6$ pelo caminho

mais "acima"conectando os nós ao longo do caminho com capacidades disponíveis. A escolha de o caminho através dos vértices mais "acima"é feita para reduzir e corrigir erros que originem da resolução manual do problema.



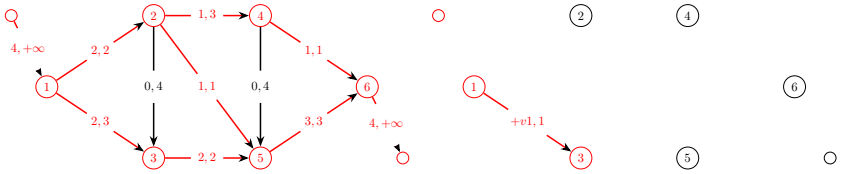
Valor de fluxo actual de $v1$ para $v6$ calculado com valor de 2.

Iteração 3 Nesta iteração, o caminho mais "acima"permite fazer ligação ao nó $v5$ através do nó $v5$ invertendo o fluxo que percorre esse ramo, e depois ligando ao nó $v4$. O caminho escolhido tem a sua capacidade (mínimo disponível nos ramos) subtraída ao grafo com as capacidades.



Valor de fluxo actual de $v1$ para $v6$ calculado com valor de 4.

Iteração 4 Nesta iteração só é possivel ligar a um nó $v3$ devido a capacidades esgotadas em todos os ramos (incluindo ramos que possam ter o seu fluxo invertido). O algoritmo termina com um fluxo máximo de 4.



Valor de fluxo total é calculado com valor de 4. Através do método de cortes mínimos consegue-se confirmar que este é o fluxo máximo.

4.4 Aplicações

O objectivo consiste em determinar uma alocação eficiente de recursos para realizar um determinado conjunto de tarefas (incluindo entregas). O problema

pode ser descrito através de uma rede (ou um grafo) contendo o conjunto mínimo de caminhos e resolvido através de um problema de minimização de fluxo. Os problemas de escalonamento podem ser resolvidos através da aplicação de algoritmo de fluxo máximo.

4.4.1 Problemas de Escalonamento

Descrição

Considerações:

Assumindo $i = 1, 2, ..., n$ como um conjunto de tarefas:

- T_i: Tempo de início de tarefa
- D_i: Duração de tarefa
- D_{ij}: Duração da deslocação entre i e j

A tarefa i pode proceder a tarefa j caso $T_j \geq T_i + D_i + D_{ij}$.

Inicialização:

- Duplicar cada nó
- Definir cada ramo como tendo uma capacidade unitária
- Introduzir um nó de origem e introduzir outro como nó de saída (não pertencente a nenhum dos nós originais)

A resolução é realizada através de um algoritmo de fluxo máximo.

Ilustração

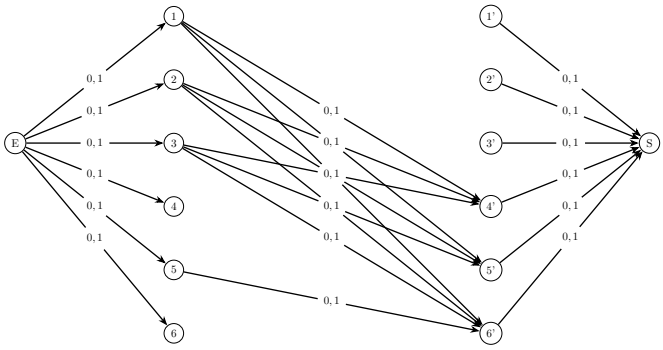
Considerando a Tabela 4.1 que contém um problema de escalonamento onde o objectivo é determinar o número mínimo de veículos necessários para servir actividades $(1, 2, 3, ..., 6)$ em vários locais $(A, B, C, ..., F)$, limitados pelo horário respectivo.

Serviço	Local	Horário
1	A	09h00-11h10
2	B	09h30-11h10
3	C	11h00-13h40
4	D	14h30-19h40
5	E	15h00-16h40
6	F	17h00-19h40

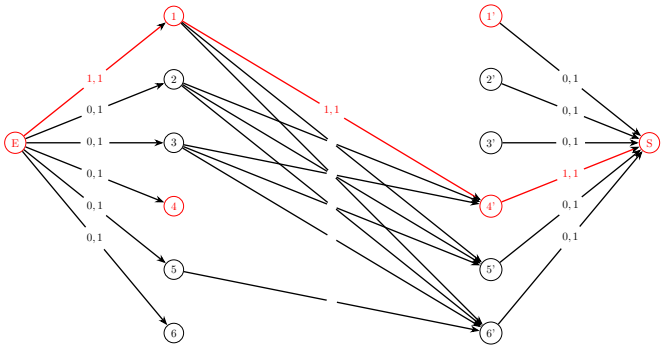
Tabela 4.1 Escalonamento de viaturas

Este problema pode ser resolvido através de uma representação no formato de um grafo onde é aplicado um algoritmo de maximização de fluxo.

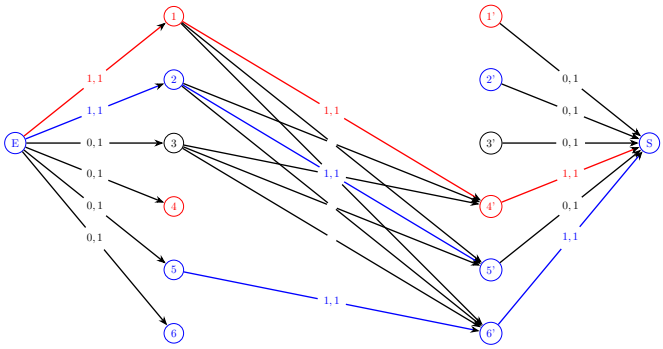
Inicialização O grafo inicial é construído duplicando os nós que representam os serviços, e fazendo a interligação entre eles indicando as suas possíveis sequências (limitados pelos horários disponíveis). O número mínimo de veículos será identificado pelo número de circuitos independentes percorridos desde o nó E até ao nó S.



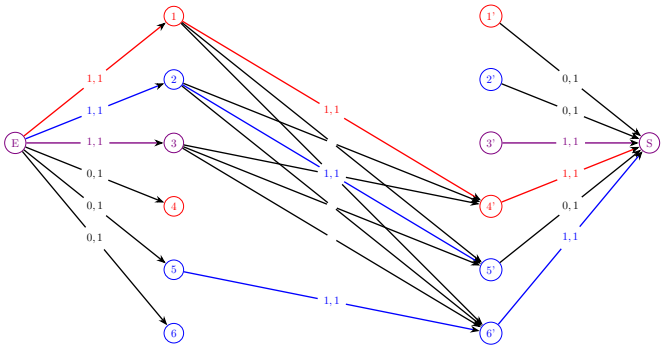
Iteração 1 Na primeira iteração, a primeira rota segue do nó E para o nó 1 continuando para o nó 4 e dirigindo-se para a saída dado que o nó 4 não liga a mais nenhum nó (excepto S). O fluxo actual entre o nó de entrada E e o nó de saída S é 1.



Iteração 2 Na segunda iteração, a rota segue desde o nó E para o nó 2 sendo ligado ao nó 5 e visto ainda ter ligação ao nó 6 passa por ele saíndo depois para o nó de saída S. O fluxo actualizado entre E e S torna-se 2.



Iteração 3 Na terceira iteração, o único caminho disponível é o que passa pelo nó 3, não havendo mais nenhum serviço que necessite ser realizado. Como tal, o fluxo total fica com uma capacidade de 3 indicando que são necessários 3 veículos para realizar as tarefas especificadas.



São necessários 3 veículos para cumprir todos os requisitos, implicando que as rotas para cada veículo são respectivamente:

$V1$ realiza $E-1-4-S$
$V2$ realiza $E-2-5-6-S$
$V3$ realiza $E-3-S$

4.4.2 Planeamento com Janelas Temporais

Descrição

Considerações:

Assumindo $i = 1, 2, ..., n$ como o índice de um período temporal, e $j = 1, 2, ..., m$ como índice da tarefa:

- S_i: Instance temporal i
- T_J: Tarefa j
- D_{ij}: Afetação do instante temporal i a tarefa j

Inicialização:

- Separar os conjuntos referentes a Instantes Temporais (S_i) e Serviços/Recursos (T_j)
- Introduzir um nó de entrada E e outro saída S (não pertencente a nenhum dos nós originais)
- Definir cada ramo que liga a tarefa a um instante temporal com capacidade unitária
- Definir as capacidades de recursos em cada ramo do nó E ao respectivo nó instante temporal S_i
- Definir as necessidades de serviço em cada ramo do respectivo nó de serviço T_j ao nó S

A resolução é realizada através de um algoritmo de fluxo máximo.

Ilustração

Considerando a Tabela 4.2 que contém um problema de planeamento com janelas temporais onde o objectivo é determinar a afectação entre um número de serviços/trabalhos $(1, 2, 3, 4, 5)$ a um número mínimo de recursos, incluindo tempo $(Semana1, Semana2, ...Semana5)$.

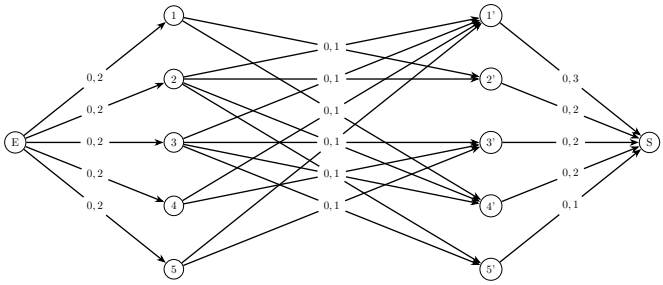
Trabalho	S1	S2	S3	S4	S5	Duração
1	-	x	x	x	x	3
2	x	x	-	-	-	2
3	-	-	x	x	x	2
4	x	x	x	-	-	2
5	-	x	x	-	-	1

Tabela 4.2 Planeamento com Janelas Temporais

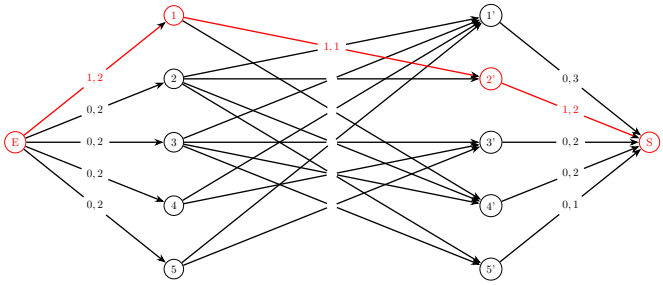
Este problema pode ser resolvido através de uma representação no formato de um grafo onde é aplicado um algoritmo de maximização de fluxo. Os serviços/trabalhos são válidos num horizonte temporal definido (5 semanas), tendo cada um os seus requisitos individuais. Os recursos estão limitados através de uma capacidade especificada (2 pessoas).

Inicialização O grafo inicial é construído colocando os periodos temporais ordenados no lado esquerdo $(S1, S2, S3, S4, S5)$, com a sua capacidade limitante $(2, 2, 2, 2, 2)$, e do lado direito colocando os serviços/trabalhos

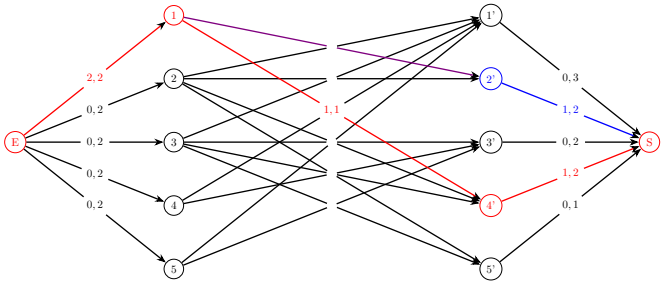
$(T1, T2, T3, T4, T5)$ que precisam de ser realizados, com os seus requisitos mínimos$(3, 2, 2, 2, 1)$. As ligações entre os periodos temporais e os serviços a realizar fazem-se através de ramos com um determinado valor de capacidade (neste caso 1) que indicam a quantidade de recursos que se podem utilizar simultaneamente para realizar uma determinada tarefa.



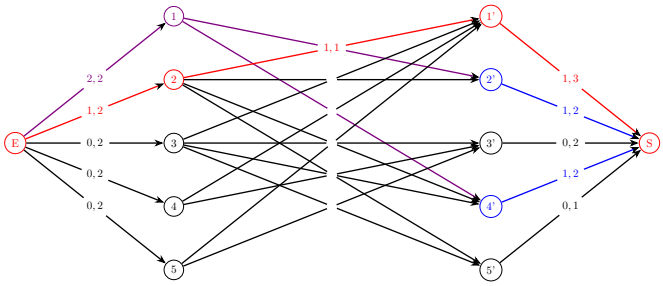
Iteração 1 Na primeira iteração, a primeira rota segue do nó E para o nó 1 continuando para o nó $2'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S é 1.



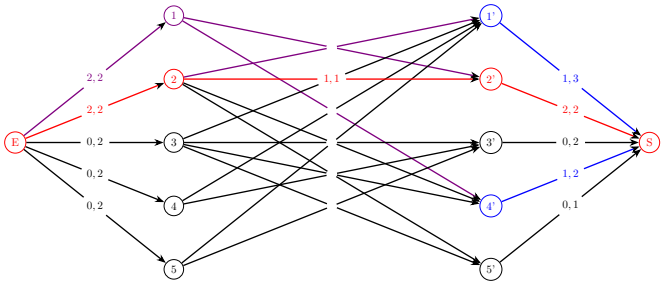
Iteração 2 Na segunda iteração, a rota segue do nó E para o nó 1 continuando para o nó $4'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 2, esgotando a os recursos disponíveis durante a primeira semana.



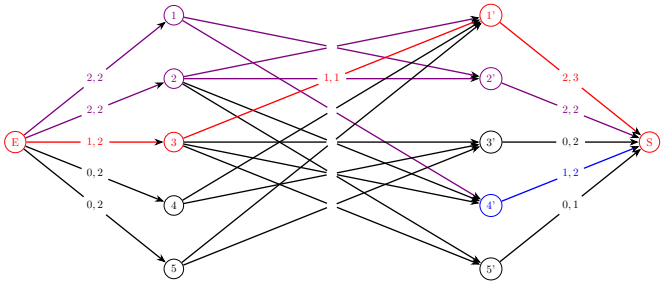
Iteração 3 Na terceira iteração, a rota segue do nó E para o nó 2 continuando para o nó $1'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 3.



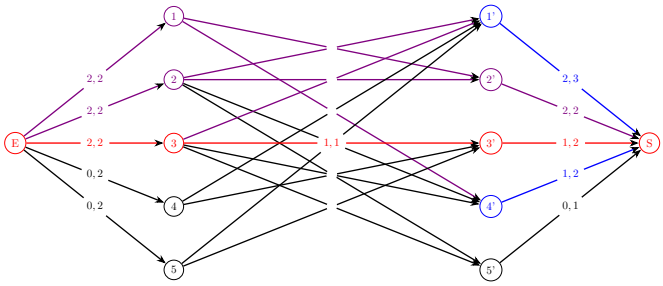
Iteração 4 Na quarta iteração, a rota segue do nó E para o nó 2 continuando para o nó $2'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 4, esgotando a os recursos disponíveis durante $S2$, e os requisitos de $T2$.



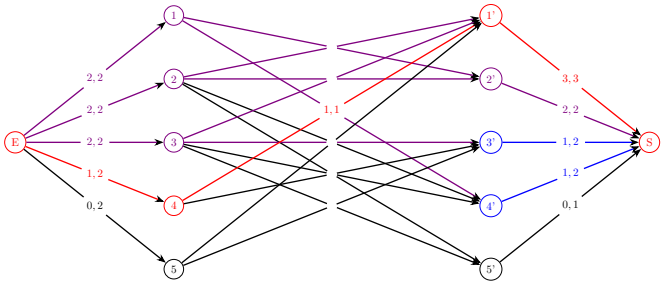
Iteração 5 Na quinta iteração, a rota segue do nó E para o nó 3 continuando para o nó $1'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 5.



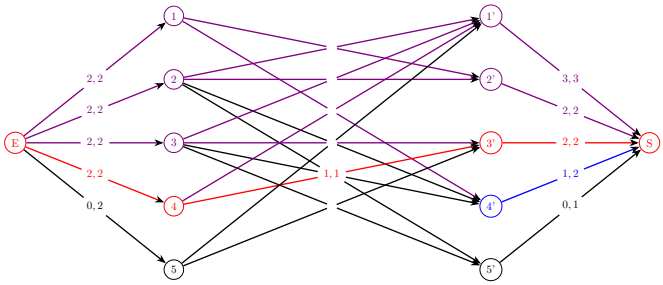
Iteração 6 Na sexta iteração, a rota segue do nó E para o nó 3 continuando para o nó $3'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 6, esgotando os recursos disponíveis para $S3$.



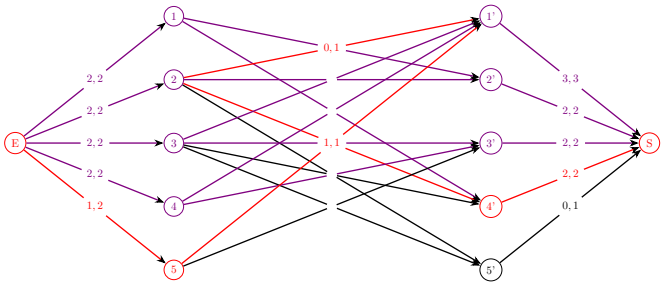
Iteração 7 Na sétima iteração, a rota segue do nó E para o nó 4 continuando para o nó $1'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 7, cumprindo os requisitos para realizar o $T1$.



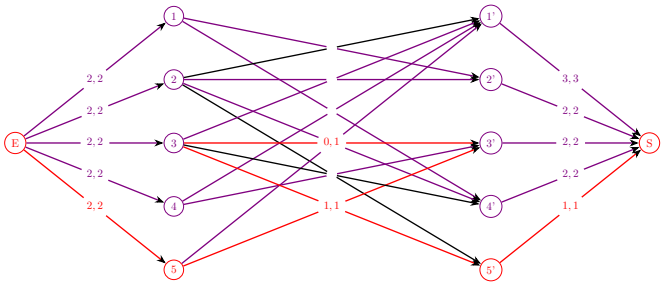
Iteração 8 Na oitava iteração, a rota segue do nó E para o nó 4 continuando para o nó $3'$ e dirigindo-se para a saída S. O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 8, cumprindo os requisitos para realizar o $T3$ e esgotando a capacidade do recurso de $S4$.



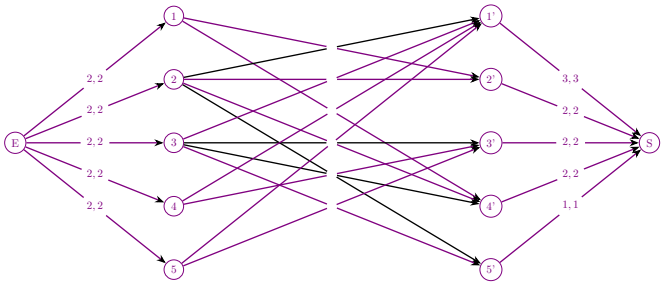
Iteração 9 Na nona iteração, a rota segue do nó E para o nó 5 continuando para o nó $1'$, retornando para o nó 2 e dirigindo-se para o nó $4'$ finalmente saindo pela saída S. O ramo entre 2 e $1'$ passa a ter capacidade zero (dado que o fluxo circula em sentido contrário, anulando o existente). O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 9, cumprindo os requisitos para realizar o $T4$.



Iteração 10 Na décima iteração, a rota segue do nó E para o nó 5 continuando para o nó $3'$, retornando para o nó 3 e dirigindo-se para o nó $5'$ finalmente saindo pela saída S. O ramo entre 3 e $3'$ passa a ter capacidade zero (dado que o fluxo circula em sentido contrário, anulando o existente). O fluxo actual entre o nó de entrada E e o nó de saída S passa a ser 10, cumprindo os requisitos para realizar o $T5$, e esgotando os recursos disponíveis em $S5$.



Resultado O resultado pode ser visto no grafo seguinte.



A afectação de trabalhos T à respectiva semana S é o seguinte:

$S1$ realiza $T2$ e $T4$
$S2$ realiza $T2$ e $T4$
$S3$ realiza $T1$ e $T5$
$S4$ realiza $T1$ e $T3$
$S5$ realiza $T1$ e $T3$

Capítulo 5
Localização de Instalações

A determinação eficiente da localização de uma ou várias instalações produz vantagens significativas em termos da capacidade para produzir valor e reduzir desperdícios. Podem classificar-se como instalações terrenos, edifícios, equipamentos, e outros objectos que precisem de ser posicionados de forma a maximizar o seu benefício.

O planeamento de instalações pode ocorrer em vários níveis:

Global Avaliação qualitativa avaliando o volume de recursos disponíveis numa determinada região

- Disponibilidade e transporte de matéria prima e semi-acabados.
- Energia e combustíveis.
- Água e saneamento (escoamento de resíduos)
- Transportes em geral.
- Serviços governamentais entre outros.

Supra Avaliação qualitativa considerando qualidade dos recursos disponiveis na região

- Edifícios considerando tamanho, número e localização relativa, assim como custos (armazenagem, etc).
- Mão-de-obra, ambiente sindical e social.
- Mercado local (procura).
- Incentivos e impostos municipais.
- Alternativas de transporte considerando qualidade das vias e possiveis saturações.
- Factores geográficos, clima e segurança.
- Sistemas de comunicação.

Macro Factores a nivel estratégico tal como estrutura de cada edifício e subunidades, tal como departamentos

- Criação de núcleos sociais.
- Meio rural, urbano, periferia ou grande centro.

- Integração com empresas locais a nivel de processo de fabrico e necessidades de mão-de-obra, matéria prima, energia, etc.
- Análise de previsão futura de procura e recursos disponíveis.

Micro Impacto na comunidade local, colaboradores, e estruturamento do processo de fabrico.

- Localização relativa a transportes como aeroportos, portos marítimos, rodovias e ferrovias.
- Acesso aos funcionários da empresa.
- Estruturação dos departametnos considerando mobiliário e postos de trabalho.
- Segurança local contra acidentes (inundações, incendios, etc).

Sub-Micro Estruturação dos postos de trabalho individuais, optimizando para ergonomia

- Fluxos de informação dentro da empresa.
- Localização de posto de trabalho (secretária, etc).

O foco principal no Problema de Localização de Instalações (PLI) é a minimização dos custos, considerando custo de transportes, aprovisionamentos, mão-de-obra, visibilidade, consumiveis (combustiveis, electricidade, àgua, etc,). Este problema tem um impacto significativo devido principalmente aos custos de realocação de uma instalação, que normalmente sendo extremamente elevados, tornam-se inviáveis.

5.1 Formulação Matemática

Variáveis de Decisão

$$x_{i,j} : \text{quantidade de procura do cliente } j \text{ satisfeita pela instalação } i \tag{5.1}$$

$$y_i : \begin{cases} 1, & \text{instalação } i \text{ utilizada} \\ 0, & \text{caso-contrário} \end{cases} \tag{5.2}$$

Parámetros

$$c_{i,j} : \text{Custo de fornecimento ao cliente } j \text{ pela instalação } i \tag{5.3}$$

$$f_i : \text{custo fixo de posicionar instalação no local } i \tag{5.4}$$

$$d_j : \text{Procura do cliente } j \tag{5.5}$$

Conjuntos

$$V : \text{lista de vértices}$$

F : lista de instalações disponíveis

Função Objectivo

$$min \sum_{i \in F} f_i y_i + \sum_{i,j \in V} c_{i,j} x_{i,j} \tag{5.6}$$

Restrições

$$\sum_{i \in F} x_{i,j} = d_j, \forall j \in V \tag{5.7}$$

$$x_{i,j} \leq M.y_i, \forall i \in F, j \in V \tag{5.8}$$

$$x_{i,j} \geq 0, \qquad \forall y_i \in \{0,1\}, \forall i \in F, j \in V \tag{5.9}$$

A variável de decisão $x_{i,j}$ especificada em (5.1) é contínua, indicando a quantidade relativa de procura do cliente i que é satisfeita pela instalação j. A variável $y_{i,j}$ em (5.2) é binária e indica que a instalação i é utilizada quando é 1, e indica que não é utilizada sendo 0. Os parametros $c_{i,j}$ em (5.3) e f_i em (5.4) indicam o custo de fornecimento ao cliente i pela instalação j e o custo de posicionar uma instalação no local i, respectivamente. Considerando a função objectivo, apresentada em (5.6), espera-se minimizar o custo de posicionamento da instalação e o custo total de transporte de produto da instalação para cada cliente. As restrições (5.7) asseguram que a procura de cada cliente j é satisfeita na sua totalidade pelas instalações i . As restrições (5.8) activam a variável y_i quando existe uma procura satisfeita em $x_{i,j}$ definida para a instalação i. As restrições (5.9) especifica que $x_{i,j}$ é não-negativa, e que y_i é uma variável binária.

Outras restrições podem ser adicionadas como extensões a este tipo de problema, considerando-se restrições de capacidade máxima para cada armazém (PLI com Capacidades), vários produtos em cada armazém (PLI Multiproduto), fluxos de produtos entre armazéns (PLI com transferências) e integrado com o problema de roteamento de veículos (PLI com Roteamento).

5.2 Algoritmos

Para a solução do problema de localização de instalações são paresentados os seguintes algoritmos:

- Algoritmo Gravítico Euclidiano Ballou, 2004
- Algoritmo Rectilineo Larson e Sadiq, 1983
- Algoritmo de Factores Ballou, 2004
- Algoritmo de Reilly Reilly, 1931
- Algoritmo de Interacção Espacial Ballou, 2004

5.2.1 Algoritmo Gravítico Euclediano

5.2.1.1 Descrição

Este algoritmo minimiza a distância total entre armazéns e clientes. A distância é medida absoluta entre dois pontos utilizando a métrica de geometria euclediana (verificada através da aplicação do teorema de Pitágoras).

Sendo:

a, b : coordenadas da nova instalação

x_i, y_i : coordenadas do cliente i

w_i : peso atribuído ao cliente i

K : factor de escala (normalmente 1)

Considera-se que o objectivo é a minimização de todas as distãncias (ou custos) somadas entre o armazém localizado em (a, b) e todos os clientes (x_i, y_i).

$$\min \text{ C} = \sum_{i=1}^{n} w_i d_i$$

em que o peso w_i pode ser composto por multiplos factores, entre os quais se destacam:

- r_i que representa o custo unitário de transporte para o local i.
- V_i representando o fluxo de mercadorias para o local i.
- Outros factores.

sendo:

$$w_i = r_i V_i$$

e a seguinte equação de distãncia euclediana para determinar a distância entre o armazém (a, b) e cada um dos clientes (x_i, y_i):

$$d_i = K\sqrt{(a - x_i)^2 + (b - y_i)^2}$$

Inicialização

O algoritmo é uma abordagem iterativa que começa por estimar os valores iniciais da localização da instalação (a, b) utilizando as seguintes equações:

$$a = \frac{\sum_{j=1}^{m} x_j r_j V_j}{\sum_{j=1}^{m} r_j V_j}$$

$$b = \frac{\sum_{j=1}^{m} y_j r_j V_j}{\sum_{j=1}^{m} r_j V_j}$$

Com os valores obtidos de a e b calculam-se os valores respectivos de distância do armazém para cada cliente d_i utilizando a equação:

$$d_i = K\sqrt{(a - x_i)^2 + (b - y_i)^2}$$

Iterações seguintes

A cada iteração, a localização do armazém (a, b) é actualizada utilizando as expressões modificadas:

$$a = \frac{\sum_{j=1}^{m} \frac{x_j r_j V_j}{d_j}}{\sum_{j=1}^{m} \frac{r_j V_j}{d_j}}$$

$$b = \frac{\sum_{j=1}^{m} \frac{y_j r_j V_j}{d_j}}{\sum_{j=1}^{m} \frac{r_j V_j}{d_j}}$$

Se o valor de a, b estimado na iteração actual tiver uma variação inferior a um limite estipulado dos valores de a, b da iteração anterior, então o algoritmo termina. Caso o valor de variação esteja acima do limite, então repete-se este processo iterativo. O valor de limite estipulado é um valor pequeno que permite avaliar se o algoritmo convergiu para a localização de custo mínimo.

5.2.1.2 Ilustração

	x	y	V	R
C1	0	0	60	0.5
C2	2	1	40	0.5
C3	4	3	35	0.5
C4	1	5	15	0.5
C5	6	6	75	0.5

Tabela 5.1 Dados de localização de clientes

Iteração 1

$$a_1 = \frac{0 + 40 + 70 + 7.5 + 225}{30 + 20 + 17.5 + 7.5 + 37.5} = \frac{342.5}{112.5} = 3.044$$

	x_i	y_i	V_i	R_i	V_iR_i	$x_iV_iR_i$	$y_iV_iR_i$
C1	0	0	60	0.5	30	0	0
C2	2	1	40	0.5	20	40	20
C3	4	3	35	0.5	17.5	70	52.5
C4	1	5	15	0.5	7.5	7.5	37.5
C5	6	6	75	0.5	37.5	225	225
Sum					112.5	342.5	335

Tabela 5.2 Dados de localização de clientes, com volumes e custos de transporte

$$b_1 = \frac{0 + 20 + 52.5 + 37.5 + 225}{30 + 20 + 17.5 + 7.5 + 37.5} = \frac{335}{112.5} = 2.978$$

$$d1_1 = 1\sqrt{(3.044 - 0)^2 + (2.978 - 0)^2} = 4.259$$

$$d1_2 = 1\sqrt{(3.044 - 2)^2 + (2.978 - 1)^2} = 2.237$$

$$d1_3 = 1\sqrt{(3.044 - 4)^2 + (2.978 - 3)^2} = 0.956$$

$$d1_4 = 1\sqrt{(3.044 - 1)^2 + (2.978 - 5)^2} = 2.876$$

$$d1_5 = 1\sqrt{(3.044 - 6)^2 + (2.978 - 6)^2} = 4.227$$

$$\begin{aligned} &Custo_1 \\ &= 30 \times 4.259 + 20 \times 2.237 + 17.5 \times 0.956 + 7.5 \times 2.876 + 37.5 \times 4.227 \\ &= 369.3043 \end{aligned}$$

Iteração 2 ... n-1

$$a_2 = \frac{\frac{0}{4.259} + \frac{40}{2.237} + \frac{70}{0.956} + \frac{7.5}{2.876} + \frac{225}{4.227}}{\frac{30}{4.259} + \frac{20}{2.237} + \frac{17.5}{0.956} + \frac{7.5}{2.876} + \frac{37.5}{4.227}} = 3.210$$

$$b_2 = \frac{\frac{0}{4.259} + \frac{20}{2.237} + \frac{52.5}{0.956} + \frac{37.5}{2.876} + \frac{225}{4.227}}{\frac{30}{4.259} + \frac{20}{2.237} + \frac{17.5}{0.956} + \frac{7.5}{2.876} + \frac{37.5}{4.227}} = 2.843$$

$$d2_1 = 1\sqrt{(3.210 - 0)^2 + (2.843 - 0)^2} = 4.288$$

$$d2_2 = 1\sqrt{(3.210 - 2)^2 + (2.843 - 1)^2} = 2.205$$

$$d2_3 = 1\sqrt{(3.210 - 4)^2 + (2.843 - 3)^2} = 0.805$$

$$d2_4 = 1\sqrt{(3.210 - 1)^2 + (2.843 - 5)^2} = 3.088$$

$$d2_5 = 1\sqrt{(3.210 - 6)^2 + (2.843 - 6)^2} = 4.213$$

$$\begin{aligned} &Custo_2 \\ &= 30 \times 4.288 + 20 \times 2.205 + 17.5 \times 0.805 + 7.5 \times 3.088 + 37.5 \times 4.213 \\ &= 367.9285 \end{aligned}$$

Sabendo que o valor de custo ainda não convergiu para um valor em específico (em que a diferença de $Custo_{i-1}$ para $Custo_i$ é inferior a um valor mínimo), as iterações continuam.

Iteração n (Final)

Os resultados finais de todas as 20 iterações, até o valor de custo convergir para um valor estável (entre 0.01 neste caso) estão apresentados na Tabela 5.3

Iteração	a_i	b_i	di_1	di_2	di_3	di_4	di_5	$Custo_i$
1	3,04	2,98	4,26	2,24	0,96	2,88	4,23	369,3043
2	3,21	2,84	4,29	2,20	0,81	3,09	4,21	367,9825
3	3,28	2,85	4,34	2,24	0,74	3,13	4,17	367,8069
4	3,32	2,87	4,38	2,28	0,69	3,15	4,12	367,7012
5	3,35	2,89	4,42	2,32	0,66	3,16	4,09	367,6266
6	3,38	2,90	4,45	2,35	0,63	3,17	4,06	367,5738
7	3,41	2,91	4,48	2,37	0,60	3,18	4,03	367,5365
8	3,43	2,92	4,50	2,39	0,58	3,19	4,01	367,5103
9	3,44	2,93	4,52	2,41	0,56	3,20	4,00	367,4918
10	3,46	2,94	4,54	2,42	0,55	3,21	3,98	367,4788
11	3,47	2,94	4,55	2,43	0,53	3,21	3,97	367,4696
12	3,48	2,95	4,56	2,44	0,52	3,22	3,96	367,4632
13	3,49	2,95	4,57	2,45	0,51	3,22	3,95	367,4586
14	3,49	2,95	4,57	2,46	0,51	3,23	3,95	367,4554
15	3,50	2,95	4,58	2,46	0,50	3,23	3,94	367,4531
16	3,51	2,96	4,59	2,47	0,50	3,23	3,94	367,4515
17	3,51	2,96	4,59	2,47	0,49	3,24	3,93	367,4504
18	3,51	2,96	4,59	2,48	0,49	3,24	3,93	367,4496
19	3,52	2,96	4,60	2,48	0,49	3,24	3,93	367,4490
20	3,52	2,96	4,60	2,48	0,48	3,24	3,92	367,4486

Tabela 5.3 Dados de localização de clientes, com volumes e custos de transporte

O valor calculado na primeira iteração apresenta uma solução inicial proposta para o ponto (3.04; 2.98) com o custo de 369,3043. A ultima iteração termina no ponto (3.52; 2.96) com o custo de 367,4486.

5.2.2 Algoritmo Rectilineo

5.2.2.1 Descrição

Este algoritmo minimiza a distância total entre armazéns e clientes, mas considerando que as movimentações são alinhadas aos eixos cartesianos, i.e,

movimentos rectilíneos. A distância é medida somando as distâncias em cada eixo.

Sendo:

$$a, b : \text{coordenadas da nova instalação}$$

$$x_i, y_i : \text{coordenadas do cliente } i$$

$$w_i : \text{peso atribuído ao cliente } i$$

Considera-se que o objectivo é a minimização de todas as distãncias (ou custos) somadas entre o armazém definido em (a, b) e todos os clientes (x_i, y_i).

$$\min \text{C} = \sum_{i=1}^{n} (w_i \times d_i)$$

em que o peso w_i pode ser composto por multiplos factores, entre os quais se destacam:

- V_i representando o fluxo de mercadorias para o local i.
- Outros factores.

sendo:

$$w_i = V_i$$

e a seguinte equação de distãncia (rectilínea, ou também conhecida como distância de Manhattan) para determinar a distância entre o armazém (a, b) e cada um dos clientes (x_i, y_i):

$$d_i = |a - x_i| + |b - y_i|$$

Esta equação pode ser dividida em duas componentes (dX_i, dY_i) que podem ser calculadas em separado:

$$dX_i = |a - x_i|$$

$$dY_i = |b - y_i|$$

E substituindo na função objectivo, separa-a também em duas componentes que podem ser minimizadas independentemente:

$$\min \text{C} = \sum_{i=1}^{n} (w_i \times dX_i) + \sum_{i=1}^{n} (w_i \times dY_i)$$

A solução óptima das equações tem a coordenada no eixo x idêntica a uma das coordenadas existêntes (de instalações ou clientes utilizados) em x_i assim como umaa coordenada no eixo y é idêntica a uma das coordenadas existêntes em y_i. No entanto, as duas coordenadas x, y não podem coincidir com nenhuma instalação existente definida em x_i, y_i.

5.2.2.2 Ilustração

Considerando o algoritmo apresentado anteriormente, pode aplicar-se ao exemplo da tabela 5.4. Neste exemplo, temos a localização de várias máquinas (x, y) que precisam ser acedidas multiplas vezes cada dia (indicado na coluna $Volume$) e é necessário escolher a localização de um posto de trabalho (a, b) que minimize o custo do fluxo de transporte.

Maquina	X	Y	$Volume$
1	25	51	15
2	20	33	10
3	31	40	25
4	55	25	13
5	50	20	15
6	6	11	10

Tabela 5.4 Dados de localizações de máquinas (x, y), e volume de acessos.

Visto que as coordenadas da localização do posto de trabalho podem ser calculadas de forma independente, calcula-se numa primeira fase a coordenada a e seguidamente, a coordenada b.

Calculo A A tabela 5.5 faz o calculo da componente a que corresponde à coordenada no eixo x, considerando todas as coordenadas x_i existentes (obtidas a partir da localização das máquinas). O custo fluxo é calculado para cada $i = 1...6$ em $V_i \times |x_i - a|$ obtendo-se as várias colunas que são somadas, fornecendo assim o custo de fluxo de transporte associado a cada uma das coordenadas seleccionadas.

			$V_i \times \lvert x_i - a \rvert$					
i	V_i	x_i	$a = 25$	$a = 20$	$a = 31$	$a = 55$	$a = 50$	$a = 6$
1	15	25	0	75	90	450	375	285
2	10	20	50	0	110	350	300	140
3	25	31	150	275	0	600	475	625
4	13	55	390	455	312	0	65	637
5	15	50	375	450	285	75	0	660
6	10	6	190	140	250	490	440	0
Sum			1155	1395	1047	1965	1655	2347

Tabela 5.5 Calculo para todos os valores de a possíveis.

Pode-se verificar que a tabela 5.5 identificou como mínimo a coordenada $a = 31$ que retorna um custo de 1047. Isto pode ser visto resumidamente na tabela 5.6.

Calculo B A tabela 5.7 faz o calculo da componente b que corresponde à coordenada no eixo y, considerando todas as coordenadas y_i existentes

a	$\sum_{i=1}^{n} V_i \times \lvert x_i - a \rvert$
25	1155
20	1395
31	1047
55	1965
50	1655
06	2347

Tabela 5.6 Custo calculados para a.

(obtidas a partir da localização das máquinas). O custo fluxo é calculado para cada $i = 1...6$ em $V_i \times |y_i - b|$ obtendo-se as várias colunas que são somadas, fornecendo assim o custo de fluxo de transporte associado a cada uma das coordenadas seleccionadas.

			$V_i \times \lvert y_i - b \rvert$					
i	V_i	y_i	$a = 51$	$a = 33$	$a = 40$	$a = 25$	$a = 20$	$a = 11$
1	15	51	0	270	165	390	465	600
2	10	33	180	0	70	80	130	220
3	25	40	275	175	0	375	500	725
4	13	25	338	104	195	0	65	182
5	15	20	465	195	300	75	0	135
6	10	11	400	220	290	140	90	0
Sum			1658	964	1020	1060	1250	1862

Tabela 5.7 Calculo para todos os valores de b possíveis

Pode-se verificar que a tabela 5.7 identificou como mínimo a coordenada $b = 33$ que retorna um custo de 1047. Isto pode ser visto resumidamente na tabela 5.8.

b	$\sum_{i=1}^{n} V_i \times \lvert y_i - b \rvert$
51	1658
33	964
40	1020
25	1060
20	1250
11	1862

Tabela 5.8 Custo calculados para b.

Após o calculo das coordenadas nos dois eixos é necessário verificar se elas coincidem com as coordenadas de alguma máquina. Se não existir, a solução é aceite como final. Caso existam, escolhe-se a segunda coordenada que produz o menor custo fluxo de transporte. Neste caso, a nova coordenada é $31, 33$ (para a, b) com um custo total de $1047 + 964 = 2011$.

5.2.3 Algoritmo de Factores/Pontuações

5.2.3.1 Descrição

Este algoritmo permite considerar varios aspectos (como custos, e outros parâmetros) relevantes para a tomada de decisão, quando são apresentadas várias possibilidades. Muitas das vezes, os vários aspectos não podem ser quantificados facilmente, daí se proceder a um ajuste para ser possível fazer uma avaliação a partir de parâmetros qualitativos. Dependendo do caso, será mais apropriado utilizar o modelo aditivo ou multiplicativo, tal como apresentados abaixo.

$$S_j = \sum_{i=1}^{m} (W_i \times F_{ij})$$

$$S_j = \prod_{i=1}^{m} (W_i \times F_{ij})$$

Considera-se que:

S_j : pontuação total para a localização j

F_{ij} : pontuação para o factor i na localização j

W_i : peso para o factor i (em %)

m : nº de factores

O modelo tem várias fases:

1. Definição dos factores mais determinantes e seus pesos considerando a sua importância.
2. Atribuir uma pontuação a cada localização dependendo da sua capacidade em satisfazer cada um dos factores.
3. Multiplicar/Adicionar os pesos de cada factor pela nota de cada localização.
4. Somar todos os pontos obtidos por cada localização e concluir qual é a que apresenta melhores condições à empresa.

5.2.3.2 Ilustração

Um exemplo da aplicação deste modelo é apresentado na tabela seguinte (tabela 5.9):

		Local A		Local B	
Factor de Localização	Peso	Score	Ponderado	Score	Ponderado
Prox. de concorrentes	5	4	20	5	25
Considerações sobre arrendamento	4	3	12	5	20
Espaço de estacionamento	1	0	0	7	7
Prox. de empresas complementares	6	5	30	4	24
Modernidade do armazém	7	6	42	8	56
Facilidade de acesso do cliente	5	4	20	3	15
Impostos	7	6	42	1	7
Prox. de vias de comunicação	2	1	2	8	16
		Total A	168	Total B	170

Tabela 5.9 Calculo do modelo multiplicativo de pontuações.

5.2.4 Algoritmo de Reilly

5.2.4.1 Descrição

Este modelo assenta na observação que duas entidades (por exemplo cidades) atraem clientes (consumidores) de uma outra entidade (cidade intermédia) em directa proporção com o numero de consumidores existentes e proporção inversa ao quadrado das distâncias entre as duas cidades para a cidade intermediária.

$$\frac{B_a}{B_b} = \frac{P_a}{P_b} \times \frac{{D_b}^2}{D_a}$$

Considera-se que:

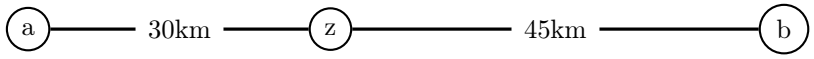
$B_a, B_b =$ volume de negócio da cidade intermédia atraída pela cidade a, ou b

$P_a, P_b =$ população da cidade a e b

$D_a, D_b =$ distãncia da cidade intermédia à cidade a, ou b

Verifica-se que a percentagem da população da cidade z atraída para a é cerca de 53 % e para b é de 47%.

5.2.4.2 Ilustração



$P_a =$ 50 000

$$P_b = \ 75\ 000$$

$$\frac{B_a}{B_b} = (\frac{P_a}{P_b}) \times (\frac{D_b}{D_a})^2 = (\frac{50k}{75k}) \times (\frac{45}{30})^2 = 1.500$$

Sabendo que $B_a = 1.5 \times B_b$, então pode-se verificar que o total de volume de negócios $(B_a + B_b)$ normalizado para B_b corresponde a $1.5 \times B_b + B_b = 2.5$. Sendo $2.5 \times B_b$ equivalente aos 100% de volume de negócios faz que B_a represente $\frac{1}{2.5} = 0.4$, i.e. 40% e B_a seja os restantes 60%.

5.2.5 Algoritmo de Interacção Espacial

5.2.5.1 Descrição

É um modelo gravitico que determina o poder atractivo, ou seja, a conveniência ou quão desejável é um determinado local. Este modelo permite avaliar um conunto de localizações alternativas. A variedade (massa) de retalho atraí consumidores, e a distância aos consumidores funciona como factor repelente. Sendo i o índice do centro populacional e j o índice do local de retalho temos:

$$E_{ij} = P_{ij} \times C_i = \frac{\frac{S_j}{T_{ij}^a}}{\sum_{j=1}^{z} \frac{S_j}{T_{ij}^a}} \times C_i$$

Considerando que:

E_{ij} : procura esperada do centro pop. i atraído ao local de retalho j

P_{ij} : probabilidade de consumidores de i viajarem até ao local j

C_i : procura de consumidor no local i

S_i : tamanho do retalho no local retalho j

T_{ij} : tempo de viagem centro populacional i ao local de retalho j

n : número de locais de retalho j

a : valor estimado empiricamente

5.2.5.2 Ilustração

Dois centros de retalho atraem consumidores dos centros populacionais de C_1, C_2 e C_3. O local de retalho R_1 tem $500000m^2$ enquanto que o local de

retalho R_2 tem cerca de $1000000m^2$. Os consumidores de C_1, C_2 e C_3 têm respectivamente um poder de compra estimado em 10, 15 e 7 milhões. O parametro a é estimado tendo o valor de 2 unidades.

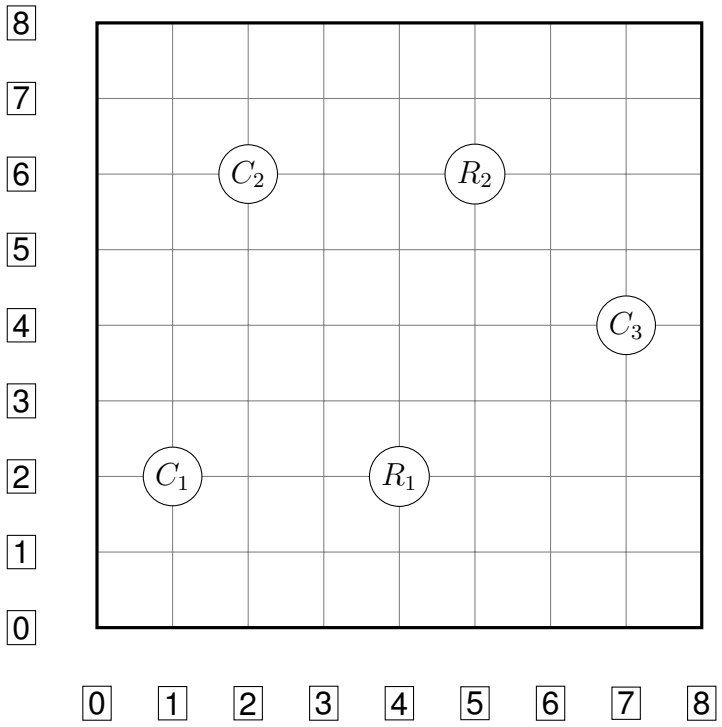


Figura 5.1 Localização de centros populacionais e centros retalho em Km (e assumindo tempo de 10mins/Km).

Considerando este cenário, o calculo do potencial de vendas de cada local de retalho (R_1 e R_2) está demonstrado na tabela 5.10.

Centro Pop.	T_{ij}		T_{ij}^2		$\frac{S_j}{T_{ij}^2}$		P_{ij}		Potencial	E_{ij}	
	R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2		R_1	R_2
C_1	30	56.6	900	3204	556	312	0.640	0.360	10	6.403	3.597
C_2	44.7	30	1998	900	250	1111	0.184	0.816	5	0.919	4.081
C_3	36	28.3	1296	801	386	1249	0.236	0.764	7	1.652	5.348

Tabela 5.10 Potencial de vendas de R_1 e R_2.

Capítulo 6
Problema do Caixeiro-Viajante

6.1 Descrição

O problema do caixeiro-viajante consiste em determinar a rota que um vendedor percorre ao visitar um conjunto de cidades. O vendedor tem que visitar cada uma das cidades começando e terminando na mesma. O desafio deste problema está em minimizar a distância total percorrida pelo vendedor. Resumidamente, pretende-se encontrar o circuito hamiltoniano de menor custo.

6.2 Formulação Matemática

Variáveis de Decisão

$$x_{ij} = \begin{cases} 1, & \text{arco } i,j \text{ utilizado} \\ 0, & \text{caso-contrário} \end{cases}$$

Parámetros

c_{ij} : Custo de navegação entre no arco de j para j

Função Objectivo

$$min \sum_{i,j \in A}^{n} c_i x_i \tag{6.1}$$

Restrições

$$\sum_{i \in V}^{n} x_{i,j} = 1, \forall j \in V, i \neq j \tag{6.2}$$

$$\sum_{j \in V}^{n} x_{i,j} = 1, \forall i \in V, i \neq j \tag{6.3}$$

$$U_i - U_j + n * x_{i,j} < n - 1, (2 \leq i, j \leq n, i \neq j) \tag{6.4}$$

$$x_{i,j} \in \{0, 1\}, \forall i, j \in V \tag{6.5}$$

Conjuntos

$$A = \text{Lista de arcos}$$

$$V = \text{Lista de nós / vértices}$$

$$n = \text{Número total de nós}$$

A variável de decisão $x_{i,j}$ especificada em (6.1) é binária, sendo 1 quando o arco $x_{i,j}$ é utilizado, e zero caso contrário. Os custos de navegação em cada arco são especificados no parâmetro $c_{i,j}$ em (6.1). Considerando a função objectivo, apresentada em (6.1), espera-se minimizar o custo total de percorrer um conjunto de cidades, considerando o custo $c_{i,j}$ para qualquer arco (i, j) percorrido. As restrições (6.2) asseguram que cada nó i tem um e apenas um arco de saída seleccionado para qualquer nó j, assim como as restrições (6.3) asseguram que cada nó j tem um e apenas um arco de entrada seleccionado de qualquer nó i. As restrições (6.5) são restrições de eliminação de sub-circuitos (proveniente da formulação MTZ) utilizando uma restrição de indexação de nós, que indica a sequência de nós a serem visitados, requerendo uma selecção do nó inicial, o nó 1. Estas restrições forçam que não existam sub-circuitos que não contenham o nó 1, e dado que outras restrições forçam que cada nó tenha apenas um arco de entrada e saída, isto garante a existência de apenas uma rota. As restrições (6.5), especificam que $x_{i,j}$ é uma variável binária.

6.3 Algoritmos Construtivos

De forma a resolver o Problema do Caixeiro-Viajante (Robinson, 1949) são apresentados os seguintes algoritmos:

- Vizinho Mais Próximo
- Varrimento Gillett e Miller, 1974
- Clark and Wright Clarke e Wright, 1964

Na tabela 6.1 descrevem-se as coordenadas dos nós (locais).

Os algoritmos vão utilizar como demonstração os nós da figura 6.1 representando os locais a visitar.

Nó	X	Y
1	2	0
2	4	0
3	6	2
4	5	4
5	4	5
6	6	6
7	2	6
8	1	5
9	2	4
10	0	2
V	3	3

Tabela 6.1 Coordenadas dos nós.

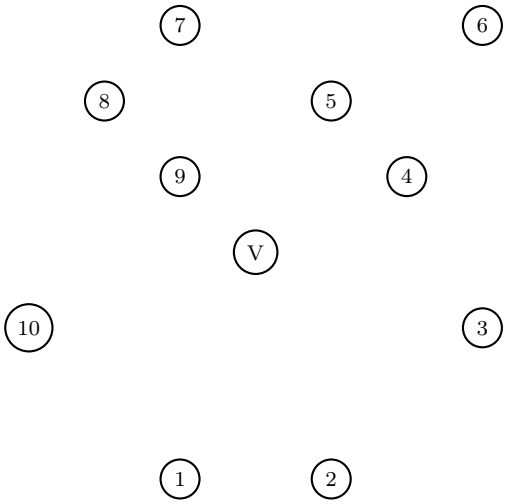


Figura 6.1 Nós a serem visitados com ponto de partida no nó V.

6.3.1 Algoritmo de Vizinho Mais Próximo

6.3.1.1 Descrição

O algoritmo de vizinho mais próximo tem os seguintes passos:

1. Escolher 1 nó como ponto de início
2. Seleccionar o nó não vizitado mais próximo ao actual, e adicionar à rota
3. Se existirem nós não vizitados, repetir passo 2, caso-contrário, termina a rota.

Para obter soluções alternativas, e potencialmente melhores, iniciar o raio numa direcção diferente, e/ou escolher outro nó base como ponto de origem.

6.3.1.2 Ilustração

As figuras 6.2, 6.3, 6.4 ilustram o funcionamento da heuristica Vizinho Mais Próximo. Começando no nó V, o nó mais próximo é o 9, que é adicionado à rota, repetindo-se o procedimento para os nós mais próximos 8, 7, 5, 4, 6, 3, 2, 1, 10 e finalmente fechando a rota em V não havendo mais nós para visitar.

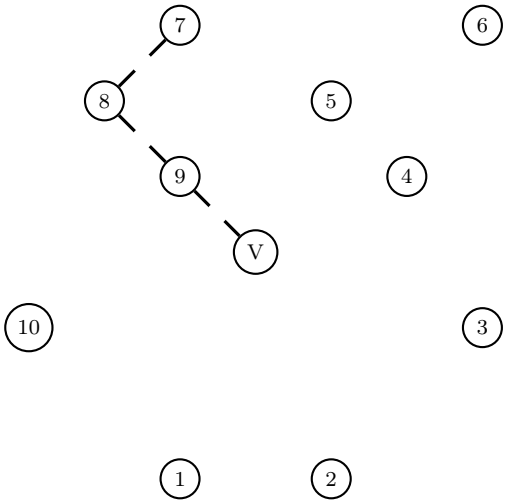


Figura 6.2 Primeiros 3 nós a serem visitados através da heurisitica Vizinho Mais Próximo.

6.3.2 Algoritmo de Varrimento

6.3.2.1 Descrição

O algoritmo de varrimento permite construir a rota a partir de um determinado nó inicial, começando por gerar um raio a partir do mesmo nó e criando a rota num sentido contra-relógio (ou vice-versa) à medida que o raio passa nos nós.

Pode ser descrito através dos seguintes passos:

1. Centrar o raio num nó base numa direcção aleatória
2. Rodando o raio no sentido do relógio (ou contra-relógio) adicionar um nó à rota actual.

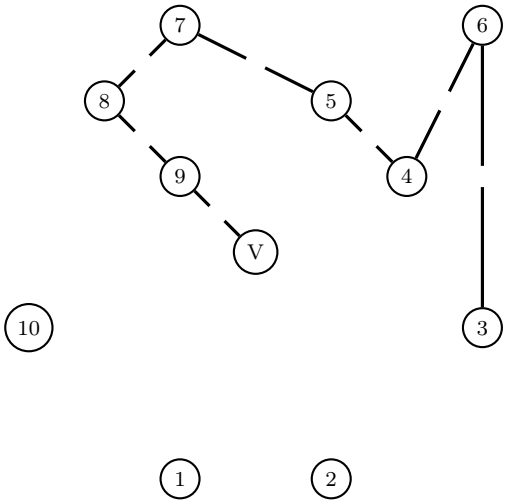


Figura 6.3 Próximos 4 nós a serem adicionados aos nós préviamente seleccionados.

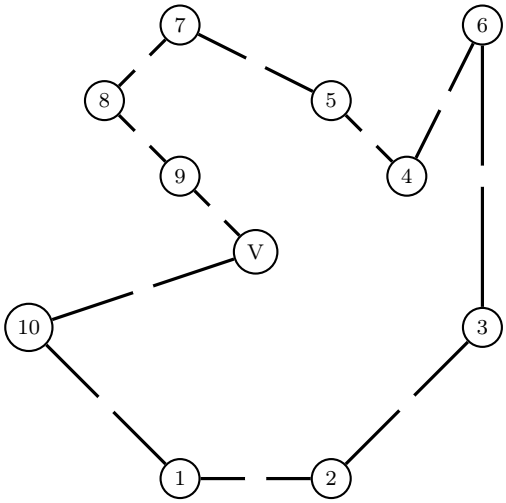


Figura 6.4 Rota completa utilizando a heuristica Vizinho Mais Próximo.

3. Se existirem nós não vizitados, repetir passo 2, caso-contrário, termina a rota, regressando ao nó de origem.

Para obter soluções alternativas, e potencialmente melhores, iniciar o raio numa direcção diferente, e/ou escolher outro nó base como ponto de origem.

6.3.2.2 Ilustração

As figuras 6.5, 6.6, 6.7, 6.8 ilustram o funcionamento da heuristica de Varrimento. Começando no nó V, e utilizando o raio que passa pelo nó 6 (como exemplo) e roda no sentido contra-relógio adicionando nós até completar a rotação, regressando ao nó de partida. Neste exemplo, o primeiro nó é o nó 6, seguido de 5, 7, 9, 8, 10, 1, 2, 3, 4 e finalmente fechando a rota em V não havendo mais nós para visitar, tendo o raio completado uma rotação completa.

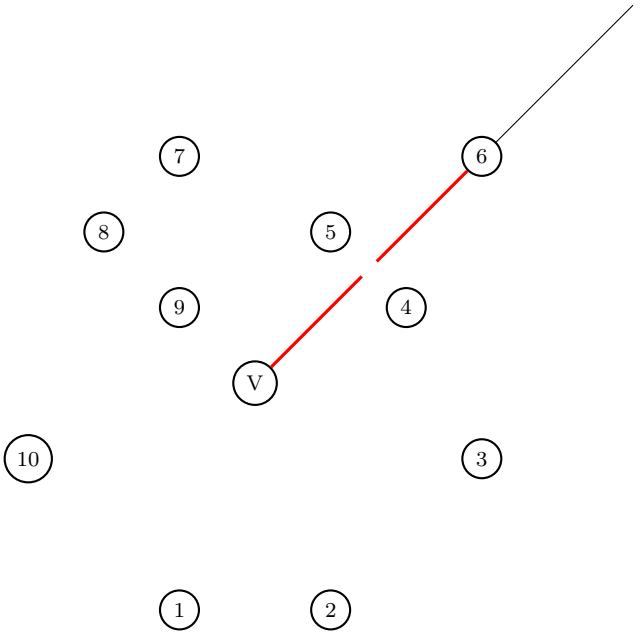


Figura 6.5 Inicialização do algoritmo de Varrimento a partir de nó V e arco a passar pelo nó 6.

6.3.3 Algoritmo de Clark and Wright

O algoritmo de Clark and Wright permite construir rotas considerando uma função que ordena os nós a serem adicionados por máxima poupança, assumindo uma solução inicial em que cada local é visitado uma vez regressando de seguida ao nó de origem. Utilizando os dados do problema anteriormente

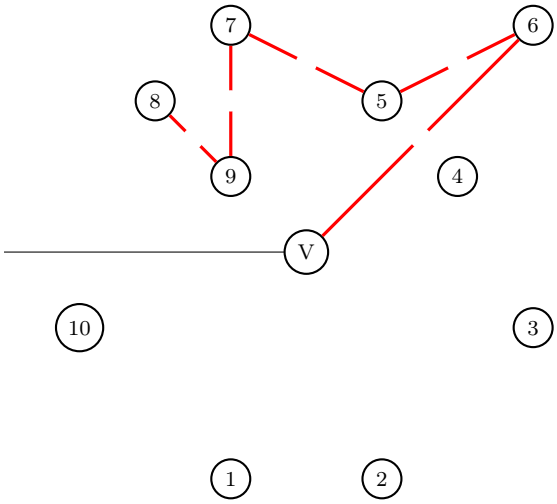


Figura 6.6 Rota construida com rotação contra-relógio do raio com os nós 6, 5, 7, 9 e 8.

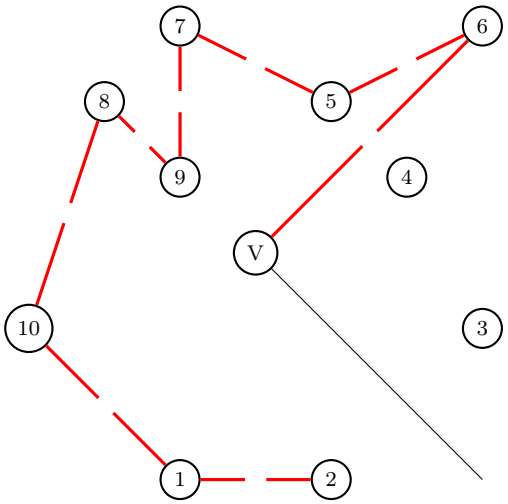


Figura 6.7 Nós 10, 1 e 2 adicionados à rota actual após intersecção com o raio.

apresentado procede-se ao calculo de várias informações relevantes. A tabela 6.2 apresenta a matriz de distâncias entre todos os pares de nós.

A tabela 6.3 apresenta a matriz de poupanças obtidas pela adição de um determinado arco i, j a um dos vértices externos da rota actual.

A lista apresentada na tabela 6.4 mostra os arcos ordenados por poupança ascendenemente de forma a serem mais facilmente identificados, com preferência para os arcos com maior poupança relativamente aos restantes.

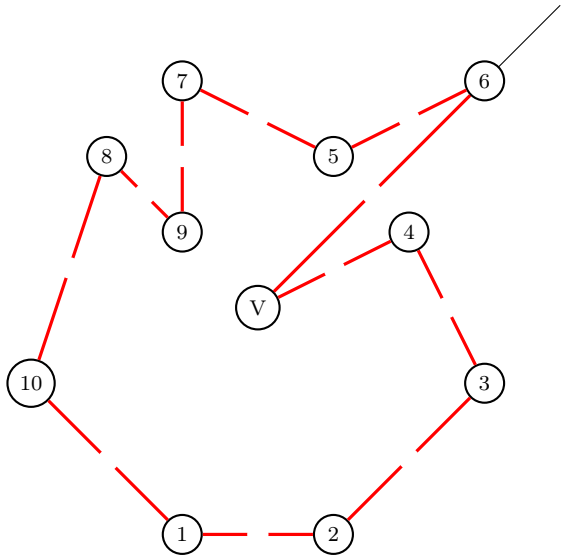


Figura 6.8 Rota completa com nós 3 e 4 após rotação do raio completa usando algoritmo de Varrimento.

D_{ij}	1	2	3	4	5	6	7	8	9	10	V
1		2,00	4,47	5,00	5,39	7,21	6,00	5,10	4,00	2,83	3,16
2	2,00		2,83	4,12	5,00	6,32	6,32	5,83	4,47	4,47	3,16
3	4,47	2,83		2,24	3,61	4,00	5,66	5,83	4,47	6,00	3,16
4	5,00	4,12	2,24		1,41	2,24	3,61	4,12	3,00	5,39	2,24
5	5,39	5,00	3,61	1,41		2,24	2,24	3,00	2,24	5,00	2,24
6	7,21	6,32	4,00	2,24	2,24		4,00	5,10	4,47	7,21	4,24
7	6,00	6,32	5,66	3,61	2,24	4,00		1,41	2,00	4,47	3,16
8	5,10	5,83	5,83	4,12	3,00	5,10	1,41		1,41	3,16	2,83
9	4,00	4,47	4,47	3,00	2,24	4,47	2,00	1,41		2,83	1,41
10	2,83	4,47	6,00	5,39	5,00	7,21	4,47	3,16	2,83		3,16
V	3,16	3,16	3,16	2,24	2,24	4,24	3,16	2,83	1,41	3,16	

Tabela 6.2 Matriz de distâncias para todos os nós.

S_{ij}	1	2	3	4	5	6	7	8	9	10
1		4,3	1,9	0,4	0,0	0,2	0,3	0,9	0,6	3,5
2	4,3		3,5	1,3	0,4	1,1	0,0	0,2	0,1	1,9
3	1,9	3,5		3,2	1,8	3,4	0,7	0,2	0,1	0,3
4	0,4	1,3	3,2		3,1	4,2	1,8	0,9	0,7	0,0
5	0,0	0,4	1,8	3,1		4,2	3,2	2,1	1,4	0,4
6	0,2	1,1	3,4	4,2	4,2		3,4	2,0	1,2	0,2
7	0,3	0,0	0,7	1,8	3,2	3,4		4,6	2,6	1,9
8	0,9	0,2	0,2	0,9	2,1	2,0	4,6		2,8	2,8
9	0,6	0,1	0,1	0,7	1,4	1,2	2,6	2,8		1,7
10	3,5	1,9	0,3	0,0	0,4	0,2	1,9	2,8	1,7	

Tabela 6.3 Matriz de poupanças gerada pela expressão $S_{ij} = d_{0i} + d_{0j} - d_{ij}$.

i	j	Poupança	i	j	Poupança	i	j	Poupança
8	7	4,58	8	6	1,97	9	1	0,58
7	8	4,58	6	8	1,97	1	9	0,58
2	1	4,32	3	1	1,85	4	1	0,40
1	2	4,32	10	2	1,85	5	2	0,40
6	4	4,24	1	3	1,85	1	4	0,40
6	5	4,24	10	7	1,85	2	5	0,40
4	6	4,24	2	10	1,85	10	5	0,40
5	6	4,24	7	10	1,85	5	10	0,40
10	1	3,50	5	3	1,79	7	1	0,32
3	2	3,50	7	4	1,79	10	3	0,32
2	3	3,50	3	5	1,79	1	7	0,32
1	10	3,50	4	7	1,79	3	10	0,32
6	3	3,40	10	9	1,75	6	1	0,19
3	6	3,40	9	10	1,75	1	6	0,19
7	6	3,40	9	5	1,41	10	6	0,19
6	7	3,40	5	9	1,41	6	10	0,19
4	3	3,16	4	2	1,28	8	2	0,16
3	4	3,16	2	4	1,28	8	3	0,16
7	5	3,16	9	6	1,18	2	8	0,16
5	7	3,16	6	9	1,18	3	8	0,16
5	4	3,06	6	2	1,08	9	2	0,10
4	5	3,06	2	6	1,08	9	3	0,10
9	8	2,83	8	4	0,94	2	9	0,10
8	9	2,83	4	8	0,94	3	9	0,10
10	8	2,83	8	1	0,89	5	1	0,01
8	10	2,83	1	8	0,89	10	4	0,01
9	7	2,58	7	3	0,67	1	5	0,01
7	9	2,58	3	7	0,67	4	10	0,01
8	5	2,06	9	4	0,65	7	2	0,00
5	8	2,06	4	9	0,65	2	7	0,00

Tabela 6.4 Lista de poupanças ordenada por arco (i, j).

Na tabela 6.5 apresenta-se a solução inicial formada por multiplas rotas iniciais em que com cada nó é visitado apenas uma vez, com a rota saindo do vértice base V e regressando a ele logo após passar por um nó. A tabela apresenta para cada rota o valor da distância, e inclui o calor da distância total agregada de todas as rotas.

6.3.3.1 Descrição do Algoritmo Paralelo

Este algoritmo começa por criar uma rota para cada nó a partir do nó de origem, e retornando ao mesmo, e iterativamente agregar arcos i, j de maxima poupança aos nós de cada uma das rotas criadas, reduzindo o numero de rotas totais até ficar com uma única rota que passa por todos os nós. É considerado paralelo pois o procedimento de escolha de arcos não é focado na optimização de uma unica rota, mas de multiplas simultaneamente.

Rotas	Dist
V1V	3,16
V2V	3,16
V3V	3,16
V4V	2,24
V5V	2,24
V6V	4,24
V7V	3,16
V8V	2,83
V9V	1,41
V10V	3,16
Total	57,5

Tabela 6.5 Solução inicial com rotas para cada nó incluindo a distância total (57.5).

Pode ser exemplificado através dos seguintes passos:

1. Criar uma solução inicial partindo do nó base e visitando apenas um outro nó, retornando ao nó de origem de seguida.
2. Calcular uma matriz de poupanças baseada na expressão $S_{ij} = d_{0i} + d_{0j} - d_{ij}$.
3. Ordenar todas as poupanças numa lista de forma descendente (maior para menor).
4. Seleccionar o arco de poupança no topo da lista.
5. Escolhendo o arco de máxima poupança i, j, verificar se os nós são compativeis com os nós extremos de alguma rota (ou de um par de rotas) préviamente construida, e caso positivo adiciona à rota rota (ou interliga o par de rotas).
6. Caso existam arcos com potenciais poupanças que permitam extender uma rota ou interligar um par de rotas, repetir passo 5, caso contrário terminar o algoritmo.

Para obter soluções alternativas, e potencialmente melhores, iniciar o raio numa direcção diferente, e/ou escolher outro nó base como ponto de origem.

6.3.3.2 Ilustração

Neste exemplo (6.6) começa-se pelo arco de maior poupança que liga os nós $7, 8$ com uma poupança de 4.58. O arco seguinte com maior poupança liga os nós $1, 2$ com um valor de 4.32, seguido do arco $4, 6$ com o valor de 4.24. Esta ultima rota ($V46V$) é expandida pelo próximo arco de maior poupança $(5, 6)$ pelo nó 6 que é comum a ambas, somando os valores de poupança respectivos $4.24 + 4.24$. Ao executar este procedimento iterativamente as rotas vão sendo agregadas pelos nós extremos até formar apenas uma unica rota. Comparando com a distância da solução inicial de 57.5 e sabendo que a poupança total

é de 33.09 determina-se que rota final tem uma distancia total de 24.44 e é descrita pela sequencia $V-10-1-2-3-4-6-5-7-8-9-V$.

Rotas	Poupança	Valido
V78V	4,58	Não
V12V	4,32	Não
V46V	4,24	Não
V465V	8,49	Não
V1012V	7,82	Não
V10123V	11,32	Não
V10123465V	22,96	Não
V1012346578V	30,70	Não
V10123465789V	33,09	Sim
Total Dist.	24,44	
Total Poup.	33,09	

Tabela 6.6 Iterações da construção da rota para o caixeiro-viajante paralelo.

A figura 6.9 demonstra o resultado do algoritmo de Clark and Wright paralelo utilizado par resolver o problema do Caixeiro-Viajante.

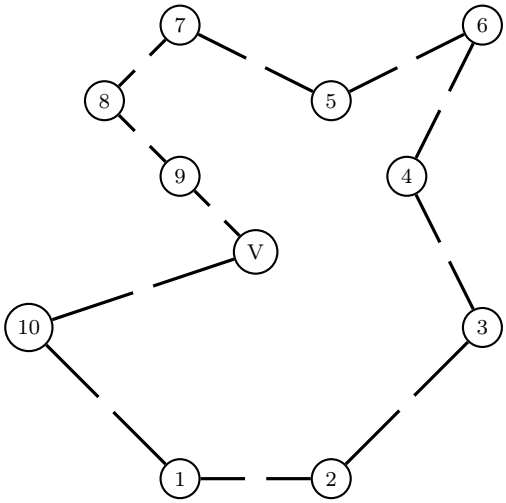


Figura 6.9 Solução com a rota do caixeiro-viajante utilizando Clark and Write Paralelo.

6.3.3.3 Descrição Algoritmo Sequencial

Este algoritmo começa por criar uma rota para cada nó a partir do nó de origem, retornando ao mesmo de seguida. Iterativamente agrega arcos de i, j de maxima poupança numa unica rota extendendo-os pelos nós extremos e reduzindo o numero de rotas totais até ficar com uma única rota que passa por todos os nós. É considerado sequencial pois o procedimento de escolha de arcos é orientado à extensão da rota actualmente seleccionada, através dos seus nós extremos e arcos compatíveis de máxima poupança.

Pode ser exemplificado através dos seguintes passos:

1. Criar uma solução inicial partindo do nó base e visitando apenas um outro nó retornando ao nó de origem de seguida.
2. Calcular uma matriz de poupanças baseada na expressão $S_{ij} = d_{0i} + d_{0j} - d_{ij}$.
3. Ordenar todas as poupanças numa lista de forma descendente (maior para menor).
4. Seleccionar o arco de poupança no topo da lista.
5. Escolhendo o próximo arco de máxima poupança com os nós i, j compativel com os nós extremos da rota actual, adiciona o novo arco a essa rota.
6. Caso existam arcos com potenciais poupanças que permitam extender a rota actual, repetir passo 5, caso contrário terminar o algoritmo.

Para obter soluções alternativas, e potencialmente melhores, iniciar o raio numa direcção diferente, e/ou escolher outro nó base como ponto de origem.

6.3.3.4 Ilustração

Neste exemplo (6.7) começa-se pelo arco de maior poupança que liga os nós $7, 8$ com uma poupança de 4.58. Esta ultima rota ($V78V$) é expandida pelo próximo arco de maior poupança $(6, 7)$ pelo nó 6 que é comum a ambas, somando os valores de poupança respectivos $4.24 + 3.40 = 7.98$. Esta rota continua a ser extendida, de seguida pelo arco $4, 6$ com uma poupança de 4.24 que totaliza $4.24 + 3.40 + 4.24 = 12.22$. Ao executar este procedimento iterativamente as rotas vão sendo agregadas pelos nós extremos até formar apenas uma unica rota. Comparando com a distância da solução inicial de 57.5 e sabendo que a poupança total é de 30.95 determina-se que rota final tem uma distancia total de 26.59 e é descrita pela sequencia $V - 10 - 1 - 2 - 3 - 4 - 6 - 7 - 8 - 9 - 5 - V$.

A figura 6.10 demonstra o resultado do algoritmo de Clark and Wright Sequencial utilizado par resolver o problema do Caixeiro-Viajante.

Rotas	P	Valido
V78V	4,58	Não
V678V	7,98	Não
V4678V	12,22	Não
V34678V	15,39	Não
V234678V	18,88	Não
V1234678V	23,21	Não
V101234678V	26,70	Não
V1012346789V	29,53	Não
V10123467895V	30,95	Sim
Total Dist.	26,59	
Total Poup.	30,95	

Tabela 6.7 Iterações da construção da rota para o caixeiro-viajante sequencial.

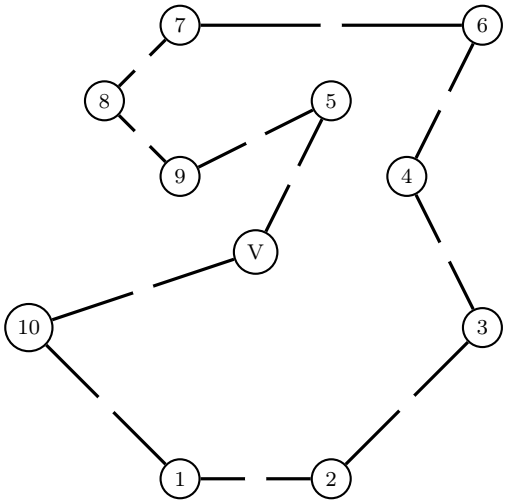


Figura 6.10 Solução com a rota do caixeiro-viajante utilizando Clark and Write Sequencial.

6.4 Algoritmos de Melhoria

A rota obtida através dos algoritmos utilizados para resolver o Problema do Caixeiro-Viajante raramente será a rota optima, e como tal, existe oportunidades para melhorias. Uma forma de o fazer é utilizar algoritmos de melhoria como os especificados de seguida:

- 2-Opt (Intra-Rota)

Algoritmo 2-Opt

6.4.0.1 Descrição

O algoritmo consiste em percorrer iterativamente todos os pares de arcos da rota e conecta-los de uma forma alternativa, verificando se o resultado permite obter uma rota com comprimento inferior. Caso seja inferior, aceita a nova rota como nova solução.

O algoritmo pode ser descrito através dos seguintes passos:

1. Escolher 1 par de arcos da rota.
2. Sendo o primeiro arco E_1 e o segundo arco E_2, e para cada um o nó origem N_o e o de destino N_d fazer as trocas de nós: $E_1.N_d$ liga a $E_2.N_o$, $E_2.N_o$ liga a $E_1.N_d$.
3. Inverter sentido dos arcos entre os nós da rota criada: $E_1.N_d$ até $E_2.N_o$.
4. Verificar se a nova rota tem uma distância total inferior á rota original, e caso se confirme, aceita a nova rota como solução.
5. Regressa ao passo 1, até um número especificado de verificações ser atingido.

6.4.0.2 Ilustração

O procedimento parte de uma rota já pré construída, tal como visto na Figura 6.11. Nesta iteração escolhem-se dois arcos com os nós $4, 6$ e $5, V$. Procedendo ás trocas ficamos com os novos arcos ligando $4, 5$ e $6, V$. Os arcos entre os nós 6 e 5 são todos invertidos de sentido. Isto produz a nova rota mostrada na Figura 6.12. Considerando que os arcos $4, 6$ e $5, V$ tinham as distâncias respectivas de $2.24 + 2.24 = 4.49$, e os novos arcos $4, 5$ e $6, V$ têm a distância de $1.41 + 4.24 = 5.65$ verifica-se que a solução não é superior, logo sendo descartada.

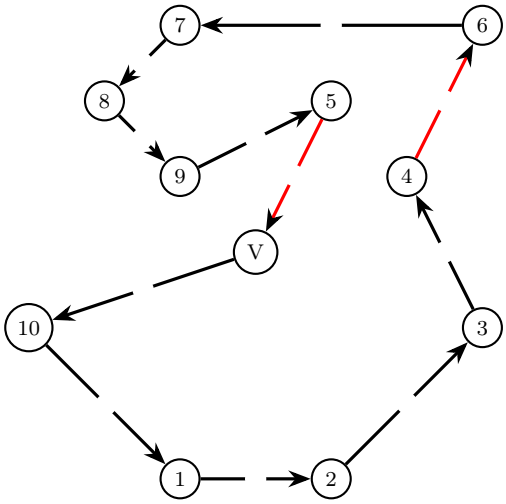


Figura 6.11 Rota original com 2 arcos seleccionados.

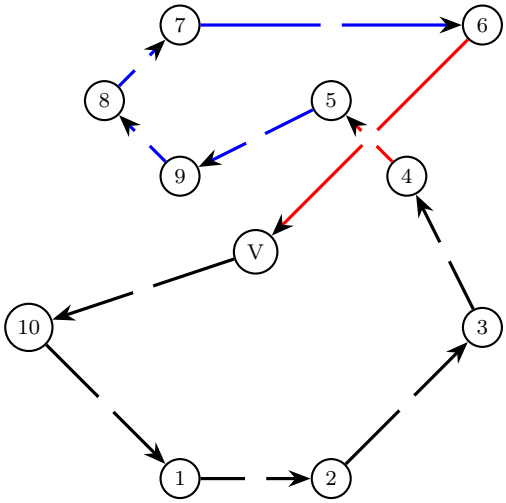


Figura 6.12 Rota nova com 2 arcos trocados e arcos intermédios com sentido invertido.

Capítulo 7
Problema de Roteamento de Veículos

7.1 Descrição

O problema de roteamento de veículos consiste em determinar as rotas necessárias para visitar um conjunto de clientes por um ou mais veículos. Cada veículo tem que visitar cada uma dos clientes começando e terminando numa base (geralmente só existe uma, mas poderão existir várias). O desafio deste problema está em minimizar a distância total percorrida pelos veículos.

7.2 Formulação Matemática

Variáveis de Decisão

$$x_{ij} = \begin{cases} 1, & \text{arco } i,j \text{ utilizado} \\ 0, & \text{caso-contrário} \end{cases} \tag{7.1}$$

Parámetros

$$c_{i,j} - \text{Custo de navegação entre no arco de } j \text{ para } j \tag{7.2}$$

Conjuntos

$$S - \text{Lista de nós}$$

$$A - \text{Lista de arcos}$$

$$V - \text{Lista de vértices}$$

Função Objectivo

$$min \sum_{i,j \in A} c_{i,j} x_{i,j} \tag{7.3}$$

Restrições

$$\sum_{i \in V} x_{i,j} = 1, \forall j \in V, i \neq j \tag{7.4}$$

$$\sum_{j \in V} x_{i,j} = 1, \forall i \in V, i \neq j \tag{7.5}$$

$$\sum_{i \in V} x_{i,0} = \sum_{j \in V} x_{0,j} \tag{7.6}$$

$$\sum_{j \in V} x_{0,j} \leq K \tag{7.7}$$

$$\sum_{i \in S, j \in S} x_{i,j} \leq |S| - r(S), \forall (S \subseteq V, S \neq 0) \tag{7.8}$$

$$x_{i,j} \in \{0, 1\}, \forall i, j \in V \tag{7.9}$$

A variável de decisão $x_{i,j}$ especificada em (7.1) é binária, sendo 1 quando o arco $x_{i,j}$ é utilizado, e zero caso contrário. Os custos de navegação em cada arco são especificados no parâmetro $c_{i,j}$ em (7.2).

Considerando a função objectivo, apresentada em (7.3), espera-se minimizar o custo de percorrer o total do conjunto de cidades, considerando o custo $c_{i,j}$ para qualquer arco (i, j) percorrido.

As restrições (7.4) asseguram que cada nó i tem um e apenas um arco de saída seleccionado para qualquer nó j, assim como as restrições (7.5) asseguram que cada nó j tem um e apenas um arco de entrada seleccionado de qualquer nó i.

As restrições (7.6) asseguram que qualquer rota começa sempre no nó 0, e termina no nó 0 e a restrição (7.7) limita o número de rotas criadas a um máximo de K através do número de arcos de saída do nó 0.

As restrições em (7.8) garantem que não existem ciclos, sendo as restrições de eliminação de sub-circuitos.

As restrições (7.9), especificam que $x_{i,j}$ é uma variável binária.

7.3 Algoritmos Construtivos

Para a resolução do problema de roteamento de veículos são apresentados os seguintes algoritmos:

- Vizinho Mais Próximo
- Varrimento Gillett e Miller, 1974
- Clark and Wright Clarke e Wright, 1964

A diferença entre estes algoritmos e as versões similares aplicadas no Problema do Caixeiro-Viajante é que estes algoritmos agora contam com utilização de capacidades dos veículos como factores limitantes. Como valores para os exemplos a nivel das necessidades em cada nó e capacidade do veículo são utilizados os valores na Tabela 7.1. A mesma tabela descreve também as coordenadas dos vários nós.

Nó	X	Y	Cap. (unid.)
1	2	0	75
2	4	0	140
3	6	2	60
4	5	4	120
5	4	5	90
6	6	6	85
7	2	6	100
8	1	5	25
9	2	4	75
10	0	2	80
V	3	3	
Veiculo			300

Tabela 7.1 Coordenadas de cada nó e capacidade requisitada do veículo.

Os algoritmos vão utilizar para ilustração os nós da Figura 7.1 representando os clientes a serem fornecidos.

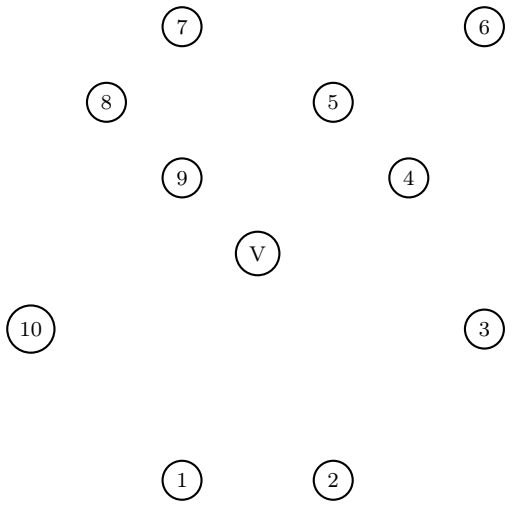


Figura 7.1 Nós a serem visitados com ponto de partida no nó V.

7.3.1 Algoritmo de Vizinho Mais Próximo

7.3.1.1 Descrição

O algoritmo de vizinho mais próximo constrói rotas atendendo à capacidade dos veículos escolhendo o nó mais próximo iterativamente até a capacidade ser esgotada. Pode ser exemplificado através dos seguintes passos:

1. Escolher um nó base como ponto de início
2. Seleccionar o nó não vizitado mais próximo ao actual que não exceda a capacidade do veículo e adicionar à rota, caso contrário, completa a rota retornando à base.
3. Se existirem nós não vizitados, repetir passo 2, caso-contrário, termina a rota.

Para obter soluções alternativas, e potencialmente melhores, iniciar o raio numa direcção diferente, e/ou escolher outro nó base como ponto de origem.

7.3.1.2 Ilustração

As figuras 7.2, 7.3, 7.4 ilustram o funcionamento da heuristica de Vizinho mais Proximo num contexto utilizando veículos com capacidades. Começando no nó V, o primeiro nó mais próximo é o nó 9, seguido dos nós 8, 7 e 5 e regressando a V dado a que não existem nós cuja entrega ainda seja possível com a capacidade restante do veículo. As seguinte rota usa o nó 4, 6 e 3, e finalmente a ultima rota termina com os nós 10, 1 e 2. A capacidade de 300 unidades não é excedida, tendo cada uma das rotas respectivamente um total de 290, 265 e 295 unidades.

7.3.2 Algoritmo de Varrimento

7.3.2.1 Descrição

O algoritmo de varrimento permite construir multiplas rotas a partir de um determinado nó inicial, começando por gerar um raio a partir do mesmo nó e criando as rotas num sentido contra-relógio (ou vice-versa) à medida que o raio passa nos nós.

Pode ser descrito através dos seguintes passos:

1. Centrar o raio num nó base numa direcção aleatória
2. Rodando o raio no sentido do relógio (ou contra-relógio) adicionar um nó à rota actual que não exceda a capacidade do veículo, caso contrário, completa a rota retornando à base.

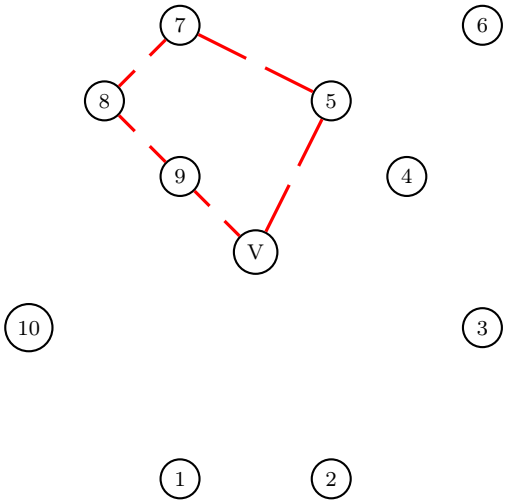


Figura 7.2 Primeira rota ($V-9-8-7-5-V$) usando capacidade de $75+25+100+90 = 290$.

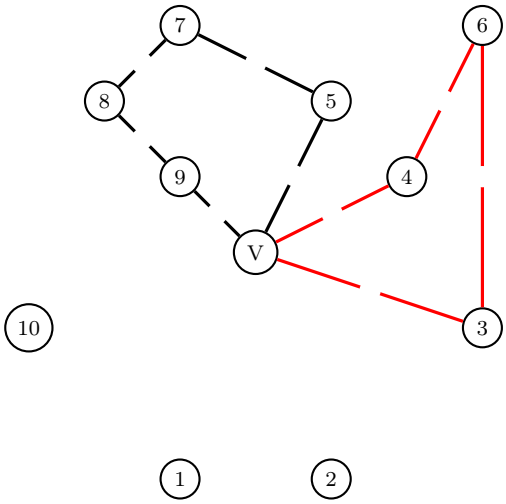


Figura 7.3 Segunda rota ($V-4-6-3-V$) usando capacidade de $120+85+60 = 265$.

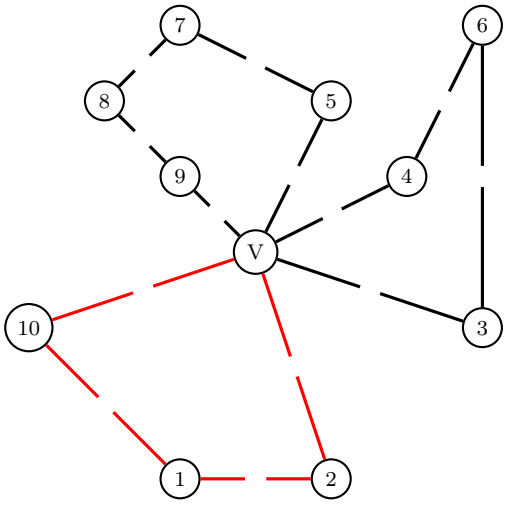


Figura 7.4 Ultima rota ($V-10-1-2-V$) usando capacidade de $80+75+140=295$.

3. Se existirem nós não vizitados, repetir passo 2, caso-contrário, termina a rota.

Para obter soluções alternativas, e potencialmente melhores, iniciar o raio numa direcção diferente, e/ou escolher outra base como ponto de origem.

7.3.2.2 Ilustração

As figuras 7.5, 7.6, 7.7, 7.8 ilustram o funcionamento da heuristica de Varrimento com capacidades. Começando no nó V utiliza o raio que passa pelo nó 6 (como exemplo) e rodando no sentido contra-relógio vai adicionando nós até completar a rotação, ou até a capacidade do veículo ser esgotada, e regressando de seguida ao nó de partida. Neste exemplo, a primeira rota começa com o primeiro nó sendo o nó 6, seguido de 5, 7, com as capacidades de $85+90+100=275$ e regressando a V. A segunda rota parte de V para o nó 9, seguindo para 8, 10, 1, terminando em V, com capacidades de $75+25+80+75=255$. A terceira rota começa em V passando por 2 e 3, regressando a V com capacidades de $140+60=200$. Finalmente, a ultima rota consiste tem apenas o nó 4, com capacidade de 120 retornando para V devido a não haver mais nós para visitar, e tendo o raio completado uma rotação completa.

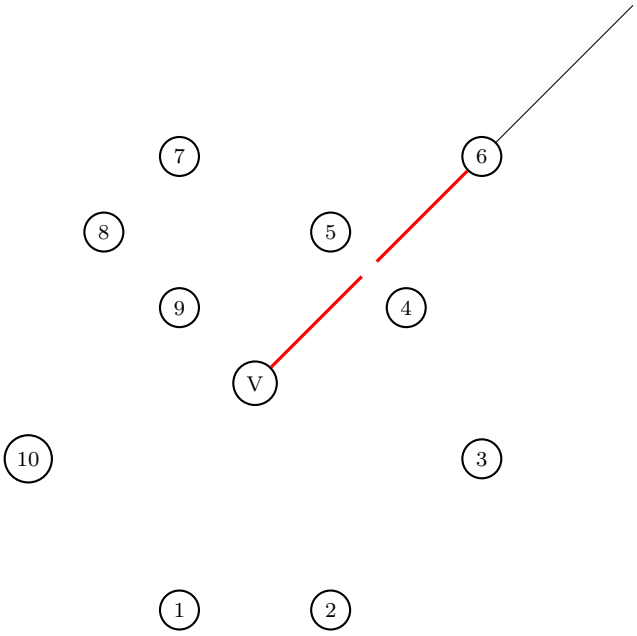


Figura 7.5 Inicialização do algoritmo de Varrimento a partir de nó V e arco a passar pelo nó 6.

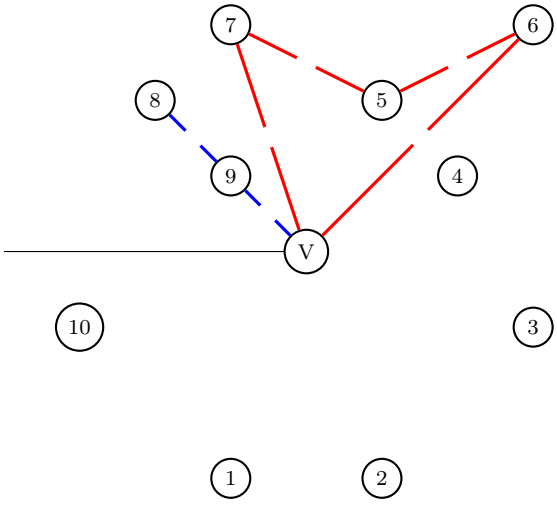


Figura 7.6 Rota construida com rotação contra-relógio do raio com os nós 6, 5, 7, 9 e 8.

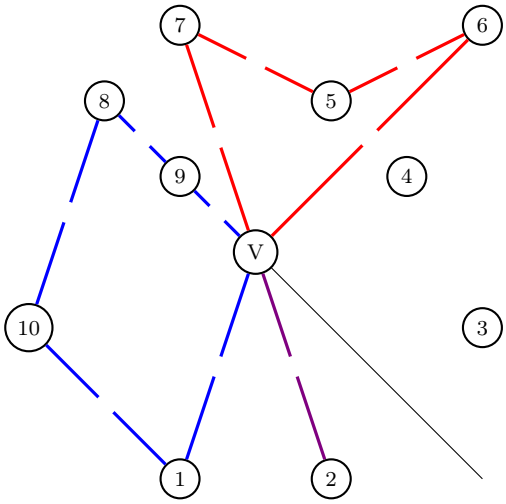


Figura 7.7 Nós 10, 1 e 2 adicionados à rota actual após intersecção com o raio.

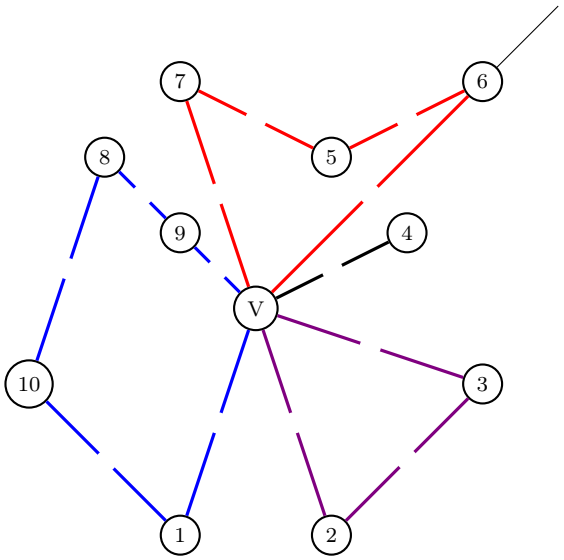


Figura 7.8 Rotas completas com nós 3 e 4 após rotação do raio completa usando algoritmo de Varrimento.

7.3.3 Algoritmo de Clark and Wright

O algoritmo de Clark and Wright permite construir rotas considerando uma função que ordena os nós a serem adicionados por máxima poupança, assumindo uma solução inicial em que cada veículo visita apenas um dos clientes. Utilizando os dados do problema anteriormente apresentado procede-se ao calculo de várias informações relevantes. A tabela 7.2 apresenta a matriz de distâncias entre todos os pares de nós.

D_{ij}	1	2	3	4	5	6	7	8	9	10	V
1		2,00	4,47	5,00	5,39	7,21	6,00	5,10	4,00	2,83	3,16
2	2,00		2,83	4,12	5,00	6,32	6,32	5,83	4,47	4,47	3,16
3	4,47	2,83		2,24	3,61	4,00	5,66	5,83	4,47	6,00	3,16
4	5,00	4,12	2,24		1,41	2,24	3,61	4,12	3,00	5,39	2,24
5	5,39	5,00	3,61	1,41		2,24	2,24	3,00	2,24	5,00	2,24
6	7,21	6,32	4,00	2,24	2,24		4,00	5,10	4,47	7,21	4,24
7	6,00	6,32	5,66	3,61	2,24	4,00		1,41	2,00	4,47	3,16
8	5,10	5,83	5,83	4,12	3,00	5,10	1,41		1,41	3,16	2,83
9	4,00	4,47	4,47	3,00	2,24	4,47	2,00	1,41		2,83	1,41
10	2,83	4,47	6,00	5,39	5,00	7,21	4,47	3,16	2,83		3,16
V	3,16	3,16	3,16	2,24	2,24	4,24	3,16	2,83	1,41	3,16	

Tabela 7.2 Matriz de distâncias para todos os nós.

A tabela 7.3 apresenta a matriz de poupanças obtidas pela adição de um determinado arco i, j a um dos vértices externos da rota actual.

S_{ij}	1	2	3	4	5	6	7	8	9	10
1		4,3	1,9	0,4	0,0	0,2	0,3	0,9	0,6	3,5
2	4,3		3,5	1,3	0,4	1,1	0,0	0,2	0,1	1,9
3	1,9	3,5		3,2	1,8	3,4	0,7	0,2	0,1	0,3
4	0,4	1,3	3,2		3,1	4,2	1,8	0,9	0,7	0,0
5	0,0	0,4	1,8	3,1		4,2	3,2	2,1	1,4	0,4
6	0,2	1,1	3,4	4,2	4,2		3,4	2,0	1,2	0,2
7	0,3	0,0	0,7	1,8	3,2	3,4		4,6	2,6	1,9
8	0,9	0,2	0,2	0,9	2,1	2,0	4,6		2,8	2,8
9	0,6	0,1	0,1	0,7	1,4	1,2	2,6	2,8		1,7
10	3,5	1,9	0,3	0,0	0,4	0,2	1,9	2,8	1,7	

Tabela 7.3 Matriz de poupanças gerada pela expressão $S_{ij} = d_{0i} + d_{0j} - d_{ij}$.

A lista apresentada na tabela 7.4 mostra os arcos ordenados por poupança ascendenemente de forma a serem mais facilmente identificados, com preferência para os arcos com maior poupança relativamente aos restantes.

Na tabela 7.5 apresenta-se a solução inicial formada por multiplas rotas iniciais em que com cada nó é visitado apenas uma vez, com a rota saindo do vértice base V e regressando a ele logo após passar por um nó. A tabela

i	j	Poupança	i	j	Poupança	i	j	Poupança
8	7	4,58	8	6	1,97	9	1	0,58
7	8	4,58	6	8	1,97	1	9	0,58
2	1	4,32	3	1	1,85	4	1	0,40
1	2	4,32	10	2	1,85	5	2	0,40
6	4	4,24	1	3	1,85	1	4	0,40
6	5	4,24	10	7	1,85	2	5	0,40
4	6	4,24	2	10	1,85	10	5	0,40
5	6	4,24	7	10	1,85	5	10	0,40
10	1	3,50	5	3	1,79	7	1	0,32
3	2	3,50	7	4	1,79	10	3	0,32
2	3	3,50	3	5	1,79	1	7	0,32
1	10	3,50	4	7	1,79	3	10	0,32
6	3	3,40	10	9	1,75	6	1	0,19
3	6	3,40	9	10	1,75	1	6	0,19
7	6	3,40	9	5	1,41	10	6	0,19
6	7	3,40	5	9	1,41	6	10	0,19
4	3	3,16	4	2	1,28	8	2	0,16
3	4	3,16	2	4	1,28	8	3	0,16
7	5	3,16	9	6	1,18	2	8	0,16
5	7	3,16	6	9	1,18	3	8	0,16
5	4	3,06	6	2	1,08	9	2	0,10
4	5	3,06	2	6	1,08	9	3	0,10
9	8	2,83	8	4	0,94	2	9	0,10
8	9	2,83	4	8	0,94	3	9	0,10
10	8	2,83	8	1	0,89	5	1	0,01
8	10	2,83	1	8	0,89	10	4	0,01
9	7	2,58	7	3	0,67	1	5	0,01
7	9	2,58	3	7	0,67	4	10	0,01
8	5	2,06	9	4	0,65	7	2	0,00
5	8	2,06	4	9	0,65	2	7	0,00

Tabela 7.4 Lista de poupanças ordenada por arco (i, j).

apresenta para cada rota o valor da distância e os requisitos que impactam a capacidade do veículo. Além disso, a tabela inclúi também o valor total da distância agregada de todas as rotas e a carga total a ser transportada.

7.3.3.1 Descrição do Algorimo Paralelo

Este algoritmo começa por criar uma rota para cada nó a partir do nó de origem, retornando ao mesmo de seguida. Iterativamente agrega arcos de i, j de maxima poupança a uma das rotas que ainda não tenha atingido o limite de capacidade do veículo. É considerado paralelo pois o procedimento de escolha de arcos é orientado à extensão de qualquer rota compatível através de seus nós extremoscom os arcos compatíveis de máxima poupança. Pode ser exemplificado através dos seguintes passos:

Rotas	Dist	Cap
V1V	3,16	75
V2V	3,16	140
V3V	3,16	60
V4V	2,24	120
V5V	2,24	90
V6V	4,24	85
V7V	3,16	100
V8V	2,83	25
V9V	1,41	75
V10V	3,16	80
Total	57,5	850

Tabela 7.5 Solução inicial com rotas para cada nó incluindo a distância total (57.5) e capacidade requisitada do veículo.

1. Criar uma solução inicial partindo de um nó base e visitando apenas um nó com um veículo e retornando ao depósito.
2. Calcular uma matriz de poupanças baseada na expressão $S_{ij} = d_{0i} + d_{0j} - d_{ij}$.
3. Ordenar todas as poupanças numa lista de forma descendente (maior para menor).
4. Seleccionar a poupança no topo da lista e analisar o primeiro arco considerando capacidade.
5. Se adicionando o arco com os nós i e j não excede a capacidade do veículo, então adiciona-se o nó à rota respectiva, caso contrário, elimina-se o arco da lista.
6. Caso existam arcos com potenciais poupanças, repetir passo 5, caso contrário terminar o algoritmo.

7.3.3.2 Ilustração

Neste exemplo apresentado na Tabela 7.6 começa-se por seleccionar o arco de maior poupança que liga os nós $7, 8$ com uma poupança de 4.58, e contribuindo $100 + 25 = 125$ unidades para o uso da capacidade de veículo nessa rota. O próximo arco com maior poupança cria uma rota paralela com os nós $1, 2$, com uma poupança de 4.32 e utilizando uma capacidade de $75+140 = 215$ unidades. O terceiro arco de maior poupança $4, 6$ gera uma nova rota paralela com uma poupança de 4.24 utilizando $120+85 = 205$ unidades. O arco de maior poupança seguinte $6, 5$ extende uma rota actual sem atingir o limite de capacidade, tendo uma poupança de $4, 24 + 4, 24 = 8.49$ e aumentando o uso de capacidade para $120 + 85 + 90 = 295$ unidades. Esta extensão faz com que a capacidade não possa ser expandida para servir outro nó, fazendo com que esta rota esteja completa. Inicia-se a próxima iteração com o arco de maior poupança disponível. Ao executar este procedimento ite-

rativamente as rotas vão sendo construidas pela agregação dos nós extremos até serem limitadas pela capacidade do veículo. Comparando com a distância da solução inicial de 57.5 e sabendo que a poupança total é de 24.38 determina-se que as rotas finais têm uma distancia total de 33.16. As rotas completas são descritas pelas sequências $V-4-6-5-V$, $V-10-1-2-V$ e $V-3-7-8-9-V$, tendo utilizado as respectivas capacidades do veículo de 295, 295 e 260 unidades.

Rotas	P	Cap	Valido
V78V	4,58	125	Não
V12V	4,32	215	Não
V46V	4,24	205	Não
V465V	8,49	295	Sim
V1012V	7,82	295	Sim
V789V	7,40	200	Não
V3789V	8,07	260	Sim
Total Dist.	33,16	850	
Total Poup.	24,38		

Tabela 7.6 Iterações da construção da rota para o Caixeiro-Viajante Paralelo.

A figura 7.9 demonstra o resultaado do algoritmo Clark and Wright Paralelo utilizado para resolver o problema de roteamento de veículos com capacidade.

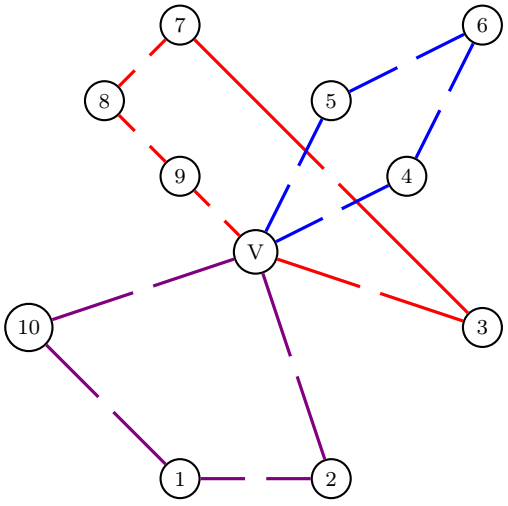


Figura 7.9 Solução com as rotas utilizando Clark and Write Paralelo.

7.3.3.3 Descrição do Algoritmo Sequencial

Este algoritmo começa por criar uma rota para cada nó a partir do nó de origem, retornando ao mesmo de seguida. Iterativamente agrega arcos de i, j de maxima poupança à rota actualmente seleccionada enquanto que a mesma não atingir o limite de capacidade do veículo. É considerado sequencial pois o procedimento de escolha de arcos é orientado à extensão da rota actualmente seleccionada, através dos seus nós extremos e arcos compatíveis de máxima poupança. Pode ser exemplificado através dos seguintes passos:

1. Criar uma solução inicial partindo de um nó base e visitando apenas um nó com um veículo e retornando ao depósito.
2. Calcular uma matriz de poupanças baseada na expressão $S_{ij} = d_{0i} + d_{0j} - d_{ij}$.
3. Ordenar todas as poupanças numa lista de forma descendente (maior para menor).
4. Seleccionar a poupança no topo da lista e analisar o primeiro arco considerando capacidade.
5. Se adicionando o arco com os nós i e j não excede a capacidade do veículo, então adiciona-se o nó à rota respectiva, caso contrário, elimina-se o arco da lista.
6. Definir esta rota actualizada como rota actual (que contém o arco adicionado no passo 5).
7. Identificar os nós das extremidades da rota actual.
8. Determinar o arco com maior poupança que permite expandir a rota actual num dos nós extremos.
9. Adicionar o arco identificado à rota actual.
10. Seleccionar novo arco com maior poupança para criar nova rota. Caso existam arcos com potenciais poupanças, repetir passo 5, caso contrário terminar o algoritmo.

Para obter soluções alternativas, e potencialmente melhores, iniciar o raio numa direcção diferente, e/ou escolher outro nó base como ponto de origem.

7.3.3.4 Ilustração

Neste exemplo apresentado na Tabela 7.7 começa-se por seleccionar o arco de maior poupança que liga os nós $7, 8$ com uma poupança de 4.58, e contribuindo $100 + 25 = 125$ unidades para o uso da capacidade de veículo. Esta ultima rota ($V78V$) é expandida pelo próximo arco de maior poupança $(6, 7)$ ligando o nó 7 que é comum a ambas, somando os valores de poupança respectivos $4.24 + 3.40 = 7.98$ e adicionando mais 85 unidades ao uso da capacidade. Esta rota continua a ser extendida, de seguida pelo arco $5, 6$ com uma poupança de 4.24 que totaliza $4.24 + 3.40 + 4.24 = 12.22$ e ao adicionar 90 unidades ao uso de capacidade atinge a capacidade total de 300 sendo

o limite do veículo. Isto faz com que esta rota não possa ser extendida por mais nenhum nó, estando completa. Inicia-se a próxima iteração com o arco de maior poupança disponível. Ao executar este procedimento iterativamente as rotas vão sendo construidas pela agregação dos nós extremos até serem limitadas pela capacidade do veículo. Comparando com a distância da solução inicial de 57.5 e sabendo que a poupança total é de 23.86 determina-se que as rotas finais têm uma distancia total de 33.68. As rotas são descritas pelas sequências $V-5-6-7-8-V$, $V-10-1-2-V$ e $V-3-4-9-V$, tendo utilizado as respectivas capacidades do veículo de 300, 295 e 255 unidades.

Rotas	P	Cap	Valido
V78V	4,58	125	Não
V678V	7,98	210	Não
V5678V	12,22	300	Sim
V12V	4,32	215	Não
V1012V	7,82	295	Sim
V34V	3,16	180	Não
V349V	3,81	255	Sim
Total Dist.	33,68	850	
Total Poup.	23,86		

Tabela 7.7 Iterações da construção das rotas para o Caixeiro-Viajante Sequencial.

A figura 7.10 demonstra o resultaado do algoritmo Clark and Wright Sequencial utilizado para resolver o problema de roteamento de veículos com capacidade.

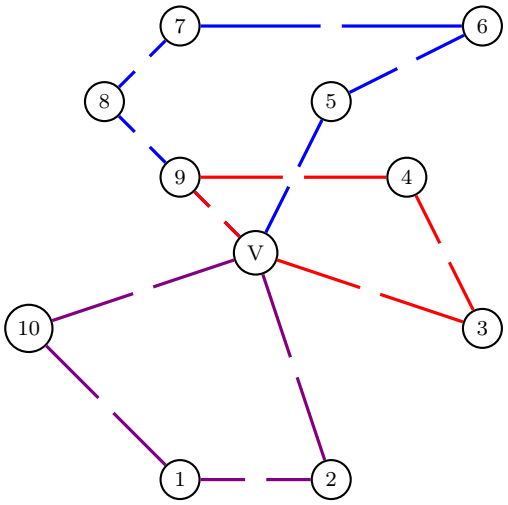


Figura 7.10 Solução com as rotas utilizando Clark and Write Sequencial

7.4 Algoritmos de Melhoria

As rotas obtida através dos algoritmos utilizados para resolver o Problema de Roteamento de Veículos dificilmente serão as rotas optimas, e como tal, existem oportunidades para melhorias. Uma forma de o fazer é utilizar algoritmos de melhoria como os especificados de seguida:

- 2-Opt (Intra-Rota)
- Realocação (Inter-Rota)
- Troca (Inter-Route)

7.4.1 Algoritmo 2-Opt

7.4.1.1 Descrição

O algoritmo consiste em percorrer iterativamente todos os pares de arcos da rota e conecta-los de uma forma alternativa, verificando se o resultado permite obter uma rota com comprimento inferior. Caso seja inferior, aceita a nova rota como nova solução. O algoritmo pode ser descrito através dos seguintes passos:

1. Escolher 1 par de arcos da rota.
2. Sendo o primeiro arco E_1 e o segundo arco E_2, e para cada um o nó origem N_o e o de destino N_d fazer as trocas de nós: $E_1.N_d$ liga a $E_2.N_o$, $E_2.N_o$ liga a $E_1.N_d$.
3. Inverter sentido dos arcos entre os nós da rota criada: $E_1.N_d$ até $E_2.N_o$.
4. Verificar se a nova rota tem uma distância total inferior á rota original, e caso se confirme, aceita a nova rota como solução.
5. Regressa ao passo 1, até um número especificado de verificações ser atingido.

7.4.1.2 Ilustração

O procedimento parte de uma rota já pré construída, tal como visto na Figura 7.11. Nesta iteração escolhem-se dois arcos com os nós $4, 6$ e $5, V$. Procedendo ás trocas ficamos com os novos arcos ligando $4, 5$ e $6, V$. Os arcos entre os nós 6 e 5 são todos invertidos de sentido. Isto produz a nova rota mostrada na Figura 7.12. Considerando que os arcos $4, 6$ e $5, V$ tinham as distâncias respectivas de $2.24 + 2.24 = 4.49$, e os novos arcos $4, 5$ e $6, V$ têm a distância de $1.41 + 4.24 = 5.65$ verifica-se que a solução não é superior, logo sendo descartada.

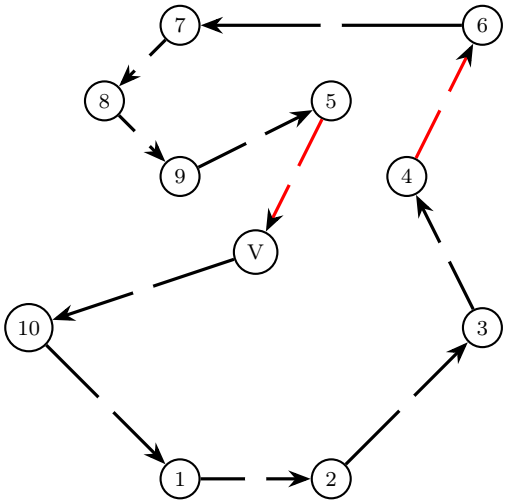


Figura 7.11 Rota original com 2 arcos seleccionados.

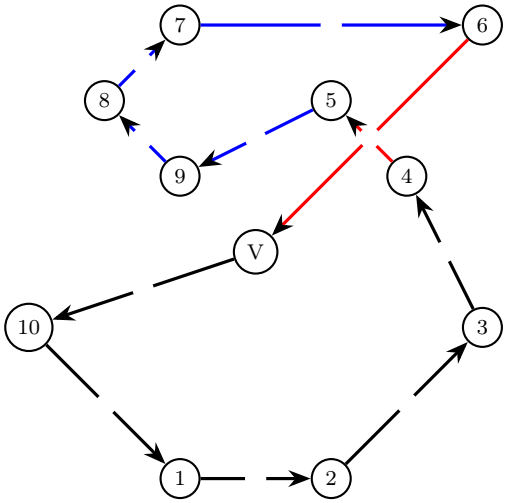


Figura 7.12 Rota nova com 2 arcos trocados e arcos intermédios com sentido invertido.

7.4.2 Realocação

7.4.2.1 Descrição

O algoritmo consiste em seleccionar iterativamente um nó de uma rota e inseri-lo, ou realoca-lo numa outra rota, verificando se o resultado permite

obedecer ás limitações de capacidade das rotas e também obter um comprimento inferior das duas rotas modificadas. Caso seja inferior, aceita a nova configuração de rotas como nova solução. A troca tem que ter em consideração as limitações de capacidade utilizada de cada rota, apenas podendo fazer a troca se os limites não forem excedidos. O algoritmo pode ser descrito através dos seguintes passos:

1. Escolher 1 nó de uma rota e escolher uma rota onde o nó vai ser introduzido.
2. Verificar se a troca dos nós excede o limite de capacidade utilizada em cada rota e caso afirmativo, regressar ao passo 1, caso negativo, continuar para o passo 3.
3. Verificar se as novas rotas geram uma distância total inferior à configuração de rotas originais, e caso se confirme, aceita as novas rotas como solução.
4. Regressa ao passo 1, até um número especificado de verificações ser atingido.

7.4.2.2 Ilustração

O procedimento parte de uma solução já pré construída, com várias rotas, tal como visto na Figura 7.13.

Nesta iteração escolhe-se um nó de uma rota (ex: nó 5 da rota $V5789V$) e uma rota onde vai ser inserido (ex: $V463V$). A rota $V5789V$ tem os requisitos de capacidade de 290 unidades, e perdendo o nó 5, reduzem-se para 200 unidades. Originalmente com $2.24 + 2.24 + 1.41 + 1.41 + 1.41 = 8.71$ terá a denominação $V578V$ e um comprimento de $2.24 + 2.24 + 1.41 + 2.83 = 8.72$.

A rota $V463V$ tem os requisitos de capacidade de 265 unidades, e adquirindo o nó 5 ficando $V5463V$ como se verifica na Figura 7.14, os requisitos aumentam para 355 unidades, excedendo a capacidade máxima especificada para cada rota.

Caso os limites de capacidade fossem cumpridos, verificava-se se a distância total teria reduzido e, caso afirmativo, aceitava-se a alteração das rotas como nova solução.

7.4.3 Troca

7.4.3.1 Descrição

O algoritmo consiste em seleccionar iterativamente pares de nós de rotas diferentes e trocando-os entre as rotas, verificando se o resultado permite obter um comprimento inferior das duas rotas. Caso seja inferior, aceita a nova configuração de rotas como nova solução. A troca tem que ter em consideração

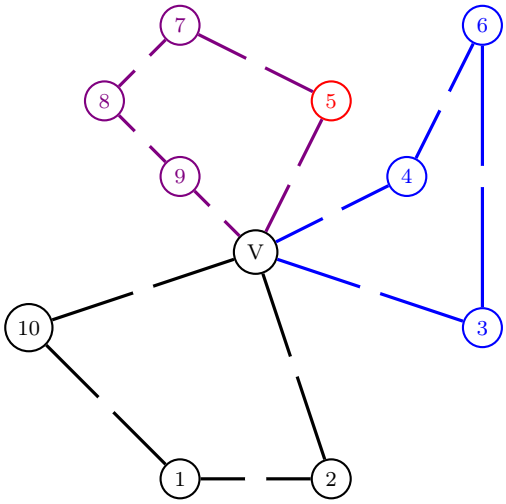


Figura 7.13 Selecção de nó 5 para realocação.

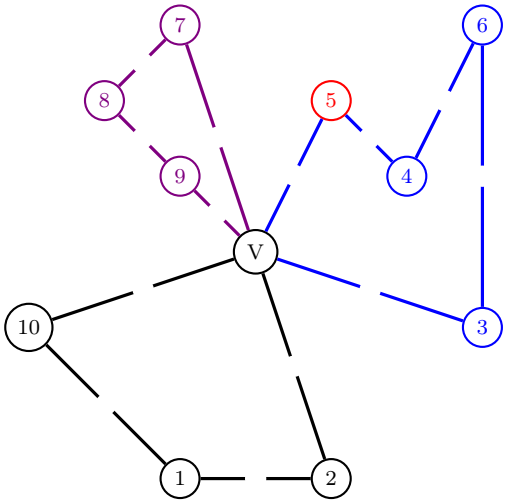


Figura 7.14 Rotas modificadas com realocação do nó 5.

as limitações de capacidade utilizada de cada rota, apenas podendo fazer a troca se os limites não forem excedidos. O algoritmo pode ser descrito através dos seguintes passos:

1. Escolher 1 par de nós de rotas diferentes.

2. Verificar se a troca dos nós excede o limite de capacidade utilizada em cada rota e caso afirmativo, regressar ao passo 1, caso negativo, continuar para o passo 3.
3. Verificar se as novas rotas geram uma distância total inferior à configuração de rotas originais, e caso se confirme, aceita as novas rotas como solução.
4. Regressa ao passo 1, até um número especificado de verificações ser atingido.

7.4.3.2 Ilustração

O procedimento parte de uma solução já pré construída, com várias rotas, tal como visto na Figura 7.15. Nesta iteração escolhem-se dois nós de rotas diferentes (ex: 9 e 10). A rota $V5789V$ tem os requisitos de capacidade de 290 unidades, enquanto a rota $V1012V$ requer uma capacidade de 295 unidades. Sabendo que o nó 9 representa 75 unidades e o nó 10 representa 80 unidades, a troca gera uma rota $V57810V$ necessitando de 295 unidades de capacidade, e outra rota $V912V$ com um requisito de 290 unidades de capacidade, tal como se verifica na Figura 7.16. Ambas as rotas estão abaixo do limite máximo de capacidade do veículo de 300 unidades, logo pode-se verificar a distância total.

A primeira nova rota $V57810V$ tem uma distância de $2.24 + 2.24 + 1.41 + 3.16 + 3.16 = 12.22$ e a segunda rota $V912V$ tem a distância de $1.41 + 4.00 + 2.00 + 3.16 = 10.57$. A distância total das duas rotas adicionadas corresponde a $12.22 + 10.57 = 22.79$ unidades.

Comparativamente com as rotas originais, a rota $V5789V$ tem uma distância de $2.24 + 2.24 + 1.41 + 1.41 + 1.41 = 8.71$ e a rota $V1012V$ tem uma distância de $3.16 + 2.83 + 2.00 + 3.16 = 11.15$ correspondendo a uma distância total de $8.71 + 11.15 = 19.86$ unidades.

Como as rotas geradas através da troca de nós não produzem uma distância total inferior à rota inicial, as novas rotas são descartadas.

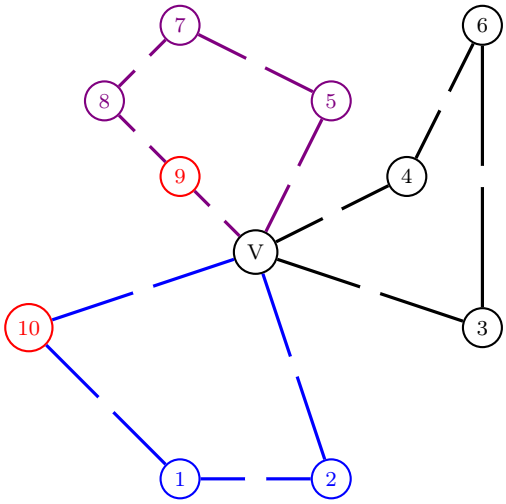


Figura 7.15 Selecção de nós 9 e 10 para troca entre rotas.

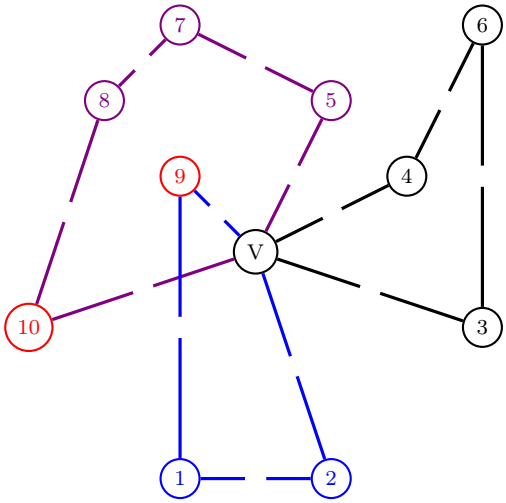


Figura 7.16 Rotas modificadas com nós 9 e 10 trocados.

Referências

Ballou, R.H. (2004). *Business Logistics/supply Chain Management: Planning, Organizing, and Controlling the Supply Chain*. Pearson International edition. Pearson Prentice Hall. ISBN: 9780131230101.

Bellman, Richard (1958). «ON A ROUTING PROBLEM». Em: *Quarterly of Applied Mathematics* 16, pp. 87–90.

Clarke, G. e J. W. Wright (1964). «Scheduling of Vehicles from a Central Depot to a Number of Delivery Points». Em: *Operations Research* 12.4, pp. 568–581. ISSN: 0030364X, 15265463.

Dijkstra, E. W. (dez. de 1959). «A note on two problems in connexion with graphs». Em: *Numerische Mathematik* 1.1, pp. 269–271. ISSN: 0945-3245. DOI: 10.1007/BF01386390.

Floyd, Robert W. (jun. de 1962). «Algorithm 97: Shortest Path». Em: *Commun. ACM* 5.6, p. 345. ISSN: 0001-0782. DOI: 10.1145/367766.368168.

Ford, L. R. e D. R. Fulkerson (1956). «Maximal Flow Through a Network». Em: *Canadian Journal of Mathematics* 8, pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

Gillett, Billy E. e Leland R. Miller (1974). «A Heuristic Algorithm for the Vehicle-Dispatch Problem». Em: *Operations Research* 22.2, pp. 340–349. ISSN: 0030364X, 15265463.

Kruskal, Joseph B. (1956). «On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem». Em: *Proceedings of the American Mathematical Society* 7.1, pp. 48–50. ISSN: 00029939, 10886826. (Acedido em 04/07/2023).

Larson, Richard C. e Ghazala Sadiq (1983). «Facility Locations with the Manhattan Metric in the Presence of Barriers to Travel». Em: *Operations Research* 31.4, pp. 652–669. ISSN: 0030364X, 15265463.

Prim, R. C. (1957). «Shortest connection networks and some generalizations». Em: *The Bell System Technical Journal* 36.6, pp. 1389–1401. DOI: 10.1002/j.1538-7305.1957.tb01515.x.

Reilly, W.J. (1931). *The Law of Retail Gravitation*. W.J. Reilly.

Robinson, J. (1949). *On the Hamiltonian Game (a Traveling Salesman Problem)*. Research memorandum. Rand Corporation.