3
O Problema de Roteirização de Veículos

3.1
O Caso Mais Simples: o TSP

O problema do caixeiro viajante (*Traveling Salesman Problem* – TSP) é um dos problemas mais clássicos da matemática. Ele consiste em se encontrar um percurso de comprimento mínimo, partindo de um dado local de início, visitando um conjunto de pontos na melhor ordem possível, e retornando ao local de origem, utilizando para isso somente os caminhos existentes que ligam esses pontos. Traduzindo este problema para uma linguagem matemática, baseada na teoria de grafos, deseja-se encontrar a rota mínima, que se inicia em um dado nó de um grafo, percorre todos os demais nós deste grafo, e retorna ao nó inicial. Na maior parte das vezes, deseja-se ainda que cada nó seja visitado apenas uma única vez, mas isto só é possível em grafos que atendem a algumas condições específicas, como será visto adiante.

Segundo *Lawler et al* [5], este clássico problema teve sua origem com Merrill Flood na universidade de Princeton, e sua primeira solução foi proposta no artigo "*Solution of a Large Scale Traveling Salesman Problem*" (Dantzig, Fulkerson & Johnson, 1954), sendo a partir daí tema de inúmeras teses e artigos pelo mundo. A grande dificuldade encontrada em se resolver este problema se dá devido ao fato dele ser classificado como um problema NP-completo, ou seja, um problema que não possui solução em um tempo delimitado por um polinômio. Isso faz com que a dificuldade na resolução cresça de forma alarmante conforme se aumenta o número de nós no grafo em questão. Para se ilustrar esta dificuldade, um problema contendo apenas 23 nós, sendo resolvido por um computador capaz de realizar uma comparação a cada nanosegundo (um bilionésimo de segundo), levaria cerca de 178 séculos para varrer todas as soluções possíveis [6]. Entretanto, muito progresso foi feito nas últimas décadas no que diz respeito a técnicas de solução, e em maio de 2004, um problema de 24.978 nós, representando localidades da Suécia, foi resolvido de forma satisfatória, tendo ficado provado que não existe caminho mais curto que o encontrado na ocasião.

Para modelarmos este problema, faremos uso de um grafo *G(N, A),* onde *N* é um conjunto contendo todos os nós do grafo, e *A* é um conjunto com todas os arcos. Um arco *uv* liga o nó *u* ao nó *v* e possui um custo C_{uv}, que neste caso será o seu comprimento propriamente dito. Consideraremos o grafo *G* completo, ou seja, com arcos ligando todos os pares de nós. Alguns exemplos reais se enquadram nestas condições, como por exemplo, um TSP envolvendo um helicóptero, que precisa definir a melhor rota para se visitar um conjunto de plataformas petrolíferas. Este helicóptero pode ir diretamente de uma plataforma qualquer para outra plataforma qualquer, sem precisar passar por plataformas intermediárias.

Além disso, consideraremos também que nossos grafos satisfazem uma desigualdade triangular, ou seja, que a menor distância entre dois nós é a aresta que os conecta. Na prática, isso é natural se considerarmos os custos de cada arco como sendo o seu comprimento físico.

O TSP tradicional define que a rota procurada só pode passar por cada nó uma única vez. Entretanto, isso só é possível se o grafo for classificado como Hamiltoniano; caso contrário, o problema não tem solução. Entretanto, como assumiremos que o grafo é completo e que há uma desigualdade triangular, é sempre possível encontrar uma solução que não visite nenhum nó do grafo mais de uma vez.

3.1.1
Alguns Métodos Heurísticos de Solução do TSP

Primeiramente, é importante definirmos o conceito de "heurística". Na literatura, muitas vezes é feita referência a esta palavra, como uma classificação de um método de resolução de um problema. Na realidade, os modelos de resolução de problemas combinatórios podem ser grosseiramente divididos em três categorias: métodos exatos, heurísticas e meta-heurísticas.

Os métodos exatos são aqueles que chegam à solução ótima de um problema. Toda vez que forem executados, estes métodos devem chegar a esta mesma solução ótima, consistindo em um modelo matemático puramente determinístico. Entretanto, estes métodos são muito complexos e custosos em termos computacionais, sendo sua aplicação inviabilizada para muitos tipos de problemas. O TSP, bem como o problema de roteirização de veículos, são dois

tipos de problemas que se enquadram neste grupo. Estes problemas podem ser classificados na categoria NP-completo, que é uma subcategoria dos problemas NP (N*on-deterministic polynomial time)*. Esta subcategoria engloba todos os problemas cujo tempo de solução não pode ser limitado por um polinômio, conforme mencionado na seção anterior. Em outras palavras, os problemas NP-completo têm sua complexidade aumentada de forma radical conforme o problema cresce em tamanho. Sendo assim, para problemas NP-completos de tamanhos razoáveis, que são comumente encontrados no mundo real, os métodos exatos se mostram inviáveis do ponto de vista prático, sendo sua aplicação restrita a pequenas instâncias de dados. Um exemplo de método exato é o *Branch and Bound* e seus derivados.

Uma heurística, por sua vez, é um algoritmo, ou uma seqüência de passos, que visa encontrar uma solução não necessariamente ótima, mas sim boa o suficiente para um dado problema. Isso é feito explorando-se de forma inteligente o espaço amostral de soluções. Como este espaço pode possuir um tamanho descomunal, e normalmente o possui, nem todas as soluções são avaliadas, mas somente aquelas que se encontram em determinadas áreas abordadas pelo algoritmo da heurística. Boas heurísticas são aquelas eficientes em determinar quais áreas são mais interessantes. Sendo assim, abre-se mão de uma certa acuracidade na solução em prol de um tempo de computação aceitável. Além disso, heurísticas são normalmente caracterizadas por passos simples, que não envolvem uma matemática muito complexa, mas que ao serem executados segundo a definição do algoritmo, levam a soluções razoavelmente boas. A diferença de qualidade das diversas heurísticas se reflete diretamente no quão distante da solução ótima uma dada solução encontrada se situa, além do tempo que se leva para chegar a esta solução.

Finalmente, existem as meta-heurísticas. Estes métodos de solução possuem um conceito mais geral que uma simples heurística, consistindo em um modelo de aplicação mais generalizada e abrangente, geralmente mais robustos e capazes de encontrar boas soluções para diferentes instâncias de dados. Existem muitas definições teóricas que visam diferenciar heurísticas e meta-heurísticas, mas em síntese, meta-heurísticas usualmente possuem algum tipo de memória; são métodos capazes de sair de ótimos locais, descartando uma boa solução em prol de outra pior, porém que ofereça boas condições de levar a um ótimo global; são

aplicáveis a diversos tipos de problemas. Além disso, normalmente possuem como parte do algoritmo alguns elementos probabilísticos associados. É muito comum encontrarmos meta-heurísticas que se utilizam de outras heurísticas como parte de seu algoritmo. Algumas meta-heurísticas conhecidas são os *Ant Colony Systems*, técnicas de *Tabu Search*, *Stimulated Annealing*, entre outras.

Entre as heurísticas conhecidas para a solução do TSP, podemos citar o algoritmo do vizinho mais próximo (*Nearest Neighbor*). Este método é de aplicação muito simples e rápida, como mostra a figura 1. Inicia-se em um nó de origem. Deste, segue-se para o nó mais próximo ainda não visitado, e dessa forma continua-se até que todos os nós tenham sido visitados. Retorna-se então para o nó de origem. Este método é usualmente utilizado para se encontrar uma solução inicial para o problema, solução esta que é então gradativamente melhorada por outras técnicas e modelos. Veremos adiante que os algoritmos AS e ACS são exemplos de modelos que utilizam uma solução inicial encontrada a partir do *Nearest Neighbor*. Uma versão mais genérica deste algoritmo, aplicável ao problema de roteirização de veículos, também será vista nas seções seguintes.

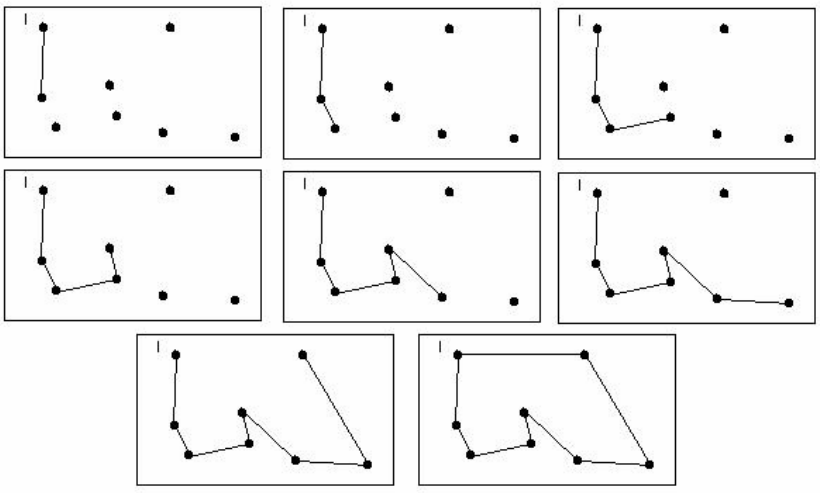

Figura 1 - Algoritmo do vizinho mais próximo

Outro método usual é o de inserção mais próxima (*Nearest Insertion Algorithm*). Inicia-se com uma rota parcial composta de apenas um nó inicial. A partir daí, adiciona-se outros nós à rota, um a um, de acordo com a seguinte lógica: acha-se o custo C_{kj} mínimo para todos os pares de nós k e j, sendo k um dentre os nós não-pertencentes à rota parcial atual, e j um dentre os nós

pertencentes à rota, para todo k e j. O nó k que possua este custo C_{kj} mínimo é, então, selecionado para entrar na rota. Para se definir em que posição o nó k deve ser inserido na rota, escolhe-se o arco ab pertencente à rota atual que minimiza a expressão $C_{ak} + C_{kb} - C_{ab}$, ou seja, o arco ab que, quando substituído pelos arcos ak e kb, adiciona o menor valor possível ao custo total da rota. O algoritmo continua até que todos os nós estejam na rota.

Um método similar ao anterior é o de adição mais próxima (*Nearest Addition Algorithm*). Este método difere do anterior pelo fato de que o nó k é adicionado necessariamente adjacente ao nó j que fornece o menor custo C_{kj}, ou seja, um dos dois arcos incidentes a j é removido e substituído por dois arcos conectando k à rota atual.

Muitas outras heurísticas foram desenvolvidas desde a proposição do problema, há cerca de meio século. Mais recentemente, algumas meta-heurísticas emergiram como métodos mais eficientes de solução do problema. Entre as técnicas mais recentes propostas estão as meta-heurísticas baseadas em colônias de formigas. Nas seções 4 e 5 apresentaremos em detalhes os algoritmos conhecidos como *Ant System* e *Ant Colony System*, que são diretamente aplicados ao TSP e representam uma nova e original forma de resolução.

3.2
O Caso Generalizado: o VRP

O problema de roteirização de veículos (*Vehicle Routing Problem* – VRP) pode ser visto como o caso mais geral do problema do caixeiro viajante. O problema basicamente consiste em definir rotas de custo mínimo para veículos, partindo de um depósito, de forma que a totalidade de clientes de uma região seja atendida por estas rotas. Mais especificamente, deseja-se minimizar o custo total das rotas, mas garantindo que todo cliente seja atendido por um e somente um veículo. Estes clientes estão dispostos geograficamente sobre uma dada área ou região, bem como o depósito, de onde todas as rotas se iniciam e terminam.

Este problema tem sido tema de inúmeros trabalhos científicos, devido à aplicabilidade que possui no dia-a-dia. Uma empresa de varejo que precise entregar seus produtos aos seus clientes precisa definir por que caminhos seus caminhões seguirão, bem como quantos caminhões serão necessários. Um

distribuidor que abastece diversas lojas com seus produtos também se enquadra na definição do problema. Mesmo dentro dos muros de um armazém este problema se mostra presente, como por exemplo no modo mais eficiente de se realizar o *picking* de produtos dispostos em diversos setores do estoque. Uma companhia aérea, que precisa recolher carga e/ou passageiros em diversas cidades, é outro exemplo do problema, e muitos outros casos poderiam ser enumerados.

Pode-se reparar que o problema é muito aplicável ao mundo real, oferecendo um entendimento físico muito simples e direto. Para deixar o problema mais próximo ainda da realidade, algumas restrições adicionais podem ser adicionadas. O *Capacitated Vehicle Routing Problem* (CVRP) é uma variante do problema que define uma capacidade máxima de carga para cada carro ou caminhão, e uma demanda de cada cliente a ser atendido. Desta forma, a solução ótima para o problema não depende apenas das distâncias físicas entre os clientes, mas também precisa garantir que a capacidade dos veículos não seja excedida em momento algum. Usualmente, por VRP já se entende que exista esta restrição de capacidade, ficando como opcional o uso da letra C na definição do problema. Adicionalmente, pode-se definir ainda uma outra restrição de capacidade: o tempo total de cada rota. Isso representa a realidade de forma muito precisa, pois sempre existem jornadas de trabalho a serem respeitadas, bem como capacidades de combustível dos veículos.

Uma restrição adicional, que aumenta significativamente a dificuldade de implementação de algoritmos para o problema, é a utilização de janelas de tempo. Neste tipo de problema, cada cliente possui um intervalo de tempo durante o qual deve ser servido pelas rotas. Cada intervalo de tempo consiste em uma hora de início de atendimento e uma hora de fim, e um veículo que alcance um cliente antes da hora de início do atendimento deverá aguardar. Isso deixa o problema mais próximo ainda da realidade, uma vez que estabelecimentos comerciais e/ou industriais, por exemplo, possuem um horário de funcionamento definido, e portanto devem ser atendidos neste período. As janelas de tempo serão mais bem detalhadas adiante. Outras restrições podem ser adicionadas, como por exemplo uma frota heterogênea de veículos, ou seja, diferentes carros ou caminhões com diferentes capacidades.

Nesta dissertação, o foco principal será no problema de roteirização de veículos com janelas de tempo, que será abordado e discutido em detalhe.

Formalmente, o problema de roteirização com janelas de tempo pode ser descrito, tal como o problema do caixeiro viajante, por um grafo *G(N,A)*, onde *N* é um conjunto contendo todos os nós, e *A* é um conjunto contendo todas as arestas. Os nós representam os clientes, e as arestas os caminhos que ligam estes clientes. O depósito de onde partirão as rotas também é considerado um nó – usualmente, o nó 1. Assim como no TSP, também consideraremos o grafo como sendo completo, o que não representa de forma alguma uma redução da dificuldade do problema. Cada aresta *ij* possui um custo C_{ij} associado. No caso do TSP, este custo era o comprimento de cada arco. No caso do VRPTW, este custo também poderia ser considerado o comprimento de cada arco. Veremos adiante, porém, que precisaremos fazer cálculos de somas e subtrações destes comprimentos com tempos de serviço, para avaliarmos se as janelas de tempo são respeitadas. Como as dimensões seriam diferentes (tempo e distância), correríamos o risco de deixar uma das variáveis desproporcionalmente mais significante que a outra, visto que o sentido físico dos cálculos se perderia. Dessa forma, para manter a coerência física entre as variáveis, podemos utilizar o tempo necessário para se percorrer cada aresta como sendo o custo C_{ij}. Esta medida não altera a proporcionalidade do problema, desde que seja considerado que todos os veículos possuem uma mesma velocidade média, e que todas as arestas do problema são percorridas com esta mesma velocidade média. Essa premissa será adotada em todo o desenvolvimento desta dissertação. Feito isso, os custos C_{ij} podem ser entendidos tanto como o comprimento de cada aresta quanto como o tempo necessário para percorrê-las, pois estes valores serão diretamente proporcionais. Esta é apenas uma formalidade que visa manter a coerência entre as unidades, e sua importância será revelada com o desenvolvimento dos algoritmos.

No depósito, existem *M* veículos à disposição, cada um com uma capacidade *cap*. Cada cliente *i* possui uma demanda d_i associada, bem como um tempo de serviço s_i. Além disso, cada cliente possui também sua janela de tempo, durante a qual deverá ser servido. Esta janela é dada por $[b_i, e_i]$, onde b_i representa o horário de início do atendimento e e_i é o horário final do atendimento, ou seja, é a hora limite para que um carro chegue ao cliente para ser atendido. Veículos que desejem ser atendidos deverão, então, chegar ao cliente entre b_i e e_i. O depósito possui demanda zero e tempo de serviço zero, e sua janela de tempo se estende desde a hora zero até uma hora grande o suficiente para que um caminhão retorne

de qualquer cliente que esteja atendendo. Então, se o cliente mais distante do depósito é também o que possui um horário fim de atendimento mais elevado, como por exemplo um cliente a 2 horas de distância com um horário fim de atendimento de 20 horas, o horário fim da janela do depósito precisa ser de no mínimo $e_0 = (20 + \text{tempo de serviço do cliente em questão} + 2)$, para garantir que um carro que chegue a este cliente momentos antes de sua janela se fechar consiga retornar ao depósito. Este exemplo nos serve ainda de ilustração para o sentido físico dos custos das arestas, explicado no parágrafo anterior. Foi dito que um cliente se situa "a 2 horas de distância" do depósito. Esse seria o custo da aresta que liga o depósito a esse cliente. Se a velocidade média dos veículos fosse de 60 km/h, por exemplo, isso representaria uma aresta de 120km de comprimento. Para evitar que somemos 20 horas, mais o tempo de serviço do cliente (de 1h, por exemplo), com a distância de 120km, o que totalizaria 141 unidades, passamos a considerar o custo da aresta como o tempo necessário para percorrê-la, no caso, 2h. Dessa forma, evitamos que os comprimentos das arestas possuam um peso desproporcionalmente maior nos cálculos. Ao final do problema, se quisermos saber o comprimento físico total das rotas, bastará que se multiplique o tempo total de percurso pela velocidade média dos carros.

O objetivo do problema é encontrar rotas de custo total mínimo que atendam todos os clientes, respeitando as restrições de capacidade dos veículos e das janelas de tempo. Além disso, deseja-se encontrar o número mínimo M de veículos necessários para a realização da tarefa. Na literatura, é comum definir uma hierarquização neste problema, na qual o número de veículos é mais importante que o custo total das rotas. Ou seja, uma solução com um número inferior de veículos é sempre preferível a uma solução de custo total menor, porém com um número maior de veículos. Normalmente, estes objetivos só se mostram antagônicos quando as janelas de tempo são estreitas, o que faz com que um caminhão tenha que andar mais para atender aos clientes que estão com suas janelas de tempo abertas, representando um custo mais elevado. Se as janelas são mais flexíveis, a restrição que tende a ficar ativa é a de capacidade dos caminhões. Isso é intuitivo, mas pode ser constatado através de simulações.

Barán e Schaerer [7] propuseram, em 2003, uma modificação no algoritmo proposto por Gambardella *et al* (conhecido como MACS-VRPTW), visando avaliar várias soluções simultaneamente, e não hierarquicamente, conforme

descrito acima. Segundo os autores, deve-se procurar uma solução ótima de *Pareto*, o que é feito mantendo-se um vetor de soluções que otimizam os diversos objetivos do problema. Dessa forma, o vetor conteria soluções que minimizassem o número de veículos utilizados, bem como soluções que minimizassem o custo total das rotas, ou quaisquer outros objetivos que desejássemos. Fazendo uso do conceito de dominância, novas soluções que surgissem poderiam se sobrepor a uma ou mais soluções presentes neste vetor, o que faria com que estas fossem substituídas pelas novas soluções. Nessa dissertação, entretanto, o foco será dado ao algoritmo original proposto por Gambardella *et al*, que representa a base para a maioria das futuras modificações. Antes disso, porém, mostra-se necessária a exposição de alguns métodos mais simples de construção de soluções para o VRP, conforme será visto a seguir.

3.2.1
Alguns Métodos Heurísticos de Solução do VRP

Nesta seção, falaremos de dois métodos heurísticos simples e conhecidos para a construção de soluções para o problema de roteirização de veículos. Como no caso do TSP, estes métodos são chamados de "heurísticas", pois procuram obter uma solução não necessariamente ótima, mas sim boa o suficiente para o problema. Estes métodos são o *Clarke e Wright* e o *Nearest Neighbor*.

O método de *Clarke e Wright* [8] é de simples execução. A idéia básica do método consiste em se criar um ranking contendo todos os arcos do problema (cada arco conectando um par de nós), ordenados de forma decrescente de acordo com os ganhos que tal arco ofereceria se fosse incluído em alguma rota. Ou seja, considerando-se uma hipotética solução trivial para o problema, na qual cada cliente seria atendido por uma rota própria do tipo depósito-cliente-depósito, define-se que os arcos mais atrativos são aqueles que oferecem um maior potencial de redução do comprimento total das rotas caso sejam percorridos, em substituição de outros dois arcos. Por exemplo, o ganho de um arco i-j representa a magnitude da economia que se obtém por percorrer o caminho 1-i-j-1 ao invés de 1-i-1-j-1.

Formalmente, deve-se calcular o ganho de cada arco da seguinte maneira:

$$G_{ij} = C_{1i} + C_{1j} - C_{ij} \quad (1)$$

O nó 1 é o depósito. Ao invés de um veículo sair do depósito, visitar o nó *i*, retornar ao depósito, para só em seguida visitar o nó *j*, e retornar mais uma vez ao depósito, o ganho sugere que é mais econômico o veículo sair do depósito, visitar o nó *i*, seguir diretamente para o nó *j*, e só então retornar ao depósito. Ou seja, o ganho de cada arco representa quantas unidades de custo podem ser "economizadas" caso o veículo deixe de retornar ao depósito entre as visitas aos nós em questão.

Entretanto, podem existir diversas restrições, como a capacidade dos veículos, o tempo total de percurso, etc. Sendo assim, para que estas restrições sejam levadas em conta, o algoritmo deve funcionar da seguinte forma:

1- Selecione o arco *ij* de maior ganho da lista
2- Se ambos *i* e *j* ainda não pertencerem a nenhuma rota:
    a. Se a rota 1-i-j-1 for viável
        i. Inclua a rota 1-i-j-1
        ii. Tome o próximo arco da lista e reinicie o algoritmo
    b. Se a rota 1-i-j-1 não for viável
        i. Tome o próximo arco da lista e reinicie o algoritmo
3- Se somente *i* ou somente *j* pertencerem a alguma rota
    a. Se este nó for extremo desta rota
        i. Agregue o arco *ij* a esta rota se ela permanecer factível
        ii. Tome o próximo arco da lista e reinicie o algoritmo
    b. Se este nó não for extremo desta rota
        i. Tome o próximo arco da lista e reinicie o algoritmo
4- Se ambos *i* e *j* pertencerem a alguma rota
    a. Se *i* e *j* pertencerem a rotas diferentes e forem extremos (extremo é um nó adjacente ao depósito em uma dada rota)
        i. Una as duas rotas se a rota obtida permanecer factível
        ii. Tome o próximo arco da lista e reinicie o algoritmo
    b. Se *i* ou *j* não forem extremos, ou pertencerem à mesma rota
        i. Tome o próximo arco da lista e reinicie o algoritmo

Sempre que se diz “se a rota permanecer factível”, quer-se dizer que se deve avaliar se as restrições são atendidas. Por exemplo, supondo que exista uma rota 1-4-5-1, e deseja-se avaliar o arco 5-6, conclui-se que o nó 5 já está incluído na rota, mas como é extremo, pode-se agregar o arco 5-6 à rota. Neste ponto, se houver restrição de capacidade, deve-se avaliar se a demanda do nó 6, somada às demandas dos nós 4 e 5, já presentes na rota, pode ser atendida pela capacidade do veículo. Além disso, se houver restrição de tempo total de ciclo, deve-se avaliar se os tempos $t_{14} + t_{45} + t_{56} + t_{61}$, mais eventuais tempos de serviço que existam nestes nós, são menores que o limite máximo t_{max}.

Após todos os arcos serem avaliados em ordem decrescente, se ainda houver clientes que não foram atendidos, estes devem ser incluídos em rotas individuais, da forma 1-k-1, onde *k* é o cliente não atendido.

Uma outra heurística similar, que será utilizada como solução inicial em diversas meta-heurísticas, é o *Nearest Neighbor*. Este modelo já foi apresentado na seção 3.1.1, aplicado ao TSP. Para o caso do VRP, a idéia principal é a mesma, porém deve-se atentar para as restrições do problema. Isso faz com que o funcionamento do modelo seja muito próximo àquele do algoritmo de *Clarke e Wright*, com a diferença de que os arcos não são ordenados estaticamente de acordo com seus ganhos; no *Nearest neighbor*, quem são ordenados são os nós adjacentes ao nó atual, em ordem decrescente de custos para serem alcançados. Sendo assim, para cada nó visitado, uma nova lista dos nós mais próximos é construída. Por “próximos” entenda-se “com os menores custos para serem alcançados”. Será utilizado o termo “distância” entre dois nós para indicar o custo da aresta que os conecta, apesar de sabermos que este custo está relacionado ao tempo necessário para se percorrer tal arco.

De forma mais detalhada, o algoritmo pode ser descrito por:

1- Construa uma lista com os nós adjacentes ao nó atual, ainda não visitados, em ordem decrescente de distância
2- Tome desta lista o nó *i* com o menor custo
    a. Se o nó *i* puder ser incluído na rota atual sem que a solução se torne infactível
        i. Inclua o nó *i* na rota
        ii. Defina o nó *i* como o nó atual e reinicie o algoritmo
    b. Se o nó *i* não puder ser incluído na rota atual

i. Elimine o nó *i* da lista de nós adjacentes ao nó atual e retorne ao passo 2

3- Se a lista estiver vazia, tome o depósito (inclua o nó inicial na rota), e reinicie as restrições de capacidade, tempo de ciclo, etc., com o depósito como nó atual

4- Se não houver mais nós a serem visitados, o algoritmo está terminado

Esta é a forma mais simples e intuitiva do *Nearest Neighbor*. Entretanto, para abordarmos o problema de forma mais eficiente, é preciso fazer algumas alterações neste algoritmo. Se quisermos considerar ainda janelas de tempo, estas alterções se mostram ainda mais necessárias para o bom funcionamento do algoritmo. Basicamente, o que se deve fazer é utilizar um custo ponderado no lugar da mera distância, de forma que ao se selecionar o nó com o menor custo ponderado, esta escolha leve em conta não só o custo de se percorrer o arco, mas também a espera que poderá incorrer caso se chegue a este dado nó antes do início de sua janela de tempo, e a urgência de se atender este nó, devido à proximidade do fim de sua janela de tempo. Conforme sugerido por Pellegrini [9], deve-se dar pesos para cada um destes aspectos. A figura 2 ilustra duas situações distintas: na primeira, há espera, e na segunda não:

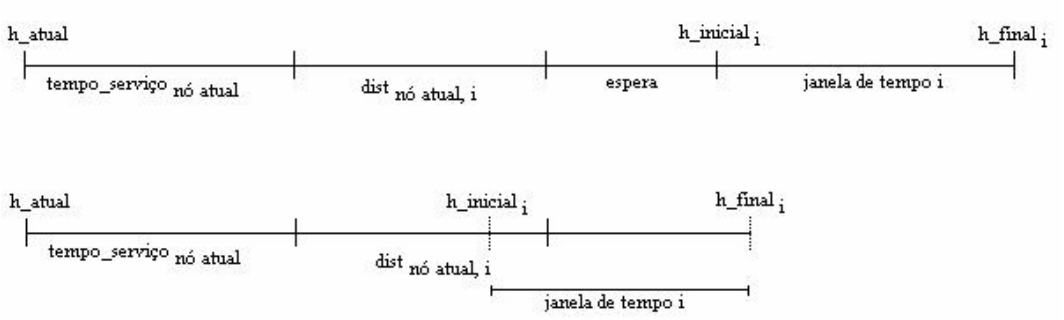

Figura 2 – Intervalos de tempo no ciclo de serviço, para o VRPTW

Sendo assim, a espera relacionada ao nó *i* pode ser caracterizada por: $(h\_inicial_i - (h\_atual + tempo\_serviço_{nó\ atual} + dist_{nó\ atual,\ i}))$, onde $h\_inicial_i$ é o horário de início da janela de tempo do nó *i*, *h_atual* é a hora atual (que corresponde ao início do serviço do nó atual), $tempo\_serviço_{nó\ atual}$ é o tempo que o nó atual leva para realizar seu serviço, e $dist_{nó\ atual,\ i}$ é a distância do nó atual para

o nó *i* (lembrando que, para o VRP, esta distância deve ser entendida como o tempo necessário para se percorrer o arco *nó_atual-i*, conforme descrito anteriormente). Se esta equação for menor que zero, teremos uma situação em que não há espera, ou seja, define-se *espera = 0*. Formalmente, tem-se então:

$$espera = \max\left(h\_inicial_i - \left(h\_atual + tempo\_serviço_{nó\_atual} + dist_{no\_atual,i}\right), 0\right) \quad (2)$$

Dessa forma, quando $(h\_atual + tempo\_serviço_{nó\ atual} + dist_{nó\ atual,\ i})$ for maior que $h\_inicial_i$, tem-se que *espera* = 0, situação esta que corresponde à segunda situação na figura acima.

Define-se urgência como $(h\_final_i - (h\_atual + tempo\_serviço_{nó\ atual} + dist_{nó\ atual,\ i}))$, sendo ela positiva se for possível chegar até o nó *i* antes que sua janela de tempo se feche. Na figura acima, ambas as situações mostram uma urgência positiva, pois o final da janela de tempo *i* é posterior a $(h\_atual + tempo\_serviço_{nó\ atual} + dist_{nó\ atual,\ i})$, ou seja, é possível atender o cliente *i* dentro da janela de tempo.

Dessa forma, para o algoritmo de vizinho mais próximo (*Nearest Neighbor*), deve-se procurar pelo nó que minimize um certo valor *x = a*distância + b*espera + c*urgência*, onde *a, b* e *c* são parâmetros, e ainda que apresente *urgência > 0*, que ainda não tenha sido visitado e que não viole quaisquer outras restrições do problema, como capacidade, tempo de ciclo, etc. Esta forma de implementação do algoritmo será o modelo de *Nearest Neighbor* que será adotado nesta dissertação.

Veremos agora o funcionamento de um algoritmo aqui proposto que implementa as funcionalidades do *Nearest Neighbor* aplicado ao VRP com janelas de tempo, conforme descrito acima. Os *scripts* foram elaborados com o auxílio do software *MatLab 6.0 v12*.

Como dados de entrada, precisamos ter as características físicas do problema, bem como os dados das janelas de tempo e capacidade de carga dos veículos. Sendo assim, usaremos como argumentos: uma matriz N x N contendo todos os custos C_{ij} entre todos os pares de nós do problema (matriz *dist)*; dois vetores 1 x N contendo os valores iniciais b_i e finais e_i das janelas de tempo de cada um dos *N* nós (vetores *h_inicial* e *h_final*); um vetor 1 x N com as demandas d_i de cada nó, lembrando que a demanda do nó 1, que convencionamos ser o

depósito, deve ser igual a zero (vetor *demanda*); uma variável com o valor da capacidade máxima de carga dos veículos, considerados todos iguais (*cap*); finalmente, um vetor 1 x N contendo os tempos de serviço s_i de cada nó, incluindo do depósito (vetor *tempo_serv*). Estes dados são suficientes para identificar completamente o problema. A saída do algoritmo, por sua vez, será composto de três partes: o vetor *caminho*, contendo a seqüência de nós do melhor caminho encontrado; a variável *comp*, com o custo total de tal caminho (comprimento total do caminho, que conforme mencionado acima, é equivalente ao tempo total que um veículo precisará rodar para percorrer o caminho encontrado) e a variável *carros*, com o número de veículos utilizados na solução encontrada.

Cabe ressaltar aqui que a variável *comp* representa o tempo total em trânsito, que é diretamente proporcional à distância total percorrida pelos veículos; se dobrarmos o valor de *comp*, ou seja, se dobrarmos o tempo de trânsito, também seria dobrada a distância total percorrida, dado que nossos veículos são considerados como tendo uma velocidade média constante. O valor de *comp* poderia indicar o custo com combustível, por exemplo, ou a quilometragem total dos carros. O problema também pode ser visto de outra forma: se somarmos ao valor de *comp* os tempos de serviço de todos os nós, teremos o tempo total necessário para que as rotas sejam percorridas. Isso pode indicar, por exemplo, o número de horas trabalhadas, que precisarão ser pagas aos motoristas e ajudantes. Dessa forma, a saída do algoritmo nos possibilita compreender o problema de diversas formas. Entretanto, o que se deseja minimizar juntamente com o número de veículos utilizados é o comprimento total das rotas, que corresponde ao somatório dos custos C_{ij} de cada aresta percorrida, desconsiderando os tempos de serviço de cada nó. A isso corresponde a variável *comp*.

Dito isso, podemos passar para o funcionamento do algoritmo de implementação do *Nearest Neighbor*, que é utilizado no cálculo de uma solução factível inicial para o VRPTW.

As adaptações no *Nearest Neighbor* tradicional que foram propostas nesta seção, visando deixá-lo apto a considerar as janelas de tempo, também foram implementadas, representando uma grande melhoria no seu desempenho. Segundo estas melhorias, não se consideram as meras distâncias entre os nós como critério de construção do caminho, mas leva-se em conta a urgência e a espera de forma ponderada, como veremos abaixo. A seguir, segue o algoritmo:

```
function [caminho,comp,carros] = nearest_neighbor(dist,h_inicial,h_final,demanda,cap,tempo_serv)
%caminho sera o conjunto de rotas encontrado, comp sera o comprimento total destas, e carros o numero de rotas

h_atual = 0; %hora atual
cap_atual = 0; %capacidade utilizada pelo carro atual
caminho = [1]; %caminho composto de 1 ou mais rotas
a = 0.6;
b = 0.2;
c = 0.2;
indice = 1; %aponta para o ultimo no visitado
flags = zeros(1,length(demanda)); %indica quais nos ja foram visitados
comp = 0; %comprimento inicial zero

while min(flags)==0&indice<=2*length(demanda) %ainda existem nos nao visitados
   no_atual = caminho(indice);
   distancias = dist(no_atual,:); %vetor de origem no no_atual
   for j = 1:length(distancias)
      espera(j) = max(h_inicial(j)-h_atual-tempo_serv(no_atual)-distancias(j),0);
      urgencia(j) = h_final(j)-h_atual-tempo_serv(no_atual)-distancias(j);
      distancias(j) = a*distancias(j)+b*espera(j)+c*urgencia(j); %distancias possui as distancias ponderadas
   end
   carga = demanda + cap_atual;
   candidato = no_atual;
   dist_candidato = 1000; %inicializaçao com um numero grande
   for j = 1:length(distancias)
      if (j~=no_atual)&(~ismember(j,caminho))&(carga(j)<=cap)&(urgencia(j)>=0)&(distancias(j)<dist_candidato)
         candidato = j;
         dist_candidato = distancias(j);
      end
   end
   if candidato==no_atual %caso nao seja encontrado nenhum no mais proximo, deve-se voltar ao deposito
      h_atual = 0;
      cap_atual = 0;
      candidato = 1;
   else
      h_atual = h_atual + max(dist(no_atual,candidato)+tempo_serv(no_atual),h_inicial(candidato)-h_atual);
      cap_atual = cap_atual + demanda(candidato);
   end
   caminho = [caminho candidato]; %adiciona-se o no selecionado ao caminho
   comp = comp + dist(no_atual,candidato); %contabiliza-se a distancia percorrida
   indice = indice + 1;
   flags(candidato)=1; %sinaliza que o no selecionado ja foi visitado
end
caminho = [caminho 1]; %adiciona-se o retorno ao deposito
comp = comp + dist(candidato,1); %contabiliza-se o deslocamento final para o deposito
carros = 0;
for j = 1:length(caminho)
   if caminho(j)==1
      carros = carros + 1;
   end
end
carros = carros-1;
```

Quadro 1 – Função *nearest_neighbor*

Como descrito anteriormente, os argumentos desta função são somente os dados físicos do problema. Basicamente, são definidos: o vetor *caminho*, que inicialmente possui somente o depósito (nó 1), e contadores *h_atual* e *cap_atual* que serão atualizados a cada nó que for acrescentado ao caminho. Um vetor *flags* indica quais os nós que já foram visitados, e uma variável *índice* indica qual foi o último nó visitado.

Considerando o nó atual como referência, deseja-se calcular a distância ponderada deste para todos os outros nós. Isto é feito calculando-se a urgência e a espera para cada um dos nós. Feito isso, calcula-se um vetor *distancias* que possui em cada posição o valor ponderado entre a urgência, a espera e a distância propriamente dita, extraída da matriz de distâncias. Esta ponderação é feita com o auxílio de três parâmetros *a, b* e *c*, conforme descrito anteriormente. A implementação deste processo é feita através do primeiro *while*. Inicialmente, verifica-se se ainda existem nós não visitados, através de uma análise do vetor *flags*, inicialmente composto de zeros. Para cada nó visitado, um elemento correspondente no vetor *flags* é alterado para 1, até que todo o vetor seja composto de 1, indicando que todos os nós foram visitados. Além disso, verifica-se o *índice*, que indica a posição atual no vetor *caminho*, ou seja, o comprimento do caminho encontrado até o momento. Este *índice* não pode ser maior que duas vezes o número de nós, ou isto indicaria algum erro de execução no algoritmo.

Feito isso, define-se como nó atual o último nó do vetor *caminho*, e para este nó são tomadas as distâncias para todos os outros nós. Com base nestas distâncias, bem como nos dados do problema, pode-se calcular a espera e a urgência de cada nó com relação ao nó atual. Fazendo a ponderação com os parâmetros *a, b* e *c*, chega-se ao valor ponderado das distâncias do nó atual para todos os outros nós, e estas informações ficam armazenadas no vetor *distâncias*.

O algoritmo procura em seguida pelo menor valor deste vetor *distancias*, e define o nó referente a este menor valor como *candidato* a ser incluído no caminho, desde que este nó ainda não seja membro do caminho atual e não infrinja as restrições de capacidade e de janela de tempo. Se não houver nó que respeite estas condições, define-se como candidato o nó 1, significando que o veículo não possui opções a não ser o retorno ao depósito. Neste caso, os contadores de tempo e carga são zerados. Se entretanto houver nó que atenda a estas condições, a atualização do tempo e da carga é feita somando-se a *h_atual* o tempo de serviço do nó atual, o tempo de trânsito até o nó candidato, e eventuais esperas que possam incorrer, e somando-se à *cap_atual* a demanda do nó candidato. O nó *candidato* pode então ser adicionado ao final do vetor *caminho*, e as variáveis *comp, índice* e o vetor *flags* podem ser atualizados.

Para concluir a função, inclui-se no caminho o último retorno ao depósito, e soma-se esta distância percorrida ao comprimento total do caminho. O último *for*

do algoritmo é responsável por calcular o número de veículos utilizados no caminho encontrado. Este valor é uma das respostas da função, ao lado do caminho propriamente dito e de seu comprimento. Este *for* simplesmente verifica o número de ocorrências do nó 1 no caminho, indicando o número de retornos ao depósito. Isso conclui a execução da função *nearest_neighbor*. Será visto adiante que uma outra função auxiliar, chamada *simula_formiga*, fará uso de técnicas semelhantes para calcular a atratividade dos nós. A compreensão do *nearest_neighbor* se mostra, então, fundamental para o entendimento do restante dos algoritmos.