Universidade Federal de Pernambuco

Graduação em Ciência da Computação

Centro de Informática



2011.2

Análise comparativa de algoritmos de otimização para o problema de Roteamento de Veículos (VRP)

**Trabalho de Graduação**

**Aluno** João Carlos Procópio Florêncio {jcpf@cin.ufpe.br}

**Orientador** Tsang Ing Ren {tir@cin.ufpe.br}

Universidade Federal de Pernambuco

Centro de Informática

João Carlos Procópio Florêncio

Análise comparativa de algoritmos de otimização para o problema de Roteamento de Veículos (VRP)

*This work is an approach to a known Combinatorial Optimization problem that have many practical applications to logistics and Operation Research in general. The main objective of this work is to present the state of art in relation to Combinatorial Optimization techniques used to solve the VRP and its variations. This study must include an implementation of some algorithms to solve VRP and with its results make a comparative analisys when submitted to the main benchmarks available in the field.*

*Este trabalho de graduação deve abordar um problema conhecido de Otimização Combinatorial que tem muitos fins práticos na área de logística e Operation Research. O objetivo deste trabalho é, então, apresentar o estado da arte no que se diz respeito às técnicas de Otimização Combinatorial utilizadas para resolver o VRP e suas variações. Este estudo deve incluir a implementação dos alguns algoritmos para resolver o VRP e em cima deles fazer uma análise comparativa dos resultados quando submetidos aos principais benchmarks disponíveis da área.*

Aos meus pais, Clovis Pires Florêncio e Maria Aparecida Procópio Florêncio.

Agradecimentos

Neste período de conclusão do curso, este trabalho vem selar o fim de mais um ciclo importante da formação para qualquer estudante de Universidade. Agradeço a Deus, ao apoio incondicional dos meus pais em todo meu período de minha formação e à ajuda indispensável do orientador deste meu trabalho, Tsang Ing Ren. Agradeço também aos colegas de faculdade que juntos formaram uma sinergia de trabalho para todos concluirmos esse curso com êxito!

Índice de Tabelas

[Tabela 1 - Custos de deslocamento 29](#_Toc311539084)

[Tabela 2 - Demanda dos Clientes 30](#_Toc311539085)

[Tabela 3 - Custos de junção de rotas 30](#_Toc311539086)

[Tabela 4 - Melhores resultados encontrados para a classe de problemas R1 51](#_Toc311539087)

[Tabela 5 - Melhores resultados encontrados para a classe de problemas R2 52](#_Toc311539088)

[Tabela 6 - Melhores resultados encontrados para a classe de problemas C1 52](#_Toc311539089)

[Tabela 7 - Melhores resultados encontrados para a classe de problemas C2 52](#_Toc311539090)

[Tabela 8 - Melhores resultados encontrados para a classe de problemas RC1 53](#_Toc311539091)

[Tabela 9 - Melhores resultados encontrados para a classe de problemas RC2 53](#_Toc311539092)

Índice de Figuras

[Figura 1 - Caminhando pelo espaço de soluções 13](#_Toc311539127)

[Figura 2 - Espaço de soluções com múltiplos mínimos locais 14](#_Toc311539128)

[Figura 3 - Região viável em um PL 16](#_Toc311539129)

[Figura 4 - Curva de Pareto 17](#_Toc311539130)

[Figura 5 - Problema do VRP 17](#_Toc311539131)

[Figura 6 - Decomposição do espaço de soluções pelo B&B 21](#_Toc311539132)

[Figura 7 - Amadurecimento por otimalidade 22](#_Toc311539133)

[Figura 8 - Amadurecimento por Limitante 23](#_Toc311539134)

[Figura 9 - Amadurecimento por Inviabilidade 23](#_Toc311539135)

[Figura 10 - Processo de Cutting-plane 25](#_Toc311539136)

[Figura 11 - Junção de duas rotas 28](#_Toc311539137)

[Figura 12 - Agrupamento dos clientes 34](#_Toc311539138)

[Figura 13 - Conceito de Colônia de Formigas 38](#_Toc311539139)

[Figura 14 - Algoritmo ACS-TIME 43](#_Toc311539140)

[Figura 15 - Algoritmo ACS-VEI 44](#_Toc311539141)

[Figura 16 - Replicação do depósito 45](#_Toc311539142)

[Figura 17 - Algoritmo new\_active\_ant 47](#_Toc311539143)

[Figura 18 - Operação de CROSS-EXCHANGE 49](#_Toc311539144)

[Figura 19 - Instância R101, com o algoritmo NN\_Gambardella alcançou o custo de 2015,21. 54](#_Toc311539145)

[Figura 20 - Instância R101, com o algoritmo CROSS-EXCHANGE alcançou o custo de 1949,92. 55](#_Toc311539146)

[Figura 21 - Instância R101, com o algoritmo MACS-VRPTW alcançou o custo de 1172,03. 56](#_Toc311539147)

[Figura 22 - Instância C101 (reduzido para 25), com o algoritmo NN\_Gambardella alcançou o custo de 257.92. 57](#_Toc311539148)

[Figura 23 - Instância C101 (reduzido para 25), com o algoritmo CROSS-EXCHANGE alcançou o custo de 196.87. 58](#_Toc311539149)

[Figura 24 - Instância C101 (reduzido para 25), com o algoritmo MACS-VRPTW alcançou o custo de 191,81. 59](#_Toc311539150)

Conteúdo

[1 Introdução 9](#_Toc311539760)

[1.1 Contexto e Motivação 9](#_Toc311539761)

[1.2 Problemas de Otimização Matemática 11](#_Toc311539762)

[2 Descrição do Problema 16](#_Toc311539763)

[3 Estado da Arte 18](#_Toc311539764)

[3.1 Histórico 19](#_Toc311539765)

[3.2 Algoritmos Exatos 20](#_Toc311539766)

[3.2.1 Branch-and-Bound 20](#_Toc311539767)

[3.2.2 Branch-and-Cut 23](#_Toc311539768)

[3.3 Heurísticas 24](#_Toc311539769)

[3.3.1 Métodos Contrutivos 25](#_Toc311539770)

[3.3.2 Algoritmos de Duas Fases 31](#_Toc311539771)

[3.4 Meta-heurísticas 33](#_Toc311539772)

[3.4.1 Simulated Annealing 34](#_Toc311539773)

[3.4.2 Tabu-Search 35](#_Toc311539774)

[3.4.3 Algoritmos Genéticos 36](#_Toc311539775)

[4 Sistema de Colônias de Formigas (ACS) 37](#_Toc311539776)

[5 MACS-VRPTW 40](#_Toc311539777)

[5.1 Colônias ACS-TIME e ACS-VEI 41](#_Toc311539778)

[5.2 Modelo da Solução 44](#_Toc311539779)

[5.3 Procedimento da Solução construtiva 44](#_Toc311539780)

[5.3.1 Procedimento da Inserção 47](#_Toc311539781)

[5.3.2 Procedimento de Busca Local: CROSS-EXCHANGE 48](#_Toc311539782)

[5.4 Resultados 49](#_Toc311539783)

[6 Conclusão 59](#_Toc311539784)

[7 Referências 60](#_Toc311539785)

# Introdução

## Contexto e Motivação

Sabe-se que no contexto de mundo atual, toda empresa que deseja sobreviver em um mercado competitivo deve procurar novas formas de se diferenciar, seja tanto no aspecto do atendimento ao cliente, na melhoria da qualidade do produto ou na minimização dos custos de produção. Em qualquer aspecto que se analise, a forma que vem se mostrando mais eficiente, em termos de prover maiores ganhos e de se destacar em relação à concorrência, é a de aplicação de novas tecnologias que venham a atuar diretamente nos pontos citados.

Nesse contexto, este trabalho pretende abordar o problema da otimização de logística no sentido de se minimizar os custos com transporte advindo das necessidades da empresa em deslocar para atender seus clientes. Problema este que ainda está longe de se alcançar soluções ótimas e, por isso, a motivação do trabalho em enxergar o estado da arte no que se refere ao problema proposto. A melhoria nesse sentido além trazer economia de custos, também traz melhoria no serviço de atendimento ao cliente.

Empresas que tem que se deslocar por uma cidade diariamente para atender seus clientes sofrem do “Problema do Roteamento de Veículos” conhecido por sua sigla em inglês VRP (Vehicle Routing Problem) e que está mais bem descrito na seção de “Descrição do Problema”. Quando a demanda de visitas requisitadas pelos clientes aumenta neste tipo de empresa é comum vermos aplicações de técnicas pouco eficientes e que geralmente, além de não gerar rotas de baixo custo, são passíveis de muitos erros técnicos como atendimento fora do horário disponível do cliente, prazos de atendimento previstos em contrato extrapolados e ordem de preferência de atendimentos entre clientes não seguindo os reais critérios da empresa.

Com a utilização das técnicas apropriadas é possível eliminar os riscos de erros na solução e reduzir custos operacionais relativos ao transporte muitas vezes superiores a 20%[4] quando comparado com um solução dada por um indivíduo especialista. E por que este problema trata-se de um desafio para a área da computação?

Muitas empresas alocam pessoas comuns à resolução de um problema de roteamento de veículos no dia-a-dia. Isso pode levar alguns a pensar que se trata de um problema simples e sem grandes complicações. No entanto, apesar de não parecer desafiador, ele é, na verdade, é um problema de grande complexidade computacional, estando na classe de problemas não só de NP-Completo como também de NP-Difícil [4] e, portanto, com custo de tempo exponencial para encontrar a solução ótima. Sem a utilização de técnicas de otimização combinatória, mesmo um computador pode não chegar a uma solução satisfatória em tempo razoável.

Tendo isso em vista, iremos abordar neste trabalho o assunto primeiramente descrevendo com detalhes no capítulo 2 todos os aspectos que o problema envolve. Entendido o problema, partiremos para entender como o VRP ganhou destaque nas pesquisas científicas e quais as principais abordagens que o problema vem sendo atacado, sempre apresentando as vantagens e desvantagens de cada tipo. Para irmos mais a fundo no problema selecionamos a técnica de MACS-VRPTW, uma das técnicas mais bem sucedidas para VRP utilizando a analogia de otimização de Colônia de formigas, para mostrar com nível superior de detalhes as metodologias utilizadas na resolução. Junto com a técnica mostraremos os resultados que conseguimos com a implementação da técnica. Ao fim apresentaremos as conclusões sobre tudo que foi visto neste trabalho.

## Problemas de Otimização Matemática

Todo problema de otimização caracteriza-se basicamente por ter uma função objetivo, que pode se chamar função custo quando se deseja minimizá-la ou função utilidade quando se deseja maximizá-la, e um conjunto de restrições que delimitam o espaço de soluções viáveis, ou seja, a região onde estão as soluções que podem ser aceitas. A função objetivo contém um conjunto de variáveis as quais valores devem ser atribuídos de forma sistemática de modo a caminhar pelo espaço de busca e encontrar aquela que otimize o resultado que se busca, no caso de um problema maximização encontrar o maior valor possível enquanto em um de minimização encontrar o menor valor. Em ambos os casos a solução deve satisfazer o conjunto de restrições impostas para ser aceita. A formatação de um problema de otimização matematicamente se dá da seguinte forma:

Onde é o conjunto de soluções viáveis que a função pode gerar.

Podemos interpretar o espaço de soluções como sendo um subconjunto do espaço euclidiano . Cada variável é uma dimensão do espaço. Para uma função com duas variáveis é possível formar em um espaço bidimensional e com a adição de uma terceira dimensão para o resultado da função com x e y como entrada é possível observar o comportamento da função à medida que o x e o y da função sofrem variação. Na figura 1 abaixo mostramos um gráfico bidimensional em que a variação do valor da função provoca alterações na escala de cinza. Para valores maiores obtemos um cinza mais escuro e para valores menores um cinza mais claro. Com isso, podemos observar o espaço de soluções de forma panorâmica no espaço bidimensional e vemos que à medida que se aproxima do centro a função gera valores menores. Nela é possível perceber também que as marcas em forma de “x”, que representam as soluções encontradas, variam em direção ao mínimo local que se encontra no meio do espaço de busca. A heurística utilizada para a busca da solução ótima, neste caso, foi a de procurar soluções vizinhas que minimizassem o custo da função da mesma forma que uma esfera que rola sobre um plano inclinado se estabiliza no momento que alcança um “vale” que seria o mínimo local da função. No entanto essa heurística nem sempre é a mais adequada, como veremos mais a frente.

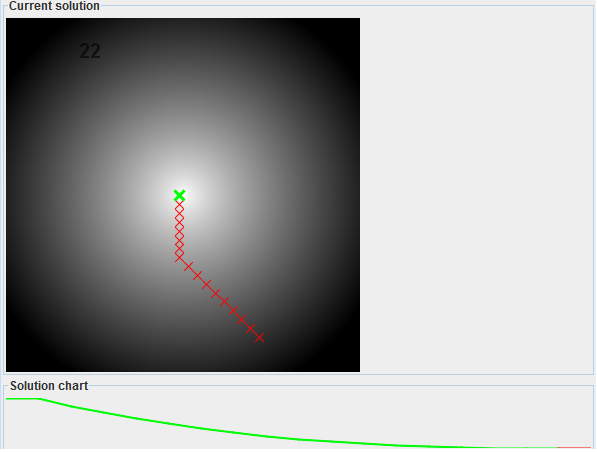


Figura 1 - Caminhando pelo espaço de soluções

Otimização Combinatória

A heurística de busca mostrada na seção anterior funciona apenas para um número limitado de casos. Funções com espaços de busca mais complexos e com múltiplas restrições dificultam o processo de busca e, por isso, a estratégia com que esse espaço vai ser varrido tem de ser feita de forma mais elaborada. Na figura 2 mostramos no espaço euclidiano uma função objetivo onde a simples busca por um mínimo local não funcionaria.

Em alguns casos a natureza do problema proporciona um atalho para a solução ótima, e quando isso é possível, encontra-se a solução de forma objetiva em tempo polinomial. Para os casos onde esse atalho não é facilmente identificável, a busca por força bruta pela melhor solução se torna necessária. Para esse subconjunto de problemas, onde a aplicação de força bruta é necessária, podemos classificá-los como problemas **Otimização Combinatória**. É nesta classe de problema que se encontra o VRP (*Vehicle Routing Problem*), o problema que será tratado nesse trabalho.

Para problemas desse tipo, muitos algoritmos foram sugeridos com o objetivo de determinar uma estratégia de busca inteligente de forma a percorrer o espaço de soluções da forma mais eficiente possível. Esses algoritmos são aplicação geral, pois se adaptam a qualquer problema que envolva busca em um espaço de soluções. Os mais conhecidos são: algoritmos genéticos, Tabu Search, Simulated Annealing, Particle Swarm, etc.

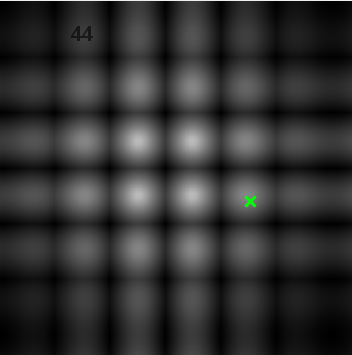


Figura 2 - Espaço de soluções com múltiplos mínimos locais

Programação Linear

Em casos onde a função objetivo é linear e o conjunto de restrições fecha o espaço de busca em forma de um politopo definimos esse tipo de problema de otimização como de programação linear. O problema pode ser descrito na forma canônica da seguinte forma:

Onde x representa as variáveis a serem determinadas, C e b são vetores de coeficientes conhecidos e A é uma matriz de coeficientes. A primeira linha descreve a função objetivo que se deseja maximizar ou minimizar, na segunda linha é o conjunto de restrições que determina o politopo convexo onde as soluções viáveis se encontram e na terceira é uma regra geral que determina que todas as variáveis da função devem ser sempre positivas. Esta última restrição é determinada porque todos os problemas de programação linear não fazem sentido para valores negativos e por isso essa restrição já é pré-determinada.

Em programação linear, mesmo sem nenhum atalho gerado pela natureza do problema é possível encontrar na maioria dos casos a solução ótima de forma relativamente rápida. Isso deve ao fato da função ser linear e não possuir múltiplos mínimos locais. O algoritmo mais bem sucedido para resolver programação linear é o Simplex proposto por Dantzig em 1947. Na solução proposta por ele, o espaço é varrido somente pelos vértices do politopo até que se encontre um vértice cuja solução seja superior em relação a todos os vértices vizinhos.

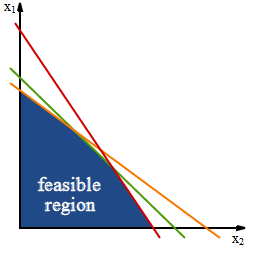


Figura 3 - Região viável em um PL

Otimização Multi-objetivo

Apesar de um problema típico de otimização conter somente uma função objetivo existem casos em que o problema é composto por mais de um objetivo e estes são conflitantes entre si. Quando isso acontece, ele é chamado de **Otimização Multi-objetivo**. No caso de uma definição de um projeto estrutural, por exemplo, é possível que o engenheiro queira por um lado uma estrutura rígida e da mesma forma querer que ela seja leve. Sabe-se, no entanto, que estes dois objetivos são conflitantes. Caminhos que levem a uma estrutura mais rígida provavelmente vão distanciar a estrutura do objetivo de ela ser leve e vice-versa. O conjunto de trade-offs do projeto que não possam ser melhorados sem que um critério quebre outro critério é chamado de conjunto de Pareto. E ele pode ser descrito em uma curva que gera a chamada *Fronteira de Pareto*. Uma solução é conhecida como “Pareto Optimal” ou “Pareto Efficient” caso ela não seja dominada por nenhum dos objetivos, ficando, assim, em um meio termo. A figura abaixo mostra uma curva de pareto.

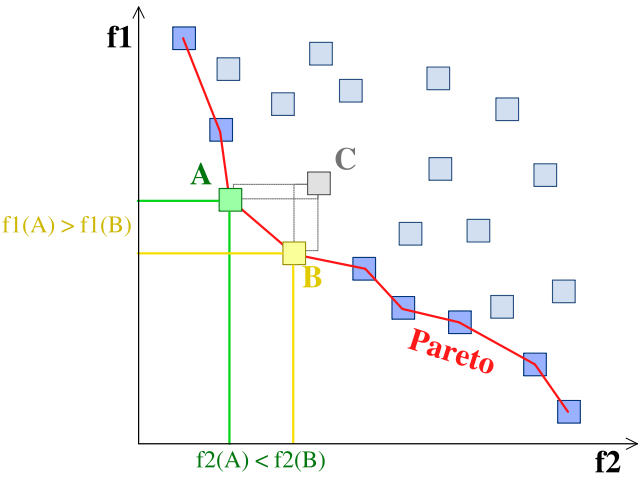


Figura 4 - Curva de Pareto

# Descrição do Problema

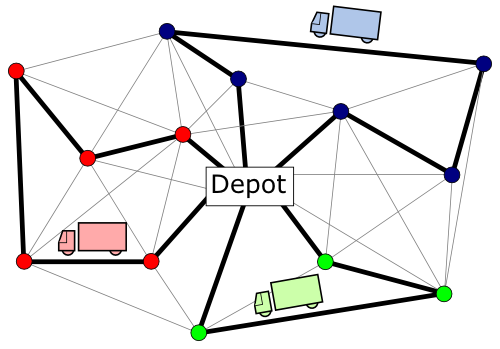


Figura 5 - Problema do VRP

O problema de roteamento de veículos (em inglês: *Vehicle Routing Problem - VRP*), assim como o próprio nome já diz, consiste em buscar a melhor rota que uma dada frota de veículos deve percorrer de modo a atender/visitar um conjunto de clientes distribuídos geograficamente em uma região. Quando se fala em "melhor rota", podemos entender como aquela que deve minimizar os custos envolvidos no transporte sem, contudo, violar as restrições impostas pelo problema. Estas restrições variam bastante dependendo do contexto em que se encontra. Podemos citar como exemplo de restrição: a janelas de tempo disponíveis de atendimento a um determinado cliente. Pois caso algum veículo seja alocado para atender um cliente em um horário que ele não esteja disponível, a empresa, ao invés de economizar, terá prejuízo ao realizar uma viagem sem ter condições de prestar o atendimento previsto. É a partir de restrições como esta que surgem as variações do VRP. Abaixo podemos citar as principais, tais como roteamento de veículos com:

* **Janelas de tempo** (VRP with Time Window) – Limita os veículos a visitarem os clientes dentro de suas respectivas janelas de tempo que devem representar o horário de funcionamento do estabelecimento. Da mesma forma, os próprios veículos podem ter janelas de tempo que os impeça de atender clientes em determinados de horários, estas janelas representariam o seu horário de trabalho, por exemplo: das 8:00 às 12:00, uma pausa para o almoço e das 14:00 às 18:00.
* **Capacidade de carga** (Capacited VRP) – Limita o veículo em sua capacidade de armazenar as cargas demandadas pelo cliente. Dessa forma, os veículos ficam restritos a atender apenas um conjunto clientes no qual a soma das cargas demandadas não exceda a sua capacidade de armazenamento. Neste tipo de problema, é comum o veículo volte recorrentemente a fim empresa para fazer novos carregamentos e, consequentemente, poder atender mais clientes.
* **Múltiplos depósitos** (Multiple Depots VRP) – É o caso onde existem múltiplos depósitos distribuídos pela cidade e deseja-se minimizar os custos encontrando a melhor distribuição dos veículos da empresa pelos depósitos disponíveis. Esse tipo de consideração é importante ser levado em conta quando os clientes não estão agrupados ao redor dos depósitos. Caso contrário, seria melhor resolver o problema solucionando um VRP para cada depósito.
* **Atendimento em múltiplos dias** (Period VRP) – Quando a quantidade de clientes a serem atendidos excede o espaço de tempo de um dia, o agendamento dos atendimentos passam a ser feitos para múltiplos dias. Essa nova consideração implica algumas conseqüências. Quando se transfere uma atividade de um dia para o outro, essa mudança pode gerar custos relacionados a multas de contratos, interrupção de atividades e insatisfação do cliente que se não forem considerados impactam no resultado final da solução.
* **Entrega e recolhimento** (Pick-up and Delivery VRP) – Existem casos em que o problema não se limita a entregar pedidos aos clientes, além disso, o veículo pode também realizar recolhimentos que deve impactar na sua capacidade de armazenamento. Por isso, o algoritmo deve considerar essa nova variável ao calcular sua disponibilidade de armazenamento ao longo da viagem;
* **Roteamento dinâmico** (Dynamic VRP) – É uma restrição existente em quase todo caso real de roteamento de veículos. Quando se despacha um veículo para visitar um conjunto de clientes, podem ocorrer ao longo do dia mudanças que não eram previstas no momento de sua saída. Visitas podem ser canceladas e outras inseridas na rota do veículo. No entanto, a rota não pode ser modificada arbitrariamente, correndo o risco de deixar a solução ineficiente. Para isso, o roteamento dinâmico deve modificar as rotas ao longo do dia sem, contundo, encarecer desnecessariamente a solução.

Muitas outras variações podem ser formuladas ou combinadas entre si de forma que atenda às restrições onde a solução deva ser aplicada. Neste trabalho trataremos da variação mais geral e comum no mundo real, que é a combinação das restrições de janela de tempo (VRPTW) com restrição de cargas (CVRP) para os veículos.

# Estado da Arte

Nesta seção serão tratados tópicos que ajudem o leitor a perceber as abordagens que o problema de roteamento de veículos tem sido tratado ao longo do tempo. Primeiramente, será apresentado um breve histórico do problema, mostrando como o problema surgiu, as primeiras abordagens para resolvê-lo e como evoluiu essa área de pesquisa. A partir daí, serão apresentadas três formas que o problema pode ser atacado: com algoritmos exatos, que são lentos, mas sempre apresentam soluções ótimas, as heurísticas que são rápidas, mas possuem uma eficácia limitada no resultado e as metas-heurísticas que tem eficácia ampla e alcança resultados próximos do ideal em tempo razoável.

## Histórico

O problema de roteamento de veículos é um dos mais estudados em otimização combinatorial. Ele foi introduzido inicialmente por Dantzig e Ramser em 1959 e foi descrito inicialmente em cima de um problema do mundo real que era o de otimizar a entrega de gasolina em diversas “estações de serviço” pré-determinadas. O resultado deste trabalho foi o de desenvolver a primeira formulação do problema com programação matemática e a primeira abordagem algorítmica.

Em 1964 Clarke e Wright propuseram uma heurística gulosa que melhorava os resultados em relação ao algoritmo de Dantzig-ramser. A partir daí, vários novos artigos foram publicados abordando o problema de várias outras formas e encontrando soluções otimizadas para vários tipos de VRP com diferentes tamanhos de instâncias.

Hoje, o problema roteamento de veículos é atacado principalmente em três frentes: algoritmos exatos, heurísticas e meta-heurísticas. Em relação aos algoritmos extatos, por buscarem apenas a solução ótima, ele têm dificuldades em resolver problemas com instâncias maiores. As maiores instância que podem ser resolvidas, hoje, com algoritmos exatos em tempo razoável são de até 50 clientes. Instâncias maiores podem ser resolvidas otimamente apenas em casos particulares. Já resolução do problema com abordagem heurística consegue resultados bastante otimizados quando aplicados em problemas de naturezas específicas, de modo que explorarando essas características venha a encontrar o caminho da solução boa mais rapidamente. No entanto, heurísticas não podem ser usadas amplamente e aplicadas para diferentes contextos de problemas. Normalmente são usadas em conjunto com outras técnicas de heurísticas e principalmente meta-heurísticas com o intuito de realizar um busca local. Em relação à meta-heurística, ela tem se apresentado como a abordagem com maior grau de sucesso. Suas técnicas baseadas na evolução da solução através da conjugação de busca local com busca global têm alcançado bons resultados em tempos relativamente curtos. Embora não garanta a solução ótima, a aplicação dessa classe de algoritmos têm produzido resultados satisfatórios para o mercado e hoje é usado vastamente em muitas aplicações do mundo real.

## Algoritmos Exatos

Nesta seção serão apresentados os principais algoritmos exatos usados para resolver o VRP. Quando se fala em algoritmos exatos entende-se como as técnicas de otimização que dão como resultado a solução ótima do problema.

### Branch-and-Bound

É uma técnica que resolve problemas de otimização devolvendo uma solução ótima. A função objetivo é . A idéia está em gerar uma árvore onde de todas as possíveis soluções estejam nas folhas. No entanto, sabe-se que varrer todas as soluções de forma bruta inviabilizaria grande parte dos problemas já que o espaço de soluções cresce exponencialmente à medida que aumenta a entrada. Para resolver isso, ao longo da varredura ocorrem com “podas” sistemáticas na árvore de forma que evite que o algoritmo percorra regiões da árvore onde é presumível que a solução ótima não esteja.

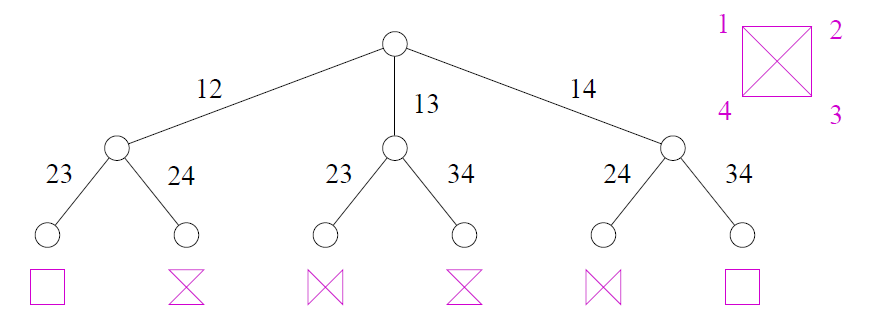


Figura 6 - Decomposição do espaço de soluções pelo B&B

O algoritmo é baseado no paradigma do dividir para conquistar. Dado um conjunto S de possíveis soluções para o problema, ele é dividido em vários subconjuntos de soluções . Busca-se solucionar as instâncias menores para no fim se obter a solução do problema inteiro. No entanto, ao executar o algoritmo em um computador sabe-se que não é possível gerar toda a árvore para começar a varrer o espaço de soluções. Ao invés disso, enumera-se a árvore de forma implícita. Isso é possível graças ao uso de limitantes que permitem identificar os possíveis ramos sem que seja necessário realmente construir a árvore. Cada nó da árvore tem seu limitante superior e seu limitante inferior .

No geral o algoritmo funciona como o próprio nome já diz: são feita operações de *branch* que se entende por ramificar um nó e operações de *bound* que é identificar os limitantes da ramificação e decidir se vai varrer o ramo ou se vai amadurecê-lo, ou seja não ramificá-lo mais, caso idenfique que a solução ótima não se encontra nela. Esse estado, onde sabe-se que o ramo não possui a solução ótima, é também conhecido como um ramo *fruitless*. É possível identificar que um ramo é *fruitless* a partir de três casos:

* **Otimalidade –** Ocorre quando encontra a solução ótima do ramo, ou seja, o valor correspondente ao limitante superior do espaço de soluções. No exemplo abaixo, em é conhecida uma solução de custo 20, e ela é ótima uma vez que o custo de qualquer solução de . Logo deve ser amadurecido.

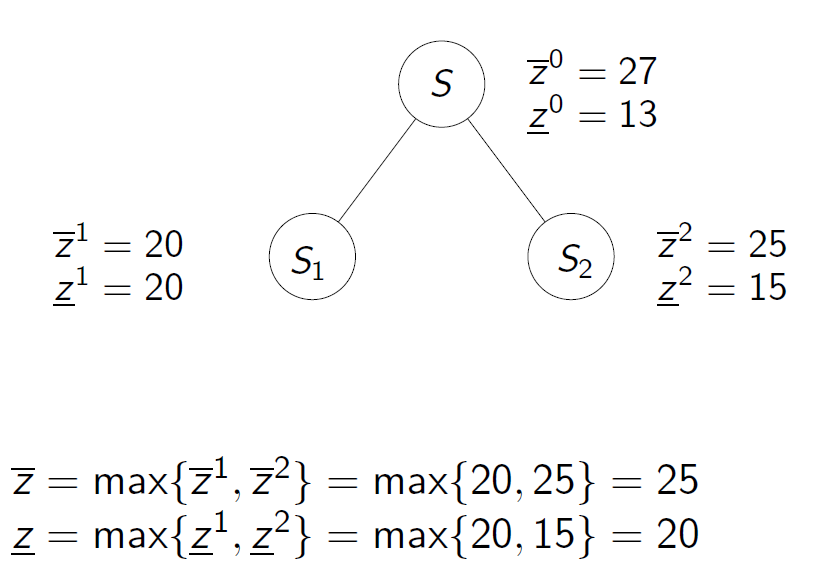
****

Figura 7 - Amadurecimento por otimalidade

* **Limitante –** Ocorre quando o limitante superior é menor do que o limitante inferior de outro ramo. No exemplo abaixo a solução de maior custo em tem custo limitado a 20 (), mas em já é conhecida solução de custo 21 (). Logo deve ser amadurecido.

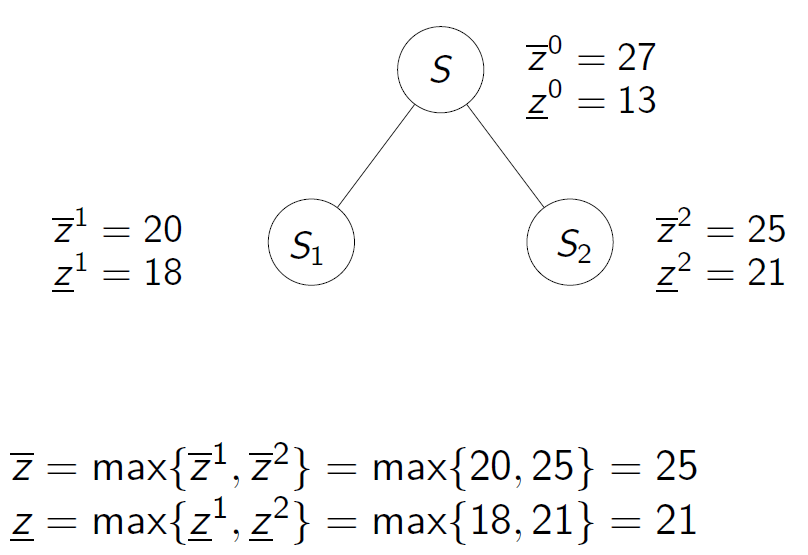


Figura 8 - Amadurecimento por Limitante

* **Inviabilidade –** Quando é possível identificar que um subconjunto de soluções contêm somente soluções inviáveis. No caso abaixo o nó representa o subconjunto de todas as soluções que contêm o item 1 e o item 2. Mas este subconjunto é vazio já que a soma dos pesos destes itens ultrapassa a capacidade da mochila. Logo é amadurecido.

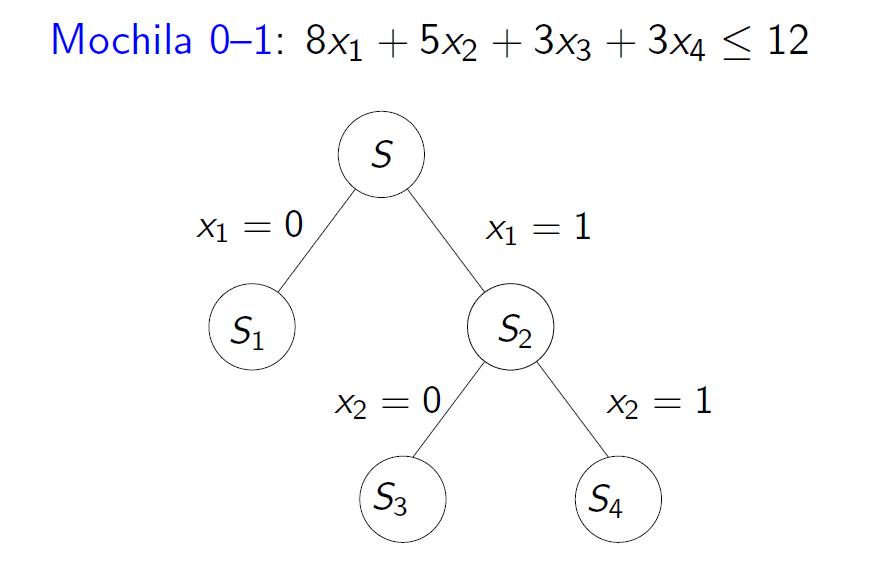


Figura 9 - Amadurecimento por Inviabilidade

Portanto, não é possível amadurecer um ramo i da árvore onde .

Além disso, devem-se considerar as estratégias de B&B (Branch-and-Bound) que devem ser tomadas. Sabe-se que, por exemplo que calcular os limitantes de forma forte, ou seja, conseguir um alto grau de limitação, envolve um certo custo computacional, mas permite amadurecer os ramos mais rapidamente. Em contrapartida, limitantes fracos, apesar de serem calculados mais facilmente, força que a árvore seja explorada mais profundamente percorrendo ramos onde não se encontra a solução ótima. Outra estratégia se refere a quantidade de ramos que um nó pode gerar e se essa decomposição de nós vai ser fixa, gerando um número pré-determinado de ramos, ou variável. E por fim decidir como vai percorrer a árvore, podendo ser por profundidade, largura ou escolhendo o nó com melhor limitante dual.

### Branch-and-Cut

Usa-se Branch-and-Cut para resolver modelos de MIPs (Mixed Integer Program). Este procedimento gerencia uma árvore de busca que consiste de nós. Cada nó representa um Programa Linear (LP) ou subproblema QP para ser processado; isso é, para ser resolvido, para ser verificada sua integralidade, e talvez ser melhor analisado. Nós são chamados ativios se eles ainda não foram processados, ou seja, após um nó ter sido processado, ele se torna inativo. *Branch-and-Cut* processa os nós ativos até que não hajam mais ou algum limite seja alcançado.

Como falado no tópico anterior, um *branch* é a criação de novos nós a partir do nó pai. Tipicamente, um *branch* ocorre quando os limites em uma variável são modificados, com os novos limites permanecendo em vigor para esse novo nó e para qualquer um dos seus descendentes. Por exemplo, se um branch ocorre em uma variável binária, ou seja, com um limitante inferior de 0 e um limitante superior de um, então o resultado vão ser dois novos nós, um nó com o limitante superior modificado de 0, e o outro nó com um limitante inferior modificado de 1. Os dois nós vão, conquentemente, ter domínios de soluções completamente diferentes.

O procedimento de *cut*  é uma restrição adicionada ao modelo. O propósito de adicionar um corte é para limitar o tamanho do domínio da solução para problemas de LP contínuos ou QP representados nos nós, enquanto não elimina soluções inteiras legais. O resultado é, portanto, reduzir o número de *branches* requeridos para resolver o MIP (Mixed Integer Programming). A título de visualização, mostramos uma figura que apresenta um corte feito no domínio de soluções de um LP pelo Branch-and-cut. Sabemos que soluções não-inteira não são aceitas em MIPs. Como mostra na figura o ponto ótimo da relaxação linear (azul) do polígono verde, não possuem coordenadas inteiras. Assim, não é uma solução válida para o problema. A idéia é utilizar retas que “cortem” o ponto fora. Essa reta (ou hiperplano em dimensões maiores), chamada de plano de corte, nada mais é do que uma desigualdade que é satisfeita por todos os pontos viáveis (vermelhos) e não é satisfeita pelo ponto que queremos remover (azul). Às vezes, decidir se uma desigualdade representa um plano de corte é NP-difícil! Na prática, usam-se heurísticas.

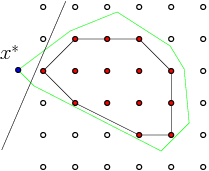


Figura 10 - Processo de Cutting-plane

Dessa forma o algoritmo realiza decomposições do domínio de soluções com o *branch* e em seguida corte nesses sub-domínios a fim de cortar regiões quem não possuam soluções viáveis e inteiras.

## Heurísticas

Essa classe de técnicas procura resolver os problemas de otimização através da observação da natureza do problema. É possível alcançar soluções de boa qualidade caminhando por uma região onde se tenha uma maior densidade de boas soluções baseado na forma que o problema está disposto. Logicamente, essa região não garante que nela esteja incluída a solução ótima. No entanto, varrendo essa região menor é possível em um curto espaço de tempo encontrar uma solução satisfatória.

Porém Heurísticas normalmente apresentam aplicabilidade muito restrita, uma modelagem que serve bem para resolver um tipo de entrada, por exemplo: entradas cujo conjunto de clientes a serem atendidos possui predominantemente janelas de tempo largas ou estão sempre *cluserizados* geograficamente, podem obter péssimos resultados quando submetidos a outros tipos de entrada. Por isso, a modelagem da heurística deve ter como foco a aplicação final do algoritmo estando sempre atento para os tipos de entrada que ela deve resolver.

Em razão dessas particularidades dos algoritmos heurísticos, as técnicas apresentadas nesta seção, diferente das outras, são de aplicação exclusiva do VRP, já que se utiliza das características do VRP determinar os procedimentos que devem ser seguidos para alcançar uma solução. Nas outras técnicas pequenas adaptações permitem que se resolva outros problemas de otimização.

### Métodos Construtivos

Nos métodos construtivos, o algoritmo deve construir gradativamente uma solução viável à medida que observa o custo da solução. Apresentamos nesta seção os algoritmos: Nearest Neigbhor de Gambardella e Savings Method de Clark e Wright.

Nearest Neigbhor Heuristic

Esta seja talvez a heurística mais intuitiva para as pessoas em geral. Sua idéia básica concentra-se no fato de que existe uma probilidade de que escolher o próximo cliente mais próximo para visitar irá gerar uma economia de percurso por minimizando o tempo de deslocamento a cada cliente que se visita. Sabe-se no entanto que essa heurística nem sempre é válida, mas que pode trazer bons resultados dependendo da topologia do problema. Casos em que os clientes apresentem grande disparidade de distância entre seus vizinhos provavelmente seria bem solucionado pela heurística do *Vizinho mais Próximo*.

Existem várias versões desse algoritmo, mas neste trabalho apresentaremos a abordagem de Gambardella de 1999 que usou no seu artigo do MACS-VRPTW para construir uma solução inicial que pudesse ser otimizada pelo seu algoritmo de Colônia de Formigas.

No algoritmo, o veículo começa de um ponto inicial, que no caso do VRP é o depósito, e a partir deste momento inicia sua heurística iterativamente. A cada ponto que o veículo visitava, o algoritmo observaria os vizinhos do ponto atual e em busca do vizinho mais próximo e viável. O cálculo de proximidade não leva em conta somente a distância entre os clientes. Neste algoritmo de Gambardella que leva em consideração a janela de tempo que os clientes podem ser atendidos o cálculo da distância é feito da seguinte forma:

Onde é a distância do ponto i ao j, é o momento de início da janela de tempo escolhida para atender o cliente e é momento que o veículo está pronto para percorrer um novo caminho em direção ao próximo cliente. Vale ressaltar que essa distância só é válida caso o cliente seja viável.

Quando se fala em vizinho viável, entende-se que ele não viole nenhuma restrição imposta pelo problema. As duas restrições mais comuns são de janela de tempo e capacidade do veículo. Caso exista restrição de carga deve-se verificar se a soma total das demandas dos clientes já visitados mais o possível novo cliente excede a capacidade do veículo. Caso exceda ele deve escolher outro cliente para visitar e se não haja mais nenhum cliente que possa ser visitado sem que exceda a capacidade de carga o veículo deve voltar para o depósito. A partir do momento que o veículo volta para o depósito, o algoritmo considera como se estivesse adicionando um novo veículo ao problema, por conta disso, no momento que ele for visitar um novo cliente saindo do depósito sua carga está zerada assim como seu tempo de percurso.

Já se houver restrição de Janela de tempo para os clientes, deve-se observar se existe algum momento que o veículo pode chegar ao cliente dentro de uma janela de tempo. Isso só é possível caso exista pelo menos uma janela que não tenha terminado no momento que ele chegar ao cliente. Por isso, caso chegue um momento que não exista nenhum cliente que seja mais possível visitar por conta de inviabilidade de janela de tempo o veículo deve voltar para o depósito. E da mesma forma que aconteceu com o veículo que voltou por conta do limite de sua capacidade, ele tem a sua carga e tempo de percurso zerado, de tal forma que torna possível a visita a qualquer outro cliente uma vez que no momento zero qualquer janela de tempo deve estar aberta ou deve abrir em algum momento.

Esse processo de escolha do próximo cliente é realizado até que não exista mais nenhum cliente para ser visitado. O número de rotas, então, vai corresponder a quantidade de vezes que o algoritmo saiu do depósito.

Savings: Clark and Wright

Foi o primeiro algoritmo proposto para resolver o problema de roteamento de veículos. Seu conceito básico é gerar economia de custos juntos duas rotas em uma como ilustrado na figura 11, onde o ponto 0 representa o depósito. Essa junção deve atender as restrições do problema, como: janela de tempo e capacidade do veículo.

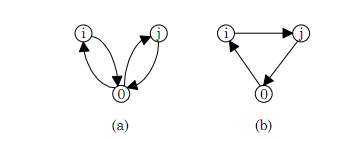


Figura 11 - Junção de duas rotas

Para avaliar a melhoria em relação à economia de custo causada por uma fusão de rotas realizamos os seguintes cálculos. Denotando o custo de transporte entre dois pontos dados i e j por , o custo total de transporte na figura 11 (a) é:

De forma equivalente, o custo de transporte na figura 11(b) é:

Combinando as duas rotas obtemos a economia de custos :

Quanto maior for o mais atrativa essa mudança for, maior a probabilidade dela ser escolhida pra sofrer a mudança.

O primeiro passo para o algoritmo é construir uma rota para cada cliente, ou seja, n veículos serão deslocados para atender n clientes. Logicamente esta é a solução mais simples possível e que ao mesmo tempo já garante se o problema possui uma solução viável ou não. A seguir o processo de fusão de rotas é iniciado de acordo com os cálculos de economia de custos mostrados acima. Todas as possíveis fusões são enumeradas e ordenadas de acordo com o grau de melhoria que elas gerariam no resultado final. Cada fusão é representada por um par de pontos i-j. Do topo da lista ordenada o algoritmo considera um par de pontos por vez. Quando um par de pontos i-j é considerado, duas rotas que visitam i e j são combinadas somente se essa fusão puder ser feita sem deletar a conexão direta estabelecida entre os dois clientes (pontos), e se o total de demanda da rota resultante não exceder a capacidade do veículo. A partir de então existem duas versões que podem ser utilizadas, uma seqüencial e outra paralela. A primeira construir uma rota rota por vez, enquanto a segunda versão constrói várias rotas ao mesmo tempo. Na versão seqüencial deve-se voltar para o começo da lista ordenada toda vez que uma nova conexão é realizada, uma vez que as combinações que não eram viáveis pode ter se tornado viável com a mudança feita.

Podemos ilustrar um exemplo da resolução do algoritmo com um exemplo prático. Considere um problema com 5 clientes. O custo de transporte são mostrados na tabela 1, onde o ponto zero corresponde ao depóstico e os custos são simétricos, ou seja, o valor que é gasto para ir de *a* para *b* é o mesmo de ir de *b* para *a* e, por isso, a tabela está preenchida somente em sua metade superior.

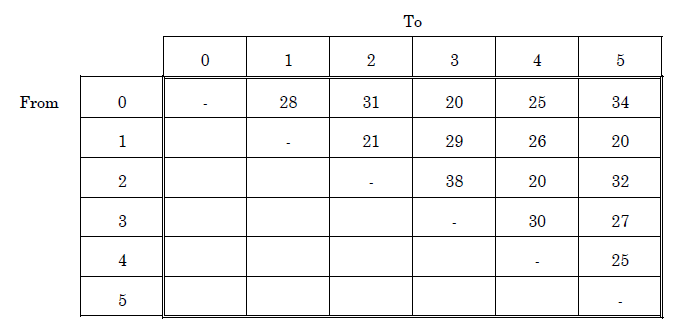


Tabela 1 - Custos de deslocamento

Considere também as seguintes demandas necessárias para abastecer cada cliente na tabela 2. A capacidade do veículo é de 100 unidades.

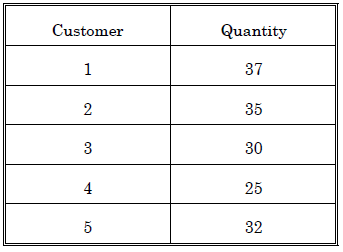


Tabela 2 - Demanda dos Clientes

As economias são calculados para valores localizados na tabela 3. Como inicialmente existe uma rota para cada cliente, cada par representa o custo de juntar dois clientes em uma mesma rota.

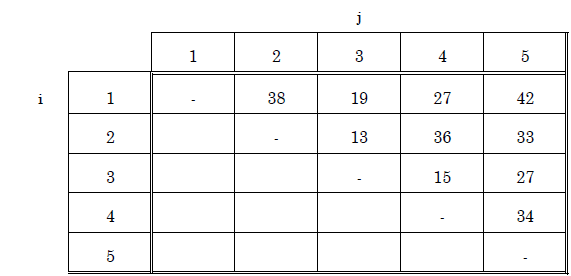


Tabela 3 - Custos de junção de rotas

Dessa forma é possível ranquear os valores em ordem descendente em relação às economias geradas. Isso gera a seguinte lista de pares:

[1-5; 1-2; 2-4; 4-5; 2-5; 1-4; 3-5; 1-3; 3-4; 2-3]

A partir de agora o algoritmo é mostrado na versão sequencial e em seguida apresentaremos a versão paralela. Primeiramente, consideramos o primeiro par da lista (1-5), eles podem ser fundidos na mesma rota desde que a restrição de capacidade do veículo não seja ultrapassada. Como o cliente 1 demanda 37 unidades e o cliente 5 demanda 32 a soma de seus valores (69) não excede a capacidade do veículo de 100 unidades. Então estabelecemos a conexão dos pontos 1-5, e por isso, esse par passarão a ser vizinhos de rota na solução final.

Em seguida consideramos o par 1-2. Caso essa conexão seja feita, isso geraria uma rota com a seqüência 2-1-5 (ou 5-1-2), porque já estabelecemos que o cliente 1 e 5 são vizinhos. Em razão disso a demanda total dessa rota passaria a extrapolar a capacidade do veículo em 4 unidades. Por isso, 1 e 2 não podem ser conectados.

Já se os pontos 2 e 4 fossem, que é o próximo par dar lista, estaríamos gerando mais de um rota no mesmo estágio (1-5 e 2-4). Já que a versão seqüencial do algoritmo é limitado a fazer apenas uma rota por vez, desconsideramos esse par por enquanto.

A combinação do próximo par, 4 e 5, resulta na rota 1-5-4 (ou 4-5-1) com uma demanda total de 94, ou seja, é uma conexão viável. Com isso, concluímos que não é possível adicionar mais nenhum novo cliente a rota uma vez que sua capacidade não suporta mais nenhuma carga de um novo possível cliente. A partir de então, fechamos a rota 0-1-5-4-0 e passamos a considerar uma nova rota. Como o par seguinte é 2-3 e essa conexão é viável pois não excede a capacidade do veículo realizamos essa conexão e terminamos o processo de fusões de rotas já que nenhum par de rotas podem ser conectados.

O algoritmo seqüencial gerou, então, uma solução com duas rotas. O custo de transporte para cada rota foi de 98 para 0-1-5-4-0 e 89 para 0-2-3-0. Totalizando um custo de 187.

Na versão paralela, da mesma forma que na versão seqüencial, o par 1 e 5 são conectados já que são viáveis. No entanto o algoritmo paralelo considera a criação de mais de uma rota por vez e, por isso, os pontos 2 e 4 são também combinados. Finalmente, os pontos 3 e 5 são combinados. Dessa forma o algoritmo gera duas rotas 0-1-5-3-0 e -2-4-0 com custo de transporte total de 171.

Vale notar que o número de rotas pode ser reduzido durante o processo do algoritmo paralelo. Duas rotas 0-1-2-0 e 0-3-4-0 serão combinados em uma rota se a conexão 2 e 3 for estabelecida; neste caso a rota resultante seria 0-1-2-3-4-0. Por isso, o algoritmo paralelo pode muitas vezes gerar resultados melhores que o algoritmo seqüencial. No entanto, ele requer um cuidado muito maior no gerenciamento de muitas rotas em paralelo e por conta disso, devem ser considerados o fator de tempo e qualidade de resposta na hora de decidir a versão mais apropriada para a aplicação.

### Algoritmos de Duas Fases

É um conjunto de algoritmos que resolve o problema do VRP divindo-o em dois problemas distintos:

1. **Agrupamento** – Processo que agrupa o conjunto de clientes que demandam atendimento por critério normalmente de localização geográfica sem que esses grupos gerem automaticamente soluções inviáveis. Por exemplo: gerar um agrupamento onde o veículo que vai percorrê-lo não tenha condições de atender em tempo hábil todos os clientes dentro de suas respectivas janelas de tempo ou que estoure sua capacidade de carga;
2. **Roteamento** – Processo de gerar uma rota, ou seja, um seqüenciamento de visitas que o veículo vai realizar para atender um conjunto de clientes;

Existem abordagens **Cluster-first Route-second** que realiza primeiro o agrupamento dos clientes e em seguida realiza o seqüenciamento dos atendimentos previstos de cada agrupamento e alocando a cada grupo um veículo. Neste tipo de algoritmo várias versões foram desenvolvidas e que geraram bons resultados para algumas instâncias do VRP, podemos citar: Fisher e Jaikumar (1981), The petal Algorithm, The sweep algorithm e Taillard. Esse número de trabalhos em cima desta abordagem mostrou que ela pode ser bem competitiva em relação a outras técnicas para determinados tipos de instância.

O ato de agrupar um conjunto de clientes que compartilhem de localizações geográficas semelhantes limita o espaço de busca drasticamente a um subconjunto de soluções de boa qualidade. Isso torna o algoritmo muito mais eficiente e permite que resolva sub-problemas muito mais simples. No entanto, há perigo de gerar um agrupamento inviável, ou seja, que viole uma restrição é o que mais limita os algoritmos de duas fases de ser bem sucedido para vários tipos de instâncias do problema. Restrições fortes de capacidade de carga, janelas de tempo ou restrição de tipo de veículo permitido a atender determinados clientes atrapalha a formação de bons agrupamentos e o torna pouco eficiente entradas fortemente restritas.

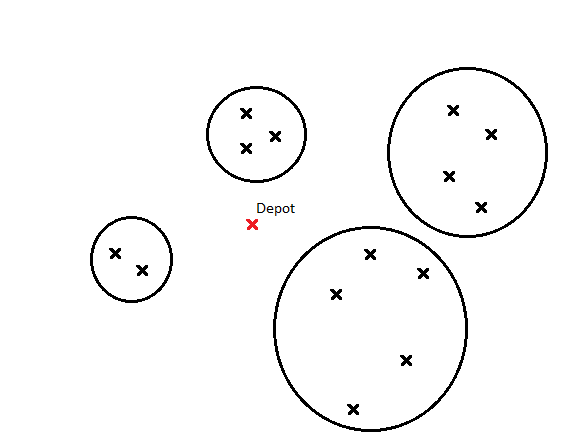


Figura 12 - Agrupamento dos clientes

Existe a subclasse de algoritmos **Route-first Cluster-second**. Sua idéia básica é primeiramente fazer um roteamento completo do grafo de clientes como se fosse somente um veículo, sem se preocupar com as restrições do problema, e daí decompô-lo em rotas de veículos viáveis. Essa abordagem melhor se adéqua a problemas com quantidade livre de veículos. Beasley foi que observou que essa segunda fase do algoritmo é um *Shortest Path Problem* e pode ser resolvido com custo de tempo de . No entanto continua com a dificuldade de resolver um TSP grande na primeira fase. Talvez por conta disso, não seja conhecido nenhuma experiência computacional que tenha demonstrado resultados competitivos dessa classe de algoritmos em relação aos algoritmos mais bem sucedidos.

## Meta-heurísticas

Meta-heurísticas são métodos heurísticos de uso geral para problemas de otimização onde não se sabe um algoritmo que encontre a solução ótima de forma eficiente (problemas NP-Completo). Através de combinações aleatórias e do histórico de navegação pelo espaço de busca, eles realizam operações de busca local e global de forma a percorrer o espaço de busca forma eficiente.

Neste tópico serão apresentadas as principais meta-heurísticas usadas para o problema de roteamento de veículos. No entanto, não abordaremos agora a técnica de colônia de formigas. Este será visto no próximo capítulo, uma vez que se trata da técnica que entraremos em mais detalhes a fim de explicar o algoritmo desenvolvido MACS-VRPTW.

### Simulated Annealing

É uma técnica inspirada na Termodinâmica. Para entender o que acontece, é interessante apresentar uma análise física. Vamos dividir o processo em duas fases: na primeira a temperatura do sólido é aumentada para um valor grande o suficiente no qual ele se funde; na segunda um resfriamento deve ser realizado lentamente até que o material se solidifique, sendo acompanhado e controlado esse *arrefecimento*. Nesta segunda fase, executada lentamente, os átomos que compõem o material organizam-se numa estrutura uniforme com energia mínima. Isto faz com que os átomos desse material ganhem energia para se movimentarem livremente e, ao arrefecer de forma controlada, este movimento é gradativamente inibido e os átomos se organizam numa configuração com menor energia interna, para ter, como resultado prático, uma redução dos defeitos do material.

        De forma análoga, o *Simulated Annealing* encontra uma solução de baixo custo por se tratar daquela que tem baixo nível “energia interna”. A técnica se resume a substituir a solução atual por uma solução próxima, ou seja, na sua vizinhança no espaço de soluções, escolhida de acordo com uma função objetivo e com um parâmetro to, a temperatura. Quanto maior  to, maior a componente aleatória que será incluída na próxima solução escolhida. À medida que o algoritmo avança, o valor de to é decrementado, e o algoritmo começa a convergir para uma solução ótima local.

        Uma das principais vantagens deste algoritmo é permitir testar soluções mais distantes da solução atual e dar mais independência do ponto inicial da pesquisa. Isto é importante pois se  iniciarmos com a entrada de soluções boas, poderia ser que o algoritmo não evoluísse e aceitasse a solução atual como a melhor. Como vemos, não é o caso do *Simulated Annealing*.

        A cada geração de um novo vizinho s’ de s, é testada a variação ∆ do valor da função objetivo, isto é, ∆ = f (s’) – f (s), onde temos as seguintes situações:

* ∆ < 0: Há uma redução de energia, que implica que a nova solução é melhor que a anterior. O método aceita a solução e s’ passa a ser a nova solução atual.
* ∆ = 0: Caso de estabilidade, não havendo redução de energia. Na verdade, situação pouco provável de acontecer na prática. A aceitação da solução é, portanto, indiferente.
* ∆ ≥ 0: Houve um aumento do estado de energia. A aceitação desse tipo de solução é mais provável a altas temperaturas e bastante improvável a temperaturas reduzidas.

        Para reproduzir essas características, geralmente usa-se uma função conhecida por fator de *Boltzmann*, que é dada por e(-∆/t), onde a temperatura regula a probabilidade de soluções com pior custo.

### Tabu-Search

A técnica Tabu Search também é um método de busca local. No entanto, ela possui mecanismos que permitem contornar o problema das soluções locais, visando obter a solução global (ótima) através do uso de estruturas que tentam imitar a memória dos seres humanos. Tais estruturas de memória permitem uma análise sistemática do espaço das soluções através do registro do percurso realizado nas últimas iterações do algoritmo.

        Naturalmente, apresenta uma complexidade maior em relação ao Simulated Annealing já que utiliza um método de busca dinâmica na vizinhança de uma solução. Em geral, três pontos são muito importantes nesta técnica:

* O Tabu Search permite analisar as soluções na vizinhança da solução atual. As soluções obtidas em cada iteração são também analisadas, tentando extrair características que possam ser comuns a boas soluções e, desse modo, definir as “direções de busca” que podem conduzir às melhores soluções;
* É possível desenvolver métodos que incentivam a exploração de boas soluções (cuja qualidade é também analisada) e afastam a busca das direções que levam a soluções mais fracas;
* Sendo estruturas flexíveis, o uso das memórias permite obter informações sobre a influência dos movimentos de uma solução na estrutura e qualidade da solução seguinte.

        Nem sempre o resultado de uma iteração é uma solução melhor que a anterior (por exemplo, ao não ser respeitada uma restrição inicialmente imposta). Este aparente recuo poderá conduzir à exploração de outras zonas e, deste modo, superar a limitação de um mínimo local. A ação da memória é determinante para evitar o aparecimento de ciclos, ou que soluções já conhecidas sejam novamente alcançadas.

        O que acontece é que algumas soluções são marcadas como “Tabu”, de forma que sejam evitadas. Naturalmente, a existência de um conjunto de soluções Tabu poderá impedir a obtenção de soluções ainda desconhecidas, eventualmente melhores do que as que são conhecidas. Para superar esta limitação, torna-se necessária a inserção de relaxações nos movimentos “tabu”. Assim, é definida uma função f(i,s) que tem como parâmetros a iteração i e a solução atual s e retorna a tendência de se contrariar um determinado Tabu.

### Algoritmos Genéticos

Um Algoritmo Genético (AG) segue passos inspirados no processo biológico de evolução natural segundo a teoria de Darwin. Seguem a idéia de sobrevivência do mais forte (as soluções com maiores aptidões de cada geração). É um algoritmo estocástico (não é determinístico) que trabalha com uma população de soluções simultaneamente e utiliza apenas informações de custo e recompensa. Não requer nenhuma outra informação auxiliar (como por exemplo o gradiente).

        Normalmente, eles são fáceis de serem implementados e se adaptam bem a computadores paralelos. São facilmente hibridizados com outras técnicas e funcionam com parâmetros contínuos ou discretos.

        Os Algoritmos Genéticos manipulam uma população de indivíduos (possíveis soluções). Os indivíduos são combinados (cross-over) uns com os outros, produzindo filhos que podem sofrer ou não mutação (normalmente com uma baixa probabilidade).  As populações evoluem através de sucessivas gerações até encontrar a solução ótima.

# Sistema de Colônias de Formigas (ACS)

O Sistemas de Colônia de Formigas (Ant Colony System - ACS) é mais uma meta-heurística inspirada em fenômenos da natureza. A idéia está baseada no fato das colônias de formigas serem capazes de otimizar naturalmente seus caminhos em busca do alimento através de rastros de feromônios deixados por elas mesmas.

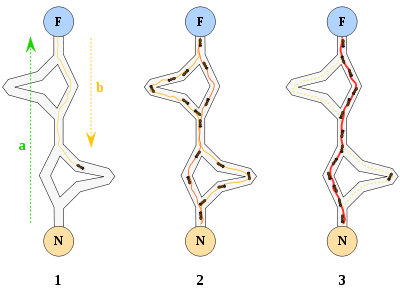


Figura 13 - Conceito de Colônia de Formigas

A regra básica em torno de um sistema de colônia de formigas é que toda vez que uma formiga acha uma rota para o alimento, em seu caminho de volta para o formigueiro ela libera seu feromônio. Ao longo do tempo, várias formigas vão encontrando a própria rota para o alimento. Estas também vão deixar seus rastros, que vão se sobrepondo e formando *n* caminhos com diferentes níveis de feromônio. As formigas que percorrem posteriormente o caminho para o alimento usam a informação dos rastros pré-existentes junto com um fator de aleatoriedade para decidir os seus próximos passos. O resultado quando o processo está amadurecido é que um caminho de feromônio prevalecerá e esse será um caminho viável (alcance o objetivo de chegar ao alimento) e de baixo custo (tenha o menor percurso).

Esse fenômeno existente nas colônias de formigas pode ser aplicado computacionalmente e resolver de forma eficiente problemas do TSP (*Traveling Salesman Problem*) e os problemas que derivam dele, tais quais: mTSP e VRP. Diferente das outras técnicas já analisadas, o ACS tem um bom nível de compatibilidade com problemas de roteirização, e por isso, apresentam facilidade na adequação do algoritmo ao problema. Além disso, ele contém os elementos básicos de toda meta-heurística:

* **Intensificação** – A melhor rota recebe é valorizada e novas rotas são incentivadas a manter esse caminho;
* **Diversificação** – Rotas alternativas são abertas ao longo do tempo a fim de encontrar novos “vales” no espaço de soluções;

Na modelagem de um algoritmo de otimização é comum que alguns conceitos sejam ajustados a fim de se adaptar ao problema que se deseja resolver. No caso do ACS, ao contrário de uma colônia de formigas comum que deseja ir de um ponto a outro, o objetivo é realizar um tour em cima de um grafo de modo que todos os vértices sejam cobertos. Neste grafo, cada aresta contém um peso proporcional ao custo de deslocamento do ponto ao e um nível de feromônio associado . Sobre este grafo formigas devem realizar um *tour* sobre todos os pontos do grafo. A maneira como realiza seu *tour* é determinada nó a nó. A cada vértice do grafo que uma formiga cobre, ela deve realizar a decisão sobre qual próximo vértice ela deve percorrer dentre um conjunto de nós viáveis . Para isso, uma regra probabilística é usada: com probabilidade o nó com maior , é escolhido (intensificação). Enquanto que com probabilidade o nó é escolhido com uma probabilidade proporcional a , (diversificação):

Onde e são parâmetros: é o peso relativo dado à importância do valor heurístico, enquanto determina a importância relativa da diversificação versus intensificação: quanto menor maior a probabilidade de usar a regra probabilística descrita na equação 1.

Uma vez que as formigas realizaram seu tour, é realizado um procedimento de busca local que deve varrer todas as soluções vizinhas com intuito de se certificar de que a solução encontra-se de fato em um mínimo local. Em seguida a melhor solução encontrada é usada para atualizar o rastro de feromônio. A partir daí, o ciclo é mais uma vez inciado até que se alcance um critério de parada que deve ser: um número fixo de soluções foram geradas, um tempo fixo de CPU foi gasto, ou nenhuma melhora na solução foi alcançada durante um dado número de iterações.

No ACS, existem dois tipos de atualização do feromônio: a atualização local e a atualização global. Enquanto a atualização global atua na intensificação de busca próximo a melhor solução encontrada, a atualização local busca diversificar a solução desmotivando a construção de rotas sobre a melhor já encontrada. Esses efeitos são alcançados através da aplicação dos seguintes artifícios:

* Toda vez que termina um ciclo do ACS a melhor rota é usada para atualizar as trilhas de ferômonio pelo grafo;
* Em contrapartida durante os *tours* feitos pelas formigas, cada vez que uma aresta é usada o seu ferômonio é degradado de forma a tornar aquele caminho menos atrativo para as futuras formigas que passarem por nós vizinhos;

No ACS, apenas a melhor solução encontrada é usada para fazer a atualização global do feromônio. Na estratégia de atualização proposta por Dorigo, Maniezzo e Colorni em 1991 com o Ant System (AS) todas as rotas eram usadas para atualizar a matriz de feromônio. Foi provado, no entanto, que a melhor estratégia é a proposta por Gambardella e Dorigo que é usada em ACS. A lógica é que uma “rota preferida” é memorizada na matriz de feromônio e futuras formigas usarão esta informação para gerar novas soluções na vizinhança da rota preferida. O é atualizado como segue:

Onde é um parâmetro e é o comprimento de , o menor caminho gerado pelas formigas desde o começo da computação. Esta atualização é aplicada ao fim de cada ciclo, ou seja, cada vez que a fase construtiva termina.

Para a atualização local, toda vez que uma formiga se move do nó para o nó , o nível de feromônio na aresta (i,j) é diminuído de acordo com a seguinte regra:

Onde é o valor inicial das trilhas. Foi descoberto que é um bom valor para este parâmetro onde é o comprimento da solução incial produzida pela heurística do vizinho mais próximo (Flood, 1956) e é o número de nós.

# MACS-VRPTW

MACS-VRPTW é um algoritmo específico de meta-heurística que se baseia no Sistema de Colônia de Formigas para resolver o problema de roteamento de veículos com capacidade de carga limitada para os veículos e janelas de tempo para os clientes. Sua sigla significa *Multi Ant Colony System for VRPTW* e tem esse nome porque o algoritmo roda mais de um ACS de forma simultânea.

Ao contrário de outros algoritmos que buscam minimizar somente o caminho total percorrido pelos veículos, o MACS-VRPTW tem dois objetivos simultâneos:

1. Minimizar a quantidade total de veículos;
2. Minimizar o caminho total percorrido pelos veículos;

Sendo o primeiro objetivo é levado com maior prioridade em relação ao segundo, ou seja, se foi alcançado uma solução que consiga realizar o percurso com uma quantidade menor de veículos, essa solução vai ser aceita mesmo que o custo, em termos de caminho total percorrido, seja maior.

No algoritmo, duas ACS são usadas:

* **ACS-TIME** – Busca minimizar, com uma frota de veículos, o deslocamento total que deverá ser realizado para atenda todos os clientes dentro dos critérios pré-estabelecidos de janela de tempo e capacidade;
* **ACS-VEI** – Busca encontrar um solução viável, que atenda todos os clientes corretamente, com uma frota de veículos ;

Quando o algoritmo é iniciado, uma solução inicial viável é computada usando a heurística do vizinho mais próximo. Esta solução inicial é usada para determinar a melhor solução e a quantidade inicial de veículos . A partir daí, duas *threads* serão ativadas, uma para o ACS-TIME que vai minimizar a solução com veículos e outra para o ACS-VEI que vai minimizar a solução com veículos. Aguarda-se até ou o ACS-TIME retorne uma solução com um custo total de deslocamento menor ou o ACS-VEI retorne solução viável com um veículo a menos. No momento que alguma delas retornar uma nova solução, o é atualizado. Caso esta nova solução tenha uma quantidade de veículos menor do que , as *threads do* ACS-TIME e do ACS-VEI serão encerradas e as duas *threads* são reiniciadas, sendo que agora com decrementado em uma unidade. Esse laço será perpetuado até que seja alcançado um critério de parada.

## Colônias ACS-TIME e ACS-VEI

O ACS-TIME é um procedimento muito parecido com o ACS usado para construir soluções do TSP (*Traveling Salesman Problem*). Esse procedimento recebe como entrada uma solução incial para uma frota de veículos. Com essa entrada, é possível inicializar as trilhas de feromônio do grafo correspondente à região de atuação. A partir de então serão geradas formigas que serão ativadas pelo procedimento *active\_ant* com o modo de busca local ativado. Cada formiga irá gerar uma solução, e ao fim desse processo, caso a melhor solução encontrada seja viável e tenha um custo menor do que a melhor solução atual, ela será guardada em como a nova melhor solução e usada como parâmetro para realizar a atualização global de feromônio do grafo. Este procedimento se repetirá até alcançar um critério de parada.

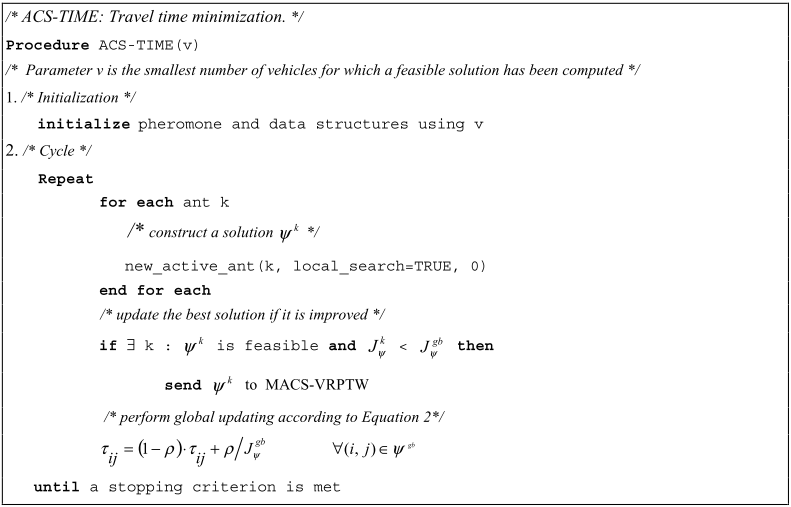


Figura 14 - Algoritmo ACS-TIME

Em relação ao ACS-VEI, diferente do ACS-TIME, o objetivo é de encontrar uma solução viável com a quantidade de veículos passada por parâmetro. Para isso, ele retira da solução atual um veículo, e as atividades que passarem a ficar desalocadas deverão sofrer tentativas de inserção ao longo do procedimento. A nova solução inviável gerada com a retirada do veículo deve ser utilizada para inicializar o grafo do feromônio. Um vetor com nome IN é gerado, cada célula do vetor deve corresponder à quantidade de vezes foi gerada uma solução pelo ACS sem que determinado cliente fosse visitado.

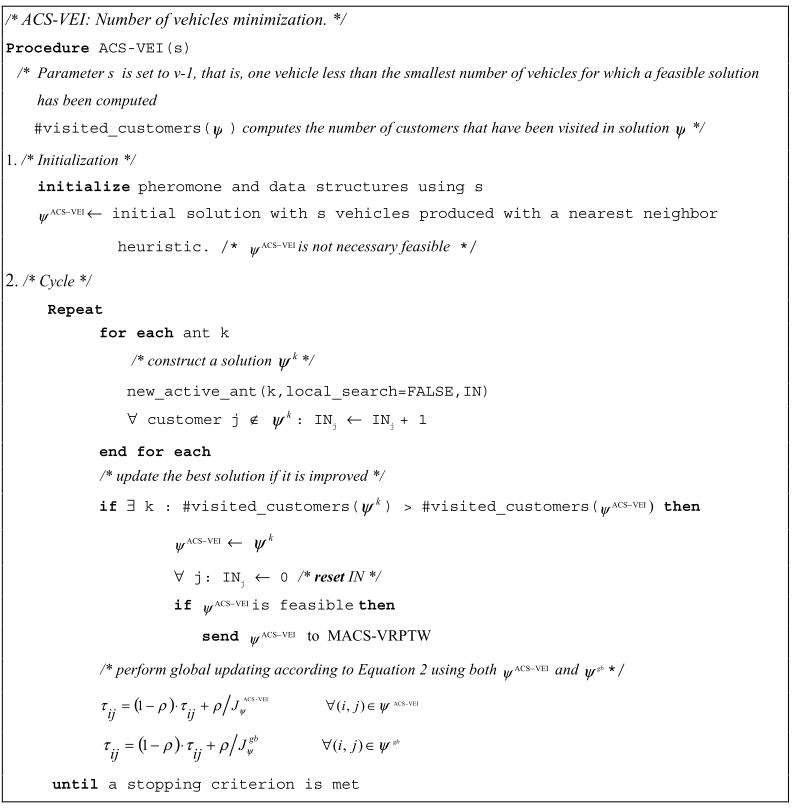


Figura 15 - Algoritmo ACS-VEI

No ciclo principal, a tentativa de se gerar uma solução viável se dá de forma parecida com o ACS-TIME, formigas são geradas e ativadas pelo *active\_ant*, porém desta vez a o procedimento de busca local vai desativado, já que não faz sentido fazer busca local em cima de uma solução inviável e o vetor IN é passado como parâmetro. Após a ativação da formiga a solução gerada é analisada e o vetor IN é atualizado através da incrementação de uma unidade nas células em que o cliente correspondente não foi visitado. Esse vetor é usado no procedimento *active\_ant* para dar prioridade àqueles clientes que estão tendo dificuldades em entrar em na solução. Uma vez que todas as formigas foram ativadas, é verificado se a quantidade de clientes que foram visitados pelas soluções geradas sofreu alguma melhora, em caso positivo a melhor solução é guardada em e o vetor IN é reinicializado, e caso ainda a nova solução seja viável ela é enviada para o MACS-VRPTW como a nova melhor solução encontrada . Caso uma nova solução viável não seja encontrada, o feromônio é atualizado globalmente e o processo é refeito.

## Modelo da Solução

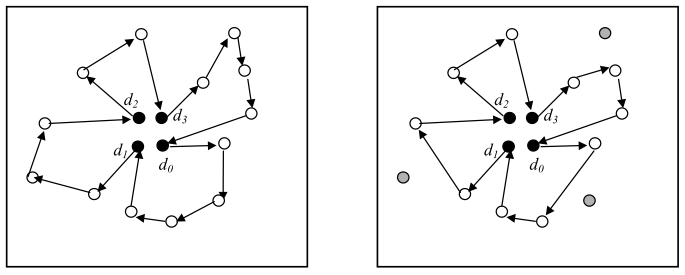


Figura 16 - Replicação do depósito

Para entender bem o algoritmo é importante entender o tipo de solução que ele gera. No MACS-VRPTW a solução gerada foi adaptada para que se transformasse na solução do TSP, ou seja, um caminho que passe por todos os pontos do grafo. Para realizar essa adaptação, é preciso replicar o depósito a quantidade de vezes necessárias para que tenha a mesma quantidade veículos. Esses depósitos devem ser idênticos em termos de localização e de conexão com os outros pontos. Na solução, o caminho deve sair de um dos depósitos e partir para percorrer todos os pontos. Logicamente em algum momento ele passará pelos depósitos replicados, e cada vez que passar isso corresponderá a um novo veículo saindo do depósito no horário inicial de saída. Essa abordagem tem a vantagem de que as trilhas em direção ao depósito não ficam mais atrativas com o aumento da quantidade de veículos. Isso afeta positivamente a qualidade das soluções produzidas pelo procedimento construtivo.

## Procedimento da Solução construtiva

ACS-TIME e ACS-VEI usam o mesmo procedimento construtivo chamado *active\_ant*. Ele é similar ao usado no procedimento construtivo da ACS para o TSP. Seu objetivo é construir uma solução para o VRP utilizando o modelo de solução descrito na seção anterior. Então a formiga deve escolher um dos depósitos replicados para iniciar sua rota e em seguida escolher sucessivamente o próximo cliente a visitar dentre aqueles que são viáveis usando o mecanismo de intensificação e diversificação como foi descrito na equação 1. Neste mecanismo, o processo da escolha da formiga vai ser determinado probabilisticamente pelo nível de atratividade de cada vértice possível. Esta atratividade vai ser determinada pelo resultado da combinação dos fatores:

* Percepção local de melhor caminho (heurística do vizinho mais próximo);
* Atratividade provocada pela trilha de feromônio;
* E nos casos do *active\_ant* chamado pelo ACS-VEI o (quando é chamado por ACS-TIME IN não é usado e tem valor padrão de zero) que determina a quantidade de vezes que o vértice não foi inserido na solução em tentativas anteriores;

Esse processo de escolha se repetirá até o momento que a formiga não encontre mais nenhum vértice viável do grafo para visitar. Para ser um vértice viável no MACS-VRPTW ele deve obedecer a regra da janela de tempo e da capacidade de carga do veículo.

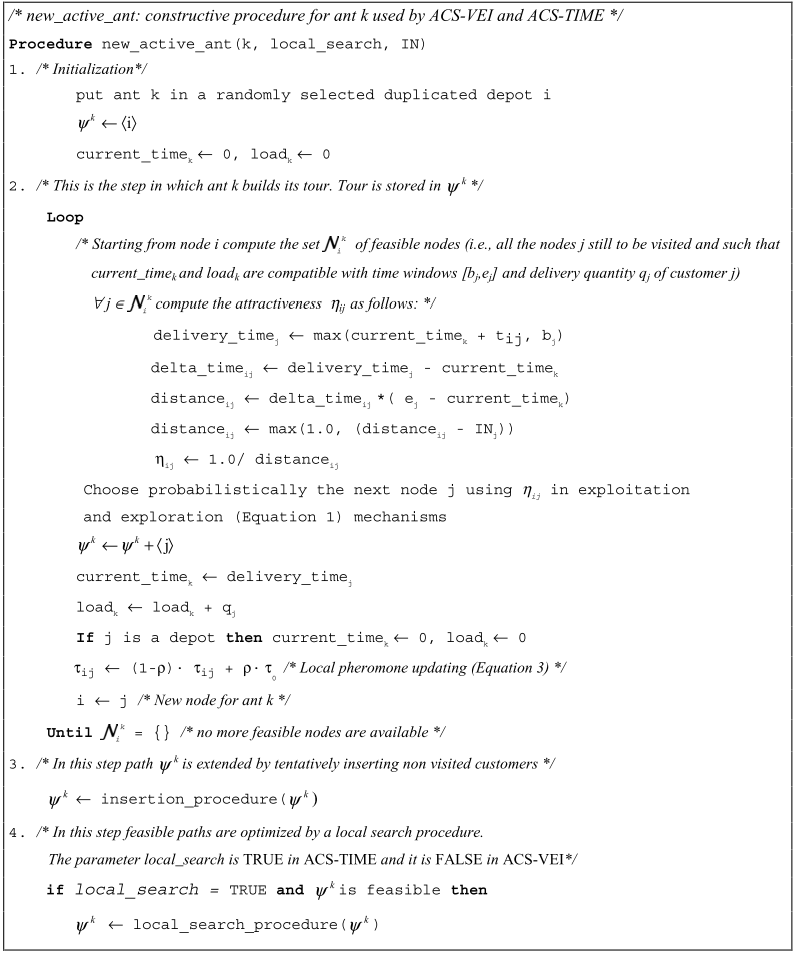


Figura 17 - Algoritmo new\_active\_ant

No processo de construção da solução, a cada iteração em que a formiga percorre uma aresta do grafo, uma atualização local é realizada para decrementar o nível de feromônio da aresta usada e com isso diminuir seu nível de atratividade como descrito na equação 3.

É comum, no entanto, a formiga percorrer todos os vértices viáveis e, mesmo assim, ainda restarem vértices que ainda não foram visitados. Como o objetivo deste procedimento é gerar uma solução viável e completa para o MACS-VRPTW foi introduzido um procedimento de inserção que tenta na “força bruta” inserir todos os vértices que ainda não entraram na solução em algum ponto da rota da formiga. Como na construção do caminho pela formiga sobram poucos vértices não visitados, o procedimento da inserção se mostra eficiente no seu objetivo de completar a solução.

Por fim, um procedimento de busca local é realizado com a finalidade de explorar de forma mais objetiva as soluções vizinhas a fim de atingir o mínimo local da região de exploração atual. O procedimento usado no MACS-VRPTW é o CROSS Exchange, ele realiza diversas trocas de segmento de rotas entre os veículos. Para cada par de veículos são testadas todas as combinações de troca de rotas e a melhor troca encontrada é escolhida. Além desse procedimento é possível também adicionar o procedimento de busca local INTRA Exchange que ao invés de trocar parte da rota e depois voltar para ela mesma, a troca é realizada e dessa vez termina a rota no “depósito replicado” do par vizinho.

### Procedimento da Inserção

No tópico anterior foi descrito que o MACS-VRPTW usa um procedimento de inserção para completar rotas incompletas geradas pelo método *active\_ant*. Esse procedimento recebe uma solução parcial do VRP e um conjunto de clientes (vértices do grafo) que devem ser incluídos na solução. O procedimento consiste basicamente em percorrer todas as rotas da solução buscando por folgas entre os atendimentos dos veículos que possa ser inserido uma das atividades desalocadas sem que quebre nenhuma restrição imposta pelo problema. Essa iteração é feita até que não exista mais nenhum cliente para ser inserido ou que não seja encontrada nenhuma brecha viável para inserir os clientes restantes.

Note que este procedimento visa somente completar a solução sem se preocupar com sua qualidade da solução que está sendo alterada. Por isso, essa técnica é recomendada apenas para inserir poucos clientes que estejam faltando, uma vez que a tendência é que esse procedimento deteriore a solução até então gerada. Caso o procedimento seja aplicado para inserir muitos clientes, a solução não terá qualidade alguma.

### Procedimento de Busca Local: CROSS-EXCHANGE

É um procedimento que realiza busca por soluções vizinhas visando alcançar o mínimo local da região de busca onde se encontra. No MACS-VRPTW ele é aplicado toda vez que acha uma nova solução pelo procedimento *active\_ant*. O objetivo é que uma vez que a colônia de formigas achou uma solução de boa qualidade explorar seus vizinhos de modo fechar o espaço de busca pela região e, dessa forma, melhorar em relação a solução já encontrada.

A idéia do procedimento é realizar trocas de segmentos de rotas entre os veículos. Essas trocas não podem violar o conjunto de restrições do problema e devem sempre apresentar uma melhora em relação à resposta anterior. O procedimento pode ser descrito da seguinte forma:

1. Para cada par de rotas (r1 e r2) existente na solução faça:
   1. Gere todos os possíveis subconjuntos das rotas r1 e r2;
   2. Simule troca de cada possível par de segmentos de rotas entre r1 e r2;
   3. Para cada simulação que apresentar um resultado melhor em relação a atual melhor solução realize a troca como mostra a figura 18 e continue o procedimento;

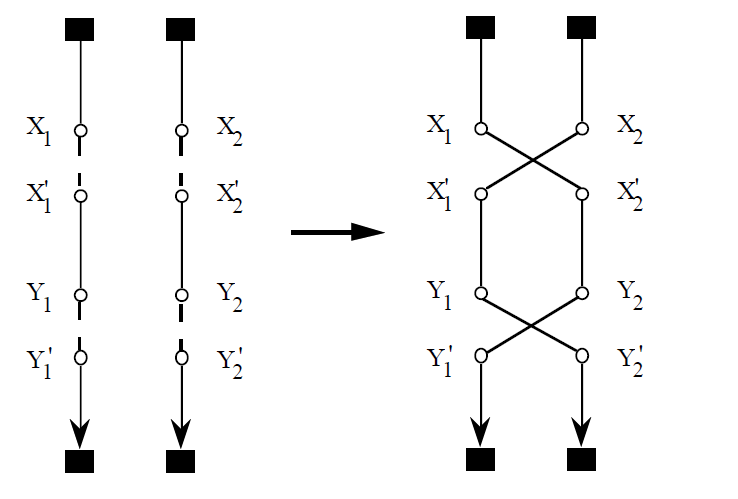


Figura 18 - Operação de CROSS-EXCHANGE

## Resultados

O algoritmo MACS-VRPTW foi baseado no artigo de Gambardella de 1999 [5]. Os resultados obtidos pela implementação dele são mostrados nessa seção. Além disso, será feita uma análise dos resultados alcançados. A implementação possui algumas diferenças em relação ao algoritmo implemntado por [16]. Podemos listar as seguintes diferenças:

* O critério de parada é definido pela quantidade de ciclos que o resultado não evoluiu. Na implementação proposta pelo artigo é posto um tempo fixo de parada;
* O algoritmo implementado não considera a capacidade do veículo;
* Na implementação original, foi adicionado ao algoritmo o procedimento de busca local INTRA-EXCHANGE além do CROSS-EXCHANGE. Neste trabalho a busca local é realizada apenas com CROSS-EXCHANGE;
* Existem diferenças implícitas não idenficadas que afetaram a qualidade do resultado em relação ao que foi apresentado no artigo;

O algoritmo foi testado em cima dos problemas de benchmark de Solomon [6] (Solomon, 1987) composto por 6 tipos de problemas diferentes (C1, C2, R1, R2, RC1, RC2). Cada conjunto de problemas tem entre 8 e 12 problemas de tamanho de 100 nós. Os nomes dos seis problemas têm os seguintes significados: C significa que são problemas cujos nós estão agrupados (clusters). O R significa que os nós estão espalhados de forma randômica. E RC significa que são os nós espalhados de forma híbrida, parte dos nós estão agrupados e parte estão espalhados randomicamente. Em relação à numeração os problemas que começam com 1 têm janelas de tempo estreitas e pequena capacidade de carga do veículo. Enquanto os que começam com 2 têm largas janelas de tempo e alta capacidade de carga. Por isso, vai ser possível notar nos resultados que problemas do tipo 2 têm bem menos rotas e, conseqüentemente, cada veículo atende muito mais clientes. Vale salientar que a versão do algoritmo apresentado neste trabalho não considera a capacidade do veículo, portanto, apenas as janelas de tempo são consideradas.

Os experimentos foram feitos, executando 3 vezes para cada entrada diferente, e o critério de parada do algoritmo foi de encerrar o procedimento após 10 iterações sem melhoria na solução. A implementação do código foi feita em C#. Os experimentos foram feitos com os seguintes parâmetros:

* = 10 formigas;
* ;
* ;
* ;

Abaixo são os resultados são mostrados. Na primeira coluna é identificada a instância do problema. Nas colunas seguintes é possível observar os resultados mais avançados até o momento, com a referência devida ao artigo, em seguida os resultados do artigo para a técnica do MACS-VRPTW, e por fim os resultados deste trabalho.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Outros artigos | | | MACS-VRPTW | | Este trabalho | |
| Instance | Vehicles | Distance | Artigo | Vehicles | Distance | Vehicles | Distance |
| R101 | 20 | 1642.88 | Jung and Moon [13] | 19 | 1652.47 | 20 | 1772.03 |
| R102 | 18 | 1472.62 | Alvarenga et al. [3] | 17 | 1677.01 | 18 | 1646.11 |
| R103 | 14 | 1213.62 | Rochat and Taillard [18] | 13 | 1383.26 | 15 | 1416.22 |
| R104 | 11 | 976.61 | Jung and Moon [13] | 10 | 1038.39 | 13 | 1481.22 |
| R105 | 13 | 1360.78 | Jung and Moon [13] | 14 | 1414.54 | 16 | 1662.39 |
| R106 | 13 | 1239.37 | Ribas *et al*. [14] | 13 | 1358.63 | 14 | 1448.62 |
| R107 | 11 | 1073.34 | Jung and Moon [13] | 11 | 1188.28 | 13 | 1359.27 |
| R108 | 10 | 948.57 | Alvarenga [2] | 10 | 1087.19 | 12 | 1388.94 |
| R109 | 13 | 1101.84 | Jung and Moon [13] | 12 | 1284.76 | 13 | 1411.10 |
| R110 | 12 | 1072.41 | Jung and Moon [13] | 11 | 1198.72 | 13 | 1413.94 |
| R111 | 12 | 1053.50 | Jung and Moon [13] | 11 | 1250.95 | 12 | 1413.78 |
| R112 | 10 | 953.63 | Rochat and Taillard [18] | 10 | 1088.97 | 11 | 1333.95 |

Tabela 4 - Melhores resultados encontrados para a classe de problemas R1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Other works | | | MACS-VRPTW | | Este trabalho | |
| Instance | Vehicles | Distance | Work | Vehicles | Distance | Vehicles | Distance |
| R201 | 8 | 1147.80 | Oliveira and Vasconcelos [12] | 4 | 1434.60 | 5 | 1470.15 |
| R202 | 8 | 1034.35 | Jung and Moon [13] | 3 | 1349.58 | 4 | 1362.93 |
| R203 | 6 | 874.87 | Jung and Moon [13] | 3 | 1168.08 | 4 | 1335.71 |
| R204 | 5 | 735.80 | Oliveira and Vasconcelos [12] | 3 | 1131.28 | 3 | 1038.55 |
| R205 | 5 | 954.16 | Oliveira and Vasconcelos [12] | 3 | 1141.01 | 4 | 1389.70 |
| R206 | 5 | 879.89 | Jung and Moon [13] | 3 | 1192.97 | 3 | 1429.56 |
| R207 | 4 | 797.99 | Oliveira and Vasconcelos [12] | 3 | 1284.06 | 3 | 1353.94 |
| R208 | 4 | 705.45 | Jung and Moon [13] | 3 | 1176.35 | 3 | 1158.35 |
| R209 | 5 | 859.39 | Jung and Moon [13] | 3 | 1027.71 | 3 | 1412.48 |
| R210 | 5 | 910.70 | Jung and Moon [13] | 3 | 1451.83 | 4 | 1337.53 |
| R211 | 4 | 755.82 | Oliveira and Vasconcelos [12] | 3 | 1146.55 | 3 | 1272.09 |

Tabela 5 - Melhores resultados encontrados para a classe de problemas R2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Other works | | | MACS-VRPTW | | Este trabalho | |
| Instance | Vehicles | Distance | Work | Vehicles | Distance | Vehicles | Distance |
| C101 | 10 | 828.94 | Rochat and Taillard [18] | 10 | 828.94 | 10 | 838.02 |
| C102 | 10 | 828.94 | Rochat and Taillard [18] | 10 | 828.94 | 10 | 1211.21 |
| C103 | 10 | 828.06 | Rochat and Taillard [18] | 10 | 970.13 | 10 | 1149.74 |
| C104 | 10 | 824.78 | Rochat and Taillard [18] | 10 | 1284.59 | 10 | 1157.64 |
| C105 | 10 | 828.94 | Rochat and Taillard [18] | 10 | 828.94 | 10 | 937.50 |
| C106 | 10 | 828.94 | Rochat and Taillard [18] | 10 | 828.94 | 10 | 971.54 |
| C107 | 10 | 828.94 | Rochat and Taillard [18] | 10 | 828.94 | 10 | 929.14 |
| C108 | 10 | 828.94 | Rochat and Taillard [18] | 10 | 1115.55 | 10 | 1062.90 |
| C109 | 10 | 828.94 | Rochat and Taillard [18] | 10 | 1539.70 | 10 | 979.01 |

Tabela 6 - Melhores resultados encontrados para a classe de problemas C1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Other works | | | MACS-VRPTW | | Este trabalho | |
| Instance | Vehicles | Distance | Work | Vehicles | Distance | Vehicles | Distance |
| C201 | 3 | 591.56 | Rochat and Taillard [18] | 3 | 591.56 | 3 | 591.56 |
| C202 | 3 | 591.56 | Rochat and Taillard [18] | 3 | 591.56 | 4 | 1016.01 |
| C203 | 3 | 591.17 | Rochat and Taillard [18] | 3 | 736.85 | 4 | 1212.97 |
| C204 | 3 | 590.60 | Rochat and Taillard [18] | 3 | 744.96 | 4 | 1124.27 |
| C205 | 3 | 588.88 | Rochat and Taillard [18] | 3 | 588.88 | 3 | 627.92 |
| C206 | 3 | 588.49 | Rochat and Taillard [18] | 3 | 588.49 | 3 | 631.06 |
| C207 | 3 | 588.29 | Rochat and Taillard [18] | 3 | 588.29 | 4 | 754.77 |
| C208 | 3 | 588.32 | Rochat and Taillard [18] | 3 | 588.49 | 3 | 642.66 |

Tabela 7 - Melhores resultados encontrados para a classe de problemas C2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Other works | | | MACS-VRPTW | | Este trabalho | |
| Instance | Vehicles | Distance | Work | Vehicles | Distance | Vehicles | Distance |
| RC101 | 15 | 1623.58 | Rochat and Taillard [18] | 15 | 1657.63 | 16 | 1866.16 |
| RC102 | 14 | 1461.23 | Jung and Moon [13] | 13 | 1515.84 | 14 | 1763.73 |
| RC103 | 11 | 1249.86 | Jung and Moon [13] | 11 | 1475.51 | 13 | 1554.99 |
| RC104 | 10 | 1135.48 | Cordeau et al. [19] | 11 | 1252.09 | 12 | 1449.14 |
| RC105 | 16 | 1518.58 | Jung and Moon [13] | 14 | 1626.88 | 16 | 1897.60 |
| RC106 | 13 | 1377,35 | Alvarenga *et al*. [3] | 12 | 1429.61 | 14 | 1715.06 |
| RC107 | 12 | 1212.83 | Jung and Moon [13] | 12 | 1480.11 | 14 | 1597.89 |
| RC108 | 11 | 1117.53 | Jung and Moon [13] | 11 | 1352.92 | 12 | 1328.70 |

Tabela 8 - Melhores resultados encontrados para a classe de problemas RC1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Other works | | | MACS-VRPTW | | Este trabalho | |
| Instance | Vehicles | Distance | Work | Vehicles | Distance | Vehicles | Distance |
| RC201 | 9 | 1265.56 | Jung and Moon [13] | 4 | 1487.85 | 5 | 1711.00 |
| RC202 | 8 | 1095.64 | Jung and Moon [13] | 4 | 1358.45 | 4 | 1636.74 |
| RC203 | 5 | 926.89 | Oliveira and Vasconcelos [12] | 3 | 1329.36 | 4 | 1496.92 |
| RC204 | 4 | 786.38 | Jung and Moon [13] | 3 | 1167.13 | 3 | 1440.49 |
| RC205 | 7 | 1157.55 | Jung and Moon [13] | 4 | 1509.41 | 5 | 1765.14 |
| RC206 | 7 | 1054.61 | Jung and Moon [13] | 3 | 1317.78 | 4 | 1576.57 |
| RC207 | 6 | 966.08 | Jung and Moon [13] | 3 | 1171.31 | 4 | 1529.20 |
| RC208 | 4 | 778.93 | Oliveira and Vasconcelos [12] | 3 | 1095.12 | 3 | 1362.52 |

Tabela 9 - Melhores resultados encontrados para a classe de problemas RC2

Os resultados alcançados nesta implementação, apesar de não ter alcançado os níveis reais dos algoritmos, se mostraram eficientes melhorar o resultado. Nas imagens abaixo comparamos os resultados gerados pela heurística de Nearest Neighbor de Gambardella, pelo procedimento de busca local CROSS-EXCHANGE (usa Nearest Neighbor como base) e pelo MACS-VRPTW, e notamos que há uma melhora substancial entre um algoritmo e outro. No primeiro exemplo, o resultado saiu de 2015,21 com NN (Nearest Neighbor), passou para 1949,92 com o CROSS-EXCHANGE e alcançou 1772,03 com o algoritmo de Meta-heurística que é mais avançado. No segundo exemplo, o resultado começou com 257,92 com NN, passou para 196.87 com CROSS-EXCHANGE e foi para o resultado ótimo com MACS-VRPTW de 191,81.

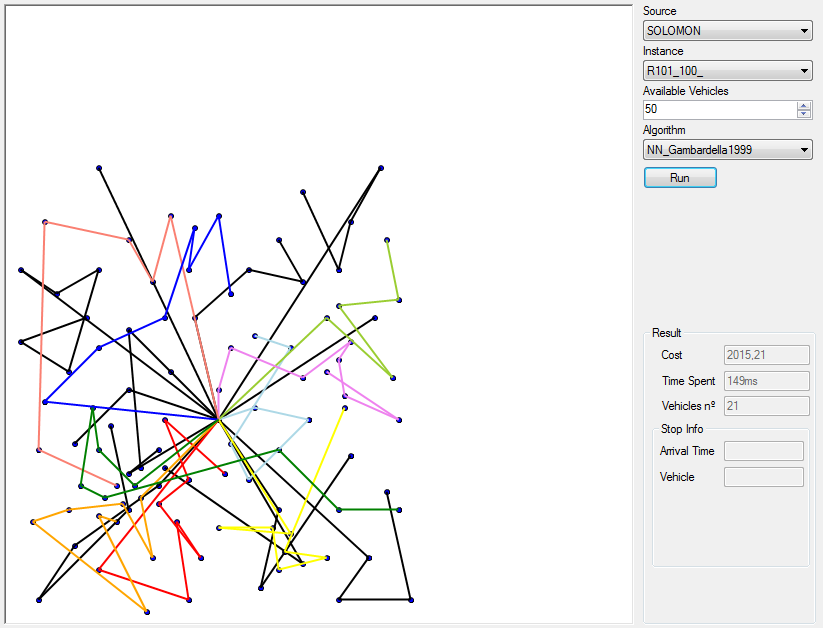


Figura 19 - Instância R101, com o algoritmo NN\_Gambardella alcançou o custo de 2015,21.

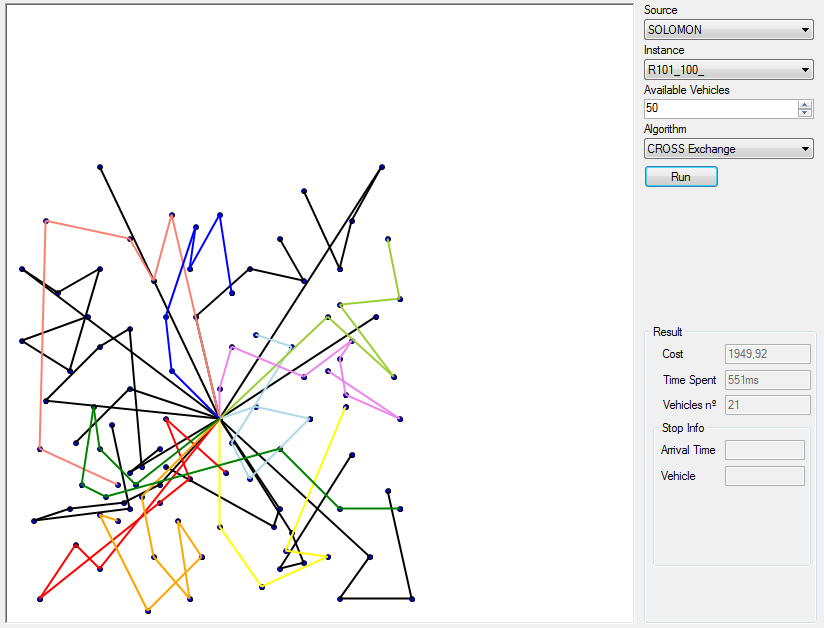


Figura 20 - Instância R101, com o algoritmo CROSS-EXCHANGE alcançou o custo de 1949,92.

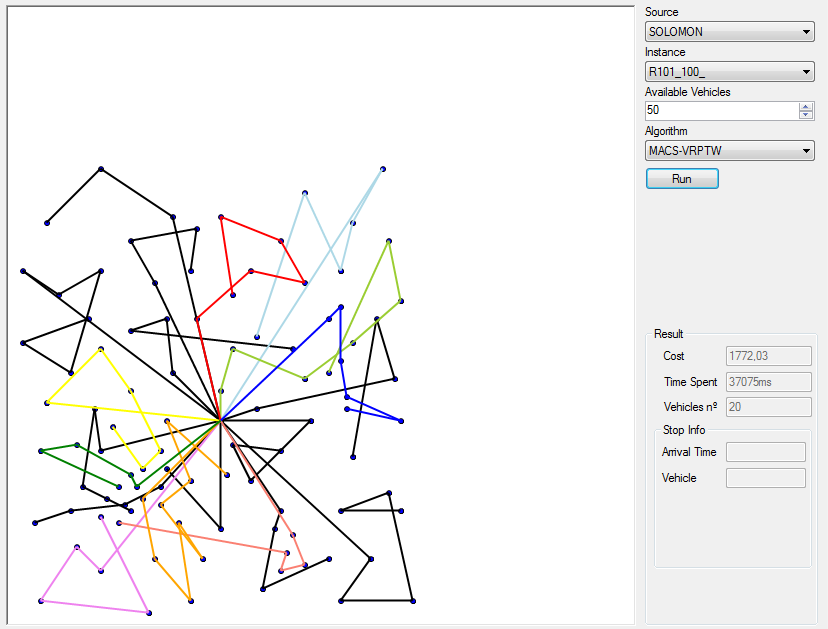


Figura 21 - Instância R101, com o algoritmo MACS-VRPTW alcançou o custo de 1172,03.

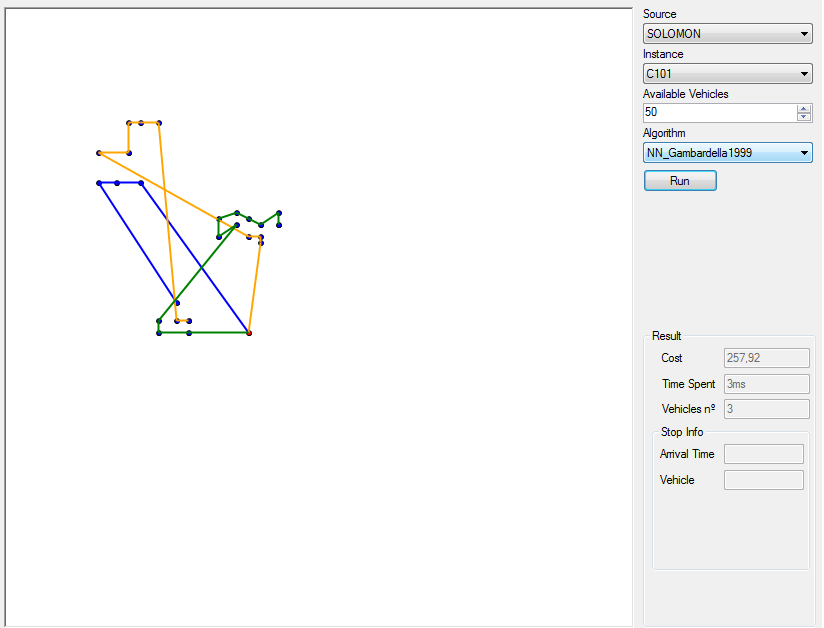


Figura 22 - Instância C101 (reduzido para 25), com o algoritmo NN\_Gambardella alcançou o custo de 257.92.

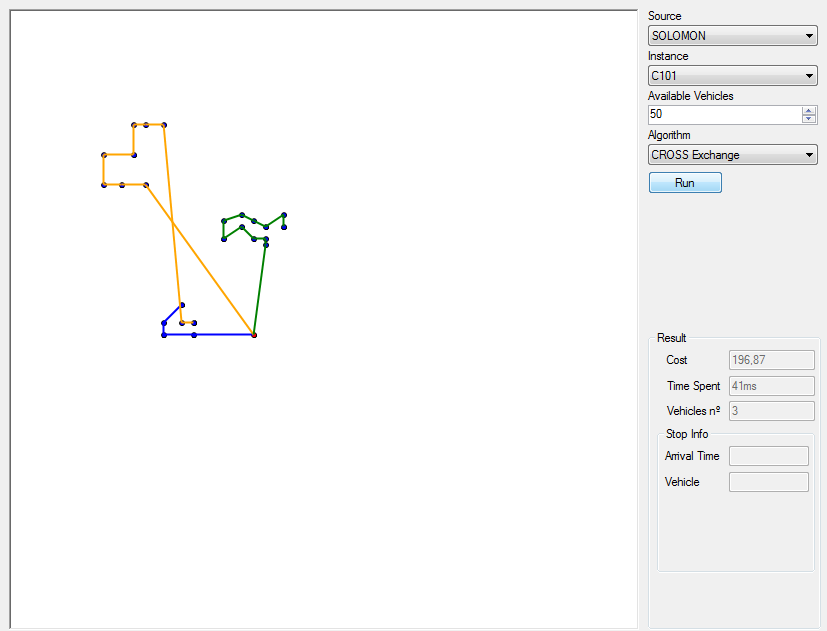


Figura 23 - Instância C101 (reduzido para 25), com o algoritmo CROSS-EXCHANGE alcançou o custo de 196.87.

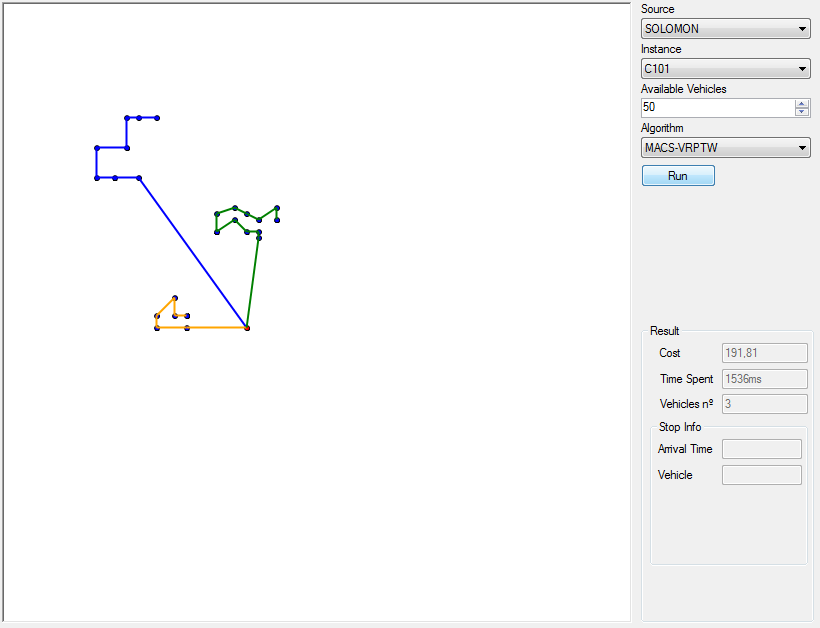


Figura 24 - Instância C101 (reduzido para 25), com o algoritmo MACS-VRPTW alcançou o custo de 191,81.

# Conclusão

Neste trabalho vimos a importância de se resolver um problema de roteamento de veículos e as várias abordagens que podem ser utilizadas para alcançar resultados satisfatórios. Sabe-se, agora, que não existe uma técnica definitiva para a resolução desse problema. Podemos dizer que se queremos resolver o VRP para pequenas entradas o uso de algoritmos exatos é o mais recomendável, pois acha soluções ótimas em um tempo aceitável. E para entradas maiores podemos optar por heurísticas e meta-heurísticas. O primeiro é mais recomendado para um uso restrito, onde se conheça os tipos de entrada que devem ser submetidos ao algoritmo que devem ser compatível à heurística usada, enquanto o segundo resolve para entradas mais gerais, em tempo relativamente curto alcançando soluções próximas das ideais. Esta última talvez seja a aboragem mais usada no mercado em geral. Com isso, toda essa exploração do estado da arte permite ao leitor hoje escolher a técnica mais adequada para o seu problema.

Na parte de estudo do MACS-VRPTW pudemos verificar que ele tem boa facilidade de encontrar soluções viáveis rapidamente. No caso dos algoritmos genéticos, por exemplo, eles podem gerar novas soluções inviáveis, o que às vezes pode ser um problema, pois para espaços de soluções muito restritos a busca se tornaria muito ineficiente. Já em relação à técnicas que usam colônias de formigas, sua geração de soluções retorna sempre uma solução viável. No entanto, mesmo com essa vantagem, vimos na seção de resultados que o MACS-VRPTW já não possui mais os melhores resultados em termos de custo da solução. Sabe-se, porém, que os melhores resultados alcançados nestes artigos provêem de técnicas de meta-heurísticas sejam elas puras ou híbridas, o que indica que o caminho para melhoria dos resultados devem estar nessa classe técnicas.

O algoritmo implementado não alcançou os resultados atingidos pelo artigo. Isso se deve, dentre outros fatores, à falta de clareza do artigo na explicação de algumas sub-técnicas que o algoritmo usava e de alguns procedimentos que não foram bem explicitados. Pretende-se, a partir de agora, trabalhar nesses resultados para alcançar os níveis alcançados pelo artigo, pesquisando formas alternativas de idenficar onde ocorreram os desvios da técnica. Feito isso, pretende-se partir para buscar formas de melhorar o algoritmo usando a mesma técnica-base de colônia de formigas. A inserção de algumas heurísticas dentro o algoritmo podem levar a melhorias de resultados em menor tempo de processamento. A partir dessas tentativas, pretende-se dar continuidade ao estudo com a finalidade de gerar melhores resultados para o benchmark de Solomon e, com isso, avançar cientificamente nesta área da otimização.

# Referências

1. Bernabé Dorronsoro Díaz. VRP Web; Disponível em: <http://neo.lcc.uma.es/radi-aeb/WebVRP/>. 13/12/2011.
2. Alvarenga, G.B. Um algoritmo híbrido para os problemas de roteamento de veículos estático e dinâmico com janela de tempo. Tese de Doutorado do Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Minas Gerais, 2005.
3. Alvarenga, G., Mateus, G. & de Tomi, G. A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. Computers & Operations Research, 2007, Vol. 34, pp. 1561-1584.
4. Ant colony Optimization for vehicle routing problems: from theory to applications;
5. For, Y., Routing, V., With, P., & Windows, T. (1999). MACS-VRPTW : A MULTIPLE ANT COLONY SYSTEM FOR VEHICLE ROUTING PROBLEMS WITH TIME WINDOWS In D . Corne , M . Dorigo and F. Glover , editors New Ideas in Optimization Chapter 5 MACS-VRPTW : A MULTIPLE ANT COLONY SYSTEM FOR. *Optimization*, 1-17.
6. Solomon. Best Know Solutions Identified by Heuristics. Disponível em: <http://web.cba.neu.edu/~msolomon/heuristi.htm>. 13/12/2011.
7. Paolo Toth, Daniele Vigo. The Vehicle Routing Problem. Editora SIAM.
8. Humberto Brandão, Germano Crispim. A hybrid search method for vehicle Routing Problem with time windows. Springer Science, 19/12/2008.
9. Bruce L. Golden, Subramanian Raghavan, Edward A. Wasil. Vehicle Routing Problem: Lastest advances and new challenges. Editora Springer.
10. Cid Carvalho de Souza, Apresentação em slides:Branch-and-Bound.
11. Jens Lysgaard, Clarke & Wright's Savings Algorithm. September 1997.
12. Oliveira, H. C. B. & Vasconcelos, G. A hybrid search method for the vehicle routing problem with time windows. Annals of Operations Research, Springer Netherlands, 2010, 180, 125-144.
13. Jung, S. & Moon, B.R. A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows. Genetic and Evolutionary Computation Conference (GECCO). Morgan Kaufmann, 2002, pp. 1309-1316
14. Siebn.de, Applet: Simulação da aplicação do algoritmo Tabu Search. Disponível em: <http://siebn.de/other/tabusearch/>. 13/12/2011.
15. Clarke, G. & Wright, J.W.: "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", Operations Research, Vol. 12, 1964, pp. 568-581.
16. Aplicação de VRP para plataforma Windows. Disponível em: <http://vrp.sourceforge.net/>. 13/12/2011.
17. Kunigami, Algoritmo de Branch and Cut. Disponível em: <http://kuniga.wordpress.com/2010/10/15/algoritmo-de-branch-and-cut/>. 13/12/2011.
18. Rochat, Y. & Taillard, R.D. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics, 1995, Vol. 1, pp. 147-167.
19. Cordeau, J. F. Laporte, G. & Mercier, A. A unified Tabu Search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society, 2001, Vol. 52, pp. 928-936.