# UM ALGORITMO BASEADO EM COLÔNIA DE FORMIGAS PARA RESOLUÇÃO DO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELA DE TEMPO

**Marcelo Caramuru Pimentel Fraga**
Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG
Av. Amazonas, 7675 – Nova Gameleira / Belo Horizonte – MG / 30510-000
caramuru@hotmail.com

**Sérgio Ricardo de Souza**
Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG
Av. Amazonas, 7675 – Nova Gameleira / Belo Horizonte – MG / 30510-000
sergio@dppg.cefetmg.br

**Marcone Jamílson Freitas Souza**
Universidade Federal de Ouro Preto - UFOP
Ouro Preto – MG / 35400-000
marcone@iceb.ufop.br

**Tatiana Alves Costa**
Universidade Federal de Ouro Preto - UFOP
Ouro Preto – MG / 35400-000
tatiana@iceb.ufop.br

**Resumo**

Este trabalho apresenta uma metodologia baseada na metaheurística Ant Colony Optimization para resolver o Problema de Roteamento de Veículos com Janelas de Tempo. A resolução deste problema consiste em determinar as rotas de custo mínimo, que devem ser executadas por uma frota de veículos de mesma capacidade, para atender à demanda de um conjunto de clientes por um determinado produto. Para cada consumidor, o atendimento somente é possível dentro de um intervalo de tempo, chamado janela de tempo. A metodologia desenvolvida pode ser dividida em uma fase de construção e outra de refinamento. Na fase de construção a geração de uma solução inicial é feita através do algoritmo ACO e, posteriormente, refinada através de um método de busca local. Como o processo é auto-catalítico, as iterações seguintes aproveitam os resultados gerados pelas iterações anteriores, ajudando na convergência e produzindo melhores resultados a cada passo. A metodologia desenvolvida foi testada em um conjunto de 168 problemas-teste da literatura, proposto por Solomon, obtendo, em grande parte das instâncias testadas, um bom desempenho e, em alguns casos, resultados melhores que os relatados na literatura.
**Palavras-Chaves:** Colônia de formigas, metaheurísticas, roteamento de veículos.

**Abstract**

In this paper, we present an hybrid methodology called Ant-L, based on the Ant Colony Optimization meta-heuristic, to solve the Vehicle Routing Problem with Time Windows. The solution of this problem consists in determining the routes of minimum cost for a fleet of vehicles, of same capacity, in charge to satisfy the demand of a set of customers which the attendance is only possible inside a specific time interval. The developed methodology can be divided in two phases: construction and refinement. In the construction phase, the generation of an initial solution is made through ACO meta-heuristic. The generated solution goes to the

refinement phase through local search. The process is autocatalytic, i.e, the following iterations uses information from the previous generated results, helping in convergence and accuracy. The developed methodology was benchmarked in a set of instances created by Solomon to test efficiency on VRPTW algorithms, generating some better results than those found in literature.
**Keywords:** Ant colony system, vehicle routing, meta-heuristic.

## 1. INTRODUÇÃO

O Problema de Roteamento de Veículos com Janela de Tempo (PRVJT) é uma variante do problema clássico de roteamento de veículos (PRV), na qual o serviço de atendimento em cada consumidor deve ser iniciado dentro de um intervalo de tempo denominado janela de tempo. O PRVJT pertence à classe de problemas NP-difícil, o que significa dizer que não existem algoritmos eficientes que encontrem a solução ótima para esse problema em um tempo computacional viável. Assim, o uso de métodos exatos para solução do PRVJT é restrito a instâncias de menor porte, devido ao tempo computacional envolvido. Em contrapartida, métodos heurísticos têm sido amplamente utilizados em problemas de diversas ordens, sofrendo, no entanto, das dificuldades inerentes à indefinição da qualidade da solução determinada. Uma revisão de métodos heurísticos pode ser encontrada em Blum & Roli (2003).

Este trabalho propõe uma metodologia, baseada na metaheurística Colônia de Formigas, para solucionar o Problema de Roteamento de Veículos com Janelas de Tempo (PRVJT). Os algoritmos de Colônia de Formigas (ACO ou *Ant Colony Optimization*) foram desenvolvidos a partir da observação do comportamento de formigas reais. Ao longo dos anos, têm sido aplicados em problemas de diversas classes, como problema do caixeiro viajante Gambardella (1996), Stutzle (1999), problemas de roteamento de veículos Gambardella (1999), de alocação de recursos, dentre outros. De acordo com Dorigo (1999), os algoritmos ACO são uma das mais eficientes aplicações de algoritmos biologicamente inspirados (*Swarm Intelligence*). Uma revisão recente da metaheurística ACO pode ser encontrada em Dorigo (2005).

O foco deste trabalho é apresentar a metodologia Ant-L, desenvolvida a partir da metaheurística ACO, de forma a produzir resultados competitivos com os melhores encontrados na literatura para a classe de problemas do PRVJT. Este artigo está organizado da seguinte forma: a seção 2 apresenta uma visão geral do Problema de Roteamento de Veículos. A metaheurística ACO é descrita na seção 3. A seção 4 contém a metodologia utilizada no artigo. Na seção 5 são demonstrados os resultados computacionais. Por fim, a seção 6 contém as conclusões.

## 2. O PROBLEMA DE ROTEAMENTO DE VEÍCULOS

A formulação mais simples do PRV é denominada Problema de Roteamento de Veículos Capacitados (PRVC). Neste problema, uma frota de veículos, localizada inicialmente em um depósito, deve atender a um conjunto de consumidores que possuem diferentes demandas por produtos a serem distribuídos pela frota. A única restrição presente no PRVC é a capacidade de carga dos veículos que, obviamente, não pode ser violada. As demandas são conhecidas *a priori* e não podem ser fracionadas em sua entrega, ou seja, cada consumidor não pode ser visitado mais de uma vez para entrega de produtos. Todos os veículos possuem a mesma capacidade de carga e suas rotas iniciam e terminam no depósito.

O Problema de Roteamento de Veículos com janelas de tempo (PRVJT) é uma extensão do PRVC na qual, além da restrição de capacidade, são adicionadas restrições

relacionadas ao horário em que cada consumidor exige ser atendido. Para cada consumidor i, é associado um intervalo de tempo [ai, bi], chamado de janela de tempo, que indica o horário em que deve ser iniciado o atendimento e, além disso, um tempo de serviço ($s_i$), que indica o período de tempo no qual o veículo deve aguardar a conclusão das tarefas. O PRVJT consiste em designar um conjunto de rotas que atenda a essas restrições, minimize o número total de veículos utilizado e o custo total da rota.

Dentre as estratégias mais comumente utilizadas para solucionar o PRVJT, destacam-se os métodos de programação matemática, os métodos heurísticos e as metaheurísticas. Para solucionar o PRVJT utilizando-se algum método de programação matemática, é necessário que o problema seja expresso através de algum modelo matemático que o represente. A maioria dos métodos matemáticos desenvolvidos para o PRVJT tem, como princípio básico, relaxar alguma das restrições do modelo matemático, transformando o problema de roteamento em problemas mais simples, geralmente problemas de atribuição, para os quais já são conhecidos algoritmos eficientes de solução.

Métodos de programação matemática, quando aplicados a problemas como o PRV, somente são aplicados a instâncias de pequeno porte (isto é, com número reduzido de consumidores) ou, ainda, quando tratam alguns casos particulares desse problema. Nas demais situações, o tempo computacional gasto para encontrar a solução ótima para o problema torna a aplicação destes algoritmos praticamente inviável.

Devido às limitações apresentadas no uso exclusivo de métodos exatos, surge a necessidade da utilização de técnicas heurísticas para solucionar problemas combinatoriais como o PRV. Observa-se, contudo, que a solução encontrada por uma heurística não é, seguramente, a ótima. Porém, uma boa heurística encontra soluções satisfatoriamente próximas da ótima.

Como evolução das heurísticas e visando tentar escapar de mínimos locais encontrados durante a solução de problemas de otimização, foram propostas as chamadas metaheurísticas, que constituem, atualmente, em um dos principais focos de interesse no que se refere a métodos de solução para problemas de otimização combinatória.

## 3. ALGORITMOS DE COLÔNIA DE FORMIGAS

Os algoritmos baseados em Colônia de Formigas foram propostos inicialmente em Dorigo (1991) (veja, para uma revisão completa, Dorigo (2005)), através de um algoritmo chamado *Ant System*, inspirado no comportamento real das formigas. Segundo esse mesmo artigo, as formigas são capazes de encontrar o caminho mais curto entre a fonte de alimento e o formigueiro, através da cooperação entre os indivíduos e comunicação indireta. Inicialmente, elas exploram, aleatoriamente, a área ao redor do formigueiro à procura de comida. Enquanto se deslocam, depositam sobre o solo uma substância volátil chamada feromônio, que as auxilia a encontrar o caminho de volta ao formigueiro. Desta forma, quando uma formiga estabelece uma trilha ou caminho entre a fonte de alimento e o formigueiro, o caminho percorrido ficará marcado por um rastro desta substância. As demais formigas à procura de alimento detectam a presença de feromônio no solo e tendem a escolher o caminho com a maior concentração do mesmo. As formigas que escolheram o caminho mais curto farão o percurso em menor tempo e o rastro de feromônio será reforçado com uma freqüência maior que nos caminhos mais longos. Assim, os caminhos mais eficientes, ou seja, de menor distância percorrida, receberão maior quantidade de feromônio e tenderão a ser os mais escolhidos. Por ser uma substância volátil, a evaporação do feromônio evita que, com o tempo, um caminho que não esteja sendo mais utilizado continue a influenciar a decisão das formigas.

### 3.1. ALGORITMO *ANT SYSTEM*

O algoritmo *Ant System* (AS) foi o primeiro algoritmo proposto que usava a formulação de Colônia de Formigas (ACO), Dorigo (1991). Ele combina o uso de informações características do problema, conhecidas *a priori* e imutáveis, com os próprios resultados gerados, ajudando na convergência dos mesmos e tornando o processo, nesse sentido, auto-catalítico. A cada iteração, as formigas decidem, de acordo com uma medida probabilística, qual o caminho a seguir dentre aqueles não visitados. Cada formiga tem uma lista tabu, que armazena os caminhos percorridos por ela e previne que um caminho seja visitado pela formiga mais de uma vez. A probabilidade de escolha de um caminho é proporcional ao rastro de feromônio e à atratividade do mesmo - que varia de acordo com o tipo e a modelagem do problema em que o algoritmo está sendo aplicado. Se a formiga já tiver passado pelo caminho, a probabilidade de escolha é zero; caso contrário, é positiva. A escolha do caminho a ser seguido pela formiga é feita segundo a expressão (1) [Dorigo et al, (1991)].

$$p_k(r,s) = \begin{cases} \dfrac{[\tau_{(r,s)}]^{\alpha} \cdot [\eta_{(r,s)}]^{\beta}}{\sum_{u \notin M_k} [\tau_{(r,u)}(t)]^{\alpha} \cdot [\eta_{(r,u)}]^{\beta}}, & \text{se } s \notin M_k \\ 0, & \text{caso contrário} \end{cases} \qquad (1)$$

Em (1), tem-se que:

- $p_k(r,s)$: probabilidade da formiga k mover-se de nó r para nó s;
- $\tau_{(r,s)}$: quantidade de feromônio associado ao arco (r,s);
- $\eta_{(r,s)}$: atratividade do arco (r,s);
- U: conjunto de arcos não visitados;
- $M_k$: lista tabu da formiga k;
- $\alpha$ e $\beta$: parâmetros de controle da influência de $\tau_{(r,s)}$ e $\eta_{(r,s)}$. Variam na faixa [0, 1].

Caso o arco não tenha sido percorrido por formigas, a variação de feromônio no mesmo é zero; caso contrário, é positiva. Ao final de cada iteração, uma taxa de evaporação remove parte do feromônio, reduzindo a quantidade da substância nos arcos. Isso evita que as formigas fiquem presas em ótimos locais e, ao mesmo tempo, diminui a probabilidade de escolha de arcos que não foram utilizados recentemente. A expressão que determina o nível de evaporação no algoritmo *Ant System* é dada em (2) [Dorigo et al, (1991)].

$$\tau_{(r,s)}(t+1) = (1-\rho) \cdot \tau_{(r,s)}(t) + \Delta\tau_{(r,s)}(t) \qquad (2)$$

Nesta expressão, tem-se que:

- v: taxa de evaporação;
- $\tau_{(r,s)}(t)$: quantidade de feromônio associado ao arco (r, s) na iteração t.
- $\Delta\tau_{(r,s)}(t)$: variação do feromônio no arco (r, s) na iteração t.

A quantidade de feromônio a ser depositada nos arcos varia de acordo com a qualidade da solução (distância percorrida). Isso é representado através de (3) [Dorigo et al, (1991)].

$$\begin{cases} \Delta\tau^{k}_{(r,s)} = 0, \text{ se a formiga k não passar pelo arco (r,s);} \\ \Delta\tau^{k}_{(r,s)} = \dfrac{Q}{L^{k}(t)}, \text{ se a formiga k passar pelo arco (r,s).} \end{cases} \quad (3)$$

Em (3), tem-se que:

- Q: quantidade de feromônio de cada formiga;
- $L^k(t)$: distância percorrida pela formiga k na iteração t;
- $\Delta\tau^{k}_{(r,s)}$: quantidade de feromônio no arco (r, s) depositado pela formiga k.

### 3.2. ALGORITMO *ANT COLONY SYSTEM*

O algoritmo *Ant Colony System* (ACS) foi introduzido em Gambardella (1996), Dorigo (1997a) e Dorigo (1997b), de forma a melhorar a performance do algoritmo *Ant System*. Uma das modificações do ACS em relação ao AS é um novo critério de seleção, chamado pseudo-randômico-proporcional, apresentado em (4), na qual o parâmetro $q_0$ controla a formação das soluções, se de forma probabilística ou de forma gulosa.

$$s = \begin{cases} \arg\max_{u \notin M} \left\{ [\tau_{(r,u)}]^{\alpha} \cdot [\eta_{(r,u)}]^{\beta} \right\}, & \text{se } q \le q_0 \\ \text{exp}ressão(1), & \text{caso contrário} \end{cases} \quad (4)$$

Um número aleatório q é gerado e comparado ao parâmetro $q_0$, cada vez que a formiga necessita escolher o próximo nó para se mover. O algoritmo funciona de forma gulosa se o valor de q for menor que o parâmetro $q_0$. Desta forma, a geração da solução valoriza o aprendizado gerado pela deposição do feromônio. Caso o valor de q seja maior que $q_0$, a geração da solução será de feita de forma probabilística, segundo a expressão (1. O processo de deposição e evaporação do feromônio no ACS foi modificado em relação ao AS, de modo que a matriz de feromônio seja atualizada de forma global e de forma local. A atualização global é feita para recompensar apenas os caminhos pertencentes à melhor solução global, ou seja, a cada iteração, após todas as formigas terem gerado suas soluções, apenas a formiga que gerou a melhor solução desde o início do algoritmo irá depositar feromônio. De acordo com Dorigo (1997b), a atualização global do ACS evita a lenta convergência dos resultados, concentrando as buscas na vizinhança da melhor solução. Segundo Ellabib (2003), a fase final do algoritmo AS (fase de evaporação) é substituída, no ACS, por uma atualização local do feromônio, com a diferença de ser aplicada durante a construção das soluções. Cada vez que uma formiga se move de um arco para outro, a quantidade de feromônio associado ao arco é reduzida. O resultado da atualização local é a modificação dinâmica da atratividade dos caminhos. Logo, cada vez que um arco é percorrido por uma formiga, ele se torna ligeiramente menos desejável para as demais. Um processo chamado *Daemon Actions* pode ser usado, opcionalmente, para auxiliar no processo centralizado de tomada de decisões pelo ACS, em situações como ativar um procedimento de busca local ao final de cada iteração ou depositar uma quantidade extra de feromônio sobre a melhor solução. Esses são exemplos de decisões que não podem ser tomadas pelas formigas individualmente. A figura 1 mostra o pseudo-código do algoritmo *Ant Colony System*.

```
1 Repita
2    Aleatoriamente posicione m formigas em n nós
```

```
3    Para nó = 1 até n faça
4      Para formiga = 1 até m faça
5        Escolha o próximo nó para se mover
6        Aplique a regra de atualização local do feromônio
7      fim para
7      Calcule a distância percorrida pela formiga m
8    fim para
9    Executar Daemon-Actions (opcional)
10   Aplique a atualização global do feromônio
11 Até Condição-de-saída
```

**Figura 1**: Pseudo-código *Ant Colony System.*

## 4. METODOLOGIA ANT-L

A metodologia Ant-L proposta consiste em, inicialmente, definir uma forma de representação das soluções geradas. Em seguida, são definidos os movimentos de realocação e a estrutura de vizinhança utilizada no algoritmo. Além disso, também é definida uma função de avaliação, para controle da qualidade das soluções geradas. Cabe ressaltar que a metodologia Ant-L foi elaborada e implementada especificamente para o PRVJT. Sua descrição geral é realizada a seguir e o pseudo-código associado está apresentado na figura 3.

### 4.1. FORMA DE REPRESENTAÇÃO DE UMA SOLUÇÃO

A representação da solução no PRVJT consiste em um conjunto de rotas, geradas através do algoritmo utilizado para a solução do problema, que possuem um determinado custo associado, atribuído por uma função de avaliação, definida em (5.1). A forma de representação de uma solução é demonstrada na tabela 1. Nela, cada campo constitui uma rota, tendo custo dado pelo campo específico.

| | | | |
|---|---|---|---|
| 0-11-19-10-0 | 0-7-8-0 | 0-2-18-6-13-0 | 0-5-16-17-0 |
| 0-14-15-4-0 | 0-12-3-0 | 0-9-20-1-0 | Custo = 513.3 |

**Tabela 1**: Forma de representação de uma solução.

### 4.2. FUNÇÃO DE AVALIAÇÃO

A função de avaliação utilizada é lexicográfica, ou seja, formada por um conjunto de objetivos distintos, classificados segundo um critério de prioridades. Nesta metodologia, dois objetivos são considerados, na seguinte ordem de prioridade: primeiro, deseja-se minimizar o número de veículos utilizados e segundo, minimizar a distância total percorrida. Cada nova solução gerada é comparada com a anterior, avaliando-se primeiramente o critério de maior prioridade e, somente caso a avaliação com relação a este critério produza um resultado melhor ou igual à solução anterior, o próximo objetivo é avaliado.

### 4.3. MOVIMENTOS DE REALOCAÇÃO

Um movimento de realocação consiste em retirar um consumidor de uma rota qualquer e inseri-lo em uma outra posição, que pode ser tanto dentro da mesma rota quanto em uma outra rota distinta. O movimento de realocação dentro de uma mesma rota é chamado

de realocação Intra-rota. Já o movimento de realocação envolvendo duas rotas distintas é denominado realocação Inter-rotas. Na figura 4, o consumidor 4 estava situado entre o consumidor 10 e o 0 (depósito) e foi realocado para entre os consumidores 6 e 3 da mesma rota.

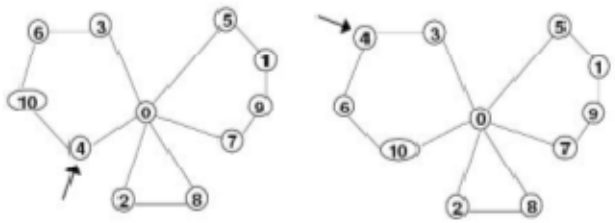

**Figura 4**: Movimento de realocação Intra-rota.

O movimento de realocação envolvendo duas rotas distintas é chamado Inter-Rotas. Na figura 5, o consumidor 4 estava situado entre os consumidores 10 e 0 (depósito) de uma rota e foi realocado para entre os consumidores 2 e 8 de uma outra rota.

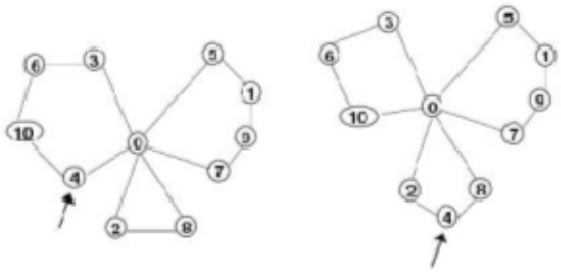

**Figura 5**: Movimento de realocação Inter-rotas.

A estrutura de vizinhança considerada neste trabalho combina os dois tipos de movimentos anteriores. Assim, nesta estrutura, um vizinho s' é obtido de uma solução s através de movimentos de realocação intra-rotas ou inter-rotas.

### 4.4. DESCIDA COM PRIMEIRO DE MELHORA

O método tradicional de descida/subida requer que toda a vizinhança seja explorada a cada iteração. Isso pode gastar muito tempo computacional em instâncias maiores. O método de descida com primeira melhora interrompe a busca e move para a próxima vizinhança assim que um melhor vizinho é encontrado. Desta forma, apenas no pior caso toda a vizinhança é explorada. Dada uma função f, uma solução s e uma vizinhança N(.), a figura 6 mostra o procedimento de descida com primeira melhora.

```
procedimento Primeira_Melhora ( f(.), N(.), s);
1.  V = { s' ∈ N(s) };
2.  enquanto ( |V| > 0 ), faça
3.      selecione s' e V;
4.      se f(s') < f(s);
5.          então s ← s';
6.              V = { s' ∈ N(s) };
7.          senão V ← V – {s'};
8.      fim se
9.  fim enquanto
10. retorne s;
```

**fim** Primeira_Melhora

**Figura 6**: Pseudo-código da Descida com primeiro de melhora.

### 4.5. MODIFICAÇÕES FEITAS AO ALGORITMO ACS PADRÃO

No intuito de produzir melhores resultados, algumas modificações são introduzidas no algoritmo ACS apresentado na figura 1. Em lugar de gerar um conjunto fixo de soluções por iteração, são geradas novas soluções, até que surja uma solução melhor que a melhor solução da iteração anterior. Isto é feito até um máximo valor percentual, relativo ao número de consumidores total da instância (foi adotado, para fins de implementação, o valor de 10%). O processo de atualização local e global do feromônio foi modificado, de forma a permitir um maior controle dos parâmetros envolvidos e da matriz de máximos e mínimos de feromônio. Cada formiga possui uma quantidade de feromônio Q, relacionada à maior e à menor distância da instância e à quantidade de iterações que o algoritmo será executado.

A cada iteração, uma lista restrita de consumidores candidatos (LRCC) é gerada, contendo todos os consumidores (com exceção do depósito) da instância a ser solucionada. Além disso, uma lista tabu para nascimento de novas formigas (LTPNNF) e uma lista tabu geral das formigas (LTGF) são geradas contendo, inicialmente, apenas o depósito. Um vetor de rotas geral (VRG) é criado e, inicialmente, não contém nenhuma rota. Em seguida, uma formiga é atribuida a um consumidor qualquer, escolhido aleatoriamente dentre os consumidores da LRCC, de modo que o consumidor selecionado nunca pertencerá a LTPNNF. A LTPNNF evita que, em uma futura iteração, uma formiga seja colocada ou em uma posição que não permita a formação de novas rotas ou em uma posição já colocada anteriormente.

Um vetor de rotas para a formiga é então iniciado e, caso o consumidor escolhido já pertença a alguma rota, o vetor rota da formiga (VRF) receberá, desse modo, a rota do consumidor. A formiga escolhe, dentre os consumidores da LRCC, para qual consumidor se moverá, segundo expresso por (4). Cada formiga possui sua própria lista tabu (LTF) que, inicialmente, é uma cópia da LTGF e para onde serão movidos os consumidores analisados. A função da LTGF é evitar que as formigas visitem consumidores que estejam no meio de rotas, ou seja, que não estejam ou na primeira ou na última posição dos vetores de rotas das formigas. A união entre rotas geradas pelas formigas só é possível quando os consumidores envolvidos estão nas extremidades de cada rota. Quando o consumidor escolhido pela formiga já pertencer a uma rota, o algoritmo selecionará a rota do consumidor escolhido e tentará realizar a união.

A cada tentativa mal-sucedida de movimento da formiga (violação de qualquer uma das restrições existentes), a LTF receberá o consumidor envolvido, para que ela não retorne aos consumidores já analisados. Caso o movimento da formiga de um consumidor para outro respeite todas as restrições, a LTGF e o VRG são atualizados, e a LTF recebe novamente a LTGF. A formiga então, move-se para o último consumidor da rota gerada e continua seus movimentos, até que não existam mais movimentos possíveis, ou seja, até que todos os consumidores pertençam à LTF. A formiga, então, é eliminada; sua rota gerada fica armazenada no VRG; a LRCC e a LTPNNF são atualizadas de acordo com a LTGF; e uma nova formiga será gerada. Quando não existirem mais consumidores na LRCC, o custo da solução gerado pelo conjunto de formigas é analisado e a matriz de feromônio é atualizada.

```
1  CustoMelhorSolução ← 0
2  Enquanto < > CondiçãoSaída
3     Inicia parâmetros
4     CustoMelhorSoluçãoAnterior ← ∞
5     ContaSoluções ← 0
```

```
6    Enquanto ContaSoluções ≤ % * NúmeroConsumidoresInstância OU
     CustoSoluçãoGerada ≤ CustoMelhorSoluçãoAnterior
7      ListaRestritadeConsumidoresCandidatos (LRCC) ← Consumidores da instância
8      ListaTabuParaNascimentodeNovasFormigas (LTPNNF) ← Depósito
9      Lista Tabu Geral das Formigas (LTGF) ← Depósito
10     Enquanto LRCC < > 0
11       Selecione aleatoriamente um dos consumidores da LRCC
12       Posicione uma formiga no consumidor selecionado
13       Lista Tabu da Formiga (LTF) ← LTGF
14       Enquanto LTF < NúmeroConsumidoresInstância
15         De acordo com o critério de seleção, escolha o próximo movimento
16         Atualizar: LTF, LRCC, VGR
17       Fim enquanto
18     Fim enquanto
19     Calcule o CustoSoluçãoGerada
20     Se CustoSoluçãoGerada < CustoMelhorSoluçãoAnterior
21       Então saia do laço enquanto (1)
22       Senão ContaSoluções ← ContaSoluções + 1
23     Fim se
24   Fim enquanto
25   Aplique Busca Local na SoluçãoGerada
26   Se CustoSoluçãoGerada < CustoMelhorSolução
27     Então MelhorSolução ← SoluçãoGerada
28     CustoMelhorSolução ← CustoSoluçãoGerada
29   Fim se
30 Fim enquanto
```

**Figura 3**: Ant-L pseudo-código.

## 5. RESULTADOS COMPUTACIONAIS

O algoritmo Ant-L foi implementado em linguagem C++, utilizando o compilador Builder 5.0 da Borland, e executado em um computador Pentium 4 2.4 GHz, com 512 MB de memória RAM, sob plataforma Windows XP Pro. Foram utilizadas, para teste, as instâncias introduzidas por Solomon (1987), usadas amplamente como referência de desempenho de algoritmos para o PRVJT. Essas instâncias são formadas por três grupos distintos de consumidores: C - os consumidores estão distribuídos geograficamente em grupos de consumidores próximos uns dos outros; R - os consumidores são distribuídos aleatoriamente e distantes uns dos outros; e RC - uma mistura dos dois grupos anteriores.

O algoritmo desenvolvido foi executado uma única vez para cada instância, obtendo apenas uma idéia inicial do comportamento da metodologia, mas demonstrando, inicialmente, resultados promissores. Na tabela 2, são mostrados os melhores resultados obtidos pela aplicação da metodologia Ant-L proposta nas instâncias Solomon de 100 consumidores. Os melhores resultados encontrados na literatura são reportados na tabela nas colunas “Nº_Vei Literat” e “Dist Literat“ e foram obtidos de Sintef. As colunas “Nº_Vei Ant-L” e “Dist Ant-L” mostram os melhores resultados produzidos pela metodologia proposta.

Para o grupo C de instâncias, todos os ótimos globais são conhecidos na literatura. Em todas as instâncias do grupo o número de veículo foi equivalente ao da literatura, e em 11 das 17 instâncias o algoritmo Ant-L foi capaz de encontrar a solução ótima. Em duas instâncias do subgrupo RC1 e em seis de oito instâncias do subgrupo RC2, o número de veículos foi equivalente ao relatado na literatura. Para o subgrupo R1 duas instâncias tiveram o número de veículos equivalente ao relatado na literatura e outra obteve um resultado melhor, ou seja,

com um menor número de veículos. O algoritmo Ant-L alcançou bons resultados o subgrupo R2. Em 7 das 11 instâncias existentes, os resultados obtidos tiveram o mesmo número de veículos dos relatados na literatura.

| Inst. | Nº_Vei Literat | Dist Literat | Nº_Vei Ant-L | Dist Ant-L | Inst. | Nº_Vei Literat | Dist Literat | Nº_Vei Ant-L | Dist Ant-L |
|---|---|---|---|---|---|---|---|---|---|
| C101 | 10 | 828,94 | 10 | 828,94 | RC204 | 3 | 798,41 | 3 | 859,33 |
| C102 | 10 | 828,94 | 10 | 843,8 | RC205 | 4 | 1297,19 | 4 | 1509,34 |
| C103 | 10 | 828,06 | 10 | 860,78 | RC206 | 3 | 1146,32 | 4 | 1214,89 |
| C104 | 10 | 824,78 | 10 | 871,6 | RC207 | 3 | 1061,14 | 3 | 1131,11 |
| C105 | 10 | 828,94 | 10 | 828,94 | RC208 | 3 | 828,14 | 3 | 944,547 |
| C106 | 10 | 828,94 | 10 | 828,94 | R101 | 19 | 1645,79 | 18 | 1628,48 |
| C107 | 10 | 828,94 | 10 | 828,94 | R102 | 17 | 1486,12 | 17 | 1449 |
| C108 | 10 | 828,94 | 10 | 828,94 | R103 | 13 | 1292,68 | 14 | 1267,26 |
| C109 | 10 | 828,94 | 10 | 923,96 | R104 | 9 | 1007,24 | 10 | 1050,94 |
| C201 | 3 | 591,56 | 3 | 591,56 | R105 | 14 | 1377,11 | 14 | 1454,43 |
| C202 | 3 | 591,56 | 3 | 591,56 | R106 | 12 | 1251,98 | 13 | 1272,15 |
| C203 | 3 | 591,17 | 3 | 603,37 | R107 | 10 | 1104,66 | 11 | 1152,22 |
| C204 | 3 | 590,6 | 3 | 607,1 | R108 | 9 | 960,88 | 10 | 993,004 |
| C205 | 3 | 588,88 | 3 | 588,88 | R109 | 11 | 1194,73 | 12 | 1222,98 |
| C206 | 3 | 588,49 | 3 | 588,49 | R110 | 10 | 1118,59 | 11 | 1212,93 |
| C207 | 3 | 588,29 | 3 | 588,29 | R111 | 10 | 1096,72 | 11 | 1113,56 |
| C208 | 3 | 588,32 | 3 | 588,32 | R112 | 9 | 982,14 | 10 | 1041,65 |
| RC101 | 14 | 1696,94 | 15 | 1670,06 | R201 | 4 | 1252,37 | 4 | 1308,15 |
| RC102 | 12 | 1554,75 | 13 | 1508,68 | R202 | 3 | 1191,7 | 4 | 1151,16 |
| RC103 | 11 | 1261,67 | 12 | 1358,27 | R203 | 3 | 939,54 | 3 | 1019,68 |
| RC104 | 10 | 1135,48 | 10 | 1247,06 | R204 | 2 | 825,52 | 3 | 798,661 |
| RC105 | 13 | 1629,44 | 15 | 1594,21 | R205 | 3 | 994,42 | 3 | 1081,77 |
| RC106 | 11 | 1424,73 | 13 | 1429,66 | R206 | 3 | 906,14 | 3 | 976,047 |
| RC107 | 11 | 1230,48 | 11 | 1364,29 | R207 | 2 | 890,61 | 3 | 865,821 |
| RC108 | 10 | 1139,82 | 11 | 1185,49 | R208 | 2 | 726,75 | 2 | 776,077 |
| RC201 | 4 | 1406,91 | 4 | 1567,78 | R209 | 3 | 909,16 | 3 | 1004,32 |
| RC202 | 3 | 1365,645 | 4 | 1267,27 | R210 | 3 | 939,34 | 3 | 999,195 |
| RC203 | 3 | 1049,62 | 3 | 1190,58 | R211 | 2 | 892,71 | 3 | 864,182 |

**Tabela 2**: Instâncias Solomon - 100 consumidores

Na tabela 3 são mostrados os melhores resultados obtidos por métodos heurísticos diversos nas instâncias Solomon de 100 consumidores. Nesta tabela, (a) é o valor médio do número de veículos da solução; (b) é a média da distância total percorrida; e (c) é a média do tempo computacional gasto, em segundos. Nesta tabela é apresentada, também, na coluna "Total", a soma de todos os valores de cada grupo de instâncias. Os dados da tabela 3 foram obtidos de Chen (2005).

| Problem Type | R1 | R2 | C1 | C2 | RC1 | RC2 | Total |
|---|---|---|---|---|---|---|---|
| **TS-P** | 12,6<br>1294,7<br>639 | 3,1<br>1185,9<br>722 | 10<br>861<br>435 | 3<br>602,5<br>431 | 12,6<br>1465<br>586 | 3,4<br>1476,1<br>662 | 427<br>64679<br>32957 |
| **TS-T** | 12,17<br>1209,35<br>13774 | 2,82<br>980,27<br>20232 | 10<br>828,38<br>14630 | 3<br>589,86<br>16375 | 11,5<br>1389,22<br>11264 | 3,38<br>1117,44<br>11596 | 410<br>57953<br>833390 |
| **MACS** | 12<br>1217,73<br>1800 | 2,73<br>967,75<br>1800 | 10<br>828,38<br>1800 | 3<br>589,86<br>1800 | 11,63<br>1382,42<br>1800 | 3,25<br>1129,19<br>1800 | 407<br>57525<br>100800 |
| | 11,92 | 2,73 | 10 | 3 | 11,5 | 3,25 | 405 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **GA** | 1221,1<br>- | 975,43<br>- | 828,48<br>- | 589,93<br>- | 1389,89<br>- | 1159,37<br>- | 57523<br>- |
| **HGA** | 13,2<br>1227<br>- | 5<br>980<br>- | 10,1<br>861<br>- | 3,25<br>619<br>- | 13,5<br>1427<br>- | 5<br>1223<br>- | 478<br>59405<br>84000 |
| **SATS** | 13,1<br>1213,16<br>- | 4,6<br>952,3<br>- | 10<br>841,92<br>- | 3,3<br>612,75<br>- | 12,7<br>1415,62<br>- | 5,6<br>1120,37<br>- | 470<br>57799<br>15400 |
| **TESA** | 12,08<br>1215,14<br>1474 | 2,91<br>953,43<br>3882 | 10<br>828,38<br>201 | 3<br>589,86<br>1220 | 11,75<br>1385,47<br>916 | 3,25<br>1142,48<br>2669 | 411<br>57467<br>100639 |
| **Ant-L** | 12,58<br>1238,22<br>1338,58 | 4,25<br>1355,63<br>2352,48 | 10<br>849,43<br>1029,4 | 3<br>593,45<br>578,35 | 12,5<br>1419,72<br>1208,15 | 3,5<br>1210,61<br>1440,413 | 427<br>59139<br>69963 |

**Tabela 3**: Ant-L vs. outras metodologias

## 6. CONCLUSÕES

Neste trabalho, foi apresentado um algoritmo híbrido denominado Ant-L, baseado na metaheurística Ant Colony Optimization para solucionar o Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). Na metodologia desenvolvida, as soluções para o PRVJT são geradas pela metaheurística ACO e refinadas com busca local descida com primeiro de melhora. O método foi avaliado em 56 instâncias teste, produzindo diversos resultados próximos aos melhores relatados na literatura, ressaltando a eficiência do método e de sua implementação na resolução do PRVJT.

## 7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Blum, C. & Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, vol. 35, n. 3, pp. 268-308.

[2] Chen, C.-H. & Ting, C.-J., 2005. Algorithms for vehicle routing and scheduling problems with time windows constraints. *Journal of the Eastern Asia Society for Transportation Studies*, vol. 6, pp. 2822-2836.

[3] Desrochers, M., Desrosiers, J., & Solomon, M., 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, vol. 40, n. 2, pp. 342-354.

[4] Dorigo, M. & Blum, C., 2005. Ant colony optimization theory: A survey. *Theoretical Computer Science*, vol. 344, pp. 243-278.

[5] Dorigo, M., Caro, G. D., & Gambardella, L. M., 1999. Ant algorithms for discrete optimization. *Articial Life*, vol. 5, n. 2, pp. 137-172.

[6] Dorigo, M. & Gambardella, L. M., 1997a. Ant colonies for the traveling salesman problem. *BioSystems*, vol. 43, pp. 73-81.

[7] Dorigo, M. & Gambardella, L. M., 1997b. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53-66.

[8] Dorigo, M., Maniezzo, V., & Colorni, A., 1991. The ant system: An autocatalytic optimization process. Technical Report 91-016 Revised. Dipartimento di Elettronica, Politécnico di Milano, Italy.

[9] Ellabib, I., Basir, O. A., & Calamai, P., 2003. A new ant colony system updating strategy for vehicle routing problem with time windows. *In MIC2003: The Fifth Metaheuristics International Conference*, pp. 18-1 - 18-6.

[10] Gambardella, L. M. & Dorigo, M., 1996. Solving symmetric and asymmetric tsps by ant colonies. *In ICEC96 Proceedings of the IEEE Conference on Evolutionary Computation*, pp. 622-627. IEEE Press.

[11] Gambardella, L. M., Taillard, E., & Agazzi, G., 1999. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. D. & Glover, F., eds, *New Ideas in Optimization*, pp. 63-76. McGraw-Hill.

[12] Solomon, M. M., 1987. Algorithms for vehicle routing and scheduling problems with time windows constraints. *European Journal of Operational Research*, vol. 35, pp. 254-266.

[13] Stutzle, T. & Dorigo, M., 1999. Aco algorithms for the traveling salesman problem. In Miettinen, K., Makela, M. M., Neittaanmaki, P., & Periaux, J., eds, *Recent advances in genetic algorithms, evolution strategies, evolutionary programming, genetic programming and industrial applications*. John Wiley and Sons.

[14] http://www.sintef.no/static/am/opti/projects/top/vrp/bknown.html, 30/06/2006 – 16:20