NECHELLEY ALVES PEREIRA DE LIMA

APLICAÇÃO DA METAHEURÍSTICA COLÔNIA DE FORMIGAS EM UM PROBLEMA DA ENGENHARIA FLORESTAL

LAVRAS – MG

2019

NECHELLEY ALVES PEREIRA DE LIMA

APLICAÇÃO DA METAHEURÍSTICA COLÔNIA DE FORMIGAS EM UM PROBLEMA DA ENGENHARIA FLORESTAL

Trabalho de conclusão de curso apresentado à Universidade Federal de Lavras, como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

Prof. Dr. Dilson Lucas Pereira

Orientador

LAVRAS – MG

2019

NECHELLEY ALVES PEREIRA DE LIMA

APLICAÇÃO DA METAHEURÍSTICA COLÔNIA DE FORMIGAS EM UM PROBLEMA DA ENGENHARIA FLORESTAL

Trabalho de conclusão de curso apresentado à Universidade Federal de Lavras, como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 26 de Junho de 2019.

Prof. Dr. Mayron César de Oliveira Moreira	UFLA
Prof. Dr. Raphael Winckler de Bettio	UFLA

Prof. Dr. Dilson Lucas Pereira
Orientador

LAVRAS – MG
2019

NECHELLEY ALVES PEREIRA DE LIMA

APLICAÇÃO DA METAHEURÍSTICA COLÔNIA DE FORMIGAS EM UM PROBLEMA DA ENGENHARIA FLORESTAL

Trabalho de conclusão de curso apresentado à Universidade Federal de Lavras, como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 26 de Junho de 2019.

Prof. Dr. Mayron César de Oliveira Moreira UFLA
Prof. Dr. Raphael Winckler de Bettio UFLA

Prof. Dr. Dilson Lucas Pereira
Orientador

LAVRAS – MG
2019

AGRADECIMENTOS

Agradeço a meus pais, Adilson e Elizabete, por todos apoio que me deram e incentivo para a conclusão do curso.
Ao meu orientador Dilson, por ter me acolhido e me orientado na conclusão deste trabalho, além do grande crescimento acadêmico que me proporcionou.
Por fim, e não menos importante, agradeço aos meus amigos que compartilharam todos os meus bons e maus momentos durante a graduação.

RESUMO

O presente artigo aborda um problema da engenharia florestal que pode ser interpretado como um problema de roteamento multi período. Encontrar uma solução ótima torna-se difícil, dado o problema pertencer a classe dos problemas *NP-HARD*, com isso foi usada a abordagem de desenvolvimento de heurísticas construtivas e em seguida a aplicação da metaheurística de Colônia de Formigas nas melhores heurísticas desenvolvidas, obtendo assim resultados satisfatórios quando comparado as heurísticas sem Colônia de Formigas e um *framework* genérico de Colônia de Formigas.

Palavras-chave: Heurística. Metaheurística. Colônia de Formigas. Engenharia Florestal. Problema de Roteamento de Veículos Multi Período.

ABSTRACT

This paper addresses a problem of forest engineering that can be interpreted as a multi-period routing problem. Finding an optimal solution becomes difficult, since the problem belongs to the class of the NP-HARD problems, using the development approach of constructive heuristics and then the application of the metaheuristic of Ant Colony in the best heuristics developed, thus obtaining satisfactory results when compared to the heuristics without Ants Colony and a generic Ant Colony framework.

Keywords: Heuristic. Metaheuristics. Ant Colony. Forest engineering. Multi-period Vehicle Routing Problem.

SUMÁRIO

1 INTRODUÇÃO

Existe um problema na engenharia florestal onde companhias florestais, que produzem madeira em diversas fazendas, precisam planejar a execução de atividades que antecedem o plantio nas sub-regiões denominadas talhões, tais como: controle de pestes, irrigação, limpeza, etc. Em geral, existe um conjunto de atividades que devem ser executadas em cada talhão por equipes especializadas. As atividades devem ser executadas em uma sequência pré-definida. As equipes podem fazer parte do corpo de funcionário da própria cia. ou podem ser terceirizadas. É de interesse da cia. florestal que todas as atividades sejam concluídas da forma mais eficiente possível.

Em Ciência da Computação, é comum que se deseje solucionar problemas encontrando uma solução ótima, que seria a melhor opção possível conforme a definição do problema escolhido. O que é ótimo depende do contexto, podendo ser desde a solução que consuma menos recursos a uma que forneça o maior lucro. Porém, em alguns casos, como o conhecido *Travelling Salesman Problem* (TSP) (BLUM; ROLI, 2003), encontrar a solução ótima poderia levar séculos, tendo em vista que se trata de um problema *NP-Hard*.

Devido a isso, uma alternativa encontrada na área são as heurísticas, que têm como objetivo encontrar soluções satisfatórias para a maioria das instâncias do problema em um tempo aceitável. Uma heurística até pode encontrar uma solução ótima em algumas instâncias, porém, não existe garantia disso.

Além do TSP, um outro problema clássico de otimização que se enquadra na categoria de problemas difíceis, é o problema de roteamento de veículos (PRV), que foi primeiramente apresentado em *The Truck Dispatching Problem* (DANTZIG; RAMSER, 1959). Neste problema existem veículos, cada um com uma capacidade de carga, e clientes que demandam de cargas. Todos os veículos saem do mesmo ponto de origem, o depósito, e a solução do problema consiste em descobrir rotas de forma que todos os clientes recebam a carga que precisam e o custo total do caminho percorrido seja o menor possível.

Uma outra variante do problema TSP é o problema de roteamento de veículos multi períodos (FRANCIS; SMILOWITZ; TZUR, 2008), que é semelhante ao PRV, contudo o planejamento não é apenas para um único dia, mas sim uma sequência de dias, e com isso, acrescenta-se agora restrições de quando os clientes querem/podem serem visitados. Por exemplo um dado cliente pode querer/poder receber visitas apenas no dias ímpares.

O problema das companhias florestais, tema deste projeto, pode ser interpretado como um problema de roteamento de veículos multi períodos. As equipes substituem os veículos, os talhões substituem os clientes, ao invés de cargas, cada equipe precisa executar serviços em cada talhão. Os serviços devem ser executadas em uma sequência pré-definida, esta característica não é encontrada em PRVs na literatura. Além disso cada equipe possui uma eficiência diferente para cada atividade de cada cliente. Outra característica dos serviços nesse problema, é a possibilidade de uma execução fracionada, sendo possível acabar um turno sem que a atividade esteja totalmente concluída, porém a mesma deve ser a primeira a ser executada no turno seguinte, e também não é obrigatório que os serviços de um cliente sejam todos executados um após o outro, sendo assim após a execução de um serviço em um cliente o serviço em sequência não precisa ser feito ainda, a equipe pode ir para outro cliente. Além disso o problema florestal é multi períodos, ou seja, a solução será composta pelas rotas juntamente das atividades que as equipes fizeram ao longo de vários dias e não apenas um único dia como no PRV. Deste ponto em diante, o problema da engenharia florestal sera referido como problema de roteamento de atividades sequenciais (PRAS). Uma definição formal do PRAS será apresentada na Seção 2.

Com isso em mente o objetivo do projeto foi desenvolver uma heurística para o PRAS, que fosse capaz de produzir soluções de boa qualidade em um baixo tempo computacional.

Este documento esta organizado da seguinte maneira. Na Seção 2, é apresentada uma definição formal do problema. Na Seção 3, são apresentadas as heurísticas construtivas desenvolvidas e seu resultado. Na Seção 4, é apresentada a meta heurística de colônia de formigas e o desenvolvimento de uma heurística guiada por ela, bem como seus resultados. Nas seções 5 e 6, é discutido a conclusão do presente trabalho e suas possíveis aplicações futuras.

2 DEFINIÇÃO DO FORMAL PROBLEMA

Tem-se um conjunto de clientes, dentre eles um depósito central, representados por um grafo completo $G = (V,E)$, onde $V = \{1,2,...,n\}$. O vértice 1 representa o depósito central e os demais vértices representam os clientes. Para cada aresta $\{u,v\}$ está associado um valor $t_{\{u,v\}} \in \mathbb{R}_+$ que representa o tempo para se locomover entre os vértices u e v.

Uma sequência de atividades $(a_1,a_2,...,a_A)$, $A \in \mathbb{N}$, deve ser realizada em cada um dos clientes. Sabe-se que alguns clientes já tiveram atividades executadas em períodos anteriores ao estado atual do problema. Denota-se por $s(v)$, o índice da primeira atividade a ser executada no cliente v. Assim, assume-se que as atividades $a_1,...,a_{s(v)-1}$ já foram executadas em v, e que nenhuma das atividades $a_{s(v)},...,a_A$ foi executada, sendo que a atividade $s(v)$ é considerada como não iniciada, ou seja, não existe progresso referente a ela.

A execução das atividades é realizada por K equipes, que trabalham em turnos de $T \in \mathbb{R}_+$ unidades de tempo. Tem-se que uma equipe k leva $t_{iv}^k \in \mathbb{R}_+$ unidades de tempo para executar a atividade a_i no cliente v.

O problema consiste em desenvolver um itinerário para a execução de atividades para cada equipe em cada um dos turnos de modo que:

- Em cada turno, cada equipe deixa o depósito, visita alguns clientes, executando atividades em cada talhão visitado, e retornando ao depósito,
- A soma dos tempos das proporções das atividades executadas por cada equipe e dos tempos de deslocamento entre clientes e o depósito central não deve ultrapassar T, o tamanho do turno.
- As atividades $a_{s(v)},...,a_A$ de um cliente v devem ser executadas em ordem, isto é, a atividade de índice i, $i > s(v)$, só pode começar a ser executada em um turno se a atividade de índice $i-1$ já tiver sido executada por completo naquele turno ou em um turno anterior.
- Nem todo cliente precisa ter alguma atividade realizada em cada turno.
- A última atividade executada por uma equipe em um turno não precisa ser executada completamente, neste caso, esta atividade deverá ser a primeira atividade executada no turno seguinte por aquela equipe.
- Cada uma das atividades $a_{s(v)},...,a_A$ do cliente v deverá ter sido executada por completo ao fim do último turno.

O objetivo do problema é encontrar um itinerário que minimize o número de turno necessários para realizar todas as atividades de todos os clientes por completo.

Este problema é não trivial, pois é computacionalmente inviável testar todas as combinações possíveis para atestar qual delas é a que apresenta melhor resultado. Tendo isso em vista, algoritmos heurísticos constituem uma boa opção para a resolução do problema.

3 HEURÍSTICAS CONSTRUTIVAS

Inicialmente, foram desenvolvidas 4 heurísticas construtivas na linguagem Python versão 3. Posteriormente, os dois melhores destes 4 algoritmos foram utilizados como base para dois algoritmos baseados na meta heurística Colônia de Formigas. As heurísticas construtivas são descritas a seguir.

3.1 Algoritmo construtivo 1

Este foi o primeiro algoritmo desenvolvido e tem como ideia base usar o tempo que um equipe gasta para executar uma atividade em um cliente como parâmetro de custo, assim como será feito nos demais algoritmos. Em cada rodada, cada time escolhe sua melhor atividade, mas apenas o que obtiver melhor resultado executará a atividade realmente, a seguir está o pseudo código do algoritmo:

Algorithm 1: ALGORITMO CONSTRUTIVO 1

```
1  Seja listaComTempos uma lista de tupla(i, a), sendo i o cliente e a a atividade,
     representando todas as atividades de todos os clientes que ainda não foram
     concluídas;
2  Seja equipesProblema a lista com todas as equipes do problema, sendo que cada
     equipe possui um relógio que demarca onde a mesma está na linha do tempo;
3  Seja solucao uma lista iniciada vazia;
4  while listaComTempos não estiver vazia do
5      Seja listaDeEscolhas uma lista iniciada vazia, onde serão colocadas as
         melhores escolhas de cada equipe;
6      foreach equipe em equipesProblema do
7          Adicionar na listaDeEscolhas a tupla(j, b, k), sendo j o cliente, b a
             atividade, e k a equipe. Esta tupla representa a escolha da equipe que se
             caracteriza como a atividade que a mesma terminará primeiro e está
             disponível para ser executada no momento, de acordo com o relógio da
             equipe, considere disponível uma atividade cujas as antecessoras já foram
             executada e ela não;
8      end
9      Seja melhorAtividade a tupla(j, b, k) na listaDeEscolhas que terminará
         primeiro;
10     A equipe k executará a atividade b do cliente j;
11     Remove o par (j, b) da listaComTempos;
12     Adiciona melhorAtividade na solucao;
13 end
14 Retorna a solucao;
```

3.2 Algoritmo construtivo 2

Após o desenvolvimento do primeiro algoritmo, experimentou-se o que aconteceria caso as equipes pudessem esperar para uma atividade ser feita, por exemplo, supondo que existe uma equipe no algoritmo 1 que escolheria a atividade A que gastaria 10 horas, nesse algoritmo poderia escolher a atividade B que depende de outra equipe terminar a antecessora da mesma, mas que em compensação a equipe gaste apenas 5h para executara mais o tempo de 2h de esperar para a antecessora ser finalizada, segue o algoritmo:

Algorithm 2: ALGORITMO CONSTRUTIVO 2

```
1  Seja listaComTempos uma lista de tupla(i, a), sendo i o cliente e a a atividade,
     representando todas as atividades de todos os clientes que ainda não foram
     concluídas;
2  Seja equipesProblema a lista com todas as equipes do problema, sendo que cada
     equipe possui um relógio que demarca onde a mesma esta na linha do tempo;
3  Seja solucao uma lista iniciada vazia;
4  while listaComTempos não estiver vazia do
5  |  Seja listaDeEscolhas uma lista iniciada vazia, onde serão colocadas as
   |    melhores escolhas de cada equipes;
6  |  foreach equipe em equipesProblema do
7  |  |  Adicionar na listaDeEscolhas a tupla(j, b, k), sendo j o cliente, b a
   |  |    atividade, e k a equipe. Esta tupla representa a escolha da equipe que se
   |  |    caracteriza como a atividade que a mesma terminará primeiro de acordo
   |  |    com o relógio, levando em conta o tempo de espera para inicio da
   |  |    atividade, sendo que as atividades ainda não finalizadas também podem
   |  |    ser selecionadas;
8  |  end
9  |  Seja melhorAtividade a tupla(j, b, k) na listaDeEscolhas que terminará
   |    primeiro;
10 |  A equipe k executará a atividade b do cliente j;
11 |  Remove o par (j, b) da listaComTempos;
12 |  Adiciona melhorAtividade na solucao;
13 end
14 Retorna a solucao;
```

3.3 Algoritmo construtivo 3 e 4

Com base nos algoritmos 1 e 2, foi decidido tentar uma nova abordagem onde, ao invés de apenas a melhor equipe executar uma atividade, todas as equipes em cada rodada do algoritmo, toda equipe executa a sua melhor atividade tentando assim criar uma melhor distribuição

das atividades entre as equipes, sendo que esta abordagem foi aplicada nos algoritmos 1 e 2, resultando nos algoritmos 3 e 4 respectivamente, seguem os pseudo-códigos dos mesmos:

Algorithm 3: ALGORITMO CONSTRUTIVO 3

```
1  Seja listaComTempos uma lista de tupla(i, a), sendo i o cliente e a a atividade,
     representando todas as atividades de todos os clientes que ainda não foram
     concluídas;
2  Seja equipesProblema a lista com todas as equipes do problema, sendo que cada
     equipe possui um relógio que demarca onde a mesma esta na linha do tempo;
3  Seja solucao uma lista iniciada vazia;
4  while listaComTempos não estiver vazia do
5      foreach equipe em equipesProblema do
6          Seja a tupla(j, b, k), sendo j o cliente, b a atividade, e k a equipe. Esta tupla
             representa a escolha da equipe que se caracteriza como a atividade que a
             mesma terminará primeiro e esta disponível para ser executada no
             momento de acordo com o relógio da equipe, considere disponível uma
             atividade cujas as antecessoras já foram executada e ela não;
7          A equipe k executa a atividade b do cliente j;
8          Remove o par (j, b) da listaComTempos;
9          Adiciona melhorAtividade na solucao;
10     end
11 end
12 Retorna a solucao;
```

3.4 Resultados das heurísticas construtivas

Para a realização dos testes foram geradas instâncias artificias do problema, estas randomicamente formadas seguindo como critério de formação a quantidade de equipes, o número de clientes e a quantidade de atividades total que o serviço pode ter. Sendo assim, para a criação do total de 14 casos de testes variou-se estes parâmetros, afim de compreender a eficácia de cada algoritmo para instâncias pequenas e grandes.

Estas instâncias artificias são geradas por um algoritmo desenvolvido em Python versão 3, onde o algoritmo recebe a quantidade de clientes, a quantidade de atividades total e o número de equipes. Em seguida, é definido em qual atividade cada cliente começará de forma aleatória usando a função *random* que segue o algoritmo de Mersenne Twister (MATSUMOTO; NISHIMURA, 1998), enquanto que a distância entre os clientes é definida usando distribuição normal com esperança matemática igual a 30 minutos e desvio padrão de 10 minutos. Além disso, o tempo gasto por cada equipe para concluir uma atividade de um cliente é definida usando distribuição normal com esperança matemática igual a 20 horas e desvio padrão de 10 horas.

Algorithm 4: ALGORITMO CONSTRUTIVO 4

```
1  Seja listaComTempos uma lista de tupla(i, a), sendo i o cliente e a a atividade,
     representando todas as atividades de todos os clientes que ainda não foram
     concluídas;
2  Seja equipesProblema a lista com todas as equipes do problema, sendo que cada
     equipe possui um relógio que demarca onde a mesma esta na linha do tempo;
3  Seja solucao uma lista iniciada vazia;
4  while listaComTempos não estiver vazia do
5      Seja listaDeEscolhas uma lista iniciada vazia, onde serão colocadas as
         melhores escolhas de cada equipes;
6      foreach equipe em equipesProblema do
7          Seja tupla(j, b, k), sendo j o cliente, b a atividade, e k a equipe. Esta tupla
             representa a escolha da equipe que se caracteriza como a atividade que a
             mesma terminará primeiro de acordo com o relógio, levando em conta o
             tempo de espera para inicio da atividade, sendo que as atividades ainda
             não finalizadas também podem ser selecionadas;
8          A equipe k executa a atividade b do cliente j;
9          Remove o par (j, b) da listaComTempos;
10         Adiciona melhorAtividade na solucao;
11     end
12 end
13 Retorna a solucao;
```

Sendo assim, cada instância artificial possui características condizentes com uma instância real do problema.

Os testes foram realizados em um máquina com 8gb de memória ram, processador Intel Core I7 de quarta geração e usando o sistema operacional Ubuntu que executou todos os casos de testes para cada algoritmo desenvolvido. A tabela 3.1 apresenta as características dos casos de teste e a tabela 3.2 apresenta os resultados obtidos.

Com base na tabela de resultados das heurísticas é visível que o algoritmo 2 possui melhores soluções para problemas grandes, seguido do algoritmo 1. Por isso, eles foram escolhidos como base para a aplicação da meta heurística de ACO.

Tabela 3.1 – Parâmetros dos Casos de Testes

	Quantidade de Clientes	Número de Atividades	Número de Equipes
Caso 1	50	10	5
Caso 2	60	10	5
Caso 3	70	10	5
Caso 4	80	10	5
Caso 5	90	10	5
Caso 6	50	15	5
Caso 7	50	20	5
Caso 8	50	25	5
Caso 9	50	30	5
Caso 10	50	10	6
Caso 11	50	10	7
Caso 12	50	10	8
Caso 13	50	10	9
Caso 14	90	30	9

Tabela 3.2 – Comparação das Heurísticas

		Algoritmo 1	Algoritmo 2	Algoritmo 3	Algoritmo 4
Caso 1	Tempo(s)	0.037	0.043	0.007	0.009
	Solução(dias)	106d 6.029h	**93d 5.625h**	118d 2.252h	124d 1.038h
Caso 2	Tempo(s)	0.044	0.052	0.009	0.011
	Solução(dias)	101d 6.518h	**89d 2.495h**	98d 4.798h	130d 2.394h
Caso 3	Tempo(s)	0.067	0.077	0.013	0.016
	Solução(dias)	**105d 4.713h**	110d 0.839h	115d 3.821h	158d 3.250h
Caso 4	Tempo(s)	0.083	0.092	0.017	0.019
	Solução(dias)	**139d 2.295h**	141d 2.736h	146d 0.740h	204d 0.947h
Caso 5	Tempo(s)	0.103	0.119	0.021	0.026
	Solução(dias)	167d 4.473h	**150d 2.531h**	177d 2.522h	202d 4.234h
Caso 6	Tempo(s)	0.051	0.062	0.010	0.012
	Solução(dias)	124d 6.421h	**120d 4.810h**	138d 3.845h	188d 4.415h
Caso 7	Tempo(s)	0.069	0.080	0.014	0.017
	Solução(dias)	167d 4.749h	**165d 2.640h**	188d 4.301h	243d 3.075h
Caso 8	Tempo(s)	0.084	0.093	0.017	0.020
	Solução(dias)	222d 1.284h	**214d 7.574h**	240d 1.101h	281d 7.751h
Caso 9	Tempo(s)	0.101	0.119	0.021	0.025
	Solução(dias)	318d 2.363h	**241d 0.552h**	276d 2.222h	358d 7.863h
Caso 10	Tempo(s)	0.050	0.061	0.008	0.010
	Solução(dias)	83d 5.558h	**78d 4.595h**	88d 3.387h	119d 1.071h
Caso 11	Tempo(s)	0.051	0.062	0.007	0.009
	Solução(dias)	64d 6.180h	**55d 2.005h**	77d 0.583h	88d 1.944h
Caso 12	Tempo(s)	0.054	0.073	0.007	0.009
	Solução(dias)	54d 4.660h	**50d 2.379h**	52d 6.473h	76d 6.968h
Caso 13	Tempo(s)	0.071	0.076	0.007	0.009
	Solução(dias)	49d 3.828h	**43d 4.552h**	50d 2.851h	62d 5.392h
Caso 14	Tempo(s)	0.596	0.671	0.068	0.081
	Solução(dias)	212d 1.454h	**184d 0.634h**	227d 1.439h	324d 7.473h

4 METAHEURÍSTICAS

Ibrahim H. Osman define meta heurística da seguinte forma: "Uma meta heurística é formalmente definida como um processo de geração iterativo que orienta uma heurística subordinada, combinando conceitos inteligentemente diferentes para explorar e se aproveitar do espaço de busca, estratégias de aprendizagem são usadas para estruturar informações para encontrar soluções eficientemente próximas da solução ótima."(OSMAN; LAPORTE, 1996).

Pode se dizer que a intenção é apresentar uma ideia para solucionar problemas heuristicamente porém genérica. Uma meta heurística guia uma heurística na busca da solução no espaço de buscas, estruturando a informação conforme definido pela mesma.

Dentre os vários exemplos de metaheurísticas, podemos citar colônia de formigas, busca tabu, algoritmo genético, entre outros (GLOVER; KOCHENBERGER, 2006). Para o desenvolvimento deste trabalho foi selecionada a meta heurística colônia de formigas, dado sua utilização com sucesso no problema de roteamento de veículos (BELL; MCMULLEN, 2004) (YU; YANG; YAO, 2009) e em variações deste problema (MONTEMANNI et al., 2005) (DONATI et al., 2008), a metaheurística é definida a seguir.

4.1 Colônia de formigas

Colônia de formigas é uma meta heurística que se baseia no comportamento das formigas em seu eco sistema, essas formigas são guiadas por feromônios exalados por elas mesmas. Dessa forma, quando saem em busca de mantimentos, deixam por onde passam esses feromônios para que as formigas posteriores saibam o caminho percorrido e assim cheguem mais rapidamente aos mantimentos, pois o caminho mais curto tende a ficar com mais feromônios (COLORNI et al., 1992).

Um exemplo prático desse sistema é supor que existe um conjunto de formigas em um formigueiro e, próximo a este, um alimento para elas. Inicialmente as formigas percorreram os arredores do formigueiro liberando feromônios pelo caminho, algumas delas encontraram um caminho até o alimento e por consequência voltaram para o alimento para pegar mais, assim o caminho até o alimento conterá mais feromônios do que os caminhos criados aleatoriamente, e as demais formigas começaram a seguir o caminho com mais feromônios e por fim um caminho significantemente curto é criado entre o formigueiro e o alimento através dos feromônios.

No algoritmo de colônia de formigas, soluções do problema são formadas por vários componentes. Cada formiga constrói soluções selecionando componentes. No caso do PRAS,

cada formiga poderia executar os algoritmos 1, 2, 3 ou 4. Porém, ao escolher uma nova componente, cada formiga leva em consideração a quantidade de feromônio. Após cada rodada de construção ocorrem dois procedimentos, o primeiro a evaporação onde a quantidade de feromônios em todo o espaço é decrementado seguindo um fator de decremento, e em seguida ocorre o depósito de feromônios em que as componentes que apareceram nas melhores soluções recebem um aumento de feromônio para assim iniciar uma nova rodada de formigas até que um critério de parada seja atingido.

Seguem as fórmulas matemáticas, já adaptadas ao problema de PRAS, que definem formalmente esses comportamentos:

- A fórmula que define a chance de uma formiga ir para o cliente i executar a atividade a, sendo τ^k a quantidade de feromônio referente a equipe k executar a atividade a do cliente i; η o valor do inverso do tempo gasto para que a equipe k execute a atividade a do cliente i; α e β são constantes que definem o peso dos feromônios nas trilhas; e o valor, e *vist* é o conjunto composto dos pares ordenados (cliente, atividade) porém apenas os que podem ser visitados e feitos naquele instante (YU; YANG, 2011):

$$p^k(i,a) = \begin{cases} \frac{[\tau^k_{(i,a)}]^\alpha \times [\eta_{(i,a)}]^\beta}{\Sigma_{(j,b)\in vist}[\tau^k_{(j,b)}]^\alpha \times [\eta_{(j,b)}]^\beta} & (j,b) \in vist \\ 0 & \text{senão} \end{cases} \tag{4.1}$$

- A fórmula para evaporação de feromônios, sendo ${\tau^k_{(i,a)}}^{novo}$ a nova quantidade de feromônio referente a equipe k executar a atividade a do cliente i; ${\tau^k_{(i,a)}}^{antigo}$ a antiga quantidade de feromônio referente a equipe k executar a atividade a do cliente i, e $desc$ um fator de decremento sendo $desc \in \mathtt{R}_+$ e $desc < 1$:

$${\tau^k_{(i,a)}}^{novo} = {\tau^k_{(i,a)}}^{antigo} \times desc \tag{4.2}$$

- A fórmula para incremento de feromônios, sendo ${\tau^k_{(i,a)}}^{novo}$ a nova quantidade de feromônio referente a equipe k executar a atividade a do cliente i; ${\tau^k_{(i,a)}}^{antigo}$ a antiga quantidade de feromônio referente a equipe k executar a atividade a do cliente i; L o conjunto de todas as formigas que escolheram alocar a equipe k para executar a atividade a do cliente

i; θ_f o valor que a solução da formiga f obteve e M o fator de acréscimo onde $M \in \mathbb{R}_+$:

$$\tau_{(i,a)}^{k\ \ novo} = \tau_{(i,a)}^{k\ \ antigo} + \sum_{f \in L} \frac{M}{\theta_f} \tag{4.3}$$

4.2 Heurísticas usando colônia de formigas

Finalizados os testes com os algoritmos construtivos foram desenvolvidas duas heurísticas que utilizam a meta heurística de colônia de formigas e os algoritmos construtivos 1 e 2 como base, porém para que o conceito da meta heurística seja aplicado os algoritmos construtivos foram adaptados, sendo esta adaptação baseada em mudar a seleção de atividade fazendo-se agora guiada pela fórmula de feromônios apresentada no capítulo 2, de forma explicativa antes o método de seleção era totalmente determinístico decidindo a atividade por meio da eficiência da equipe, já agora o método tornou-se não determinístico pois a escolha é feita aleatoriamente com maiores chances de uma atividade curta e com mais feromônios ser escolhida.

Após a implementação dos algoritmos ACO foram feitos testes com diferentes configurações de parâmetros da metaheurística, tendo como valores que apresentaram melhores resultados tem-se *desc* igual a 0.9, *ascr*, α e β igual a 1. Os pseudo-códigos são apresentados em Algoritmo ACO 1 e Algoritmo ACO 2, usando os algoritmos construtivos 1 e 2, respectivamente.

Algorithm 5: ALGORITMO ACO 1

```
1  Seja iteracoes a quantidade de iteracoes que o algoritmo ira executar, no caso 100;
2  Seja quantidadeFormigas a quantidade de formigas usadas por iteração, no caso 10;
3  Seja desc o fator de decrescimento com valor 0.9;
4  Seja ascr o fator de acréscimo com valor 1;
5  Seja solucaoFinal iniciada com vazio, guardara a melhor solucao em cada iteração;
6  foreach i in iteracoes do
7      Seja formigas a lista com as soluções encontradas, iniciada com vazio;
8      foreach f in quantidadeFormigas do
9          Chama a execução do algoritmo construtivo 1 adaptado ao ACO e coloca a
             solução em formigas;
10     end
11     Diminui o valor de todos os feromônios multiplicando por desc;
12     foreach formiga in formigas do
13         Aumenta os feromônios para todos as escolhas, essas composta de equipe
             com atividade do cliente, que pertencem a formiga, para que na próxima
             iteração a equipe tenha mais chances de selecionar aquela atividade do
             cliente;
14     end
15     Seja melhorFormiga aquela que apresentou melhor resultado;
16     if melhorFormiga possuir resultado melhor que solucaoFinal then
17         solucaoFinal recebe melhorFormiga;
18     else
19         solucaoFinal se mantém;
20     end
21 end
22 Retorna a solucaoFinal;
```

Algorithm 6: ALGORITMO ACO 2

```
1  Seja iteracoes a quantidade de iteracoes que o algoritmo ira executar, no caso 100;
2  Seja quantidadeFormigas a quantidade de formigas usadas por iteração, no caso 10;
3  Seja desc o fator de decrescimento com valor 0.9;
4  Seja ascr o fator de acréscimo com valor 1;
5  Seja solucaoFinal iniciada com vazio, guardara a melhor solucao em cada iteração;
6  foreach i in iteracoes do
7      Seja formigas a lista com as soluções encontradas, iniciada com vazio;
8      foreach f in quantidadeFormigas do
9          Chama a execução do algoritmo construtivo 2 adaptado ao ACO e coloca a
             solução em formigas;
10     end
11     Diminui o valor de todos os feromônios multiplicando por desc;
12     foreach formiga in formigas do
13         Aumenta os feromônios para todos as escolhas, essas composta de equipe
             com atividade do cliente, que pertencem a formiga, para que na próxima
             iteração a equipe tenha mais chances de selecionar aquela atividade do
             cliente;
14     end
15     Seja melhorFormiga aquela que apresentou melhor resultado;
16     if melhorFormiga possuir resultado melhor que solucaoFinal then
17         solucaoFinal recebe melhorFormiga;
18     else
19         solucaoFinal se mantém;
20     end
21 end
22 Retorna a solucaoFinal;
```

5 RESULTADOS E DISCUSSÕES

Para fins de comparação o algoritmo desenvolvido foi comparado a um *framework* chamado Formigueiro desenvolvido pelo professor da Universidade Federal de Lavras (UFLA) Dilson Lucas Pereira, este é um *framework*, hospedado em github.com/dilsonpereira/Formigueiro, que aplica a meta heurística de colônia de formigas para a resolução de qualquer problema.

Para a realização dos testes foram usados os mesmos casos de uso apresentados na tabela 3.1 e na mesma máquina apresentada anteriormente. Todos os casos de testes foram resolvidos pelo *framework* e em seguida pelos algoritmos ACO desenvolvidos, a tabela 5.1 apresenta os resultados obtidos.

Tabela 5.1 – *Framework* X Algoritmos Desenvolvidos

		Algoritmo 1	Algoritmo 2	ACO 1	ACO 2	*Framework*
Caso 1	Tempo(s)	0.037	0.043	58.062	64.391	248.940
	Solução(dias)	106d 6.029h	93d 5.625h	**84d 2.890h**	87d 1.295h	206d 5.76h
Caso 2	Tempo(s)	0.044	0.052	70.263	77.635	245.463
	Solução(dias)	101d 6.518h	89d 2.495h	**82d 1.487h**	84d 7.950h	210d 1.52h
Caso 3	Tempo(s)	0.067	0.077	103.235	113.301	299.967
	Solução(dias)	105d 4.713h	110d 0.839h	**98d 3.057h**	101d 6.794h	230d 4.08h
Caso 4	Tempo(s)	0.083	0.092	129.527	143.346	461.633
	Solução(dias)	139d 2.295h	141d 2.736h	**125d 4.490h**	129d 1.456h	280d 6.32h
Caso 5	Tempo(s)	0.103	0.119	163.374	179.342	503.500
	Solução(dias)	167d 4.473h	150d 2.531h	**139d 6.546h**	143d 2.213h	280d 5.84h
Caso 6	Tempo(s)	0.051	0.062	81.574	90.979	224.510
	Solução(dias)	124d 6.421h	120d 4.810h	**109d 4.311h**	113d 2.276h	292d 1.44h
Caso 7	Tempo(s)	0.069	0.080	107.042	119.003	282.530
	Solução(dias)	167d 4.749h	165d 2.640h	**150d 4.910h**	157d 6.266h	548d 4.48h
Caso 8	Tempo(s)	0.084	0.093	133.181	147.753	330.857
	Solução(dias)	222d 1.284h	214d 7.574h	**189d 0.778h**	198d 6.918h	963d 4.16h
Caso 9	Tempo(s)	0.101	0.119	155.115	173.660	362.276
	Solução(dias)	318d 2.363h	241d 0.552h	217d 2.190h	228d 2.253h	**182d 0.8h**
Caso 10	Tempo(s)	0.050	0.061	74.123	83.165	230.112
	Solução(dias)	83d 5.558h	78d 4.595h	**69d 1.412h**	72d 6.698h	164d 5.68h
Caso 11	Tempo(s)	0.051	0.062	77.924	88.143	257.750
	Solução(dias)	64d 6.180h	55d 2.005h	**49d 4.015h**	52d 0.640h	164d 5.68h
Caso 12	Tempo(s)	0.054	0.073	80.216	91.388	287.727
	Solução(dias)	54d 4.660h	50d 2.379h	42d 4.827h	**42d 4.222h**	164d 1.36h
Caso 13	Tempo(s)	0.071	0.076	94.585	109.241	286.984
	Solução(dias)	49d 3.828h	43d 4.552h	**35d 4.375h**	36d 7.427h	152d 3.6h
Caso 14	Tempo(s)	0.596	0.671	814.85	930.968	1510.945
	Solução(dias)	212d 1.454h	184d 0.634h	177d 3.559h	**175d 7.603h**	1544d 5.92h

Com base na tabelas 5.1 é possível afirmar que, apesar da heurística construtiva 2 ter apresentado melhores resultados anteriormente, ao utilizar a metaheurística de colônia de formigas o algoritmo ACO que apresentou melhores resultados foi o algoritmo ACO que utilizou a heurística construtiva 1. Um motivo para isto é o fato de que a primeira solução encontrada no algoritmo construtivo 1 não é tão interessante, porém ao executar o algoritmo diversas vezes a taxa de melhoramento da solução, após cada iteração, cresce mais do que a taxa de melhoramento do algoritmo construtivo 2.

Além disso, é apresentando uma melhora significativa de aproximadamente 17% quando comparadas as soluções do algoritmo ACO 1 e a heurística contrutiva 1, no caso de teste 5 por exemplo. Assim como, o algoritmo ACO 2 apresenta uma melhora de aproximadamente 9% em relação ao algoritmo construtivo 2. Demonstrando o aperfeiçoamento do algoritmo pela utilização da metaheurística.

Já em comparação ao *framework* os demais algoritmos apresentam excelentes resultados, tanto em relação ao tempo dispendido para encontrar uma solução quanto a qualidade da solução encontrada. O motivo do ocorrido se deve a vantagem de um algoritmo desenvolvido especificamente para solucionar um problema comparado a um genérico, desta forma, o algoritmo especializado consegue se aproveitar das características específicas do problema para encontrar soluções melhores. Os resultados se mantém consistentes as variações de quantidade de equipes, atividades e clientes, demonstrando o ponto apresentado.

6 CONCLUSÃO

Neste trabalho, o problema PRAS foi atacado por meio da criação de heurísticas construtivas guiadas pela metaheurística de colônia de formigas, sendo desenvolvidos 4 algoritmos construtivos e 2 algoritmos ACO. Obteve-se resultados satisfatórios demonstrando que além de possível, é eficiente utilizar a metaheurística de colônia de formigas para a solução deste problema.

Para futuros projetos seria interessante a utilização de outras metaheurísticas como a busca tabu e os algoritmos genéticos, aplicadas ao problema PRAS e comparadas ao algoritmo desenvolvido no presente trabalho. Outra vertente, seria o desenvolvimento de algoritmos de buscas locais, como uma tentativa de melhorar os resultados encontrados pelas heurísticas deste trabalho.

Para o futuro, também poderia procurar cias florestais que fornecessem casos reais do problema para aplicar a metaheurística deste trabalho, verificando sua eficiência e o impactos do algoritmo no rendimento e dia-a-dia destas cias.

REFERÊNCIAS

BELL, J. E.; MCMULLEN, P. R. Ant colony optimization techniques for the vehicle routing problem. **Advanced engineering informatics**, Elsevier, v. 18, n. 1, p. 41–48, 2004.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM computing surveys (CSUR)**, Acm, v. 35, n. 3, p. 268–308, 2003.

COLORNI, A. et al. Distributed optimization by ant colonies. In: CAMBRIDGE, MA. **Proceedings of the first European conference on artificial life**. [S.l.], 1992. v. 142, p. 134–142.

DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. **Management science**, Informs, v. 6, n. 1, p. 80–91, 1959.

DONATI, A. V. et al. Time dependent vehicle routing problem with a multi ant colony system. **European journal of operational research**, Elsevier, v. 185, n. 3, p. 1174–1191, 2008.

FRANCIS, P. M.; SMILOWITZ, K. R.; TZUR, M. The period vehicle routing problem and its extensions. In: **The vehicle routing problem: latest advances and new challenges**. [S.l.]: Springer, 2008. p. 73–102.

GLOVER, F. W.; KOCHENBERGER, G. A. **Handbook of metaheuristics**. [S.l.]: Springer Science & Business Media, 2006. v. 57.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. **ACM Transactions on Modeling and Computer Simulation (TOMACS)**, ACM, v. 8, n. 1, p. 3–30, 1998.

MONTEMANNI, R. et al. Ant colony system for a dynamic vehicle routing problem. **Journal of Combinatorial Optimization**, Springer, v. 10, n. 4, p. 327–343, 2005.

OSMAN, I. H.; LAPORTE, G. **Metaheuristics: A bibliography**. [S.l.]: Springer, 1996.

YU, B.; YANG, Z. Z. An ant colony optimization model: The period vehicle routing problem with time windows. **Transportation Research Part E: Logistics and Transportation Review**, Elsevier, v. 47, n. 2, p. 166–181, 2011.

YU, B.; YANG, Z.-Z.; YAO, B. An improved ant colony optimization for vehicle routing problem. **European journal of operational research**, Elsevier, v. 196, n. 1, p. 171–176, 2009.