

TP2 Computação Natural – ACO / Max Clique

Diogo Oliveira Neiss e Guilherme de Mello Nunes

<https://github.com/diogoneiss/AntColonyOptimization-for-MaxClique>

Introdução

Metaheurísticas de *Ant Colony Optimization* são bem eficientes para soluções aproximativas de problemas np-difícies em grafos, realizando uma espécie de heurística gulosa estocástica, repetidamente buscando soluções golosas rápidas e aproveitando partes boas de soluções passadas para encontrar soluções melhores futuras.

O problema da clique-máxima consiste em encontrar a clique maximal máxima do grafo. Podemos encontrar a clique tanto pelos vértices que a compõe tanto pelas arestas. Dada o speedup do algoritmo ao utilizar a abordagem de vértices, preferimos considerar essa abordagem.

Solucionaremos isso construindo cliques maximais aleatórios baseando-se no movimento de cada formiga, com a formiga que produziu a melhor clique maximal de cada iteração depositando "feromônio" nos vértices que comporaram essa clique, e o feromônio dos demais "evaporando" com a passagem do tempo. O movimento das formigas se baseia na quantidade de feromônio em cada vértice, o seja, se ela está no vértice v_0 , cada vértice ligado a v_0 tem uma probabilidade de deslocamento. A formiga prefere caminhos que outras formigas já percorreram, escolhendo com maior probabilidade o vértice com mais feromônio. Isso garante *exploitation*, já que ela utiliza soluções para tentar construir uma melhor, e o componente probabilístico dos demais vértices garante *exploitation*, já que elas sempre começam de vértices aleatórios e, mesmo que um vértice v_1 tenha muito feromônio, ela ainda pode acabar indo para v_2 , evitando máximos locais.

Conforme o tempo passa, o feromônio inicial evaporará quase que por completo, com as formigas então compondo soluções passadas para criar soluções futuras, priorizando o refinamento do espaço de soluções ao invés de explorar soluções novas no espaço de busca.

Implementação

Modelamos o problema da clique máxima como encontrar o maior conjunto de vértices que compõe uma clique ao longo de várias repetições. Cada clique será maximal, já que construímos a solução até esgotar vértices que poderiam entrar na clique, mas **não necessariamente é máxima**.

Otimização do tempo de execução

Um dos gargalos do execução completa do algoritmo é a criação das cliques maximais, que é interpretado numa metáfora biológica como o *caminho de uma formiga*, operação que é repetida para cada formiga em cada iteração, ou seja, acelerar sua execução oferece speedup quase linear para o algoritmo.

Para resolver esse problema, criamos três funções principais:

- **construct_clique(G, tau, n, alpha)**

Essa função é responsável por criar a clique estocásticamente.

1. Escolhemos um vértice v_0 aleatório para começar e criamos uma lista de candidatos, inicialmente todos os vizinhos de v_0 .
2. Baseado nos feromônios de cada vértice, registrados no vetor τ , e o fator α , criamos um vetor de probabilidades para cada vértice candidato, de forma que vértices com mais feromônio tenham maior probabilidade.
3. Escolher vértice v_c para entrar na clique e registrar seus vizinhos em outra lista de candidatos.
4. Atualizar os candidatos, fazendo a interseção entre os candidatos do início da iteração com os de v_c , já que todo candidato futuro a entrar na clique também deve ser vizinho dos dois.
5. Verificar se o vértice é de fato vizinho de todos os vértices presentes na clique, e somente se for, o adicionamos a lista de vértices. Isso geralmente seria feito.
6. Verificar a lista de candidatos. Se tivermos algum, repetimos os passos 3-5. Quando a lista estiver vazia, terminamos nossa clique maximal.

- **ant_clique(G, ants_per_v, maxCycles, tauMin, tauMax, alpha=1.05, evaporation_rate=0.9, plot=False, parallel=False)**

Nessa função fazemos de fato o algoritmo de ACO, de acordo com os parâmetros especificados. Basicamente, criamos estruturas de dados para contagem de métricas. Estamos acompanhando a iteração do algoritmo que atingiu o melhor valor, assim como o número de cliques repetidas. Fazemos a conta de cliques repetidas com um dicionário tuplas x ocorrência, em que se uma clique for composta pelos mesmos vértices mas em ordens diferentes, consideramos a mesma.

O algoritmo segue esses passos:

1. Determinar a quantidade de formigas, `total_ants`, de acordo com o parâmetro `ants_per_V`
2. Criar loop para iterar `maxCycles` vezes;
3. Montar `total_ants` cliques estocásticas com a função de construção;
4. Escolher a maior clique desse conjunto;
5. Registrar a ocorrência dela no dicionário de ocorrência de cliques;
6. Comparar a melhor clique encontrada anteriormente com essa. Se for melhor, a trocamos;
7. Evaporar o feromônio, multiplicando por `evaporation_rate`;
8. Calcular a adição de feromônio, baseando-se na distância entre essa solução local e a melhor encontrada até o momento;
9. Incrementar os vértices da solução local no vetor `tau`;
10. Arrumar os limites min/max de feromônio, controlados por `tauMin`, `tauMax`, de forma que o vetor `tau` respeite os limites;
11. Repetir o passo 2, e quando ele terminar, retornamos a melhor clique encontrada e as métricas desejadas;

O algoritmo executa razoavelmente rápido, já que o gargalo é a função de construção de cliques, que devido a implementação com *Numba* é super rápida. Será que podemos melhorar?

● Paralelismo

Um ponto importante desse algoritmo é paralelismo. A função parece um bom candidato de paralelismo, vide a criação independente de cliques estocásticas, então teoricamente se criamos 50 cliques, usar 5 processos paralelos diminuiria a carga de trabalho para o tempo de execução de ~10 cliques. Experimentalmente isso foi uma abordagem horrível, já que o tempo de execução aumentou bastante. Imaginamos que a causa disso seja o tempo de serialização das estruturas de dados. Como o fluxo de um sistema paralelo é o *fork* dos dados, execução e *join*, observamos que a maior parte do tempo de execução é justamente o *fork*, que demora bastante, já que o grafo inteiro e outras variáveis precisam ser copiados para a thread que executará, algo que demora com grafos grandes. Em linguagens de nível menor, poderíamos definir o grafo como variável global imutável atômica, de forma que seu compartilhamento seria feito em $O(1)$, algo que não conseguimos fazer com sucesso em *python*.

Logo, como boa parte do tempo de execução é gasto no *fork*, não foi encontrado *speedup* de execução paralelo, sendo então melhor fazer as cliques aleatórias sequencialmente.

● experiment

Nessa função fazemos o intervalo de confiança do ACO, variando a semente aleatória de acordo com o número de variações aleatórias desejadas, combinando ao final as melhores soluções para escolher a melhor clique.

Resumidamente, recebemos o número de `random_runs` e, para cada índice, fixamos a semente aleatória, rodamos a função `ant_clique`, registramos os resultados e comparamos com a melhor clique vista até o momento, trocando-a se melhor.

Experimentos, Resultados e Análises

O estudo abordou o problema de maximização de clique em grafos completos testado nas bases de dados:

- Problema 1 (“fácil”) - 500 nós e 62.624 arestas
Tamanho do clique máximo: 13
<http://cedric.cnam.fr/~porumbed/graphs/dsjc500.5.col>
- Problema 4 (“difícil”) - 700 nós, 121.728 arestas
Tamanho do clique máximo: 44
https://iridia.ulb.ac.be/~fmascia/files/DIMACS/p_hat700-2.clq

O processo experimental teve como objetivo gerar conclusões sobre a sensibilidade dos principais parâmetros do ACO (número de formigas, número de iterações, taxa de evaporação, tau (feromônio) máximo por vértice e tau (feromônio) mínimo por vértice e fator alpha – peso de feromônio na probabilidade por vértice) sobre os resultados do problema de maximização de clique. Para cada configuração de parâmetros foram realizadas execuções com 50 diferentes *seeds* aleatórias.

Os resultados para cada combinação de parâmetros foram avaliados por quatro principais métricas:

- Clique máxima encontrada;
- Desvio padrão do clique máximo (entre sementes aleatórias);
- Métrica de Velocidade:
 - Número de iterações do ACO necessárias para que 80% das execuções aleatórias (40 execuções) atinjam clique maximal = clique máxima encontrado;
- Métrica de Corretude:
 - % de execuções aleatórias finalizadas com clique maximal abaixo do clique máximo encontrado;

Foi realizada uma busca por grade (*grid search*) sobre os parâmetros mencionados, com o seguinte domínio por parâmetro:

```
sementes_aleatorias = 50
numero_formigas     = [0.02, 0.05, 0.06, 0.07, 0.1, 0.2, 0.5]
numero_iteracoes    = [10, 50, 100, 500, 1000]
tau_min             = [0.01, 0.1, 0.5]
tau_max             = [2, 3, 6, 8]
alpha               = [0, 0.75, 0.98, 1, 1.1, 2]
taxa_evaporacao    = [0.8, 0.9, 0.95, 0.98, 0.99]
```

Totalizando 12.600 (combinações de parâmetros) * 50 (sementes aleatórias) = 630.000 execuções para cada base de dados.

Como o tamanho do vetor de combinações é 12600, quantidade bem grande, criamos uma lógica paralela de experimentação, seguindo uma lógica de memorização de resultados:

1. Dividir as combinações em batches, de acordo com o número de CPUs;
2. Para cada batch, redistribuir a execução do experimento nas CPUs desejadas;

3. Em cada experimento, verificamos se os parâmetros estão no cache. Se estiver, pulamos este experimento. Senão, rodamos a função de experimentação e retornamos os resultados;
4. Ao final da execução paralela, agregamos os resultados de cada CPU e salvamos no cache.

Isso proveu speedup substancial, já que a execução demoraria mais de 40 horas para o caso *easy* em minha máquina, reduzindo a execução total bastante.

Como discutimos anteriormente, grande parte do gargalo é o *fork*, em que o grafo é serializado e copiado para cada thread. Usando primitivas de sincronização de baixo nível, poderíamos melhorar ainda mais o tempo de execução, porém para o escopo desse trabalho essa melhora já foi suficiente.

• Resultados Gerais de Clique Máximo e Desvio Padrão

Em nossa análise inicial, examinamos a distribuição geral dos resultados para as métricas de "clique máximo" e "desvio padrão do clique máximo" em todas as 12.600 combinações de parâmetros avaliadas.

Para cada combinação de parâmetros, normalizamos o desvio padrão pelo valor de clique máximo encontrado, criando uma medida de desvio padrão percentual em relação ao clique máximo:

Resultados para o problema fácil



Figura 1 – Distribuição de Valores de Clique Máximo e Desvio Padrão para Grafo “Fácil”

Resultados para o problema difícil



Figura 2 - Distribuição de Valores de Clique Máximo e Desvio Padrão para Grafo "Difícil"

Os resultados indicam que, de maneira geral, o ACO é altamente eficaz na resolução do problema de maximização de clique. Nas várias execuções realizadas, o algoritmo foi capaz de identificar cliques máximos iguais ao valor ótimo global em 53% dos casos para o problema considerado "fácil" e em 33% dos casos para o problema classificado como "difícil".

Descobrimos que o desvio padrão foi satisfatoriamente baixo para os resultados obtidos. Para o problema "fácil", 99% das execuções mostraram um valor de desvio padrão que foi menor que 5% do valor de clique máximo. Para o problema "difícil", esse percentual foi de 97%. Estes resultados ressaltam a robustez estatística do algoritmo ACO, confirmando que os valores de clique máximo encontrados podem ser devidamente atribuídos às respectivas combinações de parâmetros que os geraram.

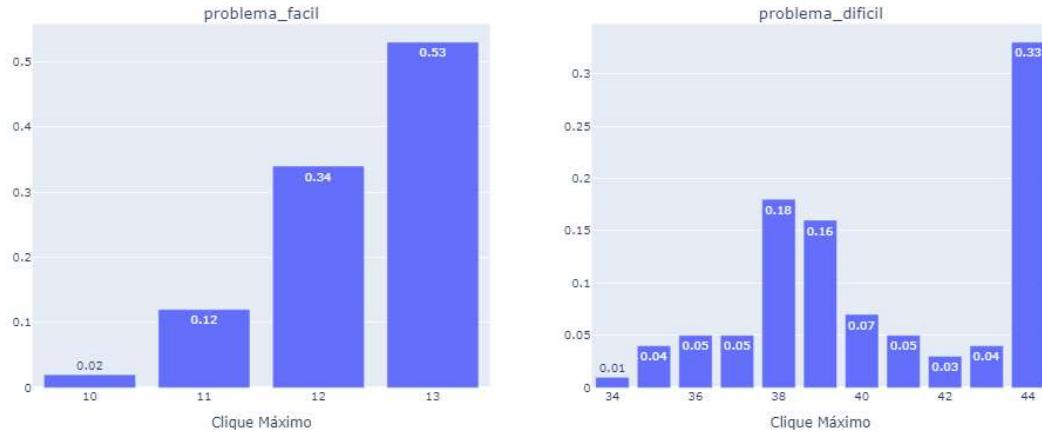


Figura 3 – Distribuição de Valores de Clique Máximo por Grafo

• Resultados de Clique Máximo e Desvio Padrão por Parâmetro

Prosseguindo com a análise de resultados, buscou-se entender quais parâmetros tiveram mais influência nos valores de clique máximo encontrados. A análise foi feita através de topográficos avaliando a distribuição de clique máximo por parâmetro. Regiões mais escuras indicam maior concentração de casos:

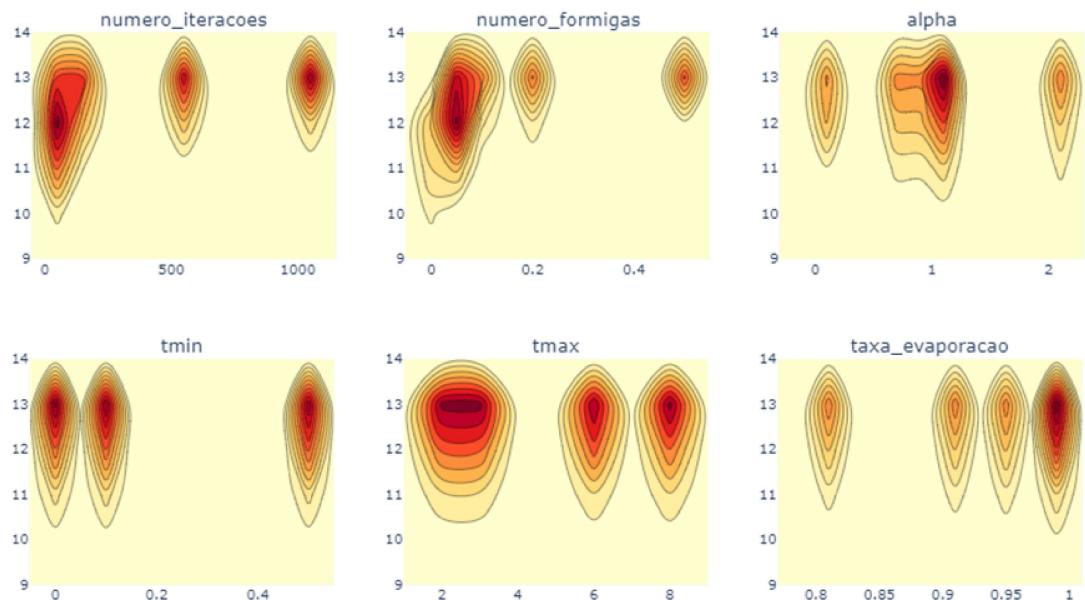


Figura 4 – Topográfico de Distribuição de Clique Máxima por Parâmetro (Grafo “Fácil”)

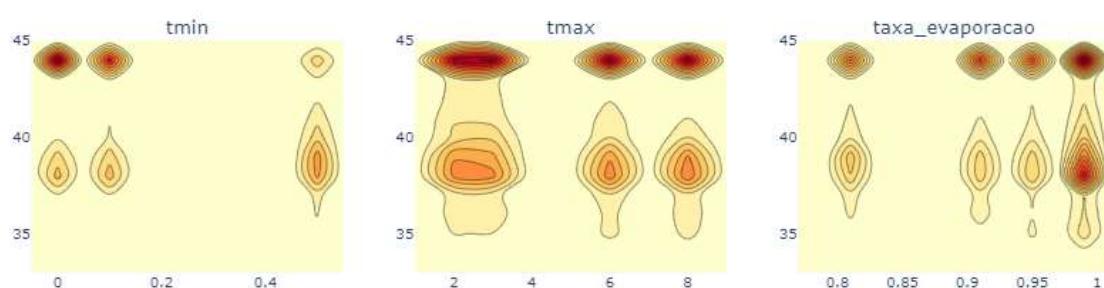
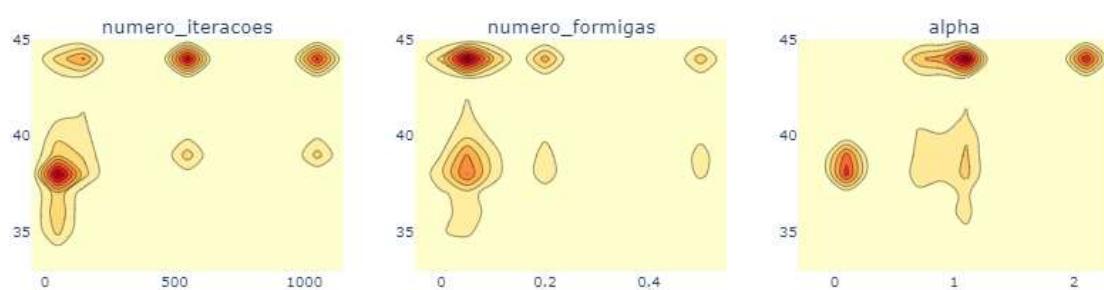


Figura 5 - Topográfico de Distribuição de Clique Máxima por Parâmetro (Grafo “Difícil”)

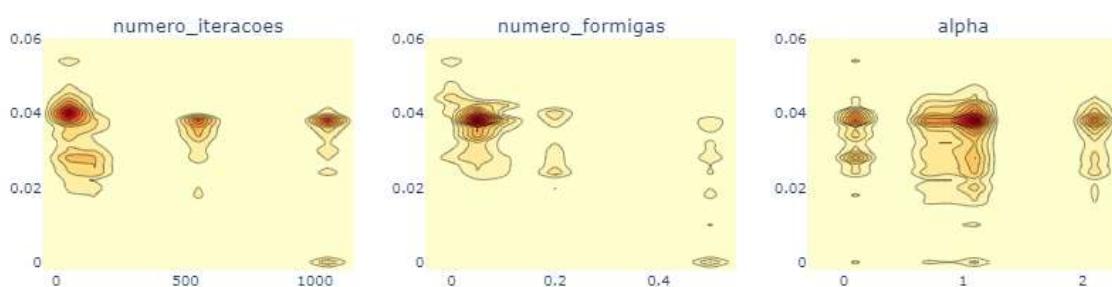


Figura 6 - Topográfico de Distribuição de Desvio Padrão por Parâmetro (Grafo “Fácil”)

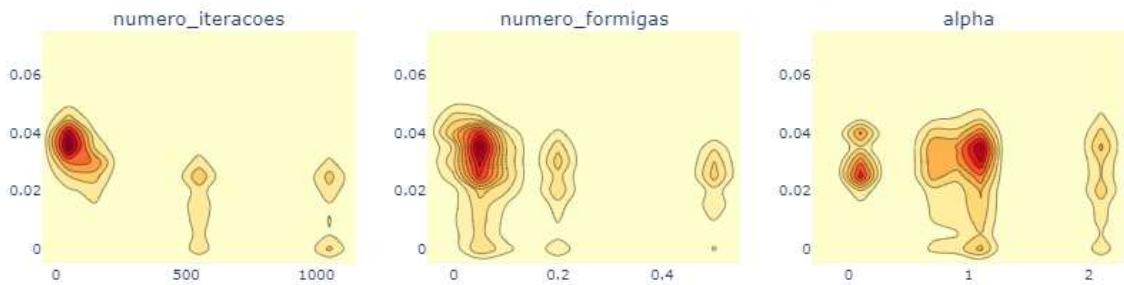


Figura 7 - Topográfico de Distribuição de Desvio Padrão por Parâmetro (Grafo “Difícil”)

- **Número de Iterações:** Como esperado, um número maior de iterações resultou em cliques máximos maiores e um desvio padrão menor para ambos os grafos (fácil e difícil). Isso é intuitivo, já que mais iterações permitem ao algoritmo explorar mais apropriadamente o espaço de soluções.
- **Número de formigas:** Percebemos no grafo “fácil” que um número maior de formigas, tende a gerar melhores resultados de clique máximo, porém essa melhora é acompanhada de um aumento no desvio padrão. Isso sugere que, embora mais formigas possam aumentar a diversidade de soluções encontradas, também introduzem uma maior variabilidade nos resultados. Para o grafo “difícil” o número de formigas não se mostrou um parâmetro causal ao clique máximo, porém correlato ao número de iterações.
- **Fator alpha:** Constatamos valores de alpha entre 1 e 1.1 produzem os melhores cliques máximos no grafo “difícil”. Isso confirma a importância de se equilibrar a influência do feromônio e do *desirability* na probabilidade de escolha de um vértice.
- **Tau máximo e mínimo por vértice:** No caso do grafo “fácil” tau_maximo e tau_minimo mostraram pouca correlação com o valor de clique_maximo. Já no grafo “difícil”, constatamos que valores menores de tau_minimo tendem a produzir melhores resultados de cliques máximos. A razão para isso pode estar na dinâmica do ACO: um tau mínimo menor implica que os caminhos menos promissores (aqueles com baixas quantidades de feromônio) são menos propensos a serem seguidos pelas formigas, focando a busca em caminhos mais promissores, melhorando assim a eficácia geral da busca.
- **Taxa de evaporação:** A taxa de evaporação não demonstrou ter um impacto significativo nos valores de clique máximo. Entretanto, é importante notar que uma taxa de evaporação muito alta poderia potencialmente resultar em uma convergência prematura para soluções subótimas, enquanto uma taxa muito baixa poderia levar a uma exploração excessiva e ineficiente do espaço de busca.

● Resultados Gerais de Métricas de Corretude e Velocidade

A criação de métricas de velocidade e corretude é crucial para entender a eficácia e eficiência do algoritmo ACO na resolução do problema de busca de cliques máximos. A métrica de velocidade, que mede o número de iterações necessárias para que 80% das execuções atinjam o clique máximo encontrado, permite avaliar a rapidez com que o algoritmo converge para uma solução potencialmente ótima. Uma velocidade mais alta sugere uma otimização mais eficiente.

Por outro lado, a métrica de corretude, que é a percentagem de execuções que finalizam com um clique maximal abaixo do clique máximo encontrado, fornece uma medida de quão consistentemente o algoritmo pode encontrar a solução ótima. Uma medida de intervalo de confiança sobre a capacidade da parametrização encontrar o valor de clique máximo global. Juntas, essas métricas fornecem um quadro abrangente da eficácia do algoritmo ACO na resolução do problema.

Foram analisados os resultados das métricas de velocidade e corretude apenas para casos em que clique_maximo = máximo global. Esse é um pré-requisito para considerar uma solução como boa, tendo em vista o alto volume de casos encontrados em cada grafo (53% em “fácil” e 33% em “difícil”).

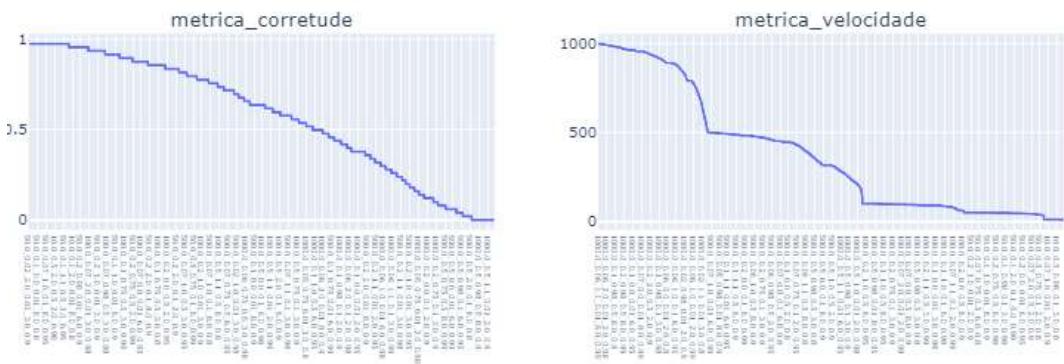


Figura 8 – Distribuição de Métricas de Corretude e Velocidade para Grafo “Fácil”



Figura 9 – Distribuição de Métricas de Corretude e Velocidade para Grafo “Difícil”

Nota-se que o objetivo é minimizar ambas as métricas - velocidade e corretude. Minimizando o número de iterações para que 80% das execuções atinjam o clique máximo, estamos essencialmente otimizando a velocidade de convergência e eficiência do algoritmo. A minimização da percentagem de execuções que terminam com um clique maximal abaixo do clique máximo sugere que o algoritmo tem maior probabilidade de encontrar a solução ótima, indicando um

intervalo de confiança de clique máximo mais consistente, e tendo maior probabilidade de produzir resultados confiáveis e de qualidade em várias execuções.

Os resultados gerais indicaram boa dispersão de ambas as métricas sobre as diferentes configurações de parâmetros. A análise de distribuição das métricas por parâmetro foi útil para segmentar e aprofundar os resultados obtidos.

- **Resultados de Métricas de Corretude e Velocidade por Parâmetro**

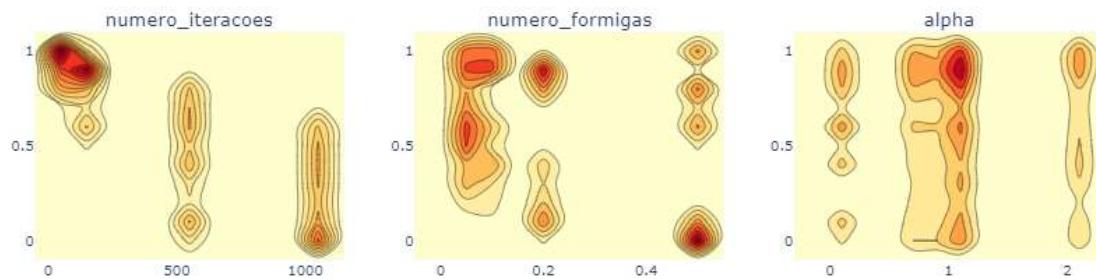


Figura 10 – Topográfico de Distribuição de Métrica de Corretude por Parâmetro (Grafo “Fácil”)

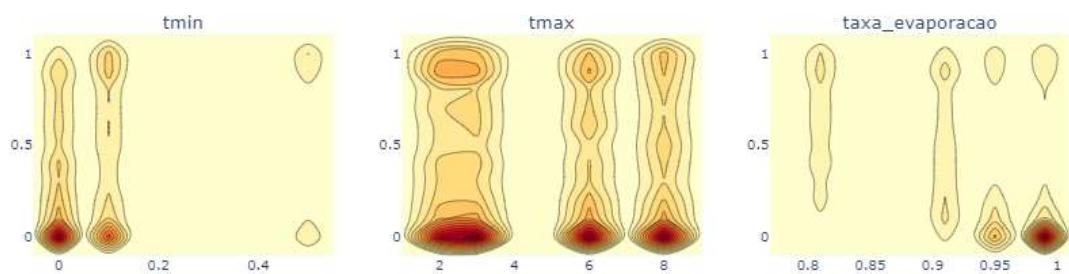
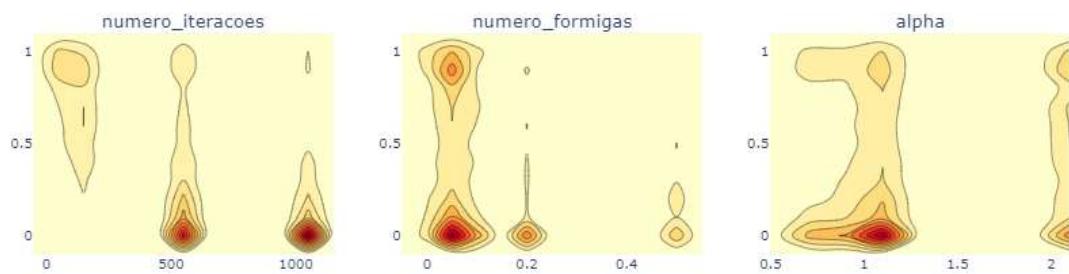


Figura 11 – Topográfico de Distribuição de Métrica de Corretude por Parâmetro (Grafo “Difícil”)

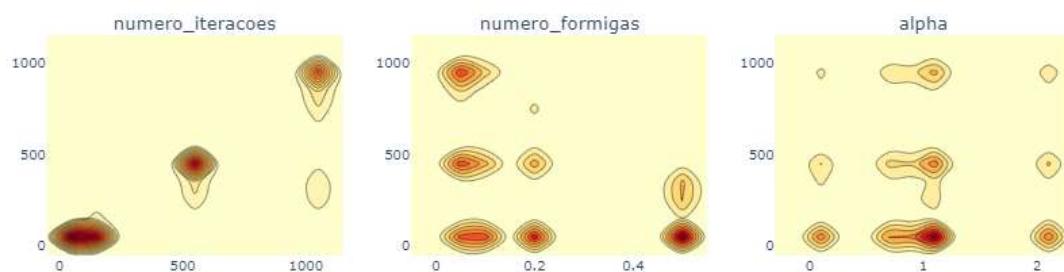


Figura 12 – Topográfico de Distribuição de Métrica de Velocidade por Parâmetro (Grafo “Fácil”)

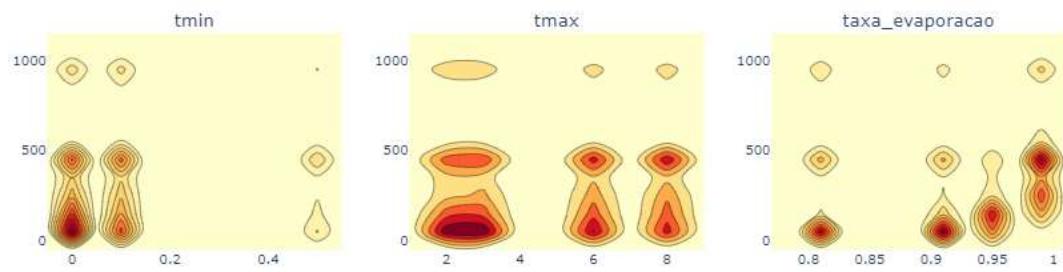
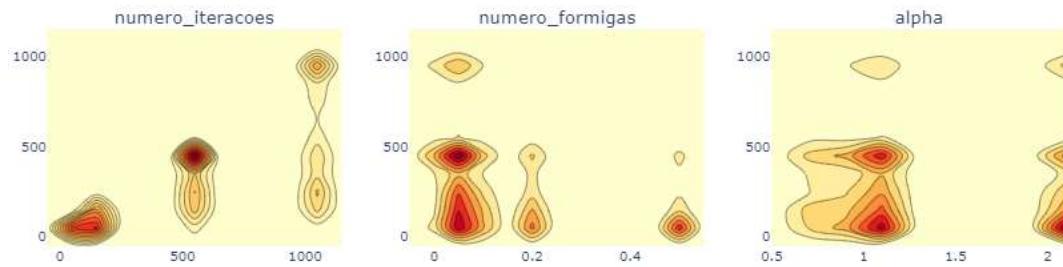


Figura 13 – Topográfico de Distribuição de Métrica de Velocidade por Parâmetro (Grafo “Difícil”)

Nesta seção, focamos nos parâmetros que se mostraram mais relevantes na análise das métricas de corretude e velocidade: número de iterações, número de formigas e taxa de evaporação.

- **Número de Iterações:** O número de iterações mostrou-se diretamente correlacionado com ambas as métricas. Em termos de corretude, um maior número de iterações permitiu que o algoritmo explorasse mais o espaço de soluções, reduzindo o número de execuções que terminavam com um clique maximal abaixo do clique máximo encontrado.
- **Número de Formigas:** O número de formigas teve impacto significativo nas duas métricas. Para a métrica de corretude, um maior número de formigas permitiu que mais soluções fossem exploradas em paralelo, aumentando a probabilidade de se alcançar o clique máximo. Para a métrica de velocidade, o aumento no número de formigas também resultou em uma maior rapidez na obtenção do clique máximo, dado que mais soluções puderam ser exploradas simultaneamente.
- **Taxa de Evaporação:** A taxa de evaporação teve um impacto importante em ambas as métricas. Uma taxa de evaporação de 0.99 mostrou-se eficaz na minimização da métrica de corretude, no entanto, para a métrica de velocidade, essa mesma taxa de evaporação resultou em execuções mais lentas. Isso porque, embora uma alta taxa de evaporação promova a exploração de novas soluções, ela também pode levar a uma exploração excessiva, causando atrasos na convergência para a clique máxima.

A taxa de evaporação desempenha um papel crucial no equilíbrio entre a exploração de novas soluções (o que melhora a corretude) e a exploração de soluções já conhecidas (o que melhora a velocidade). Um valor de 0.99 favorece a exploração, resultando em uma maior corretude, mas também em uma menor velocidade.

- **Resultados de Métrica Mista**

Para finalizar a análise dos resultados experimentais, introduzimos uma métrica mista que combina as métricas de corretude e velocidade. Para calcular a métrica mista, utilizamos o valor inverso das versões normalizadas das duas métricas, e multiplicamos esses valores entre si.

Essa métrica foi analisada nas execuções que conseguiram encontrar o clique máximo global (condição essencial para considerar uma execução como bem-sucedida). O objetivo da Métrica Mista é identificar as configurações de parâmetros que oferecem o melhor trade-off entre corretude e velocidade. Em outras palavras, estamos em busca dos parâmetros que, dentre as execuções bem-sucedidas, conseguem encontrar o clique máximo global de maneira mais rápida e consistente.

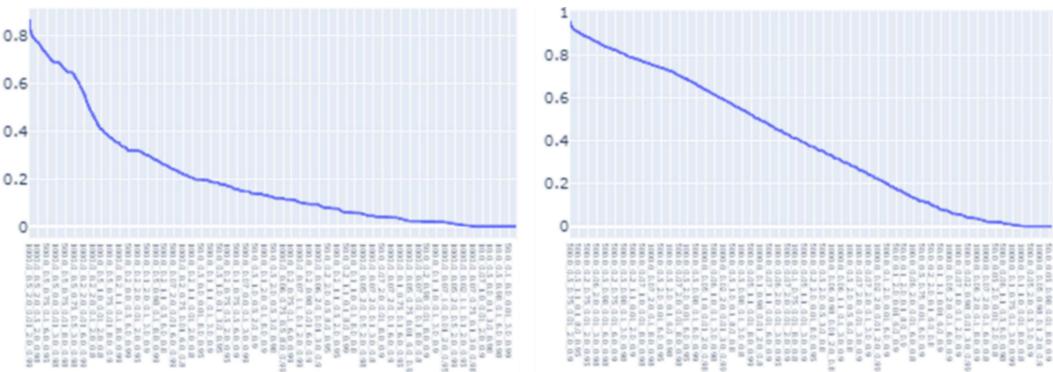


Figura 14 – Distribuição da Métricas Mista para Grafos “Fácil” (esq) e “Difícil” (dir)

Realizamos uma análise da distribuição da métrica mista em relação a cada par de parâmetros, agrupados por (tamanho da solução: número de formigas e iterações, exploration e exploitation: alpha e taxa de evaporação e domínio de feromônio: tau_max e tau_min).

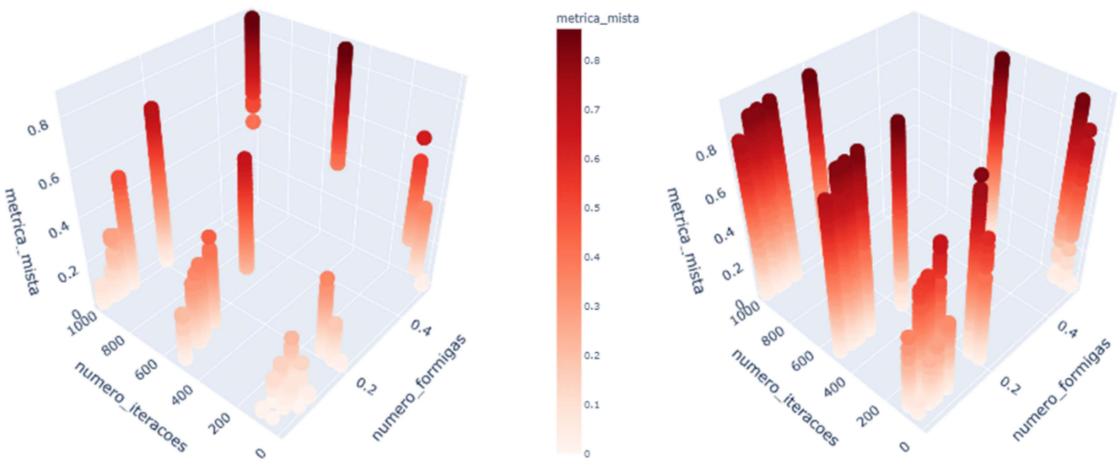


Figura 15 – Distribuição de métrica mista para parâmetros de tamanho da solução (formigas / iterações)

Os resultados confirmaram as expectativas levantadas na seção anterior: para ambos os grafos modelados, observamos uma relação direta entre a Métrica Mista e o número de formigas e iterações. A Figura 15 mostra que valores menores da Métrica Mista correspondem a regiões de baixos valores dos dois parâmetros. À medida que aumentamos o número de formigas e iterações,

os valores da Métrica Mista também aumentam, atingindo o valor máximo quando ambos os parâmetros estão em seus valores máximos.

Notavelmente, para o grafo "difícil", as configurações com o número máximo de formigas e iterações não conseguiram encontrar o clique máximo global. Por outro lado, no grafo "fácil", essas mesmas configurações resultaram nas soluções de melhor qualidade encontradas.

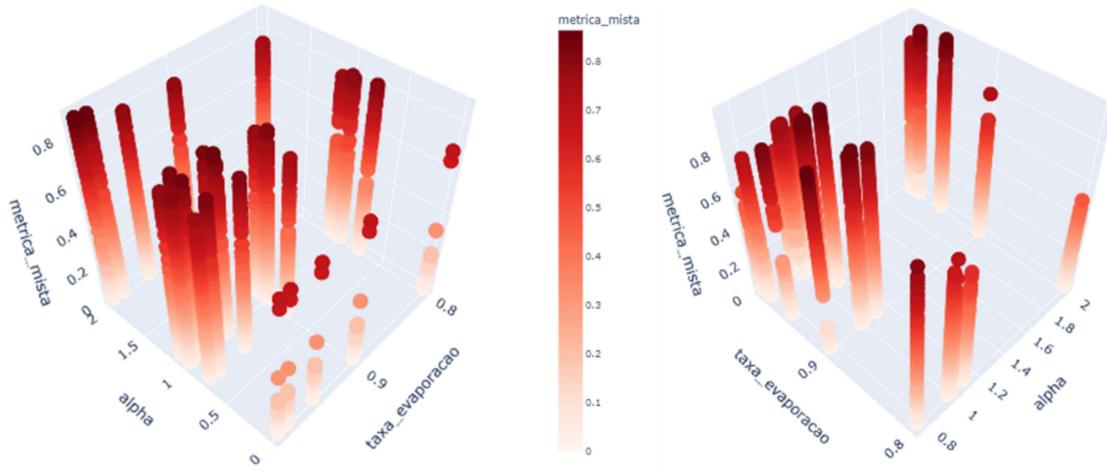


Figura 16 – Distribuição de métrica mista para parâmetros de exploration/explotation (α / taxa_evaporação)

Confirmado nossas expectativas, a taxa de evaporação não apresentou um impacto significativo na Métrica Mista. Isto porque essa métrica representa um trade-off entre velocidade e corretude, e a taxa de evaporação afeta essas duas componentes de maneiras opostas. Observamos que valores muito baixos de α produzem resultados piores, enquanto valores próximos de 1 resultam nos melhores resultados.

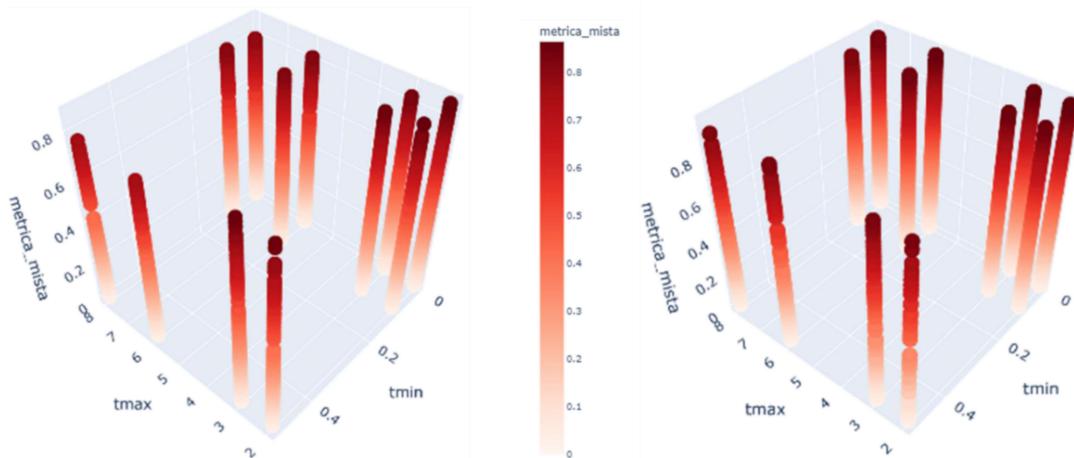


Figura 17 – Distribuição de métrica mista para parâmetros de domínio feromônio (τ_{min} / τ_{max})

Finalmente, observamos que os parâmetros τ_{min} e τ_{max} não exercem uma influência significativa sobre a Métrica Mista. As diferentes configurações desses parâmetros resultaram em

valores próximos para a Métrica Mista, sugerindo que a faixa de variação de feromônio não é um fator crucial para o desempenho do algoritmo ACO no problema de maximização de clique.

Fizemos também plots explicando as métricas individuais para um dado experimento em um grafo dentro de todo o intervalo de confiança, podendo ver as cliques máximas por iteração, taxa média de feromônio,e as repetições de clique por tamanho

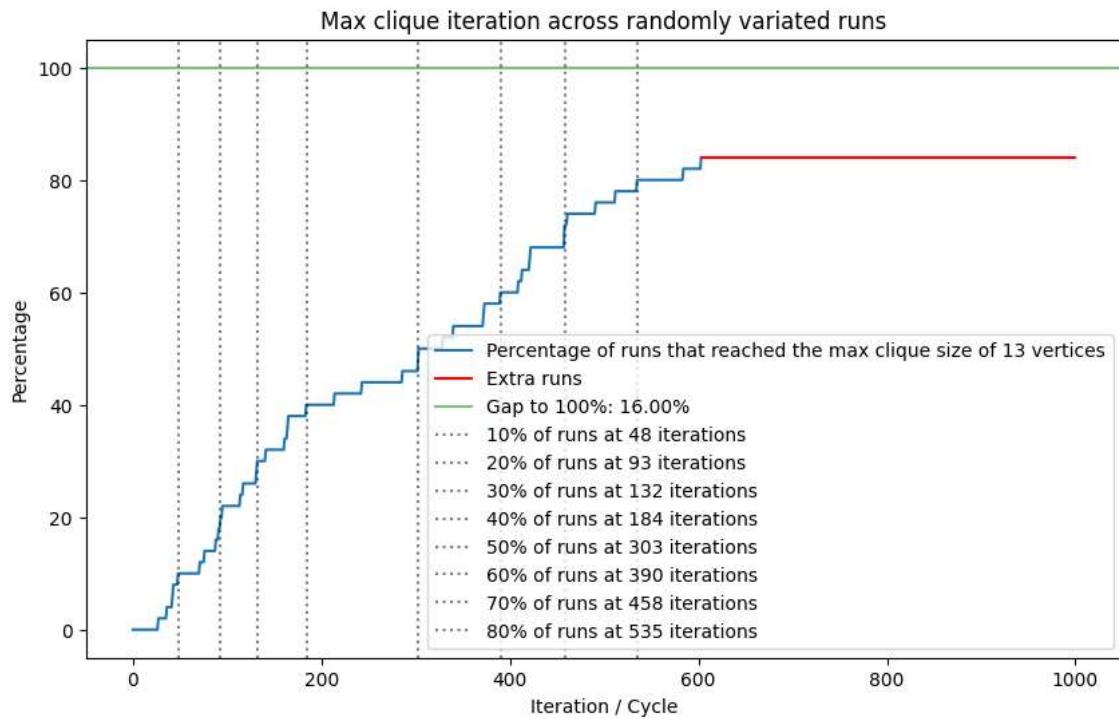


Figura 18: métricas de velocidade e corretude para um exemplo de grafo fácil, ilustrando também a convergência em cliques de tamanho menor que o máximo

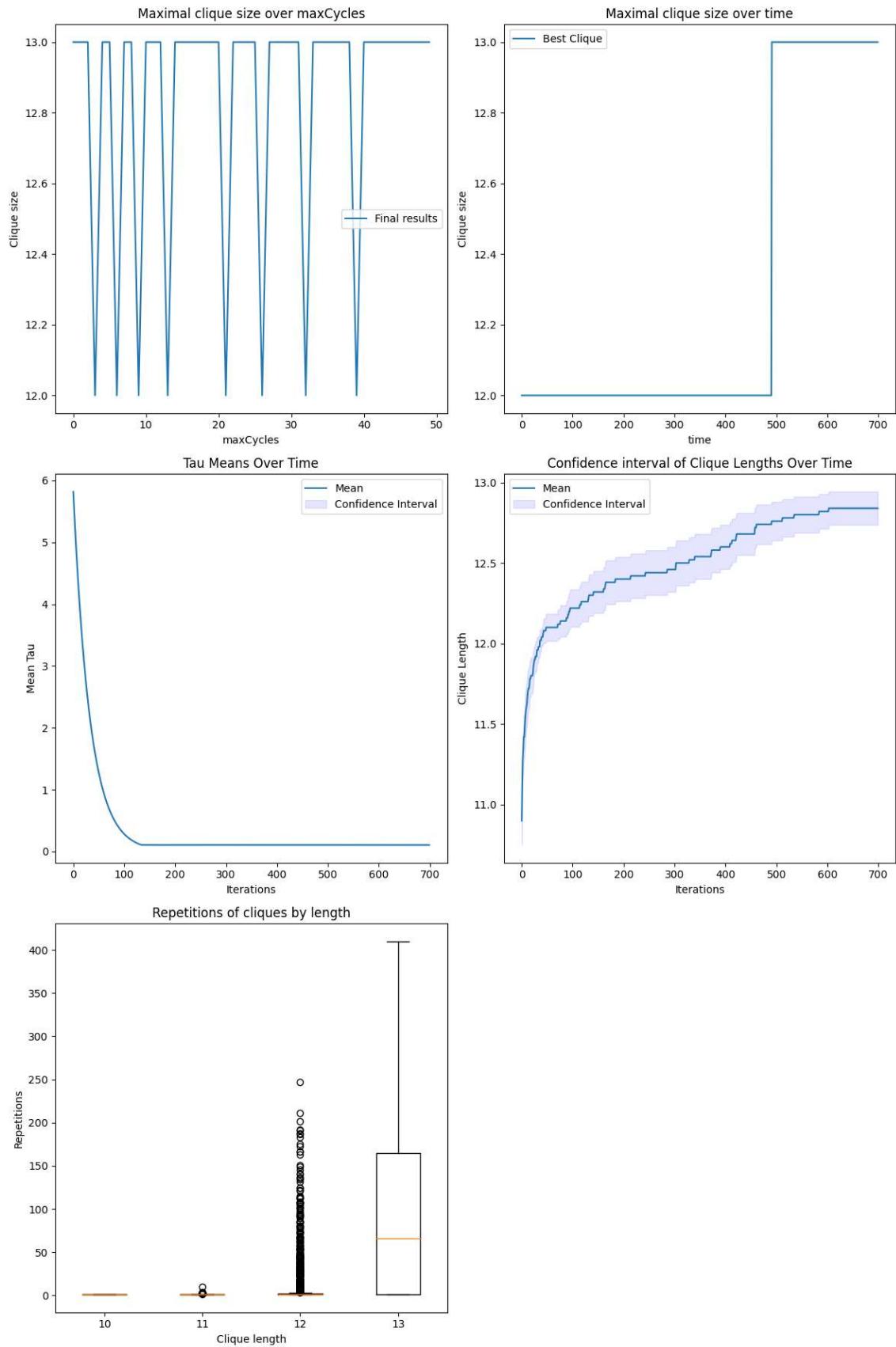


Figura 19: Métricas individuais ao longo do intervalo de confiança para uma dada experimentação

Conclusões

Analisamos mais de 10 000 permutações de parâmetros do algoritmo para o problema da clique máxima em dois grafos, concluindo que os resultados experimentais do artigo referência estão bons, podendo melhorar apenas no número de formigas, que depende bastante da natureza do grafo estudado, sendo então interessante ir além do que foi sugerido de 30 formigas, que não performou tão bem nos nossos exemplos. Acabou sendo melhor utilizar uma métrica de formigas por vértice, de forma que o cálculo de formigas seja dinâmico. Talvez seja interessante em trabalhos futuros verificar como incluir o número de arestas nessa conta, generalizando para mais grafos

Referências

Solnon, Christine & Fenet, Serge. (2006). A study of ACO capabilities for solving the Maximum Clique Problem. *Journal of Heuristics*. 12. 155-180.
10.1007/s10732-006-4295-8.