Uma implementação Paralela do Algoritmo Colônia de Formiga para Seleção de Instâncias

Maria Eduarda Oliveira de Brito[1], **Cristiane Neri Nobre (orientadora)**[1]

[1]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Av. Dom José Gaspar, 500 – 30.535-610 – Belo Horizonte – MG – Brazil

maria.brito@sga.pucminas.br, nobre@pucminas.br

Abstract. *With the increase in data volume, computational complexity has grown, making data analysis challenging. A common solution is to reduce the training set, and a widely adopted technique is instance selection. However, many of these techniques are computationally complex, resulting in extended execution times. In this study, we applied parallelism to the ant colony optimization (ACO) instance selection algorithm to reduce execution time. The performance of the parallel version of ACO was evaluated on various datasets with Machine Learning (ML) algorithms such as Decision Tree, Random Forest, and SVM. Overall, we achieved considerable results, with an average 52% reduction in execution time compared to the sequential ACO performance and the original base. Additionally, the overall data quality was maintained.*

Resumo. *Com o aumento do volume de dados, a complexidade computacional cresceu, tornando a análise de dados desafiadora. Uma solução comum é reduzir o conjunto de treinamento, e uma técnica amplamente adotada é a seleção de instâncias. No entanto, muitas dessas técnicas são computacionalmente complexas, o que resulta em tempos de execução prolongados. Neste estudo, aplicamos paralelismo ao algoritmo de seleção de instâncias com colônia de formigas (ACO) para reduzir o tempo de execução. O desempenho da versão paralela do ACO foi avaliada em diversos conjuntos de dados com algoritmos de Aprendizado de Máquina (ML) como Árvore de Decisão, Random Forest e SVM. No geral, alcançamos resultados consideráveis, com uma redução média de 52% no tempo de execução em comparação com o desempenho do ACO sequencial e a base original. Além disso, a qualidade dos dados no geral foi mantida.*

1. Introdução

Aprendizado de Máquina (ML) é uma área da Inteligência Artificial que busca desenvolver algoritmos e modelos capazes de aprender a partir de dados, ao invés de serem programados de forma explícita (Shojaee et al., 2023).

Uma das tarefas mais comuns em Aprendizado de Máquina é a classificação, utilizada em diversas áreas como reconhecimento de padrões, detecção de fraudes e diagnóstico médico (Czarnowski, 2021). Essa tarefa consiste em atribuir uma ou mais categorias (rótulos) a um conjunto de dados com base em exemplos rotulados. Esse conjunto é conhecido como conjunto de treinamento e é utilizado para induzir um modelo capaz de fazer previsões precisas sobre novas instâncias não vistas.

O sucesso da classificação depende da qualidade do conjunto de treinamento, já que o uso excessivo de amostras pode levar a redundância e aumentar a complexidade computacional e de memória durante a fase de treinamento dos algoritmos de aprendizado de máquina (Czarnowski, 2021). Esse aumento na complexidade pode tornar o processo impraticável para algoritmos de ML como Máquina de Vetores de Suport (SVM), especialmente quando o número de instâncias é muito grande. Nesse sentido, há uma demanda crescente por métodos eficientes de seleção de instâncias que possam reduzir o tempo de treinamento e otimizar o desempenho desses modelos.

A seleção de instâncias é um problema comum em muitas áreas da ciência de dados e aprendizado de máquina. O objetivo é encontrar um subconjunto de instâncias que otimize o desempenho do modelo preditivo e/ou reduza o tempo de processamento (de Melo Menezes et al., 2021). Existem várias abordagens para a seleção de instâncias, incluindo técnicas de amostragem aleatória, algoritmos genéticos (Kordos et al., 2022), e colônia de formigas (Akinyelu et al., 2020). A seleção cuidadosa das instâncias de dados tem um impacto substancial no processo de aprendizado de máquina. Investir tempo na seleção criteriosa pode resultar em modelos mais precisos e eficientes. Ao selecionar instâncias relevantes, é possível obter um conjunto de dados mais representativo, o que geralmente contribui para um modelo mais robusto e generalizado.

Nesse contexto, investigaremos o uso da Colônia de Formigas (Ant Colony Optimization - ACO), que se mostrou uma técnica promissora para a seleção de instâncias, capaz de encontrar soluções ótimas de forma eficiente (Shojaee et al., 2023). A colônia de formigas é uma técnica de otimização inspirada no comportamento coletivo das formigas reais. As formigas individuais são consideradas agentes simples, mas juntas são capazes de encontrar caminhos eficientes para a comida ou para o ninho. O algoritmo da colônia de formiga utiliza uma heurística baseada em feromônios para guiar a busca por soluções ótimas. A ideia é que as formigas deixem um rastro de feromônios ao caminhar, e outras formigas usem esse rastro para encontrar o caminho mais curto (Shojaee et al., 2023).

A Seleção de Instâncias com Colônia de Formigas apresenta várias vantagens em comparação com outras técnicas renomadas de seleção de instâncias, como IB2, IB3, Algoritmo Genético (GA) e a família de algoritmos DROP. Um estudo comparativo, realizado por Hu et al. (2012), investigou a eficácia do ACO em relação ao GA na tarefa de calcular a ordem genética dos genes associados à doença de Alzheimer. O resultado revelou que o ACO obteve um desempenho superior. Isso pode ser atribuído à sua capacidade de explorar de maneira mais eficaz o espaço multidimensional de busca.

Por ser baseada em uma abordagem metaheurística, ela é capaz de lidar com problemas complexos e não lineares, além de encontrar soluções ótimas ou próximas do ótimo em um tempo razoável (Dorigo et al., 2006). Além disso, ela é capaz de lidar com grandes bases de dados, reduzindo significativamente o tamanho do conjunto de dados sem perda de informação.

No entanto, foi observado no artigo de Gonçalves e Nobre (2022) que o processamento do ACO, desenvolvido no Laboratório de Inteligência Computacional Aplicada (LICAP) da PUC Minas, torna-se muito lento quando utilizado em bases de dimensionalidade maiores do que 1.573 instâncias, podendo levar horas ou até mesmo dias para ser concluído.

Diante disso, o objetivo deste trabalho apresenta uma abordagem de seleção de instâncias baseada em colônia de formigas com paralelismo, focando na redução do tempo de execução do ACO, mantendo a mesma qualidade da solução já comprovada em Gonçalves e Nobre (2022). Essa abordagem visa tornar o ACO ainda mais viável para a redução de custos computacionais em modelos de aprendizado de máquina com grandes bases de dados.

O restante deste trabalho está organizado da seguinte forma: A Seção 2 traz definições importantes para o entendimento do assunto abordado, tais como Seleção de Instâncias com ACO e diferentes tipos de paralelismo usados no ACO. A Seção 3 apresenta os trabalhos relacionado ao tema deste artigo. Na Seção 4, é descrita a metodologia adotada neste artigo, englobando em 4.1 a descrição das bases de dados que serão empregadas, e em 4.2 apresenta a versão paralela do algoritmo de Seleção de Instâncias com Colônia de Formigas desenvolvida neste trabalho, além dos parâmetros empregados no ACO. Os resultados alcançados são divulgados na Seção 5, acompanhados das respectivas análises. A Seção 6 traz as considerações finais, com limitações encontradas no trabalho e propostas de trabalhos futuros.

2. Referencial Teórico

2.1. Seleção de Instâncias com Colônia de Formigas

Segundo Kordos et al. (2022), a seleção de dados, que inclui seleção de recursos e instâncias, são uma etapa importante do processo de pré-processamento de dados utilizada em problemas de aprendizado de máquina.

A seleção de instâncias é uma etapa crucial do pré-processamento de dados em aprendizado de máquina, que tem como objetivo reduzir o nível de ruído dos dados selecionando um subconjunto de instância representativo do conjunto de dados original, de forma a reduzir o tamanho do conjunto de dados e melhorar a eficiência e a qualidade do modelo (Akinyelu et al., 2020).

No entanto, a seleção de instâncias é um problema NP-difícil, o que inviabiliza a análise de todas as combinações possíveis de instâncias selecionadas dentro de um tempo razoável (Kordos et al., 2022). Por isso, é necessário recorrer a técnicas *heurísticas* e *metaheurísticas* para realizar a seleção de instâncias. Nesse contexto, a heurística baseado em ACO proposta por Dorigo et al. (2006) é uma das alternativas utilizadas para encontrar uma solução aproximada para o problema e foi empregada neste trabalho. Um pseudocódigo de ACO básico é mostrado em Algoritmo 1.

A Seleção de Instâncias com Colônia de Formigas é inspirada no comportamento das formigas em busca de alimento (Akinyelu et al., 2020). As formigas são capazes de encontrar o caminho mais curto entre o ninho e uma fonte de alimento, depositando feromônios que atraem outras formigas para seguir o mesmo caminho (Dorigo et al., 2006). O ACO, proposto inicialmente por Marco Dorigo em seu doutorado, é um método que utiliza um algoritmo metaheurístico que simula esse comportamento das formigas, buscando encontrar o subconjunto de instâncias mais representativo dos dados originais (de Melo Menezes et al., 2021).

O processo de Seleção de Instâncias com Colônia de Formigas começa com a criação de uma população inicial de soluções candidatas, composta por subconjuntos

aleatórios de instâncias do conjunto de dados original. Em seguida, é aplicado um algoritmo de busca baseado no ACO, que utiliza uma matriz de feromônios para guiar a seleção das instâncias mais relevantes.

Algoritmo 1: Pseudocódigo ACO

```
I ← conjunto_de_dados;
InicializarFeromonios();
melhor_solucao ← ∅;
repita
    solucao_atual ← ConstruirSolucao(I);
    AplicarBuscaLocal(solucao_atual) /*opcional*/
    se VerificarQualidade(solucao_atual) >
     VerificarQualidade(melhor_solucao) então
        melhor_solucao ← solucao_atual;
    fim
    AtualizarFeromonio(solucao_atual);
até condição seja alcançada;
Resultado: melhor_solucao
```

Fonte: Adaptado de Dorigo et al. (2006) e Anwar et al. (2015)

A matriz de feromônios é atualizada a cada iteração do algoritmo, levando em consideração a qualidade das soluções encontradas até o momento. As formigas virtuais são incentivadas a seguir caminhos que levam a soluções mais promissoras, depositando feromônios nas instâncias que fazem parte dessas soluções, ao mesmo tempo, a matriz de feromônios evapora gradualmente, diminuindo a influência das soluções antigas e incentivando a busca por novas soluções (González et al., 2022).

O processo de seleção continua até que um critério de parada seja alcançado, como um número máximo de iterações ou uma taxa de convergência das soluções. A solução final é o subconjunto de instâncias que apresentou o melhor desempenho em relação aos critérios de avaliação definidos.

2.2. Paralelismo no método Colônia de Formigas

De acordo com Astudillo et al. (2022), o paralelismo envolve a execução simultânea de várias tarefas em diferentes processadores ou núcleos de processamento. Essa técnica é fundamental para melhorar a eficiência e escalabilidade de sistemas computacionais. Além disso, Astudillo et al. (2022) destacam duas modalidades de paralelismo: o de dados, que se refere à execução simultânea da mesma instrução ou tarefa em diferentes conjuntos de dados dentro de cada núcleo de processamento, e o de tarefas, que é uma forma de paralelização realizando múltiplas funções (ou múltiplas instâncias de uma função) ao mesmo tempo em núcleos separados.

A paralelização do algoritmo ACO é uma área bem pesquisada, devido à natureza inerentemente distribuída da técnica (Peake et al., 2019). No ACO, o paralelismo pode ser usado para executar diferentes colônias de formigas em paralelo, cada uma delas explorando diferentes subconjuntos de instâncias (Shojaee et al., 2023). Isso permite que o algoritmo encontre soluções mais rapidamente e aumente a probabilidade de encontrar soluções melhores.

Adicionalmente, é importante destacar a natureza intrinsecamente paralela do ACO, uma vez que cada formiga é capaz de construir seu caminho de maneira independente das demais. Isso possibilita a implementação do algoritmo de forma paralela, tanto nos dados quanto nos domínios populacionais (González et al., 2022).

3. Trabalhos Relacionados

Muitos esforços científicos têm sido direcionados para a etapa de pré-processamento de seleção de instâncias. Com esse objetivo, diversos trabalhos na literatura têm utilizado a heurística de Seleção de Instâncias com Colônia de Formigas (ACO), e várias tentativas de redução do tempo de execução têm sido realizadas com base no uso de técnicas de paralelismo.

Anwar et al. (2015) propuseram uma extensão do algoritmo ADR-Miner para produzir modelos de classificação mais precisos. A modificação no ADR-Miner permitiu a utilização de dois algoritmos de classificação diferentes, na primeira e segunda fases, respectivamente, para avaliação dos conjuntos reduzidos candidatos e construção do classificador final. Foram utilizados cinco algoritmos de classificação diferentes sendo eles Máquina de Vetores de Suporte (SVM), K-NN, Árvore de Decisão C4.5, Algoritmo de Indução de Regra Ripper e o classificador Naive-Bayes, e todas as possíveis combinações entre eles para testar a eficácia dessa abordagem, sendo que o ADR-Miner com emparelhamento SVM-SVM obteve a melhor classificação. Os resultados mostraram que essa abordagem melhora a precisão preditiva em vários casos e o uso de um par de algoritmos de classificação diferentes em cada fase do ADR-Miner obteve melhores resultados do que o uso do mesmo algoritmo em ambas as fases.

Peake et al. (2019) apostaram em uma variante restrita da matriz de feromônios com complexidade de memória linear, que armazena valores de feromônios apenas para membros de um conjunto candidato de próximos movimentos. Além disso, foram avaliados dois métodos de seleção para movimentos fora do conjunto candidato, resultando em um tempo razoável e melhores soluções no Art TSP (Problema do Caixeiro Viajante) por meio do ACO. Vale ressaltar que essa implementação foi a primeira avaliação de uma implementação paralela do MAX-MIN Ant System que é um algoritmo baseado no ACO. É fundamental ressaltar que os testes foram realizados em até 100.000 instâncias, e somente nessa quantidade de instâncias a solução se degradou ligeiramente em comparação com o método sequencial, apresentando uma diferença mínima de 2%. Vale destacar que, para quantidades de instâncias inferiores a 100.000, os resultados foram consistentes.

Diferentemente de trabalhos que visam apenas reduzir o tempo de execução do ACO por meio de paralelismo, em de Melo Menezes et al. (2021), os autores mostram como o *Muenster Skeleton Tool for High-Performance Code Generation* (Musket), uma ferramenta de esqueletização baseada em *Domain Specific Language* (DSL), pode ser usada para implementar o ACO em *Graphics Processing Unit* (GPU) de forma paralela. O Musket gera código de alto desempenho, com tempos de execução comparáveis aos de implementações de baixo nível, facilitando a paralelização para o programador. Os experimentos realizados neste trabalho incluem instâncias do Problema do Caixeiro Viajante (TSP) e foram realizados para até 1291 cidades. A implementação ACO com o Musket obteve resultados consideráveis em termos de tempo de execução, mostrando que o Musket é adequado para o desenvolvimento do ACO.

Em Zhou et al. (2018), os autores propõem o uso da computação SIMD (do inglês, Single Instruction, Multiple Data) baseada em CPU (do inglês, Central Processing Unit) para ACOs, que foi investigada poucas vezes em literaturas anteriores. O modelo de vetor paralelo proposto acelera a construção do *tour* de cada formiga por meio de instruções vetoriais, com cada formiga mapeada para um núcleo de CPU. Além disso, é proposta uma nova abordagem de seleção proporcional de fitness, denominada Roleta baseada em vetores (VRW), na etapa de construção do circuito. O algoritmo é testado em instâncias do Problema do Caixeiro Viajante (TSP) padrão e comparado com ACOs baseados em GPU de alto desempenho, demonstrando o forte potencial de ACOs paralelos baseados em CPU.

Em Skinderowicz (2020), assim como em Peake et al. (2019), é usado o eficiente algoritmo MAX-MIN Ant System (MMAS) de Otimização de Colônias de Formigas (ACO) como objeto de pesquisa. O MMAS é implementado pelos autores em GPUs, com novas ideias discutidas para melhorar a implementação paralela baseada em GPU. Uma implementação eficiente da estrutura da lista tabu é apresentada, e seis variantes MMAS são avaliadas em problemas do Caixeiro Viajante (TSP). Os resultados mostram que a implementação MMAS dos autores é competitiva com implementações ACO paralelas baseadas em GPU e CPU *multi-core*. Para a GPU Nvidia V100 Volta, os tempos obtidos foram significativamente menores sendo $7,18\times$ e $21,79\times$, e a variante MMAS mais rápida gerou mais de 1 milhão de soluções candidatas por segundo ao resolver uma instância de 1.002 cidades.

Pascariu et al. (2022) abordam o problema de gerenciamento de tráfego ferroviário em tempo real (rtRTMP), propondo uma solução para o Problema de Seleção de Roteamento de Trens (TRSP), a fim de reduzir o tamanho das instâncias rtRTMP. O modelo TRSP aprimorado leva em conta as restrições de tempo de reutilização de material rodante e estimativa de trem, propagação de atraso, e um algoritmo paralelo de Otimização de Colônias de Formigas (ACO) é proposto (ACO-TRSP) para acelerar a exploração do espaço de busca. Uma campanha computacional completa é realizada em um estudo de caso francês com distúrbios de horário e interrupções de infraestrutura para analisar o impacto do modelo e algoritmo TRSP no rtRTMP. Os resultados mostram que o modelo apresentado melhora a correlação entre soluções TRSP e rtRTMP, e o algoritmo ACO-TRSP proposto supera o algoritmo do estado-da-arte.

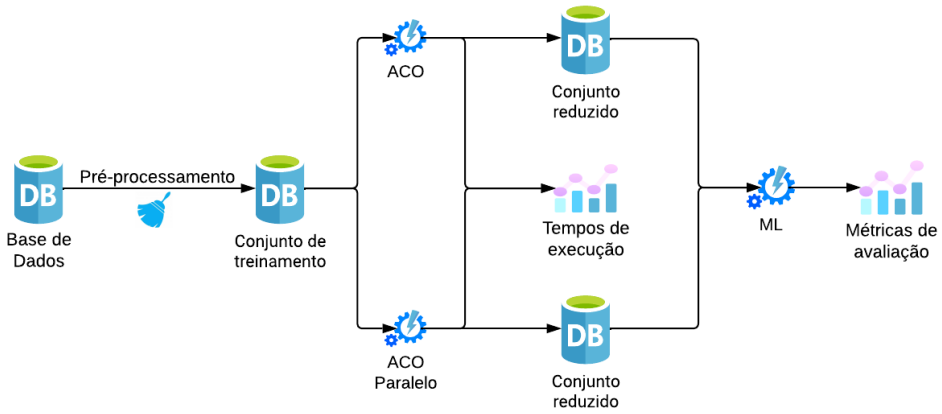
Este trabalho assemelha-se aos estudos apresentados nesta seção, pois buscou uma abordagem complementar, especificamente aprimorando a técnica do ACO por meio do emprego do paralelismo. No entanto, distingue-se dos estudos anteriores devido à sua base na exploração das *threads* utilizando a linguagem de programação *Python*, por meio da biblioteca *joblib*, conhecida e consolidada no mercado.

4. Metodologia

A metodologia empregada neste trabalho compreendeu várias etapas, iniciando com a coleta de bases de dados e prosseguindo com o pré-processamento dessas bases, detalhado na Seção 4.1. Em seguida, realizou-se a execução tanto do algoritmo utilizado em Gonçalves e Nobre (2022) quanto da versão paralela desenvolvida neste estudo. Posteriormente, foram analisados os tempos de execução para avaliar a eficácia dos resultados obtidos. Por fim, com os conjuntos de dados reduzidos em mãos, eles foram submetidos

a algoritmos de ML, como SVM, Árvore de Decisão e *Random Forest*, a fim de verificar se houve melhoria na qualidade do modelo ou se pelo menos a qualidade original foi mantida. A Figura 1 ilustra as etapas descritas acima.

Figura 1. Metodologia



4.1. Descrição da base de dados

Com o intuito de avaliar o desempenho da nova versão da Colônia de Formigas na seleção de instâncias, este estudo analisou um conjunto composto por cinquenta e nove bases públicas de dados, bem como duas bases de dados de natureza privada. Essas bases de dados exibem uma considerável variabilidade em relação ao número de instâncias nelas contidas.

Para viabilizar a aplicação do ACO, foi necessário realizar a conversão de todos os atributos categóricos presentes nas bases de dados originais para uma representação numérica. Para esse propósito, optou-se pela aplicação das técnicas de codificação conhecidas como *ordinal encoding* e *one-hot encoding*, as quais possibilitam a transformação dos valores categóricos em números inteiros ordinais. É relevante destacar que o ACO é concebido para operar exclusivamente com atributos numéricos, razão pela qual essa fase de pré-processamento revelou-se indispensável.

A Tabela 1 na parte referente a "Descrição das bases de dados" apresenta informações detalhadas sobre o número de instâncias, tipos de atributos de entrada, quantidade total de atributos e links para acesso às bases de dados que serão utilizadas neste estudo. Destaca-se que a classificação dos atributos de entrada foi baseada nas descrições fornecidas pelos autores das bases de dados.

A primeira base de dados privada é de depressão em crianças e adolescentes, com idades entre 10 e 16 anos. Nesta pesquisa, foram entrevistados 363 adolescentes. A base de dados é composta por 87 atributos que contêm informações essenciais para a análise da depressão nesse grupo etário. Esses atributos abrangem diversos aspectos, incluindo dados demográficos, sociais e relacionados à saúde mental. Além disso, os casos de depressão foram categorizados em dois grupos distintos.

A segunda base consiste em entrevistas com 18.185 presos, dos quais foram selecionadas 14.499 instâncias. Ela engloba 2.986 atributos numéricos, sendo 2.046 deles

discretos e 940 contínuos, conforme relatado por Oliveira et al. (2023). Após o pré-processamento da base, foram mantidos somente 130 recursos para análise.

Tabela 1. Descrição das bases de dados e tempos de execução dos algoritmos

		Descrição das bases de dados								Análise dos tempos de execução		
Base de Dados	**N° Instâncias**	**Recurso Contínuo**	**Recurso Discreto**	**Recurso Cat. Ord.**	**Recurso Cat.Não Ord.**	**Recurso Cat.Bin.**	**Quantidade Total**	**Dados Ausentes**	**Link Da Base**	**Temp Sequencial**	**Temp Paralelo**	**Ganho Em %**
Post-Operative Patient	90	-	-	4	3	2	9	3	Link	1.03 s	6.18 s	-500.0
Breast Cancer Coimbra	116	7	2	-	-	-	0	0	Link	1.87 s	6.60 s	-252.9
Higher Education	145	-	1	12	14	5	32	0	Link	3.12 s	7.06 s	-126.2
Iris	150	4	-	-	1	-	5	0	Link	3.55 s	7.22 s	-103.3
Drug Classification	200	1	1	1	1	2	6	0	Link	7.31 s	8.35 s	-14.2
Sonar	208	60	-	-	-	1	61	0	Link	8.34 s	8.73 s	-4.6
Glass Classification	214	9	-	-	1	-	10	0	Link	8.70 s	8.99 s	-3.3
Heart	270	2	4	1	3	4	14	0	Link	15.65 s	11.39 s	27.2
Heart Failure	299	2	5	-	-	6	13	0	Link	20.09 s	13.18 s	34.3
Heart Attack Analysis	303	1	5	1	3	4	14	0	Link	21.12 s	13.56 s	35.7
Haberman	306	-	3	-	-	1	4	0	Link	21.57 s	13.48 s	37.5
Lung Cancer	309	-	1	-	-	15	16	0	Link	22.38 s	13.99 s	37.4
Vertebral Column	310	6	-	-	-	1	7	0	Link	22.65 s	13.70 s	39.5
Liver Disorders	345	1	5	-	-	1	7	0	Link	30.59 s	16.47 s	46.1
Ionosphere	351	32	-	-	-	2	34	0	Link	32.42 s	17.10 s	47.2
Depressão	363	86	-	-	-	1	87	0	Privada	37.41 s	18.26 s	51.1
Student Performance	397	-	5	8	7	13	33	0	Link	44.92 s	21.26 s	52.6
Cirrhosis	418	9	2	1	2	5	19	1033	Link	52.40 s	23.09 s	55.9
Musk	476	-	166	-	-	1	167	0	Link	78.47 s	31.32 s	60.0
KC2	522	17	4	-	-	1	22	0	Link	102.27 s	40.20 s	60.6
Monk3	554	-	-	-	4	3	7	0	Link	115.51 s	46.08 s	60.1
Monk1	556	-	-	-	4	3	7	0	Link	119.83 s	46.26 s	61.3
Breast Cancer Wisconsin	569	30	-	-	-	1	31	0	Link	125.82 s	48.71 s	61.2
Indian Patient's Liver	583	5	4	-	-	2	11	0	Link	128.93 s	52.24 s	59.4
Monk2	601	-	-	-	4	3	7	0	Link	142.04 s	55.53 s	60.9
Australian	690	6	-	-	4	5	15	0	Link	209.33 s	78.86 s	62.3
Diabetes	768	2	6	-	-	1	9	0	Link	288.59 s	103.78 s	64.0
Mammographic Masses	830	-	1	2	2	1	6	0	Link	356.31 s	126.41 s	64.5
TicTacToe	959	-	-	-	9	1	10	0	Link	543.51 s	184.68 s	66.0
Tokyo1	959	37	4	-	-	2	43	0	Link	553.91 s	184.89 s	66.6
PC1	1109	17	4	-	-	1	22	0	Link	868.00 s	272.90 s	68.5
Hill Valley	1212	100	-	-	-	1	101	0	Link	18.0 min	5.0 min	72.2
Cyber Security	1247	-	3	2	6	-	11	0	Link	19.0 min	6.0 min	68.4
Titanic	1309	1	3	1	4	2	11	0	Link	23.0 min	7.0 min	69.5
Mofn 3 7 10	1324	-	-	-	-	11	11	0	Link	23.0 min	7.0 min	69.5
Banknote	1372	4	-	-	-	1	5	0	Link	26.0 min	8.0 min	69.2
Contraceptive Method	1473	-	2	3	2	3	10	0	Link	32.0 min	9.0 min	71.8
Yeast	1484	7	-	-	1	1	9	0	Link	33.0 min	9.0 min	72.7
Fetal Health	2126	9	13	-	1	-	23	0	Link	98.0 min	28.0 min	71.4
Marketing Analysis	2205	1	16	1	1	7	26	24	Link	108.0 min	32.0 min	70.3
Seismic	2584	-	11	-	2	3	16	0	Link	173.0 min	50.0 min	71.0
Room Occupancy	2665	5	-	-	-	1	6	0	Link	185.0 min	54.0 min	70.8
Mobile pricing	3000	4	15	-	-	2	21	0	Link	275.0 min	79.0 min	71.2
Splice	3186	-	-	-	1	180	181	0	Link	324.0 min	97.0 min	70.0
Vaccines	3353	8	1	13	3	17	42	999	Link	317.0 min	93.0 min	70.6
Abalone	4177	7	1	-	1	-	9	0	Link	767.0 min	219.0 min	71.44
Spambase (Spam)	4601	57	-	-	-	1	58	0	Link	971.0 min	293.0 min	69.8
Employee Future	4653	-	2	3	1	3	9	0	Link	1007.0 min	305.0 min	69.7
Brain Stroke	4981	2	1	-	2	6	11	0	Link	1234.0 min	390.0 min	68.3
Banana	5300	2	-	-	-	1	3	0	Link	1502.0 min	481.0 min	67.9
Optdigits	5620	-	60	-	-	3	63	0	Link	1864.0 min	616.0 min	66.9
Shill Bidding	6321	8	1	-	-	1	10	0	Link	2556.0 min	835.0 min	67.3
Wine	6463	11	-	1	-	1	13	38	Link	2686.0 min	856.0 min	68.1
Twonorm	7400	20	-	-	-	1	21	0	Link	4137.0 min	1330.0 min	67.8
Mushrooms	8124	-	-	-	17	6	23	2480	Link	5532.0 min	2122.0 min	61.6
Zomato	9497	3	4	1	7	3	18	0	Link	9008.0 min	3351.0 min	62.7
Home Equity	10459	-	23	-	-	1	24	0	Link	11500.0 min	4718.0 min	58.9
Phishing Websites	11055	-	8	-	-	23	31	0	Link	14587.0 min	6277.0 min	56.9
Body Performance	13393	9	1	1	-	1	12	0	Link	26092.0 min	11419.0 min	56.2
Presos	14252	129	1	-	-	-	130	0	Privada	31808.0 min	13910.0 min	56.2
Eeg Eye State	14980	14	-	-	-	1	15	0	Link	35499.0 min	15405.0 min	56.6

*Esta tabela está ordenada de forma crescente, por número de instâncias.
Cat. Não Ord - Recursos Categóricos Não Ordinais.
S - Segundos.
Temp - Tempo de execução do algoritmo.
Cat. Ord - Recursos Categóricos Ordinais.
Cat. Bin - Recursos Categóricos Binários
Min - Minutos.

4.2. Versão paralela do algoritmo de Seleção de Instâncias com Colônia de Formigas

Este trabalho apresenta uma melhoria do algoritmo desenvolvido em Hott et al. (2022) no Laboratório de Inteligência Computacional Aplicada (LICAP) da PUC Minas e modificado em Gonçalves e Nobre (2022), que utiliza o método de Otimização por Colônia de Formigas (ACO) implementado na linguagem *Python* versão 3.10, para seleção de instâncias. Especificamente, propõe-se uma implementação paralela para acelerar o processo de seleção de instâncias e melhorar a qualidade do modelo gerado.

O estudo realizado por Gonçalves e Nobre (2022) demonstrou que a seleção de instâncias com ACO é uma estratégia eficaz para reduzir o nível de ruído e assim por

sua vez reduzir o tamanho do conjunto de dados significativamente, melhorando a performance dos algoritmos de classificação *Random Forest*, SVM e Árvore de Decisão em conjuntos de dados grandes. Contudo, os tempos de execução do algoritmo originalmente desenvolvido por Hott et al. (2022) muitas vezes são inviáveis, demorando horas ou até mesmo dias para bases com tamanho superior a 1.573 instâncias. Com base nisso, foi desenvolvida uma nova versão do código com o objetivo de reduzir o tempo de processamento e manter ou até mesmo melhorar a qualidade do modelo gerado.

4.2.1. Descrição das modificações realizadas no algoritmo

Para paralelizar o processo de busca de soluções, foi adicionada a biblioteca *joblib*[1] e sua função *Parallel*[2], que permite várias tarefas sejam executadas simultaneamente em várias *threads*. As principais modificações realizadas foram:

1. Uso do módulo *Parallel*: Paralelizar o processo de busca de soluções. Isso permite que várias formigas encontrem soluções simultaneamente, acelerando o processo de otimização.
    1.1. O *loop while* existente na função *run_colony* é retirado da mesma e adicionando em uma função que é denominada *ant_search_solution*, essa alteração é necessária para possibilitar o uso do paralelismo.
    1.2. É criada uma lista de formigas contendo informações sobre cada uma delas, incluindo um índice, a própria formiga, a constante Q e a taxa de evaporação de feromônio;
    1.3. A função *ant_search_solution* é executada em paralelo em cada formiga da lista usando a biblioteca *joblib*, e o resultado é armazenado em uma lista;
    1.4. A lista de formigas na colônia é atualizada com os resultados encontrados por cada formiga.

Em resumo é criada uma lista de formigas com informações sobre cada uma delas, executa a busca de soluções de cada formiga em paralelo e atualiza a lista de formigas na colônia com os resultados encontrados.

O algoritmo original possui alguns parâmetros que devem ser ajustados pelo usuário, tais como a quantidade inicial de feromônio em cada caminho de busca (*initial_pheromone*), a quantidade de feromônio depositada por uma formiga (Q) e a taxa de evaporação de feromônio (*evaporation_rate*) ao final de cada iteração, que é necessário estar em um intervalo entre $[0, 1]$. Neste trabalho, os valores iniciais adotados foram *initial_pheromone* = 1, Q = 1 e *evaporation_rate* = 0.1, os quais foram mantidos pois outros valores testados não produziram alterações significativas nos resultados. É importante destacar que, neste trabalho, foram empregadas duas *threads* para sua execução, embora tenham sido testados outros valores, não se observaram alterações significativas no desempenho.

Para a execução do ACO[3] e do ACO paralelo, foi utilizado um computador com o Processador Intel(R) Core(TM) i7-4770 CPU *3.40GHz*, com 4 núcleos, 8 *threads*, e *32GB* de Memória RAM.

[1] A Documentação oficial pode ser encontrada em https://joblib.readthedocs.io/en/stable

[2] A Documentação oficial pode ser encontrada em https://joblib.readthedocs.io/en/stable/generated/joblib.Parallel.html

[3] É a implementação padrão sequencial

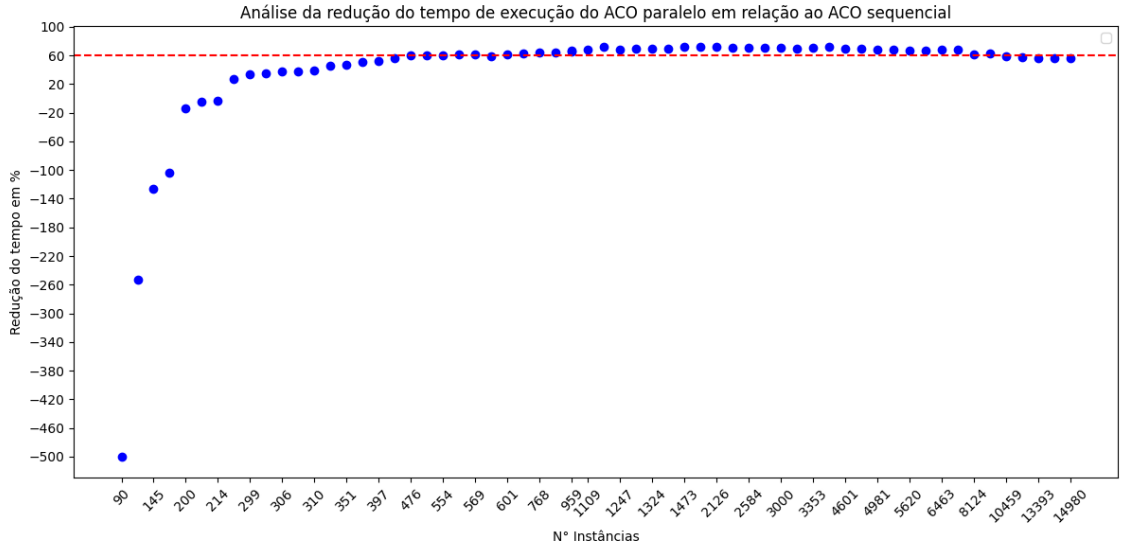
Por fim, disponibilizamos um repositório no GitHub[4] contendo o código desenvolvido neste estudo. E também o código original implementado em Hott et al. (2022) com a melhorias introduzidas por Gonçalves e Nobre (2022). Além disso, é possível encontrar orientações detalhadas para instalar e executar o código. No repositório, também está disponível os *scripts* de ML utilizados para avaliar a qualidade das bases de dados.

5. Resultados e Discussões

Após a execução dos algoritmos ACO e ACO paralelo (implementado neste estudo), observou-se uma redução percentual média de 52% no tamanho da base de dados para ambas as versões do ACO. Os resultados estão detalhados na Tabela 1, na seção "Análise dos tempos de execução". Nessa tabela, o tempo de execução para o ACO sequencial é representado como "Tempo Sequencial", enquanto o ACO paralelo é referenciado como "Tempo Paralelo". A última coluna exibe o ganho percentual no tempo de execução da versão paralela em relação à versão sequencial.

Outro ponto a ser observado na Tabela 1, são a quantidade e os tipos de atributos que não influenciam o tempo de execução do algoritmo. O fator determinante para a variação no tempo de execução é a quantidade de instâncias, quanto mais instâncias, maior o tempo necessário para a execução do algoritmo.

Figura 2. ACO paralelo vs ACO sequencial

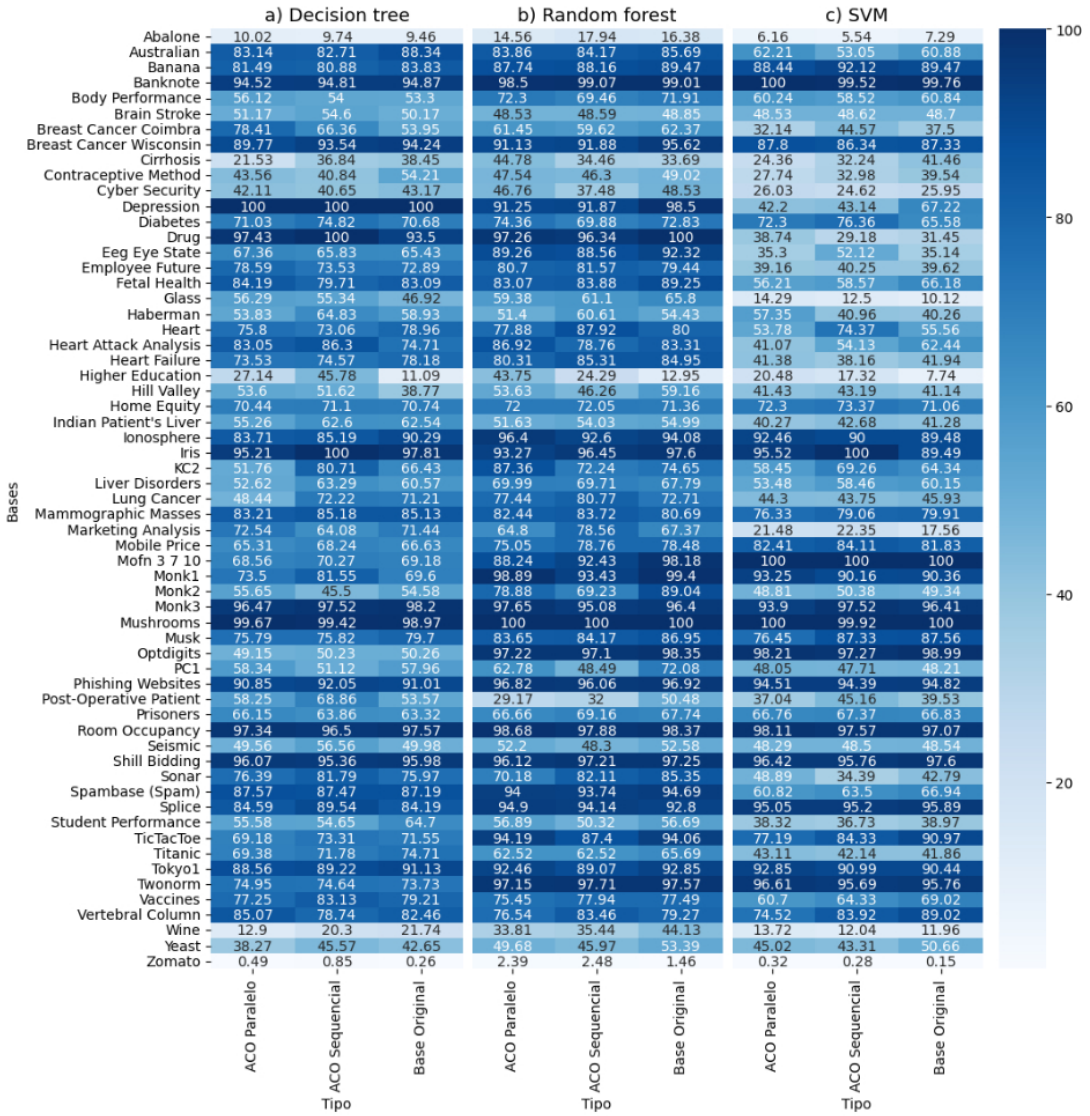


É importante observar que o uso do ACO paralelo pode não ser ideal para conjuntos de dados muito pequenos, com menos de 418 instâncias. Isso se deve ao *overhead* introduzido pela criação e coordenação de *threads*, que, em conjuntos de dados pequenos, pode consumir uma parcela significativa do tempo total de execução, tornando o processamento paralelo menos vantajoso. No entanto, para conjuntos de dados com mais de 418 instâncias, o uso do ACO paralelo mostrou-se promissor, com uma redução de cerca de 63% no tempo de execução. Para melhor compreensão desse contexto, A Figura 2 ilustra

[4]Os códigos desenvolvidos podem ser obtidos em https://github.com/eduardaoliveiradebrito/IC-TCC-colonia-de-formigas

um gráfico de dispersão contendo as 61 bases consideradas neste estudo. Para facilitar a legibilidade, apenas 30 bases são identificadas no eixo y, embora todas as 61 bases estejam representadas no gráfico. O eixo x representa o número de instâncias, enquanto o eixo y exibe o ganho percentual, permitindo uma visualização mais clara do panorama geral. Além disso, uma linha tracejada foi inserida no ponto de redução do tempo de 60%, evidenciando a melhoria conforme o número de instâncias.

Figura 3. Mapa de calor do F-Measure em algoritmos de ML



Além da análise comparativa do ganho de desempenho em termos de tempo de execução, conduziu-se uma análise comparativa utilizando os algoritmos de ML: *Random Forest*, Árvore de Decisão e SVM. A métrica avaliada foi a *F-Measure* (Combina as métricas de precisão (*precision*) e *recall* (revocação) em um único valor, fornecendo uma

medida geral do desempenho de um modelo), considerando três conjuntos de dados: a base original, a base reduzida com ACO e a base reduzida com o ACO paralelo.

Conforme ilustrado no mapa de calor da Figura 3, observa-se que a qualidade geral do modelo permaneceu semelhante em comparação com a base de dados original, tanto para o ACO quanto para o ACO paralelo. No entanto, existem exceções em bases de dados específicas, como *Cirrhosis*, *KC2* e *Student Performance*, onde a nova base gerada pelo ACO paralelo registrou uma ligeira queda de cerca de 11% na qualidade do modelo. Por outro lado, observamos melhorias de 15% em bases como *Breast Cancer Coimbra*, *Higher Education* e *Hill Valley*. Essas variações excepcionais levantam questionamentos sobre as razões por trás dessas perdas e ganhos.

Apesar das pequenas perdas percentuais em algumas bases, a maioria manteve sua qualidade, e algumas até apresentaram ganhos significativos. Considerando nosso objetivo principal de reduzir o tempo de execução e manter o desempenho dos algoritmos de ML, é justo afirmar que, em quase todas as bases de dados, o algoritmo cumpriu seu propósito.

6. Considerações Finais

O objetivo principal deste trabalho foi reduzir o tempo de execução do algoritmo de seleção de instâncias ACO, permitindo uma significativa redução do conjunto de dados, mantendo ou melhorando sua qualidade. A redução do conjunto de dados por meio da seleção criteriosa de instâncias apresenta benefícios notáveis, como a simplificação do modelo, resultando em tempos de treinamento mais rápidos. Além disso, ao focar em instâncias mais relevantes, é possível aprimorar a eficiência do algoritmo de Machine Learning, melhorando a precisão das previsões e diminuindo o risco de overfitting.

A técnica de paralelismo aplicada neste estudo mostrou-se satisfatória, proporcionando uma redução do tempo de execução com média de 63%, e mantendo uma redução do número de instâncias com média 52%. Além disso, a qualidade geral dos dados permaneceu estável.

Como limitações, destacamos a utilização de um computador com baixo poder de processamento, o que impossibilitou o uso do paralelismo de núcleos. Além disso, ajustes mais refinados no número de *threads* são necessários, uma vez que poucos testes foram realizados devido ao tempo restrito, onde o número de *threads* pode afetar significativamente o tempo de execução.

Quanto a trabalhos futuros, consideramos quatro áreas de foco: 1) Explorar o uso do paralelismo *multiprocessing*, que inicialmente se mostrou promissor em relação ao paralelismo de *multithreading* ; 2) Realizar uma investigação detalhada das variações possíveis no desempenho dos modelos de ML em relação às três variações de bases de dados utilizadas neste estudo; e 3) Ampliar o escopo da pesquisa, testando conjuntos de dados maiores, com mais de 10.000 instâncias, tanto balanceados quanto desbalanceados, para avaliar seu impacto na qualidade do modelo; 4) Investigar o impacto do número de *threads* para bases maiores que 10.000 instâncias. É importante ressaltar que, neste trabalho, foi realizada uma breve investigação na variação do número de *threads* que se mostrou promissora. No entanto, não foi possível obter dados suficientes para afirmar com segurança que esta abordagem seria eficaz.

Referências

A. A. Akinyelu, A. E. Ezugwu, e A. O. Adewumi. Ant colony optimization edge selection for support vector machine speed optimization. *Neural Computing and Applications*, 32(15):11385–11417, Aug 2020. ISSN 1433-3058. doi: 10.1007/s00521-019-04633-8. URL https://doi.org/10.1007/s00521-019-04633-8.

I. M. Anwar, K. M. Salama, e A. M. Abdelbar. Instance selection with ant colony optimization. *Procedia Computer Science*, 53:248–256, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.07.301. URL https://www.sciencedirect.com/science/article/pii/S1877050915018049. INNS Conference on Big Data 2015 Program San Francisco, CA, USA 8-10 August 2015.

A. Astudillo, J. Gillis, G. Pipeleers, W. Decré, e J. Swevers. Speed-up of nonlinear model predictive control for robot manipulators using task and data parallelism. In *2022 IEEE 17th International Conference on Advanced Motion Control (AMC)*, pages 201–206, 2022. doi: 10.1109/AMC51637.2022.9729271.

I. Czarnowski. Firefly algorithm for instance selection. *Procedia Computer Science*, 192:2269–2278, 2021. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2021.08.240. URL https://www.sciencedirect.com/science/article/pii/S1877050921017373. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 25th International Conference KES2021.

B. A. de Melo Menezes, N. Herrmann, H. Kuchen, e F. Buarque de Lima Neto. High-level parallel ant colony optimization with algorithmic skeletons. *International Journal of Parallel Programming*, 49(6):776–801, Dec 2021. ISSN 1573-7640. doi: 10.1007/s10766-021-00714-1. URL https://doi.org/10.1007/s10766-021-00714-1.

M. Dorigo, M. Birattari, e T. Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006. doi: 10.1109/MCI.2006.329691.

P. González, R. Prado-Rodriguez, A. Gábor, J. Saez-Rodriguez, J. R. Banga, e R. Doallo. Parallel ant colony optimization for the training of cell signaling networks. *Expert Systems with Applications*, 208:118199, 2022. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2022.118199. URL https://www.sciencedirect.com/science/article/pii/S0957417422013586.

A. C. M. Gonçalves e C. N. Nobre. Análise de desempenho do algoritmo colônia de formigas para seleção de instâncias. *Pontifícia Universidade Católica de Minas Gerais*, 01, 2022. Aguardando disponibilização na web.

H. R. Hott, C. R. Jandre, P. H. S. Xavier, A. Miloud-Aouidate, D. M. Miranda, M. A. Song, L. E. Zarate, e C. N. Nobre. Selection of representative instances using ant colony - a case study in a database of children and adolescents with attention-deficit/hyperactivity disorder. *Pontifícia Universidade Católica de Minas Gerais*, 01, 2022. Aguardando disponibilização na web.

B.-Q. Hu, R. Chen, D.-X. Zhang, G. Jiang, e C.-Y. Pang. Ant colony optimization vs genetic algorithm to calculate gene order of gene expression level of alzheimer's disease. In *2012 IEEE International Conference on Granular Computing*, pages 169–172, 2012. doi: 10.1109/GrC.2012.6468612.

M. Kordos, M. Blachnik, e R. Scherer. Fuzzy clustering decomposition of genetic algorithm-based instance selection for regression problems. *Information Sciences*, 587:23–40, 2022. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2021.12.

016. URL https://www.sciencedirect.com/science/article/pii/S0020025521012378.

F. Oliveira, M. Balbino, L. Zárate, F. Ngo, R. Govindu, A. Agarwal, e C. Nobre. Predicting inmates misconduct using the shap approach. *Artificial Intelligence and Law*, pages 1–27, 03 2023. doi: 10.1007/s10506-023-09352-z.

B. Pascariu, M. Samà, P. Pellegrini, A. D'Ariano, J. Rodriguez, e D. Pacciarelli. Effective train routing selection for real-time traffic management: Improved model and aco parallel computing. *Computers Operations Research*, 145:105859, 2022. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2022.105859. URL https://www.sciencedirect.com/science/article/pii/S0305054822001332.

J. Peake, M. Amos, P. Yiapanis, e H. Lloyd. Scaling techniques for parallel ant colony optimization on large problem instances. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 47–54, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361118. doi: 10.1145/3321707.3321832. URL https://doi.org/10.1145/3321707.3321832.

Z. Shojaee, S. A. Shahzadeh Fazeli, E. Abbasi, F. Adibnia, F. Masuli, e S. Rovetta. A mutual information based on ant colony optimization method to feature selection for categorical data clustering. *Iranian Journal of Science*, 47(1):175–186, Feb 2023. ISSN 2731-8109. doi: 10.1007/s40995-022-01395-2. URL https://doi.org/10.1007/s40995-022-01395-2.

R. Skinderowicz. Implementing a gpu-based parallel max–min ant system. *Future Generation Computer Systems*, 106:277–295, 2020. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2020.01.011. URL https://www.sciencedirect.com/science/article/pii/S0167739X19309550.

Y. Zhou, F. He, N. Hou, e Y. Qiu. Parallel ant colony optimization on multi-core simd cpus. *Future Generation Computer Systems*, 79:473–487, 2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2017.09.073. URL https://www.sciencedirect.com/science/article/pii/S0167739X16304289.