Análise de Viabilidade na Utilização de Algoritmos Meta-heurísticos como Mineradores em Redes Blockchain

Luiz Felipe Fonseca Rosa, Luiz Antonio Rodrigues, André Luiz Brun

[1]Universidade Estadual do Oeste do Paraná - UNIOESTE
Cascavel – PR – Brasil

{luiz.rosa8, luiz.rodrigues, andre.brun}@unioeste.br

Abstract. *Proof-of-Work (PoW) mining in blockchains consumes large amounts of energy on intensive hash computations without generating practical value. As an alternative, this study investigates the use of metaheuristics — Genetic Algorithm, Simulated Annealing, and Particle Swarm Optimization (PSO) — in a mining model that solves useful computational problems, such as the Traveling Salesman Problem, where valid hashes represent solutions. Results indicate that Simulated Annealing efficiently found good solutions. The Genetic Algorithm was functional but incurred high computational costs, while PSO showed inferior performance. Despite its potential, metaheuristic-based mining still faces challenges regarding its feasibility and practical application in blockchain systems.*

Resumo. *A mineração baseada em Proof-of-Work (PoW) em blockchains consome grandes quantidades de energia em cálculos hash intensivos, sem gerar valor prático. Como alternativa, esta pesquisa investiga o uso de metaheurísticas — Algoritmo Genético, Simulated Annealing e Particle Swarm Optimization (PSO) — em um modelo de mineração que resolve problemas computacionais úteis, como o do Caixeiro Viajante, utilizando soluções válidas como hashes. Os resultados indicam que o Simulated Annealing encontrou boas soluções de forma mais eficiente. O Algoritmo Genético foi funcional, mas com alto custo computacional, enquanto o PSO apresentou desempenho inferior. Apesar do potencial, a mineração via meta-heurísticas ainda enfrenta desafios quanto à sua viabilidade e aplicação prática em blockchains.*

1. Introdução

A tecnologia *blockchain* tem se consolidado como uma solução inovadora, segura e confiável para a realização de transações entre participantes em um modelo *peer-to-peer*, mesmo na ausência de confiança mútua entre os nós da rede [Belotti et al. 2019]. Trata-se de uma estrutura de dados descentralizada, organizada como uma lista encadeada de blocos, na qual cada bloco contém um conjunto de dados [Miah et al. 2019]. Esses dados variam conforme a aplicação e podem incluir transações financeiras, certificados de posse, registros de saúde, entre outros. As *blockchains* desempenham um papel crucial em diversos setores, como finanças, artes, saúde, Internet das Coisas (IoT), computação em nuvem, entre muitos outros [Greve et al. 2018].

Para garantir a confiabilidade das transações entre dois nós, as *blockchains* utilizam protocolos de validação de blocos. O mecanismo de consenso do Bitcoin, conhecido

como Prova de Trabalho (PoW - *Proof-of-Work*), baseia-se em um oráculo randômico que seleciona um líder responsável por coordenar o consenso em cada rodada. Esse líder é escolhido por meio de um desafio criptográfico altamente complexo, cuja resolução exige grande poder computacional. O processo é denominado mineração, e os participantes dispostos a competir para encontrar a solução são chamados de mineradores [Antonopoulos 2014, Greve et al. 2018].

Embora o PoW proporcione elevados níveis de segurança à rede, seu custo energético é significativo [Lashkari and Musilek 2021]. A crescente dificuldade dos desafios criptográficos torna inviável a resolução por um único nó, exigindo a atuação coordenada de diversos mineradores para manter a rede operante em tempo hábil [Asif and Hassan 2023].

Essa robustez, no entanto, tem um custo elevado: o desperdício energético. Em março de 2018, estimava-se que cerca de 28 quintilhões de hashes eram processados por segundo na rede Bitcoin, resultando em apenas 2 a 3 blocos válidos por intervalo de tempo — cerca de 200 mil transações por dia. Isso representa uma razão de aproximadamente 8,7 quintilhões de hashes por transação, mesmo nos melhores cenários [de Vries 2018]. No mesmo ano, o consumo energético da rede Bitcoin foi estimado em 2,55 gigawatts — valor comparável ao consumo de países como a Irlanda, com 3,1 gigawatts [de Vries 2018]. A principal crítica ao PoW reside justamente no fato de que esse esforço computacional não resulta em benefício prático adicional, sendo utilizado unicamente para encontrar um hash válido [Bizzaro et al. 2020, Shibata 2019].

Em contrapartida, outras blockchains, como Ethereum e Solana, adotam o protocolo *Proof-of-Stake* (PoS), ou Prova de Participação. Nesse modelo, a seleção do validador do bloco é baseada na quantidade de tokens que o participante possui, de modo que quanto maior esse valor, maior a probabilidade de o nó ser escolhido como oráculo [Asif and Hassan 2023]. O PoS substitui o gasto energético do PoW por um comprometimento financeiro, oferecendo como principal vantagem sua eficiência energética. Segundo a Ethereum Foundation, o PoW consumia cerca de 5,13 gigawatts, enquanto o PoS consome apenas 2,62 megawatts — uma redução de 99,9% no consumo de energia [Asif and Hassan 2023].

Nesse contexto, surgem propostas emergentes que visam conciliar a segurança do PoW com a eficiência energética do PoS, como a utilização de algoritmos meta-heurísticos como mecanismo de mineração. Protocolos como *Proof-of-Search* e *Proof-of-Evolution* propõem o uso de algoritmos de otimização para resolver problemas computacionais úteis durante o processo de mineração [Shibata 2019, Bizzaro et al. 2020]. Embora ainda exijam alto poder computacional, essas abordagens transformam o esforço em resultados com valor prático, mantendo altos níveis de segurança devido à complexidade dos problemas e ao uso de criptografia [Asif and Hassan 2023].

Algoritmos meta-heurísticos são estratégias de otimização projetadas para encontrar soluções ótimas — ou próximas da ótima — em problemas complexos e com espaços de busca vastos. Combinando elementos estocásticos e determinísticos, esses algoritmos exploram o espaço de soluções em busca de máximos ou mínimos globais, dependendo do problema [Blum and Roli 2003]. Dentre os principais exemplos, destacam-se o Algoritmo Genético (AG), *Simulated Annealing* (SA) e *Particle Swarm Optimization*

(PSO) [Gaspar-Cunha et al. 2012], que se inspiram em processos naturais como evolução biológica, comportamento de enxames e física térmica — características comuns à chamada computação evolutiva [Kennedy and Eberhart 1995].

Diante desse cenário, o objetivo deste trabalho é investigar e avaliar a viabilidade do uso de algoritmos meta-heurísticos — em especial AG, SA e PSO — como alternativas viáveis ao modelo tradicional de mineração em redes *blockchain*. A proposta busca não apenas mitigar o desperdício de recursos computacionais, como também transformar esse esforço em resultados úteis e aplicáveis.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta os principais trabalhos relacionados ao tema e discute suas abordagens. Na Seção 3, é detalhada a proposta de solução desenvolvida neste trabalho. A Seção 4 discute os experimentos realizados e os resultados obtidos por meio de simulações. Por fim, a Seção 5 apresenta as conclusões e aponta direções para trabalhos futuros.

2. Trabalhos Relacionados

O *Proof-of-Search* [Shibata 2019] combina a formação de consenso em Blockchain com a solução de Problemas de Otimização propondo um novo consenso, que permite que uma *blockchain* seja usada para resolver problemas de busca e otimização. Qualquer usuário pode submeter um trabalho para encontrar uma solução de um problema de otimização. Segundo o autor, "*Proof of work* nada mais é do que um repetitivo cálculo de *hashes*, o que acaba sendo um gasto de poder computacional e eletricidade, que poderiam ser aplicados para resolver problemas úteis".

Outros consensos tentam resolver ambos os problemas, como o *proof-of-useful-work* [Ball et al. 2017], o Grindcore [Halford 2014] e o Primecoin [King 2013], mantendo a segurança da rede e transformando o gasto computacional em algo mais útil. O primeiro, utiliza o poder computacional para resolver problemas de vetores ortogonais, enquanto o Primecoin utiliza o gasto energético na busca por cadeias de números primos. Porém, não está claro o quanto de demanda existem para tais problemas. O Grindcore resolve os problemas implementando uma prova de pesquisa (*proof-of-research*), recompensando mineradores que disponibilizem seus recursos para a pesquisa da *Berkeley Open Infrastructure for Network Computing* (BOINC). A desvantagem é que a *blockchain* está totalmente associada a uma entidade, o que pode impactar a continuidade da rede [Shibata 2019].

A principal contribuição do *proof-of-search* (PoSe) é utilizar o poder computacional do *proof-of-work* de forma mais significativa do que o Primecoin e *proof-of-useful-work*, não sendo dependente de nenhum terceiro como o Grindcore.

O *proof-of-search* permite que o poder computacional gasto no PoW seja usado para encontrar soluções ótimas (ou próximas da ótima) para instâncias de problemas de otimização. Neste protocolo, um usuário submete um problema de otimização juntamente com um programada chamada de avaliador, sendo o *nonce* a concatenação entre a solução candidata e seu valor resposta avaliado.

A Prova de Evolução (*Proof of Evolution* - PoE) [Bizzaro et al. 2020] também deixa claro a importância do consenso de prova de trabalho, sobretudo no quesito de segurança, afinal, uma quantia muito grande de poder computacional deve ser investida

para realizar a solução do *puzzle*. Apesar disso, outros modelos de consenso propostos não possuem as mesmas características do PoW, tais como a dificuldade inerente do problema, a facilidade de verificação pública da solução, a homogeneidade na complexidade dos desafios, a capacidade de ajuste dinâmico da dificuldade, a sensibilidade ao bloco e a impossibilidade de reutilização, além da independência da distribuição dos cálculos.

O *Proof of Evolution* é baseado no *Proof of Search*, no qual a principal contribuição do consenso é manter todas as propriedades-base do PoW enquanto usa parte da energia e do poder computacional para resolver algoritmos genéticos, permitindo cooperação entre os mineradores para melhorar a qualidade das soluções dos AGs. A estrutura de ambos os protocolos é semelhante, sendo a do PoE definida da seguinte forma: um *nonce* é uma tripla de valores (solução, *fitness* e complexidade). Todos os trabalhos realizados devem ser um AG com uma API (application program interface) fixa. Mineradores podem submeter as soluções antes de finalizar a execução do *job*, permitindo cooperação.

Diferente dos trabalhos apresentados nesta seção, a solução proposta explora múltiplas meta-heurísticas (AGs, SA, PSO), ampliando o escopo de aplicação. Cada algoritmo tem perfis distintos (exploração vs. explotação), permitindo selecionar a técnica mais adequada ao tipo de problema (por exemplo, SA para espaços de busca contínuos, PSO para otimização colaborativa). Além disso, ao contrário do Grindcore e do PoSe (que exige submissão de problemas por usuários), a solução proposta pode operar de forma autônoma, sem depender de terceiros ou infraestrutura externa.

3. A Solução Proposta

Neste trabalho, são implementados três algoritmos meta-heurísticos: Algoritmo Genético (AG), *Simulated Annealing* (SA) e *Particle Swarm Optimization* (PSO). Esses métodos foram escolhidos por sua comprovada eficácia em buscas heurísticas por soluções ótimas ou quase ótimas, especialmente em problemas de alta complexidade e com espaços de busca extensos. A proposta é empregar tais algoritmos para resolver problemas computacionais relevantes e aplicáveis na indústria ou em cenários reais, utilizando para isso a energia computacional do processo de mineração em redes blockchain.

No contexto desta pesquisa, o problema selecionado foi o clássico Problema do Caixeiro Viajante (PCV), que consiste em encontrar o caminho de menor custo (ou menor distância) para visitar um conjunto de cidades uma única vez e retornar ao ponto de partida. Este problema é notoriamente difícil do ponto de vista computacional, sendo amplamente utilizado como referência para avaliação de algoritmos de otimização.

A abordagem proposta consiste em minerar blocos utilizando, como nonce, indivíduos gerados pelos algoritmos evolutivos. Cada indivíduo representa uma possível solução para o problema em questão — no caso, uma rota candidata no PCV — sendo validado conforme critérios definidos de dificuldade e estrutura do bloco. O processo detalhado dessa aplicação encontra-se descrito no pseudocódigo apresentado no Algoritmo 1.

Inicialmente, o algoritmo - AG, PSO ou SA - é inicializado. Esta etapa cria a população inicial, um conjunto de indivíduos candidatos a *nonce*, representados como sequências de cidades do Problema do Caixeiro Viajante. Enquanto o algoritmo não

Algorithm 1 Mineração do Bloco usando Algoritmos Genéticos

```
1:  procedure MINERARBLOCO( )
2:      INICIALIZAALGORITMOEVOLUTIVO( )
3:      while algoritmoNaoConvergiu do
4:          if TESTARHASH(melhorIndividuo, i) < DificuldadeDaRede then
5:              RETURN(individuo)          ▷ Retorna o indivíduo válido imediatamente
6:          EVOLUIRALGORITMO( )
7:      i ← 0
8:      while true do
9:          if TESTARHASH(melhorIndividuo, i) < DificuldadeDaRede then
10:             nonce ← melhorIndividuo + i                ▷ Combina explicitamente
11:             RETURN(nonce)                              ▷ Retorna o nonce válido
12:         i ← i + 1
```

convergir, cada indivíduo da população atual é testado como *nonce* para o bloco em mineração. Isso envolve verificar que o *hash* gerado ao combinar o *nonce* com os dados do bloco é menor que a dificuldade estabelecida pela rede. Se nenhum indivíduo for válido, o algoritmo continua a evoluir: novos indivíduos são gerados, passando por possíveis mutações, e o teste se repete. Caso haja algum elemento que retorne um *hash* válido para a rede, este elemento é então retornado e o algoritmo para, minerando assim o bloco.

Quando o algoritmo evolutivo converge - atingindo um critério como número máximo de iterações ou uma solução aceitável - sem ter um bloco minerado, inicia-se uma busca por força bruta, similar ao processo de mineração do *Bitcoin*. Toma-se o melhor indivíduo encontrado, inicializa-se uma variável i com valor zero e incrementa-se i iterativamente, concatenando cada valor ao melhor indivíduo e testando o *hash* resultante. O processo termina quando um *hash* válido, menor que a dificuldade da rede, é encontrado.

O *nonce* no protocolo proposto se trata de uma tupla de valores: solução encontrada e qualidade da solução. Esses dois elementos são essenciais para verificar se o nó realmente encontrou uma solução válida e o quão boa ela é, sendo simples de ser validada e verificada por outros nós.

Em todas as implementações, as soluções válidas são submetidas ao processo de verificação do *hash* que, no contexto deste trabalho, serão todos os indivíduos de todas as populações do AG, todas as soluções testadas pelo SA e todos os elementos componentes do PSO.

A definição das dificuldades foi baseada em um conceito fundamental das redes *blockchain*: o *target*. Esse valor representa o limite superior que um *hash* de bloco deve atingir para ser considerado válido. O *target* é calculado utilizando a Equação 1.

$$\textit{target} = 1 \ll (256 - \textit{Dificuldade}) \tag{1}$$

Essa relação indica que, à medida que a dificuldade aumenta, o *target* diminui, reduzindo a quantidade de *hashes* válidos e, consequentemente, aumentando o esforço

computacional necessário para encontrar uma solução válida. Esse mecanismo é essencial para o controle da taxa de geração de blocos em sistemas baseados em *Proof-of-Work*. Nos testes a seguir,

4. Resultados

Foram definidas três dificuldades para a realização dos testes: a) fácil, definida como sendo 12; b) intermediária, tendo o valor 15 e c) difícil, tendo o valor de 18. Além disso, foram utilizados três arquivos de entrada contendo diferentes problemas de otimização, resultando em nove cenários distintos. Cada um destes cenários foi testado utilizando tos rês algoritmos, totalizando 27 execuções experimentais.

Os três arquivos de entrada utilizados no problema foram escolhidos por possuírem tamanhos variados, permitindo a avaliação do desempenho dos algoritmos em diferentes escalas do problema de otimização e estão disponíveis no repositório *TSPLIB95* [1], um conjunto de instâncias clássicas para o Problema do Caixeiro Viajante. Os arquivos selecionados foram: a) berlin52.csv, que representa 52 cidades; b) fnl4461.csv, com 4.461 cidades; e d) d15112.csv, com 15.112 cidades.

Os três algoritmos de otimização foram executados 11 vezes para cada conjunto de dados, correspondente aos diferentes tamanhos de instâncias do problema (52, 4461 e 15.112 cidades) e para cada nível de dificuldade (fácil, intermediária e difícil). Para garantir a precisão das análises, foi considerada a média dos 10 últimos resultados de cada execução, minimizando o impacto de eventuais flutuações nos valores obtidos nas primeiras iterações. Esse procedimento visa fornecer uma avaliação mais confiável do desempenho dos algoritmos em diferentes cenários.

Os resultados são apresentados comparando a frequência das soluções encontradas pelo algoritmo genético ou por força-bruta, o tempo médio de execução, a quantidade média de *hashes* testada e a distância média obtida.

4.1. Frequência de Soluções Encontradas

A primeira análise realizada refere-se ao cômputo do número de vezes que o *hash* foi obtido pelo método de otimização (*Proof of Search* - PoSe) e quantas vezes foi determinado pela busca por força bruta (*Proof of Work* - PoW).

A Tabela 1 apresenta a frequência com que as soluções foram encontradas através do PoSe proposto e PoW para cada um dos cenários. Os valores identificados pelo PoSe são aqueles cujo *hash* do bloco foi encontrado antes do algoritmo de otimização terminar (ou convergir). Por outro lado, os registros marcados como PoW correspondem àqueles em que o algoritmo convergiu antes de obter uma *hash* válido, iniciando, a partir desse ponto, uma busca por força bruta. Cada célula apresenta a distribuição PoSe/PoW. Observa-se que, para o cenário fácil, os três algoritmos de otimização encontram soluções em sua totalidade, exceto no caso do *Simulated Annealing* para o problema de 52 cidades, onde o número de soluções encontradas através do PoSe é ligeiramente inferior.

Já no cenário intermediário, o AG e o PSO têm um desempenho robusto, com a maioria das soluções encontradas sendo obtidas antes da convergência do algoritmo, exceto no caso de 52 cidades para o AG, onde houve uma divisão entre PoSe (5 casos) e

[1] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/`

PoW (5 casos). O *Simulated Annealing*, por outro lado, apresentou um comportamento mais instável, encontrando mais soluções através do PoW (9 de 10 execuções) em maior número de cidades.

Para o cenário difícil, o PSO teve um desempenho superior aos demais, onde encontrou a totalidade dos *hashes* dentro da execução do algoritmo de otimização para todas as rotas. O *Simulated Annealing* teve um desempenho inferior em problemas de 52 (onde não pode encontrar nenhuma hash antes de convergir) e 4461 cidades (onde o PoW foi necessário em 4 execuções), encontrando todos os *hashes* na rota de 15112 cidades. O Algoritmo Genético apresentou um desempenho mais equilibrado em termos de soluções encontradas. Para o problema de 52 cidades, ele foi capaz de encontrar o hash em apenas uma execução. Para as outras duas instâncias do problema ele pôde alcançar um hash válido em 90% das execuções para o 4461 cidades e nas dez execuções para o problema composto de 15112 cidades.

Tabela 1. Frequência das soluções resolvidas por cenário (PoSe/PoW).

Entrada	Dificuldade	SA		AG		PSO	
		PoSe	PoW	PoSe	PoW	PoSe	PoW
52 cidades	Fácil	7	**3**	10	0	10	0
	Intermediária	1	**9**	5	**5**	10	0
	Difícil	0	**10**	1	**9**	10	0
4.461 cidades	Fácil	10	0	10	0	10	0
	Intermediária	10	0	10	0	10	0
	Difícil	7	3	9	1	10	0
15.112 cidades	Fácil	10	0	10	0	10	0
	Intermediária	10	0	10	0	10	0
	Difícil	10	0	10	0	10	0

4.2. Tempo Médio de Execução

O segundo critério de análise foi o tempo gasto para a execução dos métodos de otimização e a solução por força bruta. A Tabela 2 mostra o tempo médio (em segundos) de execução para cada algoritmo de otimização em função da dificuldade do problema e do tamanho do conjunto de cidades. Observa-se que, para o cenário fácil, o tempo médio de execução é relativamente baixo, com o *Simulated Annealing* apresentando o melhor desempenho em todos os tamanhos de problema. O PSO, por sua vez, apresentou resultados próximos ao SA. Já o Algoritmo Genético demandou mais tempo para ser executado em comparações às outras técnicas.

No cenário intermediário, o Algoritmo Genético foi a estratégia mais demorada para ser executada, independente do número de cidades presentes no problema. Ao utilizarmos 52 e 15.112 cidades, o método de SA foi o mais eficiente em termos de tempo. Já no caso de 4.461 cidades, quem executou mais rapidamente foi o PSO (30,517 segundos), frente ao SA, que levou 36,635 segundos em média para ser executado.

Para o cenário difícil, o Algoritmo Genético e o PSO apresentaram tempos de execução substancialmente mais longos, principalmente em instâncias maiores, como no caso de 15112 cidades, quando o AG demandou, em média, 30,950 minutos para ser

executado. Nesse contexto, o *Simulated Annealing* foi o algoritmo que mostrou maior eficiência, com um tempo médio de execução consideravelmente mais baixo.

Tabela 2. Tempo médio de execução (em segundos).

Entrada	Dificuldade	SA	AG	PSO
52 cidades	Fácil	0,049	0,074	0,071
	Intermediária	0,356	29,750	0,458
	Difícil	5,880	44,304	11,494
4.461 cidades	Fácil	2,607	12,351	4,491
	Intermediária	36,635	46,376	30,517
	Difícil	160,647	613,533	508,101
15.112 cidades	Fácil	10,286	33,892	13,632
	Intermediária	98,357	271,676	166,283
	Difícil	750,705	1.856,988	1.363,116

4.3. Quantidade Média de *Hashes* Testados

O terceiro critério levantado refere-se à quantidade média de *hashes* testados por cada abordagem até a obtenção de um *hash* válido. A Tabela 3 apresenta a quantidade média de indivíduos testados pelos algoritmos ao longo das dez execuções. Para o cenário fácil, os três algoritmos testaram um número relativamente baixo de soluções possíveis, com o Algoritmo Genético tendo a maior média, seguido do PSO e do *Simulated Annealing*, respectivamente.

No cenário intermediário, como esperado, o número de *hashes* testados aumentou significativamente, com o Algoritmo Genético apresentando a maior média, particularmente no problema de 52 cidades, onde foram testados 800.436 possíveis soluções. Este comportamento é decorrente do fato de o AG testar diversas versas a mesma rota, fato que ocorre quanto a população é muito homogênea. O PSO e o *Simulated Annealing* também aumentaram o número de testes, mas em menor escala, com o PSO mantendo um número de testes relativamente equilibrado.

Para o cenário difícil, todos os algoritmos apresentaram uma quantidade considerável de soluções testadas, com o Algoritmo Genético novamente avaliando a maior quantidade de *hashes*. O *Simulated Annealing* e o PSO, por outro lado, apresentaram números mais baixos em comparação ao AG, mas ainda assim significativos, indicando a necessidade de explorar um número maior de soluções em problemas mais difíceis. Novamente o AG foi executado até atingir o critério de parada e entrar no processo de PoW visto que a população mostrou-se muito homogênea e não foi capaz de obter maior volume de *hashes* distintos.

4.4. Distância Média Obtida

A última métrica levantada refere-se à qualidade da solução do TSP obtida pelos modelos de otimização. Neste cenário, quanto menor o valor encontrado, melhor o desempenho do algoritmo.

A Tabela 4 mostra a distância média obtida pelos três algoritmos para o problema do TSP. Como esperado, ao aumentarmos o número de cidades, o custo do trajeto formado

Tabela 3. Quantidade média de *hashes* testados.

Entrada	Dificuldade	SA	AG	PSO
52 cidades	Fácil	2.444	5.410	4.514
	Intermediária	3.593	800.436	23.602
	Difícil	3.855	1.364.673	273.451
4.461 cidades	Fácil	2.695	6.707	2.678
	Intermediária	23.897	21.414	12.133
	Difícil	127.088	378.517	288.183
15.112 cidades	Fácil	4.302	5.306	3.007
	Intermediária	39.631	52.752	43.353
	Difícil	255.902	399.141	332.472

também aumentou. No cenário fácil, o *Simulated Annealing* e o AG apresentaram comportamento similares, mas com o AG sendo superior. Já o PSO obteve soluções inferiores para as três instâncias do problema.

No cenário intermediário, o Algoritmo Genético obteve a menor distância média em comparação com o PSO e o *Simulated Annealing*, que apresentaram distâncias médias ligeiramente mais altas. Para o cenário difícil, o desempenho foi mais equilibrado entre os três algoritmos, com o PSO mostrando um desempenho ligeiramente superior ao *Simulated Annealing* e ao Algoritmo Genético em termos de distâncias médias.

Já no cenário difícil, para 52 cidades o AG se mostrou muito superior aos outros dois métodos, obtendo a menor distância. Para 4.461 cidades, o SA foi superior aos outros dois, com o PSO obtendo as maiores distâncias. O mesmo ocorreu para 15.112 cidades, onde o SA obteve o melhor resultado.

Tabela 4. Distância média obtida para o Problema do Caixeiro Viajante.

Entrada	Dificuldade	SA	AG	PSO
52 cidades	Fácil	$1,444 \times 10^4$	$2,029 \times 10^4$	$3,003 \times 10^4$
	Intermediária	$1,105 \times 10^4$	$1,041 \times 10^4$	$3,017 \times 10^4$
	Difícil	$1,084 \times 10^4$	$0,944 \times 10^4$	$2,950 \times 10^4$
4.461 cidades	Fácil	$8,081 \times 10^6$	$7,925 \times 10^6$	$8,334 \times 10^6$
	Intermediária	$6,577 \times 10^6$	$7,516 \times 10^6$	$8,331 \times 10^6$
	Difícil	$3,747 \times 10^6$	$5,248 \times 10^6$	$8,315 \times 10^6$
15.112 cidades	Fácil	$1,330 \times 10^8$	$1,310 \times 10^8$	$1,340 \times 10^8$
	Intermediária	$1,271 \times 10^8$	$1,234 \times 10^8$	$1,341 \times 10^8$
	Difícil	$1,000 \times 10^8$	$1,053 \times 10^8$	$1,341 \times 10^8$

4.5. Comparação dos modelos com o método de força bruta

A Tabela 5 demonstra os valores médios de tempo (em milissegundos) e número de *hashes* testados das 10 execuções para cada uma das dificuldades de execução do modelo de força bruta. Observa-se que o tempo de execução é ordens de grandeza inferior ao dos modelos que utilizam algoritmos meta-heurísticos no processo de mineração, sendo algo esperado, afinal, o algoritmo de mineração por força bruta é especializado em achar os *hashes* e não

possui outros cálculos matemáticos envolvidos (tais como gerar indivíduos, comparar indivíduos, gerar números aleatórios, mutações, seleções e etc.).

Outro resultado que podemos extrair da mesma tabela é que, independentemente do modelo utilizado, o número de *hashes* calculados para as diferentes dificuldades foi bastante similar ao do modelo de força bruta. Observou-se uma média de aproximadamente 3.000 *hashes* para a dificuldade fácil, 50.000 para a dificuldade média e 200.000 para a dificuldade difícil. Esses valores indicam uma consistência no volume de cálculos necessários, mesmo com a variação do modelo aplicado.

No entanto, o Algoritmo Genético (AG) apresentou um comportamento distinto em relação aos demais modelos, realizando significativamente mais cálculos de *hash*. Isso ocorre porque o AG avalia, em cada execução, um conjunto fixo de 1.500.000 indivíduos antes de recorrer ao modelo de força bruta. Como o AG converge rapidamente para uma solução ótima e opera com uma taxa de mutação de apenas 5%, sua capacidade de explorar novas possibilidades de *hash* é limitada. Com isso, ele tende a permanecer em um subconjunto reduzido de soluções, muitas vezes recalculando os mesmos *hashes*, o que gera uma quantidade expressiva de operações repetidas e desnecessárias no processo de mineração.

Apesar do PSO também testar 1.500.000 (1 milhão e 500 mil) elementos, o modelo explora uma área muito mais abrangente do que o AG, facilitando assim para achar um *hash* correto, tendo em vista que não repete tantos elementos quanto o AG. Assim, o PSO consegue ficar mais próximo do PoW nesse cenário de cálculos de *hashes*. Esta mesma propriedade foi a possível causadora do PSO ter tido menos qualidade no *fitness* das soluções.

Uma das possíveis soluções para resolver esse problema do AG seria aumentar a taxa de mutação, de forma a explorar mais o conjunto e testar mais *hashes*. Porém, isso pode ter um efeito colateral e acabar diminuindo o *fitness* médio da solução.

Tabela 5. Quantidade média de *hashes* testados e tempo médio de execução para a abordagem de força bruta.

Dificuldade	Tempo de Execução (s)	Número de *hashes* Testados
Fácil	0,0051	2.597
Intermediária	0,1098	73.947
Difícil	0,4068	261.303

4.6. Análise Estatística dos Modelos Meta-Heurísticos

Os testes estatísticos foram conduzidos para avaliar o desempenho dos algoritmos *Simulated Annealing* (SA), *Particle Swarm Optimization* (PSO) e *Algoritmo Genético* (AG) em relação ao tempo de execução e ao número de *hashes* calculados.

Inicialmente, foi aplicada uma análise de normalidade aos dados utilizando o teste de Shapiro-Wilk, cujos resultados indicaram que ambas as métricas não seguem uma distribuição normal ($p = 0.00000$). Assim, a comparação entre os algoritmos foi realizada por meio do teste de Kruskal-Wallis, com um nível de significância de 5%, sendo esta uma alternativa não paramétrica ao ANOVA. Os resultados indicam que não há diferenças estatisticamente significativas entre os algoritmos para ambas as métricas analisadas ($p =$

0.06542 para tempo e $p = 0.63196$ para número de *hashes*). Isso sugere que nenhum dos modelos meta-heurísticos apresentou uma vantagem estatística relevante sobre os demais no contexto avaliado.

Por outro lado, ao analisarmos a influência da dificuldade da instância no desempenho dos algoritmos por meio de uma regressão linear, observamos que a dificuldade tem um impacto significativo tanto no tempo de execução ($p = 7.02 \times 10^{-6}$, $R^2 = 0.151$) quanto no número de *hashes* calculados ($p = 2.03 \times 10^{-8}$, $R^2 = 0.225$). Em média, cada aumento no nível de dificuldade acrescenta aproximadamente 349 segundos ao tempo de execução e 209.900 novos *hashes* calculados.

Esse alto crescimento no número de *hashes* está diretamente relacionado ao comportamento do AG, que testa um grande número de candidatos antes de convergir para uma solução final. O AG, ao avaliar cerca de 1.500.000 elementos a cada execução, explorando um espaço de busca significativamente menor que seus concorrentes (por possuir baixa taxa de mutação), resulta em um número muito superior de cálculos de *hashes* em comparação aos demais algoritmos. Essa característica pode explicar a influência desproporcional do AG sobre os resultados globais, especialmente nas instâncias mais difíceis.

Dessa forma, os resultados indicam que, enquanto a dificuldade tem um impacto significativo nas métricas analisadas, a escolha do algoritmo, por si só, não apresenta uma diferença estatisticamente relevante em termos de desempenho. No entanto, o comportamento distinto do AG sugere que ajustes na parametrização, como um aumento na taxa de mutação, poderiam otimizar a exploração do espaço de busca, reduzindo a redundância de cálculos e melhorando sua eficiência no contexto da mineração.

4.7. Discussão dos Resultados

Os resultados obtidos evidenciam as diferenças significativas entre os algoritmos de otimização empregados para a mineração de blocos e sua viabilidade prática. Uma das principais observações foi o elevado tempo de execução quando comparado com métodos de força bruta simples. Isso era esperado, tendo em vista que não serão apenas cálculos de *hashes* e somas simples, mas sim vários outros passos computacionalmente custosos: geração de indivíduos, mutações, verificações, cálculos de *fitness*, etc., para centenas de milhares de indivíduos. Este elevado tempo resultante do alto poder computacional gasto pode ser um dos fatores que impossibilitaria seu uso em redes reais, devido à falta de incentivos aos mineradores e aos usuários.

Outro ponto relevante é o desempenho do algoritmo Particle Swarm Optimization (PSO), que apresentou resultados insatisfatórios em termos de *fitness*. Enquanto outros algoritmos conseguiam encontrar soluções mais próximas do ideal, o PSO consistentemente obteve valores menos otimizados, indicando uma possível dificuldade na adaptação à natureza do problema do caixeiro viajante. Isso pode estar relacionado à necessidade de uma representação mais refinada das partículas ou a uma maior exigência de exploração em espaços de busca de alta complexidade.

Esse baixo desempenho pode ser explicado também pelo fato de que, durante o processo de mineração, um *hash* válido é frequentemente encontrado antes que o algoritmo consiga convergir. Além disso, a performance inferior do PSO pode estar associada a uma baixa variação ou taxa de mutação, o que faz com que o algoritmo caia repetidamente em máximos locais. Uma possível solução para esse problema seria uma melhor

calibração de seus parâmetros, utilizando, por exemplo, técnicas como a otimização bayesiana para ajustar dinamicamente os valores de configuração do algoritmo.

Por outro lado, notou-se que o SA obteve um bom custo-benefício para a tarefa proposta, em comparação com o PSO e com o AG, garantindo excelentes soluções para o problema do TSP enquanto encontra o *hash* com menos cálculos e recálculos do que seus concorrentes. Isso se deve principalmente a simplicidade da natureza do algoritmo e de sua boa parametrização.

Além disso, a utilização de algoritmos de otimização para mineração se mostrou pouco vantajosa em termos de custo computacional. O gasto energético e computacional para rodar tais algoritmos é significativamente maior em comparação com os métodos tradicionais de Proof-of-Work (PoW), tornando inviável a sua adoção para redes blockchain convencionais. A otimização proposta, ainda que teoricamente interessante, introduz um overhead computacional que desestimula sua adoção em cenários práticos.

Ademais, do ponto de vista da segurança, a abordagem ainda pode ter um impacto positivo em sua utilização. O uso de técnicas de otimização complexas pode aumentar a segurança da rede ou, pelo menos, mantê-la no mesmo nível das abordagens tradicionais, uma vez que a solução do problema de mineração continua exigindo um alto poder computacional e um custo significativo para um ataque bem-sucedido. Contudo, uma análise mais aprofundada sobre possíveis vulnerabilidades e impactos a longo prazo seria necessária para validar essa hipótese.

5. Conclusões

O presente trabalho apresentou uma abordagem para buscar reutilizar a mesma rede e o mesmo poder computacional para resolver dois problemas concomitantes: a validação de blocos e o desperdício energético em blockchains. Para isso, tenta-se fazer convergir um algoritmo evolutivo e, cada elemento deste algoritmo é testado como *hash* válido de um bloco a ser minerado em uma rede *blockchain*. Caso o algoritmo convirja antes de ser encontrado um *hash* válido, segue-se via força bruta. Caso o *hash* seja encontrado antes de convergir, retorna-se o melhor elemento encontrado até aquele ponto.

Embora os algoritmos meta-heurísticos possam apresentar um consumo computacional elevado para a mineração de blocos, eles demonstram potencial na busca por soluções eficientes em redes *blockchain*. O desafio reside em equilibrar segurança e desempenho, tornando essas abordagens viáveis para aplicações práticas. Com isso, abre-se caminho para novas pesquisas e aprimoramentos que possam viabilizar a adoção desses algoritmos em cenários reais.

Os experimentos realizados demonstraram que o uso de algoritmos de otimização na mineração de blocos apresenta desafios consideráveis, especialmente em termos de viabilidade computacional. Embora algumas abordagens tenham mostrado capacidade de encontrar soluções, o alto custo computacional associado as torna pouco atrativas para uso prático.

Apesar dessas limitações, a pesquisa sugere que o uso de algoritmos meta-heurísticos pode oferecer resultados relevantes para aprimorar mecanismos de mineração, principalmente em questões relacionadas à segurança e eficiência do processo. No entanto, melhorias são necessárias para que essas técnicas se tornem competitivas com abor-

dagens tradicionais.

Como trabalhos futuros, sugere-se a implementação de um protocolo mais completo e robusto, utilizando um ou vários algoritmos de otimização para realizar buscas heurísticas. Isso possibilitaria um estudo aprofundado sobre a viabilidade de transferências de dados em redes distribuídas, além de permitir a paralelização da busca heurística em diferentes máquinas, como em um ambiente de computação em nuvem, visando reduzir a sobrecarga de execução.

Este protocolo pode incluir uma API (*Application Programming Interface*) de desenvolvimento que permita ao desenvolvedor do algoritmo de mineração submeter o problema à rede utilizando funções pré-definidas compatíveis com a *blockchain*. Por exemplo, o desenvolvedor pode fornecer os dados, uma função de verificação, uma função de mutação e uma função para gerar novos indivíduos, todas adaptadas ao problema específico em questão, independentemente de sua natureza. Isso facilitaria a implementação de problemas e soluções genéricas, indo muito além dos três algoritmos vistos neste trabalho e sendo capaz de resolver qualquer tipo de problema, não apenas o TSP.

Referências

Antonopoulos, A. M. (2014). *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st edition.

Asif, R. and Hassan, S. (2023). Shaping the future of ethereum: exploring energy consumption in proof-of-work and proof-of-stake consensus. *Frontiers in Blockchain*, 6.

Ball, M., Rosen, A., Sabin, M., and Vasudevan, P. N. (2017). Proofs of useful work. *IACR Cryptology ePrint Archive*, 2017:203. Accessed: Jun. 29, 2017.

Belotti, M., Božić, N., Pujolle, G., and Secci, S. (2019). A vademecum on blockchain technologies: When, which, and how. *IEEE Communications Surveys & Tutorials*, 21(4):3796–3838.

Bizzaro, F., Conti, M., and Pini, M. S. (2020). Proof of evolution: leveraging blockchain mining for a cooperative execution of genetic algorithms. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 450–455.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.

de Vries, A. (2018). Bitcoin's growing energy problem. 2:801–805.

Gaspar-Cunha, A., Takahashi, R., and Antunes, C. (2012). *Manual de computação evolutiva e metaheurística*. Ensino. Imprensa da Universidade de Coimbra / Coimbra University Press.

Greve, F., Sampaio, L., Abijaude, J., Coutinho, A. A., Brito, I., and Queiroz, S. (2018). *Blockchain e a Revolução do Consenso sob Demanda*, page v. 30.

Halford, R. (2014). Gridcoin: Crypto-currency using berkeley open infrastructure network computing grid as a proof of work. Online.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948.

King, S. (2013). Primecoin: Cryptocurrency with prime number proof-of-work. Online.

Lashkari, B. and Musilek, P. (2021). A comprehensive review of blockchain consensus mechanisms. *IEEE Access*, 9:43620–43652.

Miah, M. S. U., Rahman, M., Hossain, M. S., and Rupai, A. (2019). *Introduction to Blockchain*.

Shibata, N. (2019). Proof-of-search: Combining blockchain consensus formation with solving optimization problems. *IEEE Access*, 7:172994–173006.