PONTÍFICIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Graduação em Engenharia de Software

Emmanuel Viglioni, Lucas Machado de Oliveira Andrade, Marcelo Aguilar Araújo D'Almeida, Paulo Victor Pimenta Rubinger

RELATÓRIO TRABALHO PRÁTICO

Belo Horizonte
2024

Emmanuel Viglioni, Lucas Machado de Oliveira Andrade, Marcelo Aguilar Araújo D'Almeida, Paulo Victor Pimenta Rubinger

RELATÓRIO TRABALHO PRÁTICO

Relatório apresentado para disciplina de Fundamentos de Projeto e Análise de Algoritmos, para conclusão do Trabalho Prático

Professor(a): João Caram Santos de Oliveira.

Belo Horizonte

2024

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização do Problema e Objetivos do Trabalho

No contexto da disciplina de Fundamentos de Projeto e Análise de Algoritmos, o presente trabalho prático tem como objetivo a implementação e comparação de diferentes técnicas de resolução para um problema de otimização intratável, pertencente à classe NP. O problema em questão envolve a maximização do valor obtido na venda de energia por uma empresa produtora, através de um leilão em que diversas empresas interessadas fazem ofertas por lotes específicos de energia.

A partir deste problema, nosso grupo se propôs a desenvolver soluções utilizando quatro abordagens distintas: backtracking, algoritmos gulosos, divisão e conquista, e programação dinâmica. Cada abordagem foi implementada individualmente por um membro do grupo, permitindo uma avaliação detalhada e comparativa tanto em termos de eficiência temporal quanto na qualidade dos resultados obtidos.

Este relatório técnico está organizado da seguinte forma: inicialmente, apresentamos uma descrição detalhada do problema. Em seguida, cada abordagem de implementação é discutida em separado, com explicações sobre as decisões de projeto, estratégias adotadas e resultados dos testes realizados. Posteriormente, fizemos uma comparação entre os resultados das diferentes abordagens, analisando tanto o desempenho quanto a qualidade das soluções encontradas. Por fim, são apresentadas as considerações finais sobre o trabalho realizado, destacando as lições aprendidas e as possíveis melhorias para futuras implementações.

A divisão das tarefas foi feita da seguinte maneira: Emmanuel Viglioni ficou responsável pela implementação da solução utilizando divisão e conquista; Lucas Machado de Oliveira Andrade desenvolveu a abordagem de programação dinâmica; Marcelo Aguilar Araújo D'Almeida trabalhou nas estratégias gulosas; e Paulo Victor Pimenta Rubinger implementou a solução via backtracking. Essa divisão permitiu uma análise aprofundada e individualizada de cada técnica, contribuindo para uma compreensão mais ampla e detalhada do problema e das soluções possíveis.

1.2 Problema

No cenário atual, a gestão eficiente de recursos é crucial para a competitividade e sustentabilidade das empresas, especialmente aquelas envolvidas na produção e distribuição de energia. Empresas produtoras de energia frequentemente enfrentam o desafio de maximizar seus lucros ao vender sua produção a diversas empresas interessadas. Este problema de otimização, conhecido como o problema do leilão de energia, envolve a venda de uma quantidade fixa de energia a diferentes compradores, cada um fazendo ofertas específicas para lotes de tamanhos determinados.

A complexidade do problema reside na necessidade de escolher quais ofertas aceitar de forma que o valor total obtido seja maximizado, considerando que cada empresa interessada comprará apenas um lote do tamanho exato da oferta. Essa característica confere ao problema uma natureza combinatória intratável, tipicamente pertencente à classe NP. A dificuldade em encontrar a solução ótima aumenta exponencialmente com o número de empresas interessadas e suas respectivas ofertas, exigindo a aplicação de técnicas avançadas de projeto e análise de algoritmos. Abaixo segue um exemplo prático do problema que estamos lidando:

> "Uma empresa produtora de energia possui uma quantidade X de energia, medida em megawatts, para vender. Seu objetivo é vender sua energia produzida, obtendo o maior valor possível no conjunto de suas vendas. As vendas serão realizadas por leilão: cada empresa interessada dará um lance por um lote de K megawatts, oferecendo um valor V por este lote. As interessadas só comprarão um lote do tamanho exato da oferta. Por exemplo, suponha que a produtora tenha 1000 megawatts para venda e tenhamos os seguintes lances do leilão:
>
> - Interessada I1, 500 megawatts, 500 dinheiros
> - Interessada I2, 500 megawatts, 510 dinheiros
> - Interessada I3, 400 megawatts, 520 dinheiros
> - Interessada I4, 300 megawatts, 400 dinheiros
> - Interessada I5, 200 megawatts, 220 dinheiros
> - Interessada I6, 900 megawatts, 1.110 dinheiros
>
> Uma possível venda seria vender para as interessadas I1 e I2, com valor total de 1.010 dinheiros. Outra venda possível, com maior valor, seria para as interessadas I2, I4 e I5 com valor total de 1.130 dinheiros. Veja, por exemplo, que se for feita a venda para as interessadas I3, I4 e I5, o valor total seria de 1.140 dinheiros, mesmo sem vender toda a energia produzida."

2 IMPLEMENTAÇÕES

2.1 BackTracking

Para a solução do problema de leilão de energia utilizando o algoritmo de Backtracking, o método calcular recebe a capacidade total de megawatts e a lista de ofertas como parâmetros. O objetivo é encontrar a combinação de ofertas que maximize o valor total, sem exceder a capacidade disponível.

O algoritmo começa registrando o tempo de início da execução. Em seguida, inicializa um objeto Resultado com valor máximo zero, uma lista vazia de ofertas selecionadas e tempo de execução zero. A função recursiva backtrack é então chamada para iniciar a busca pela melhor combinação de ofertas.

Estrutura da Função Backtracking

A função backtrack é responsável por explorar todas as possíveis combinações de ofertas, verificando se cada combinação resulta em um valor total maior que o atualmente armazenado no melhorResultado. A cada chamada recursiva, a função considera adicionar uma nova oferta à lista de selecionadas, verifica se a capacidade permite essa adição, e atualiza o valor e capacidade restantes.

Parâmetros da função backtrack:

capacidade: a capacidade restante de megawatts.
ofertas: a lista de todas as ofertas disponíveis.
indice: o índice da oferta atual sendo considerada.
selecionadas: a lista de ofertas selecionadas até o momento.
valorAtual: o valor total das ofertas selecionadas até o momento.
melhorResultado: o melhor resultado encontrado até o momento.

Procedimento da função backtrack:

- Se valorAtual for maior que o valor máximo armazenado no melhorResultado, atualiza o melhorResultado com o novo valor e a lista de ofertas selecionadas.
- Para cada oferta a partir do índice atual, verifica se a oferta pode ser considerada sem exceder a capacidade.
- Se a oferta for considerada, adiciona-a à lista de selecionadas, atualiza a

capacidade e o valor atual, e chama a função recursivamente para continuar a busca.

- Após explorar a inclusão da oferta, remove-a da lista de selecionadas e restaura a capacidade e o valor atual para considerar outras combinações.
- Após completar a busca, o tempo de execução é registrado e atualizado no melhorResultado, que é então retornado.

Resultados

Para os testes, incrementou-se o número de empresas interessadas na base de dados até que o tempo de execução fosse aproximadamente 30 segundos. Para 32 empresas, obteve-se um tempo médio de aproximadamente 25 segundos, enquanto que para 33 empresas, o tempo médio foi de aproximadamente 49 segundos.

Ofertas selecionadas considerando 33 empresas:

Conjunto 1:
Valor máximo que pode ser obtido: 29197 dinheiros
Tempo de execução: 49097388292 nanosegundos
Ofertas selecionadas:

- E1 comprou 393 MW por 1443 dinheiros
- E2 comprou 383 MW por 1660 dinheiros
- E3 comprou 399 MW por 1299 dinheiros
- E4 comprou 402 MW por 1487 dinheiros
- E5 comprou 380 MW por 1644 dinheiros
- E6 comprou 387 MW por 1181 dinheiros
- E7 comprou 410 MW por 1644 dinheiros
- E9 comprou 423 MW por 1600 dinheiros
- E10 comprou 399 MW por 1697 dinheiros
- E12 comprou 424 MW por 1622 dinheiros
- E15 comprou 425 MW por 1484 dinheiros
- E16 comprou 390 MW por 1678 dinheiros
- E17 comprou 387 MW por 1507 dinheiros
- E19 comprou 406 MW por 1328 dinheiros

- E20 comprou 412 MW por 1463 dinheiros
- E22 comprou 409 MW por 1171 dinheiros
- E23 comprou 392 MW por 1286 dinheiros
- E29 comprou 396 MW por 1439 dinheiros
- E32 comprou 390 MW por 1051 dinheiros
- E33 comprou 391 MW por 1513 dinheiros

Conjunto 2:

Valor máximo que pode ser obtido: 29166 dinheiros

Tempo de execução: 48819586000 nanosegundos

Ofertas selecionadas:

- E2 comprou 419 MW por 1428 dinheiros
- E6 comprou 383 MW por 1470 dinheiros
- E8 comprou 392 MW por 1610 dinheiros
- E9 comprou 389 MW por 1435 dinheiros
- E10 comprou 399 MW por 1320 dinheiros
- E12 comprou 393 MW por 1610 dinheiros
- E14 comprou 389 MW por 1231 dinheiros
- E15 comprou 397 MW por 1655 dinheiros
- E16 comprou 388 MW por 1031 dinheiros
- E17 comprou 431 MW por 1492 dinheiros
- E18 comprou 386 MW por 1424 dinheiros
- E21 comprou 388 MW por 1347 dinheiros
- E24 comprou 416 MW por 1481 dinheiros
- E25 comprou 385 MW por 1588 dinheiros
- E27 comprou 410 MW por 1612 dinheiros
- E28 comprou 404 MW por 1551 dinheiros
- E30 comprou 415 MW por 1602 dinheiros
- E31 comprou 396 MW por 1241 dinheiros
- E32 comprou 396 MW por 1486 dinheiros
- E33 comprou 422 MW por 1552 dinheiros

Conjunto 3:

Valor máximo que pode ser obtido: 28580 dinheiros

Tempo de execução: 49919296708 nanosegundos

Ofertas selecionadas:

- E2 comprou 389 MW por 1578 dinheiros
- E4 comprou 422 MW por 1362 dinheiros
- E7 comprou 434 MW por 1490 dinheiros
- E8 comprou 396 MW por 1289 dinheiros
- E9 comprou 394 MW por 1580 dinheiros
- E10 comprou 430 MW por 1572 dinheiros
- E11 comprou 399 MW por 1638 dinheiros
- E13 comprou 431 MW por 1432 dinheiros
- E15 comprou 410 MW por 1618 dinheiros
- E18 comprou 394 MW por 1523 dinheiros
- E19 comprou 398 MW por 1332 dinheiros
- E20 comprou 414 MW por 1666 dinheiros
- E21 comprou 417 MW por 1698 dinheiros
- E22 comprou 417 MW por 1300 dinheiros
- E25 comprou 437 MW por 1581 dinheiros
- E26 comprou 427 MW por 1499 dinheiros
- E28 comprou 426 MW por 1286 dinheiros
- E31 comprou 436 MW por 1537 dinheiros
- E32 comprou 440 MW por 1599 dinheiros

Conjunto 4:

Valor máximo que pode ser obtido: 28055 dinheiros

Tempo de execução: 49690506477 nanosegundos

Ofertas selecionadas:

- E1 comprou 399 MW por 1481 dinheiros
- E3 comprou 395 MW por 1268 dinheiros
- E4 comprou 427 MW por 1403 dinheiros
- E5 comprou 392 MW por 1425 dinheiros
- E6 comprou 416 MW por 1511 dinheiros
- E9 comprou 420 MW por 1641 dinheiros
- E10 comprou 418 MW por 1336 dinheiros
- E11 comprou 418 MW por 1326 dinheiros

- E13 comprou 420 MW por 1226 dinheiros
- E14 comprou 414 MW por 1227 dinheiros
- E15 comprou 409 MW por 1405 dinheiros
- E16 comprou 416 MW por 1651 dinheiros
- E19 comprou 430 MW por 1631 dinheiros
- E20 comprou 437 MW por 1422 dinheiros
- E22 comprou 422 MW por 1598 dinheiros
- E23 comprou 402 MW por 1495 dinheiros
- E24 comprou 434 MW por 1661 dinheiros
- E26 comprou 405 MW por 1547 dinheiros
- E27 comprou 398 MW por 1495 dinheiros

Conjunto 5:

Valor máximo que pode ser obtido: 29251 dinheiros

Tempo de execução: 49547351375 nanosegundos

Ofertas selecionadas:

- E1 comprou 425 MW por 1603 dinheiros
- E3 comprou 437 MW por 1616 dinheiros
- E5 comprou 410 MW por 1584 dinheiros
- E6 comprou 399 MW por 1609 dinheiros
- E7 comprou 390 MW por 1674 dinheiros
- E8 comprou 386 MW por 1428 dinheiros
- E9 comprou 382 MW por 1431 dinheiros
- E10 comprou 426 MW por 1430 dinheiros
- E11 comprou 396 MW por 1415 dinheiros
- E12 comprou 439 MW por 1579 dinheiros
- E13 comprou 383 MW por 1427 dinheiros
- E14 comprou 427 MW por 1697 dinheiros
- E16 comprou 392 MW por 1681 dinheiros
- E20 comprou 389 MW por 1418 dinheiros
- E22 comprou 429 MW por 1695 dinheiros
- E23 comprou 440 MW por 1403 dinheiros
- E27 comprou 425 MW por 1439 dinheiros

- E32 comprou 395 MW por 1478 dinheiros
- E33 comprou 397 MW por 1644 dinheiros

Conjunto 6:

Valor máximo que pode ser obtido: 28489 dinheiros

Tempo de execução: 49849577958 nanosegundos

Ofertas selecionadas:

- E3 comprou 432 MW por 1532 dinheiros
- E4 comprou 420 MW por 1618 dinheiros
- E5 comprou 413 MW por 1424 dinheiros
- E8 comprou 391 MW por 1497 dinheiros
- E9 comprou 419 MW por 1494 dinheiros
- E10 comprou 429 MW por 1498 dinheiros
- E14 comprou 434 MW por 1481 dinheiros
- E15 comprou 417 MW por 1484 dinheiros
- E17 comprou 416 MW por 1435 dinheiros
- E19 comprou 428 MW por 1487 dinheiros
- E20 comprou 404 MW por 1364 dinheiros
- E21 comprou 411 MW por 1605 dinheiros
- E24 comprou 425 MW por 1661 dinheiros
- E25 comprou 399 MW por 1456 dinheiros
- E26 comprou 408 MW por 1421 dinheiros
- E29 comprou 382 MW por 1459 dinheiros
- E30 comprou 433 MW por 1558 dinheiros
- E31 comprou 422 MW por 1429 dinheiros
- E33 comprou 384 MW por 1586 dinheiros

Conjunto 7:

Valor máximo que pode ser obtido: 29459 dinheiros

Tempo de execução: 48947803000 nanosegundos

Ofertas selecionadas:

- E1 comprou 417 MW por 1499 dinheiros

- E2 comprou 425 MW por 1644 dinheiros
- E4 comprou 408 MW por 1652 dinheiros
- E5 comprou 404 MW por 1560 dinheiros
- E7 comprou 403 MW por 1639 dinheiros
- E10 comprou 437 MW por 1655 dinheiros
- E13 comprou 396 MW por 1381 dinheiros
- E16 comprou 388 MW por 1505 dinheiros
- E17 comprou 415 MW por 1622 dinheiros
- E19 comprou 434 MW por 1471 dinheiros
- E21 comprou 384 MW por 1639 dinheiros
- E23 comprou 392 MW por 1424 dinheiros
- E24 comprou 417 MW por 1681 dinheiros
- E25 comprou 432 MW por 1505 dinheiros
- E27 comprou 383 MW por 1566 dinheiros
- E28 comprou 428 MW por 1610 dinheiros
- E30 comprou 402 MW por 1582 dinheiros
- E32 comprou 409 MW por 1329 dinheiros
- E33 comprou 426 MW por 1495 dinheiros

Conjunto 8:

Valor máximo que pode ser obtido: 28956 dinheiros

Tempo de execução: 51562105708 nanosegundos

Ofertas selecionadas:

- E1 comprou 380 MW por 1455 dinheiros
- E3 comprou 419 MW por 1486 dinheiros
- E4 comprou 388 MW por 1544 dinheiros
- E5 comprou 394 MW por 1399 dinheiros
- E6 comprou 420 MW por 1592 dinheiros
- E7 comprou 382 MW por 1435 dinheiros
- E8 comprou 389 MW por 1641 dinheiros
- E9 comprou 403 MW por 1441 dinheiros
- E11 comprou 430 MW por 1455 dinheiros
- E14 comprou 429 MW por 1672 dinheiros

- E15 comprou 390 MW por 1669 dinheiros
- E19 comprou 424 MW por 1418 dinheiros
- E20 comprou 420 MW por 1517 dinheiros
- E21 comprou 410 MW por 1458 dinheiros
- E23 comprou 382 MW por 1440 dinheiros
- E28 comprou 426 MW por 1632 dinheiros
- E30 comprou 434 MW por 1629 dinheiros
- E31 comprou 427 MW por 1454 dinheiros
- E33 comprou 439 MW por 1619 dinheiros

Conjunto 9:

Valor máximo que pode ser obtido: 28176 dinheiros

Tempo de execução: 49025411584 nanosegundos

Ofertas selecionadas:

- E3 comprou 411 MW por 1677 dinheiros
- E5 comprou 428 MW por 1423 dinheiros
- E6 comprou 389 MW por 1556 dinheiros
- E7 comprou 439 MW por 1370 dinheiros
- E8 comprou 406 MW por 1440 dinheiros
- E9 comprou 387 MW por 1,54 dinheiros
- E10 comprou 390 MW por 1649 dinheiros
- E11 comprou 382 MW por 1453 dinheiros
- E12 comprou 404 MW por 1412 dinheiros
- E14 comprou 388 MW por 1539 dinheiros
- E15 comprou 389 MW por 1396 dinheiros
- E17 comprou 402 MW por 1573 dinheiros
- E18 comprou 398 MW por 1277 dinheiros
- E20 comprou 391 MW por 1307 dinheiros
- E21 comprou 398 MW por 1407 dinheiros
- E24 comprou 394 MW por 1149 dinheiros
- E27 comprou 431 MW por 1313 dinheiros
- E28 comprou 391 MW por 1573 dinheiros
- E31 comprou 385 MW por 1173 dinheiros

- E33 comprou 395 MW por 1335 dinheiros

Conjunto 10:

Valor máximo que pode ser obtido: 28567 dinheiros

Tempo de execução: 49374412542 nanosegundos

Ofertas selecionadas:

- E2 comprou 402 MW por 1286 dinheiros
- E3 comprou 398 MW por 1662 dinheiros
- E4 comprou 391 MW por 1187 dinheiros
- E5 comprou 381 MW por 1447 dinheiros
- E7 comprou 392 MW por 1653 dinheiros
- E12 comprou 405 MW por 1316 dinheiros
- E13 comprou 380 MW por 1643 dinheiros
- E14 comprou 389 MW por 1407 dinheiros
- E15 comprou 420 MW por 1390 dinheiros
- E16 comprou 428 MW por 1598 dinheiros
- E18 comprou 418 MW por 1672 dinheiros
- E19 comprou 397 MW por 1453 dinheiros
- E22 comprou 384 MW por 1385 dinheiros
- E24 comprou 387 MW por 1107 dinheiros
- E26 comprou 408 MW por 1392 dinheiros
- E28 comprou 402 MW por 1308 dinheiros
- E29 comprou 382 MW por 1540 dinheiros
- E30 comprou 402 MW por 1313 dinheiros
- E31 comprou 431 MW por 1669 dinheiros
- E33 comprou 403 MW por 1139 dinheiros

33 Empresas - Média de tempo de execução: 49560.341912600 milissegundos

2.2.1 Guloso - Estratégia 1 - Ordenação pelo maior valor,

A estratégia gulosa é um método eficiente para resolver problemas de otimização, onde a melhor escolha local em cada etapa leva a uma solução globalmente ótima. Vamos detalhar a implementação dessa estratégia no contexto de

seleção de ofertas de energia, explicando cada passo e justificando as decisões tomadas.

1. Início da Contagem do Tempo de Execução

O algoritmo começa registrando o tempo de início da execução. Isso é feito para calcular o tempo total necessário para a conclusão do processo, fornecendo uma medida de desempenho.

2. Ordenação das Ofertas pelo Valor

As ofertas são ordenadas em ordem decrescente de valor total. A ideia é priorizar as ofertas que trazem maior retorno financeiro, de modo que sejam consideradas primeiro pelo algoritmo.

Exemplo de Ordenação: Imagine que temos uma capacidade total de 10 MW e as seguintes ofertas:

- Oferta 1: 5 MW por R$60
- Oferta 2: 3 MW por R$40
- Oferta 3: 4 MW por R$50

Ordenando pelo valor total, teríamos:

- Oferta 1: R$60 (5 MW)
- Oferta 3: R$50 (4 MW)
- Oferta 2: R$40 (3 MW)

3. Seleção das Ofertas

Após a ordenação, o algoritmo percorre a lista de ofertas e, para cada oferta, verifica se a capacidade restante é suficiente para aceitá-la:

- Se a capacidade permitir: A oferta é adicionada à lista de ofertas selecionadas (ofertasSelecionadas).
- Redução da Capacidade Disponível: A capacidade disponível é reduzida pelo valor em megawatts da oferta aceita.
- Atualização do Valor Total: O valor da oferta é adicionado ao valorTotal.

Processo de Seleção:

- Capacidade Inicial: 10 MW
- Avaliando a Oferta 1 (5 MW por R$60):

- Capacidade restante: 5 MW
- Valor total acumulado: R$60
- Ofertas selecionadas: [Oferta 1]
- Avaliando a Oferta 3 (4 MW por R$50):
- Capacidade restante: 1 MW
- Valor total acumulado: R$110
- Ofertas selecionadas: [Oferta 1, Oferta 3]
- Avaliando a Oferta 2 (3 MW por R$40):
- Capacidade restante: 1 MW (não suficiente para 3 MW)
- Valor total acumulado: R$110 (sem alteração)
- Ofertas selecionadas: [Oferta 1, Oferta 3]

4 . Registro do Tempo Final e Cálculo do Tempo de Execução

Uma vez que todas as ofertas foram avaliadas, o algoritmo registra o tempo final de execução. A diferença entre o tempo final e o tempo de início fornece o tempo total de execução.

Após a ordenação, o algoritmo percorre a lista de ofertas e, para cada oferta, verifica se a capacidade restante é suficiente para aceitá-la:

- Se a capacidade permitir: A oferta é adicionada à lista de ofertas selecionadas (ofertasSelecionadas).
- Redução da Capacidade Disponível: A capacidade disponível é reduzida pelo valor em megawatts da oferta aceita.
- Atualização do Valor Total: O valor da oferta é adicionado ao valorTotal.

Vantagens da Estratégia Gulosa Ordenação por Valor

- A estratégia gulosa é simples de implementar e geralmente muito eficiente em termos de tempo de execução, pois toma decisões rápidas e diretas.
- Ao ordenar as ofertas pelo maior valor total, o algoritmo maximiza o retorno financeiro para cada unidade de capacidade utilizada. Isso garante que as ofertas mais valiosas sejam consideradas primeiro, resultando em uma utilização eficiente da capacidade disponível.

Desvantagens da Estratégia Gulosa Ordenação por Valor

- A abordagem gulosa pode não resultar na melhor solução global em todos os casos. Por exemplo, se uma combinação de ofertas menores resultar em um valor total maior, a estratégia gulosa pode não detectar isso se focar apenas nas ofertas de maior valor

- Em problemas onde a relação entre valor e capacidade é complexa ou não linear, a estratégia gulosa pode levar a soluções subótimas, pois não revisa decisões anteriores para garantir a otimalidade global.

Tempo de Ordenação

Para a ordenação das ofertas, o algoritmo utiliza a estrutura Collections.sort, que implementa o algoritmo TimSort. TimSort é um algoritmo híbrido, baseado em divisão e conquista, combinado com inserções em busca binária. Ele possui uma complexidade de tempo de O(n log n), o que é eficiente para a ordenação de listas de tamanho considerável. Essa escolha garante que a fase de ordenação do algoritmo seja rápida e eficiente, contribuindo para o desempenho geral do processo.

Número de Empresas	**Tempo Médio de Execução (milissegundos)**
33	0.103340000000
66	0.126830000000
99	0.293170000000
132	0.239110000000
165	0.245060000000
198	0.254960000000
231	0.223460000000
264	0.302620000000
297	0.211070000000
330	0.210680000000

Os resultados do algoritmo guloso com a estratégia de ordenação por maior valor em ordem decrescente mostram que o tempo de execução aumenta conforme o número de empresas aumenta. Observa-se um desempenho aceitável para quantidades moderadas de empresas, mas variações no tempo de execução podem ocorrer devido à complexidade dos dados. Esses resultados indicam que o algoritmo está funcionando de maneira eficiente para conjuntos de dados de tamanho moderado, mas podem surgir desafios com conjuntos de dados mais complexos.

2.2.2 Guloso - Estratégia 2 - Ordenação pelo maior valor do Megawatt

Na estratégia 2, o algoritmo guloso ordena as ofertas pelo valor por megawatt (R$/MW) de forma decrescente. O objetivo é maximizar a utilização eficiente da capacidade disponível, priorizando as ofertas que oferecem o maior valor por unidade de capacidade.

1. Ordenação das Ofertas pelo Valor por Megawatt

As ofertas são ordenadas em ordem decrescente de valor por megawatt. Isso significa que as ofertas que oferecem maior retorno financeiro por unidade de megawatt serão consideradas primeiro.

Exemplo de Ordenação: Imagine que temos uma capacidade total de 10 MW e as seguintes ofertas:

- Oferta 1: 5 MW por R$60 (12 R$/MW)
- Oferta 2: 3 MW por R$40 (13.33 R$/MW)
- Oferta 3: 4 MW por R$50 (12.5 R$/MW)

Ordenando pelo valor por megawatt, teríamos:

- Oferta 2: 13.33 R$/MW (3 MW por R$40)
- Oferta 3: 12.5 R$/MW (4 MW por R$50)
- Oferta 1: 12 R$/MW (5 MW por R$60)

2. Seleção das Ofertas

Após a ordenação, o algoritmo percorre a lista de ofertas e, para cada oferta, verifica se a capacidade restante é suficiente para aceitá-la:

- Se a capacidade permitir: A oferta é adicionada à lista de ofertas selecionadas (ofertasSelecionadas).
- Redução da Capacidade Disponível: A capacidade disponível é

reduzida pelo valor em megawatts da oferta aceita.

- Atualização do Valor Total: O valor da oferta é adicionado ao valorTotal.

Processo de Seleção:

- Capacidade Inicial: 10 MW
- Avaliando a Oferta 2 (3 MW por R$40):
- Capacidade restante: 7 MW
- Valor total acumulado: R$40
- Ofertas selecionadas: [Oferta 2]
- Avaliando a Oferta 3 (4 MW por R$50):
- Capacidade restante: 3 MW
- Valor total acumulado: R$90
- Ofertas selecionadas: [Oferta 2, Oferta 3]
- Avaliando a Oferta 1 (5 MW por R$60):
- Capacidade restante: 3 MW (não suficiente para 5 MW)
- Valor total acumulado: R$90 (sem alteração)
- Ofertas selecionadas: [Oferta 2, Oferta 3]

3. Registro do Tempo Final e Cálculo do Tempo de Execução

O algoritmo registra o tempo final de execução e calcula a diferença entre o tempo final e o tempo de início para obter o tempo total de execução.

4. Retorno do Resultado

O algoritmo retorna um objeto Resultado, que contém:

- valorTotal: O valor total acumulado pelas ofertas selecionadas.
- ofertasSelecionadas: A lista de ofertas que foram aceitas.
- tempoDeExecucao: O tempo total de execução do algoritmo.

Vantagens da Estratégia Gulosa - Ordenação por Valor por Megawatt

- Ao ordenar as ofertas pelo maior valor por megawatt, o algoritmo garante que cada unidade de capacidade seja utilizada da forma mais eficiente possível em termos de valor monetário.
- A abordagem gulosa, ao priorizar o maior valor por megawatt,

facilita a tomada de decisões locais ótimas, que frequentemente levam a uma boa solução global.

Desvantagens da Estratégia Gulosa - Ordenação por Valor por Megawatt

- Se a distribuição dos valores por megawatt não seguir uma tendência clara, ordenar apenas por esse critério pode não resultar na melhor solução global.
- Em cenários onde a relação entre valor e capacidade não é linear, a abordagem gulosa pode resultar em soluções subótimas, pois não revisa decisões anteriores para garantir a otimalidade global.

Número de Empresas	Tempo Médio de Execução (milissegundos)
33	0.012130000000
66	0.016470000000
99	0.026930000000
132	0.044680000000
165	0.039210000000
198	0.050790000000
231	0.051140000000
264	0.112180000000
297	0.077140000000
330	0.079910000000

Os resultados do algoritmo guloso, utilizando a estratégia de ordenação por valor do megawatt em ordem decrescente, mostram que o tempo médio de execução aumenta conforme o número de empresas consideradas cresce. Para quantidades moderadas de empresas, o desempenho é satisfatório, porém há variações no tempo de execução que podem ser influenciadas pela complexidade dos dados. Em resumo, o algoritmo funciona de maneira eficiente para conjuntos de dados de tamanho moderado, mas pode enfrentar desafios com conjuntos de dados mais complexos.

2.2 Divisão e Conquista

Divisão e Conquista é uma técnica poderosa e amplamente utilizada na área de algoritmos e estrutura de dados. Para entender melhor, imagine que você tem uma tarefa complexa para realizar. Ao invés de tentar resolver tudo de uma vez, você a divide em partes menores e mais manejáveis. Você resolve cada parte individualmente e depois combina as soluções para resolver o problema original. Esse é o princípio básico da Divisão e Conquista.

Na nossa implementação, optamos por utilizar a técnica de memorização para otimizar a resolução do problema, evitando recalcular sub problemas já resolvidos e melhorando significativamente a eficiência do algoritmo. A memorização funciona como um bloco de notas: quando um subproblema é resolvido pela primeira vez, armazenamos o resultado nesse bloco de notas junto com uma chave única que representa esse subproblema. Essa chave, no nosso caso, é gerada a partir da capacidade restante e do índice da oferta sendo analisada. Se o algoritmo se deparar com o mesmo subproblema novamente, ele simplesmente busca o resultado no bloco de notas usando a chave, sem precisar recalculá-lo. A estrutura de dados que utilizamos para implementar essa técnica foi o HashMap, por ser eficiente tanto na inserção quanto na busca de elementos através de chaves.

Além da memorização, a ordenação inicial das ofertas pelo valor por megawatt (de forma decrescente) também foi crucial para a otimização do algoritmo. Essa ordenação garante que as ofertas mais vantajosas sejam avaliadas primeiro, maximizando o uso da capacidade disponível e aumentando a chance de encontrar a solução ótima mais rapidamente. Imagine, por exemplo, que temos uma capacidade de 100 MW e duas ofertas: uma de 60 MW por R$100 e outra de 40 MW por R$90. Se avaliarmos a oferta de menor valor primeiro, podemos acabar escolhendo uma combinação de ofertas menos vantajosa. Ao ordenar pelo valor por megawatt, garantimos que a oferta de maior valor seja sempre considerada a primeira.

A implementação do algoritmo se divide em duas partes principais: o método "calcular", que atua como a porta de entrada, e o método "calcularRecursivo", responsável pela divisão do problema e combinação dos resultados. O método "calcular" inicia o cronômetro para medir o tempo de execução, ordena as ofertas pelo

valor por megawatt e chama o método "calcularRecursivo" para obter o resultado final, que inclui as ofertas selecionadas, o tempo de execução e o valor máximo alcançado.

O método "calcularRecursivo" é o coração do algoritmo. Ele verifica primeiro se a capacidade restante é zero ou se todas as ofertas já foram processadas. Se uma dessas condições for verdadeira, significa que chegamos ao final da recursão e o método retorna 0. Caso contrário, o algoritmo gera a chave única para o subproblema atual e verifica se o resultado já está armazenado no HashMap. Se estiver, o resultado é recuperado e utilizado. Caso contrário, o algoritmo avalia a oferta atual e calcula o valor máximo que pode ser obtido incluindo ou não essa oferta na solução. Para isso, ele chama recursivamente o método "calcularRecursivo" para os subproblemas gerados a partir dessas duas possibilidades.

Após avaliar as duas opções, o algoritmo seleciona a que resulta no maior valor e armazena o resultado no HashMap, juntamente com a chave correspondente. Esse processo se repete recursivamente até que todos os sub problemas sejam resolvidos. A classe auxiliar "ResultadoParcial" é utilizada para armazenar o valor e as ofertas selecionadas para cada subproblema, facilitando a recuperação dos resultados memorizados.

Para ilustrar o funcionamento do algoritmo, considere um exemplo com capacidade total de 10 MW e as seguintes ofertas:

- Oferta 1: 5 MW por R$60
- Oferta 2: 3 MW por R$40
- Oferta 3: 4 MW por R$50

O algoritmo, após ordenar as ofertas, começaria avaliando a Oferta 1. A partir daí, ele se dividiria em dois subproblemas: um incluindo a Oferta 1 e outro não. O subproblema que inclui a Oferta 1 teria uma capacidade restante de 5 MW e avaliaria as Ofertas 2 e 3. Já o subproblema que não inclui a Oferta 1 teria uma capacidade restante de 10 MW e também avaliaria as Ofertas 2 e 3. Esse processo se repetiria recursivamente até que todas as combinações fossem avaliadas e a solução ótima fosse

encontrada.

A análise dos resultados obtidos nos testes demonstra a eficiência do algoritmo de Divisão e Conquista com memorização. Para avaliar o desempenho do algoritmo, foram realizados testes com conjuntos de dados de tamanho crescente, variando de 10 a 33 empresas, e realizamos uma execução com 1000 empresas para testar o comportamento com grandes entradas. Para cada tamanho de conjunto, foram executados 10 testes, e o tempo médio de execução foi registrado. A tabela abaixo apresenta os resultados obtidos:

Número de Empresas	Tempo Médio de Execução (segundos)
10	0.001300845900
20	0.007489479300
30	0.025200595800
33	0.031487412500
1000	0.075831600000

Como podemos observar na tabela, o tempo de execução do algoritmo aumenta gradualmente conforme o número de empresas cresce. Esse comportamento é esperado, visto que a complexidade do algoritmo é influenciada pelo tamanho do conjunto de dados. No entanto, mesmo com o aumento do número de empresas, o tempo de execução se mantém dentro de limites aceitáveis, demonstrando a eficiência da implementação.

Para o problema base, do leilão que encontra-se na Seção "1.2 - Problema", o algoritmo de divisão e conquista encontrou os resultados: I4 comprou 300MW por 400 dinheiros, I3 comprou 400MW por 520 dinheiros e I5 comprou 200MW por 220 dinheiros. Obtendo-se assim o lucro máximo de 1140 dinheiros e essa solução foi encontrada em 0.000445250000 segundos

A capacidade de lidar com conjuntos de dados de tamanho considerável em

tempo hábil, juntamente com a garantia de encontrar a solução ótima, torna essa implementação uma ferramenta poderosa para auxiliar na tomada de decisão em leilões de energia, maximizando o lucro da empresa produtora.

2.3 Programação Dinâmica

O algoritmo de programação dinâmica utiliza como base a criação de uma tabela que armazena as melhores soluções possíveis para o problema. Assim, ao finalizar sua execução, o último dado inserido na tabela é o valor desejado. Entretanto, como apenas o valor é armazenado, é necessário posteriormente executar um segundo algoritmo para identificar os candidatos que contribuíram para a solução. Logo, a programação dinâmica é capaz de encontrar um resultado para o problema, garantindo que o resultado final seja exato, mas não é capaz de encontrar todos os resultados possíveis. Ou seja, se um problema possui múltiplas soluções possíveis, a programação dinâmica encontrará uma dessas soluções.

Na implementação, a solução ocorreu em três etapas. Na primeira etapa, houve a construção inicial da tabela dinâmica. Para isso, é necessário definir quais valores serão utilizados para as linhas e quais serão utilizados para as colunas.

As linhas da tabela dinâmica referenciam as possibilidades de resultados, ou seja, os candidatos. Foi definido que a tabela possuiria a quantidade de ofertas mais uma linha, sendo a primeira linha a base.

Para as colunas da tabela, há a necessidade de referenciar as soluções. Logo, a tabela foi criada com a quantidade limite de megawatts disponíveis para a venda mais um, incluindo o valor inicial de 0, que é a base do problema.

Por fim, como o problema está relacionado à maximização do valor de venda em dinheiro, todos os dados foram inicializados com 0 a princípio.

Na segunda etapa da implementação, foi feito o preenchimento da tabela. Cada célula da tabela deve ser preenchida individualmente (totalizando dois loops), percorrendo cada coluna para a primeira oferta, depois todas as colunas da segunda

oferta e assim sucessivamente. O loop externo percorre as linhas da tabela, que correspondem às ofertas, e as colunas percorrem os valores de capacidade de venda.

No preenchimento da tabela, a cada iteração de célula, é feita uma verificação inicial para verificar se é possível realizar a compra da oferta com base no valor da coluna. Isso ocorre somente se a quantidade de megawatts para venda for inferior ao índice da coluna, que é referente à capacidade de venda que está sendo analisada no momento. Ou seja, se uma empresa possui uma oferta para compra de 300 megawatts, mas a análise está sendo feita na coluna de valor 200, não há megawatts disponíveis para a venda dessa oferta. Nesse caso, a oferta não é inserida na solução, e o valor da célula é o mesmo da linha anterior. Entretanto, caso seja possível inserir a oferta na solução, é feito um cálculo para obter o maior valor ao comparar o resultado da oferta anterior e retirando registros da oferta anterior, somando a oferta atual à solução.

Após o preenchimento completo da tabela dinâmica, o resultado maximizado em dinheiro é retirado do último dado da tabela.

Na terceira etapa, é feita a busca das ofertas que fizeram parte da solução final. Para isso, é feita uma busca pelas linhas da tabela, iniciando na última linha preenchida e no último valor preenchido na coluna. Para cada linha, é feita a comparação se o resultado da célula observada se difere do resultado presente na mesma coluna e na linha anterior. Caso seja diferente, isso significa que a oferta faz parte da solução, sendo então inserida na lista de ofertas para a solução. A análise é repetida para a linha anterior e para a coluna atual, reduzida pela quantidade de megawatts da oferta selecionada. Caso os resultados sejam iguais, a oferta não faz parte da solução, e a análise segue para a linha anterior, mantendo a coluna.

Para reduzir o número de iterações, caso a coluna da tabela chegue a 0, isso significa que o algoritmo está na base da solução. Portanto, não é necessário analisar ofertas anteriores, e o loop é finalizado.

Para os resultados, encontrados nos problemas padrões, foi encontrado o resultado de 26725 dinheiros em tempo médio de 2 milissegundos, com 25 registros analisados. Para isso, a programação dinâmica encontrou os seguintes dados.

E25 comprou 417 MW por 1330 dinheiros
E24 comprou 387 MW por 1280 dinheiros
E23 comprou 415 MW por 1624 dinheiros
E22 comprou 403 MW por 1277 dinheiros
E20 comprou 393 MW por 1533 dinheiros
E19 comprou 392 MW por 1293 dinheiros
E18 comprou 397 MW por 1492 dinheiros
E16 comprou 390 MW por 1206 dinheiros
E15 comprou 387 MW por 1542 dinheiros
E14 comprou 390 MW por 1228 dinheiros
E13 comprou 403 MW por 1568 dinheiros
E12 comprou 406 MW por 1353 dinheiros
E11 comprou 400 MW por 1463 dinheiros
E9 comprou 416 MW por 1386 dinheiros
E7 comprou 416 MW por 1540 dinheiros
E6 comprou 382 MW por 1498 dinheiros
E5 comprou 399 MW por 1214 dinheiros
E4 comprou 385 MW por 1333 dinheiros
E3 comprou 410 MW por 1565 dinheiros

Já nos resultados do segundo problema, foi encontrado o resultado de 40348 dinheiros em tempo médio de 3 milissegundos, com 50 registros analisados. Para isso, a programação dinâmica encontrou os seguintes dados.

E25 comprou 476 MW por 2480 dinheiros
E24 comprou 589 MW por 3164 dinheiros
E23 comprou 279 MW por 1311 dinheiros
E22 comprou 363 MW por 2007 dinheiros
E21 comprou 467 MW por 2038 dinheiros
E20 comprou 358 MW por 1847 dinheiros
E18 comprou 487 MW por 2617 dinheiros
E17 comprou 483 MW por 2828 dinheiros
E16 comprou 240 MW por 1234 dinheiros

E15 comprou 385 MW por 2023 dinheiros
E14 comprou 346 MW por 1552 dinheiros
E13 comprou 213 MW por 1115 dinheiros
E11 comprou 310 MW por 1381 dinheiros
E10 comprou 495 MW por 2259 dinheiros
E9 comprou 558 MW por 2983 dinheiros
E7 comprou 232 MW por 1209 dinheiros
E6 comprou 297 MW por 1499 dinheiros
E4 comprou 433 MW por 2327 dinheiros
E3 comprou 240 MW por 1210 dinheiros
E2 comprou 398 MW por 1768 dinheiros
E1 comprou 313 MW por 1496 dinheiros

Posteriormente foram utilizados o aumento dos dados, iniciando 33 ofertas, e aumentando sucessivamente até o limite de 330. Analisando assim o tempo de execução da programação dinâmica

Número de Empresas	**Tempo Médio de Execução (milissegundos)**
33	0.9728200
99	2.1097401
231	4.3042302
330	5.9852002

Como é possível observar na tabela, houve um aumento gradativo no tempo médio de execução, analisando o tempo médio para 10 bases de testes em cada caso. Isso ocorre devido à complexidade do código em cada uma de suas etapas.

Para a etapa de preenchimento da tabela ocorrem N x M interações, sendo N a quantidade de ofertas do problema e M o valor da capacidade de venda, como esse segundo valor foi fixado em 800, há apenas a variação do número de interações de N.

Já para a etapa de busca de registros, novamente há, para o pior caso, N

interações, para o melhor caso, apenas uma interação, que ocorre quando apenas há um dado na solução e esse é o último registro da tabela.

Logo, o crescimento do tempo de execução é esperado que possua uma relação de crescimento linear, que está relacionada à complexidade no pior caso de 2N.

3 COMPARAÇÕES DE RESULTADOS

3.1 Comparação dos Tempos de Execução

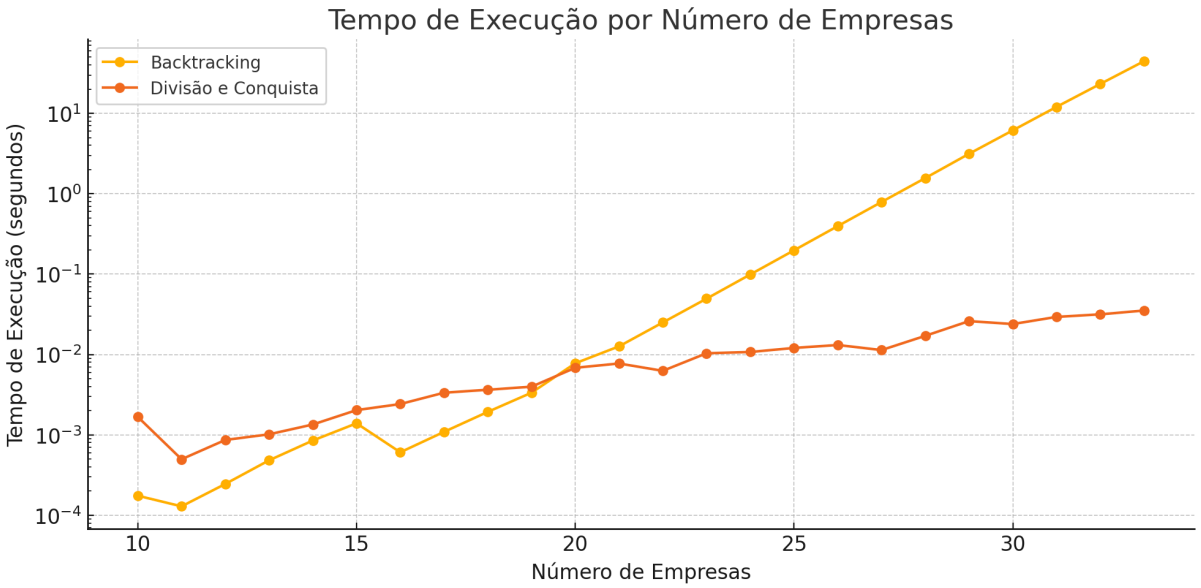
Os resultados obtidos a partir da execução de testes comparativos entre os algoritmos de Divisão e Conquista com memorização e Backtracking para resolver o problema de maximização de valor das ofertas revelam dados significativos sobre a eficiência e escalabilidade dessas abordagens em diferentes cenários.

Número de Empresas	Backtracking (segundos)	Divisão e Conquista (segundos)
10	0.000175	0.001673
11	0.000129	0.000497
12	0.000245	0.000864
13	0.000482	0.001016
14	0.000850	0.001345
15	0.001392	0.002032
16	0.000607	0.002418
17	0.001090	0.003346
18	0.001932	0.003639
19	0.003348	0.003975
20	0.007772	0.006831
21	0.012623	0.007726
22	0.025016	0.006257

23	0.049472	0.010349
24	0.098975	0.010740
25	0.197747	0.012061
26	0.395743	0.013120
27	0.789178	0.011348
28	1.566360	0.017066
29	3.138923	0.026012
30	6.141354	0.023905
31	12.039084	0.029358
32	23.218705	0.031570
33	44.563526	0.035229

Inicialmente, observamos que o algoritmo de Divisão e Conquista com memorização apresenta tempos médios de execução consistentemente menores em comparação com o algoritmo de Backtracking em todos os casos testados. Isso sugere que a estratégia de Divisão e Conquista, aliada à memorização para evitar recalculados, resulta em uma abordagem mais eficiente para resolver o problema em questão.

Ao analisar a evolução dos tempos de execução conforme o número de empresas aumenta, notamos uma diferença significativa entre os dois algoritmos. Enquanto o algoritmo de Backtracking demonstra um aumento exponencial no tempo de execução à medida que o problema se torna mais complexo, o algoritmo de Divisão e Conquista mantém um crescimento mais controlado, o que o torna mais escalável para problemas com um grande número de empresas e ofertas.



Com base no gráfico acima, onde os valores foram multiplicados proporcionalmente, a fins visuais e didáticos, podemos analisar e visualizar a complexidade dos algoritmos, enquanto o BackTracking se mostra exponencial, conforme a entrada. O algoritmo de divisão e conquista oferece uma menor complexidade e consequentemente uma linearidade no gráfico.

Uma análise mais aprofundada revela que a eficiência do algoritmo de Divisão e Conquista está intrinsecamente ligada à sua capacidade de resolver subproblemas de forma independente e reutilizar soluções já calculadas por meio da memorização. Isso reduz drasticamente o número de cálculos redundantes, resultando em um tempo de execução global mais rápido e um desempenho superior em comparação com o algoritmo de Backtracking.

Além disso, é importante destacar que o algoritmo de Divisão e Conquista, quando combinado com a técnica de memorização, não apenas oferece uma solução mais eficiente em termos de tempo de execução, mas também garante a obtenção da solução ótima para o problema de maximização de valor das ofertas dentro da capacidade total disponível. Isso é crucial em cenários onde a precisão e a qualidade da solução são requisitos essenciais.

Em resumo, com base nos resultados dos testes e na análise comparativa

realizada, é recomendável utilizar o algoritmo de Divisão e Conquista com memorização para resolver o problema em questão. Essa abordagem oferece um desempenho superior, escalabilidade para problemas maiores e a garantia de obtenção da solução ótima, tornando-a a escolha mais eficiente e eficaz para resolver o problema de maximização de valor das ofertas.

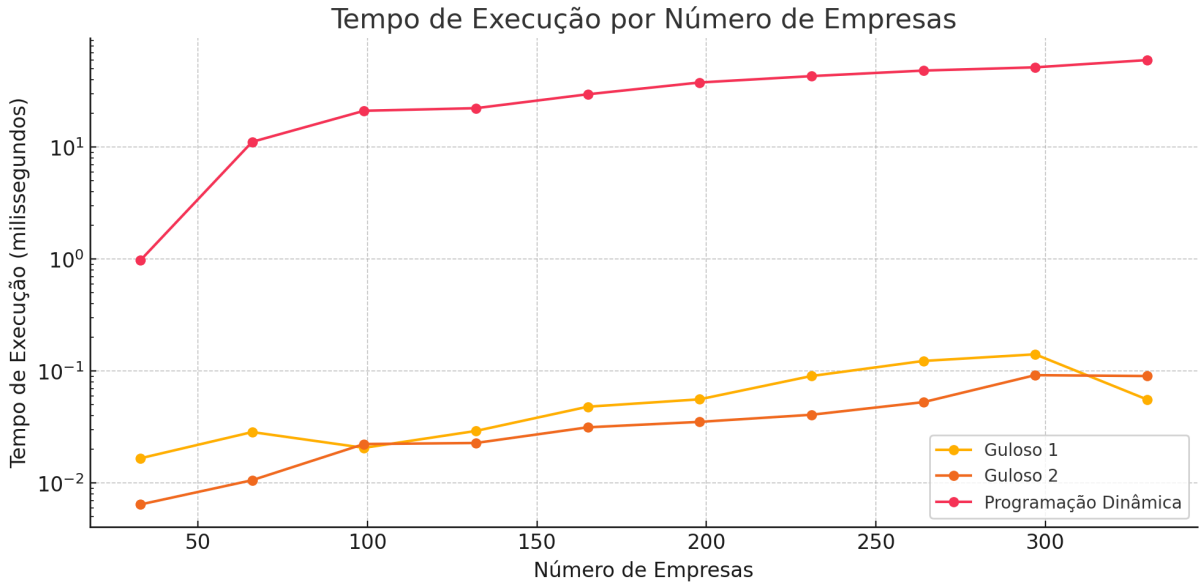
Ao adicionar o algoritmo da programação dinâmica e o guloso a análise, há a necessidade de aumentar o número de registros a fim de comparar. Afinal, é esperado que a velocidade do algoritmo guloso supere todos os demais, entretanto, ele não é capaz de garantir o resultado final esperado. Entretanto, a programação dinâmica é esperada que consiga efetuar com velocidade superior ao backtracking e a divisão e conquista, garantir o melhor resultado, mas não ser tão suficientemente rápido como o guloso. Além disso, a programação dinâmica utiliza de maior memória para armazenar os dados na tabela dinâmica e possuir não somente a solução do problema, mas de todos os outros subproblemas que originam o problema inicial, relacionado à redução da quantidade de energia disponível para venda. Ou seja, ao encontrar, pela programação dinâmica o resultado das ofertas para o problema com 8000 megawatts para venda, há também a solução para 7000, 5000, 3000 e demais valores sem precisar refazer os cálculos.

Abaixo, observa-se os dados comparativos do tempo de execução da programação dinâmica e dos algoritmos gulosos, levando em consideração que com 33 ofertas, o backtracking possuiu um tempo de execução de 44.56 segundos e o divisão e conquista de 35.23 milissegundos.

Número de Empresas	**Guloso 1 (milissegundos)**	**Guloso 2 (milissegundos)**	**Programação Dinâmica (milissegundos)**
33	0.0166203	0.0064302	0.9728200
66	0.0285099	0.0106000	1.1133200
99	0.0206201	0.0222900	2.1097401
132	0.0291800	0.0227801	2.2254996
165	0.0480100	0.0314401	2.9627798

198	0.0559098	0.0351601	3.7736599
231	0.0903001	0.0407101	4.3042302
264	0.1230300	0.0527298	4.8215502
297	0.1410601	0.0918999	5.1546999
330	0.0557301	0.0902001	5.9852002

Os dados abaixo podem ser observados de forma gráfica conforme imagem exibida abaixo.



Como é possível observar, a teoria foi confirmada através dos testes de tempo, afinal, o algoritmo guloso, apenas com N iterações consegue efetuar sua lógica, enquanto a programação dinâmica, embora ainda execute em Ordem de N, possui o dobro de interações, devido ao tempo para preencher os dados e para encontrar os resultados que foram utilizados, ela possui também um aumento no tempo de execução devido a forma como percorre as colunas. Entretanto, para executar a operação com 330 ofertas, ela possui tempo inferior ao divisão e conquista para 33 ofertas, que ocorre em 35 milissegundos, comparados com 5.98 milissegundos da programação dinâmica.

3.2 Comparação da Qualidade dos Resultados

Ao se comparar os resultados obtidos através da execução dos algoritmos,

percebe-se que todos convergem para a mesma solução ótima, com exceção do algoritmo guloso, identificando o conjunto ideal de ofertas que resulta no maior valor possível para a empresa produtora de energia, sendo 26725 dinheiros para a primeira amostra de dados e 40348 dinheiros para a segunda. Essa convergência demonstra que dentro dos algoritmos utilizados, o backtracking não utilizou uma poda demasiadamente agressiva a fim de reduzir o tempo de execução, afinal, apenas o backtracking e o guloso não garantem em completude o melhor resultado possível.

No entanto, o guloso, por priorizar um oferta a cada iteração do código e seguir com sua decisão até o final, não há garantias que todos os melhores dados serão obtidos. Entretanto, a decisão de escolha deve ser analisada para que todos a decisão final do guloso não ocasione em uma divergência muito grande do valor ótimo.

Para a primeira Estratégia do algoritmo guloso, os dados obtidos na resposta foram de 26725 dinheiros para a primeira resposta e 38673 para a segunda. Demonstrando que ele foi capaz de atingir o resultado ótimo na primeira amostra de dados, entretanto, na segunda amostra, não houve uma resposta ótima, mas um resultado próximo do esperado. Isso ocorreu porque na segunda amostra de dados foram tomadas decisões que não eram a melhor solução para o problema, que foram a inclusão dos dados E8 e E19, que não estão previstos na solução ótima, embora tenham uma melhor quantidade de dinheiro oferecido. Devido a essa decisão foram retirados os dados E23, E16, E11, E7 e E3 que estão presentes na solução ótima.

Para a segunda Estratégia do algoritmo guloso, novamente a estratégia foi capaz de obter o resultado ótimo para a primeira base de dados, mas obteve a resposta de 39271 dinheiros para o segundo. Resultado melhor que o obtido pelo algoritmo guloso anterior, entretanto, novamente houve tomadas de decisão que fizeram os dados não convergirem para o resultado ótimo. Afinal, foi retirado o E21, presente na solução ótima e foi adicionado o E12 na solução do guloso. Esse problema ocorre porque a estratégia visa priorizar dados com melhor relação de dinheiro por megawatts, o que pode fazer com que sobre energia que não possa ser vendida para outras ofertas, que embora não possuam melhor custo benefício, maximizam a quantidade de energia comprada e com isso, o dinheiro recebido.

Solução Ótima do primeiro registro de dados:

E3 comprou 410 MW por 1565 dinheiros
E4 comprou 385 MW por 1333 dinheiros
E5 comprou 399 MW por 1214 dinheiros
E6 comprou 382 MW por 1498 dinheiros
E7 comprou 416 MW por 1540 dinheiros
E9 comprou 416 MW por 1386 dinheiros
E11 comprou 400 MW por 1463 dinheiros
E12 comprou 406 MW por 1353 dinheiros
E13 comprou 403 MW por 1568 dinheiros
E14 comprou 390 MW por 1228 dinheiros
E15 comprou 387 MW por 1542 dinheiros
E16 comprou 390 MW por 1206 dinheiros
E18 comprou 397 MW por 1492 dinheiros
E19 comprou 392 MW por 1293 dinheiros
E20 comprou 393 MW por 1533 dinheiros
E22 comprou 403 MW por 1277 dinheiros
E23 comprou 415 MW por 1624 dinheiros
E24 comprou 387 MW por 1280 dinheiros
E25 comprou 417 MW por 1330 dinheiros
Totalizando 26725 dinheiros e 7588 MW vendidos

Solução Ótima do segundo registro de dados:

E1 comprou 313 MW por 1496 dinheiros
E2 comprou 398 MW por 1768 dinheiros
E3 comprou 240 MW por 1210 dinheiros
E4 comprou 433 MW por 2327 dinheiros
E6 comprou 297 MW por 1499 dinheiros
E7 comprou 232 MW por 1209 dinheiros
E9 comprou 558 MW por 2983 dinheiros
E10 comprou 495 MW por 2259 dinheiros
E11 comprou 310 MW por 1381 dinheiros
E13 comprou 213 MW por 1115 dinheiros
E14 comprou 346 MW por 1552 dinheiros

E15 comprou 385 MW por 2023 dinheiros
E16 comprou 240 MW por 1234 dinheiros
E17 comprou 483 MW por 2828 dinheiros
E18 comprou 487 MW por 2617 dinheiros
E20 comprou 358 MW por 1847 dinheiros
E21 comprou 467 MW por 2038 dinheiros
E22 comprou 363 MW por 2007 dinheiros
E23 comprou 279 MW por 1311 dinheiros
E24 comprou 589 MW por 3164 dinheiros
E25 comprou 476 MW por 2480 dinheiros
Totalizando 40348 dinheiros e 7962 MW vendidos

Resultado algoritmo guloso com primeira estratégia:

E1 comprou 313 MW por 1496 dinheiros
E2 comprou 398 MW por 1768 dinheiros
E4 comprou 433 MW por 2327 dinheiros
E6 comprou 297 MW por 1499 dinheiros
E8 comprou 614 MW por 2342 dinheiros
E9 comprou 558 MW por 2983 dinheiros
E10 comprou 495 MW por 2259 dinheiros
E13 comprou 213 MW por 1115 dinheiros
E14 comprou 346 MW por 1552 dinheiros
E15 comprou 385 MW por 2023 dinheiros
E17 comprou 483 MW por 2828 dinheiros
E18 comprou 487 MW por 2617 dinheiros
E19 comprou 709 MW por 2328 dinheiros
E20 comprou 358 MW por 1847 dinheiros
E21 comprou 467 MW por 2038 dinheiros
E22 comprou 363 MW por 2007 dinheiros
E24 comprou 589 MW por 3164 dinheiros
E25 comprou 476 MW por 2480 dinheiros
Totalizando 38673 dinheiros e 7984 MW vendidos

Resultado algoritmo guloso com segunda estratégia:

E1 comprou 313 MW por 1496 dinheiros
E2 comprou 398 MW por 1768 dinheiros
E3 comprou 240 MW por 1210 dinheiros
E4 comprou 433 MW por 2327 dinheiros
E6 comprou 297 MW por 1499 dinheiros
E7 comprou 232 MW por 1209 dinheiros
E9 comprou 558 MW por 2983 dinheiros
E10 comprou 495 MW por 2259 dinheiros
E11 comprou 310 MW por 1381 dinheiros
E12 comprou 213 MW por 961 dinheiros
E13 comprou 213 MW por 1115 dinheiros
E14 comprou 346 MW por 1552 dinheiros
E15 comprou 385 MW por 2023 dinheiros
E16 comprou 240 MW por 1234 dinheiros
E17 comprou 483 MW por 2828 dinheiros
E18 comprou 487 MW por 2617 dinheiros
E20 comprou 358 MW por 1847 dinheiros
E22 comprou 363 MW por 2007 dinheiros
E23 comprou 279 MW por 1311 dinheiros
E24 comprou 589 MW por 3164 dinheiros
E25 comprou 476 MW por 2480 dinheiros
Totalizando 39271 dinheiros e 7708 MW vendidos

Esses dados foram obtidos através dos dados de testes enviados pelo professor via Canvas no dia 20 de Junho de 2024

8000;E1;430;1043;E2;428;1188;E3;410;1565;E4;385;1333;E5;399;1214;E6;382;1498;E7;416;1540;E8;436;1172;E9;416;1386;E10;423;1097;E11;400;1463;E12;406;1353;E13;403;1568;E14;390;1228;E15;387;1542;E16;390;1206;E17;430;1175;E18;397;1492;E19;392;1293;E20;393;1533;E21;439;1149;E22;403;1277;E23;415;1624;E24;387;1280;E25;417;1330;

e outro conjunto de dados:

8000;E1;313;1496;E2;398;1768;E3;240;1210;E4;433;2327;E5;

301;1263;E6;297;1499;E7;232;1209;E8;614;2342;E9;558;2983;E10;495;2259;E11;310;1381;E12;213;961;E13;213;1115;E14;346;1552;E15;385;2023;E16;240;1234;E17;483;2828;E18;487;2617;E19;709;2328;E20;358;1847;E21;467;2038;E22;363;2007;E23;279;1311;E24;589;3164;E25;476;2480

3.3 Discussões Sobre Vantagens e Desvantagens de Cada Abordagem

3.3.1 Backtracking:

Vantagens do Backtracking

- Exploração Exaustiva: O Backtracking faz uma busca exaustiva e verifica se cada elemento é uma possível solução. Se é, continua e verifica se é uma solução completa; se não, ele retorna e busca outro candidato (poda).
- Flexibilidade: Pode ser adaptado para incluir podas e otimizações adicionais, como heurísticas que aceleram a busca por soluções viáveis.
- Simplicidade: A abordagem recursiva é intuitiva e direta de implementar para problemas de combinação e otimização.

Desvantagens do Backtracking

- Complexidade Temporal: A abordagem de força bruta pode ser ineficiente para grandes conjuntos de ofertas devido ao crescimento exponencial do espaço de busca.
- Consumo de Memória: A recursão e a necessidade de armazenar múltiplas combinações podem resultar em alto consumo de memória.

3.3.2 Divisão e Conquista:

A técnica de Divisão e Conquista, embora frequentemente associada à resolução de problemas matemáticos, demonstra grande versatilidade ao ser aplicada a desafios computacionais complexos, como o problema do leilão de energia. Sua essência reside em decompor um problema em subproblemas menores e independentes, resolvê-los recursivamente e combinar suas soluções para obter a

resposta final.

A decomposição do problema em subproblemas menores e mais simples permite atacá-lo de forma estratégica, evitando a análise exaustiva de todas as combinações possíveis, como ocorre em algoritmos de força bruta. Essa abordagem reduz significativamente o espaço de busca e, consequentemente, o tempo de execução, especialmente em problemas com alto grau de complexidade.

A Divisão e Conquista se destaca pela escalabilidade. Diferentemente de algoritmos com complexidade exponencial, que se tornam inviáveis para conjuntos de dados maiores, a Divisão e Conquista mantém um crescimento do tempo de execução mais controlado, tornando-a adequada para lidar com cenários reais, onde o volume de dados pode ser considerável.

A implementação da Divisão e Conquista se beneficia enormemente da técnica de memorização. Ao armazenar e reutilizar resultados de subproblemas já calculados, a memorização elimina cálculos redundantes, impulsionando a eficiência do algoritmo, especialmente em problemas com subproblemas sobrepostos.

Sua natureza independente dos subproblemas na Divisão e Conquista abre portas para a paralelização. Em arquiteturas de hardware com múltiplos núcleos de processamento, os subproblemas podem ser resolvidos simultaneamente, reduzindo drasticamente o tempo total de execução e potencializando a eficiência da solução.

Porém, todos os algoritmos têm também seus pontos negativos. No caso a Divisão e Conquista exige uma análise mais profunda do problema para identificar a estrutura recursiva adequada e garantir que a decomposição em subproblemas seja efetivamente vantajosa. A implementação, por sua vez, pode ser mais complexa do que algoritmos iterativos, demandando atenção especial para evitar erros de lógica na recursão.

Sem contar também que a Divisão e Conquista, por natureza, se baseia em chamadas recursivas, o que pode gerar overhead de gerenciamento de memória, especialmente em problemas com profundidade de recursão muito grande. Em

cenários com recursos de memória limitados, essa sobrecarga pode impactar o desempenho do algoritmo.

3.3.3 Guloso - Estratégia Gulosa 1:

A estratégia gulosa e o algoritmo de ordenação por maior valor oferecem várias vantagens significativas. A simplicidade e a eficiência são alguns dos principais benefícios. A estratégia gulosa é fácil de implementar e geralmente muito eficiente em termos de tempo de execução. Isso ocorre porque o algoritmo toma decisões rápidas e diretas, simplificando o processo de resolução do problema. Essa eficiência torna o algoritmo guloso uma escolha atraente para muitos problemas práticos, onde encontrar uma solução rápida é crucial.

Outra vantagem importante da estratégia gulosa é a maximização do retorno financeiro para cada unidade de capacidade utilizada. Ao ordenar as ofertas pelo maior valor total, o algoritmo garante que as ofertas mais valiosas sejam consideradas primeiro. Dessa forma, a capacidade disponível é utilizada de maneira eficiente, resultando em um aproveitamento otimizado dos recursos. Isso é particularmente útil em contextos onde os recursos são limitados e é essencial obter o máximo benefício possível de cada unidade disponível.

No entanto, apesar dessas vantagens, é importante reconhecer as limitações da abordagem gulosa. Uma das principais desvantagens é que, embora o algoritmo guloso seja eficiente, ele pode não resultar na melhor solução global. Isso ocorre porque a estratégia gulosa toma decisões baseadas em informações locais e imediatas, sem considerar o impacto dessas decisões no contexto global. Em alguns casos, uma combinação de ofertas menores pode resultar em um valor total maior, mas a estratégia gulosa pode não detectar isso, pois foca apenas nas ofertas de maior valor individual.

Além disso, em problemas onde a relação entre valor e capacidade é complexa ou não linear, a estratégia gulosa pode levar a soluções subótimas. O algoritmo não revisa decisões anteriores para garantir a otimalidade global, o que pode ser uma limitação significativa em cenários mais complexos. Em tais casos, a abordagem pode não identificar combinações de ofertas que resultariam em um valor total maior, devido à

sua ênfase na priorização de ofertas individuais com maior valor.

3.3.3-2 Algoritmo Guloso - Estratégia 2

A estratégia gulosa de ordenação por valor por megawatt oferece várias vantagens notáveis. Ao ordenar as ofertas pelo maior valor por megawatt, o algoritmo garante que cada unidade de capacidade seja utilizada da forma mais eficiente possível em termos de valor monetário. Essa abordagem facilita a tomada de decisões locais ótimas, que frequentemente levam a uma boa solução global. Isso significa que, ao focar no maior valor por megawatt, o algoritmo tende a maximizar o retorno financeiro, resultando em uma utilização eficiente dos recursos disponíveis.

Por outro lado, essa estratégia também apresenta desvantagens. Se a distribuição dos valores por megawatt não seguir uma tendência clara, ordenar apenas por esse critério pode não resultar na melhor solução global. Além disso, em cenários onde a relação entre valor e capacidade não é linear, a abordagem gulosa pode levar a soluções subótimas, pois não revisa decisões anteriores para garantir a otimalidade global. Em tais casos, a estratégia pode não identificar combinações de ofertas que resultariam em um valor total maior, devido à sua ênfase na priorização de ofertas individuais com maior valor por megawatt.

3.3.4 Programação Dinâmica:

A programação dinâmica consegue chegar ao resultado ótimo com o menor tempo de execução, quando comparado com os demais dados. Entretanto, há uma maior utilização de memória do computador, por ter que armazenar os dados da tabela por completo.

Ou seja, caso o tempo de execução seja o maior dos problemas, entretanto, haja grandes quantidades de memória disponíveis para a utilização do problema, a

programação dinâmica se coloca como uma excelente possibilidade de solução para o problema que não interfira no resultado obtido.

4 CONSIDERAÇÕES FINAIS

Em resumo, cada algoritmo possui suas vantagens e desvantagens, correspondentes tanto a sua dificuldade de implementação, garantia de obtenção do resultado ótimo e tempo de execução. Para o problema analisado, com o aumento considerável dos dados, para garantir o resultado ótimo, a programação dinâmica apresentou os melhores resultados.

Entretanto, caso o resultado não possua a necessidade de ser ótimo, mas precise de alta velocidade de execução devido a uma quantidade massiva de dados, o algoritmo guloso possui a melhor velocidade e pode ser uma boa solução.

Por fim, caso a utilização de memória seja um problema, mas seja necessário obter um resultado ótimo, o divisão e conquista mostrou um melhor tempo de execução quando comparado com o backtracking e não necessita da grande quantidade de armazenamento de dados com ocorre com a programação dinâmica.