Dados de execução

- Responsável: Guilherme Lage da Costa
- Matrícula: 792939
- Data de análise: 14 de maio de 2024
- JDK: Java 17
- Processador: AMD Ryzen 5 3600, 4.2 Ghz, 6 cores e 12 threads, 32mb de cache
- RAM: 16GB, 3000Ghz
- Sistema Operacional: Windows 11
- IDE: IntelliJ Ultimate

Sobre o algoritmo

O ***backtracking*** é uma técnica de projetos de algoritmos também conhecida como *retrocesso* ou *tentativa e erro*. Esse algoritmo é um refinamento da busca por força bruta, na qual algumas soluções podem ser descartadas sem que ao menos sejam examinadas. Neste algoritmo, são testadas, metodicamente, várias sequências de decisões até encontrar uma que seja aceitável, ou as descarta até encontrar a melhor solução possível.

De modo geral, esse algoritmo segue inicialmente o padrão de uma busca em profundidade, ou seja, uma árvore é percorrida sistematicamente. Quando essa busca falha, ou é encontrado uma folha da árvore (nós sem filhos), entra em funcionamento o mecanismo que da nome ao algoritmo de ***backtracking***, fazendo com que o algoritmo retorne pelo mesmo caminho já percorrido, de modo a buscar soluções alternativas que atendam aos critérios pré-definidos.

Detalhamento do código

Algoritmo genérico

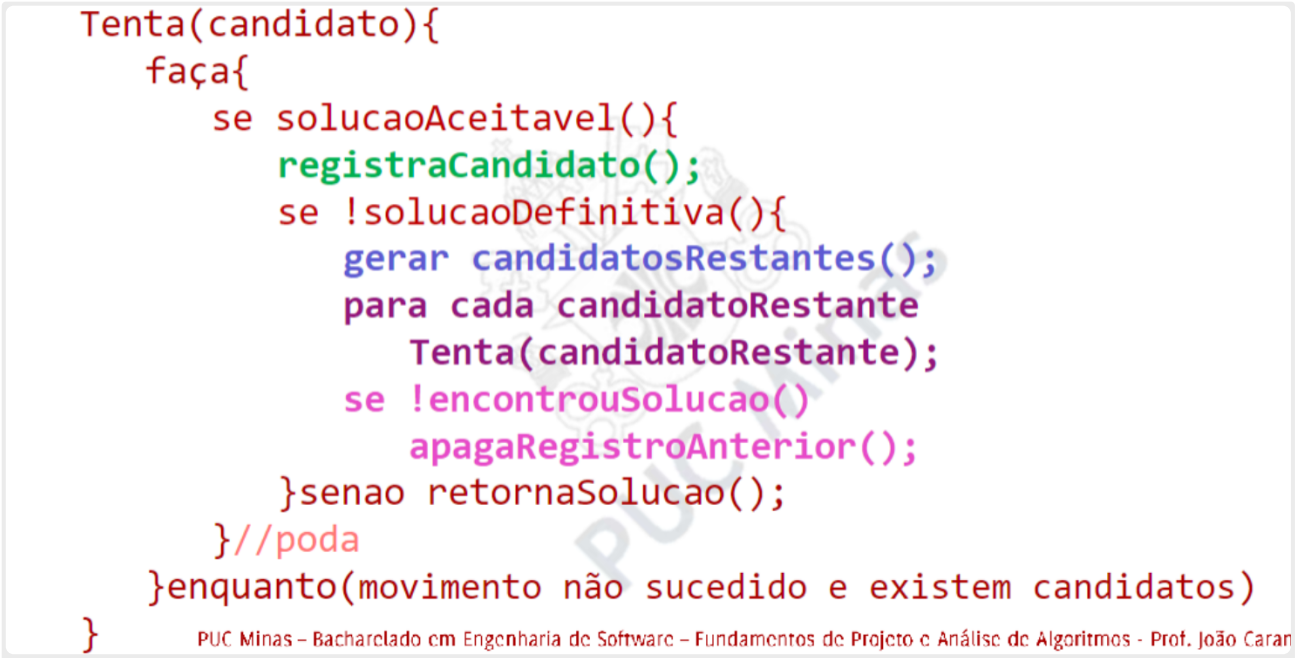


Figura 01 - Algoritmo genérico disponibilizado pelo Prof. João Caram

O algoritmo apresentado acima foi disponibilizado pelo Prof. João Caram no material Backtracking e problema do leilão de energia (TP).

Conforme pode ser observado na **Figura 01** acima, o algoritmo é executado apenas caso o último movimento não tenha sido bem sucedido ou ainda existam candidatos disponíveis. Caso a resposta seja positiva para ambas as verificações, o algoritmo é executado e faz uma primeira validação, verificando se a solução atual é aceitável, se ela for, registra-se o candidato atual à lista de candidatos analisados, se não for, o algoritmo segue sem considerar este candidato, indicado pela linha "//poda".

Mais adiante, é verificado se a solução atual **não** é a definitiva, ou seja, se foi encontrado o melhor caso por exemplo, caso não seja, registra-se os candidatos restantes e, para cada candidato, será executado, recursivamente, todo esse método novamente, caso ela seja a definitiva, é retornado esse valor e o método é finalizado. Quando não se há mais candidatos restantes, é verificado se a solução **não** foi encontrada, se não foi, descarta-se o registro anterior.

Algoritmo implementado

```
1  public void backtracking(
2              MelhorResultado melhorResultado, List<Lance> todosLances,
   List<Lance> lancesSelecionados, int indice, int lucroAtual) {
3
4          int qtdeSelecionada = lancesSelecionados.stream()
```

```
                .mapToInt(Lance::quantidade).sum();

        if (qtdeSelecionada > melhorResultado.getProdutora().quantidadeDisponivel())
            return;

        if (indice == todosLances.size()) {
            if (lucroAtual > melhorResultado.getLucroMaximizado()) {
                melhorResultado.setLucroMaximizado(lucroAtual);
                melhorResultado.setLancesSelecionados(new ArrayList<>(lancesSelecionados));
            }
            return;
        }

        Lance lanceAnalisado = todosLances.get(indice);

        if (!isNull(lanceAnalisado)) {
            lancesSelecionados.add(lanceAnalisado);
            executar(melhorResultado, todosLances, lancesSelecionados, indice + UM, lucroAtual + lanceAnalisado.valor());
            lancesSelecionados.remove(lancesSelecionados.size() - UM);
        }
        executar(melhorResultado, todosLances, lancesSelecionados, indice + UM, lucroAtual);
    }
```

No algoritmo implementado, a função responsável por executar o método de ***backtracking*** recebe cinco parâmetros, sendo eles:

- melhorResultado: `MelhorResultado`;
- todosLances: `List<Lance>`;
- lancesSelecionados: `List<Lance>`;
- indice: `int`;
- lucroAtual: `int`.

o parâmetro de `melhorResultado` possui informações sobre a empresa produtora `produtora` e compradora `compradoras`, um contador de tempo de execução `contador`, lista dos lances selecionados `lancesSelecionados`, lucro maximizado `lucroMaximizado` e a quantidade disponibilizada pela empresa produtora `quantidadeVendida`. O parâmetro `todosLances` relaciona todos os lances feitos por todas as empresas compradoras, enquanto `lancesSelecionados` relaciona todos os lances selecionados que compõem o

maior lucro possível `lucroMaximizado`. O parâmetro `indice` indica qual o índice da lista de lances `todosLances` que será analisado para verificar se cabe no melhor resultado possível, que por sua vez é armazenado no parâmetro `lucroAtual`.

O algoritmo se inicia quantificando qual é a quantidade total dos lances atualmente selecionados (*linha 4* - em megawatts) e o armazena na variável `qtdeSelecionada`, essa variável é utilizada para verificar, na *linha 7*, se essa quantidade é superior à quantidade disponível de venda. Caso a quantidade selecionada seja superior à de venda, essa solução não é aceitável e, portanto, é desconsiderada no ***return*** da *linha 8*.

Posteriormente, na condicional da *linha 10*, é verificado se o índice passado como parâmetro é igual à quantidade de lances total, se essa validação for verdadeira, significa que não há mais lances para avaliar nessa iteração, uma vez que o índice chegou ao último lance. Em seguida (*linha 11*), é verificado se o `lucroAtual` (soma do valor de todos os `lancesSelecionados`) é maior que o valor do maior lucro registrado, armazenado no atributo `lucroMaximizado` do `melhorResultado`, se for maior, atualiza-se o lucro maximizado e a lista de lances selecionados nas *linhas 12 e 13*. Se o lucro atual não for maior do que o lucro maximizado, essa solução é desconsiderada no **return** da *linha 15*.

Na *linha 18* é selecionado um novo lance que será analisado, caso ele não seja `null` (*linha 20*), prossegue para uma nova sequência de instrução. A validação se o `lanceAnalisado` não é nulo é importante pois, como o algoritmo executa recursivamente sobre a lista na posição índice + 1, pode-se chegar em um ponto em que o próximo valor é nulo, como é o caso quando o lance analisado é o último da lista, por exemplo.

Se o lance não for nulo, ele é adicionado á lista de `lancesSelecionados` (*linha 21*) e chama-se a função de **backtracking** recursivamente sobre essa nova lista de lances (*linha 22*). Quando a execução chegar ao fim, esse lance é removido da lista na *linha 23* e, em seguida, chamado uma nova execução do algoritmo na próxima posição do índice, *linha 25*.

Massa de testes utilizada

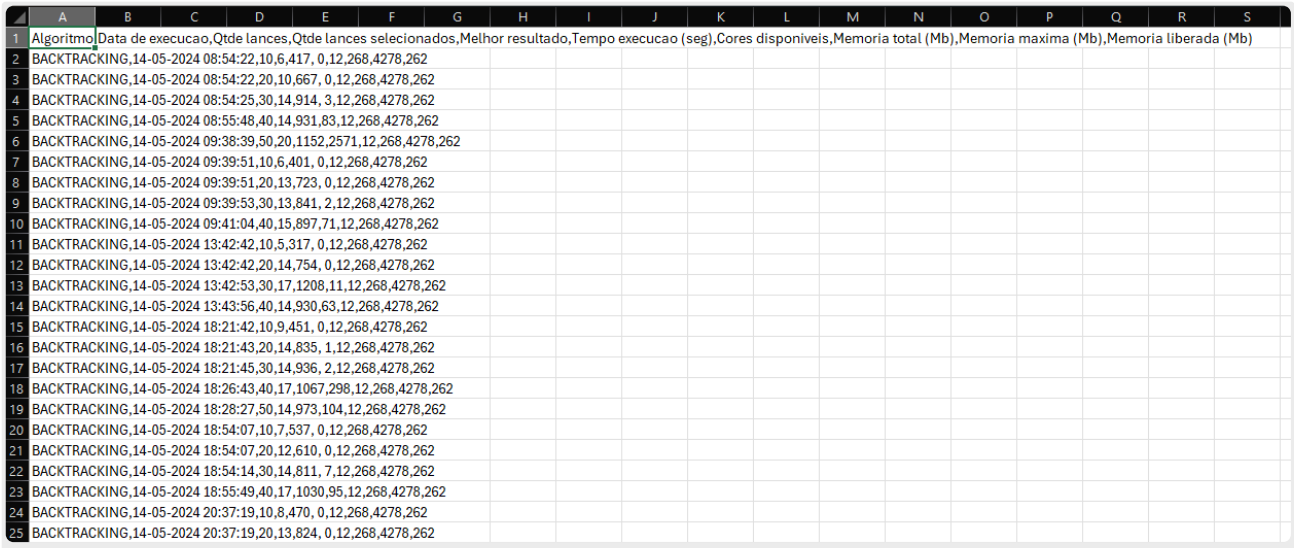
A massa de testes utilizada seguiu os seguintes parâmetros:

- **Quantidade *mínima* p/ compradora = 10** → indica a quantidade mínima que uma determinada compradora poderia solicitar em um lote;

- **Quantidade *máxima* p/ compradora = 100** → indica a quantidade máxima que uma determinada compradora poderia solicitar em um lote;
- **Quantidade disponível pela produtora = 200** → indica a quantidade total (lote total) que a empresa produtora possui, ou seja, que disponibiliza para leilão;
- **Quantidade máxima de lances p/ compradora = 1** → indica a quantidade máxima de lances que cada compradora poderia fazer;
- **Quantidades de compradoras = [10, 20, 30, 40, 50]** → foram executados testes sistemáticos com cada uma das quantidades de compradoras. Ou seja, foram simulados cenários em que haveriam 10 compradoras fazendo lances no lote ofertado pela produtora, depois, analisado o cenário considerando 20 compradoras, e assim por diante, de modo a comparar o impacto que esses cenários causariam nos testes e até onde seria possível executar os algorítimos.

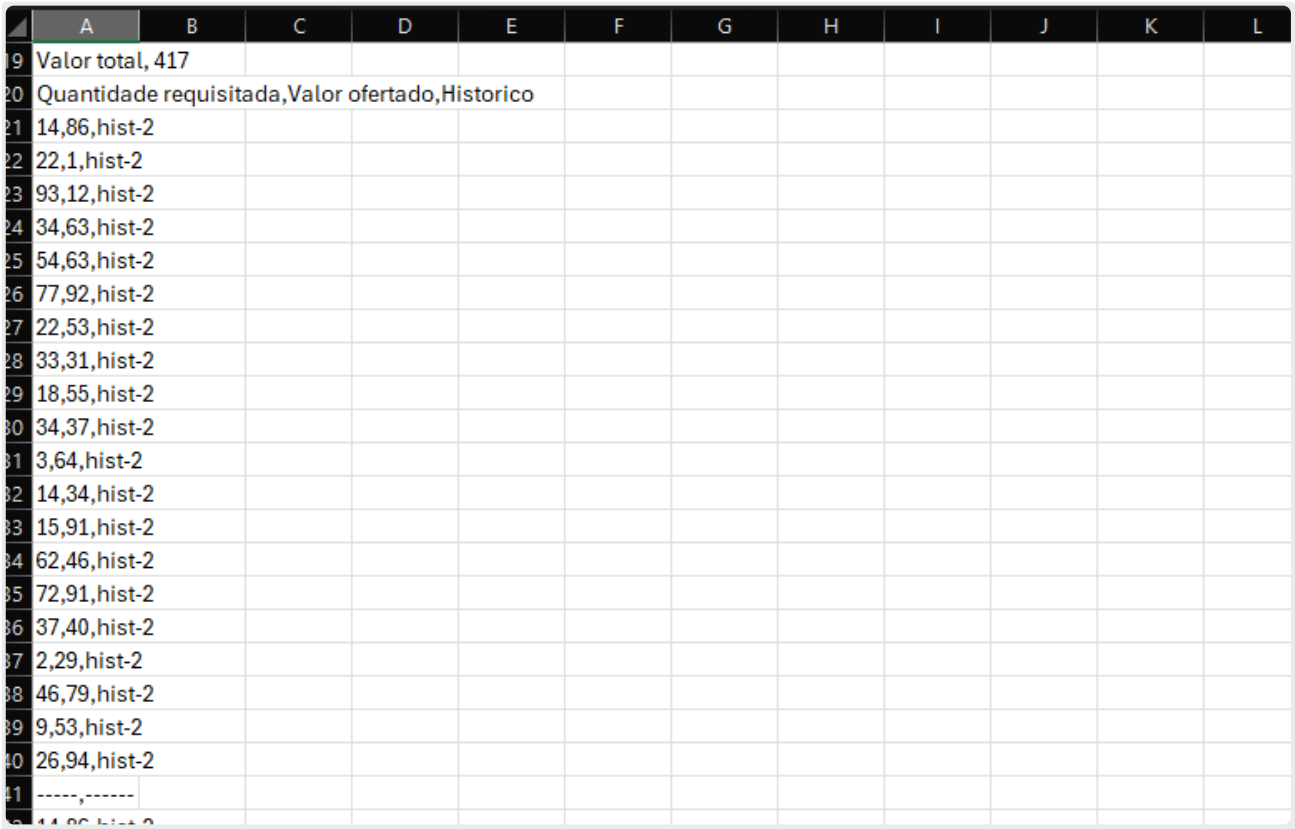
Para a criação de lances, foram utilizados valores aleatórios entre os limites de quantidade mínima (10) e máxima (100) indicados. Isso foi feito para que cenários distintos fossem criados e avaliar como o algoritmo se comportaria em cada um deles.

Os resultados gerados após cada execução do algoritmo foram armazenados automaticamente em duas planilhas: `exec-log.xls` e `hist-log.xls`. O primeiro log guarda dados gerais da execução como o tempo despendido, algoritmo utilizado, recursos computacionais disponíveis, dentre outros, o segundo log guarda os dados sobre os lances que foram feitos e os que foram escolhidos para combinar o melhor resultado.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Algoritmo	Data de execucao	Qtde lances	Qtde lances selecionados	Melhor resultado	Tempo execucao (seg)	Cores disponiveis	Memoria total (Mb)	Memoria maxima (Mb)	Memoria liberada (Mb)									
2	BACKTRACKING	14-05-2024 08:54:22	10	6	417, 0	12	268	4278	262										
3	BACKTRACKING	14-05-2024 08:54:22	20	10	667, 0	12	268	4278	262										
4	BACKTRACKING	14-05-2024 08:54:25	30	14	914, 3	12	268	4278	262										
5	BACKTRACKING	14-05-2024 08:55:48	40	14	931	83	12	268	4278	262									
6	BACKTRACKING	14-05-2024 09:38:39	50	20	1152	2571	12	268	4278	262									
7	BACKTRACKING	14-05-2024 09:39:51	10	6	401, 0	12	268	4278	262										
8	BACKTRACKING	14-05-2024 09:39:51	20	13	723, 0	12	268	4278	262										
9	BACKTRACKING	14-05-2024 09:39:53	30	13	841, 2	12	268	4278	262										
10	BACKTRACKING	14-05-2024 09:41:04	40	15	897	71	12	268	4278	262									
11	BACKTRACKING	14-05-2024 13:42:42	10	5	317, 0	12	268	4278	262										
12	BACKTRACKING	14-05-2024 13:42:42	20	14	754, 0	12	268	4278	262										
13	BACKTRACKING	14-05-2024 13:42:53	30	17	1208	11	12	268	4278	262									
14	BACKTRACKING	14-05-2024 13:43:56	40	14	930	63	12	268	4278	262									
15	BACKTRACKING	14-05-2024 18:21:42	10	9	451, 0	12	268	4278	262										
16	BACKTRACKING	14-05-2024 18:21:43	20	14	835, 1	12	268	4278	262										
17	BACKTRACKING	14-05-2024 18:21:45	30	14	936, 2	12	268	4278	262										
18	BACKTRACKING	14-05-2024 18:26:43	40	17	1067	298	12	268	4278	262									
19	BACKTRACKING	14-05-2024 18:28:27	50	14	973	104	12	268	4278	262									
20	BACKTRACKING	14-05-2024 18:54:07	10	7	537, 0	12	268	4278	262										
21	BACKTRACKING	14-05-2024 18:54:07	20	12	610, 0	12	268	4278	262										
22	BACKTRACKING	14-05-2024 18:54:14	30	14	811, 7	12	268	4278	262										
23	BACKTRACKING	14-05-2024 18:55:49	40	17	1030	95	12	268	4278	262									
24	BACKTRACKING	14-05-2024 20:37:19	10	8	470, 0	12	268	4278	262										
25	BACKTRACKING	14-05-2024 20:37:19	20	13	824, 0	12	268	4278	262										

Figura 02 - Trecho do arquivo de log `exec-log.xls`



	A	B	C	D	E	F	G	H	I	J	K	L
19	Valor total, 417											
20	Quantidade requisitada,Valor ofertado,Historico											
21	14,86,hist-2											
22	22,1,hist-2											
23	93,12,hist-2											
24	34,63,hist-2											
25	54,63,hist-2											
26	77,92,hist-2											
27	22,53,hist-2											
28	33,31,hist-2											
29	18,55,hist-2											
30	34,37,hist-2											
31	3,64,hist-2											
32	14,34,hist-2											
33	15,91,hist-2											
34	62,46,hist-2											
35	72,91,hist-2											
36	37,40,hist-2											
37	2,29,hist-2											
38	46,79,hist-2											
39	9,53,hist-2											
40	26,94,hist-2											
41	-----,------											

Figura 03 - Trecho do arquivo de log `hist-log.xls`

O arquivo de análises que compila as execuções realizadas pode ser verificado no caminho: Analises backtracking

Análise do resultado

Resultados obtidos

Para o algoritmo de `Backtracking`, foram realizados no total 42 execuções considerando os cenários de 10, 20, 30, 40 e 50 lances previamente citados, os tempos de execução obtidos em cada um desses testes são apresentados na tabela a seguir.

Tempo médio de execução (seg)					
Execução / Qtde lances	**10**	**20**	**30**	**40**	**50**
1	0	0	3	83	2571
2	0	0	2	71	104
3	0	0	11	63	
4	0	1	2	298	
5	0	0	7	95	
6	0	0	3	11	

Tempo médio de execução (seg)					
7	0	0	2	428	
8	0	0	16	171	
9	0	0	19	155	
10	0	0	5	45	

A tabela acima deve ser interpretada da seguinte forma: a primeira execução, representada pela linha com o número 1, obteve tempo de execução de 0 seg para os cenários com 10 e 20 lances, para o cenário com 30 lances foi gasto 3 seg, 83 seg para o cenário com 40 lances e 2571 seg para o cenário com 50 lances. A segunda execução, por sua vez, representado pela linha com número 2, demandou 0 seg para os cenários com 10 e 20 lances, 2 seg para o cenário com 30 lances, 71 seg para o cenário com 40 lances e 104 seg para o cenário com 50 lances, e assim em diante.

Conforme pode ser observado, na execução 7 do algoritmo, no cenário com 40 lances, o tempo de execução foi muito superior à média dos demais (428 seg), assim como na execução 1 no cenário com 50 lances (2571 seg). Por conta dessas variações, foi feita uma análise de *outliers* sobre os valores obtidos, a fim de se determinar uma média mais representativa, que não fosse deformada pelos resultados discrepantes dos demais. A análise desses valores que destoaram dos demais será feita no tópico Considerações sobre os resultados obtidos.

Com base nesses resultados, foram obtidos os seguintes tempos médios de execução.

Qtde lances	10	20	30	40	50
Média original (seg)	0	0	7	142	1338
Média - outliers (seg)	0		7	110	
Q1	0	0	2	58,5	0
Q3	0	0	12,25	202,75	0
IQR	0	0	10,25	144,25	0
Limite inferior	0	0,1	-8,375	-74,375	1337,5
Limite superior	0	0,1	22,375	358,375	1337,5

Legenda:

- Média original (seg): média original obtida, considerando *outliers*
- Média - outliers (seg): nova média calculada, desconsiderando *outliers*;

- Q1: primeiro quartil;
- Q3: terceiro quartil;
- IQR: *Interquartile Range* → intervalo comportado entre o primeiro e terceiro quartil;
- Limite inferior: limite inferior comportado, valores abaixo do limite inferior são considerados *outliers*;
- Limite superior: limite superior comportado, valores acima do limite superior são considerados *outliers*.

Considerações sobre os resultados obtidos

Primeiramente, é importante pontuar que os resultados que apresentaram tempo de execução igual a 0 seg indicam que a sua execução demandou menos de 1 seg, de modo que o seu tempo é praticamente desprezível. Esse cenário foi observado para os casos em que foi executado o algoritmo com 10 e 20 lances.

Com relação ao cenário com 30 lances, observamos que o tempo médio de execução foi de 7 segundos e não foi observado nenhum *outlier* dentro da amostragem realizada. De modo geral, para esse cenário, os dados foram bastantes consistentes.

Para o cenário com 40 lances, verificamos um *outlier* na execução de número 7, que apresentou tempo de execução igual a 428 seg. Analisando esse caso em específico (Gráfico 1), podemos observar que houve uma concentração no trecho mais à esquerda do gráfico de dispersão, o que mostra que os lances com mais valor estavam geralmente relacionados às menores quantidades, conforme demonstrado pela linha de tendência em vermelho no gráfico. Como resultado, observamos que a maioria dos lances selecionados possuíam uma quantidade relacionada mais baixa, o que fez com que fossem analisados muitos outros lances para que o a quantidade disponibilizada (200) fosse atingida.

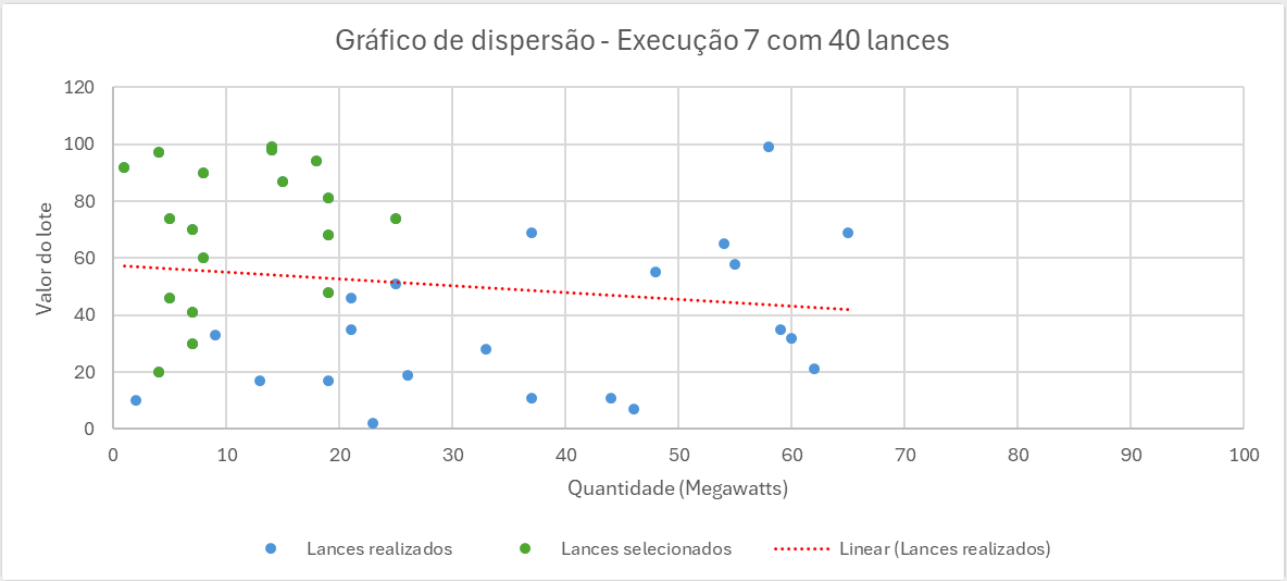


Gráfico 01 - Execução 7 com 40 lances

Se formos comparar com um cenário com tempo de execução mais próximo à média, como é o caso da primeira execução que apresentou tempo igual a 83 seg, verificamos um cenário distinto. Neste cenário, a maior concentração de lances com valores mais altos estão relacionados aos lances com quantidades mais altas, o que explica o porque da execução 7 demandar mais tempo.

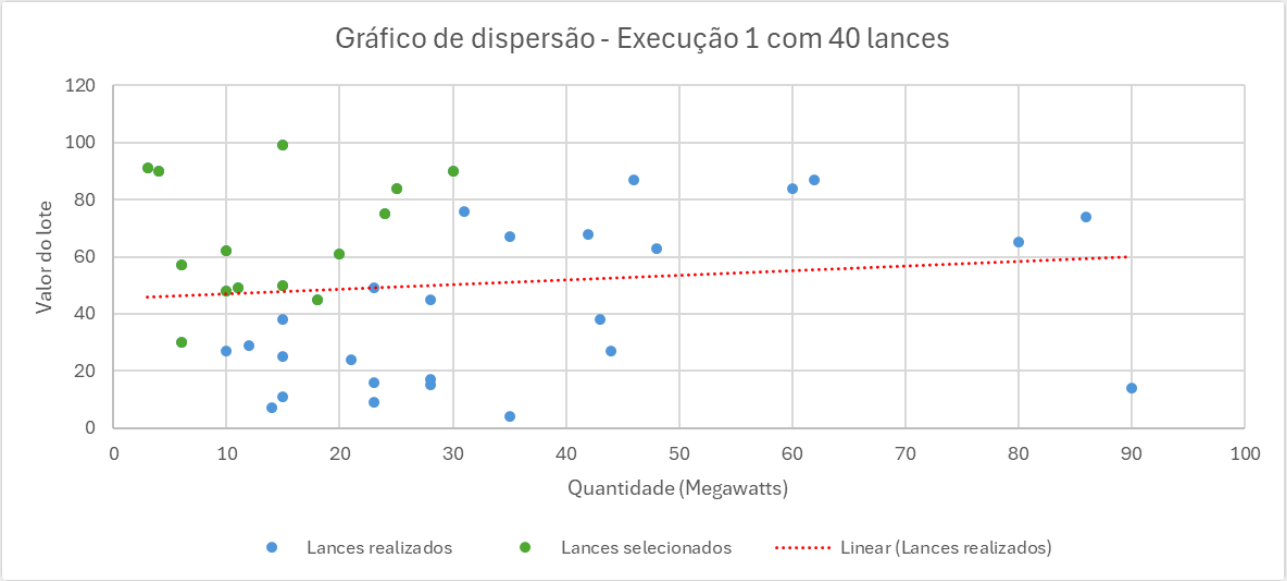


Gráfico 02 - Execução 1 com 40 lances

Para o cenário com 50 lances, não foi considerado que houve *outliers*, uma vez que apenas duas execuções foram concluídas, as demais executaram por um tempo superior