UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE CIÊNCIAS APLICADAS

PO240 – Introdução à Meta-Heurística

Tema: Problema de Corte

Orientadores:

Prof. Dr. Carla Ghidini

Guilherme Lima Correa - 173811

Limeira, 17 de julho de 2020

Resumo

O problema de corte aparece em várias industrias, tais como as de papel, colchões, vidro, plástico, metalúrgica, entre outras. O processo que ocorre nestas consiste em produzir peças menores (itens) para atender uma determinada demanda a partir do corte de peças maiores (objetos) que estão disponíveis em estoque, sempre tendo como objetivo otimizar uma função objetivo. Neste trabalho é apresentado um estudo sobre o problema de corte unidimensional, o qual a quantidade de objetos em estoque é ilimitada e deseja-se utilizar a menor quantidade de objetos para atender a demanda. Para a resolução deste problema foram desenvolvidos três métodos. O primeiro consiste na obtenção de uma solução exata através do Solver gratuito OR-TOOLS. A segunda uma heurística construtiva denominada FFD (First-Fit-Decreasing). E por último, uma meta-heurística evolutiva, também conhecida como Algoritmo Genético (AG). Para as três versões do método de resolução proposto foram realizados testes computacionais com exemplares da literatura e exemplares gerado através de uma destruição normal

Palavras-chave: problema de corte unidimensional, solução exata, heurística construtiva, meta-heurística**.**

1. Introdução

No mundo capitalista, onde a competitividade entre as empresas gera uma busca por lucro e redução de custos, há vários setores na indústria que almejam alcançar estes objetivos através da venda dos itens produzidos. Portanto, minimizar a quantidade de matéria prima utilizada é essencial.

Empresas que trabalham com o corte, isto é, objetos (peças grandes) de tamanhos padronizados são cortados em itens (peças menores) de tamanhos variados (ARENALES, 2011), como por exemplo, barras de aço, de bobinas de papel, rolos de filme, entre outras, produzem itens a partir do corte de objetos que estão disponíveis no estoque, sempre buscando identificar quais padrões otimizam os lucros. Esse processo de corte gera perdas de material indesejáveis, portanto, surge o problema de otimização que consiste em definir quais padrões de corte utilizar de modo que os itens sejam produzidos nas quantidades solicitadas pelos clientes, de forma que a quantidade de material gasta seja mínima

A dimensão do corte é relevante ao problema, neste trabalho, analisa-se o problema de corte unidimensional com estoque ilimitado, sendo que a função objetivo é reduzir quantidade de objetos cortados, consequentemente reduzir os custos na produção.

Este tipo de problema e suas variações são considerados problemas NP-difíceis, isto porque não existem algoritmos que resolvam esses tipos de problemas em tempo polinomial.

Neste trabalho propõe-se um algoritmo exatos para solucionar o problema, além de outros dois métodos heurísticos de resolução. O algoritmo exato leva a uma solução ótima, porém, tem um alto custo computacional, consequentemente o tempo para se alcançar a solução em situações práticas não é viável. Essa dificuldade no tempo de execução de algoritmos exatos justifica a aplicação de métodos heurísticos. Estes métodos não garantem soluções ótimas, porém, garantem soluções sub-ótimas em menor tempo computacional. Primeiramente, optou-se pela heurística construtiva FFD, a qual se constrói uma solução conforme um conjunto de regras. E por último, uma meta-heurística. Estas são, na verdade, métodos heurísticos que resolvem os problemas de otimização de forma genérica, e são normalmente utilizadas para resolver problemas em que não conhecemos algoritmos eficientes. Utilizam combinações de escolhas aleatórias e conhecimento histórico (resultados anteriores adquiridos pelo método) para se guiarem dentro de vizinhanças no espaço de busca. Optou-se pela meta-heurística denominada algoritmo genético.

O objetivo deste trabalho é propor e analisar métodos de solução para o problema de corte unidimensional.

2.1 Solver

No problema de corte unidimensional apenas uma dimensão é relevante no processo. Sendo assim, para modelar matematicamente este problema partimos do pressuposto que deseja-se cortar barras disponíveis de uma tamanho único L para a produção de m tipos de itens (barras menores) com tamanho $l1, l2, l3, \dots, lm$ em quantidades variadas $(d1, d2, d3, \dots, dm)$. Sabendo a demanda de cada item e o tamanho dele, pode-se determinar diversos padrões de corte, isto é, uma maneira única de se cortar o objeto. Para cada padrão de corte (j) é associado um vetor $a_j = (a_{1j}, a_{2j}, \dots, a_{mj})$, onde aij demonstra o número de itens do tipo i no padrão de corte j. Consideramos a fim de

exemplificação uma objeto de tamanho 120 cm, que precisará ser cortado em três itens $l1 = 30\ cm, l2 = 42\ cm\ e\ l3 = 45\ cm$. A figura XXXX apresenta alguns dos padrões de cortes possíveis.

Figura 1:Exemplo do Problema de corte unidimensional

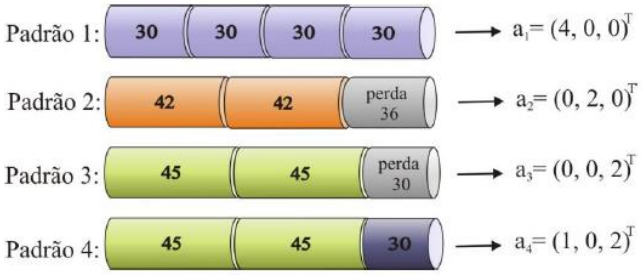


Figura 3.1. Padrões de corte e vetores associados.

Para definir um padrão de corte é necessário satisfazer o seguinte sistema:

$$l_1\alpha_1 + l_2\alpha_2 + \cdots + l_m\alpha_m \leq L \qquad (1)$$

$$\alpha_1 \geq 0, \alpha_2 \geq 0, \ldots, \alpha_m \geq 0\ e\ inteiros \qquad (2)$$

Este sistema representa que o somatório da quantidade de cada item multiplicado pelo seu tamanho tem que ser menor ou igual ao tamanho da barra, ou seja, não se pode cortar mais itens do que a barra permite. Uma forma de modelar este problema, é identificar *n* padrões de corte. A partir disso, basta determinar quantos objetos de cada padrão devem ser cortados para satisfazer a demanda, sempre utilizando a menor quantidade possível de objetos.

Modelagem matemática:

2.1.1 Índices:

i = 1,...,m Tipo de item.

j = 1,..., n Padrão de corte unidimensional.

2.2 Dados:

D_i Quantidade demandada de um determinado tipo de item i..

a_{ij} Número de peças do tipo i no padrão de corte j.

2.3 Variáveis de Decisão:

X_j Número de objetos cortados segundo o padrão de corte j.

2.4 Formulação matemática:

O modelo matemático tem como objetivo encontrar a solução otima que minimize a quantidade de objetos cortados (1), respeitando as restrições.

$$\min = \sum_{j=1}^{J} Xj \qquad (3)$$

Sujeito a:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq d_1 \qquad (4)$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq d_2 \qquad (5)$$

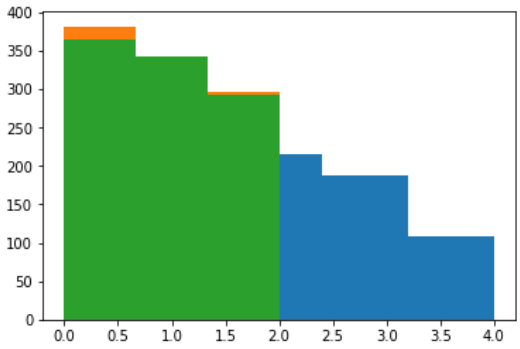
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq d_m \qquad (6)$$

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_m \geq 0 \; e \; inteiros \qquad (7)$$

Vale ressaltar que a condição das variáveis serem inteiras, torna difícil a resolução do problema. As únicas restrições deste problema são o atendimento da demanda de cada item e a não negatividade e integralidade das variáveis. A função objetivo apenas minimiza o número de objetos cortados, ou seja, ela não avalia a perda de matéria prima do corte.

Neste trabalho os padrões de corte forma definidos de duas formas. Primeiro, gera-se padrões homogêneos, isto é, padrões com um item apenas. Segundo, através de uma função que identifica quantos itens são possíveis cortar em um só objeto. A partir disso ela gera uma distribuição normal (com o centro em 0) com 1000 dados que estão distribuídos entre 0 e o máximo de itens suportados pelo objeto. Por exemplo, para o item 1 ($l1 = 30\ cm$), o máximo de vezes que este item pode ser cortado em um objeto de 120 são 4. Portanto essa função sorteará 1000 valores aleatórios entre 0 e 4.

Figura 2: Gerando Padrões com distribuição normal



Com esses valores conhecidos, sortia-se 3 números aleatórios, cada um referente a um item de corte. Ou seja, teremos um vetor que identifica um possível padrão.

$$\textit{Possível padrão} = [3,0,1]$$

Caso esse padrão satisfaça o sistema de equações 1 e 2, este é adicionada a uma lista que contêm todos os padrões aceitos. Interessante ressaltar que após essas funções, temos um número significativo de padrões e esta quantidade pode se alterar após o Algoritmo Genético e após o FFD, no qual mutações podem gerar novos padrões até então não descobertos e a heurística também se mostrou se eficaz em gerar padrões heterogêneos. Tendo armazenados todos os padrões, todos os parâmetros para modelar matematicamente estão definidos.

Este segundo método para gerar padrões é eficiente para pequenas e médias instâncias, porém para instâncias maiores o primeiro método permite que o AG comece com padrões homogêneos e através das mutações pode se gerar padrões heterogêneos.

Neste trabalho utilizou-se o Solver CBC disponibilizado por uma biblioteca do Google chamada OR-tools para se otimizar o problema unidimensional com um único tamanho de objeto a ser cortado e quantidades deste em estoque é ilimitada.

3. First-Fit-Decreasing

Foi implementado na linguagem Python também uma heurística construtiva para o problema, essa maneira de resolver não garante uma solução ótima. A heurística escolhida foi a FFD (First-Fit-Decreasing). Na qual consiste em cortar o maior item a partir de um objeto até que não seja mais possível, ou seja, até exceder o tamanho do objeto, ou, até que a demanda do item cortado seja atendida. Quando não for possível realizar o corte do maior item, aloca-se o segundo maior e assim por diante até o menor

item demandado. Ao final desse modelo temos os padrões de corte, cada padrão é usado o máximo de vezes possível (frequência) sem que a demanda seja excedida.

Os parâmetros de entrada para esta heurística são**:** o comprimento do objeto, o nome do corte, o tamanho do corte e a demanda dele. Com essas informações inicializa-se a heurística. O grande desafio dessa heurística é gerar padrões heterógenos de cortes, pois os padrões de cortem devem priorizar o item com o maior tamanho, porém assim que a demanda é satisfeita deve-se parar de cortar este item e cortar um item diferente que ainda seja menor do que o restante do objeto. Vale ressaltar, que nesta modelagem também foi considerado que quantidade de objetos em estoque é ilimitada.

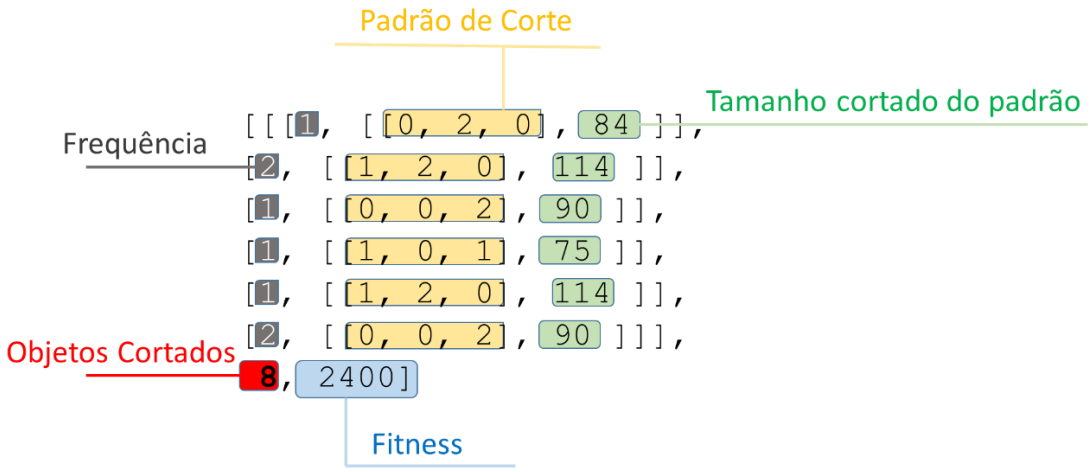
4. Algoritmo Genético

Para iniciarmos o algoritmo precisamos primeiro definir alguns parâmetros, como o tamanho da população, a quantidade de gerações, além das características básicas do problema como tamanho do objeto, tamanho dos itens e as respectivas demandas.

Com testes foi possível notar que o tamanho da população influenciava diretamente na solução final deste problema, já que valores pequenos para a população se mostraram ineficazes para problemas com muitos itens, visto que, não conseguem obter uma grande diversidade indivíduos. Ou seja, populações grande ampliam consideravelmente o espaço de busca, porém prejudicam a performance computacional do algoritmo. O tamanho definido foi de 500 indivíduos. Os números de gerações seguem a mesma performance do tamanho da população, então foi definido como 250.

Para modelar um algoritmo genético é preciso codificar o problema. A codificação escolhida para este projeto é uma adaptação da qual foi a proposta por Khalifa et al (2006), na qual os genes são processados em pares, sendo que o primeiro gene representa a quantidade de vezes que o segundo gene foi utilizado. Ou seja, o primeiro representa a frequência e o segundo o padrão de corte.

Figura 3: População



Definido como será a codificação, foi necessário estabelecer em seguida o tamanho da cadeia de DNA dos indivíduos. Isto é, quantas opções de padrão de corte compõe um indivíduo, na figura XXX são 6 genes. Caso todos os itens sejam maiores do que a metade do objeto o gene é o dobro da quantidade de itens. Caso algum item seja menor do a metade do objeto, o gene é estipulado através de um pequeno algoritmo.

Se (Quantidade de itens > 30) então

Gene_DNA = 32

Se (Quantidade de itens >= 12) então

Gene_DNA = 24

Senão se (Quantidade de itens > =5) então

Gene_DNA = 2 * Quantidade de itens

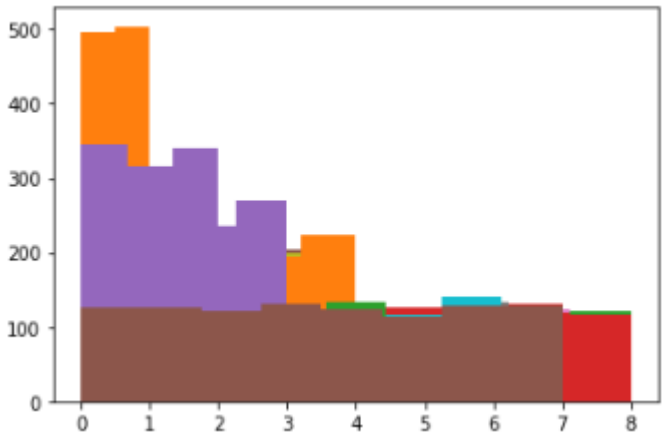
Senão

Gene_DNA= Quantidade de itens+2

A partir do momento que temos definidos os parâmetros para iniciar um algoritmo genético, inicializa-se a população. Para isso, precisa-se gerar padrões (processo já detalhado na seção do Solver), com estes identificados, gera-se um conjunto de frequências com 1000 números para cada padrão. Este conjunto é elaborado de maneira semelhante aos padrões, através de uma destruição uniforme entre 0 e o máximo de vezes que aquele padrão precisa mostrar-se para satisfazer a maior demanda dos itens. Por

exemplo, o padrão: [[4, 0, 0], 120] precisa aparecer uma vez para produzir toda a demanda do item 1. Então a frequência máxima deste padrão é 1. Já o padrão [2, 1, 0], 102] precisa aparecer 8 vezes para satisfazer a maior demanda dos itens que ele corta, no caso a demanda do item 2 é 8. Ao final desta etapa temos uma lista com 1000 números na qual todas as possíveis frequências para um padrão são determinadas.

Figura 4: Gerando frequências com distribuição uniforme



Em seguida seleciona-se padrões e frequências para montar um indivíduo. Repete-se esse processo de seleção até que o indivíduo tenha a quantidade de gene definida. Após a população ser iniciada, decodifica-se os indivíduos para determinar quantos objetos foram cortados e o fitness. Para o fitness foram propostas bonificações e penalizações.

$$Fitness = -Bi + (Quantidade\ de\ barras * P1) + (Quantidade\ do\ item\ i\ produzida\ abaixo\ da\ demanda * P2) + Quantidade\ do\ item\ i\ \ produzida\ acima\ da\ demanda * P3)$$

Ou seja, caso a quantidade produzida do item i seja igual ao valor da demanda, o fitness recebe uma bonificação (B) de 400. Para este problema, uma bonificação significa uma minimização do fitness. Caso o indivíduo, produza acima da demanda do item i, a diferença entre a produção e demanda é multiplicada 80 (P2). Caso, a produção seja inferior a demanda, a diferença é multiplicada por 200. O objetivo é limitar que indivíduos que produzam uma quantidade de itens inferior a demanda não permaneçam na população. Penaliza-se também a quantidade de barras cortadas, multiplicando a quantidade por 100 (P1).

Com a método de inicialização e decodificação implementados, inicia-se a seleção natural. Onde o primeiro passo é implementar um operador de seleção de Pais para o

Crossover, optou-se então por um torneio binário. Neste procedimento, sorteiam-se dois indivíduos ao acaso, comparam-se suas aptidões e o mais apto destes dois é selecionado, neste caso como a função objetivo é de minimização, seleciona-se aquele com a menor valor. Este procedimento é repetido até que toda a população tenha participado do torneio.

Após selecionarmos os pais, realizamos o operador genético predominante. Através de cruzamentos são criados indivíduos novos misturando as características de dois pais. Foram desenvolvidos dois operadores, onde a chance de ocorrer o primeiro e o segundo são 50%. O primeiro foi o crossover de 1 ponto, no qual um cruzamento onde temos dois pais, são gerados dois filhos idênticos aos pais. Estipula-se uma taxa de crossover de 65%. Caso ocorra o crossover, sorteia aleatoriamente o ponto de corte, deste ponto em diante o filho selecionado recebe as características do segundo pai (Tabela 2), ou seja, os padrões do segundo pai. Lembrando que este filho selecionado é cópia exata do primeiro pai até a realização do crossover.

Figura 5: Crossover de 1 Ponto

Indivíduo 1	1	1	0	1	0	1	1	1
Indivíduo 2	0	1	1	1	1	1	0	0
Descendente 1	1	1	0	1	0	1	0	0
Descendente 2	0	1	1	1	1	1	1	1

O segundo crossover neste algoritmo é uniforme, ou seja, são gerados classificadores aleatórios (0 e 1) do tamanho do número de itens do problema. Ou seja, um indivíduo com 8 padrões teremos uma sequência com 8 classificador. Caso o classificador seja igual a 1, esta posição terá o padrão copiado do pai 1 e caso o classificador seja 0, esta posição terá o padrão copiado do pai 2:

Figura 6: Crossover Uniforme

Classificador	1	1	0	1	0	1	0	1
Filho	1	1	1	1	1	1	0	1

Já para o operador de mutação, foi implementado dois métodos. O primeiro escolhe-se um padrão aleatório do filho e altera-se a frequência dele, essa substituição

não é aleatória, visto que, o número escolhido para a mudança pertence ao conjunto de frequências definido anteriormente. A taxa de mutação deste método é equivalente a 5%. O segundo método garante a criação de novos padrões para isso escolhe-se aleatoriamente um padrão pertencente ao filho. Após a escolha, identifica-se a possibilidade de adicionar mais algum item ao padrão. Ou seja, caso a perda do objeto seja maior que algum item, adiciona-se este item ao padrão. A taxa deste método é de 5%.

Figura 7: Mutação no Padrão


```
[[[1 , [[0, 2, 0], 84 ]],

[[[1 , [[1, 2, 0], 114 ]],
```

Vale ressaltar, que após os operadores de crossover e de mutação realiza-se a decodificação da população com o operador já implementado anteriormente. Por fim, concatena-se a população de pais e de filhos e escolhe-se os indivíduos com os melhores (menores) fitness para se manter na população. Este algoritmo perpetua o tamanho da população inicial.

Os critérios de paradas para o AG são um tempo limite de 30 minutos, ou até que todas as gerações sejam rodadas ou, caso o melhor fitness não melhore após 30 gerações.

5. Resultados

Para os métodos implementados (Solver, FFD, AG) forma testado diversos problemas. Os testes se iniciaram com instancias até 8 itens, em seguida com 20,30,40 e 80 itens, para problemas maiores foi mantido o tamanho do objeto 1000 a fim de gerar variabilidade de tamanhos. Vale ressaltar que o operador que gera instâncias também foi implementado neste trabalho. Este gera primeiro o tamanho do corte através de uma distribuição normal e repete-se o mesmo procedimento para as demandas. Vale observar, que este operador pode ter influenciado diretamente nos resultados obtidos.

Tabela 1: Resultados para as instâncias testadas

		Objetos Cortados	Tempo (s)	Demana não atingida
Problema 1	Solver	8	0,015	
	FFD	8	0,001	
	AG	8	53,73	-
Problema 2	Solver	8	8	
	FFD	8	0	
	AG	8	31,47	
Problema 3	Solver	127	0,04	
	FFD	127	0	
	AG	126	261	x
Problema 4	Solver	449	0,13	
	FFD	465	0	
	AG	483	97,14	
Problema 5	Solver	72	0,12	
	FFD	75	0	
	AG	79	65	
Problema 6	Solver	8342	0,38	
	FFD	9347	0,0009	
	AG	10413	739	x
Problema 7	Solver	14483	0,09	
	FFD	14842	0,0019	
	AG	17685	11183	x
Problema 8	Solver	12597	0,3	
	FFD	14424	0,005	
	AG	15078	1661	x
Problema 9	Solver	31048	1,17	
	FFD	32021	0,027	
	AG	38073	1807	x

O Solver resolveu todas as instâncias em um tempo considerado viável, portanto este ainda é o método mais eficiente para resolver problemas de corte unidimensional com a quantidade de objetos ilimitada e apenas um objeto disponível para corte. Porém, a utilização do Solver só se torna eficaz quando se tem uma grande quantidade de padrões já identificados. Isto só foi possível após as heurísticas terem sido rodadas. O Algoritmo Genético se mostrou eficaz ao gerar padrões heterogêneos para instancias pequenas, porém, mas instâncias maiores o seu desempenho não foi bom, visto que, a geração de novos padrões é aleatória no operador que inicializa a população. Mesmo com a mutação no algoritmo com uma taxa de 5% (Mutação Para Novos Padrões), o operador não foi capaz de ampliar o espaço de busca. Ou seja, para instâncias maiores o AG gerou mais

padrões homogêneos do que heterogêneos. O algoritmo se mostrou mais eficiente para problemas os quais as demandas são em média 10

Enquanto a heurística FFD é eficaz na geração de padrões heterogêneos, porém para pequenas instâncias a quantidade de padrões gerados é pequena, o que prejudica o Solver, caso ele seja implementado em seguida da heurística como foi feito neste trabalho. O algoritmo FFD, se mostrou eficiente em todas instâncias, tanto para pequenas quanto para grandes. Portanto, após o solver este é melhor do que o AG para estas instâncias

6. Conclusão

Apesar do solver ser capaz de resolver todas instâncias em um tempo viável, nota-se que a heurística FFD apresentou boas soluções para todas as instâncias testadas. O algoritmo genético se mostrou eficiente para instâncias pequenas.

Os três métodos podem ser melhorados adicionando o fator perda na função objetivo, para que ele avalie a quantidade de matéria prima desperdiçada. Além desta melhora, poderia tratar de problemas com mais objetos disponíveis em estoque e com a quantidade destes limitadas.

Apesar do algoritmo genético não ter se mostrado eficaz sua utilização não foi descartada para este tipo problema, visto que, modificações na estrutura pode melhorar seu rendimento. Além disso, a metodologia de bonificação e penalização nos permite criar diversas restrições com facilidade como, por exemplo, tempo de entrega e custo de mão de obra (GOLFETO, MORETTI, SALLES, 2007).

A taxa de mutação, tamanho da população e número de gerações influencia diretamente no AG, sendo assim para problemas de larga escala é interessante analisar estes parâmetros cuidadosamente para que se pondere os melhores tendo em vista o rendimento computacional (tempo).

REFRÊNCIAS BIBLIOGRÁFICAS

ARENELES, Marcos. ARMENTANO, Vinícius. MORABITO, Reinaldo. Pesquisa Operacional - Rio de Janeiro: Elsevier **:** ABEPRO, 2011**.** Pp. 39

Khalifa, Y., Salem, O. e Shahin, A., "Cutting Stock Waste Reduction Using Genetic Algorithms".Proceedings of the 8th Conference on Genetic and evolutionary computation, pp.1675-1680, 2006.

GOLFETO, Rodrigo. MORETTI, Antônio. SALES, Luiz. Algoritmo Genético Aplicado Ao Problema De corte Unidimensional. SBPO – Fortaleza, CE. 2007.

ANEXO

Anexo: Instância 1

Problema 1	Tamanho	Demanda
l1	108	4
l2	13	8
l3	90	7
L	194	

Anexo: Instância 2

Problema 2	Tamanho	Demanda
l1	30	4
l2	42	8
l3	45	7
L	120	

Anexo: Instância 3

Problema 3	Tamanho	Demanda
l1	117	7
l2	215	29
l3	92	32
l4	178	19
l5	151	27
l6	236	14
l7	69	26
l8	181	32
L	250	

Anexo: Instância 4

Problema 4	Tamanho	Demanda
l1	863	1000
l2	705	1500
l3	555	800
l4	402	800
L	6000	

Anexo: Instância 5

Problema 5	Tamanho	Demanda
l1	20	80
l2	50	120
l3	25	110
L	170	

Anexo: Instância 6

Problema 6	Tamanho	Demanda
l1	551	663
l2	773	1361
l3	380	771
l4	330	691
l5	472	1022
l6	432	203
l7	667	1333
l8	523	774
l9	707	442
l10	647	80
l11	730	979
l12	460	1352
l13	584	494
l14	478	1025
l15	202	962
l16	319	429
l17	396	486
l18	485	675
l19	790	1342
l20	336	154
L	1000	

Anexo: Instância 6 (L=1000)

{'l1': [546, 1148], 'l2': [521, 421], 'l3': [598, 875], 'l4': [640, 1682], 'l5': [501, 961], 'l6': [253, 477], 'l7': [747, 405], 'l8': [401, 226], 'l9': [576, 363], 'l10': [475, 832], 'l11': [770, 1787], 'l12': [440, 708], 'l13': [817, 852], 'l14': [752, 551], 'l15': [535, 884], 'l16': [246, 253], 'l17': [926, 827], 'l18': [481, 1154], 'l19': [339, 38], 'l20': [435, 1076], 'l21': [733, 474], 'l22': [587, 170], 'l23': [539, 1718], 'l24': [425, 992], 'l25': [451, 2114], 'l26': [495, 791], 'l27': [233, 1113], 'l28': [634, 456], 'l29': [363, 532], 'l30': [219, 637]}

Anexo: Instância 7 (L=1000)


```
{'l1': [752, 92], 'l2': [450, 908], 'l3': [487, 959], 'l4': [356, 602], 'l5': [668, 656], 'l6': [553, 336], 'l7': [609, 235],
'l8': [880, 1334], 'l9': [554, 1032], 'l10': [361, 734], 'l11': [280, 303], 'l12': [681, 893], 'l13': [702, 773], 'l14': [647,
153], 'l15': [301, 1078], 'l16': [896, 603], 'l17': [319, 19], 'l18': [244, 316], 'l19': [349, 330], 'l20': [867, 192], 'l21':
[723, 114], 'l22': [713, 838], 'l23': [384, 358], 'l24': [472, 558], 'l25': [534, 332], 'l26': [709, 47], 'l27': [226, 77], 'l2
8': [799, 671], 'l29': [656, 284], 'l30': [583, 715], 'l31': [932, 1128], 'l32': [430, 668], 'l33': [527, 244], 'l34': [439, 46
8], 'l35': [511, 565], 'l36': [317, 1139], 'l37': [476, 619], 'l38': [713, 780], 'l39': [305, 1428], 'l40': [362, 1060]}
```


Anexo: Instância 8 (L=1000)


```
{'l1': [449, 74], 'l2': [711, 711], 'l3': [404, 101], 'l4': [400, 810], 'l5': [467, 1262], 'l6': [693, 755], 'l7': [496, 240],
'l8': [905, 743], 'l9': [466, 462], 'l10': [309, 753], 'l11': [766, 731], 'l12': [505, 1464], 'l13': [765, 348], 'l14': [612, 5
76], 'l15': [453, 1341], 'l16': [289, 1837], 'l17': [547, 197], 'l18': [297, 492], 'l19': [558, 11], 'l20': [654, 886], 'l21':
[503, 1187], 'l22': [775, 1125], 'l23': [412, 861], 'l24': [543, 350], 'l25': [400, 828], 'l26': [309, 868], 'l27': [741, 352],
'l28': [467, 1189], 'l29': [841, 1400], 'l30': [914, 815], 'l31': [624, 949], 'l32': [572, 785], 'l33': [768, 993], 'l34': [62
6, 1255], 'l35': [529, 1512], 'l36': [296, 452], 'l37': [702, 171], 'l38': [393, 1288], 'l39': [221, 734], 'l40': [721, 1080],
'l41': [253, 554], 'l42': [724, 672], 'l43': [411, 33], 'l44': [305, 394], 'l45': [263, 1083], 'l46': [700, 139], 'l47': [704,
727], 'l48': [868, 255], 'l49': [432, 481], 'l50': [465, 323], 'l51': [291, 527], 'l52': [781, 1006], 'l53': [312, 1357], 'l5
4': [387, 1075], 'l55': [686, 85], 'l56': [223, 792], 'l57': [804, 509], 'l58': [631, 277], 'l59': [705, 211], 'l60': [325, 115
5], 'l61': [377, 1138], 'l62': [776, 699], 'l63': [698, 935], 'l64': [512, 93], 'l65': [272, 360], 'l66': [267, 908], 'l67': [7
95, 998], 'l68': [782, 421], 'l69': [459, 2051], 'l70': [216, 1142], 'l71': [281, 2], 'l72': [276, 923], 'l73': [423, 116], 'l7
4': [303, 872], 'l75': [492, 388], 'l76': [355, 853], 'l77': [585, 1131], 'l78': [442, 274], 'l79': [727, 314], 'l80': [364, 10
43]}
```