Análise comparativa entre heurística e algoritmo de aprendizado por reforço profundo na abordagem do problema de corte/empacotamento 2D com peças irregulares.

Felipe Silvestre Cardoso Roberto
IA 1s-2024
UNIFESP
São José dos Campos, Brasil
felipe.silvestre@unifesp.br

***Abstract*—Neste artigo, serão discutidas soluções para o problema de corte e empacotamento bidimensional com peças irregulares, que podem ou não conter furos. O estudo estará voltado a comparação entre a heurística desenvolvida, Greddy Column Generation (GCG), e um modelo treinado pelo algorítimo Proximal Policy Optimization (PPO). A análise será feita tanto em relação às soluções encontradas por ambos os algorítimos, quanto às dificuldades para desenvolvê-los. Também será comparada a heurística desenvolvida, GCG, com o famoso método da literatura que já foi utilizado para solucionar esse problema, o GRASP[1]. Junto aos algorítimos, será abordada a importância do fecho convexo como medida avaliativa para problemas desse caráter.**

I. Introdução e Motivação

A necessidade de posicionar objetos de forma que poupem o máximo de espaço possível sempre esteve presente na sociedade, desde organizar uma mala de viagem até como realizar o corte de uma chapa de aço, com intuito de evitar desperdício. Dessa forma, meios de encontrar uma solução ótima foram sendo desenvolvidos a fim de otimizar esse processo. Para uma padronização, esse dilema foi intitulado como problema de corte e empacotamento.

A questão, inicialmente, estava relacionado apenas ao unidimensional, tornando-se necessário desenvolver soluções para tornar o corte, principalmente, de materiais em uma dimensão, como barras de aço, vigas e canos, de forma que não houvesse desperdício de material. Com o passar do tempo, o problema foi adaptado para o corte/empacotamento bidimensional, que se baseia em uma lista de peças que devem ser posicionadas em uma certa área, visando maximizar o espaço utilizado. Desde as primeiras sugestões de soluções, com Gilmore e Gomory em 1960 [7], para o problema unidimensional e bidimensional, muitas variantes foram desenvolvidas, tanto em relação à abordagem, com cortes e empacotamento tridimensional, quanto em relação às soluções, inicialmente com heurísticas e chegando, aos dias atuais, com algoritmos de Deep Reinforcement Learning.

Dessa forma, a questão em análise se faz muito comum em indústrias que precisam evitar desperdício de material, ao se trabalhar com corte, ou então o aumento do espaço utilizado, com o empacotamento que, no fim das contas, representam a mesma ideia. Além disso, a solução ótima desse problema ajuda a otimizar os processos de produção, tornando-os mais eficientes, de modo que, a tomada de decisão sobre como deve ser feito o corte/empacotamento se torna acelerada, implicando, assim, em uma redução do tempo, sem contar que, com a diminuição do desperdício de material, a margem de lucro da empresa aumenta significativamente.

II. Conceitos Importantes e Trabalhos Relacionados

Com base no tema abordado, é necessário possuir alguns conceitos em mente para uma melhor compreensão do assunto que será discutido. O problema de corte/empacotamento 2D possui duas variações, a primeira com peças regulares e a segunda com irregulares, de maneira que, são definidas como peças regulares formas retangulares variadas em suas dimensões. A outra variante do problema possui exatamente a mesma ideia, mas ao invés de utilizar formas retangulares, são utilizadas peças irregulares variadas. Desse modo, formas irregulares variadas é definida como qualquer formato, seja ele convexo ou concavo, que seja diferente de retangulos. Nesse estudo, será abordado soluções para peças de diversos formatos, concavas e convexas, distribuídos em 10 datasets de instâncias, que serão apresentados ao longo do artigo.

Outro tópico que é de extrema importância possuir o conhecimento prévio para uma melhor compreensão é em relação às soluções. Nesse artigo, iremos abordar a comparação de duas sugestões de algoritimos, com seus respectivos resultados. A análise será feita entre uma heurística, a qual inicialmente chamaremos de Varredura e um algoritmo de aprendizado por reforço profundo. De forma geral, no contexto de inteligência artificial, uma heurística pode ser definida como: dado um problema de alta complexidade, um método de se aproximar da melhor solução possível de uma maneira rápida e fácil. Em outras palavras, uma heurística é um meio de buscar uma solução em um número finito, ou infinito, de possíveis estados, por meio aproximações sobre quais caminhos devem

ser tomados dentre o espaço disponivel. Em contrapartida, um algoritmo que utiliza redes neurais profundas se enquadra em um método de alta complexidade, exigindo, assim, um grande modelo estatístico e probabilístico para que seja possível estimar uma solução para o problema, além de, por ser um algoritimo de alto nivel, o tempo de execução aumenta significativamente. Dessa forma, um algoritmo de aprendizado por reforço profundo utiliza vários conceitos da literatura, que serão abordados nos proximos tópicos.

O aprendizado por reforço é um método no qual um agente aprende a realizar uma tarefa desejada através da experiência. A estrutura básica funciona da seguinte forma: o agente executa uma ação em um ambiente com o qual deve aprender a interagir. Essa ação altera o estado do ambiente, o qual é então, avaliado e fornece ao agente um feedback sobre o quão benéfica foi essa ação. Nesse momento entram as redes neurais no aprendizado por reforço profundo. Com múltiplos feedbacks das ações realizadas no ambiente, a rede neural é treinada para, ao receber um estado do ambiente, identificar a melhor ação possível. Essa aprendizagem ocorre através do processo de forward propagation - momento em que a rede recebe o estado atual do ambiente e gera uma estimativa de qual é a melhor ação - e backpropagation, ajustando os pesos da rede para compreender porque certas ações receberam determinados valores de feedback, também conhecidos como recompensas. A rede, então, escolhe uma ação baseada no estado atual do ambiente, repetindo todo esse processo inúmeras vezes. Eventualmente, o agente se torna hábil em devolver uma sequência de ações que levam a uma solução ótima sempre que encontra um ambiente semelhante.

Em ambas as soluções o fecho convexo foi utilizado como ferramenta avaliativa. Tal medida pode ser entendido como a menor região convexa (polígono convexo), que abrange por inteiro determinado conjunto de pontos. Nesse estudo, tal conjunto será os vértices de peças já posicionadas na área de corte. Se o preenchimento de um certo estado se aproximar da área do fecho convexo, isso significa que as peças estão perto uma das outras, e consequentemente, desperdiçando menos espaço.

Com essas informações em mente, será possível possuir um melhor entendimento de como os algorítimos foram desenvolvidos. Nesse sentido, o problema de corte e empacotamento, por ser uma questão muito antiga, já está há muito tempo em discussão na literatura. Assim, foram selecionado alguns artigos que possuem informações que auxiliaram na estruturação desse estudo:

A. Heuristics for Two-Dimensional Knapsack and Cutting Stock Problems with Items of Irregular Shape [1]

Nesse artigo, os autores exploram problemas de corte e empacotamento bidimensionais (2D) com itens de forma irregular, utilizando a heurística GRASP (Greedy Randomized Adaptive Search Procedure) como abordagem principal. O estudo aborda itens representados por polígonos simples, incluindo aqueles com buracos, para resolver o Problema da Mochila 0/1 e o Problema da Mochila irrestrito. A GRASP é uma meta-heurística que consiste em duas fases principais: construção e busca local. Na fase de construção, uma solução inicial é gerada de forma gulosa e aleatória, combinando critérios determinísticos com escolhas aleatórias controladas para criar soluções diversificadas. Em seguida, na fase de busca local, a solução gerada é refinada iterativamente, explorando as vizinhanças para encontrar melhorias. Se uma solução melhor for encontrada, ela substitui a solução atual, e o processo continua até que não sejam encontradas melhorias adicionais. Esta combinação permite que a GRASP percorra o espaço de soluções de forma eficiente, utilizando uma alta exploração para encontrar soluções para problemas complexos como os de corte e empacotamento.

B. An Efficient Heuristic Approach for Irregular Cutting Stock Problem in Ship Building Industry [2]

Este artigo apresenta uma abordagem eficiente para resolver o problema de corte/empacotamento, aplicando a heurística de posicionamento conhecida como bottom-left-fill (BLF). A heurística BLF posiciona os itens no canto inferior esquerdo da área de corte disponível, buscando preencher o espaço de forma organizada e sem lacunas. O princípio do BLF é simples: cada item é colocado o mais à esquerda e, dentro dessa posição, o mais embaixo possível, garantindo um uso mais eficiente do espaço disponível.

A solução com a BLF é encontrada posicionando cada peça na área de corte seguindo uma ordem específica da lista de peças. A ordem das peças na lista é crucial, pois influencia diretamente o desempenho do algoritmo. A BLF tenta posicionar cada peça no canto inferior esquerdo disponível, ajustando-a o mais à esquerda e para baixo possível. A sequência em que as peças são processadas pode impactar a eficiência do preenchimento, uma vez que diferentes ordens podem levar a diferentes arranjos, com alguns resultando em um uso de espaço mais eficiente do que outros. O artigo explora a melhor forma de ordenar essa lista para maximizar o aproveitamento do material e minimizar os espaços vazios, garantindo uma solução eficaz para o problema de corte e empacotamento.

C. A Reinforcement Learning Approach to the Stochastic Cutting Stock Problem [3]

O estudo propõe uma abordagem de aprendizado por reforço para resolver o problema de corte estocástico. O desafio é decidir quais padrões de corte usar para minimizar os custos, considerando que a demanda futura é incerta. Os autores formulam esse problema como um processo de decisão de Markov e aplicam técnicas de aprendizado por reforço para encontrar políticas de corte eficientes. Essas políticas podem ajudar a controlar os estoques de forma mais inteligente e reduzir os custos em até 80com políticas mais simples

D. A Hybrid Reinforcement Learning Algorithm for 2D Irregular Packing Problems [4]

O artigo “A Hybrid Reinforcement Learning Algorithm for 2D Irregular Packing Problems” propõe um algoritmo de aprendizado por reforço (RL) híbrido para resolver problemas

de empacotamento de peças irregulares em duas dimensões (2D). Inspirado no RL clássico, o algoritmo combina Monte Carlo learning (MC), Q-learning e Sarsa-learning para otimizar o empacotamento de peças irregulares. O estudo apresenta um modelo matemático para o problema e demonstra que o algoritmo proposto pode alcançar resultados comparáveis ou melhores do que alguns algoritmos heurísticos clássicos. Além disso, o trabalho sugere que essa abordagem pode servir como base para futuros algoritmos de deep reinforcement learning (DRL) na resolução de problemas de empacotamento.

III. Objetivos

O intuito de desenvolver esse estudo é, além de encontrar soluções para esse problema, que está a mais de meio século em discussão, comparar os resultados que foram sugeridos no início dos estudos na literatura, como as heurísticas, com soluções da atualidade, como os algoritmos de redes neurais profundas. A análise será feita tanto em relação às soluções encontradas, mas, principalmente, nas dificuldades de estruturar a arquitetar ambos os algorítimos. Também será comparado a heurística desenvolvida nesse artigo com o famoso método da literatura, o GRASP[1]. Para realizar tal comparação, foi desenvolvido todo um ambiente em Python, onde será possível posicionar as peças e avaliar o quão bom foi aquela solução, e também será possível desenvolver ambos os algoritmos, a heurística e o algoritmo de Deep Reinforcement Learning, além de poder testar suas soluções, para que, por fim, seja feita a comparação dos algoritimos. Dessa forma, ao realizar todos os testes em um único ambiente, sempre com as mesmas características e mesmos tipos de peças, será possível alcançar uma comparação mais concisa dos resultados, por não possuir as variações que utilizar ambientes e peças diferentes poderia causar. Junto a isso, todos os algorítimos avaliados serão em relação ao mesmo conjunto de peças, para unificar o modelo de avaliação de cada método.

IV. Metodologia Experimental

Para a execução dessa análise, foi desenvolvido um ambiente de corte e empacotamento com a biblioteca Turtle, do Python. Com essa interface gráfica, foi possível desenvolver todo um sistema de criação da área de corte, utilizar as peças presentes nos datasets e posicioná-las, fora o cálculo dinâmico que esse ambiente permite fazer sobre desperdício e preenchimento das peças na área. Com o ambiente pronto, foi possível implementar os algoritmos para a realização da análise.

A. Greedy Column Generation

O primeiro algoritmo desenvolvido foi inicialmente chamado de ”varredura”, uma adaptação da heurística ”bottom-left-fill”[2], mas se diferencia na forma como lida com o erro. A ideia inicial de ambos os algoritmos é posicionar a última peça da lista na posição mais à esquerda possível e sempre na parte mais inferior. O problema encontrado foi que, para que a varredura alcançasse uma solução, a lista de peças precisava estar previamente ordenada de tal forma que, ao percorrer a área de corte, a posição escolhida para a última peça fosse a ”certa”. Nesse ponto, a heurística que será apresentada se diferencia do ”BLF”[2]. No algoritmo original, o foco é estudar como ordenar a lista de peças para posicionar todas na melhor posição durante a execução do algoritmo. Assim, tornou-se fundamental encontrar uma forma para que o algoritmo desenvolvido determinasse a posição correta das peças, independentemente da ordem da lista, abordando o problema encontrado com outro olhar.

Nesse momento, foi implementado um algoritmo guloso guiado por uma função de custo. A diferença entre a nova heurística e a varredura é que, ao invés de apenas posicionar a última peça na primeira posição disponível, essa nova versão parte de uma peça inicial posicionada previamente pela varredura, em seguida testa todas as peças disponíveis e suas rotações na primeira posição disponível, encontrada pela varredura, e avalia, pela função de custo, o quão bem cada uma se encaixa naquela posição. Ao analisar todas as peças, o algoritmo escolhe a que melhor ocupa o espaço e, consequentemente, desperdiça-o menos. A função de custo foi estruturada da seguinte forma: era simulada a área de corte caso a peça em análise fosse colocada na posição (x, y) escolhida pela varredura e, então, era calculada a área do fecho convexo das peças posicionadas até o momento, ou seja, o cálculo da área do menor polígono convexo que abrange todas as outras peças. A função de custo consistia em medir o quanto as peças ocupavam do fecho convexo. Uma baixa ocupação indicava que as peças foram posicionadas longe umas das outras e, consequentemente, desperdiçava-se mais espaço. Já uma alta ocupação do fecho convexo sinalizava que as peças estavam mais próximas umas das outras, desperdiçando, assim, menos espaço. Dessa forma, era possível avaliar a eficiência de cada peça na posição (x, y) testada. A peça que melhor preenchia o fecho convexo era escolhida para a posição correspondente encontrada pela varredura. A função de custo se estruturou da seguinte forma, com PT representando o preenchimento total da área de corte caso a peça x fosse posicionada, e AF a área do fecho convexo caso a peça x fosse escolhida:

$$f(x) = PT/AF$$

Entretanto, o algoritmo ainda apresentava um problema. A função de custo avaliava perfeitamente quando a primeira posição possível estava acima da peça anterior, empilhando as peças de forma a desperdiçar menos espaço possível. Porém, quando a próxima opção de posição estava ao lado da pilha inicial de peças, a área do fecho convexo aumentava demais. Dessa forma, o algoritmo escolhia apenas a peça que menos aumentava a área do fecho convexo, e não a que melhor preenchia o espaço. Isso fez com que, mesmo com esse algoritmo guloso, não fosse possível alcançar a solução ótima. Em outras palavras, esse algoritmo apenas fazia pilhas perfeitas, mas não posicionava bem as peças após a primeira pilha.

Por fim, outro algoritmo guloso foi implementado em conjunto com esse, nomeado de Greedy Column Generation, abreviado em GCG. Essa nova versão da heurística aproveita a qualidade da versão anterior em criar pilhas com alta ocupação do fecho convexo. Nesse sentido, o algoritmo partia de uma peça inicial já posicionada pela varredura, que limitava a largura da coluna. Então, a versão anterior da heurística era executada, criando uma pilha, com base na peça inicial, com alto preenchimento do fecho convexo. Ao terminar a primeira pilha, todas as peças eram removidas e uma nova peça era usada como peça base. Dessa forma, todas as peças eram testadas com base na coluna, e uma nova função de custo escolhia qual era a melhor coluna. Em seguida, a coluna que tivesse a melhor avaliação era posicionada, e o algoritmo era executado novamente, começando uma nova série de testes de colunas ao lado da coluna posicionada. A função de custo desse novo algoritmo é uma multiplicação entre o preenchimento final do fecho convexo, o preenchimento da coluna (ou seja, a soma das áreas dos polígonos dividida pela área total da coluna), e o número de peças posicionadas. Dessa forma, o algoritmo consegue escolher qual peça inicial forma a melhor coluna naquele momento, que será escolhida, e então, enquanto for possível criar colunas, o algoritmo continua em execução. Dado um conjunto de peças a serem posicionadas, o GCG consegue encontrar uma aproximação de uma solução para esse conjunto de peças, escolhendo as melhores colunas para cada iteração. A função que escolhe a melhor coluna foi estruturada da seguinte forma:

$$f(c) = (AP/AC) * (AP/AF) * N$$

Com AP representando a área preenchida pelas peças da possível coluna, AC a area maxima que a coluna poderia ter, AF a área do fecho convexo final da coluna e N o número peças posicionadas na coluna. O algorítimo escolherá a coluna que alcançar o maior valor dessa função.

B. Proximal Policy Optimization (PPO)

Para o treinamento do algorítimos de aprendizado por reforço PPO, foi necessário utilizar as seguintes bibliotecas do python em conjunto:

1) OpenAI Gym: O OpenAI Gym oferece uma estrutura padronizada para a criação de ambientes de simulação personalizados. Esses ambientes permitem que algoritmos de aprendizado por reforço sejam testados e treinados de forma muito mais simples, de tal forma que, com o ambiente estruturado no padrão necessário da biblioteca, é possível implementar qualquer algoritmo desejado.

2) Stable Baseline3: A Stable Baselines3 (SB3) é um conjunto de implementações de algoritmos de aprendizado por reforço em PyTorch. Essa biblioteca oferece uma variedade de algoritmos para treinar agentes por reforço. A simplicidade e a otimização das implementações permitem que os desenvolvedores criem e testem agentes de forma eficiente. Além disso, o SB3 trabalha de forma compatível com o Gym, de tal forma que, com o ambiente personalizado no padrão do Gym, qualquer algoritmo do SB3 pode ser testado e implementado.

3) Numpy: A biblioteca Numpy oferece uma série de artifícios matemáticos e de manipulação de dados extremamente útil para se usar em conjunto com outras bibliotecas. Dessa forma, tal biblioteca torna muito mais simples trabalhar com os dados necessários para o algoritmo.

Por fim, foi necessário estruturar o algoritmo de aprendizado por reforço profundo para compará-lo com a heurística desenvolvida, GCG, em relação à solução e em como foram desenvolvidas. Com o ambiente pronto em Python usando a biblioteca Turtle, bastou adaptá-lo ao padrão da OpenAI Gym. Isso, combinado com a biblioteca Stable Baselines3, permitiu a implementação do algoritmo de reinforcement learning.

Para padronizar o ambiente, uma classe do tipo gym.Env foi criada conforme o problema. No método `__init__`, definimos o espaço de observação, que é o que a rede neural recebe como entrada para entender o ambiente. Esse espaço foi definido como uma matriz numpy, representando a área de corte e a lista de peças disponíveis. Também foi necessário declarar o espaço de ação, que representa as ações que a rede pode realizar. Atualmente, esse espaço inclui um valor "n" para o índice da peça a ser posicionada, um par ordenado (x, y) para a posição, e um valor "l" que varia de 0 a 3 para escolher o grau de rotação (0, 90, 180, 270 graus). Além disso, foi criado o método `step`, que executa a ação no ambiente, retornando o estado atual, a recompensa e um booleano que indica se a época terminou. Se o booleano for verdadeiro, o método `reset` é chamado para reiniciar o ambiente e começar uma nova série de ações.

Dessa forma, com o ambiente padronizado com a biblioteca Gym, já é possível treinar um modelo usando a Stable Baselines3. O algoritmo escolhido foi o PPO (Proximal Policy Optimization) devido à sua estrutura com duas redes neurais: uma para a política, que define as ações que o agente deve tomar, e outra para o valor, que estima a recompensa esperada para cada estado. Essas duas redes trabalham juntas de forma complementar: a rede de política sugere as melhores ações em cada estado, enquanto a rede de valor avalia essas ações e ajuda a refinar as decisões da rede de política. Essa configuração permite lidar bem com ambientes complexos, garantindo um processo de treinamento robusto e eficiente. Assim, o PPO mostrou-se uma escolha adequada para avaliar a eficácia do aprendizado por reforço em comparação ao Greedy Column Generation. Com o ambiente 100% configurado em todas as partes, foi necessário definir a função de custo, a qual ao realizar uma ação, retornará o quão bem foi a ação. Pela alta complexidade de definir uma função que ensine o modelo como posicionar as peças na área de corte, a medida avaliativa também possui um caráter extremamente complexo, motivo pelo qual o resultou na análise final dos algorítimos, que será discutido na conclusão. A função de recompensa se estruturou da seguinte forma: foi escolhido um modelo de função: recompensa escolhida x recompensa esperada. Para que o modelo aprenda gradualmente quais são as melhores ações. Caso o modelo não escolha a maior peça disponível, ele é punido pela diferença da maior peça disponível e a peça que escolheu. A primeira ação é avaliada pela distância em relação

aos vértices, para que o modelo aprenda que inicialmente é melhor possuir peças grandes perto dos vértices. Se ele não escolher a maior peça na primeira ação e/ou não posicioná-la perto de um dos vértices, independente das demais ações, ele será punido. Se ele posicionou a maior peça na primeira ação perto dos vértices, manterá o sistema de recompensa esperada x recompensa escolhida. Se após a primeira peça ele escolher a maior peça disponível, ele receberá como recompensa o preenchimento do fecho convexo, que o guiará para sempre posicionar as peças perto uma das outras, pois um alto preenchimento do fecho convexo significa um posicionamento compacto das peças. Por fim, se a peça escolhida não for a maior disponível, ele será punido pela diferença entre as áreas da maior peça disponível e a peça que o modelo posicionou. Dessa forma, foi possível treinar um modelo com o algorítimo PPO que aprendeu a posicionar as peças seguindo essa logica: posicionar a primeira peça perto dos vértices e depois posicionar as demais sempre com o intuito de manter um alto preenchimento do fecho convexo.

V. Resultados

Antes de apresentar os resultados de ambos os algorítimos, será compartilhado a lista de datasets das peças utilizadas para realizar a análise. Dessa forma, foi escolhido um conjunto de datasets usados por AM Del Valle, TA Queiroz, EC Xavier e FK Miyazawa no artigo Two-dimensional Irregular Cutting Stock Problem [1], artigo o qual apresentou os resultados alcançados pela heurística GRASP[1], dataset disponibilizado pelo laboratório LOCO da Unicamp, para estudos realizados sobre esse problema. Dentro do conjunto há 15 datasets, dos quais muitos foram utilizados em soluções famosas da literatura, como é o caso do dataset Albano, o qual foi criado por Antonio Albano em 1980 para implementar sua solução no artigo Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods [8]. Para esse estudo, foi utilizado apenas 10 conjuntos de peças, que serão apresentados na tabela a seguir com o nome do conjunto, quantas peças possuem nele e qual a proporção da área de corte.

TABLE I
Datasets utilizados

Nome	Numero de Peças	Dimensões da Área
FU	12	34x38
JACKOBS1	25	13x40
JACKOBS2	25	28x70
SHAPES2	28	27x15
DIGHE1	16	138x100
DIGHE2	10	134x100
ALBANO	24	101x49
DAGLI	30	66x60
MAO	20	20x25
MARQUES	24	84x104

A tabela a seguir apresenta os resultados da heurística GCG em comparação com o famoso método da literatura GRASP[1]. A medida avaliativa foi a relação de peças utilizadas por peças disponíveis. É possível notar que, na maioria dos casos, ambas as heurísticas obtiveram o mesmo resultado, e, quando diferem, foi em relação a apenas uma peça. O diferencial da heurística GCG é lidar bem com peças que possuem apenas um tipo de encaixe, como um quebra-cabeça. Junto a tabela serão apresentados algumas soluções encontradas por ambos os algoritmos, a fim de entender visualmente como ambos os algorítimos alcançam suas soluções e como se caracteriza esse diferencial da heurística GCG.

TABLE II
GCG x GRASP

Nome	GCG	GRASP
FU	100%	100%
JACKOBS1	100%	100%
JACKOBS2	100%	100%
SHAPES2	89.28%	92.8%
DIGHE1	93,75%	100%
DIGHE2	100%	100%
ALBANO	95.83%	95.83%
DAGLI	96.66%	96.66%
MAO	95%	100%
MARQUES	87.5%	100%

Agora, podemos proceder à análise detalhada dos resultados obtidos por ambos os algoritmos. Esta análise permitirá uma compreensão mais aprofundada dos diferentes caminhos e estratégias adotados por cada algoritmo para chegar às soluções. Ao examinar os resultados, será possível identificar como cada algoritmo abordou o problema. Esta avaliação comparativa fornecerá um melhor esclarecimento de como se estrutura uma solução feita pelo GCG e como se estrutura uma feita pela GRASP[1].

A. FU

1) GCG:

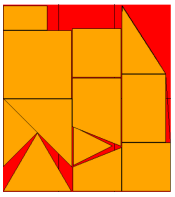

Fig. 1. Solução encontrada pela GCG.

2) GRASP:

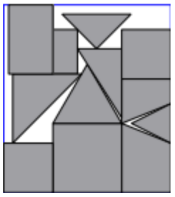

Fig. 2. Solução encontrada pela GRASP[1].

B. JACKOBS1

1) GCG:

Fig. 3. Solução encontrada pela GCG.

2) GRASP:

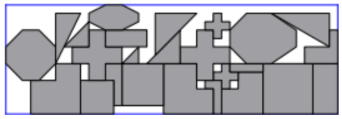

Fig. 4. Solução encontrada pela GRASP[1].

C. JACKOBS2

1) GCG:

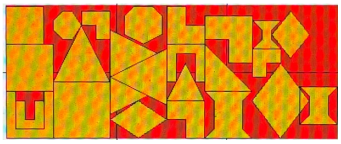

Fig. 5. Solução encontrada pela GCG.

2) GRASP:

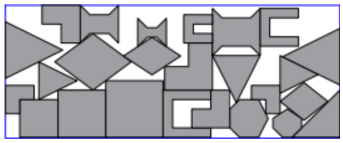

Fig. 6. Solução encontrada pela GRASP[1].

Com as imagens e, principalmente, ao analisa a figura 6, é possível perceber a capacidade do GCG em relação às peças que possuem um caráter de quebra-cabeça. Ao analisar as duas primeiras colunas, é possível além de ver encaixes quase perfeitos, também é possível entender como o algorítimo escolhe as colunas, preferindo uma coluna com muitas peças e com um índice de compacticidade alto, a uma com 100% de preenchimento da coluna e do fecho convexo, mas utilizando menos peças que a primeira coluna.

Por fim, será apresentado os resultados do modelo treinado PPO. Os testes foram fechados apenas ao dataset FU, que também sofreu algumas alterações na dimensão da área de corte. Essas particularidades serão discutidas na seção de conclusão. A tabela dos modelos PPO se estruturou dessa forma:

TABLE III
RESULTADOS PPO

Nome	Area de Corte	Utilização das peças
FU	34x38	83%
FU	41x41	100%
FU	43x43	100%

Dessa forma, devido à alta complexidade da questão abordada, os modelos não foram capazes de aprender o suficiente para posicionar todas as peças na área de corte. Será discutido na próxima seção o motivo pelo qual isso ocorreu, quando comparado à heurística GCG, e o que pode ser feito para que o modelo seja capaz resolver de forma ótima esse problema. Porém, é possível perceber que com um aumento ligeiro na área de corte, os modelos se tornam capazes de aprender a posicionar todas as peças. Essa característica abre portas de como alterar o modelo atual de treinamento para que, no futuro, a rede neural seja capaz de solucionar esse problema. Será apresentado as soluções encontradas por esses três modelos:

1) 34x38:

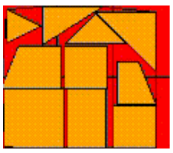

Fig. 7. PPO 34x38.

1) 41x41:

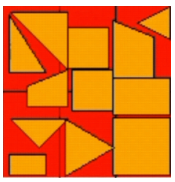

Fig. 8. PPO 41x41.

1) 43x43:

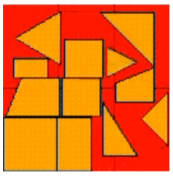

Fig. 9. PPO 43x43.

VI. CONCLUSÃO

Nesse artigo, foi apresentado o algoritimo Greedy Column Generation, uma heurística que busca soluções para o problema de corte e empacotamento 2D guiado por duas funções de custo, em que ambas utilizam a ideia do fecho convexo

para analisar a compacidade das peças. Com os resultados encontrados dessa heurística, foi possível compará-lo com o famoso e antigo método da literatura, o GRASP, em que na maior parte dos casos, ambos os algoritimos resultaram em soluções semelhantes, entretanto, o GCG pode ser avaliado de forma mais única caso as peças possuam o caráter de quebra-cabeças. Nesse sentido, ainda é possível analisar a capacidade do GCG em encontrar soluções ótimas para os mais diversos datasets da literatura.

Ademais, ao analisar os resultados do algorítimo PPO, não foi possível adquirir uma solução ótima. A complexidade do problema de corte e empacotamento, por ser altíssima, traz uma situação em que na maioria dos estados, ha um número extremamente alto de próximos estados. Tal fato atrapalha a rede neural a aprender como um todo, e em como se aproximar de uma solução para o problema. Na maioria dos treinamentos, a rede nos momentos finais de treinamento estagna em máximos locais, momento em que a rede para de tentar melhorar a solução e considera a solução atual como ótima. Entretanto, com mais testes no modelo PPO e uma mudança na política de recompensa, como a adição de uma penalidade a mais para a sobreposição de peças, e a implementação da técnica de curriculum learning — que consiste em aumentar gradualmente a dificuldade do ambiente, começando por um mais simples — possam permitir alcançar o resultado esperado.

O ponto-chave da análise entre o GCG e o PPO é a dificuldade encontrada para que ambos os algoritmos pudessem encontrar uma solução. Como o GCG se enquadra em um algorítimo de busca, é extremamente mais fácil entender o motivo pelo qual o algorítimo toma a escolha X ao invés da Y, principalmente pelo fato do desenvolvedor possuir total controle sobre todas as escolhas do algorítimo, fora a particularidade visual desse problema, em que é possível analisar visivelmente o porquê de uma escolha não ser tão boa. Em contrapartida, pela alta complexidade das redes neurais, e principalmente pela complexidade do algorítimo escolhido, o Proximal Policy Optimization (PPO), ao tentar solucionar um problema de alto nivel, como o problema de corte e empacotamento bidimensional, a maioria das escolhas do algorítimo é feito de modo probabilístico e estatístico, perdendo a noção onde o algorítimo esta ”errando”, fora o fato de, para testar uma simples mudança, é necessário realizar novamente todo o treinamento da rede para no final poder analisar se a mudança foi positiva ou negativa. Dessa forma, o tempo para que se encontre quais mudanças devem ser feitas no algorítimo aumenta, dificultando ainda mais em como encontrar a melhor estrutura para o algorítimo.

De certa forma, o ponto positivo de se utilizar de redes neurais é a capacidade de generalização que elas possuem, que em certas ocasiões é extremamente necessário, mas para que isso seja possível, a forma como estruturar o algorítimo é extremamente complexa. Entretanto, pelo fato do problema de corte e empacotamento ser um problema complexo, isso não implica que sua solução também deva ser, como foi visto nesse artigo e em muitos outros que acharam soluções para esse problema mesmo antes de existir as redes neurais, o importante sempre será analisar o problema em específico e decidir qual método deverá ser usado.

VII. REFERENCIAS

REFERENCES

[1] Del Valle, A. M., Queiroz, T. A., Miyazawa, F. K., e Xavier, E. C. (2012). Heuristics for Two-Dimensional Knapsack and Cutting Stock Problems with Items of Irregular

[2] Xu, Y. (2016). An Efficient Heuristic Approach for Irregular Cutting Stock Problem in Ship Building Industry. Mathematical Problems in Engineering.

[3] Pitombeira-Neto, A. R., e Murta, A. H. F. A Reinforcement Learning Approach to the Stochastic Cutting Stock Problem.

[4] Fang, J., Rao, Y., Zhao, X., e Du, B. (2023). A Hybrid Reinforcement Learning Algorithm for 2D Irregular Packing Problems. Mathematics,

[5] Fan, J., Wang, Z., Xie, Y., e Yang, Z. (2019). A Theoretical Analysis of Deep Q-Learning.

[6] Jesus, O. S. (2007). Decomposição da equação de Bellman. Tese de Doutorado, Instituto de Matemática e Estatística, Universidade de São Paulo.

[7] Gilmore, P. C., e Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem.

[8] ALBANO, Antonio. Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods. IEEE Transactions on Systems, Man, and Cybernetics, v. 10, n. 5, p. 242-248, 1980. DOI: 10.1109/TSMC.1980.4308549.