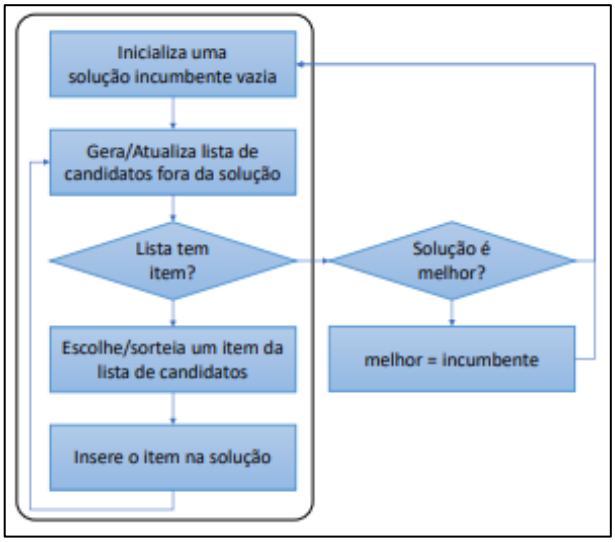
Baseado no fluxograma de heurística de busca construtiva a seguir, foi identificado no código estruturado em Python a parte do código que representa cada uma das caixas do fluxograma.



A primeira etapa de inicialização de uma solução incumbente vazia é vista em:

```
#Lista todos os itens
items = []
for item in calculado:
    items.append(item[0])

#Solução incumbente vazia
#[Peso,Lucro,[0,0,..]]
S = [0,0,[0]*n_items]
```

Gerar/Atualizar a lista de candidatos fora da solução é dividido em dois pontos:

- Cálculo de eficiência:

```
def search_construtitve(self,TAMANHO_LISTA,TAMANHO_MINIMO,sementes)
:
        #Define a lista de dados de entrada para calculado
        calculado = list(self.dados_entrada)

        #Calcule a eficiência de cada item
        for item in calculado:
            item.append(item[2]/item[1])

        #Ordene em ordem decrescente de eficiência
        calculado.sort(reverse=True,key=lambda item:item[3])
```

- Atualização lista de candidatos:

```
while(True):
                #Define número de candidatos
                n = max([math.ceil(len(items)*TAMANHHO_LISTA),TAMANHO_MINIMO])

                #Define lista de candidatos
                k = 0 #Número de candidatos definidos
                candidatos = []
                for i in items:
                    if self.dados_entrada[i-1][1] + S[0] <= peso_maximo:
                        k = k + 1
                        candidatos.append(i)
                        if k == n: break
```

Ponto de decisão “Lista tem item?” é feito através de uma condicional

```
if k == 0: break
```

Se houver itens, há a escolha ou sorteio de um item da lista de candidatos:

```
#Sorteia item da lista de candidatos
                escolhido = random.choice(candidatos)
```

Inserção de itens na solução:

```
#Atualiza a solução incunmbente
                items.pop(items.index(escolhido))
                S[2][escolhido-1] = 1
                S[0] += self.dados_entrada[escolhido-1][1]
                S[1] += self.dados_entrada[escolhido-1][2]
```

Se não houver mais itens na lista, se tem outro ponto de decisão “Solução é a melhor?”

```
if S[1] > BEST[1]:
```

Se for a melhor solução, a lista melhor recebe a lista incumbente:

```
BEST = S
                #print(S,"Melhor")
            else:
                pass
                #print(S)
```

A heurística construtiva tem por ideia construir uma solução passo a passo, elemento por elemento. No problema da mochila a heurística é adicionada à mochila a cada etapa o objeto mais valioso por unidade de peso e que não ultrapasse a capacidade máxima da mochila. É necessário

que a estrutura seja organizada de modo que facilite o recebimento dos itens que sejam selecionados de acordo com uma a lista de candidatos com um tamanho definido.

No caso do problema da mochila, são listados todos os itens e cria-se uma estrutura de lista vazia e então é realizada a ordenação decrescente dos valores das eficiências calculadas e é acrescentada nessa lista vazia os “N” melhores itens.

A partir da lista de candidatos uma forma de escolha mais comum é o sorteio de um item para ser inserido na solução. Quando não for mais possível obter uma lista de candidatos pelos critérios e restrições definidas, termina-se a geração da solução incumbente. Então verifica-se, assim, se a solução é melhor em relação às soluções já visitadas e se for ótima, então a incumbente se torna a melhor.

Quando se analisa o problema do caixeiro viajante, é visto que de maneira geral ele possui uma estrutura muito similar ao problema da mochila, e se adequa muito bem ao método de solução da heurística construtiva. Mas comparando ao código estruturado em python para atender ao problema da mochila que foi disponibilizado, são necessários alguns pontos de adequação para que o código atenda ao problema do caixeiro viajante.

Entre esses pontos pode se destacar a troca dos inputs, que deixam de ser os pesos e lucros e passam a ser aos pontos cartesianos X e Y de cada cidade. O cálculo da eficiência é trocado pelo cálculo das distâncias, e a ordenação é feita buscando as menores distância entre as cidades.

No problema do caixeiro viajante também é necessário a definir um ponto de partida, realizar os cálculos da distância do ponto para todos os outros pontos, selecionar qual é o próximo ponto através da ordenação crescente das distâncias, calcular a distância do último ponto até o primeiro e por fim somar a distância total percorrida, e solução ótima será a que tiver menor valor.