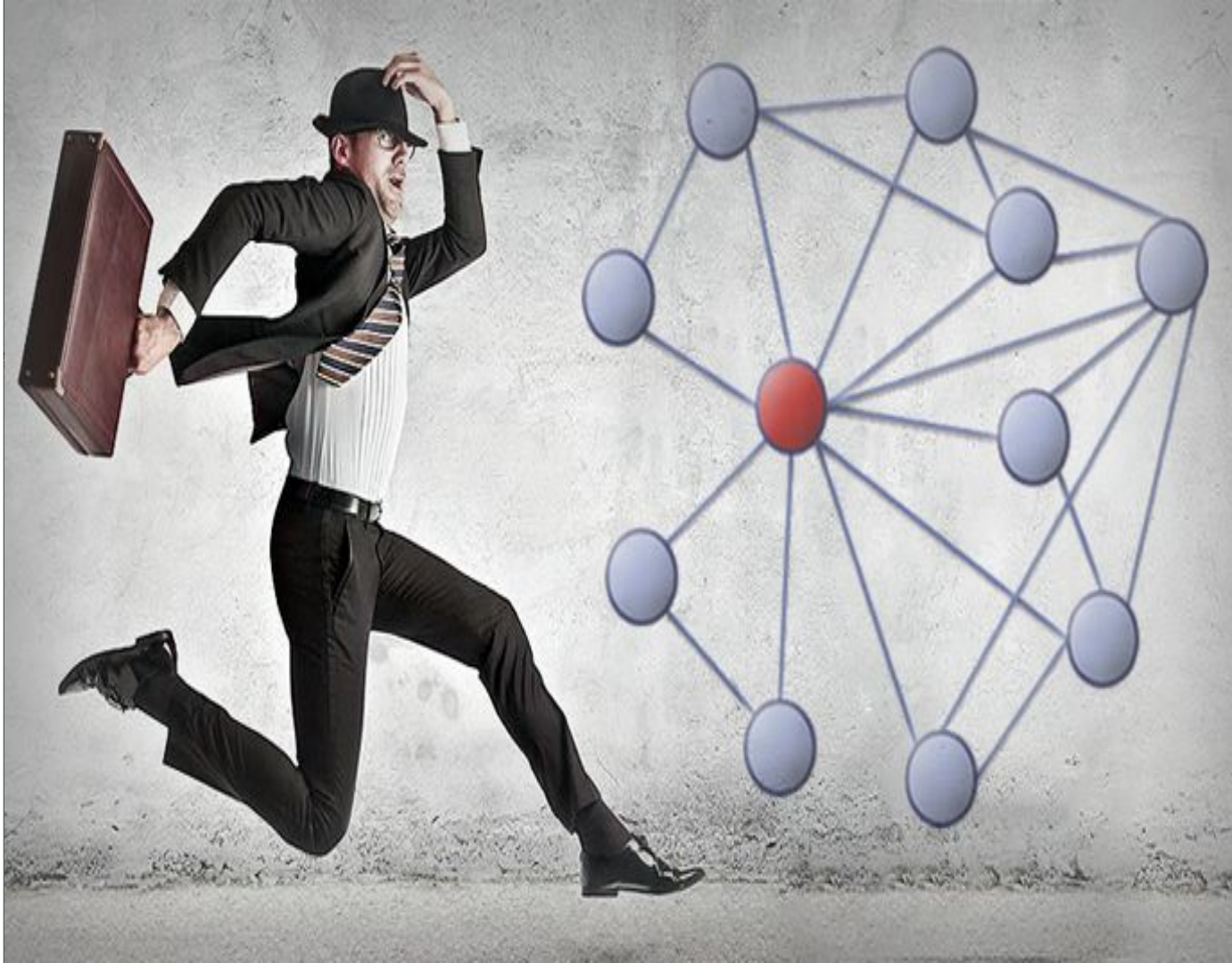
Uso de FPGA para o cálculo da função objetivo do Traveling Thief Problem

Rodolfo Pereira Araujo

IME00706(4) - Tópicos Especiais: Sistemas Embarcados

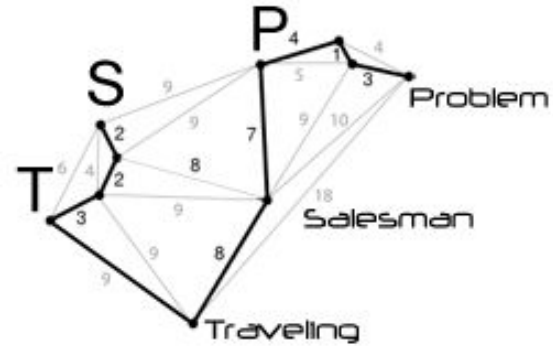
1.

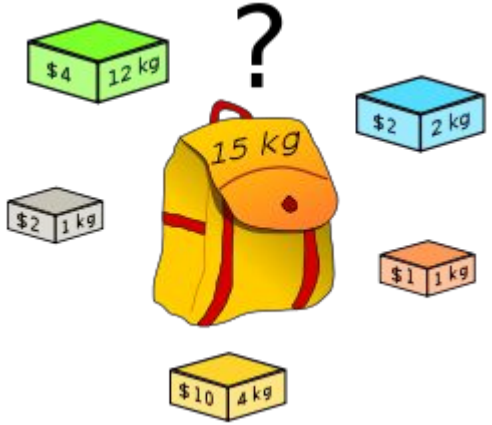
Traveling Thief Problem



Traveling Thief Problem

Problema do Mochileiro Viajante





É dado um conjunto de cidades, cada cidade contém um grupo de itens a serem coletados.
Cada item possui um peso e um valor associados, a quantidade de itens coletados é limitada pela capacidade da mochila.
O custo da rota é o produto da taxa de aluguel pelo tempo necessário para percorrê-la, que por sua vez varia conforme a carga na mochila por limitar a velocidade do carro.

Traveling Thief Problem

- Coletar um item implica em:
  - Aumentar o valor coletado;
  - Aumentar o peso transportado;
  - Diminuir a velocidade do veículo.

Traveling Thief Problem

- Coletar um item implica em:
  - Aumentar o valor coletado;
  - Aumentar o peso transportado;
  - Diminuir a velocidade do veículo.

⇒ Diminuir o valor da função objetivo.

Traveling Thief Problem

$$\max f(x,y) = \sum_{i=1}^{n}\sum_{k=1}^{m_i} p_{i,k}\ y_{i,k} - R\left(\sum_{i=1}^{n-1}\frac{d_{x_i,x_{i+1}}}{v_{max}-\nu W_{x_i}} + \frac{d_{x_n,x_1}}{v_{max}-\nu W_{x_n}}\right)$$

$$\sum_{i=1}^{n}\sum_{k=1}^{m_i} w_{i,k}\ y_{i,k} \leq W \qquad \nu = \frac{v_{max}-v_{min}}{W}$$

Traveling Thief Problem

Soma do valor dos itens

Custo da rota

$$\max f(x,y) = \sum_{i=1}^{n}\sum_{k=1}^{m_i} p_{i,k}\ y_{i,k} - R\left(\sum_{i=1}^{n-1}\frac{d_{x_i,x_{i+1}}}{v_{max}-\nu W_{x_i}} + \frac{d_{x_n,x_1}}{v_{max}-\nu W_{x_n}}\right)$$

$$\sum_{i=1}^{n}\sum_{k=1}^{m_i} w_{i,k}\ y_{i,k} \leq W$$

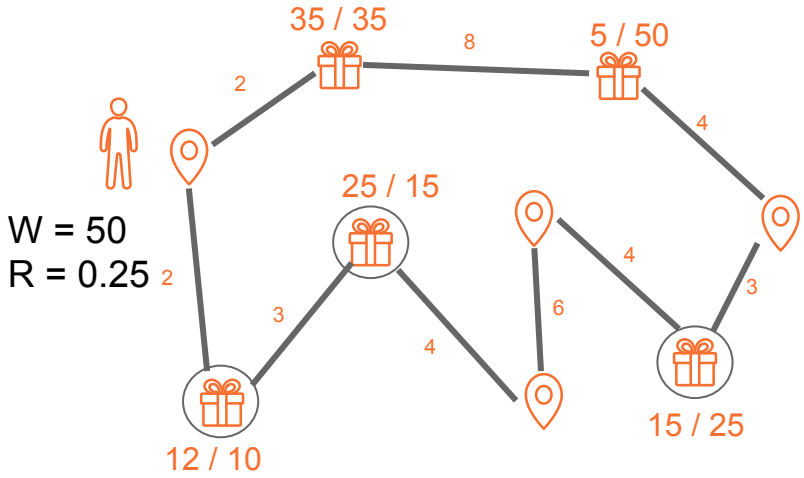
$$\nu = \frac{v_{max}-v_{min}}{W}$$

Função objetivo

A localização do item na rota influencia na velocidade deste ponto em diante, dessa forma coletar ou não um item depende no seu:

- Valor;
- Peso;
- Localização na rota

Função objetivo



d = 2 + 8 + 4 + 3 + 10 + 15 + 10 + 12 + 10 = 74

w = 0 + 0 + 0 + 0 + 25 + 0 + 0 + 15 + 10 = 50

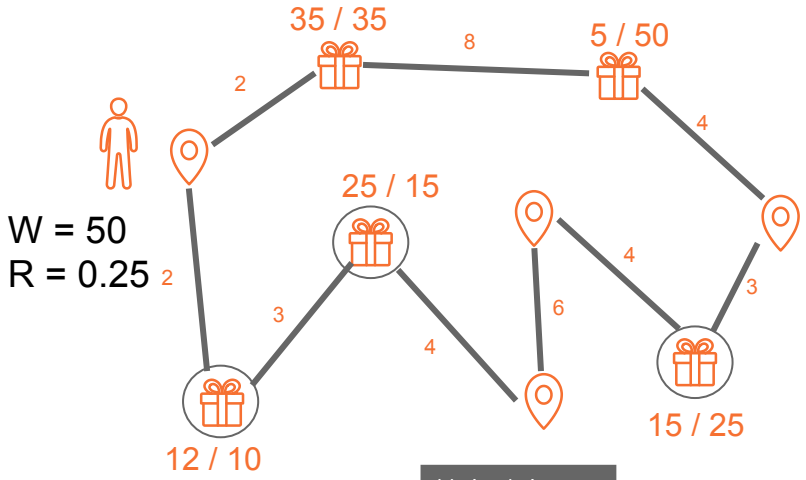
p = 0 + 0 + 0 + 0 + 15 + 0 + 0 + 25 + 12 = 52

V = 52 - 0.25x74 = **33.5**

Fixando a rota
Fixando cidade inicial
Variando itens
Melhor valor coletado
Pior função objetivo

Função objetivo

Fixando a rota
Fixando cidade inicial
Variando itens
Melhor valor coletado
Pior função objetivo



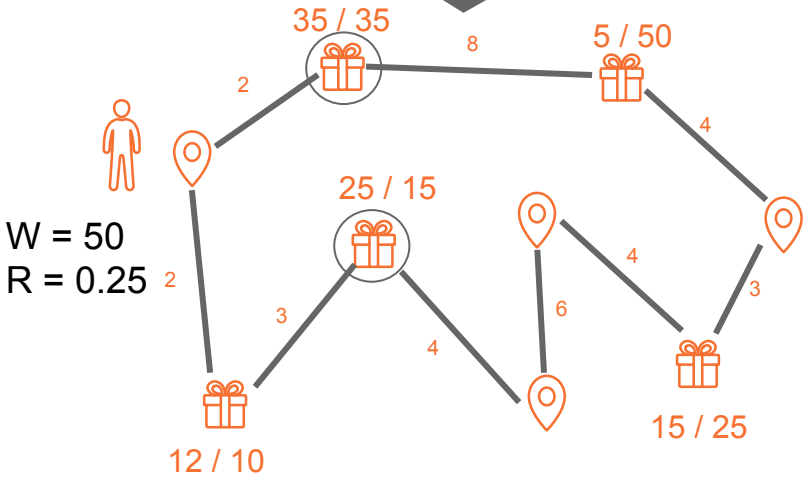
d = 2 + 8 + 4 + 3 + 10 + 15 + 10 + 12 + 10 = 74

w = 0 + 0 + 0 + 0 + 25 + 0 + 0 + 15 + 10 = 50

p = 0 + 0 + 0 + 0 + 15 + 0 + 0 + 25 + 12 = 52

V = 52 - 0.25x74 = **33.5**

Variando itens



d = 2 + 28 + 14 + 10.5 + 14 + 21 + 14 + 15 + 10 = 128.5

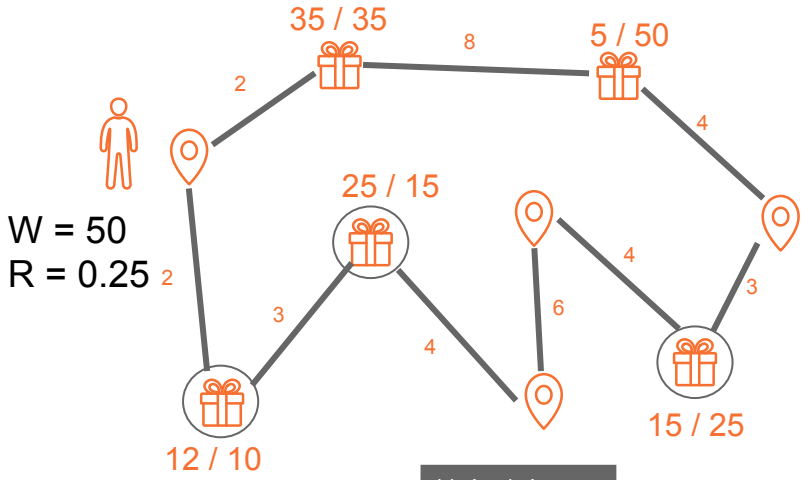
w = 0 + 35 + 0 + 0 + 0 + 0 + 0 + 15 + 0 = 50

p = 0 + 35 + 0 + 0 + 10 + 0 + 0 + 25 + 0 = 60

V = 60 - 0.25x128.5 = **27.875**

Função objetivo

Fixando a rota
Fixando cidade inicial
Variando itens
Melhor valor coletado
Pior função objetivo



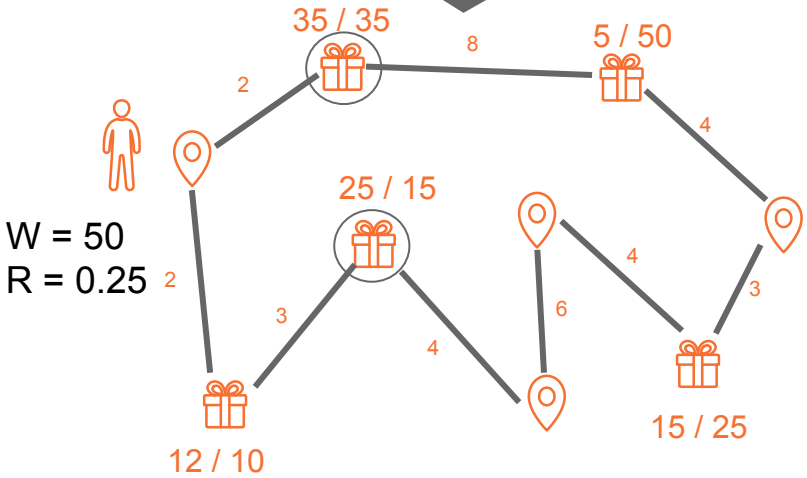
d = 2 + 8 + 4 + 3 + 10 + 15 + 10 + 12 + 10 = 74

w = 0 + 0 + 0 + 0 + 25 + 0 + 0 + 15 + 10 = 50

p = 0 + 0 + 0 + 0 + 15 + 0 + 0 + 25 + 12 = 52

V = 52 - 0.25x74 = **33.5**

Variando itens



d = 2 + 28 + 14 + 10.5 + 14 + 21 + 14 + 15 + 10 = 128.5

w = 0 + 35 + 0 + 0 + 0 + 0 + 0 + 15 + 0 = 50

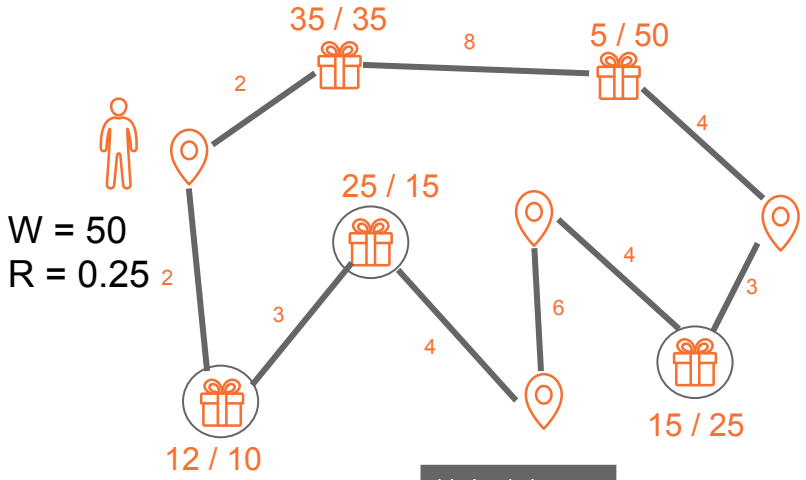
melhor

p = 0 + 35 + 0 + 0 + 10 + 0 + 0 + 25 + 0 = 60

V = 60 - 0.25x128.5 = **27.875**

Função objetivo

Fixando a rota
Fixando cidade inicial
Variando itens
Melhor valor coletado
Pior função objetivo



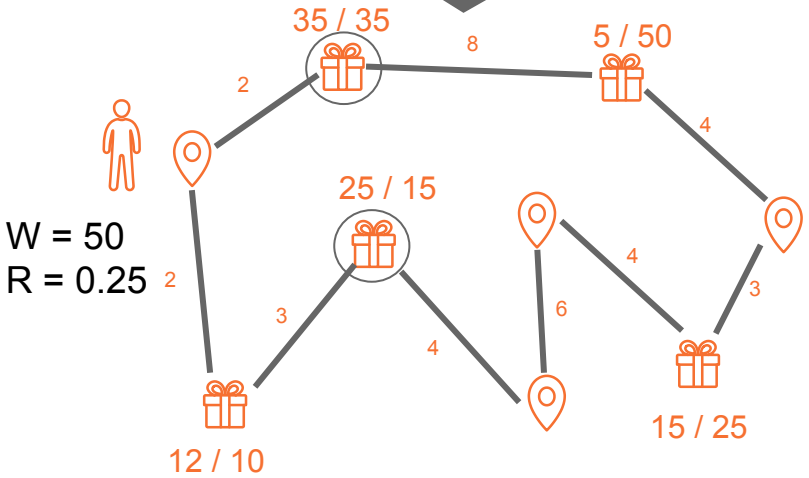
d = 2 + 8 + 4 + 3 + 10 + 15 + 10 + 12 + 10 = 74

w = 0 + 0 + 0 + 0 + 25 + 0 + 0 + 15 + 10 = 50

p = 0 + 0 + 0 + 0 + 15 + 0 + 0 + 25 + 12 = 52

V = 52 - 0.25x74 = **33.5**

Variando itens



piorou

d = 2 + 28 + 14 + 10.5 + 14 + 21 + 14 + 15 + 10 = 128.5

w = 0 + 35 + 0 + 0 + 0 + 0 + 0 + 15 + 0 = 50

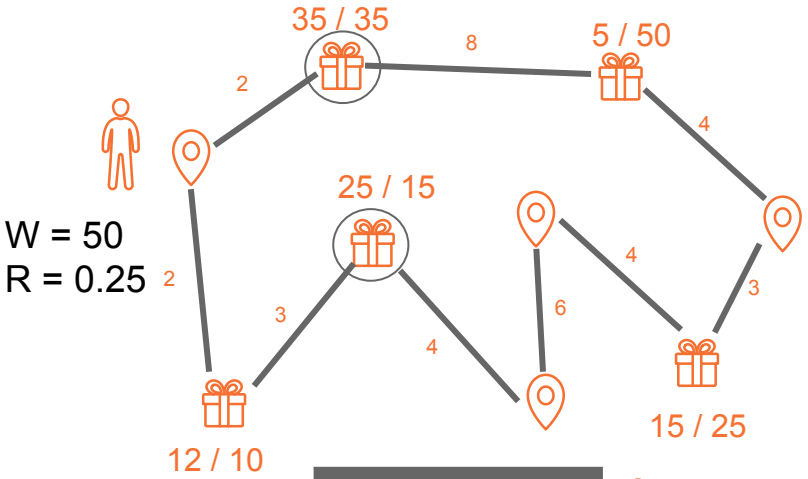
p = 0 + 35 + 0 + 0 + 10 + 0 + 0 + 25 + 0 = 60

V = 60 - 0.25x128.5 = **27.875**

piorou

Função objetivo

Fixando rota
Variando cidade inicial
Fixando itens
Mesmo valor coletado
Melhor objetivo



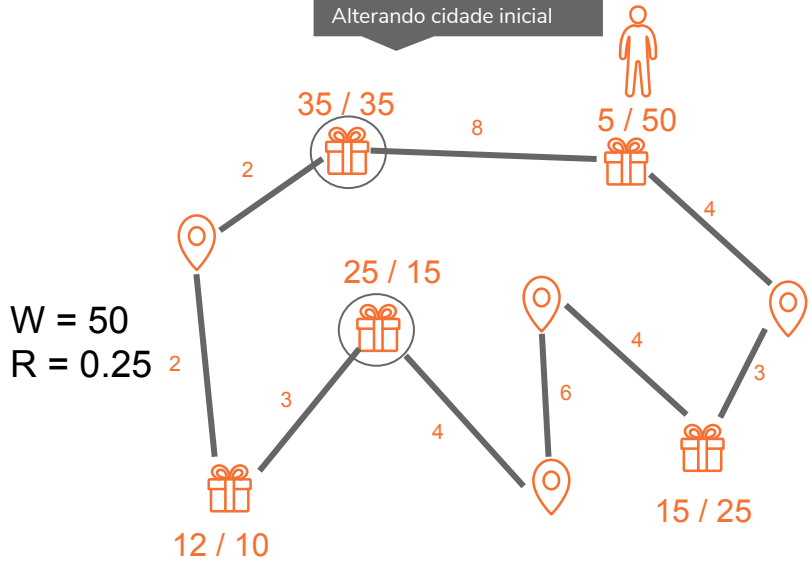
d = 2 + 28 + 14 + 10.5 + 14 + 21 + 14 + 15 + 10 = 128.5

w = 0 + 35 + 0 + 0 + 0 + 0 + 0 + 15 + 0 = 50

p = 0 + 35 + 0 + 0 + 10 + 0 + 0 + 25 + 0 = 60

V = 60 - 0.25x128.5 = **27.875**

Alterando cidade inicial



d = 4 + 3 + 4 + 6 + 4 + 4.5 + 3 + 3 + 40 = 71.5

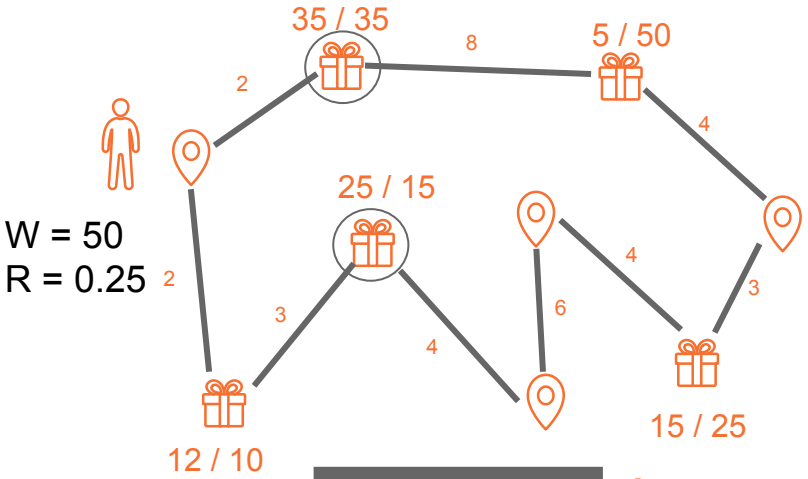
w = 0 + 0 + 0 + 0 + 0 + 15 + 0 + 0 + 35 = 50

p = 0 + 0 + 0 + 0 + 0 + 25 + 0 + 0 + 35 + 0 = 60

V = 60 - 0.25x71.5 = **42.875**

Função objetivo

Fixando rota
Variando cidade inicial
Fixando itens
Mesmo valor coletado
Melhor objetivo



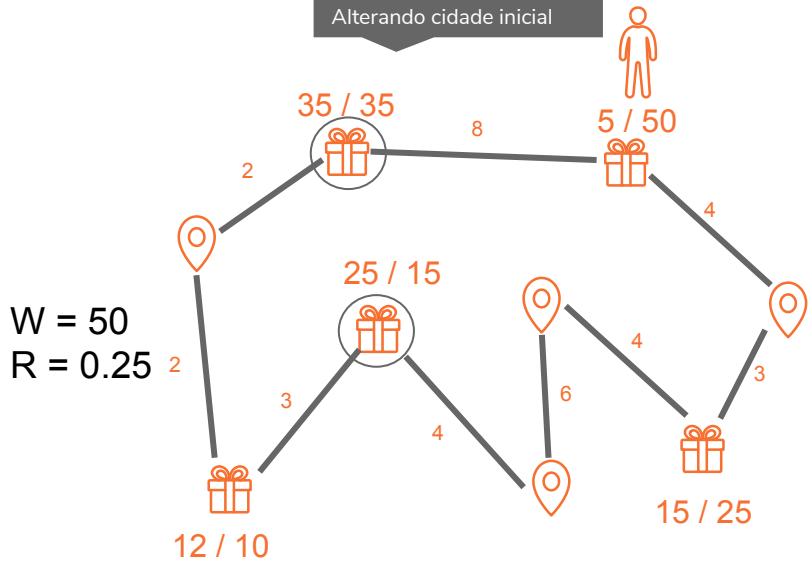
d = 2 + 28 + 14 + 10.5 + 14 + 21 + 14 + 15 + 10 = 128.5

w = 0 + 35 + 0 + 0 + 0 + 0 + 0 + 15 + 0 = 50

p = 0 + 35 + 0 + 0 + 10 + 0 + 0 + 25 + 0 = 60

V = 60 - 0.25x128.5 = **27.875**

Alterando cidade inicial



d = 4 + 3 + 4 + 6 + 4 + 4.5 + 3 + 3 + 40 = 71.5

w = 0 + 0 + 0 + 0 + 0 + 15 + 0 + 0 + 35 = 50

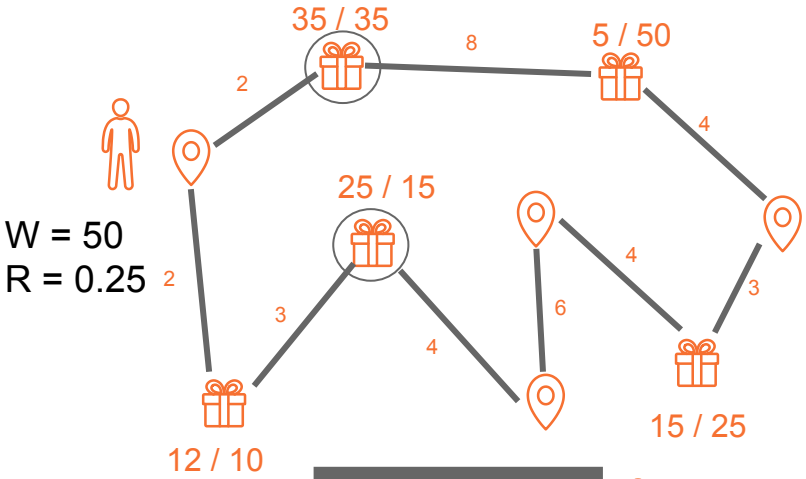
igual

p = 0 + 0 + 0 + 0 + 0 + 25 + 0 + 0 + 35 + 0 = 60

V = 60 - 0.25x71.5 = **42.875**

Função objetivo

Fixando rota
Variando cidade inicial
Fixando itens
Mesmo valor coletado
Melhor objetivo



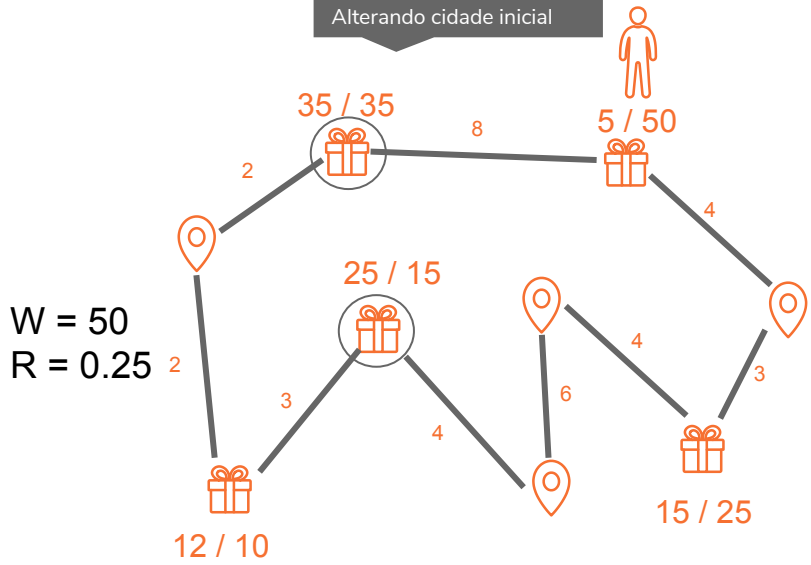
d = 2 + 28 + 14 + 10.5 + 14 + 21 + 14 + 15 + 10 = 128.5

w = 0 + 35 + 0 + 0 + 0 + 0 + 0 + 15 + 0 = 50

p = 0 + 35 + 0 + 0 + 10 + 0 + 0 + 25 + 0 = 60

V = 60 - 0.25x128.5 = **27.875**

Alterando cidade inicial



d = 4 + 3 + 4 + 6 + 4 + 4.5 + 3 + 3 + 40 = 71.5

melhor

w = 0 + 0 + 0 + 0 + 0 + 15 + 0 + 0 + 35 = 50

p = 0 + 0 + 0 + 0 + 0 + 25 + 0 + 0 + 35 + 0 = 60

melhor

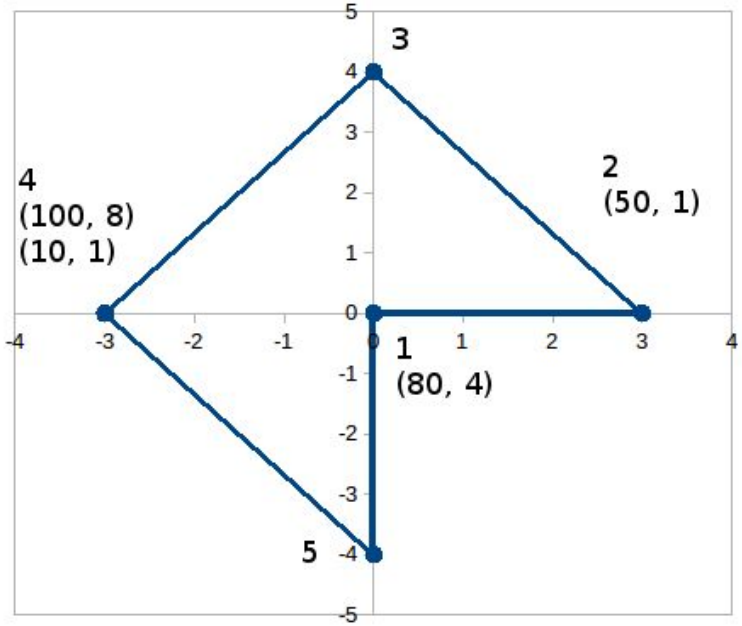
V = 60 - 0.25x71.5 = **42.875**

Valor da solução

“

Estes dois componentes foram reunidos de forma que uma solução ótima para cada problema separadamente não necessariamente corresponde para uma solução ótima do TTP.

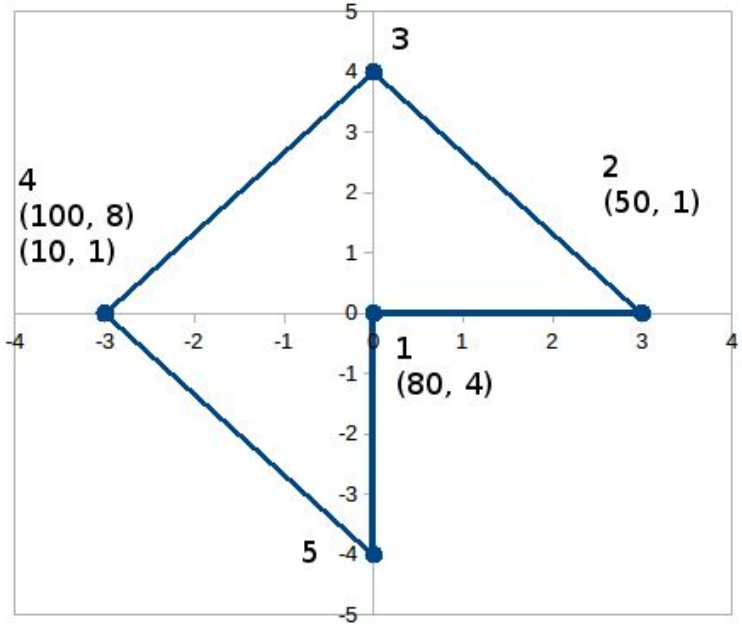
Representação da solução



rota {1, 2, 3, 4, 5} -> O(n!)

mochila {0, 1, 0, 1} -> $O(2^n)$

Representação da solução



rota {1, 2, 3, 4, 5} -> O(n!)

mochila {0, 1, 0, 1} -> $O(2^n)$

Aceita soluções inválidas

2.

Método

Método

Processador ARM chama o cálculo da função objetivo na FPGA

Calcula o valor da função objetivo

Método

```
int penalidade = -3;

double coletado = 0;
double custo = 0;
int peso = 0;

for (int j = 0; j < quantidadeItens; j++) {
    if (GET_MOCHILA_VALUE(localsearch, mochila, z, j)) {
        coletado += peso <= capacidadeMochila ? itemValor[j] : penalidade * itemValor[j];
        peso += itemPeso[j];
    }
}

peso = 0;
int indexI = GET_PERCURSO_INDEX(localsearch, x, y, 0);
int indexImaisUm;
for (int i = 0; i < quantidadeCidades; i++) {
    int inicioItensCidadeI = inicioItensCidade[i];
    int fimIntensCidadeI = inicioItensCidade[i + 1];
    indexImaisUm = GET_PERCURSO_INDEX(localsearch, x, y, (i + 1) % quantidadeCidades);
    for (int k = inicioItensCidadeI; k < fimIntensCidadeI; k++) {
        int j = indiceItemCidade[k];

        if (GET_MOCHILA_VALUE(localsearch, mochila, z, j)) {
            peso += itemPeso[j];
        }
    }
    custo += DIST_PONTO(percurso[indexI], percurso[indexImaisUm])
            / VELOCIDADE(peso, capacidadeMochila, velocidadeMinima, velocidadeMaxima);

    indexI = indexImaisUm;
}

*resposta = coletado - aluguel * custo;
```

3. Resultados

Resultados

Resultado da solução

CPU: -52681,169010

FPGA: -52681,169011

Diferença: $1{,}898 \times 10^{-9}\%$

Tempo (ciclos)

ARM: 274.792

FPGA: 1.676.124

Speed up negativo de 6,100

Resumo dos dados

$1{,}8982 \times 10^{-9}\%$

Diferença no valor da solução

-6,100

Speedup negativo

5.

Trabalhos futuros

Trabalhos futuros

Apesar do tempo não ter melhorado com o uso da FPGA existem possibilidades de melhoria

Utilizar memcpy para diminuir acesso ao barramento

4m + 6n acessos
percurso 2m acessos
mochila n acessos
itemValor 2n acessos
itemPeso 2n acessos
inicioItensCidade 2m acessos
indiceItemciade n acessos

+ FPGA
+ Otimização

Granularidade do código pode ter sido pequena, atual é O(n+m).

Utilizar diretivas de otimização do HLS como
#prama HLS unroll


Obrigado