Problema: *Set Packing Problem*

- Consiste em maximizar a métrica de **silhueta** da clusterização dos dados, feita segundo um conjunto c de atributos
- Deseja-se encontrar qual é o conjunto $c \subset S$ que maximize a silhueta, onde:
  - S é o conjunto completo, com todos atributos da base
  - $tam\_min \leq tam(c) \leq tam\_max$, parâmetros informados pelo usuário
- Ainda, c está sujeito a restrições que consistem em conjuntos de itens que não podem ocorrer simultaneamente na solução do problema
- Dois itens não poderão ocorrer simultaneamente em c se sua métrica de correlação for superior a um limite th.
- Nos testes feitos, dois itens el_a e el_b não poderão ocorrer simultaneamente se $corr(el_a, el_b) \geq th = 0,75$

Problema: *Set Packing Problem*

- As restrições produzidas serão conjuntos de pares de atributos que não podem estar presentes simultaneamente numa solução
- Além destas restrições há a restrição dos tamanhos máximo e mínimo de uma solução
- O procedimento construtivo do GRASP proposto garante a geração somente de soluções viáveis
- O procedimento de busca local do GRASP penaliza soluções inviáveis de acordo com o número de restrições de pares de conjuntos violadas
- Se a restrição de tamanhos máximo e mínimo for violada, aplica-se a penalidade como se a solução tivesse violado todos os pares de restrições, mais um (objetivo: evitar ao máximo estas soluções)

Custo de Inserção na Solução

- O custo $C_{ins}(i)$ de inserção de um item i em uma solução é dado por:

$$C_{ins}(i) = \frac{10 \cdot var(i)}{1 + NR(i)}$$

- onde:
  - $var(i)$ é a variância do item (atributo) i na base de dados
  - $NR(i)$ é o número de restrições em que i está envolvido

Penalização de Soluções Inviáveis

- O custo de uma solução c é composto pela métrica base da silhueta (sil) diminuída de um fator de penalidade (pen):

$$C_{sol}(c) = sil(k\text{-}means(c)) - pen(c)$$

- com:

$$pen(c) = log(1 + NV(c))$$

- sendo $NV(c)$ o número de restrições violadas por c (se limites de tamanho, todas $+1$)
- Quando $NV(c) = 0$, $pen(c) = 0$

Algoritmo 1: GRASP para Seleção de Atributos (*Set Packing Problem*)

entrada: L: lista de itens (atributos)
entrada: R: matriz de restrições
entrada: tam_min: tamanho mínimo para um conjunto de atributos viável
entrada: tam_max: tamanho máximo para um conjunto de atributos viável
entrada: α: fator de flexibilização da estratégia gulosa
entrada: t_e: tamanho do conjunto de elite
entrada: M: número máximo de iterações
entrada: m: número máximo de iterações sem melhorias na elite ou melhor solução encontrada

saída : e: conjunto de elite, com as t_e melhores soluções encontradas

Inicializa e;
$i \leftarrow 0$;
enquanto $i < M$ **faça**
  $c \leftarrow$ procedimento_construtivo(L, R, tam_min, tam_max, α);
  $e \leftarrow$ atualiza_elite(c, e, t_e);
  $c \leftarrow$ busca_local(c, L, tam_min, tam_max, m);
  $e \leftarrow$ atualiza_elite(c, e, t_e);
  se *m iterações sem melhorias* **então**
    Interrompe;
  fim se
  $i \leftarrow i + 1$;
fim enqto
retorna e

Algoritmo 2: GRASP: Procedimento Construtivo

entrada: L: lista de todos os itens, R: matriz de restrições, *tam_min* e *tam_max*: tamanhos mínimo e máximo para c viável, α: fator de flexibilização da estratégia gulosa
saída : c: solução candidata

$LRC \leftarrow \alpha\%$ itens i de L com menor custo de inserção $C_{ins}(i)$;
$c \leftarrow \emptyset$;
enquanto $LRC \neq \emptyset$ **faça**
  el $\leftarrow$ item retirado aleatoriamente de LRC;
  $c \leftarrow c \cup \{el\}$;
  se $tam(c) = tam\_max$ **então**
    Interrompe laço;
  fim se
  `// O passo abaixo garante a construção somente de soluções viáveis`
  Remove de LRC todos os itens ainda presentes que encontram-se em pares de R conjuntamente com *el*;
  $LRC \leftarrow \alpha\%$ itens i de LRC com menor custo de inserção $C_{ins}(i)$;
fim enqto
se $tam(c) < tam\_min$ **então**
  Reiniciar processo todo;
fim se
retorna c

Algoritmo 3: GRASP: Busca Local

entrada: c: solução candidata, L: lista de todos os itens, tam_min e tam_max: tamanhos mínimo e máximo para c viável, m: número máximo de iterações sem melhorias
saída : c: solução candidata

```
i ← 0;
enquanto i < m faça
    viz ← cópia de c;
    se tam(c) ≤ tam_min então
        Acrescenta um elemento aleatoriamente em viz;
    fim se
    senão se tam(c) ≥ tam_max então
        Retira um elemento aleatoriamente em viz;
    fim se
    senão se i está suficientemente longe de m então
        Sorteia item el de L e retira el de viz, se estiver presente ou acrescenta el em viz, caso contrário;
    fim se
    senão
        Sorteia item ela de L e retira el_a de viz, se estiver presente ou acrescenta el_a em viz, caso contrário;
        Sorteia item elb de L e retira el_b de viz, se estiver presente ou acrescenta el_b em viz, caso contrário;
    fim se
    se C_sol(viz) > C_sol(c) então
        c ← viz;
        i ← 0;
    fim se
    senão
        i ← i + 1;
    fim se
fim enqto
retorna c
```

Algoritmo 4: GRASP: Atualização da Elite

entrada: c: solução candidata, e: conjunto elite, t_e: tamanho do conjuto elite
saída : e: elite atualizada

se $c \notin e$ **então**
    se $tam(e) < t_e$ **então**
        Acrescenta c em e;
    fim se
    senão se $C_{sol}(c) > min(\{C_{sol}(x) \quad \forall x \in e\})$ **então**
        Remove x com pior custo de e;
        Acrescenta c em e;
    fim se
fim se
retorna e