4
Abordagem por Metaeurísticas

4.1
Representação de Soluções

As três metaeurísticas implementadas utilizam a mesma representação de soluções e o mesmo tipo de vizinhança.

Representamos uma solução x para o problema de PHCPM como um vetor de pares ordenados $x = ((p_1, r_1), \ldots, (p_n, r_n))$ com $|\mathcal{E}|$ posições, onde p_e e r_e indicam o período e a sala onde o evento e foi alocado.

Podemos definir movimentos s que levam de uma solução x a outra $x' = s(x)$. A *vizinhança* de uma solução, denotada por $\mathcal{N}(x)$, é composta por todas as soluções que podem ser obtidas pela aplicação de um único movimento. Um movimento de *1-troca*, por exemplo, consiste na remoção de um único evento de sua alocação corrente a um período e sala (p_e, r_e) e alocação a um novo par (p'_e, r'_e). Nas metaeurísticas implementadas, consideramos somente a vizinhança de 1-troca.

Utilizamos duas medidas para avaliação da qualidade de uma solução. A primeira delas é a quantidade de violações de restrições fortes, HCV (*Hard Constraint Violations*, ver seção 2.2.2), que diz respeito a viabilidade de uma solução: uma solução é considerava viável se e somente se $HCV = 0$. A segunda medida é a quantidade de violações de restrições fracas, SCV (*Soft Constraint Violations*, ver seção 2.2.3), que diz respeito a qualidade de uma solução viável. Estas duas medidas podem ser utilizadas para compor a função de custo das metaeurísticas conforme seja conveniente.

4.1.1
Estruturas auxiliares para cálculo de violações fortes

Mantemos três matrizes auxiliares para facilitar o cálculo da quantidade de violações de restrições fortes:

– $OrderingProblems[e, p]$ indica quantos problemas de ordenação em relação ao evento e ativam-se quando o evento e está alocado ao período

PUC-Rio - Certificação Digital Nº 0711329/CA

p, ou seja, a quantidade de eventos em $Pred(e)$ alocados após p mais os eventos em $Succ(e)$ alocados antes de p.

– $StudentClashes[e, p]$ indica quantos eventos que tem alunos em comum com o evento e estão alocados ao período p.

– $Ocupation[p, r]$ indica quantos eventos estão alocados ao período p e sala r.

Com o auxílio de tais estruturas, podemos calcular a quantidade total de violações de restrições fortes em tempo $O(|\mathcal{E}|)$.

$$OrderingProblems = \frac{\sum_{e \in \mathcal{E}} OrderingProblems[e, p_e]}{2}$$
$$StudentClashes = \frac{\sum_{e \in \mathcal{E}} StudentClashes[e, p_e]}{2}$$
$$RoomClashes = \frac{\sum_{e \in \mathcal{E}} (Ocupation[p_e, r_e] - 1)}{2}$$

Note que os somatórios são divididos por 2 porque cada violação de restrição forte entre dois eventos é contada duas vezes. O total de violações de restrições fortes é a soma destes componentes:

$$HardCost = OrderingProblems + StudentClashes + RoomClashes$$

Ainda mais simples é o cálculo da diferença da quantidade de violações de restrições fortes causada por um evento u movendo-se do período p_e e sala r_e para o período p'_e e sala r'_e. Podemos realizar tal cálculo em tempo $O(1)$.

$$\Delta^{OrderingProblems} = OrderingProblems[e, p'_e] - OrderingProblems[e, p_e]$$
$$\Delta^{StudentClashes} = StudentClashes[e, p'_e] - StudentClashes[e, p_e]$$
$$\Delta^{RoomClashes} = Ocupation[p'_e, r'_e] - (Ocupation[p_e, r_e] - 1)$$

A diferença da quantidade de violações de restrições fortes é simplesmente a soma destes três componentes:

$$\Delta^{HardCost} = \Delta^{OrderingProblems} + \Delta^{StudentClashes} + \Delta^{RoomClashes}$$

Ao realizar efetivamente um movimento do evento u, movendo-o do período p_u e sala r_u, para o período p'_u e sala r'_u, estas estruturas de dados são atualizadas.

Para cada evento v com estudantes em comum com u (ou seja, $v \in \delta(u)$), a quantidade de conflitos $StudentClashes[v, p_u]$ é reduzida uma unidade, uma vez que um dos conflitos era causado pelo evento u, que não está mais presente

PUC-Rio - Certificação Digital Nº 0711329/CA

no período p_u. Por sua vez, a quantidade de conflitos $StudentClashes[v, p'_u]$ é aumentada de uma unidade.

Se o evento u moveu-se para um período mais tarde, ou seja, $p'_u > p_u$, há mais períodos onde os eventos v que devem preceder u (ou seja, $v \in Pred(u)$) podem ser alocados sem burlar esta restrição. Consequentemente, o valor de $OrderingProblems[v, q]$, para todo período q tal que $p_u \leq q < p'_u$ é reduzido uma unidade. Por outro lado, se $p'_u < p_u$, O valor de $OrderingProblems[v, q]$ para todo período q tal que $p'_u \leq q < p_u$ é aumentado uma unidade. Raciocínio análogo pode ser utilizado para a atualização dos valores da matriz $OrderingProblems$ para os eventos v que devem suceder u (ou seja, $v \in Succ(u)$).

A atualização da matriz $Ocupation$ é mais simples, uma vez que basta reduzir uma unidade de $Ocupation[p_e, r_e]$ e aumentar de uma unidade $Ocupation[p'_e, r'_e]$.

Portanto, a atualização das estruturas de dados que auxiliam o cálculo das violações de restrições fortes, necessária quando algum evento e se move, tem um custo de ordem $O(|\delta(e)| + |Pred(e)| + |Succ(e)|)$.

4.1.2 Estruturas auxiliares para cálculo de violações fracas

Mantemos duas matrizes auxiliares para facilitar o cálculo da quantidade de violações de restrições fracas:

- $PeriodOcupation[s, p]$ indica quantos dos eventos nos quais o estudante s está matriculado ocorrem no período p, ou seja, a cardinalidade de $\{e \in \mathcal{E}^s : p_e = p\}$.
- $DayOcupation[s, d]$ indica quantos dos eventos nos quais o estudante s está matriculado ocorrem em períodos do dia d, ou seja, a cardinalidade de $\{e \in \mathcal{E}^s : Day(p_e) = d\}$.

Com o auxílio de tais estruturas, podemos calcular a quantidade total de violações de restrições fracas em tempo $O(|\mathcal{S}| \times |\mathcal{P}|)$.

$$EndOfDayEvents = \sum_{s \in \mathcal{S}} |\{p \in \{9, 18, 27, 36, 45\} : PeriodOcupation[s, p] > 0\}|$$

$$SingleEvents = \sum_{s \in \mathcal{S}} |\{d \in \mathcal{D} : DayOcupation[s, d] = 1\}|$$

$$EventSequences = \sum_{s \in \mathcal{S}} |\{p \in \mathcal{P} : |\Gamma(s, p)| = 3\}|$$

PUC-Rio - Certificação Digital Nº 0711329/CA

O conjunto $\Gamma(s, p)$ contém aqueles dentre os três períodos a partir de p que estão ocupados por eventos que o estudante s assiste, ou formalmente:

$$\Gamma(s, p) = \{k \in \{p, p+1, p+2\} : PeriodOcupation[s, k] > 0\}$$

O total de violações de restrições fracas é a soma destes componentes:

$$SoftCost = EndOfDayEvents + SingleEvents + EventSequences$$

O cálculo da diferença da quantidade de violações de restrições fracas causada por um evento e movendo-se do período p_e e sala r_e para o período p'_e e sala r'_e pode ser feito em tempo $O(|\mathcal{S}|)$. Para calcular a diferença da quantidade de violações de restrição de eventos ao fim do dia, basta observar se o período antigo e/ou o novo são os últimos do dia:

$$\Delta^{EndOfDayEvents} = \begin{cases} |\mathcal{S}^e| & \text{se } p'_e \in \{9, 18, 27, 36, 45\} \text{ e } p_e \notin \{9, 18, 27, 36, 45\} \\ -|\mathcal{S}^e| & \text{se } p'_e \notin \{9, 18, 27, 36, 45\} \text{ e } p_e \in \{9, 18, 27, 36, 45\} \\ 0 & \text{caso contrário} \end{cases}$$

Para calcular a diferença da quantidade de violações de restrição de eventos únicos no dia, basta observar se o período antigo era o único na agenda do estudante para aquele dia (o que resolverá uma violação), e se o período novo está em um dia que o estudante ainda não tem nenhum evento (o que originará uma violação):

$$\begin{aligned} \Delta^{SingleEvents} =& |\{s \in \mathcal{S}^e : DayOcupation[s, Day(p'_e)] = 0\}| \\ & - |\{s \in \mathcal{S}^e : DayOcupation[s, Day(p_e)] = 1\}| \end{aligned}$$

Para calcular a diferença da quantidade de violações de restrição de seqüências de três eventos, verificamos quantas seqüências de três eventos dependiam da alocação do evento e no período antigo - a remoção desta alocação resolverá as violações causadas por estas seqüências. Analogamente, verificamos quantas seqüências de três eventos serão criadas com a nova alocação.

$$\begin{aligned} \Delta^{EventSequences} =& \sum_{s \in \mathcal{S}^e} |\{k \in \{p'_e, p'_e - 1, p'_e - 2\} : |\Gamma^{p_e}(s, k)| = 3\}| \\ & - \sum_{s \in \mathcal{S}^e} |\{k \in \{p_e, p_e - 1, p_e - 2\} : |\Gamma(s, k)| = 3\}| \end{aligned}$$

PUC-Rio - Certificação Digital Nº 0711329/CA

O conjunto $\Gamma^q(s,p)$ contém aqueles dentre os três períodos a partir de p que estão ocupados por eventos que o estudante$s$ assiste, desconsiderando um evento no período q, ou formalmente:

$$\Gamma^q(s,p) = \{k \in \{p, p+1, p+2\} : PeriodOcupation[s,k] > 0 \\ \vee (PeriodOcupation[s,k] > 1 \wedge k = q)\}$$

Um evento no período p_e é desconsiderado porque após o movimento, o evento e que se move não estará mais neste período, descaracterizando uma sequência de três eventos que poderia envolvê-lo.

A diferença da quantidade de violações de restrições fracas é obtida pela soma destes três componentes:

$$\Delta^{SoftCost} = \Delta^{EndOfDayEvents} + \Delta^{SingleEvents} + \Delta^{EventSequences}$$

4.2 Busca Tabu aplicada ao PPHCPM

A Busca Tabu, conforme apresentada na seção 3.1, é uma metaeurística, e portanto fornece apenas a estrutura do algoritmo, a qual devemos especializar para sua aplicação em um problema específico. Neste trabalho, implementamos uma Busca Tabu para o problema de Programação de Horários de Cursos Pós-Matrícula (PPHCPM), com objetivo principal de obtenção de soluções viáveis, ou seja, soluções que respeitam todas as restrições fortes, mas que podem burlar restrições fracas. O algoritmo implementado também tem a opção de minimizar as violações a restrições fracas, porém sem a sofisticação necessária para alcançar bons resultados neste sentido.

Utilizamos um esquema tabu não-estrito com tamanho do conjunto tabu variável, e representamos este conjunto indiretamente, através de um vetor $TabuEnd$ que associa cada movimento s à iteração $TabuEnd(s)$ na qual o mesmo deixa de ser tabu. O valor de *tabu tenure* τ é variável, adaptando-se ao estado da busca para incentivar a diversificação ou intensificação, conforme necessário. A adaptação de τ observa apenas o valor da função objetivo z, sendo τ incrementado progressivamente quando a flutuação de z não é significativa (ver seção 3.1).

A vizinhança utilizada, juntamente com algumas estruturas de dados que são mantidas (ver seção 4.1), nos permite avaliar rapidamente o custo de uma troca, não sendo necessário calcular o valor total da solução após a aplicação de cada movimento.

O PPHCPM estabelece três tipos de restrições fortes (*hard constraints*)

PUC-Rio - Certificação Digital Nº 0711329/CA

e três tipos de restrições fracas (*soft constraints*) (ver seção 2.3), e utilizamos a quantidade de violações destas restrições para compor o custo de uma solução. Utilizamos duas diferentes funções de custo, de acordo com a fase da busca. Na primeira fase, onde o objetivo é obter uma solução viável (sem violação de restrições fortes), a função de custo é simplesmente a quantidade de violações de restrições fortes:

$$f(x) = HCV(x)$$

Já na segunda fase, temos como objetivo a resolução de restrições fracas, sem no entanto violar restrições fortes, portanto utilizamos a função de custo:

$$f(x) = (\alpha \times HCV(x)) + SCV(x)$$

onde a constante α é grande o suficiente para que a violação de uma única restrição forte seja mais custosa que todas as violações de restrições fracas.

Com esta implementação de Busca Tabu, atingimos o objetivo de gerar soluções viáveis para todas as instâncias consideradas nesta dissertação, conforme reportado na seção 6.

4.3 Simulated Annealing aplicado ao PPHCPM

A estratégia de *Simulated Annealing* apresentada na seção 3.2 é uma metaeurística, e portanto fornece apenas a estrutura do algoritmo, a qual devemos especializar para sua aplicação em um problema específico.

A representação de soluções e vizinhança utilizada foram apresentadas na seção 4.1), e assim como na Busca Tabu, as estruturas de dados utilizadas nos permitem avaliar rapidamente o custo de um movimento.

A função de custo é a mesma utilizada na Busca Tabu. Quando temos o objetivo de obter viabilidade, a função de custo é a quantidade de violações de restrições fortes, $f(x) = HCV(x)$. De posse de uma solução viável, o objetivo é minimizar as violações de restrições fracas, sem voltar a violar restrições fortes, resultando na função de custo $f(x) = (\alpha \times HCV(x)) + SCV(x)$, fazendo α é grande o suficiente para o objetivo desejado.

O esquema de resfriamento que utilizamos *não* é baseado em uma variável representando uma temperatura que decai progressivamente, como é comum em implementações tradicionais de *simulated annealing*. O que fazemos é vincular a probabilidade de aceitação de movimentos sub-ótimos ao quociente entre a iteração atual e a iteração final (pré-determinada), fazendo com que no início do algoritmo a probabilidade de aceitação de movimentos sub-ótimos seja maior, priorizando a *diversificação*, e conforme o algoritmo avança, esta

PUC-Rio - Certificação Digital Nº 0711329/CA

probabilidade decai, favorecendo a *intensificação*.

Se um movimento tem custo $\delta < 0$, ele é prontamente aceito. Caso contrário, ele tem uma probabilidade positiva de ser aceito, que dependerá tanto do custo do movimento quanto da iteração atual.

A probabilidade de aceitação de um movimento de custo mediano, $\tilde{\delta}$, é dada pela expressão a seguir, onde t indica a iteração atual e T indica a iteração final.

$$P(\tilde{\delta}) = e^{-\frac{kt}{T}}$$

Já a probabilidade de aceitação de um movimento sub-ótimo de custo δ em geral é dada em função de δ, do custo mediano $\tilde{\delta}$ e de $P(\tilde{\delta})$:

$$P(\delta) = \frac{\tilde{\delta} \times P(\tilde{\delta})}{\delta}$$

Desta forma, quando maior for o custo do movimento sub-ótimo, menor é a probabilidade do mesmo ser aceito.

Este esquema de resfriamento e aceitação de movimentos sub-ótimos nos permite obter um comportamento similar ao resfriamento geométrico (ver seção 3.2.2), porém permitindo a definição a priori da quantidade de iterações do algoritmo, garantidamente passando por todas as fases do algoritmo (diversificação e intensificação).

PUC-Rio - Certificação Digital Nº 0711329/CA