Projeto de Algoritmos Baseados em Florestas de Posets para o Problema de Otimização U-curve

Monografia apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
aprovação em MAC499 – Trabalho
de
Conclusão de Curso

Aluno: Gustavo Estrela de Matos

Orientador: Marcelo da Silva Reis

Centro de Toxinas, Resposta-imune e Sinalização Celular (CeTICS)

Laboratório Especial de Ciclo Celular, Instituto Butantan

São Paulo, 11 de Novembro de 2017

Resumo

A fazer.

Conteúdo

Capítulo 1

Introdução

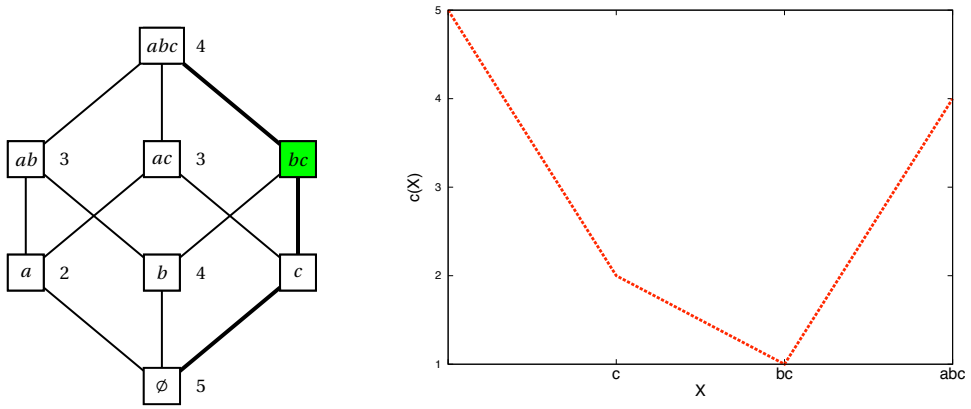
Seleção de características é uma técnica que pode ser utilizada em uma das etapas da construção de um modelo de aprendizado de máquina. Ela consiste em, dado o conjunto de características observadas nas amostras, escolher um subconjunto que seja ótimo de acordo com alguma métrica. Devemos considerar o uso de seleção de características quando a quantidade de características é muito grande, o que pode tornar o uso do modelo muito caro do ponto de vista computacional. Outra aplicação dessa técnica é em situações nas quais a quantidade de amostras é pequena comparada à complexidade do modelo original, em outras palavras, quando ocorre sobreajuste (do inglês, *overfitting*).

Mais formalmente, o problema de seleção de características consiste em um problema de otimização combinatória em que, dado um conjunto S de características, procuramos por um subconjunto $X \in \mathcal{P}(S)$ ótimo de acordo com uma função de custo $c : \mathcal{P}(S) \to \mathbb{R}_+$. É comum nas abordagens do problema explorar o fato de que o espaço de busca $\mathcal{P}(S)$ junto a relação $\subseteq$ define um reticulado Booleano [Rei12] [AG+18]. No geral, a função de custo c deve ser capaz de medir quão informativas as características X são em respeito ao rótulo Y do problema de aprendizado; portanto c costuma depender da estimação da distribuição de probabilidade conjunta de (X, Y).

Quando ocorre a estimação da distribuição de probabilidade conjunta de (X, Y), o custo das cadeias do reticulado Booleano reproduzem um fenômeno conhecido em aprendizado de máquina, o das "curvas em U". Para entender intuitivamente esse fenômeno, devemos observar que conforme subimos uma cadeia do reticulado estamos aumentando o número de características sendo consideradas, portanto existem mais possíveis valores de X, permitindo descrever melhor os valores de Y; por outro lado, também precisaríamos de mais amostras para estimar bem $\mathbb{P}(X, Y)$, e, quando isso não é possível, erros de estimação fazem com que $c(X)$, isto é, o custo de X, aumente.

Podemos então considerar um caso particular do problema de seleção de características em que a função de custo descreve "curvas em U" em todas as cadeias do reticulado Booleano. Esse caso particular é conhecido como problema U-curve e existem na literatura algoritmos ótimos para esse problema como o `U-Curve Branch and Bound (UBB)`, `U-Curve-Search (UCS)` e `Poset Forest Search (PFS)` [RFB14] [Rei12]. A solução do problema U-curve tem aplicações em problemas de aprendizado de máquina tais como como projeto de W-operadores [JCJB04] e preditores na estimação de Redes Gênicas Probabilísticas [Bar+07].

O problema U-Curve é NP-difícil [Rei12]; por conta deste fato, os algoritmos apresentados até então na literatura têm limitações tanto do ponto de vista de tempo de computação quanto do uso de memória. Dentre estes algoritmos, destacamos o PFS, que foi criado como um melhoramento do algoritmo UBB.



(a) (b)

Figura 1.1: Exemplo de instância do problema U-curve em que o conjunto de características é $S = \{a, b, c\}$. A figura 1.1(a) representa o diagrama de Hasse do reticulado Booleano $(\mathcal{P}(S), \subseteq)$, anotando ao lado de cada conjunto de características o seu custo. Os custos dos elementos da cadeia $\{\emptyset, c, bc, abc\}$, marcada em negrito, são apresentados na figura 1.1(b). O subconjunto $\{b, c\}$, marcado em verde, tem custo mínimo na cadeia em negrito e também no reticulado inteiro e, portanto é a solução ótima para esta instância. Imagem retirada de [Rei12] com permissão do autor.

O `UBB` é um algoritmo *branch-and-bound* que, a partir de uma enumeração, representa o espaço de busca como uma árvore e procura pelo mínimo global fazendo uma espécie de busca em profundidade que percorre as cadeias da árvore do espaço de busca, podando nós (e consequentemente seus descendentes) sempre que a função de custo cresce. Este algoritmo é unidirecional no sentido de que a busca em profundidade percorre as cadeias da árvore de baixo para cima, portanto se o custo dos elementos de uma cadeia nunca crescem então todos elementos desta serão visitados. A limitação deste algoritmo é evidente quando a função de custo usada é monótona não-decrescente, pois isto implica que a condição de poda nunca será verdadeira, fazendo com que todo o espaço de busca seja percorrido.

O algoritmo `PFS` contorna esta limitação porque é bidirecional. Para fazer isto, ele precisa representar o espaço de busca de duas maneiras diferentes: uma que é similar ao que o `UBB` faz, para os percorrimentos de baixo para cima, e outra que deve ser uma representação equivalente a primeira para o reticulado Booleano dual $(\mathcal{P}(S), \supseteq)$, para os percorrimentos de cima para baixo; ambas representações são feitas com florestas de posets, em uma estrutura de dados capaz de armazenar raízes e adjacências dos nós. Uma iteração do `PFS` é constituída das seguintes etapas: escolha de uma direção de percorrimento; escolha de uma raíz na floresta escolhida; ramificação (percorrimento de uma cadeia); poda na floresta escolhida; e por último, atualização da floresta dual a escolhida para que ambas representem o mesmo espaço de busca.

Existem pontos do algoritmo `PFS` que ainda não foram explorados com o intuito de melhorar seu desempenho. Dentre eles, a escolha de raízes para etapa de ramificação, que é feita de maneira arbitrária atualmente; o uso de outras estruturas de dados para representação das florestas, como por exemplo diagramas de decisão binárias ordenados (*Ordered Binary Decision Diagrams* - OBDDs) [Bry86]; e também a paralelização do código, o que parece trazer ganhos no tempo de execução do algoritmo dado que, como as árvores do espaço de busca são disjuntas, a etapa de ramificação pode ser realizada de maneira paralela com pouca informação compartilhada entre threads.

1.1 Objetivos do Trabalho

Podemos dividir os objetivos deste trabalho em objetivos gerais e específicos.

Objetivos gerais:

1. Criar algoritmos para o problema U-curve que sejam mais eficientes em consumo de tempo e/ou de memória do que as presentes soluções;
2. Verificar a qualidade das soluções encontradas no desenvolvimento de modelos de Aprendizado Computacional.

Objetivos específicos:

- Estudar o algoritmo `Poset Forest Search (PFS)`;
- Modificar a etapa de ramificação do algoritmo `PFS` e avaliar as mudanças na dinâmica do algoritmo;
- Paralelizar o algoritmo `PFS`, com as modificações feitas na etapa de ramificação (se houver melhorias com tal mudança);
- Criar um novo algoritmo, de natureza paralela e facilmente combinável com outros algoritmos, para o problema U-Curve (o algoritmo `PUCS`);
- Avaliar o consumo de recursos computacionais dos algoritmos criados, comparando com os algoritmos já presentes na literatura como o `UBB`;
- Avaliar os conjuntos de características selecionados por cada algoritmo na seleção de modelos de aprendizado computacional, usando como exemplo conjuntos de dados do repositório UCI Machine Learning Repository.

1.2 Organização do Trabalho

A fazer, resumo de cada capítulo da monografia.

Capítulo 2

Conceitos Fundamentais

2.1 O problema de seleção de características

A seleção de características é um problema de otimização combinatória em que procuramos o melhor subconjunto de um conjunto de características S. O espaço de busca desse problema é o conjunto potência de S, $\mathcal{P}(S)$, que é a coleção de todos os subconjuntos possíveis de S. A função de custo desse problema é uma função $c : \mathcal{P}(S) \to \mathbb{R}_+$.

Definição 2.1.1 (Problema de seleção de características)**.** *Seja S um conjunto de características, finito e não vazio, e c uma função de custo. Encontrar $X \in \mathcal{P}(S)$ tal que $c(X) \leq c(Y)$, $\forall Y \in \mathcal{P}(S)$.*

O espaço de busca do problema de seleção de características possui uma relação de ordem parcial definida pela relação $\subseteq$, portanto este conjunto é **parcialmente ordenado (poset)**.

Definição 2.1.2. *Uma **cadeia** do reticulado booleano é uma sequência X_1, X_2, ..., X_l tal que $X_1 \subseteq X_2 \subseteq \cdots \subseteq X_l$.*

2.2 Funções de custo

Nesta seção apresentaremos as duas funções de custo mais utilizadas durante este trabalho: a entropia condicional média (MCE) e a soma de subconjuntos. A primeira foi utilizada na seleção de modelos de aprendizado, enquanto a segunda foi utilizada para criação e solução de instâncias artificiais.

2.2.1 Custo de modelos de aprendizado computacional

A função de custo utilizada na solução do problema deve, de alguma forma, refletir a qualidade do conjunto de características avaliado. Por isso, diferentes aplicações de seleção de características podem ter diferentes funções de custo. No contexto de aprendizado de máquina, uma possível função de custo é a entropia condicional média (MCE), que já foi utilizada por exemplo na construção de W-operadores [DMJ06].

Definição 2.2.1. *Dado um problema de aprendizado em que Y é o conjunto de possíveis rótulos e $W = (w_1, ..., w_n)$, com $w_i \in A_i$, é o conjunto de variáveis. Seja $W' = (w_{I(1)}, w_{I(2)}, ..., w_{I(k)})$ um conjunto de variáveis (características) escolhidas, $\mathbf{X}$ uma vetor aleatório de tamanho k com $X_j \in A_{I(j)}$, e $log0 = 0$. Então, a **entropia condicional** de Y dado $\mathbf{X} = \mathbf{x}$ é:*

$$H(Y|\mathbf{X}=\mathbf{x}) = -\sum_{y\in Y} \mathbb{P}(Y=y|\mathbf{X}=\mathbf{x}) log \mathbb{P}(Y=y|\mathbf{X}=\mathbf{x})$$

Definição 2.2.2. *Sob o mesmo contexto definido em 2.2.1, definimos a* ***entropia condicional média*** *como:*

$$\mathbb{E}[H(Y|\mathbf{X})] = \sum_{\mathbf{x}\in\mathbf{X}} H(Y|\mathbf{X}=\mathbf{x})\mathbb{P}(\mathbf{X}=\mathbf{x})$$

A função H, em teoria da informação, mede o inverso da quantidade média de informação que uma variável tem. Esta função atinge valor máximo quando a distribuição de probabilidade da variável aleatória em questão é uniforme (todos valores que ela pode assumir são equiprováveis), e tem valores baixos quando essa distribuição é concentrada.

Problemas de aprendizado em que os rótulos tem uma distribuição concentrada são mais fáceis do que os problemas em que essa distribuição é menos concentrada. Tome como exemplo o problema de classificar o lançamento de uma moeda $\mathbf{x}$ em y (cara ou coroa); se toda moeda $\mathbf{x}$ é não viciada, então a distribuição de $\mathbb{P}(y|\mathbf{x})$ é pouco concentrada, por outro lado, quando a moeda é viciada, a distribuição de $\mathbb{P}(y|\mathbf{x})$ é concentrada e é mais fácil classificar este problema. Em termos mais formais, o erro do melhor classificador do problema mais fácil é menor do que o erro do melhor classificador do problema mais difícil.

Portanto, como a função H é capaz de medir a concentração da distribuição de Y dado $\mathbf{X}=\mathbf{x}$, e quanto maior esta concentração mais fácil é o modelo de aprendizado, podemos dizer que a função de custo $\mathbb{E}[H(Y|\mathbf{X})]$ pode representar a qualidade do modelo de classificação que usa o conjunto de características de $\mathbf{X}$.

Agora, como já entendemos o funcionamento da função de custo MCE e como ela se relaciona com a qualidade do conjunto de características avaliado, vamos entender o que acontece no modelo de aprendizado e na função de custo que usamos como exemplo quando percorremos uma cadeia do reticulado.

Uma cadeia do poset pode ser vista como uma sequência de possíveis escolhas de conjuntos de características ao qual a cada passo adicionamos uma característica. Isso significa que a cada passo dado a variável $\mathbf{x}$ ganha uma componente a mais. Quando estamos no início da cadeia, poucas variáveis do problema são consideradas, portanto há uma grande abstração dos dados dos objetos sendo classificados, e conforme subimos uma cadeia, diminuímos a abstração dos dados e isso faz com que a distribuição de Y dado $\mathbf{x}$ se concentre.

Essa concentração da distribuição da probabilidade indica que o custo dos subconjuntos deve diminuir conforme subimos por uma cadeia do reticulado, ou seja, este raciocínio nos leva a pensar que adicionar características sempre melhora a classificação; de fato, o valor de $\mathbb{E}[H(Y|\mathbf{X})]$ deve diminuir (até algum ponto de saturação) conforme aumentamos o número de variáveis do problema. Mas se isso é verdade, por que fazemos seleção de características? A inconsistência entre esse raciocínio e a motivação para seleção de característica é que essa linha de raciocínio negligenciou o fato de que problemas de classificação (supervisada) dependem de uma amostra da distribuição de Y dado $\mathbf{X}=\mathbf{x}$, ou seja, não sabemos nem ao menos calcular $H(Y|\mathbf{X}=\mathbf{x})$, podemos apenas estimar o seu valor a partir da amostra.

A amostra da distribuição de Y dado $\mathbf{X}=\mathbf{x}$ é obtida do conjunto de treinamento do problema de aprendizado e quando o número de amostras não é grande o suficiente a qualidade do classificador é comprometida. Além disso, o número de amostras necessárias deve crescer conforme aumentamos a complexidade do modelo de aprendizado utilizado. Considerando que quando subimos uma cadeia do reticulado booleano estamos aumentando a complexidade do modelo, temos que, a partir de um certo ponto, a qualidade do classificador que utiliza tal conjunto de características deve piorar.

Portanto, é esperado que a função de custo descreva um formato de U nas cadeias do reticulado. No começo da cadeia, o custo deve diminuir por conta da maior granularidade dos dados de entrada, até algum ponto onde a limitação no número de amostras combinada com o aumento da complexidade do modelo causem erros de estimação que aumentam o erro do classificador criado em tal modelo.

No cálculo da entropia condicional média, o efeito do aumento da complexidade de $\mathbf{X}$ é a estimação ruim de $\mathbb{P}(Y = y|\mathbf{X} = \mathbf{x})$. Contorna-se este problema modificando a entropia condicional média para penalizar a entropia de Y quando $\mathbf{x}$ foi observado poucas vezes. A função de custo utilizada é, então:

$$\hat{\mathbb{E}}[H(Y|\mathbf{X})] = \frac{N}{t}\sum_{\mathbf{x}\in\mathbf{X}} H(Y|\mathbf{X} = \mathbf{x})\mathbb{P}(\mathbf{X} = \mathbf{x})$$

2.2.2 Soma de subconjuntos

Para se avaliar o desempenho dos algoritmos criados neste trabalho, utilizamos instâncias artificiais que são reduções do problema da soma de subconjuntos. Este problema consiste em, dado um conjunto finito de inteiros não-negativos S e um inteiro não-negativo t, descobrir se há um subconjunto de S que soma t. Podemos resolver este problema com a solução de uma instância do problema de seleção de características onde o conjunto de características é S' uma cópia de S e a função de custo é c:

$$c(X) = |t - \sum_{x\in X} x|, \text{ para todo } X \in \mathcal{P}(S').$$

Assim como a função de custo MCE, a função de custo de somas de subconjuntos também apresenta um formato interessante nas cadeias do reticulado booleano. Para toda cadeia com elementos $A \subseteq B \subseteq C$ vale que $c(B) \leq max\{c(A), c(B)\}$. Vamos provar esta propriedade para dois casos disjuntos, quando $|t - \sum_{b\in B} b| > 0$ e quando $|t - \sum_{b\in B} b| \leq 0$. Começamos a demonstração definindo $D = B \setminus A$ e $E = C \setminus B$.

- se $|t - \sum_{b\in B} b| > 0$, então:

$$\begin{aligned} c(B) &= |t - \sum_{b\in B} b| \\ &\leq |t - \sum_{b\in B} b + \sum_{d\in D} d| \quad \text{(pois } S \text{ contém apenas números positivos e } t - \sum_{b\in B} b > 0) \\ &= |t - \sum_{a\in B\setminus D} a| \\ &= |t - \sum_{a\in A} a| \\ &= c(A) \end{aligned}$$

  portanto, $c(B) \leq c(A)$, logo $c(B) \leq max\{c(A), c(C)\}$.

- se $|t - \sum_{b \in B} b| \leq 0$, então:

$$\begin{aligned} c(B) &= |t - \sum_{b \in B} b| \\ &\leq |t - \sum_{b \in B} b - \sum_{e \in E} e| \quad \text{(pois } S \text{ contém apenas números positivos e } t - \sum_{b \in B} b \leq 0) \\ &= |t - \sum_{c \in B \cup E} c| \\ &= |t - \sum_{c \in C} c| \\ &= c(C) \end{aligned}$$

  portanto, $c(B) \leq c(C)$, logo $c(B) \leq max\{c(A), c(C)\}$.

Como acabamos de provar para os dois casos possíveis, temos que $c(B) \leq max\{c(A), c(C)\}$. □

2.3 O problema U-Curve

As duas funções de custo apresentadas na seção 2.2.1 descrevem curvas que tem um formato em U (a menos de oscilações) nas cadeias do reticulado booleano, vamos definir esta propriedade agora.

Definição 2.3.1. *Uma cadeia é dita* ***maximal*** *se não existe outra cadeia no reticulado que contenha propriamente esta cadeia.*

Definição 2.3.2. *Uma função de custo c é dita* ***decomponível em curvas U*** *se para toda cadeia maximal* $X_1, ..., X_l$, $c(X_j) \leq max\{c(X_i), c(X_k)\}$ *sempre que* $X_i \subseteq X_j \subseteq X_k$, $i, j, k \in \{1, ..., l\}$.

Vamos considerar então o problema de seleção de características em que a função de custo utilizada é decomponível em curvas U. Este é o problema central deste trabalho.

Definição 2.3.3 (Problema U-Curve)**.** *Dados um conjunto finito e não-vazio S e uma função de custo c decomponível em curvas em U, encontrar um subconjunto* $X \in \mathcal{P}(S)$ *tal que* $c(X) \leq c(Y)$, $\forall Y \in \mathcal{P}(S)$.

O problema U-Curve é um caso particular do problema de seleção de características com uma propriedade que nos permite achar o mínimo global sem a necessidade de avaliar cada ponto do reticulado booleano. Isso é possível porque a propriedade U-Curve (da decomponibilidade da função de custo em curvas U) nos garante que o custo dos elementos de uma cadeia não podem cair uma vez que aumentaram. Sejam por exemplo dois elementos $A \subseteq B$ de $\mathcal{P}(S)$, então:

- se $c(B) > c(A)$, então $c(X) > c(A)$ para todo X do intervalo $[B, \mathcal{P}(S)]$;
- se $c(A) > c(B)$, então $c(X) > c(B)$ para todo X do intervalo $[\emptyset, A]$;

Desta maneira, quando um problema de seleção de características tem uma função de custo decomponível em curvas U a menos de algumas oscilações, é vantajoso aproximar a solução deste problema pela solução encontrada por um algoritmo de busca do problema U-Curve. Tal abordagem não é ótima, porém, como existem poucas oscilações da função de custo, é provável que a solução encontrada ainda seja próxima da melhor solução.

Capítulo 3

O algoritmo Parallel U-Curve Search

O algoritmo `Parallel U-Curve Search (PUCS)` foi desenvolvido para resolver o problema U-Curve particionando o espaço de busca em partes que podem ser resolvidas independentemente e de forma paralela. Além disso, a dinâmica desse algoritmo depende de parâmetros que determinam o tempo de execução e qualidade da solução obtida, permitindo ao usuário adequar o algoritmo aos recursos computacionais disponíveis.

3.1 Princípios

Seja S o conjunto de características do problema em questão. O primeiro passo do particionamento é escolher arbitrariamente S' um subconjunto de S; de maneira complementar, definimos $\overline{S'} = S \setminus S'$. Agora, sejam $X, Y \in \mathcal{P}(S)$ e $\sim$ a relação:

$$X \sim Y \iff (X \cap S') = (Y \cap S')$$

Esta relação é de equivalência, pois nela valem:

- reflexividade

$$X \sim X, \text{ pois } (X \cap S') = (X \cap S')$$

- simetria

$$\begin{aligned} X \sim Y &\iff \\ (X \cap S') = (Y \cap S') &\iff \\ (Y \cap S') = (X \cap S') &\iff \\ Y \sim X & \end{aligned}$$

- transitividade,

$$\begin{aligned} X \sim Y, Y \sim Z &\Rightarrow \\ (X \cap S') = (Y \cap S') = (Z \cap S') &\Rightarrow \\ (X \cap S') = (Z \cap S') &\Rightarrow \\ X \sim Z & \end{aligned}$$

Portanto, o conjunto das classes de equivalência definidas por $\sim$ é uma partição do espaço de busca original. Tome como exemplo o conjunto $S = \{a, b, c\}$; se $S' = a$, então existem duas classes de equivalência no particionamento do espaço de busca que definimos, formados pelos conjuntos $\{\emptyset, b, c, bc\}$ e $\{a, ab, ac, abc\}$.

Pela definição da relação $\sim$ temos que a presença de cada característica de S' em uma dada parte do reticulado não muda, isto é, ou ela está presente em todos subconjuntos da parte ou não está presente em nenhum, portanto, dizemos que estas variáveis são **fixas**. De modo análogo, as variáveis de $\overline{S'}$ são **livres**. Tanto variáveis fixas quanto livres podem definir reticulados Booleanos junto a relação de ordem parcial $\subseteq$.

O conjunto $\mathcal{P}(S')$ induz um reticulado Booleano em que cada elemento representa uma classe de equivalência do espaço de soluções do problema original, chamamos este de **reticulado externo**. Para cada classe de equivalência (nó do reticulado externo), o conjunto $\mathcal{P}(\overline{S'})$ induz um outro reticulado Booleano (**reticulado interno**) em que cada elemento representa um subconjunto de problema original. Seja $A \in \mathcal{P}(S')$ um elemento do reticulado externo, então cada $B \in \mathcal{P}(\overline{S'})$ do reticulado interno em A representa o conjunto $X = B \cup A$ do espaço de busca do problema original. A figura 3.1 apresenta um exemplo de particionamento feito pelo `PUCS` em um reticulado Booleano com cinco características.

Os reticulados internos e externo elucidam a estrutura recursiva do problema de seleção de características e sugerem que podemos construir uma solução ao problema original a partir de soluções de outros problemas, sobre os reticulados externo e internos, abordagem conhecida em computação como divisão e conquista. Seja $\langle S, c\rangle$ uma instância do problema de seleção de características, S' o conjunto de variáveis fixas, $\overline{S'}$ o conjunto de variáveis livres, e $A \in \mathcal{P}(S')$ um subconjunto que é nó do reticulado externo, então podemos definir um outro problema de seleção de características $\langle \overline{S'}, c_A\rangle$ em que

$$c_A(X) = c(X \cup A).$$

Resolver a instância $\langle \overline{S'}, c_A\rangle$ é essencialmente achar o mínimo do problema inicial restrito a classe de equivalência de A, dizemos também que estamos resolvendo a parte A. Se soubermos em qual classe o mínimo global reside, podemos resolver apenas tal parte e garantir que a solução encontrada é a solução do problema original.

3.2 Dinâmica

Com as estruturas de reticulado interno e externo, o `PUCS` resolve uma instância do problema U-Curve em duas etapas. Na primeira, o algoritmo percorre o reticulado externo, fazendo podas sempre que possível, e armazena cada parte que é candidata a conter o mínimo global do problema. Na segunda etapa, para cada parte candidata, resolve-se o problema U-Curve auxiliar que é equivalente ao problema original, mas restrito a parte de interesse; em seguida, escolhe-se como resposta o conjunto custo mínimo entre as soluções dos problemas parciais.

3.2.1 Condições de poda

As podas eliminam do reticulado externo intervalos da forma $[X, \mathcal{P}(S')]$ ou $[\emptyset, X]$ e são realizadas sempre que a hipótese de curva em U implica que todas as partes contidas nestes intervalos não contém o mínimo global. Para entender o critério de poda, vamos definir que a **ponta superior** de um reticulado Booleano $\mathcal{P}(A)$ é o próprio conjunto A e a **ponta inferior** deste reticulado é o conjunto vazio. Note que no reticulado interno de uma parte P a ponta inferior

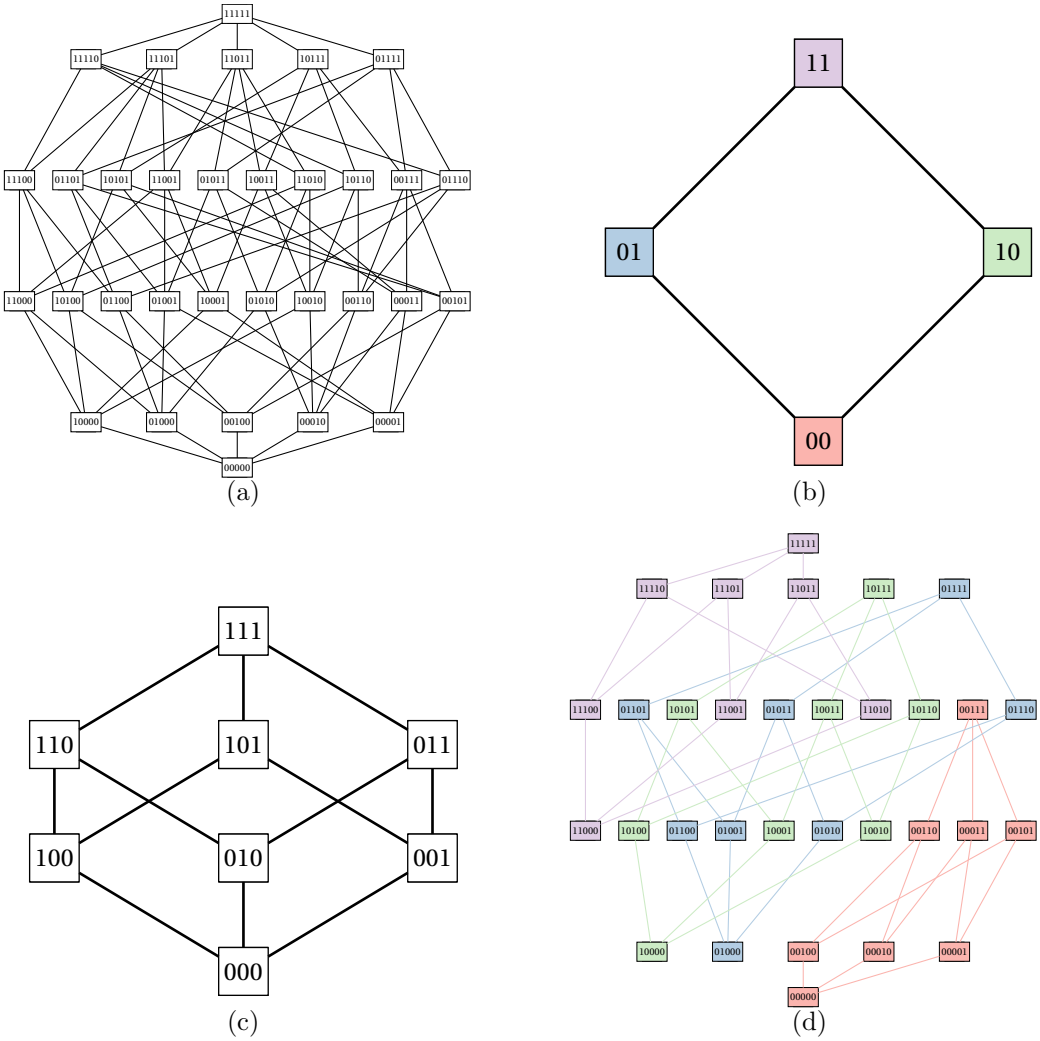


Figura 3.1: Exemplo de particionamento feito pelo algoritmo `PUCS` *em uma instância com cinco características; o reticulado Booleano desta instância é representado na figura 3.1(a). Neste particionamento, as duas primeiras variáveis formam o conjunto de variáveis fixadas, definindo o reticulado externo (figura 3.1(b)) enquanto as outras três definem os reticulados internos, que são cópias do reticulado da figura 3.1(c). A figura 3.1(d) mostra o reticulado Booleano original, sem as arestas que ligam duas partes diferentes, e a cor de cada nó representa a qual parte tal nó pertence, de acordo com as cores do reticulado externo em 3.1(b) Note que, de fato, cada parte forma um reticulado pequeno de mesmo tamanho e com mesma estrutura que o reticulado da figura 3.1(c)*

representa o próprio conjunto de características P, enquanto que a ponta superior representa o conjunto de características $P \cup \overline{S'}$.

Teorema 3.2.1 (Critério de poda para o reticulado externo do algoritmo `PUCS`)**.** *Sejam S um conjunto de características e S' um conjunto de variáveis fixas no particionamento definido pelo algoritmo* `PUCS`*. Dados $P, Q \in \mathcal{P}(S')$ dois elementos do reticulado externo com $Q \subseteq P$; se a ponta inferior do reticulado interno de P tem custo maior do que a ponta inferior do reticulado interno de Q, então todas as partes do intervalo $[P, \mathcal{P}(S')]$ tem apenas conjuntos de características com custo maior do que o custo da ponta inferior de Q.*

Demonstração. Se o custo da ponta inferior do reticulado interno de P é maior do que a de Q, então:

$$c_Q(\emptyset) < c_P(\emptyset)$$
$$c(\emptyset \cup Q) < c(\emptyset \cup P)$$
$$c(Q) < c(P)$$

Como $Q \subseteq P$, temos que existe uma cadeia que passa pelas pontas inferiores de Q e P. Além disso, para qualquer conjunto de características $X \in \mathcal{P}(S)$, com $P \subseteq X$, a hipótese de curva em U garante que:

$$c(P) \leq max\{c(Q), c(X)\}$$

e como $c(P) > c(Q)$, temos que $c(X) \geq c(P)$, isto é, qualquer elemento do reticulado Booleano original que cobre a ponta inferior de P tem custo estritamente maior do que o custo da ponta inferior de Q. Note que para qualquer parte R do intervalo $[P, \mathcal{P}(S')]$, vale que $P \subseteq R$, e como a ponta inferior de P não contém nenhum elemento de $\overline{S'}$, então qualquer conjunto de características da parte R cobre a ponta inferior de P e portanto tem custo estritamente maior do que o custo da ponta inferior da parte Q. □

Teorema 3.2.2 (Critério dual de poda para o reticulado externo do algoritmo `PUCS`)**.** *Sejam S um conjunto de características e S' um conjunto de variáveis fixas no particionamento definido pelo algoritmo* `PUCS`*. Dados $P, Q \in \mathcal{P}(S')$ dois elementos do reticulado externo com $Q \subseteq P$; se a ponta superior do reticulado interno de P tem custo menor do que a ponta superior do reticulado interno de Q, então todas as partes do intervalo $[\emptyset, Q]$ tem apenas conjuntos de características com custo maior do que o custo da ponta superior de P.*

Demonstração. Se o custo da ponta superior do reticulado interno de Q é maior do que a de P, então:

$$c_P(\overline{S'}) < c_Q(\overline{S'})$$
$$c(\overline{S'} \cup P) < c(\overline{S'} \cup Q)$$

Como $Q \subseteq P$, temos que existe uma cadeia que passa pelas pontas superiores de Q e P. Além disso, para qualquer conjunto de características $X \in \mathcal{P}(S)$, com $\{Q \cup \overline{S'}\} \supseteq X$, a hipótese de curva em U garante que:

$$c(Q \cup \overline{S'}) \leq max\{c(P \cup \overline{S'}), c(X)\}$$

e como $c(Q \cup \overline{S'}) > c(P \cup \overline{S'})$, temos que $c(X) \geq c(Q \cup \overline{S'})$, isto é, qualquer elemento do reticulado Booleano original que é coberto pela ponta superior de Q tem custo estritamente maior do que o custo da ponta superior de P. Note que para qualquer parte R do intervalo $[\emptyset, Q]$, vale que $R \subseteq Q$, e como a ponta superior de Q contém todos os elementos de $\overline{S'}$, então qualquer conjunto de características da parte R é coberto pela ponta superior de Q e portanto tem custo estritamente maior do que o custo da ponta superior da parte P. □

3.2.2 Passeio aleatório no reticulado externo

O passeio do `PUCS` se inicia escolhendo arbitrariamente um nó inicial que pertence ao espaço de busca, então a cada passo escolhe-se aleatoriamente um vizinho do nó corrente, que também deve pertencer ao espaço de busca, e verificam-se as condições de poda. Caso elas sejam verdadeiras, o procedimento de poda elimina parte do reticulado; se o vizinho escolhido foi removido nesta etapa, então escolhe-se outro vizinho. O vizinho escolhido torna-se então o nó corrente e o procedimento é repetido até que não seja possível escolher um vizinho; quando isto ocorre e o espaço ainda não foi esgotado, escolhe-se novamente um início de passeio arbitrariamente. Todo nó visitado é automaticamente removido do espaço de busca, e os passeios aleatórios são repetidos até que o espaço de busca tenha sido esgotado, ou seja, todo nó foi ou visitado ou removido em alguma poda.

Durante a realização dos passeios aleatórios, precisamos armazenar quais são as partes candidatas a conterem o mínimo global. As podas deste algoritmo removem do espaço de busca apenas partes que obrigatoriamente não contém o mínimo global, portanto qualquer outra parte é candidata a conter tal elemento. Desta forma, como toda parte é visitada ou podada, temos que o conjunto de nós visitados e não podados é exatamente o conjunto de candidatos a conterem o subconjunto de custo ótimo.

As figuras 3.2 - 3.4 mostram a dinâmica do `PUCS` ao resolver uma instância do problema U-Curve.

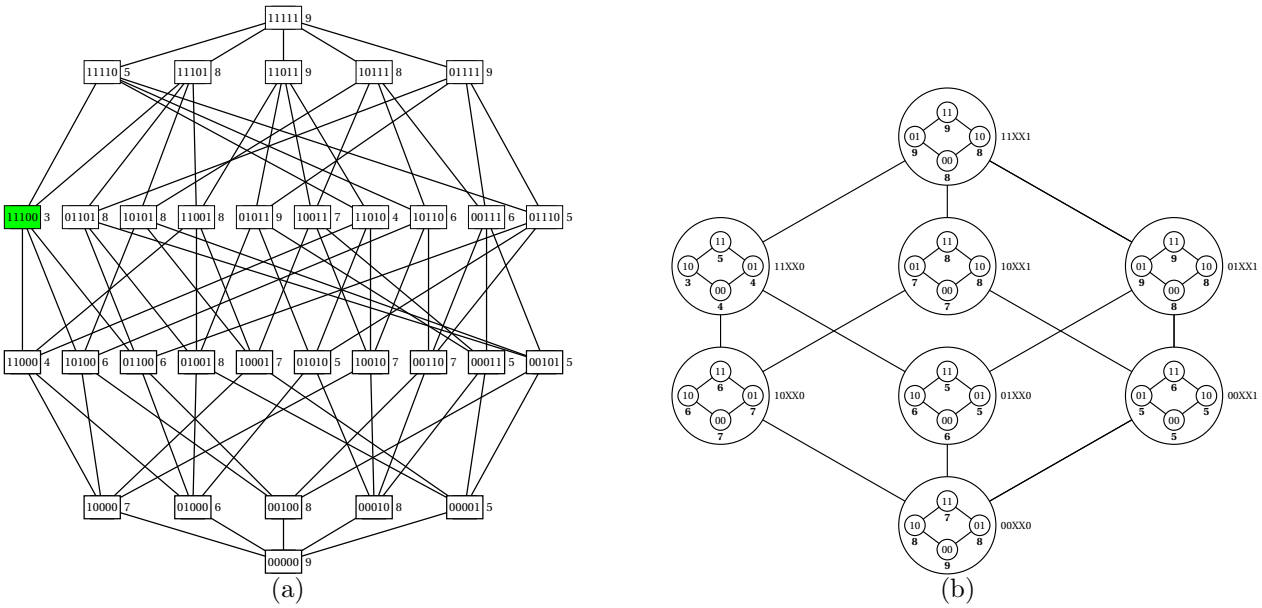


Figura 3.2: Uma instância do problema U-Curve e o seu particionamento quando o conjunto de variáveis fixas S' é composto pela primeira, segunda e última variável. No reticulado externo, denotamos por X don't cares, que são as variáveis livres do particionamento. O subconjunto colorido em verde é o elemento de custo mínimo desta instância.

- Figura 3.2(a): uma instância do problema U-Curve com cinco características e com a função de custo anotada ao lado dos nós do reticulado.
- Figura 3.2(b): o particionamento do espaço de busca quando são fixadas a primeira, segunda e última característica.

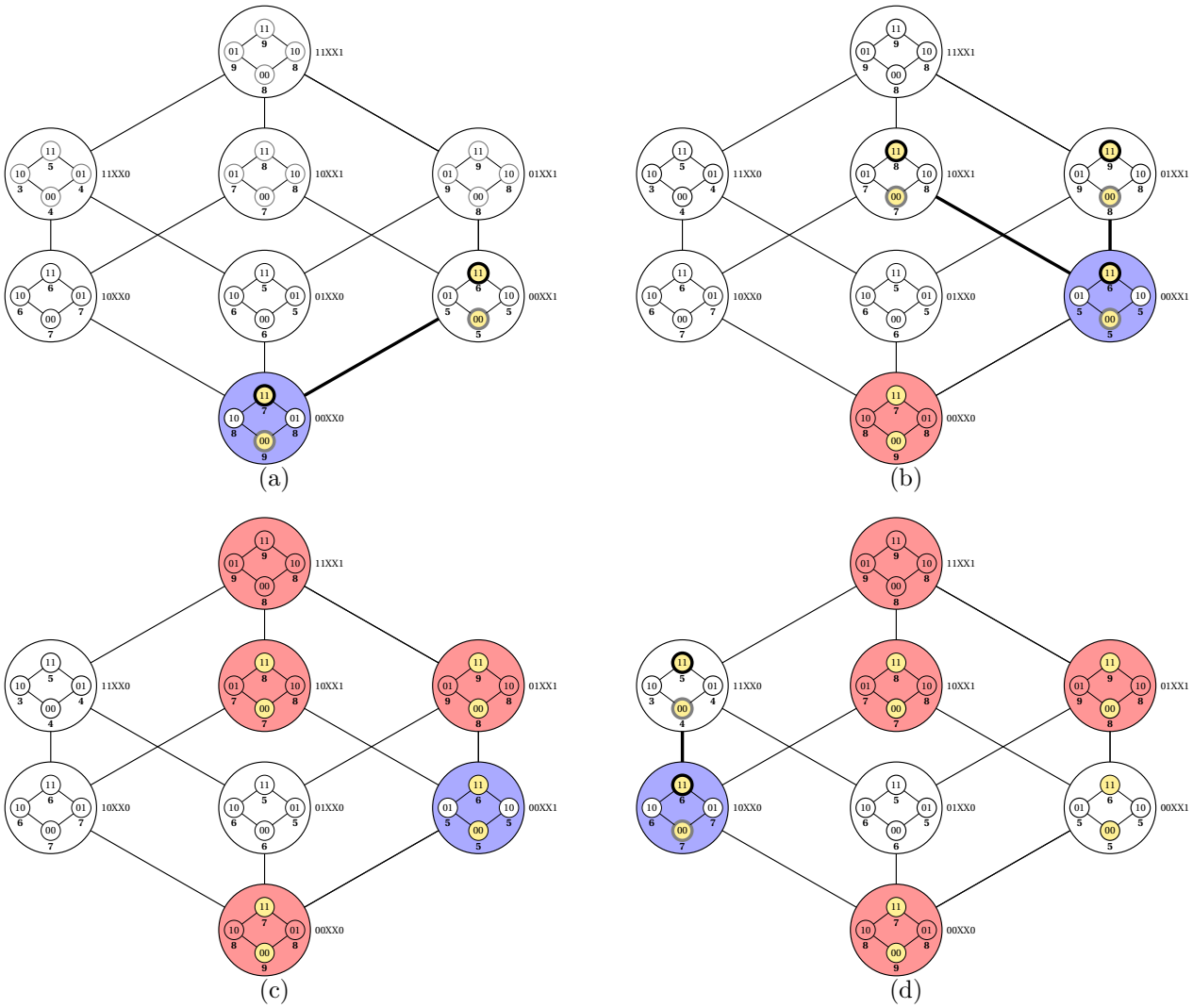


Figura 3.3: Dinâmica do algoritmo `PUCS` *ao resolver a instância apresentada na figura 3.2*

- Figura 3.3(a) a parte 00XX0 é escolhida arbitrariamente para ser o início do passeio aleatório. O vizinho 00XX1 é escolhido aleatoriamente como candidato a ser o próximo nó do passeio. Como o custo da ponta superior de 00XX1 (6) é menor do que o custo da ponta superior de 00XX0 (7), o intervalo de partes $[000, 000]$ é removido do espaço de busca.

- Figura 3.3(b) a pontas inferiores das partes 10XX1 (7) e 01XX1 (8) têm custo maior do que a ponta inferior de 00XX1 (5), portanto os intervalos de partes $[101, 111]$ e $[011, 111]$ são removidos do espaço de busca.

- Figura 3.3(c) todos os vizinhos de 00XX1 foram podados, portanto esta parte torna-se candidata a conter o mínimo, e iniciamos um novo passeio.

- Figura 3.3(d) a parte 10XX0 é escolhida arbitrariamente como início de passeio. O custo da ponta superior de 10XX0 (6) é maior do que o custo da ponta superior de 11XX0 (5), portanto o intervalo de partes $[000, 100]$ é removido do espaço de busca.

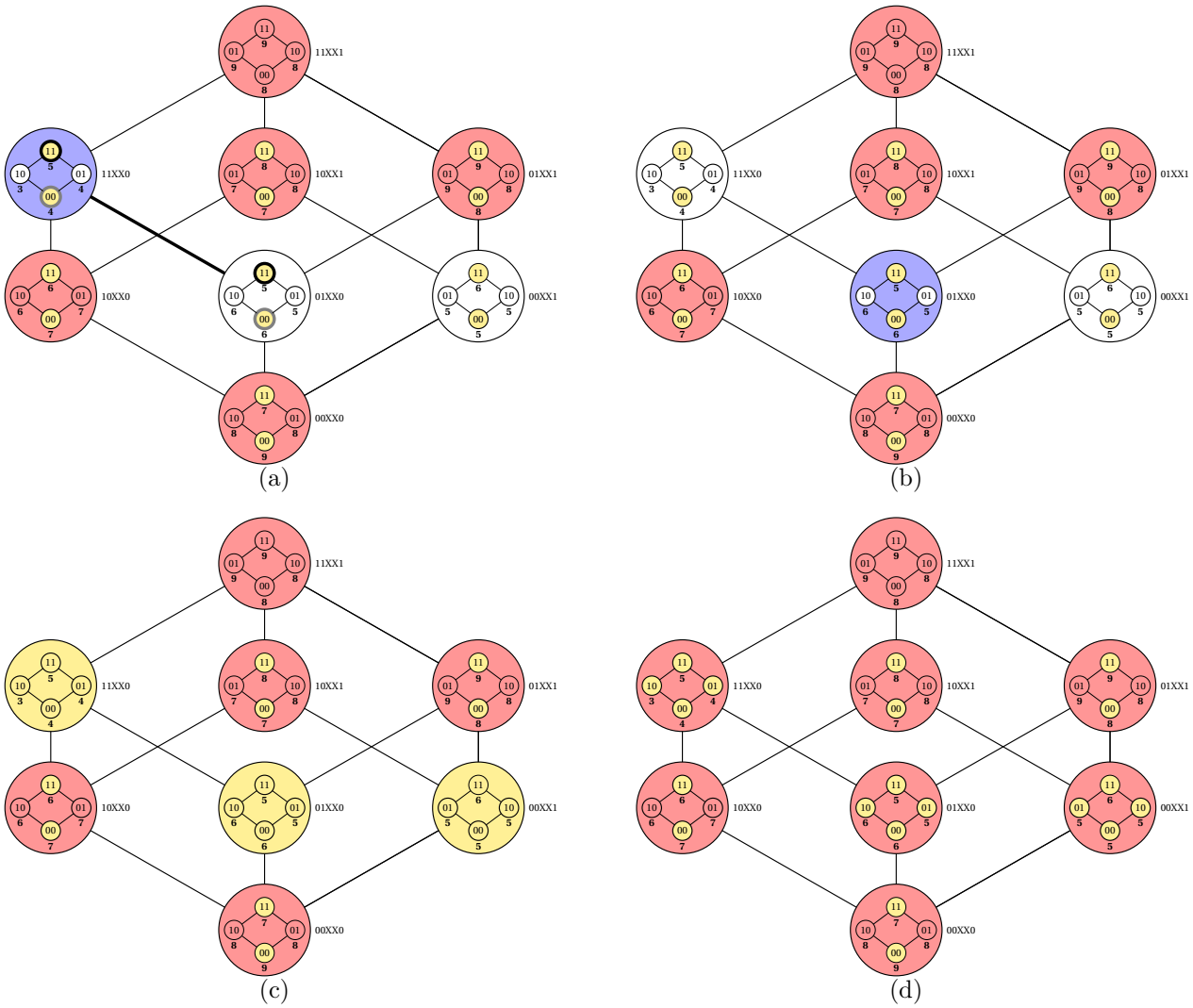


Figura 3.4: Continuação da figura 3.3

- Figura 3.4(a) Nenhuma poda é realizada ao comparar 11XX0 com 01XX0, portanto a última torna-se o nó corrente e a primeira torna-se candidata a conter o mínimo.
- Figura 3.4(b) A parte 01XX0 torna-se o nó corrente, mas como não possui vizinho no espaço de busca é removida do espaço de busca e torna-se candidata a conter o mínimo.
- Figura 3.4(c) Todo nó foi removido do espaço de busca por podas ou por visitas. Resolve-se as partes candidatas a conter o mínimo.
- Figura 3.4(d) O elemento de custo mínimo entre as partes é 11100, que é de fato a solução ótima do problema com custo 3.

3.2.3 Solução das partes

Ao fim do passeio aleatório, teremos uma coleção de partes que precisam ser resolvidas para se obter o conjunto de custo mínimo. Nesta etapa, o `PUCS` constrói para cada parte uma instância auxiliar do problema U-Curve que é equivalente ao problema original, porém restrito

a parte de interesse. Seja $\langle S, c\rangle$ a instância do problema original e S' o conjunto de variáveis fixas no particionamento feito pelo `PUCS` neste problema, então, dada uma parte $A \in \mathcal{P}(S')$, o conjunto de custo mínimo nesta parte é exatamente a solução ótima do problema U-Curve auxiliar $\langle \overline{S'}, c_A\rangle$ em que $c_A(X) = c(X \cup A)$ para qualquer $X \in \mathcal{P}(\overline{S'})$.

Para solucionar os problemas auxiliares, podemos chamar um outro algoritmo de seleção de características, ótimo ou sub-ótimo, e podemos inclusive chamar o próprio `PUCS`, tornando o algoritmo recursivo. Chamamos o último algoritmo na sequência de chamadas recursivas de **algoritmo base**; o `PUCS` é algoritmo base apenas no caso em que cada parte contém apenas um elemento. A escolha do algoritmo base é crítica no desempenho da chamada do `PUCS` no que diz respeito a uso de recursos computacionais e também na qualidade da solução obtida.

3.3 Parâmetros de funcionamento

Na seção anterior apresentamos a dinâmica básica do algoritmo `PUCS`, porém por simplicidade não definimos alguns parâmetros que regem o funcionamento do mesmo. Apesar de ser fácil entender a dinâmica do algoritmo sem conhecer estes parâmetros, eles tem papel crítico no desempenho do mesmo, tanto no quesito de uso de recursos computacionais quanto na qualidade da solução encontrada. Estes parâmetros são p, l, e algoritmo base.

O parâmetro p define a quantidade de variáveis fixas no particionamento do espaço de busca e deve estar contido no intervalo $(0, 1]$, sendo a proporção de variáveis que devem ser fixadas; desta forma:

$$|S'| = \lceil |S| * p \rceil$$
$$|\overline{S'}| = |S| - \lceil |S| * p \rceil$$

Portanto, quanto maior o p, maior o tamanho do reticulado externo ($|\mathcal{P}(S')|$) e menor o tamanho dos reticulados internos ($|\mathcal{P}(\overline{S'})|$) e vice-versa. Note que quando p é pequeno o algoritmo `PUCS` deve ser semelhante ao algorítmo base, já que o tamanho das partes continua semelhante; quando o p é grande, o particionamento é mais "fino" porque as partes se tornam menores e consequentemente há mais partes.

Como vimos na seção 3.2.3, a estrutura criada no particionamento do problema nos permite fazer chamadas recursivas do `PUCS`. O parâmetro l determina a quantidade de chamadas recursivas que acontecerão até que o algoritmo base seja chamado. Ao fazer chamadas recursivas estamos particionando o espaço de busca seguidas vezes e portanto, assim como o parâmetro p, quando aumentamos o valor de l, o tamanho da parte que será resolvida pelo algoritmo base diminui.

O algoritmo base determina como as partes serão resolvidas. Note que os teoremas 3.2.1 e 3.2.2 garantem que se a hipótese U-Curve for verdadeira, então todas as partes que foram podadas do espaço de busca não contém o mínimo global, portanto se o algoritmo base é ótimo, então o `PUCS` também é ótimo. Se o algoritmo base for uma heurística, então o `PUCS` também se comporta como uma heurística, entretanto é provável que a solução encontrada pelo `PUCS` seja melhor do que a solução dada pelo algoritmo base. Dizemos que isto é provável porque o `PUCS` faz diversas chamadas ao algoritmo base, uma para cada parte candidata, portanto percorre mais nós do que uma chamada única do algoritmo base.

3.4 Implementação do algoritmo

O algoritmo `PUCS` foi implementado no arcabouço *featsel*, usando a linguagem `C++`. Nesta seção apresentaremos detalhes sobre sua implementação

3.4.1 Controle do espaço de busca

Sempre que um nó do reticulado externo é podado ou visitado ele deve ser removido do espaço de busca, e representar este espaço explicitamente não é uma boa solução devido ao seu tamanho, que é exponencial em relação a quantidade de características fixas. A estrutura de dados utilizada deve ser eficiente tanto para inserções (de intervalos e de pontos do reticulado) quanto para consultas. Escolhemos para nossa implementação usar a estrutura de dados de diagramas de decisão binária reduzidos e ordenados (*Reduced Ordered Binary Decision Diagram* (ROBDD)) [Bry86].

3.4.2 Paralelização do código

Usamos a biblioteca *OpenMP* na paralelização do código. Esta biblioteca nos permite paralelizar o algoritmo de maneira fácil, com anotações que indicam ao compilador como blocos do código fonte podem ser processados paralelamente.

O `PUCS` foi criado com o intuito de ser um algoritmo paralelo para resolver o problema U-Curve. A particionamento do espaço foi feito exatamente para que o processo de paralelização do código fosse simples, facilitando a distribuição de trabalho entre threads e usando o mínimo de comunicação entre as mesmas. Desta forma, para paralelizar o código, basta indicar ao compilador que o particionamento e passeio pelo reticulado deve ser feito pela thread principal enquanto a solução de cada parte pode ser realizada por qualquer outra thread, usando a estrutura de *tasks* da biblioteca *OpenMP*. Desta forma, sempre que o algoritmo visita uma parte que não é podada, cria-se uma *task* que deve solucionar tal parte.

Esta abordagem pode causar cálculos supérfluos, pois partes resolvidas podem ser podadas no decorrer dos passeios aleatórios, no entanto, para evitar tais recálculos seria necessário esperar o percorrimento de todo reticulado externo antes de se resolver as partes. Esta segunda abordagem tornaria necessário armazenar e manter atualizada uma lista de partes candidatas, o que pode ser caro computacionalmente; além disso, como o passeio é feito apenas por uma thread, todas as outras seriam subutilizadas durante o passeio.

3.5 Testes com instâncias artificiais

Nesta seção apresentamos testes feitos com o `PUCS` ao solucionar instâncias artificiais do problema U-Curve onde função de custo utilizada é a de soma de subconjuntos. Estes testes foram feitos em uma servidora

3.5.1 Ajuste de parâmetros

Antes de discutir o desempenho do algoritmo, precisamos entender como os parâmetros devem ser ajustados para cada tipo de instância do problema U-Curve. Consideramos instâncias pequenas aquelas que podem ser resolvidas otimamente. Estas instâncias costumam ter no máximo por volta de 30 características e algoritmos ingênuos como a busca exaustiva podem, já para estas instâncias, se tornar muito caros computacionalmente. Já as instâncias grandes, que

não podem ser resolvidas por algoritmos ótimos, costumam ter mais do que 30 características; utilizam-se para estas algoritmos sub-ótimos, heurísticas.

Instâncias pequenas

Instâncias grandes

Os parâmetros p e l influenciam a dinâmica do algoritmo e também na granularidade do particionamento feito. Quando o valor destes parâmetros é baixo, o número de partes deve ser baixo e o desempenho do `PUCS` deve ser mais próximo ao do algoritmo base. Por outro lado, quando p e l têm valor alto o particionamento deve ser mais fino, criando mais partes; como consequência, o tempo gasto no particionamento e no percorrimento do reticulado externo deve aumentar, e além disso, o número de partes que serão resolvidas pelo algoritmo base também deve aumentar. Portanto, quanto maior o valor destes parâmetros, maior deve ser o tempo gasto pelo `PUCS`.

Se o algoritmo base usado é ótimo e a hipótese U-Curve é verdadeira, então os parâmetros p e l não devem influenciar na qualidade da solução encontrada porque nesta situação o `PUCS` é ótimo, ou seja, a solução encontrada sempre é a de custo mínimo.

Capítulo 4

Conclusão

Bibliografia

[AG+18] Esmaeil Atashpaz-Gargari, Marcelo S. Reis, Ulisses M. Braga-Neto, Junior Barrera e Edward R. Dougherty. «A fast Branch-and-Bound algorithm for U-curve feature selection». Em: *Pattern Recognition* 73.Supplement C (2018), pp. 172 –188. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.08.013. URL: http://www.sciencedirect.com/science/article/pii/S0031320317303254.

[Bar+07] J. Barrera, R. M. Cesar-Jr, D.C. Martins-Jr, R.Z.N Vencio, E. F. Merino, M. M. Yamamoto, F. G. Leonardi, C. A. B. Pereira e H. A. Portillo. «Constructing probabilistic genetic networks of *Plasmidium falciparum* from dynamical expression signals of the intraerythrocytic development cycle.» Em: *Methods of Microarray Data Analysis* V (2007), pp. 11–26.

[Bry86] R. E. Bryant. «Graph-Based Algorithms for Boolean Function Manipulation». Em: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677–691. ISSN: 0018-9340. DOI: 10.1109/TC.1986.1676819.

[DMJ06] R.M. Cesar-Jr an J. Barrera D.C. Martins-Jr. «W-operator window design by minimization of mean conditional entropy». Em: *Patter Analysis & Applications* (2006), pp. 139–153.

[JCJB04] D. C. Martins Jr, R. M. Cesar-Jr e J. Barrera. «W-operator window design by maximization of training data information». Em: *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium* (2004), pp. 162–169.

[RFB14] M. S. Reis, C. E. Ferreira e J. Barrera. «The U-curve optimization problem: improvements on the original algorithm and time complexity analysis». Em: *ArXiv e-prints* (jul. de 2014). arXiv: 1407.6067 [cs.LG].

[Rei12] M. S. Reis. «Minimization of decomposable in U-shaped curves functions defined on poset chains – algorithms and applications». Tese de doutoramento. University of Sao Paulo, 2012.