Vidal Daniel da Fontoura

Um modelo para dissertações e teses (escrevi um título mais longo para ver como se comporta a quebra de linhas e o espaçamento entre elas)

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Aurora Trinidad Ramirez Pozo.

Co-orientador: Roberto Santana .

Curitiba PR
2016

Resumo

O resumo deve conter no máximo 500 palavras, devendo ser justificado na largura da página e escrito em um único parágrafo[1] com um afastamento de 1,27 cm na primeira linha. O espaçamento entre linhas deve ser de 1,5 linhas. O resumo deve ser informativo, ou seja, é a condensação do conteúdo e expõe finalidades, metodologia, resultados e conclusões.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Palavras-chave: palavra-chave 1, palavra-chave 2, palavra-chave 3.

[1] E também não deve ter notas de rodapé; em outras palavras, não siga este exemplo... ;-)

Abstract

The abstract should be the English translation of the “resumo”, no more, no less.

Keywords: keyword 1, keyword 2, keyword 3.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná

Lista de Símbolos

α	alfa, primeira letra do alfabeto grego
β	beta, segunda letra do alfabeto grego
γ	gama, terceira letra do alfabeto grego
ω	ômega, última letra do alfabeto grego
π	pi
τ	Tempo de resposta do sistema
θ	Ângulo de incidência do raio luminoso

Capítulo 1

Introdução

Este modelo foi proposto com o intuito de padronizar e simplificar as monografias, dissertações e teses produzidas no Departamento de Informática da UFPR. Ele foi vagamente inspirado nas normas da ABNT (conforme indicado em [UFPR, 2015]), mas não as segue *ipsis litteris*. Várias alterações foram feitas com o objetivo de melhorar sua estética e tornar o texto mais legível para trabalhos na área de informática. A versão atualizada deste modelo está disponível em [Maziero, 2015].

Este modelo está baseado em uma classe especifica `ppginf.cls`, que aceita várias opções de compilação. A versão do documento pode ser:

- `defesa`: é gerado um documento em espaço 1,5, frente simples e sem as páginas iniciais adicionais; é uma versão adequada para receber as anotações dos membros da banca de defesa.
- `final`: é gerado um documento em espaço simples, frente/verso, com páginas iniciais (capa, ficha catalográfica, folha de aprovação, agradecimentos, etc). É uma versão bem mais compacta, mais ecológica e ideal para a impressão definitiva.

Para obter os melhores resultados, compile este modelo usando a seguinte sequência de passos:

```
pdflatex  main          // compilação inicial
bibtex main             // processa referências bibliográficas
pdflatex  main          // compilação final
```

ou

```
make                    // faz tudo...
```

Os principais itens considerados na formatação deste documento foram:

- Papel em formato A4, com margens de 20 mm à direita e embaixo, 30 mm nos demais lados. Não devem ser usados cabeçalhos ou rodapés além dos que estão aqui propostos.

- O texto principal do documento escrito em 12 pontos. O fonte principal do texto pode ser selecionado no arquivo `packages.tex`.
- Código-fonte, listagens e textos similares são formatados em fonte Courier 12 ou 10 pontos.
- O espaçamento padrão entre linhas é 1,5 linhas (1 linha na versão final). Não inserir espaços adicionais entre parágrafos normais. Figuras, tabelas, listagens e listas de itens devem ter um espaço adicional antes e após os mesmos.
- As páginas iniciais não são numeradas.
- O corpo do texto é numerado com algarismos arábicos (1, 2, 3, ...) a partir da introdução, ate o final do documento. Os números de página devem estar situados no alto à direita (páginas direitas) ou à esquerda (páginas esquerdas).
- Expressões em inglês, grego, latim ou outras línguas devem ser enfatizadas em itálico, como *sui generis* ou *scheduling* (use o comando `\emph{...}`).
- Para reforçar algo, deve-se usar somente **negrito**. Sublinhado ou MAIÚSCULAS não devem ser usados como forma de ênfase!
- As notas de rodapé também têm um modelo[1]. Notas de rodapé servem para fazer algum comentário paralelo; não as use para colocar URLs, referências bibliográficas ou significado de siglas.

Felizmente o LaTeX resolve a maior parte dessas questões!

[1] As notas de rodapé dever ser escritas em tamanho 10 pt, numeradas em arábico.

Capítulo 2

Referencial Teórico

2.1 PDP - Problema de Dobramento de Proteínas

Proteínas são estruturas básicas, essenciais para vida e possuem incontáveis funções biológicas. Proteínas são sintetizadas pelos ribossomos seguindo um formato provido pelo mensageiro RNA (mRNA). Durante a síntese, as proteínas dobram (enovelam) em uma estrutura tridimensional única, conhecida como conformação nativa. Este processo é chamado: dobramento de proteínas (*protein folding*). A função biológica de uma proteína depende da sua estrutura tridimensional.

As proteínas são polímeros compostos por sequências de aminoácidos (também chamados de resíduos) conectados linearmente por ligações peptídicas. Cada aminoácido é composto por um átomo central de carbono ($C\alpha$) conectado a um átomo de hidrogênio, um grupo amina, um grupo carboxila e uma cadeia lateral (*side-chain*) a qual confere a cada aminoácido uma função distinta. Uma ligação peptídica é formada por dois aminoácidos quando o grupo carboxila de uma molécula reage com o grupo amina da outra. Este processo de agregação de aminoácidos é conhecido como desidratação pois libera uma molécula de água (H_2O) [**?**]. Proteínas podem ser chamadas de cadeias polipeptídicas. Todos os aminoácidos tem o mesmo *backbone* e se diferem dos outros apenas pela *side-chain*, a qual pode ser um simples átomo de hidrogênio ou até um grupo heterocíclico complexo. A *side-chain* define as propriedades físicas e químicas dos aminoácidos de uma proteína [**?**].

2.2 Dobramento de proteínas

É o processo em que cada cadeia polipeptídica é transformada em uma estrutura compacta que realiza alguma função biológica [**?**]. Estas funções incluem controle e regulação de processos químicos essenciais para os organismos vivos [**?**]. A estrutura tridimensional mais estável é chamada de conformação nativa e é a qual permite que a proteína exerça corretamente sua função biológica [**?**, **?**].

Experimentos conduzidos por Anfinsen et al. [**?**, **?**, **?**], mostraram que as proteínas possuem apenas uma conformação nativa e que as informações essenciais que codificam a estrutura estão contidas na sequência de aminoácidos. A conformação tridimensional nativa é dada pela estrutura primária (sequência de aminoácidos) de uma proteína.

Muitas proteínas podem desnaturar por modificações no ambiente em que estão inseridas, conforme demonstrado por [**?**, **?**, **?**]. Durante o processo de desnaturação as proteínas perdem sua forma nativa (desdobram) e, consequentemente, perdem sua função. O exemplo mais conhecido de desnaturação proteica é o da clara do ovo. A clara do ovo é composta por água e albumina. A albumina é uma proteína polar, portanto solúvel em água. Ao fritar ou cozinhar o ovo, eleva-se a temperatura, levando à desnaturação da albumina que, mesmo ao retornar à temperatura original, não consegue voltar à sua conformação nativa. Além de se desdobrarem é possível que ocorram erros de dobramento na formação das proteínas causando com que a proteína não exerça sua função biológica corretamente. Estudos tentam identificar causas para os erros de dobramento das proteínas pois muitas enfermidades são causadas pelo mal dobramento de proteínas como, por exemplo, mal de Alzheimer [**?**, **?**], alguns tipos de câncer [**?**, **?**, **?**], fibrose cística [**?**], arteriosclerose [**?**], mal de Parkinson [**?**], entre outras. Portanto, entender como o processo de dobramento de proteínas ocorre é de fundamental importância. Um dos objetivos comuns das ciências biológicas é caracterizar funcionalmente sequências de proteínas através da resolução de suas conformações nativas [**?**]. Varias áreas da ciência, tais como Biologia, Medicina, Química Orgânica, realizam diferentes estudos das proteínas. Muitos destes estudos são voltados para o processo de dobramento das proteínas que pode sofrer alterações: tanto em como a conformação estará disposta no espaço, como ela estará agrupada e sobre sua má formação. Isto é muito relevante para estudos que visam à produção de medicamentos, suplementos alimentares, técnicas que manipulam o DNA, ou para formação de novos compostos proteicos sintéticos em laboratório [**?**]. É importante mencionar que apesar do avanço na grande quantidade de proteínas que se tem conhecimento por conta de projetos de sequenciamento genômico, apenas uma pequena fração de estruturas tridimensionais é conhecida.

A cristalografia de raios-X e espectroscopia de RNM são os métodos experimentais mais poderosos para o estudo da estruturas de proteínas [**?**] [**?**]. Entretanto estes métodos são altamente custosos tanto em esforços computacionais, de tempo e financeiros, e estão disponíveis apenas para algumas instituições.

Embora o conceito de dobramento de proteínas tenha surgido da área de biologia molecular, este problema é um problema interdisciplinar, o qual requer apoio de muitas áreas do conhecimento, e é considerado como um dos desafios atuais mais importantes da biologia e bioinformática [**?**].

Na biologia computacional existem dois problemas que tratam sobre o dobramento de proteínas. São eles: problema de predição estrutura de proteínas (ou PSP - *Protein Structure Prediction*), que trata de predizer a estrutura tridimensional (conformação) a partir de sua sequência (estrutura primária); e o problema de dobramento de proteínas (PDP ou PFP - *Protein*

Folding Problem), o qual trata da determinação dos passos/eventos que conduzem o dobramento a partir da estrutura primária até a conformação nativa [**?**]. Porém, na literatura, é encontrado ambos os termos sendo utilizados sem nenhuma distinção, normalmente se referindo apenas ao primeiro problema [**?**].

A ciência da computação desempenha um papel importante nisto, propondo e desenvolvendo modelos e soluções computacionais para o estudo de ambos os problemas PSP e PDP [**?**]. Muitas estratégias computacionais descrevem modelos de predição de estruturas de proteínas com diferentes níveis de detalhamento e complexidade mas que permitem uma representação fidedigna da estrutura tridimensional, sem perda de viabilidade computacional [**?**]. Dessa maneira evita-se a obrigatoriedade do uso de métodos caros, aumentando a competitividade e auxiliando na consolidação de centros de pesquisa que desenvolvem estudos nesta área [**?**].

Acredita-se que a conformação nativa de uma proteína é a sua estrutura mais estável, adquirindo um estado de energia livre mínima, o que gerou a chamada hipótese da termodinâmica [**?**]. Os modelos de predição de estruturas normalmente são baseados nas leis da termodinâmica onde o problema é modelado como um problema de minimização da energia livre a respeito das possíveis conformações que uma proteína pode assumir [**?**]. A minimização da energia livre é assumida como o principal fator para a formação da estrutura da proteína. Portanto, a conformação nativa de uma proteína é dada por aquela que possuir o menor valor de energia livre.

Segundo [**?**], um modelo computacional deve possuir algumas características:

- Um conjunto de entidades que representam os átomos e as ligações entre eles.
- Regras que definem as possíveis conformações.
- Uma função que seja computacionalmente factível, para calcular a energia livre das possíveis conformações.

A próxima subseção irá discorrer sobre alguns modelos de representação de estruturas de proteínas.

2.3 Modelos de Representação de Proteínas

Em suma existem duas classes de modelos de representação de estruturas de proteínas: analítico (também conhecido como *all atom*) e discreto (chamado também de *coarse-grained*). Os modelos analíticos possuem uma descrição detalhada da estrutura tridimensional incluindo informações de todos os átomos que constituem uma proteína. Já os modelos discretos descrevem as proteínas com um nível bastante reduzido de detalhes. Os modelos discretos recentemente ganharam maior interesse, por conta de dois fatores: a simulação de modelos analíticos nem sempre é computacionalmente possível e modelos discretos possibilitam simulações biologicamente relevantes com melhor aproveitamento computacional [**?**]. Embora simulações utilizando

modelos discretos ainda não possam ser consideradas tão preditivas quanto simulações analíticas, avanços notáveis têm: sido alcançados, referente ao uso de metodologias mais rigorosas e criação de algoritmos para melhor explorar o espaço de busca [**?**]. Esta proposta irá descrever apenas os modelos discretos pois visa a utilização do modelo hidrofóbico polar (HP) 2D.

2.3.1 Modelos Discretos

Os modelos computacionais mais simples são os conhecidos como modelos de grade (*lattice models*). Estes modelos consideram as estruturas de proteínas como um colar de esferas posicionado em uma grade. O grau de liberdade dos movimentos é restrito à estrutura da grade, que pode ser 2D (plano) ou 3D (espacial). Conformações válidas são aquelas que os aminoácidos adjacentes na sequência também são adjacentes na grade e cada aminoácido ocupe uma posição distinta na grade. Muitos modelos de grade têm sido propostos e aplicados ao PDP. Os modelos 2D-HP e 3D-HP são exemplos de modelos de grade.

Modelo Hidrofóbico-Polar HP

No modelo HP os aminoácidos são classificados em 2 tipos: Hidrofílicos (Polar) e Hidrofóbico. Consequentemente, uma proteína é representada por uma *string* de caracteres definida por um alfabeto binário $\{H, P\}$. Este modelo considera que as interações entre aminoácidos hidrofóbicos (H) representam a contribuição mais importante para a energia livre de uma proteína. Portanto existe uma relação inversamente proporcional: quanto maior for a quantidade de interações hidrofóbicas (H-H), menor será a energia livre de uma proteína. Uma interação hidrofóbica (também conhecida como contato hidrofóbico) é definida como um par de aminoácidos do tipo H-H que não sejam consecutivos na sequência mas sejam adjacentes na grade. Como dito anteriormente, uma conformação é dita válida quando nenhuma posição da grade é ocupada por mais que um aminoácido. Conformações inválidas possuem colisões entre os aminoácidos. Dada uma conformação válida para o modelo HP e n o número de interações hidrofóbicas, a energia da conformação pode ser facilmente calculada utilizando a equação 2.1:

$$E(c) = n.(-1) \tag{2.1}$$

Quando uma proteína é dobrada na sua conformação nativa, os aminoácidos hidrofóbicos tendem a se agrupar no interior da estrutura, protegidos por aminoácidos polares posicionados no exterior. Dessa maneira, um núcleo hidrofóbico é formado em proteínas dobradas [**?**]. Embora simples, a estratégia computacional de buscar uma solução para o PDP utilizando modelo HP é considerada como um problema NP-completo [**?**, **?**, **?**]. O espaço de busca do modelo HP possui algumas características mencionadas na literatura [**?**, **?**, **?**, **?**, **?**] :

- Elevada degenerescência.

- Espaço de busca multimodal.
- Muitas regiões com conformações inválidas.

A figura **??** apresenta um exemplo para os modelos HP (2D e 3D). Os pontos pretos são aminoácidos do tipo H e os brancos são aminoácidos do tipo P. As linhas pontilhadas representam as interações hidrofóbicas.

Diversos trabalhos aplicam algoritmos evolucionários e bio-inspirados ao problema de dobramento de proteínas utilizando o modelo HP. Uma decisão comum a todos trabalhos que utilizam o modelo HP é a de como representar as variáveis de entrada. Na literatura é possível encontrar basicamente três representações [**?**, **?**]:

- Coordenadas cartesianas: Este método representa a posição de cada aminoácido utilizando suas coordenadas espaciais (x,y) no plano cartesiano 2D ou (x,y,z) no plano cartesiano 3D. Geralmente, sua utilização não é adequada para algoritmos baseados em população, pois estruturas idênticas ou semelhantes podem ter coordenadas totalmente diferentes [**?**];
- Coordenadas internas: Nesta representação as conformações são representadas por conjuntos de movimentos que ditam como a estrutura final irá se parecer. Esta representação é a mais utilizada em abordagens com algoritmos evolucionários para o PDP [**?**]. Existem duas possibilidades de se representar conformações utilizando coordenadas internas:
  - Coordenadas absolutas: Este tipo de coordenada é baseado na orientação do eixo da grade onde a conformação esta contida. No caso de uma grade bidimensional os possíveis movimentos são: $\{N, S, L, O\}$ ou norte,sul,leste e oeste. Já em uma grade 3D os possíveis movimentos são: $\{N, S, L, O, F, T\}$ que correspondem aos mesmos movimentos no caso 2D porém com dois movimentos a mais: para frente e para trás.
  - Coordenadas relativas: Este tipo de representação define a posição de cada aminoácido da cadeia em relação ao movimento do seu predecessor. O conjunto de movimentos possíveis para a grade 2D é definido por $\{F, E, D\}$, que correspondem aos movimentos: frente (continuar no mesmo sentindo do aminoácido anterior), à esquerda e à direita. Em um cubo 3D, os possíveis movimentos são $\{F, E, D, C, B, \}$, possuindo dois movimentos a mais: para cima e para baixo.
- Matriz de distâncias: descreve a conformação de uma proteína através de uma matriz quadrada que representa a distância entre os aminoácidos. Este tipo de representação é raramente utilizado na literatura [**?**].

Outros Modelos

Além do modelo HP, outros modelos simples em grade são utilizados para representar a estrutura de proteínas em outros estudos encontrados na literatura. Por exemplo:

- Modelo PH (*Perturbed Homopolymer*): Proposto por Shakhnovich et al. [**?**], as reações entre aminoácidos hidrofóbicos não são levadas em consideração, mas as interações entre aminoácidos do mesmo tipo são favorecidas, ou seja, H-H e P-P, desfavorecendo ligações H-P [**?**].

- Modelo LPE (*Lattice Polymer Embedding*): Modelo proposto por Unger e Moult [**?**]. A modelagem é feita a partir de uma sequência de aminoácidos, A = $a_1, ...a_n$ atrelada a uma grade cúbica. Cada aminoácido possui um coeficiente de afinidade, definido para cada par $a_i, a_j(c(a_i, a_j))$. O objetivo da função de energia é minimizar o produto dos coeficientes pela distância entre os aminoácidos [**?**].

- Modelo HP-TSSC (*Hydrophobic-Polar Tangent Spheres Side Chain Model*): este modelo proposto por Hart et al. [**?**] é baseado no modelo HP, porém não utiliza uma grade. Neste modelo a proteína é modelada via um grafo tridimensional, onde a cadeia lateral e o *backbone* de cada aminoácido são esferas de mesmo raio [**?**].

- Modelo CGE (*Charged Graph Embedding*): Modelo descrito por Ngo et al. [**?**]. Neste modelo, uma carga (*charge*) é atribuída a cada resíduo. Entretanto, as conformações permitidas não são realistas [**?**].

- Modelo HPNX: modelo proposto por Bornberg-Bauer [**?**]. Divide os 20 aminoácidos em 3 classes: hidrofóbicos (H), positivos (P), negativos (N) e neutros (X). Este modelo, assim como o modelo HP, utiliza uma grade. Interações entre aminoácidos hidrofóbicos (H-H) representam interações de atração e diminuem a energia da conformação em 4,0, as interações entre positivos (P-P) e negativos (N-N) representam interações de repulsão e aumentam a energia livre em 1,0 e as interações entre N e P decrescem a energia em 1,0. O objetivo também consiste em minimizar a energia livre. Da mesma maneira que o modelo HP, quanto mais interações hidrofóbicas melhor será o dobramento. Porém este modelo não desconsidera o valor das demais interações [**?**].

- Modelo HP-helicoidal (Helical-HP): este modelo proposto por Thomas e Dill [**?**] considera apenas uma grade bidimensional e inclui dois tipos de interação: interações não-locais através de energia de contatos hidrofóbicos e interações locais representadas por uma tendência à formação de a-hélices (chamada de propensão hélica) [**?**].

- Modelo *off-lattice* AB: este modelo proposto por Stillinger et al. [**?**] divide os aminoácidos em duas classes de acordo com sua polaridade: Hidrofóbicos (A) e Hidrofílicos (ou polares B). Inicialmente, este modelo foi aplicado em duas dimensões (2D AB *off-lattice*) e posteriormente aplicado para três dimensões (3D AB *off-lattice*). Os aminoácidos não consecutivos interagem através de um potencial modificado de Lennard-Jones. Os ângulos de torsão entre ligações sucessivas também contribuem no cálculo da função de energia [**?**].

2.3.2 Considerações Finais

Neste seção o problema de dobramento de proteínas foi apresentado e sua importância para biológica computacional, química orgânica e medicina. Também foi mencionado que existem diversos modelos para representar estruturas de proteínas. Cada modelo tem suas peculiaridades e considera interações diferentes. Não existe um modelo que represente de maneira real o dobramento de proteínas, pois se trata de um processo ainda não completamente compreendido pelos cientistas e pesquisadores. Os modelos propostos tem diferentes níveis de detalhe e complexidade. O modelo mais simples é o HP mas apesar da sua simplicidade se apresenta como um problema NP-completo. O modelo HP será utilizado nesta proposta por conta de sua simplicidade de implementação, assim como o baixo custo computacional para realizar simulações do cálculo de energia. Utilizando o modelo HP diferentes maneiras de representar as variáveis existem. Nesta dissertação será utilizada a representação relativa pois é mencionado na literatura que esta tem uma maior capacidade de guiar algoritmos de busca a melhores resultados.

2.4 AEs - Algoritmos Evolucionários

Um algoritmo evolucionário (AE) é uma técnica de busca, altamente paralela, inspirada na teoria da seleção natural e reprodução genética de Charles Darwin. De acordo com a teoria de Darwin, a seleção natural irá favorecer os indíviduos que forem mais aptos, dessa maneira, estes indíviduos tem uma maior probabilidade de reprodução. Indivíduos com mais descendentes tem uma chance maior de perpetuarem seus códigos genéticos nas gerações futuras. O código genético é a identidade de cada indivíduo e é representado por cromossomos. Estes princípios são utilizados na implementação de algoritmos computacionais, que procuram por soluções melhores para um dado problema, evoluindo uma população de soluções codificadas em cromossomos artificais – estruturas de dados utilizadas para representar soluções factíveis para um dado problema [**?**].

De maneira geral, problemas reais de otimização possuem múltiplos objetivos a serem minimizados/maximizados e estão presentes em muitas áreas do conhecimento. Para otimizar problemas multiobjetivos, dois ou mais objetivos são considerados os quais geralmenta são conflitantes. Para estes problemas é impossível encontrar uma única solução ótima. Um conjunto de soluções é encontrado avaliando a dominância de Pareto [**?**] entre as soluções. O objetivo principal é encontrar o conjunto de soluções que sejam não dominadas entre si. Uma solução domina outra, se e somente se, for melhor em pelos um dos objetivos, sem ser pior em qualquer outro. Este conjunto de soluções constitui a fronteira de Pareto. Encontrar a fronteira real de Pareto é um problema NP-Completo [**?**], dessa maneira, o objetivo é encontrar uma boa aproximação da fronteira real.

Algoritmos Evolucionários Multi-Objetivos (AEMOs) são extensões de AEs para problemas multi-objetivos os quais aplicam conceitos da dominância de Pareto para criar diferentes estratégias para evoluir e manter a diversidade das soluções. Nesta dissertação foram explorados dois AEMOs: NSGAII [**?**] and IBEA [**?**].

2.4.1 NSGAII - Non-dominated sorting Genetic Algorithm II

O algoritmo 1 apresenta o pseudo código do NSGAII. O algoritmo recebe como parâmetro N o tamanho da população e T o número máximo de avaliações. O algoritmo inicia criando uma população com tamanho N chamada P_0. A população P_0 é classificada de acordo com aptidão e a relação de não dominância. A população P_0 é submitida ao operador de seleção: torneio binário para selecionar duas soluções que serão utilizadas para gerar descendentes. Operadores de cruzamento e mutação são aplicados na soluções selecionadas gerando duas soluções distintas descendentes. Ao fim do processo de reprodução as soluções descendentes são avaliadas e adicionadas a população chamada Q_0.

Após esta estapa, P_0 e Q_0 são adicionadas em uma população auxiliar chamada R. Utilizando o conceito de não dominância, R é ordenada criando fronteiras, onde cada solução da primeira fronteira não é dominada por nenhuma outra solução, já soluções da segunda fronteira são dominadas apenas por soluções contidas na primeira fronteira, e assim por diante. Para cada fronteira, as soluções são avaliadas utilizando um mecanismo de *Crowding-Distance* as soluções com maiores valores irão ser adicionadas na população da próxima geração chamada P_t onde t é a avaliação corrente.

Após criar e preencher P_t com as soluções não dominadas de todas as fronteiras, a população P_t é avaliada e então passa para um novo torneio binário e reprodução, dessa maneira, iniciando uma novo ciclo do algoritmo.

2.4.2 IBEA (Indicator-Based Evolutionary Algorithm)

No contexto de otimização multiobjetiva, otimizar consiste em tentar encontrar a fronteira com uma boa aproximação da fronteira real de Pareto. Entretanto, não existe uma definição geral para "uma boa aproximação". Consequentemente, indicadores de qualidade vem sendo utilizados para avaliar a qualidade da aproximação de fronteiras. O indicador *hypervolume* é um exemplo de indicador utilizado para avaliação e comparação das fronteiras.

No algoritmo IBEA, indicadores de qualidade são utilizados para avaliar o conjunto de soluções não dominadas [**?**]. Para utilizar o IBEA, é necessário definir qual indicador será utilizado para associar cada solução a um valor scalar. Um dos indicadores bastante utilizados

Algoritmo 1 NSGAII

```
1: N ← Population Size
2: T ← Max evaluations
3: P_0 ← CreatePopulation(N);
4: CalculateFitness(P_0);
5: FastNonDominatedSort(P_0);
6: Q_0 ← 0
7: while Q_0 < N do
8:     Parents ← BinaryTournament(P_0);
9:     Offspring ← CrossoverMutation(Parents);
10:    Q_0 ← Offspring
11: end while
12: CalculateFitness(Q_0);
13: t ← 0
14: while t < T do
15:    R_t ← P_t ∪ Q_t;
16:    Fronts ← FastNonDominatedSort(R_t);
17:    P_{t+1} ← 0
18:    i ← 0
19:    while P_{t+1} + Front_i < N do
20:        CrowdingDistanceAssignment(Front_i);
21:        P_{t+1} ← P_{t+1} ∪ Front_i
22:        i ← i + 1
23:    end while
24:    CrowdingDistanceSort(Front_i);
25:    P_{t+1} ← P_{t+1} ∪ Front_i[1 : (N − P_{t+1})]
26:    Parents ← BinaryTournament(P_{t+1});
27:    Q_{t+1} ← CrossoverMutation(Parents);
28:    t ← t + 1
29: end while
30: return P ← Set of non-dominated solutions.
```

em conjunto com IBEA é o *hypervolume* por conta da sua capacidade de avaliar a convergência e a diversidade do espaço de busca ao mesmo [**?**].

$$F(x_i) = \sum_{x_j \in (P - x_i)} -e^{\frac{-I_{Hy}(x_j, x_i)}{k}} \tag{2.2}$$

A equação de *fitness* do IBEA é apresentada pela equação 2.2 e é utilizada para calcular a contribuição de uma dada solução para o valor do indicador referente a população, onde k é um fator escalar dependente do I_{Hy}, o qual representa o indicador de qualidade sendo utilizado. O valor $F(x_i)$ corresponde à perda de qualidade da aproximação, da fronteira real de Pareto, se a solução x_i for removida da população [**?**].

O Algoritmo 2 recebe como parametro o tamanho da população N, o número máximo de avaliações T e o fator escalar k. O processo se inicia criando uma população P de tamanho N. Até que o critério de parada seja atingido os seguintes passos irão se repetir: um torneio binário

para selecionar indivíduos, reprodução (cruzamento e mutação) dos indivíduos selecionados para gerar descendentes, adicionar os descedentes na população auxiliar $\overline{P}$. Após a reprodução, a população $\overline{P}$ é unida com P. Enquanto o tamanho de P exceder N, o pior indíviduo avaliado pela equação 2.2 é removido da população P e os indíviduos restantes são re-avaliados. Quando o algoritmo terminar irá retornar o conjunto de soluções não dominadas é retornado.

Algoritmo 2 IBEA

```
1:  N ← Population Size
2:  N̄ ← AuxiliaryPopulationSize
3:  T ← Max Evaluations
4:  k ← Scale factor of Fitness
5:  P ← CreatePopulation(N);
6:  P̄ ← CreateEmptyAuxiliaryPopulation(N̄);
7:  m ← 0
8:  CalculateFitness(P);
9:  while m ≥ T or other stop criterion is not reached do
10:     P̄ ← BinaryTournament(P);
11:     P̄ ← CrossoverMutation(P̄);
12:     P ← P ∪ P̄
13:     m ← m + 1
14:     while Size(P) > N do
15:         x* ← WorstIndividualByFitness();
16:         RemoveFromPopulation(x*, P);
17:         CalculateFitness(P);
18:     end while
19: end while
20: return P ← Set of non-dominated solutions
```

2.5 Hiper-Heurísticas

Apesar do sucesso de métodos heurísticos e outros métodos de busca na tarefa de resolver problemas de busca computacional difíceis ainda existem dificuldades em generalizar estes métodos para diferentes problemas ou até mesmo para diferentes instâncias de um mesmo problema. Esta dificuldade provém principalmente da necessidade de selecionar os parâmetros e configurações mais adequados dos algoritmos para um problema ou para uma dada instância de um problema. Também vale mencionar a pouca orientação na tarefa de definir estes parâmetros. É neste contexto que surge uma questão: é possível automatizar o projeto e parametrização de métodos heurísticos para resolver problemas de busca computacional difíceis? [**?**]. A ideia principal é desenvolver algoritmos que sejam mais genéricos do que as implementações de metodologias atuais [**?**]. As principais abordagens já propostas para este desafio podem ser classificadas em duas categorias: configuração estática (*offline*) e controle dinâmico (*online*). Abaixo são apresentadas algumas abordagens já propostas na literatura:

- Configuração Estática (Offline):
  - Seleção de algoritmos;
  - Portfólio de algoritmos;
  - Configuração de algoritmos;
  - Ajuste de parâmetros;
  - Hiper-Heurísticas.
- Controle Dinâmico (Online):
  - Seleção adaptativa de operadores (AOS);
  - Controle de parâmetros;
  - Algoritmos meméticos adaptativos;
  - Hiper-Heurísticas

Esta seção tratará apenas de hiper-heurísticas e suas particularidades. Uma hiper-heurística pode ser vista como uma metodologia de alto nível, a qual seleciona ou cria heurísticas para resolver um dado problema ou instância de um problema. [**?**]. O objetivo principal é tentar encontrar ou construir a heurística mais adequada para cada situação. As hiper-heurísticas diferem dos métodos padrão de busca pois operam sobre o espaço de busca de heurísticas que por sua vez operam sobre o espaço de busca de um problema. Além disso, hiper-heurísticas são independentes do problema. Tradicionalmente frameworks hiper-heurísticos possuem dois níveis: [**?**]

Heurísticas de baixo nível: Um conjunto de heurísticas de baixo nível específicas. Estas heurísticas costumam diferir entre domínios de problemas. São exemplos: operadores de cruzamento, mutação e buscas locais. Em alguns casos, meta-heurísticas também, dependendo da modelagem do *framework* hiper-heurístico, podem assumir o papel de heurísticas de baixo nível.

Heurísticas de alto nível: Geralmente consistem em dois componentes: mecanismo de seleção, que gerencia quais heurísticas de baixo nível devem ser aplicadas durante a busca; um critério de aceitação, que tem a responsabilidade de decidir se irá aceitar ou não uma solução gerada, a partir da aplicação de uma heurística de baixo nível. A responsabilidade do mecanismo de seleção é selecionar, de um *pool* de heurísticas de baixo nível, a heurística que for mais adequada naquele momento. Geralmente, a escolha da heurística de baixo nível é crucial para uma boa exploração do espaço de busca, evitando que a busca fique confinada em uma região específica [**?**]. O objetivo do critério de aceitação é auxiliar o processo de busca a evitar mínimos locais assim como explorar diferentes regiões através do aceite ou rejeição de soluções geradas [**?**]. Espera-se que um bom critério de aceitação deva atingir um bom equilíbrio entre aceitar

soluções melhores assim como soluções diversificadas se a busca estiver presa em um mínimo local [**?**]. Ambos os componentes devem ser independentes de conhecimento sobre o problema.

A imagem 2.1 apresenta um diagrama exemplificando os níveis de um *framework* hiper-heurístico e suas características. Note que entre os níveis (alto e baixo) existe uma barreira de domínio, ou seja, apenas as heurísticas de baixo nível são dependentes de conhecimento do problema ou instância e as heurísticas de alto nível não são dependentes do problema.

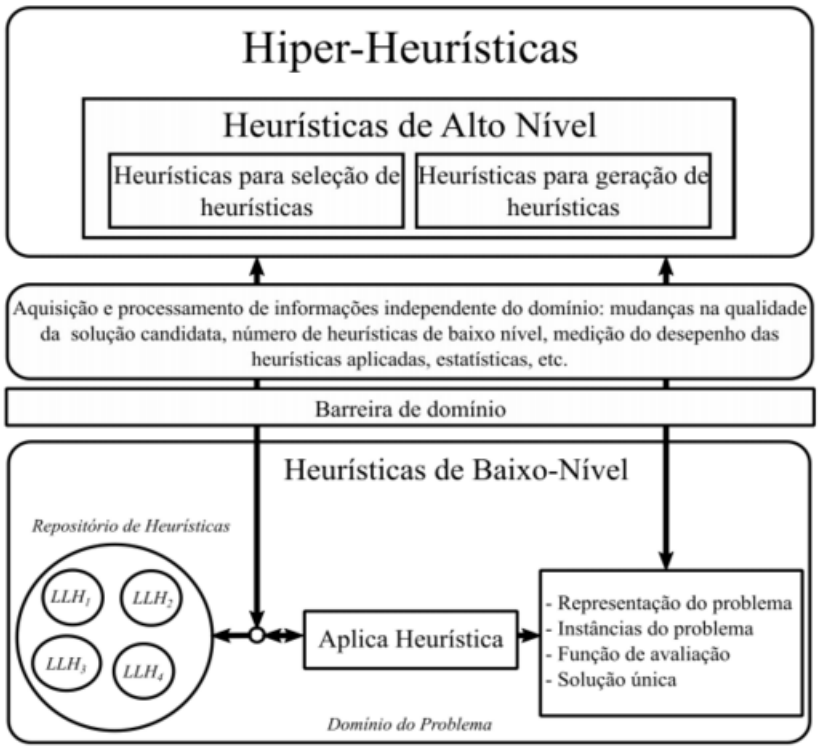

Figura 2.1: Framework Geral Hiper-Heurístico. Adaptado de [**?**]

Como cada instância ou problema possui um espaço de busca com diferentes características, os componentes da heurística de alto nível têm um grande impacto no desempenho de um *framework* hiper-heurístico. Esta é uma das razões de existir um grande interesse de pesquisa em desenvolver novos mecanismos de seleção, assim como diferentes critérios de aceitação [**?**]. Um bom mecanismo de seleção deve selecionar a heurística mais adequada em um dado momento, para guiar a busca para regiões promissoras do espaço de busca. Ao utilizar hiper-heurísticas, espera-se encontrar o método correto ou a sequência de heurísticas que mais se adequam a um problema ou instância ao invés de tentar resolver o problema diretamente. Entretanto, um importante objetivo é desenvolver métodos genéricos, que têm potencial em produzir soluções com uma qualidade aceitável, utilizando um conjunto de heurísticas de baixo nível fácil de implementar. As hiper-heurísticas podem ser classificadas de diversas maneiras. A figura 2.2 apresenta as possíveis classificações descritas na literatura.

A primeira classificação de hiper-heurísticas é baseada na sua fonte de conhecimento durante a busca: *Online* é quando a hiper-heurística toma decisões de maneira instantânea, baseando-se em métricas durante sua execução, não necessitando de treinamento prévio. *Offline*

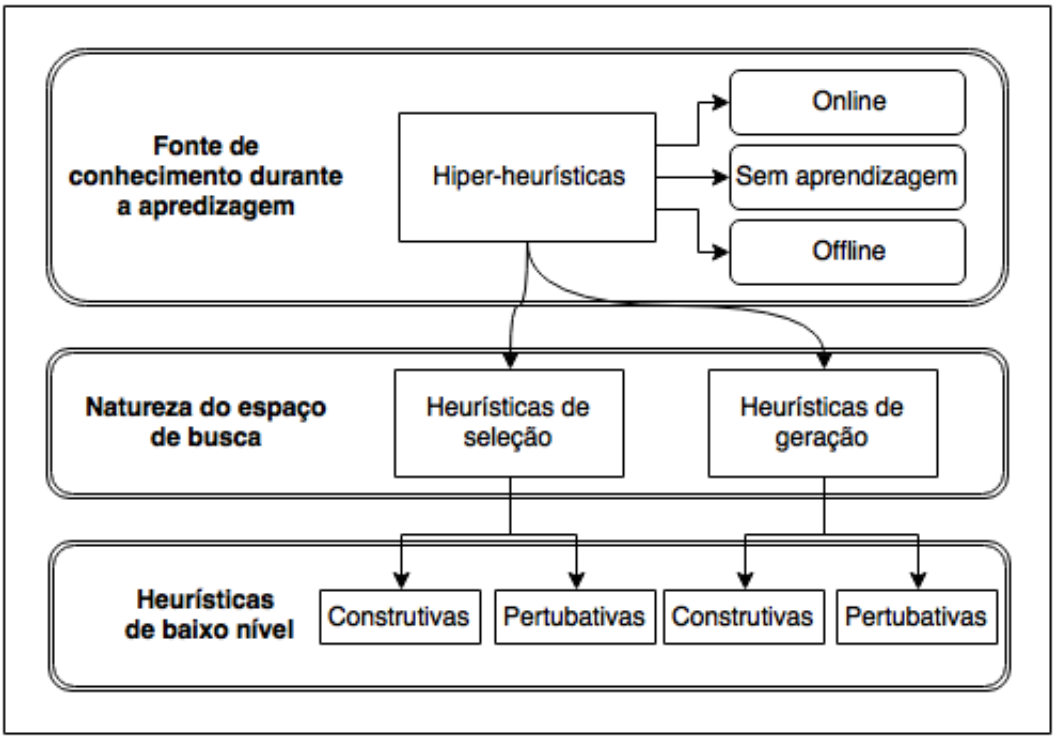

Figura 2.2: Classificação Hiper-heurísticas. Adaptado de [**?**]

necessita de treinamento prévio; estes *frameworks* tomam suas decisões baseados no que foi aprendido apenas durante o treinamento, sem atualização deste conhecimento. Os *frameworks* classificados como *No-Learning* não possuem nenhuma forma de aprendizagem. Outra classificação considera como as heurísticas de baixo nível operam sobre as soluções do problema. As heurísticas ditas perturbativas realizam pequenas perturbações nas soluções gerando novas soluções. Já heurísticas construtivas criam soluções do zero passo a passo e normalmente avaliam cada etapa da construção para obter *feedback* sobre o seu desempenho. Uma última classificação, mas não menos importante, divide as hiper-heurísticas de acordo com a natureza do seu espaço de busca. As hiper-heurísticas de seleção selecionam sequências de heurísticas a serem aplicadas para resolver um dado problema ou instância. Já as hiper-heurísticas de geração operam gerando novas heurísticas com objetivo de resolver um problema ou instância.

2.5.1 Hiper-heurísticas de Geração

Estas hiper-Heurísticas geram novas heurísticas combinando componentes de heurísticas existentes. Geralmente se utiliza programação genética (GP), ou alguma vertente de GP, como por exemplo evolução gramatical [**?**] ou programação gênica [**?**], como hiper-heurística para gerar heurísticas. A próxima sub-seção irá introduzir o conhecimento necessário para a compreensão da programação genética, assim como irá introduzir evolução gramatical, que se trata de um tipo de programação genética e que será utilizada nesta proposta.

2.6 Programação Genética (PG)

Programação Genética [**?**] é um ramo da síntese de programas que utiliza ideias oriundas da teoria da evolução natural para produzir programas. Os principais componentes da computação evolucionária são herança (cruzamento/reprodução), seleção e variação (mutação). A herança significa que os descendentes têm alguma semelhança com seus pais, pois quase todo material genético vem dos pais. A seleção trata de escolher quais pais irão se reproduzir para gerar novos descendentes; pais com maior aptidão tendem a ter maior probabilidade de serem selecionados. Esta pressão de seleção define quais indivíduos estão mais aptos que outros. Variação realiza pequenas alterações em um descendente a fim de criar novo material genético neste indivíduo e que não estava presente em nenhum dos indivíduos que o geraram. Computação evolutiva pode ser pensada como a interação destes três componentes. Uma população aleatória de programas de computador é gerada, e os operadores geneticamente inspirados (cruzamento e mutação) são repetidamente aplicados com objetivo de produzir novos programas de computador. Estes programas são avaliados utilizando uma função de *fitness* (normalmente dependente do desempenho obtido pela aplicação do programa em um problema), que determina quais destes programas são mais suscetíveis a sobreviver para gerações futuras. Os programas com maior aptidão tem mais chances de serem selecionados para o cruzamento e perpetuarem parte de seus códigos genéticos durante o processo evolutivo. Programação genética é um método de geração de programas sintaticamente válidos e a função de *fitness* é utilizada para decidir quais programas são mais adequados para o problema. Na programação genética, os programas que compõem a população são tradicionalmente representados utilizando estruturas de árvore. Existem outras estruturas que podem ser evoluídas, como por exemplo: sequências lineares de instruções ou gramáticas. Nesta proposta será utilizada uma representação gramatical linear que será explicada na seção 2.7.

2.7 Evolução Gramatical (EG)

Evolução gramatical é uma técnica relativamente nova de computação evolutiva, proposta por Ryan et al. [**?**], trata-se de um tipo de programação genética. Assim como na programação genética, o principal objetivo é encontrar um programa executável ou trecho de um programa, que obtenha um bom valor de *fitness* para o problema em questão. Na maioria dos trabalhos publicados de programação genética, expressões que representam estruturas de árvore são manipuladas, enquanto na evolução gramatical os operadores genéticos são aplicados em vetores de inteiros que posteriormente são mapeados para um programa (ou trecho de programa) através de uma gramática específica. Um dos benefícios de EG é que este mapeamento generaliza a aplicação para diferentes linguagens de programação. Ryan el al. [**?**] propõem uma técnica para gerar programas ou fragmentos de programas para qualquer linguagem de programação utilizando definições BNF. A técnica pode ser utilizada para evoluir programas por um processo evolutivo. A evolução

gramatical adota um mecanismo de mapeamento entre o genótipo (indivíduos codificados em um vetor de inteiros) e o fenótipo (programas gerados para resolver algum problema). A notação *Backus Naur Form* (BNF) é a notação utilizada para expressar a gramática de uma linguagem na forma de regras de produção. Uma gramática BNF consiste em um conjunto de terminais, os quais são itens que podem aparecer na linguagem, por exemplo: +, -, *, / etc e não terminais, que podem ser expandidos em um ou mais terminais e não terminais. Uma gramática pode ser expressada como uma tupla N, T, P, S, onde N é o conjunto de não terminais, T o conjunto de terminais, P um conjunto de regras de produção que mapeia os elementos N para T; e, por último, S, um símbolo de início e que está contido em N.

$$N = \langle expr \rangle, \langle op \rangle, \langle pre - op \rangle$$
$$T = Sin, Cos, Tan, Log, +, -, /, *, X$$
$$S = \langle expr \rangle$$

E P pode ser representada como:

$\langle expr \rangle$	::=	$\langle expr \rangle \langle op \rangle \langle expr \rangle$	(0)
	|	$(\langle expr \rangle \langle op \rangle \langle expr \rangle)$	(1)
	|	$\langle pre\text{-}op \rangle (\langle expr \rangle)$	(2)
	|	$\langle var \rangle$	(3)
$\langle op \rangle$	::=	+	(0)
	|	-	(1)
	|	/	(2)
	|	*	(3)
$\langle pre\text{-}op \rangle$	::=	Sin	(0)
	|	Cos	(1)
	|	Tan	(2)
	|	Cos	(3)
$\langle var \rangle$	::=	X	(0)

Gramática 2.1: Gramática exemplo para demonstrar como decodificar vetores de inteiros em programas de computador.

Ryan et al. [**?**] propôs o uso de um algoritmo genético (AG) para controlar quais escolhas devem ser feitas, permitindo dessa maneira que o AG controle quais regras de produção serão utilizadas. Um indivíduo (cromossomo) consiste em um vetor de tamanho variável de valores inteiros que representa o genótipo. Para fins de compreensão o processo de mapeamento de um cromossomo será demonstrado utilizando a Gramática 2.1 apresentada anteriormente nesta seção. O Algoritmo 3 apresenta o *template* geral dos programas gerados pela Gramática 2.1. A

Tabela 2.1: *Regras de produção* e o número de escolhas para cada uma.

Regra de produção	Número de escolhas
$\langle expr \rangle$	4
$\langle op \rangle$	4
$\langle pre - op \rangle$	4
$\langle var \rangle$	1

expressão $\langle expr \rangle$ apresentada na linha 2 do Algoritmo 3 é substituída por expressões matemáticas que estão codificadas pelos cromossomos (vetores de inteiros).

Algoritmo 3 *Template* geral dos algoritmos gerados

```
float symb(float x)
a = ⟨expr⟩;
return a;
```

Suponha o seguinte vetor de inteiros:

$$[220, 203, 17, 6, 108, 215, 104, 30]$$

Este vetor será utilizado para mapear o cromossomo (genótipo) em um trecho de programa (fenótipo) utilizando a gramática BNF. A tabela Table 2.1 resume o número de escolhas associada à cada regra de produção da Gramática 2.1. Existem 4 opções de regras de produção que podem ser selecionadas para a expressão $\langle expr \rangle$. Para decidir qual será selecionada, o primeiro valor no cromossomo deve ser utilizado. O valor é 220. Devemos realizar o módulo deste valor pelo número de escolhas, neste caso 4. Portanto, 220 MOD 4 = 0, o que significa selecionar a primeira opção: $\langle expr \rangle \langle op \rangle \langle expr \rangle$.

Note que a primeira expressão é novamente $\langle expr \rangle$ e da mesma maneira devemos obter o próximo valor de inteiro e realizar o módulo. O próximo valor inteiro é 203; realizando o modulo de 4, resulta em 3, que portanto seleciona a quarta opção: $\langle var \rangle$. Substituindo na expressão anterior, obtemos: $\langle var \rangle \langle op \rangle \langle expr \rangle$

Nenhuma escolha é necessária para a expressão $\langle var \rangle$, pois existe apenas uma opção X. A expressão pode ser reescrita da seguinte maneira: $X \langle op \rangle \langle expr \rangle$

Neste momento é necessário decodificar a expressão não terminal $\langle op \rangle$. Obtendo o próximo valor inteiro do cromossomo, temos 17 e para o $\langle op \rangle$ temos 4 opções $(+| - |/|*)$. O resultado de 17 MOD 4 é igual a 1, que significa selecionar: $-$. Substituindo na expressão, temos: $X - \langle expr \rangle$

Novamente é necessário fazer uma nova escolha para resolver a expressão não terminal $\langle expr \rangle$. O próximo valor do cromossomo é 6 e novamente existem 4 opções. Realizando o modulo 6 MOD 4, obtém-se 2, que seleciona $\langle pre - op \rangle (\langle expr \rangle)$. Atualizando a expressão, obtemos: $X - \langle pre - op \rangle (\langle expr \rangle)$

Resolvendo a expressão $\langle pre - op \rangle$, obtemos 108 MOD 4 = 0 que por sua vez seleciona a primeira expressão terminal Sin. Atualizando a expressão, obtemos: $X - Sin(\langle expr \rangle)$

Expandindo $\langle expr \rangle$, obtemos 215 MOD 4 = 3, que seleciona a expressão não terminal $\langle var \rangle$. Já que para a expressão $\langle var \rangle$ existe apenas uma opção, nenhuma escolha é necessária e a expressão final decodificada (fenótipo) é: $X - Sin(X)$

Note que nem todos os genes do cromossomo foram necessários para obter o fenótipo. Nos casos em que isto ocorre, os genes que não forem utilizados são desconsiderados. Além disso, pode ocorrer que um cromossomo não tenha genes suficientes para mapear um programa. Neste caso a estratégia é reutilizar os genes do cromossomo a partir do primeiro gene.

Operadores genéticos tradicionais (cruzamento e mutação) também são utilizados na EG. Além dos operadores tradicionais outros dois operadores *Prune* e *Duplicate* são peculiares à EG e serão descritos em seguida:

- *Duplicate*: Este operador (dada uma probabilidade) realiza a cópia de alguns genes. Os genes duplicados são adicionados após a última posição do cromossomo. O número de genes a serem duplicados é selecionado de maneira aleatória. A motivação por trás deste operador é que ao duplicar genes ocorre um aumento da presença de genes que são potencialmente bons, pois pertencem a um indivíduo com boa aptidão selecionado pelo operador de seleção.

- *Prune* : Este operador leva em consideração que nem sempre todos os genes, de um cromossomo, são utilizados para decodificar um programa. Dessa maneira (dada uma probabilidade) realiza o truncamento de cromossomos. O objetivo é diminuir a probabilidade que o operador de cruzamento opere em regiões dos cromossomos que não sejam utilizadas realmente.

O Algoritmo 4 apresenta o pseudocódigo da evolução gramatical (EG). Note que o pseudocódigo é muito similar a um algoritmo genético simples. Nas linhas 3 e 4 ocorre a inicialização da população e o mapeamento para programas utilizando a gramática que foi provida como entrada. Em seguida, na linha 5 ocorre a execução dos programas e na linha 6 acontece a avaliação dos indivíduos da população, baseando-se na saída obtida pelos respectivos programas. Dentro do laço principal, apresentado na linha 7, podemos observar o processo de seleção dos indivíduos pais na linha 8 e na linha 9 o processo de cruzamento destes indivíduos. Nas linhas 10 e 11 ocorre a aplicação dos operadores *Prune* e *Duplicate* respectivamente e na linha 12 podemos observar a aplicação do operador de mutação. Em seguida, nas linhas 13,14 e 15 ocorre o mapeamento dos indivíduos descendentes para programas, execução dos programas e finalmente a atribuição de *fitness* para os descendentes. Por fim, na linha 16 do laço principal, ocorre a substituição dos descendentes na população.

Algoritmo 4 Pseudocódigo da evolução gramatical

1: $AG \leftarrow$ Arquivo da gramática;
2: $populacao \leftarrow$ Inicialização a população;
3: $programas \leftarrow$ Mapeia $populacao$ para programas utilizando AG;
4: Executa os $programas$;
5: Atribui valor de *fitness* para as soluções of $populacao$ de acordo com a saída obtida pelos respectivos programas decodificados;
6: **while** Condição de parada não atingida **do**
7: $pais \leftarrow$ Seleção de indivíduos para cruzamento;
8: $descendentes \leftarrow$ Cruzamento $pais$;
9: Aplica o operator Prune nas soluções $descendentes$;
10: Aplica o operador *Duplicate* nas soluções $descendentes$;
11: Aplica o operador de mutação nas soluções $descendentes$;
12: $programas \leftarrow$ Mapeia $descendentes$ para programas utilizando AG;
13: Executa $programas$;
14: Atribui valor *fitness* para as soluções $descendentes$ de acordo com a saída obtida pelos respectivos programas decodificados;
15: $populacao \leftarrow$ Realiza substituição;
16: **end while**
17: **return** Melhor programa da $populacao$;

2.8 Programação Genética como Hiper-Heurística de Geração de Heurísticas

Nesta seção serão apresentadas questões relativas ao uso de EG como mecanismo de geração de heurísticas. Burke et al. [**?**] descrevem que muitos autores mencionam a melhor adequação de programação genética, em relação a outras técnicas de aprendizagem de máquina, para gerar heurísticas de maneira automática. Burke et al [**?**] também apontam algumas vantagens desta técnica:

- PG utiliza cromossomos de tamanho variável. Geralmente, não se sabe um tamanho ótimo para representar heurísticas de um dado domínio de problema.
- PG produz estruturas de dados executáveis. E heurísticas são tipicamente expressadas como programas ou algoritmos.
- Facilidade em identificar boas características do domínio do problema, afim de definir o conjunto terminal que será utilizado pela PG.
- Heurísticas desenvolvidas por humanos podem facilmente ser expressadas na mesma linguagem utilizada para criar o espaço de busca da PG. O conjunto de funções, relevante para o problema pode ser determinado facilmente. E adicionalmente PG pode ser suplementada com uma gramática específica.

Todas estas vantagens descritas por Burke et al. [**?**] também são consideradas ao utilizar EG, visto que se trata de uma extensão de programação genética e possui as mesmas características (cromossomo de tamanho variável, produz estruturas executáveis, etc). Burke et al. [**?**] também mencionam desvantagens, por exemplo: a cada execução da programação genética é encontrada uma melhor heurística que, por se tratar de uma técnica estocástica, os resultados podem ser distintos em diferentes execuções. Portanto se fazem necessárias múltiplas execuções, a fim de se obter um melhor conhecimento da qualidade das heurísticas que podem ser produzidas. Outra desvantagem é referente à configuração de parâmetros, que normalmente é encontrada via tentativa e erro.

Abordagem Básica

Burke et al. [**?**] descrevem uma abordagem básica para aplicar programação genética para gerar heurísticas:

1. Examinar as heurísticas existentes: Avaliar se as heurísticas já propostas para um dado problema podem ser descritas em um *framework* comum. Estas heurísticas podem ter sido criadas por humanos ou até mesmo concebidas via outras técnicas de aprendizagem. Este passo não é trivial, pois envolve o entendimento de um número diverso de heurísticas existentes, que podem operar de diferentes maneiras. Geralmente heurísticas desenvolvidas por humanos são produtos de anos de pesquisa, portanto uma boa compreensão das heurísticas existentes pode ser um trabalho difícil.

2. Um framework que utilizará as heurísticas: neste momento a preocupação é em como as heurísticas serão aplicadas para um dado problema. Em geral, os frameworks tendem a ser bem diferentes dependendo do domínio do problema.

3. Definição do conjunto terminal: neste passo a preocupação refere-se a variáveis que expressem o estado do problema. Estas variáveis irão compor os terminais da programação genética/evolução gramatical. Outros terminais também podem ser utilizados. Particularmente, constantes aleatórias podem ser úteis.

4. Definição do conjunto de funções: é necessário definir como as variáveis estarão relacionadas ou combinadas entre si. Estes relacionamentos irão compor o conjunto de funções da programação genética/evolução gramatical.

5. Identificar uma função de *fitness*: uma função de *fitness* precisa ser identificada para o problema. Geralmente, uma função simples de aptidão não irá avaliar bem os cromossomos. Introduzir alguns parâmetros pode ajudar a encontrar uma mais adequada.

6. Executar o framework: geralmente ao executar pela primeira vez um framework hiper-heurístico com programação genética, não serão produzidos bons resultados, devido à

escolha dos parâmetros. Isto é observado especialmente em casos que o pesquisador é iniciante. Portanto é essencial que as definições de parâmetros sejam cuidadosamente investigadas.

2.9 Considerações Finais

Neste capítulo foram apresentados os conceitos que permeiam a área de estudo sobre hiper-heurísticas, tendo sido discutidos os seus níveis (alto e baixo) e as classificações encontradas na literatura. Foram discutidas algumas estratégias para hiper-heurísticas de seleção e geração. As hiper-heurísticas de geração foram mais detalhadas, pois esta proposta visa o projeto automático de heurísticas de alto nível. Foram apresentados os conceitos de PG e sua extensão EG, por se tratarem de estratégias comumente utilizadas para o projeto de hiper-heurísticas de geração de heurísticas. Também foram discutidas algumas vantagens e desvantagens referentes ao uso de PG para geração de heurísticas, além de demonstrar que a EG possui as mesmas características da PG, pois se trata de uma extensão que utiliza uma gramática para gerar os programas. O funcionamento geral da EG foi demonstrado utilizando uma gramática exemplo e um vetor de inteiros e, por fim, o pseudocódigo da evolução gramatical foi apresentado. O **??** apresenta o problema de dobramento de proteínas e o **??** apresentará a proposta da aplicação de EG a este problema.

2.10 Exemplo de figura

A forma sugerida para incluir figuras em um documento LaTeX é importá-las usando o pacote `graphicx`. Como formatos gráficos sugere-se:

- Formatos *raster*, como PNG (*Portable Network Graphics*) ou JPG (*Joint Photographic Experts Group*) para fotografias; procure usar uma resolução de ao menos 150 dpi (*dots per inch*).
- Formatos vetoriais, como PDF (*Portable Document Format*) ou EPS (*Extended PostScript*) para diagramas e gráficos[1].

A maior parte das ferramentas permite exportar figuras nesses formatos (a figura do exemplo foi produzida com o *Inkscape*, um programa livre multiplataforma). A figura 2.3 mostra um exemplo de inclusão de figura em PDF.

Para mais informações consulte [Goossens et al., 1993].

[1]NUNCA use JPG ou GIF para desenhos vetoriais, pois o resultado final geralmente fica borrado.

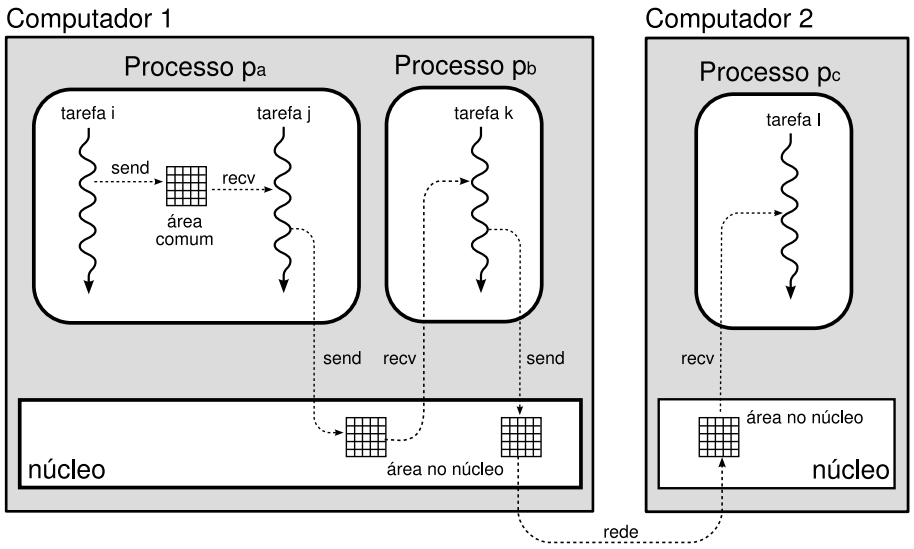

Figura 2.3: Comunicação inter-processos.

2.11 Exemplo de tabela

Tabelas são elementos importantes de um documento. No LaTeX as tabelas podem ser objetos flutuantes (definidas no ambiente `table` e referenciadas por números usando `label` e `ref`) ou objetos fixos simples, criados pelo ambiente `tabular`. A tabela 2.2 é um exemplo de tabela flutuante, cuja posição no texto pode variar em função das quebras de página.

Tabela 2.2: Os 16 modelos centrais do UCON_{ABC}

	0 (imutável)	1 (*pre-update*)	2 (*on-update*)	3 (*pos-update*)
`preA`	•	•	–	•
`onA`	•	•	•	•
`preB`	•	•	–	•
`onB`	•	•	•	•
`preC`	•	–	–	–
`onC`	•	–	–	–

2.12 Exemplo de equação

Equações destacadas devem ser numeradas como mostra a equação 2.3:

$$E = m \times c^2 \tag{2.3}$$

2.13 Exemplos de código-fonte

Códigos-fonte podem ser produzidos de forma simples através do ambiente `verbatim`, como mostra este exemplo:

```
#include <stdio.h>

int main (int argc, char *argv[])
{
   int i ;                          // uma variavel local

   for (i=0; i< 100; i++)           // um laço for
      printf ("i vale %d\n", i) ;   // uma saída na tela
}
```

No entanto, é preferível usar pacotes especializados para a edição ou inclusão de códigos-fonte, como o pacote `listings`. Eis um exemplo de código-fonte escrito com esse pacote:

```
1 #include <stdio.h>
2
3 int main (int argc, char *argv[])
4 {
5   int i ;                   // uma variável local
6
7   for (i=0; i< 100; i++)    // um laço for
8     printf ("i vale %d\n", i) ; // uma saída na tela
9 }
```

Esse pacote também permite incluir códigos-fonte de arquivos externos. Eis um exemplo:

```
1 #include <stdio.h>
2
3 int main (int argc, char *argv[])
4 {
5   int i ;                   // uma variável local
6
7   for (i=0; i< 100; i++)    // um laço for
8     printf ("i vale %d\n", i) ; // uma saída na tela
9 }
```

2.14 Conclusão

Todo capítulo (com exceção da introdução e da conclusão) deve encerrar com uma pequena conclusão local, resumindo os tópicos apresentados no capítulo e preparando o leitor para o próximo capítulo (exceto se esse for a conclusão geral). Caso o capítulo tenha apresentado resultados obtidos pelo próprio autor, estes devem ser sucintamente relembrados aqui.

Referências Bibliográficas

[Goossens et al., 1993] Goossens, M., Mittelbach, F. e Samarin, A. (1993). *The LaTeX Companion*. Addison-Wesley.

[Maziero, 2015] Maziero, C. (2015). Modelo PPGInf UFPR para teses e dissertações. `http://www.inf.ufpr.br/maziero`. Acessado em 30/11/2015.

[UFPR, 2015] UFPR, B. (2015). Manual de normalização de documentos científicos de acordo com as normas da ABNT. Relatório Técnico ISBN 9788584800025, Sistema de Bibliotecas – Universidade Federal do Paraná, Curitiba PR.

Apêndice A

Exemplo de anexo

Os apêndices são uma extensão do texto, destacados deste para evitar descontinuidade na sequência lógica ou alongamento excessivo de determinado assunto ou tópico secundário dentro dos capítulos da dissertação ou da tese. São contribuições que servem para esclarecer, complementar, provar ou confirmar as ideias apresentadas no texto dos capítulos e que são importantes para a compreensão dos mesmos.

Todos os apêndices devem vir após as referências bibliográficas e devem ser enumerados por letras maiúsculas (A, B, C, ...).

A.1 Uma Seção

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam

et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

A.1.1 Uma sub-Seção

Sed mattis, erat sit amet gravida malesuada, elit augue egestas diam, tempus scelerisque nunc nisl vitae libero. Sed consequat feugiat massa. Nunc porta, eros in eleifend varius, erat leo rutrum dui, non convallis lectus orci ut nibh. Sed lorem massa, nonummy quis, egestas id, condimentum at, nisl. Maecenas at nibh. Aliquam et augue at nunc pellentesque ullamcorper. Duis nisl nibh, laoreet suscipit, convallis ut, rutrum id, enim. Phasellus odio. Nulla nulla elit, molestie non, scelerisque at, vestibulum eu, nulla. Ut odio nisl, facilisis id, mollis et, scelerisque nec, enim. Aenean sem leo, pellentesque sit amet, scelerisque sit amet, vehicula pellentesque, sapien.

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet

erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.