UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

PEDRO DEMASI

HEURÍSTICAS BASEADAS EM APOSTAS PARA PROBLEMAS DE OTIMIZAÇÃO COMBINATÓRIA

Rio de Janeiro
2015

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

PEDRO DEMASI

HEURÍSTICAS BASEADAS EM APOSTAS PARA PROBLEMAS DE OTIMIZAÇÃO COMBINATÓRIA

Tese de Doutorado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Doutor em Informática.

Orientador: Jayme Luiz Szwarcfiter

Co-orientador: Adriano Joaquim de Oliveira Cruz

Rio de Janeiro
2015

CBIB Demasi, Pedro

Heurísticas Baseadas em Apostas para Problemas de Otimização Combinatória / Pedro Demasi. – 2015.
139 f.: il.

Tese (Doutorado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Programa de Pós-Graduação em Informática, Rio de Janeiro, 2015.

Orientador: Jayme Luiz Szwarcfiter.
Co-orientador: Adriano Joaquim de Oliveira Cruz.

1. Metaheurísticas. 2. Teorias de apostas. 3. Otimização combinatória. – Teses. I. Szwarcfiter, Jayme Luiz (Orient.). II. Cruz, Adriano Joaquim de Oliveira (Co-orient.). III. Universidade Federal do Rio de Janeiro, Instituto de Matemática, Programa de Pós-Graduação em Informática. IV. Título

CDD

PEDRO DEMASI

Heurísticas Baseadas em Apostas para Problemas de Otimização Combinatória

Tese de Doutorado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Doutor em Informática.

Aprovado em: Rio de Janeiro, _____ de _____ de _____.

Prof. Dr. Jayme Luiz Szwarcfiter (Orientador)

Prof. Dr. Adriano Joaquim de Oliveira Cruz (Co-orientador)

Prof. Dr. Josefino Cabral Melo Lima

Prof. Dr. Fábio Protti

Prof. Dr. Luiz Satoru Ochi

AGRADECIMENTOS

Agradeço, primeiramente, à minha esposa, Juliana Demasi, por estar sempre ao meu lado em todas as situações, boas ou ruins, e por constantemente me fazer aprender novas coisas e me fazer uma pessoa melhor.

Agradeço, também, a meus orientadores, Jayme e Adriano, por tudo o que fizeram por mim, e por todo o esforço e dedicação.

Agradeço a meus pais não só por sua importância indiscutível na minha vida, como também, pela influência na minha educação e formação.

Gostaria, finalmente, de agradecer à secretaria do PPGI, em especial ao Aníbal e à Adriana, por toda a ajuda e paciência para resolver problemas e explicar procedimentos, sempre que necessário.

RESUMO

Demasi, Pedro. **Heurísticas Baseadas em Apostas para Problemas de Otimização Combinatória**. 2015. 127 f. Tese (Doutorado em Informática) - PPGI, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2015.

Diversos métodos de computação evolutiva e modelos de busca heurística já foram criados ao longo das últimas décadas com resultados bastante animadores, inclusive com aplicações em problemas complexos de otimização combinatória. Resultados teóricos conhecidos como teoremas "*No Free Lunch*", porém, comprovam que a diversidade de tais métodos é fundamental para que diferentes classes de problemas possam ser atacados. Este trabalho apresenta uma formalização de teoria de apostas e, com base na mesma, propõe um novo modelo de busca heurística a fim de atacar algumas classes de problemas, principalmente os de otimização combinatória. Resultados experimentais de aplicação do método são também incluídos e analisados.

Palavras-chave: Metaheurísticas, Teorias de apostas, Otimização combinatória.

ABSTRACT

Demasi, Pedro. **Heurísticas Baseadas em Apostas para Problemas de Otimização Combinatória**. 2015. 127 f. Tese (Doutorado em Informática) - PPGI, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2015.

Many evolutionary computation methods and heuristic models were created in the last few decades with good results, in particular on applications to complex combinatorial optimizations problems. Theoretical results, such as those known as "No Free Lunch" theorems, however, indicate that the diversity of such methods is essential in order to successfully approach different classes of problems. This work presents a formalization of betting theory and,based on it, proposes a new heuristic search model in order to approach different classes of problems, with combinatorial optimization problems particularly in mind. Experimental results of the methods application are included and analyzed.

Keywords: Metaheuristics, betting theory, combinatorial optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Modelos de busca heurística podem ser utilizados como métodos genéricos de solução para diversos problemas de otimização complexos [17, 42, 58, 49, 27]. Este trabalho propõe uma nova abordagem para tal a partir de uma proposta de formalização de teoria de apostas. Dadas as bases formais, propomos um novo método de otimização de maneira a verificar os efeitos previstos na teoria de forma mais clara. Tais resultados serão posteriormente verificados em aplicação nos estudos de casos.

1.1 Motivação

Apesar da dificuldade em resolver problemas complexos de otimização pelos métodos tradicionais, sua formulação em geral é muito bem definida, além de admitirem maneiras relativamente simples de verificação de soluções. Como exemplo, podemos citar problemas NP-completos para os quais não se conhecem soluções exatas em tempo polinomial mas cujas formulações são bem definidas e, além disso, têm soluções verificáveis em tempo polinomial [15]. A aplicação de métodos de busca heurística (e, mais especificamente, de computação evolutiva) nesses casos torna-se bastante convidativa. Desse ponto de vista, os problemas podem ser considerados "bem comportados", uma vez que são muito bem definidos, as soluções são facilmente verificáveis e os objetivos não mudam ao longo da aplicação do método de resolução. Em outras palavras, a solução procurada é *estática* com relação aos dados de entrada, sendo invariante com relação ao tempo.

A utilidade de se ter um método genérico o suficiente para que possa ser

aplicado em diferentes tipos de problemas complexos também é uma grande motivação para a criação de um novo modelo. A existência dos teoremas "*No Free Lunch*" [87] para problemas de busca heurística torna ainda mais interessante essa proposta, posto que há uma necessidade de diversidade de métodos para que diferentes classes de problemas possam ter bons resultados práticos.

1.2 Objetivos

Os objetivos principais deste trabalho são enunciar, estudar e verificar novos conceitos relativos a apostas, e aplicá-los em problemas de otimização inerentemente complexos. Tais conceitos são desenvolvidos especificamente com tais ambientes de aplicação em mente.

A analogia com o mundo real se faz presente, mesmo que em passagens ilustrativas de maneira a não nos perdermos demais em abstrações. Tal ponte é importante uma vez que a própria computação evolutiva tem raízes em analogias com o meio natural em que vivemos [37, 31].

Este trabalho também tem como objetivo propor uma unificação de definições e nomenclaturas a respeito de teoria de apostas de uma maneira concisa que possa ser aproveitada em outras aplicações teóricas e práticas.

Finalmente contamos com o objetivo de dispor, ao fim do trabalho, de um ferramental, teórico e prático, para aplicá-lo em tais sistemas complexos.

1.3 Contribuição

A contribuição deste trabalho reside em propor uma base teórica original que nos permita atacar problemas de otimização complexos. Com base nos aspectos teóricos revisados, propostos e desenvolvidos, um novo modelo de busca heurística é apresentado, com a metodologia de modelagem de problemas de otimização para sua aplicação também desenvolvida.

É proposta, também, uma unificação de uma teoria de apostas de maneira concisa que permite verificar determinadas características estratégicas. Baseando-se em tal teoria propomos um método original de busca heurística, como uma maneira de verificar certas características inerentes a aplicações nessa área.

Mais especificamente, podemos citar a unificação de uma teoria geral de apostas proposta no Capítulo 2, com base em diversos estudos e resultados avulsos, reunidos em torno de uma mesma notação, em particular na Seção 2.2, além do desenvolvimento de resultados apresentados nas Seção 2.2.4 (e subseções). Podemos citar também o método de busca para problemas de otimização desenvolvido e apresentado algoritmicamente no Capítulo 3.

Por ser um trabalho de vanguarda e com potencial inovador, existem diversas limitações naturais de tais características, e a dificuldade em comparação com trabalhos existentes devido a diferenças profundas com relação que já existe atualmente. Apesar disso, e tendo tudo isso em mente, os resultados e o potencial aqui exposto ficam como importante contribuição, e a possibilidade inovadora também enriquece as contribuições já citadas.

Este trabalho também gerou publicações em congressos nacionais [20], [21] bem como uma submissão a um periódico internacional [22].

1.4 Estado da Arte

Diversos modelos baseados em métodos de busca heurística já foram propostos e aplicados com resultados interessantes [63, 75, 27]. A diversidade de tais métodos, porém, é necessária para que mais problemas possam ter soluções satisfatórias encontradas e uma gama maior de instâncias dos mesmos tenham boas chances de serem "atacadas" com sucesso.

Tudo isso se deve ao efeito provado pelos chamados teoremas "*No Free Lunch*" para buscas [87], os quais afirmam, resumidamente, que todos os métodos estocásticos de busca tendem a ter, na média, o mesmo desempenho sob todas as possíveis funções (e todas as possíveis métricas). Isso significa, basicamente, que para diferentes classes de problemas, diferentes métodos de busca estocástica terão desempenhos alternados (melhores em uns, piores em outros). Dessa forma, a variedade de algoritmos nesse estilo é, também, importante para poder atacar de maneira satisfatória a maior quantidade de problemas possível.

O presente trabalho se coloca, portanto, como uma proposta de um caminho alternativo a ser trilhado para atacar problemas de otimização inerentemente complexos.

1.5 Organização

Este trabalho está organizado em quatro capítulos, dos quais esta introdução é o primeiro.

No Segundo Capítulo apresentaremos uma formalização de teoria de apos-

tas a partir de diversos conceitos, unificando a notação de maneira conveniente e apresentando alguns resultados.

No Terceiro Capítulo usaremos a teoria de apostas apresentada e, com base nela, proporemos um novo jogo de apostas estratégico teórico, o qual será a base para o método de busca heurística proposto a seguir, com a descrição do modelo e do algoritmo de que dele deriva.

No Quarto capítulo iremos mostrar aplicações diretas do método proposto em problemas de otimização combinatória (problema do caixeiro viajante assimétrico, problema da mochila binária multidimensional e problema de satisfatibilidade) apresentando os resultados obtidos bem como discussão dos mesmos. Também é apresentado um modelo híbrido, usando algoritmos genéticos para derivar a solução inicial alimentada ao método e, assim como nos casos anteriores, os resultados são verificados e discutidos.

Traremos, finalmente, no Quinto Capítulo, nossas conclusões e propostas de trabalhos futuros.

2 FUNDAMENTOS DE UMA TEORIA DE APOSTAS

Este capítulo define e propõe uma teoria de apostas para, em cima da mesma, basear um processo evolutivo que possa ser comparável ao algoritmo padrão de computação evolutiva.

2.1 Introdução

O ato de apostar sempre esteve presente ao longo da História da humanidade, sendo uma das atividades mais antigas do homem [71]. A ação de arriscar algo na crença de que determinado acontecimento será realizado é um processo até mesmo natural, e que fazemos constantemente mesmo que seja, muitas das vezes, de maneira retórica.

As apostas, apesar disso tudo, são oficialmente ilegais em várias partes do mundo e costumam ser mesmo vistas de maneira pejorativa, em geral associadas ao vício de jogar em si. Há, também, toda uma conotação criminosa que envolve o jogo, uma vez que historicamente tal atividade teve bastante ligações com associações criminosas [71].

Este trabalho, despido de qualquer preconceito, procura aproveitar as características do jogo de apostas como forma de inspirar um método que sustente um modelo de tomada de decisões estratégicas e suas aplicações em situações de competição. Tal característica pode ser aproveitada, por exemplo, para desenvolvimentos na área de computação evolutiva, entre outros aspectos que veremos ao longo deste

texto.

Embora estejamos partindo de um processo existente no mundo real (as apostas), usaremos isso apenas como inspiração para o desenvolvimento deste trabalho. Assim sendo, nem sempre as analogias entre o modelo proposto e o mundo real farão sentido, o que poderia levar um leitor mais atento a fazer afirmações como "isso não acontece no cotidiano". Por isso fazemos questão de salientar que toda analogia de jogo de apostas serve como um ponto de partida e não como um guia restrito.

A Seção 2.2 trata das definições básicas dos termos que iremos usar ao longo do capítulo, bem como apresenta as características fundamentais sobre as quais o modelo se sustenta, constituindo boa parte da base deste trabalho. A Seção 2.3 expande o modelo básico de apostas e apresenta o conceito fundamental de esquemas, analisando sua importância e suas implicações. A Seção 2.4 trata das aplicações do modelo proposto e discutido neste capítulo.

2.2 Definições

Nesta seção vamos definir os principais conceitos sobre os quais o modelo de apostas será baseado. Boa parte do texto aqui apresentado será meramente descritivo, mas quando necessário for, as devidas demonstrações serão apresentadas.

A necessidade em uniformizar dessa maneira notações e definições advém do fato de que boa parte do conhecimento já desenvolvido em teoria de apostas não tem uma unidade forte tal qual, por exemplo, teoria de jogos. Boa parte desses conceitos podem ser desenvolvidos a partir do conteúdo de estatística e probabilidade [69], mas a maneira de manipular tais conceitos (e definir outros) varia dependendo da abordagem, quer seja mais prática, voltada principalmente para a aplicação em

casas de apostas reais [86, 60], quer seja uma abordagem mais teórica, com intuito de estudar mais profundamente os modelos matemáticos que estão envolvidos no assunto [79, 89, 80] e entendê-los melhor. Dessa maneira introduziremos, a partir desta seção, uma notação própria unificada para esse conhecimento que ajude no entendimento global do assunto e permita manipular de forma consistente tais propriedades, além de provar explicitamente determinados resultados quando for conveniente ou necessário.

O mais intuitivo seria iniciar esta seção definindo o que são apostas ou, pelo menos, o que é um jogo de apostas. Tais definições, porém, dependem de outras anteriores e, por isso, iremos seguir uma linha de apresentação que, se não é a mais óbvia, pelo menos é coerente.

Iniciemos, portanto, definindo **evento (event)**, que é um determinado acontecimento sobre o qual apostas serão feitas. O evento possui um conjunto finito de escolhas sendo que uma delas, e apenas uma, irá acontecer. Já as **escolhas (picks)** são os possíveis resultados de um certo evento. Se o evento for uma partida de futebol entre dois times A e B, por exemplo, há três escolhas possíveis: vitória do time A, vitória do time B ou empate. Chamaremos de **resultado (outcome)** a escolha de um evento que de fato ocorreu, lembrando que são exclusivas (uma escolha, e apenas uma, é o resultado em cada evento). Chamaremos de **risco (stakes)** o valor que o apostador paga para realizar uma aposta. Em geral considera-se esse valor como dinheiro, mas ao longo deste trabalho iremos abstrair essa parte, considerando-o apenas como um determinado valor representado por um número real. É possível também usar o termo "aposta" semanticamente com o significado de risco, mas nesse caso há o perigo de haver ambiguidades.

A partir de tais termos, temos implícita a definição de **aposta (bet)** como sendo o ato de pagar uma soma definida (o risco) em uma escolha de um evento.

Caso ela ocorra, o *apostador* (definido com o agente que faz a aposta) recebe uma quantia (estritamente maior) paga de volta como prêmio por ter feito a escolha correta. Chamamos essa quantia recebida pelo apostador como **prêmio**.

É necessário que haja algum tipo de intermediário no processo de apostas, e é neste ponto que entra a figura da **banca (bookmaker, house)**, basicamente quem gerencia as apostas. Ela recebe o pagamento dos apostadores (os riscos) e paga aos vencedores. Cabe a observação que o termo comumente usado no cotidiano neste caso seria "casa de apostas", entretanto entendemos que o termo "banca" não só se encaixa melhor nas propostas deste trabalho como, também, é mais sucinto e forte, sendo, portanto, o escolhido.

Um outro termo importante é o **cacife (bankroll)**, a quantidade total de recursos que o apostador tem a sua disposição para realizar uma aposta. O termo "cacife" muitas vezes é utilizado para indicar uma quantia padrão com a qual todos os jogadores iniciam um jogo de apostas arbitrário, mas não será esse o caso aqui. O termo só será empregado no sentido do total disponível para o apostador, assim como também podemos considerar que a banca tenha um cacife. Normalmente, porém, iremos supor que o mesmo é infinito (ou, pelo menos, "suficientemente grande"), a não ser quando explicitamente indicado o contrário.

O valor pago pela banca aos apostadores vencedores depende da **cotação (odds)**, que é o prêmio para cada unidade de risco, caso a escolha seja o resultado do evento. Por exemplo, se o risco do apostador foi 5 (repare a ausência de referências monetárias) e a cotação for $2,7$ o apostador receberá de volta $13,5$ ($5 \times 2,7$) caso a escolha ocorra. Note-se que, em geral o termo *odds* é associado a "probabilidade" ou "chance" já que, como veremos mais à frente, está intimamente ligado à chance de uma escolha acontecer. Neste caso, porém, a melhor analogia está ligada ao conceito de ganho (ou *payoff*), uma vez que o valor recebido pelo apostador é o risco

multiplicado pela cotação que, desta maneira, representa uma espécie de "prêmio por unidade arriscada (ou apostada)". Outra questão importante é que existem diferentes tipos de representação para as cotações, porém a adotada neste trabalho será sempre a que normalmente é conhecida como "decimal" ou "europeia". As demais notações não nos serão úteis e, portanto, serão omitidas.

Chamamos de **balanço (balance)** a diferença entre o valor recebido como prêmio e o pago pela aposta (risco) no caso do apostador, e a diferença entre todos os valores pagos pelos apostadores e o pago em prêmios no caso da banca. No exemplo do parágrafo anterior, o balanço será $13,5 - 5 = 8,5$ caso a escolha feita ocorra e $0 - 5 = -5$ caso contrário. Quando o balanço é positivo chamamos de **lucro (profit)** e, quando negativo, de **prejuízo (loss)**.

A partir das cotações, definimos como **favorito (favorite)** a escolha cuja cotação é a menor entre todas do evento. Ou, de maneira equivalente, a que tem a maior probabilidade de acontecer. Analogamente, definimos como **azarão (underdog)** a escolha cuja cotação é a maior entre todas do evento. Ou, de maneira equivalente, a que tem a menor probabilidade de acontecer. Veremos, com detalhes, na Seção 2.2 o porquê de as probabilidades serem inversamente proporcionais às cotações.

Usamos o termo **quebrar (crash)** quando o agente perde todo o seu cacife, ou seja, chegar a zero, falir. Neste caso, o apostador não pode continuar mais apostando. De maneira similar, **quebrar a banca (bring down the house)** significa acabar com todo o cacife da banca, levando-a à falência. Isso pode ocorrer por diversos motivos, sendo um deles má administração por parte da própria (cotações mal feitas por exemplo). Pode acontecer, também, por conta de uma aposta muito volumosa em uma determinada escolha (possivelmente um azarão) acarretando uma soma grande demais a ser paga. Como normalmente consideramos o cacife da banca

muito grande, para supor que a banca vá quebrar também precisamos de uma soma suficientemente grande para tal (por exemplo, infinitos apostadores tendo ganhos positivos).

Para lidar com possíveis perdas e garantir lucro, existe a **margem (over-round) da banca**, que é a superestimação, por parte da banca, da probabilidade de uma escolha acontecer de forma a diminuir sua cotação. Dessa maneira, a banca tem uma margem de vantagem contra os apostadores. Pode ser visto como uma "taxa de administração". Voltaremos a essa questão com mais detalhes na Seção 2.2. Chamamos de **banca real** a banca com margem de vantagem positiva e de **banca ideal** a banca sem margem de vantagem (ou margem zero). O valor desta margem, no mundo real, em geral depende da quantidade de escolhas de um evento. Normalmente, num evento de duas escolhas (basquete, por exemplo), gira em torno de 5%, enquanto num de três (como o futebol) fica perto de 11%.

Outro dispositivo que a banca pode lançar mão de maneira a preservar seus lucros é o **ajuste de cotações (odds switching)**, que nada mais é do que a modificação dos valores das cotações. Isso ocorre com mais freqüência devido a um acontecimento inesperado (por exemplo, o principal jogador de um time de futebol se machuca num treinamento e não vai poder participar da partida) mas, como veremos, também pode ser resultado de outras implicações.

Existem diferentes formas de a banca receber apostas e pagar os prêmios aos apostadores. O que apresentamos aqui é conhecido como sistema de **cotações fixas (fixed odds)**. Note-se que o ajuste de cotações não significa que o sistema deixou de ter cotações fixas, uma vez que tal denominação se dá ao fato de a aposta ser feita sabendo-se exatamente qual a cotação que se está pagando (e, conseqüentemente, qual o possível prêmio). Ou seja, mesmo que as cotações mudem ao longo do tempo, as apostas já feitas não são alteradas. No sistema de **cotações variáveis**, faz-se

a aposta a uma dada cotação, mas a mesma pode mudar futuramente. Ou seja, a diferença para o caso fixo é que a aposta já feita é alterada pelo ajuste de cotações neste caso.

Um outro sistema de apostas muito comum é conhecido como **parimutuel**, do francês *pari* (aposta) e *mutuel* (mútua). Neste sistema, todos os apostadores fazem suas escolhas sem saber da cotação (que não existe de fato). Ao final das apostas, a banca retira uma porcentagem de todo o valor apostado a título de taxa de administração. Todo o resto é dividido entre os apostadores cujas escolhas tenham sido o resultado, sendo o prêmio pago a cada um diretamente proporcional ao valor apostado em relação aos demais vencedores. Este é o sistema comumente usado em loterias, por exemplo, e em alguns tipos de corridas de cavalos.

A não ser quando explicitamente dito o contrário, iremos sempre considerar o sistema de apostas com cotações fixas neste trabalho.

Precisamos, agora, definir o centro de toda a discussão que é, justamente, o **jogo de apostas**. Já sabemos o que significam os termos que fazem parte desse conceito, porém precisamos formalizar o que eles representam e as relações entre os mesmos.

A entidade principal do jogo de apostas é a banca, quem controla os valores recebidos e pagos. Do ponto de vista teórico, porém, podemos encará-la como um elemento implícito ao próprio sistema, uma vez que não se faz necessário defini-la como algo com papel ativo, apenas supondo-se que suas atividades são realizadas sem problemas.

Posto isso, podemos enxergar preliminarmente os elementos principais que fazem parte do jogo de apostas como divididos em duas categorias: apostadores e

evento.

No caso dos apostadores temos o próprio conjunto dos mesmos e o cacife associado a cada um. Já no caso do evento temos as escolhas e as cotações associadas às mesmas.

Podemos, também, ver o jogo de apostas como um acontecimento isolado (ou seja, um conjunto de apostadores com determinados cacifes fazendo apostas em um evento arbitrário) ou como sequências de apostas ao logo do tempo (o que implicaria a atualização dos cacifes entre cada processo de aposta). O primeiro, naturalmente, seria um caso especial do segundo.

Mesmo sem considerar uma sequência de apostas ao longo do tempo (tratando apenas eventos isolados) é possível complicar um pouco mais os elementos do jogo de apostas. É preciso, por exemplo, considerar o conjunto de riscos de cada jogador. Podemos considerar tal conjunto como único ou não: ou seja, cada apostador faz apenas uma aposta (que pode até mesmo ser zero, ou seja, não apostar) ou permitimos que cada apostador faça uma quantidade finita, porém variável, de apostas. Outro complicador seria considerar o ajuste de cotações, sendo necessário fazer o conjunto de cotações variável com relação ao tempo.

Como se não houvesse já complicações suficientes, ainda podemos considerar que um determinado apostador pode fazer várias apostas em diversos eventos. Também pode fazer mais de uma aposta no mesmo evento, possivelmente em escolhas diferentes, como veremos na Seção 2.2.6).

Embora todos esses complicadores sejam importantes, vamos inciar nosso estudo pela maneira mais simples possível para, à medida que formos progredindo, usarmos definições mais abrangentes quando nos for conveniente.

2.2.1 Jogo de Apostas

Podemos definir o jogo de apostas como uma tupla contendo um conjunto A de apostadores (definidos com mais detalhes na Seção 2.2.2), um conjunto E de eventos (definidos com mais detalhes 2.2.3), dois números reais não negativos r_{min} e r_{max} ($0 \leq r_{min} \leq r_{max}$) representando, respectivamente, os valores de apostas mínima e máxima permitidos, e um número real positivo g representando a granularidade das apostas. Ou seja:

$$G = \langle A, E, r_{min}, r_{max}, g \rangle$$

Os valores de r_{min} e r_{max} podem ser iguais (aposta fixa), podemos ter também $r_{min} = 0$ (não há obrigatoriedade de aposta), ou mesmo $r_{max} = +\infty$ (não há limite de valores de apostas).

Já a granularidade de apostas representa a unidade mínima em que pode ser aumentado o valor de aposta. Em outras palavras, temos que r (o risco) pode ser definido como:

$$r = r_{min} + i \times g$$
$$0 \leq i \leq \frac{r_{max} - r_{min}}{g}, i \in \mathbb{N}$$

Onde i é um número natural que, basicamente, representa quantas unidades de g o risco r está acima de r_{min}.

Em casas de aposta reais, geralmente temos $g = 0,01$, que é justamente a

quantidade de casas decimais usadas monetariamente. Apesar de nos afastarmos durante o trabalho do caso real, usaremos essa mesma convenção, uma vez que é a mais comum e conveniente. Por uma questão de coerência, iremos manter a mesma convenção em todos os valores decimais (usando, portanto, duas casas). Deve-se reparar, todavia, que tais convenções introduzem potenciais diferenças de arredondamento que podem causar resultados ligeiramente diferentes em algumas contas.

2.2.2 Apostadores

Iremos, inicialmente, encarar apostadores para eventos isoladamente. Ou seja, a definição aqui dada de apostador presume a aposta em um determinado evento apenas.

Definimos formalmente o **apostador** a como uma tripla contendo um número real não negativo c representando seu cacife, um conjunto finito (possivelmente vazio) de inteiros positivos S de escolhas (cada elemento deste conjunto podendo ser encarado como um índice para o conjunto de escolhas do evento) e um conjunto de números reais não negativos R de riscos, onde $R = \{r_1, r_2, ..., r_{|R|}\}$ e $S = \{s_1, s_2, ..., s_{|S|}\}$ e $|R| = |S|$. Ou seja:

$$a = \langle c, S, R \rangle$$
$$S = (s_i), s_i \in \mathbb{N}$$
$$R = (r_i), r_i \in \mathbb{R}, r_i \geq 0$$

Note-se que estamos prevendo a possibilidade de um apostador fazer várias apostas (ou mesmo nenhuma), inclusive na mesmo escolha, a riscos distintos. Como

não estamos, contudo, preocupados com o instante de tempo em que tais apostas foram realizadas não estamos, no momento, considerando a possibilidade de ajustes de cotações.

Uma simplificação dessa definição seria restringir $|S| \leq 1$, o que significa que cada apostador pode apenas apostar em uma escolha e apenas umas vez (ou simplesmente não apostar). Uma pequena alteração para tornar mais fácil a notação, neste caso, seria restringir $|S| = |R| = 1$.

Deriva do que foi apresentado nesta seção o conjunto de apostadores A, sendo caracterizado como:

$$A = \langle (c_i), (S_i), (R_i) \rangle$$

Ao considerarmos as escolhas e riscos de cada apostador do conjunto de apostadores, representamos como s_i^j a j-ésima escolha do apostador i e r_i^j o j-ésimo risco do apostador i.

2.2.3 Eventos

Definimos um **evento** E como um par contendo um conjunto S de escolhas e um conjunto ω de cotações, ou seja:

$$E = \langle S, \omega \rangle$$

Note-se que, por esta definição, não estamos considerando o ajuste de cota-

ções, pois seria necessário fazer o conjunto ω prever variação ao longo do tempo. Consideramos, também, que sempre temos $|S| = |\omega|$ e que a escolha S_i paga ω_i como cotação.

Ao abstrairmos os significados dos eventos e passarmos a considerar as escolhas apenas como enumeração, podemos simplesmente ignorar o conjunto S sendo o evento totalmente definido pelo conjunto ω de cotações:

$$E = \omega$$

Neste caso, naturalmente, estamos ignorando o fato de as cotações poderem se ajustar ao longo tempo. Poderíamos acrescentar uma nova dimensão de tempo ao conjunto de maneira que ω_i^t passe a representar a cotação paga pela escolha i no tempo t. Note-se que o tempo, neste caso, é totalmente abstrato, discreto e arbitrário, de maneira que a distância entre t e $t+1$ é apenas uma maneira de diferenciar um determinado instante de outro seguinte. Podemos até mesmo encarar t como um indexador de apostadores (ou seja o t-ésimo apostador a fazer a aposta). No caso de ausência de ajuste de cotações o índice temporal pode ser omitido sem problemas, uma vez que $\omega_i^t = \omega_i^{t+1}, \forall t$.

Vamos olhar, agora, com um pouco mais de detalhes as propriedades do evento. Dissemos, anteriormente, que as cotações estão intimamente ligadas à probabilidade de a escolha correspondente acontecer (inversamente proporcionais) e também ponderamos sobre a margem da banca superestimando tais probabilidades. Suponhamos que p_i seja a probabilidade estimada pela banca de o resultado do evento E ser a escolha i. Então temos que:

$$\sum_{i=1}^{|\omega|} p_i = 1.0 + \epsilon$$

Sendo que $\epsilon = 0$ se a banca é ideal e $\epsilon > 0$ se a banca é real (com ϵ, portanto, indicando o total da margem da banca). Não vamos considerar, por ora, a possibilidade de $\epsilon < 0$, entretanto ela será útil no momento apropriado.

Para facilitar a notação, supomos que ρ_k seja o total de risco de todos apostadores numa determinada escolha k. Ou seja:

$$\rho_k = \sum_{i=1}^{|A|} \sum_{j=1}^{|S_i|} r_i^j, \forall s_i^j = k$$

Suponhamos uma banca ideal. Suponhamos, também, que as probabilidades estimadas pela banca sejam dadas por um oráculo e, portanto, sejam perfeitas. Ora, nesse caso, o valor esperado para o balanço da banca é zero (nem lucro nem prejuízo), uma vez que os prêmios pagos devem estar relacionados à probabilidade do resultado e não há qualquer margem para a banca. Ou seja:

$$E_k = 0$$

O valor esperado para o balanço da banca para a escolha k vai ser o lucro obtido caso o resultado não seja k somado ao prejuízo caso contrário. O lucro é dado por ρ_k, ou seja, o total apostado. Já o prejuízo será $\rho_k - \omega_k \rho_k$, ou seja, o valor apostado (recebido) menos o prêmio (pago aos apostadores que acertaram). Lembrando que o prêmio, como visto na Seção 2.2, é definido como a cotação multiplicada pelo risco. Sendo assim, temos:

$$E_k = (1 - p_k)\rho_k + p_k(\rho_k - \omega_k\rho_k)$$

Logo (supondo que $\rho_k > 0$):

$$(1 - p_k)\rho_k + p_k(\rho_k - \omega_k\rho_k) = 0$$

$$\rho_k - p_k\rho_k + p_k\rho_k - p_k\omega_k\rho_k = 0$$

$$\rho_k = p_k\omega_k\rho_k$$

$$p_k = \frac{1}{\omega_k}$$

Ou seja, a cotação paga em caso de o resultado ser a escolha é inversamente proporcional à probabilidade de a escolha ser o resultado. Claro que isso tudo considerando um cenário ideal.

Considerando, agora, uma banca real ($\epsilon > 0$), podemos levar em conta a margem da banca ϵ como sendo "diluída" entre as probabilidades estimadas para cada escolha. Ou seja:

$$\sum_{i=1}^{|\omega|} \epsilon_i = \epsilon$$

Supondo que p_i^* seja a probabilidade estimada final (acrescida da margem), neste caso teremos que $p_i^* = p_i + \epsilon_i$. Ou seja, a probabilidade estimada é adicionada da margem.

Então para o caso de uma banca real, temos:

$$p_i + \epsilon_i = \frac{1}{\omega_i}$$

$$\omega_i = \frac{1}{p_i + \epsilon_i}$$

2.2.4 Banca

Conforme já foi mencionado anteriormente, não iremos definir explicitamente a banca, uma vez que sua existência se dará sempre de forma indireta ou implícita. Nesta seção iremos justamente estudar o comportamento implícito do sistema baseado em como a banca deve "agir".

Uma das principais razões da existência da banca é obter lucro o que, a primeira vista, pode parecer estranho. Uma banca com ganho esperado zero, porém, estaria fazendo apenas movimentação de recursos de um grupo de apostadores para outro. Uma banca com prejuízo, por outro lado, estaria fazendo apenas caridade. Ou, de uma maneira mais prática, estaria apenas perdendo recursos o que, no longo prazo, a levaria à falência.

Conforme visto anteriormente, idealmente as cotações são inversamente proporcionais à probabilidade de o resultado do evento ser a escolha em questão. A banca tem, a seu favor, a margem com a qual superestima essas probabilidades de maneira a ganhar em cima dos apostadores.

Somente isso, contudo, não é suficiente. Digamos que num determinado evento uma escolha tenha tido bastante apostas e o resultado do evento seja justamente ela. A banca pode ter um prejuízo muito sério nesses casos. Como ela poderia evitar tais eventualidades? Mais do que isso: seria possível a banca garantir lucro

sempre, independente do resultado do evento? Essa seria, portanto, a situação ideal para banca.

Uma maneira simples de resolver isso seria garantir que:

$$\sum_{i=1}^{|A|}\sum_{j=1}^{|R|} r_i^j > \omega_k \rho_k, \forall k \in S$$

Ou seja, que o total de risco dos apostadores é sempre maior que a possível soma a ser paga para qualquer escolha. Se definirmos q_k como sendo a proporção de todo o risco dos apostadores no evento investida na escolha k (naturalmente que $\sum q_k = 1$), ou seja:

$$q_k = \frac{\rho_k}{\sum_{i=1}^{|S|} \rho_i}$$

Substituindo na equação anterior,acabamos com:

$$q_k \omega_k < 1, \forall k \in S$$

Ou seja:

$$p_k + \epsilon_k > q_k, \forall k \in S$$

Em outras palavras, se a probabilidade estimada pela banca para todas as escolhas for estritamente maior que a proporção de riscos naquela mesma escolha,

a banca garante que terá lucro independente do resultado do evento. Chamamos isso de **banca ótima**. Embora pareça, *a priori*, um exercício de adivinhação, a margem que a banca tem para mexer em suas cotações torna tal objetivo bem mais realizável.

Observamos que o resultado acima é relevante pois o que o mesmo está dizendo é que **o que importa não é estimar corretamente as probabilidades de cada escolha mas, sim, a proporção de apostas que serão feitas nas mesmas**. Em outras palavras, a banca não precisa estimar corretamente o resultado do evento, apenas como os apostadores irão se dividir entre as escolhas. Claro que, em alguns casos, as duas visões podem estar muito relacionadas.

No caso da analogia com o mundo real, muitas vezes tal observação pode não ser tão importante, uma vez que certamente as cotações definidas pela banca guiam as escolhas dos apostadores, ou pelo menos têm uma grande influência, ou mesmo em muitos casos a distribuição de riscos tende a seguir a probabilidade das escolhas. Do ponto de vista teórico, porém, não deixa de ser um resultado interessante.

2.2.4.1 Banca Ótima com Lucro Constante

Vimos como a banca pode garantir, de maneira ideal, que sempre terá lucro independente de qual escolha seja o resultado do evento. Somente o que foi visto, porém, não garante que o lucro seja sempre o mesmo.

Como vimos na Seção 2.2, o balanço da banca será dado pela diferença entre a soma dos riscos dos apostadores e o valor pago de prêmio aos vencedores. Ou seja, supondo k a escolha vencedora, podemos representar o balanço dessa maneira:

$$\sum_{i=1}^{|S|} \rho_i - \rho_k \omega_k$$

No caso da banca ótima que vimos na Seção 2.2.4, temos que $p_k = q_k$ e $\omega_k = \frac{1}{q_k + \epsilon_k}$, logo o balanço pode ser representado como:

$$\sum_{i=1}^{|S|} \rho_i - \frac{\rho_k}{q_k + \epsilon_k}$$

Ora, se considerarmos que todos os valores ϵ são iguais, naturalmente que o valor final desse balanço irá depender de q_k que, dificilmente, será igual para todos. Dessa maneira, embora garanta que sempre vá ter lucro, a banca não tem um valor constante de lucro, variando de acordo com a escolha que é o resultado do evento. O ideal seria que esse balanço, além de positivo, fosse sempre igual a uma determinada porcentagem do total apostado no evento.

Para obter isso, de acordo com o balanço da banca, deveríamos ter o produto $\rho_k \omega_k$ sempre constante. Ou seja:

$$\rho_k \omega_k = (1 - \lambda) \sum_{i=1}^{|S|} \rho_i, \forall k$$

Ou seja, o produto é sempre igual a uma porcentagem $(1-\lambda)$ do total $\sum_{i=1}^{|S|} \rho_i$ conforme dito anteriormente, representando um lucro de λ para a banca. Temos como cotação, portanto:

$$\omega_k = \frac{(1-\lambda)\sum_{i=1}^{|S|}\rho_i}{\rho_k}$$

Pela definição de q dada na Seção 2.2.4 temos, finalmente:

$$\omega_k = (1-\lambda)\frac{1}{q_k}$$

Esse é o valor da cotação para qualquer escolha de um evento no caso de uma banca ótima com lucro constante λ.

2.2.4.2 Parimutuel e Banca Ótima com Lucro Constante

Na Seção 2.2 definimos o sistema de apostas parimutuel. Nesta seção, iremos apresentar mais um resultado que nos parece interessante: **do ponto de vista do prêmio pago pela banca ao apostador, o sistema parimutuel com taxa de administração λ é equivalente a uma banca ótima com lucro constante igual a λ.**

Conforme já foi visto, chamamos de ρ_k o total arriscado pelos apostadores numa escolha k. Sendo assim, o total apostado em todas as escolhas será $\sum_{i=1}^{|S|}\rho_i$. No sistema parimutuel, o total a ser pago para os apostadores deve ser descontado da taxa de administração, sendo, portanto:

$$(1-\lambda)\sum_{i=1}^{|S|}\rho_i$$

Sabemos que cada apostador receberá um valor proporcional ao seu risco em relação ao total apostado naquela escolha, ou seja $\frac{r_i}{\rho_k}$. Assim, o valor que um apostador i receberá de prêmio pela escolha k será:

$$\frac{r_i}{\rho_k}(1-\lambda)\sum_{i=1}^{|S|}\rho_i$$

Sabemos que:

$$q_k = \frac{\rho_k}{\sum_{i=1}^{|S|}\rho_i}$$

Logo, o valor pago será

$$r_i\frac{(1-\lambda)}{q_k}$$

Que é justamente o valor obtido para ω_k na Seção 2.2.4.1 multiplicado pelo risco r_i.

Existem, mesmo nesse caso, diferenças entre os sistemas, sendo a mais evidente o fato de o apostador não saber, *a priori*, o valor de seu prêmio em caso de vitória no sistema parimutuel. Mas conforme foi enunciado no início desta seção, do ponto de vista do prêmio pago, eles são equivalentes.

2.2.5 Critério de Kelly

Já foi visto como uma banca pode "se comportar" de maneira ótima. Veremos, agora, como um jogador pode, teoricamente, fazer suas apostas de maneira ótima.

Suponhamos, novamente, que haja um oráculo que seja capaz de prever as probabilidades das escolhas com precisão e um apostador, sozinho, tenha acesso a essas previsões. Digamos a probabilidade dada pelo oráculo seja representada por p^*, nesse caso o valor esperado do apostador numa determinada escolha será

$$E = p^* r\omega - r$$

Uma vez que com uma probabilidade p^* aquele evento irá ocorrer. Ora, para que a aposta valha a pena, naturalmente precisamos que $E > 0$, portanto (supondo $r > 0$):

$$p^* r\omega - r > 0$$

$$p^* r\omega > r$$

$$p^* \omega > 1$$

$$p^* > \frac{1}{\omega}$$

Nada mais natural uma vez que, como já foi visto, a cotação de um evento é inversamente proporcional à probabilidade de o mesmo acontecer. O que essa fórmula nos diz, porém, é que só vale a pena apostar numa escolha caso a cotação dada pela banca seja maior que a cotação implícita dada por p^*. Ou, em outras

palavras, caso a escolha tenha sido subestimada pela banca (cotação maior do que deveria ser).

É claro que não existe tal oráculo. O que esse resultado nos diz, inicialmente, é que caso tenhamos alguma crença de que a probabilidade real de a escolha ocorrer é maior que a dada pela cotação, devemos apostar nela. O contrário também é válido, uma vez que só vale a pena arriscar uma escolha caso a cotação seja mais alta do que a que acreditamos que deveria ser.

Mesmo não possuindo um oráculo, diversas técnicas podem ser empregadas de maneira a encontrar tais probabilidades, desde as computacionais (como redes neurais), até métodos estatísticos ou simplesmente pela intuição.

Uma vez tendo essa informação, a questão que nos resta é o quanto devemos arriscar. O físico americano John Kelly respondeu a essa pergunta com uma fórmula que hoje é conhecida como Critério de Kelly [65].

Resumidamente, o que Kelly estabelece é que, uma vez conhecidos p^* e ω, podemos calcular f, a fração do cacife que devemos apostar na escolha. Ou seja, $r = c \times f$. A fórmula estabelece que:

$$f = \frac{p^*\omega - 1}{\omega - 1}$$

Podemos ver claramente que se $p^* = \frac{1}{\omega}$ teremos $f = 0$ (ou seja, não temos margem, não vale a pena apostar). Caso $p^* < \frac{1}{\omega}$ teremos f negativo, o que não faz sentido. Ou seja, de fato o critério só se aplica de acordo com a restrição encontrada anteriormente.

O resultado da aplicação do critério de Kelly é maximizar o cacife no longo prazo [65]. Existem, porém, algumas desvantagens, como por exemplo uma chance $\frac{1}{n}$ de o cacife ser reduzido a $\frac{1}{n}$ de seu valor durante uma série de apostas (50% de cair pela metade, por exemplo). Outro problema é com relação a valores de apostas mínima e máxima, pois nem sempre é possível cumprir a fração f sugerida pela fórmula.

A principal preocupação, porém, reside na premissa de se conhecer p^*. Dado que não existe um oráculo, esse valor vai ser sempre uma estimativa e se esta estiver errada com frequência os resultados podem ser desastrosos. Podemos também encarar de maneira um pouco mais otimista, sob uma ótica evolutiva: numa população de agentes realizando constantemente apostas usando o critério de Kelly, aqueles cujas estimativas de probabilidade forem as mais precisas serão justamente os que terão os maiores cacifes a longo prazo, enquanto que os mais imprecisos certamente irão quebrar em algum ponto.

2.2.6 Dutching

O termo **dutching** se refere ao ato de fazer apostas em mais de uma escolha de um determinado evento de uma só vez. Por exemplo, num jogo de futebol podemos apostar no time A e no empate. De certa maneira, a aposta simples (em apenas uma escolha), pode ser vista como um caso particular de dutching. Diz-se que o termo deriva do nome de seu criador, Arthur Flegenheimer (conhecido como Dutch Schultz), que começou a empregá-lo em corridas de cavalos [79].

O dutching tem uma importância de reduzir, do ponto de vista do apostador, as escolhas em uma questão binária: apostar num conjunto de escolhas ou em sua negação. Mesmo que haja n escolhas em um determinado evento, ao se apostar

em k delas, estamos, ao mesmo tempo, apostando que as $n - k$ restantes não irão acontecer.

O problema seria verificar qual a cotação da escolha ao fazer o dutching. Por exemplo, suponhamos que um apostador quer fazer uma aposta em duas escolhas do mesmo evento cujas cotações ω_1 e ω_2 são, respectivamente, $3,25$ e $2,50$. Se ele apostar uma mesma quantia em cada uma os prêmios serão diferentes dependendo da escolha que seja o resultado. Se queremos encarar como uma escolha só, deveríamos saber o quanto apostar em cada uma de maneira ao prêmio ser igual, independente de qual escolha seja o resultado.

Suponhamos, então, que existe uma cotação ω^* equivalente à aposta (por dutching) nessas duas escolhas. Sabemos que apostar nas duas escolhas significa que, para vencer, precisamos que uma ou outra seja o resultado. Sabendo, também, que $p = \frac{1}{\omega}$, temos:

$$\frac{1}{\omega^*} = \frac{1}{\omega_1} + \frac{1}{\omega_2}$$

$$\frac{1}{\omega^*} = \frac{\omega_1 + \omega_2}{\omega_1 \omega_2}$$

$$\omega^* = \frac{\omega_1 \omega_2}{\omega_1 + \omega_2}$$

No exemplo dado, teríamos que $\omega = \frac{3,25 \times 2,50}{3,25+2,50} = 1,41$.

Já sabemos quanto vale ω^*, agora precisamos saber quanto deve ser o risco r_1 e r_2 em cada escolha caso queiramos um risco total de r^* no dutching (obviamente teremos $r^* = r_1 + r_2$). Bem, sabemos que o prêmio deve ser $\omega^* r^*$ e, portanto,

$$\omega^* r^* = \omega_1 r_1 = \omega_2 r_2$$

$$r_1 = \frac{\omega^* r^*}{\omega_1}$$

$$r_2 = \frac{\omega^* r^*}{\omega_2}$$

Suponhamos, no exemplo anterior, que queiramos $r^* = 10$, como calculamos $\omega^* = 1,41$, temos $\omega^* r^* = 14,1$. Então $r_1 = \frac{14,1}{3,25} = 4,34$ e $r_2 = \frac{14,1}{2,50} = 5,64$. E, de fato, teremos como prêmio $4,34 \times 3,25 = 14,11$ ou $5,64 \times 2,50 = 14,10$. As pequenas diferenças são causados pelo arredondamento, inclusive $r_1 + r_2 = 9,98$.

Fizemos os cálculos iniciais do dutching para apenas duas escolhas, mas é claro que podemos generalizá-lo para n escolhas. O que nos levaria à seguinte fórmula:

$$\frac{1}{\omega^*} = \frac{1}{\omega_1} + \frac{1}{\omega_2} + ... + \frac{1}{\omega_n}$$

$$\frac{1}{\omega^*} = \sum_{i=1}^{n} \frac{1}{\omega_i}$$

$$\omega^* = \frac{1}{\sum_{i=1}^{n} \frac{1}{\omega_i}}$$

Obviamente temos que $r^* = \sum_{i=1}^{n} r_i$ e que $\omega^* r^* = \omega_i r_i = \omega_j r_j$, $\forall i, j$. Sendo assim, temos:

$$r_i = \frac{\omega^* r^*}{\omega_i}, \forall i$$

2.3 Esquemas

Um **esquema (betting system)** é alguma maneira sistemática de realizar as apostas com o objetivo de garantir lucro a longo prazo, ou seja, num número considerável de apostas (tendendo ao infinito).

De uma maneira mais formal, podemos definir um esquema como um algoritmo $\mathcal{A}$ cuja saída é uma dupla contendo um conjunto de escolhas (índices de escolhas, na verdade) e um conjunto de riscos. Ou seja, o esquema define em que escolhas apostar e qual o risco das apostas. As entradas do esquema podem ser as mais variadas possíveis.

Todos os dias nascem novos esquemas que prometem ganhos certos, e iremos investigar este assunto com mais detalhes nesta seção.

2.3.1 Martingale

O esquema Martingale é, provavelmente, o mais famoso, intuitivo e mais usado de todos [69]. Diz-se que este esquema tem origens no século XVIII [69] e seu funcionamento é bem simples: sempre que se perde uma aposta, a próxima deve ter seu risco aumentado de maneira a cobrir as perdas anteriores em caso de vitória; repete-se o processo até eventualmente ganhar a aposta, caso em que na aposta seguinte volta-se a um risco inicial.

Por exemplo, suponhamos um jogo simples de cara ou coroa (sem margem de lucro para a banca). O apostador arrisca 1 para receber 2 em caso de vitória. Digamos que ele vença, assim guarda seus ganhos e continua apostando 1. Em caso de derrota, na próxima rodada ele deve apostar 2, pois se ganhar receberá 4,

compensando os $1 + 2 = 3$ apostados. Caso volte a perder, na rodada seguinte deve apostar 4 para que o possível ganho de 8 compense os $1 + 2 + 4 = 7$ apostados. Assim sucessivamente até vencer a aposta, voltando, na rodada seguinte a apostar apenas 1.

A idéia principal do esquema é que, em cada "grupo" de apostas, tem-se lucro. No caso do exemplo anterior, sempre que vencemos uma aposta temos um lucro de 1.

Em teoria, parece que o esquema garante lucro positivo no longo prazo, porém ele depende de um fator fundamental: ter um cacife infinito (ou suficientemente grande). No exemplo das moedas, se tivermos um cacife de 100 e perdermos 7 vezes seguidas estaremos quebrados. Outro obstáculo seria o valor de aposta máximo permitido.

Ainda no caso da moeda, nossa chance de ganhar é de $0,5$ e a cotação é de $2,0$. A cada x apostas em que a última é vencedora, temos um lucro de 1. Ora, como a aposta na última rodada é de 2^x é fácil ver que o valor arriscado até a vitória foi de $2^x - 1$. Digamos que o jogador possa arcar com n perdas seguidas (ou seja, que $c \leq 2^n - 1$). Definimos como p_i a chance de o jogador ganhar na i-ésima rodada (ou seja, tendo $i - 1$ derrotas antes de vencer). Logo, $p_i = (0,5)^i = \frac{1}{2^i}$. Definimos também b_i como o valor apostado na rodada i (naturalmente temos que $b_i = 2b_{i-1}$ e $b_1 = 1$). Temos, portanto, que o valor esperado do balanço do jogador é:

$$E = p_1(2b_1 - b1) + p_2(2b_2 - b_1 - b_2) + ... + p_n(2b_n - b_1 - ... - b_n) -$$

$$(1 - p_1 - p_2 - ... - p_n)(2^n - 1)$$

$$E = p_1 b_1 + p_2(b_2 - b_1) + ... + p_n(b_n - b_1 - ... - b_{n-1}) -$$

$$(1 - \frac{2^n - 1}{2^n})(2^n - 1)$$

$$E = \frac{2^0}{2^1} + \frac{2^1 - 2^1 + 1}{2^2} + ... + \frac{2^{n-1} - 2^{n-1} + 1}{2^n}2^{n-1} - \frac{1}{2^n}(2^n - 1)$$

$$E = \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^n} - \frac{2^n - 1}{2^n}$$

$$E = \frac{2^n - 1}{2^n} - \frac{2^n - 1}{2^n}$$

$$E = 0$$

Ou seja, o valor esperado é zero. Isso significa que num jogo de apostas com banca ideal, cotação $2,00$ e probabilidade $0,50$ e que aceita um valor de aposta máximo infinito, não há expectativa de lucro. Parece óbvio neste momento que caso a cotação fosse um pouco favorável à banca (no caso de uma banca real), o valor esperado seria menor que zero e, portanto, resultaria em prejuízo.

Pode-se provar, de maneira geral, que nenhum esquema progressivo (ou seja, em que o valor dos riscos aumenta com derrotas sucessivas) dá lucro como valor esperado [86].

2.3.2 Esquemas Vencedores

Dizemos que um esquema é vencedor se seu valor esperado for positivo quando utilizado por um apostador independente da banca e de suas cotações. Um esquema vencedor é universal se ele puder ser usado por qualquer apostador (possivelmente uma quantidade infinita) e, ainda assim, garantir valor esperado positivo. Pode parecer estranho e até contraditório assumirmos a possibilidade de haver um esquema vencedor universal pois, afinal, se todos ganham, quem perde? Nesse caso, a banca teria como balanço um prejuízo infinito (supondo uma quantidade infinita de apostadores) e, portanto, quebraria.

Vamos supor, por um momento, que haja um esquema $\mathcal{A}^*$ oráculo, ou seja, capaz de calcular as probabilidades de cada escolha com perfeição. Nesse caso, usado por apenas um apostador, ele provavelmente seria vencedor. A não ser que as cotações seguissem essa mesma proporção (caso em que o esquema indicaria aposta nula). Ou seja, no pior caso ele teria esperança zero. Entretanto, ao ser usado por um número suficientemente grande de apostadores, acabaria levando à banca a ajustar as cotações de maneira que, gradativamente, q tenderia a p.

Essas especulações nos levam a duas questões fundamentais: existiriam esquemas vencedores universais? Ou, pelo menos, existiriam esquemas vencedores?

Em primeiro lugar, precisamos definir com precisão sobre qual tipo de informação estaremos lidando. Por exemplo, se considerarmos um esquema determinístico (em que em um determinado evento sempre dá o mesmo resultado para o mesmo conjunto de cotações) é quase imediato que ele não pode ser vencedor universal (pelo mesmo motivo do esquema oráculo já visto).

Vamos considerar, portanto, apenas esquemas que não façam uso de conluio, ou seja, colaboração entre diferentes apostadores. Esse tipo de esquema não pode, por exemplo, favorecer um certo jogador em detrimento de outro em um determinado evento para compensar num evento futuro. Vale notar que isso **não** significa que o esquema não possa ter memória (de eventos e resultados passados), apenas que ele é incapaz de distinguir apostadores e, portanto, não tem como funcionar sob colaboração.

Seja r_i o risco de cada aposta i feita em cada evento com cotação ω_i. O valor esperado do balanço do apostador é dado por:

$$E = \sum_{i=0}^{\infty} (p_i \omega_i r_i - r_i)$$

O que logo nos salta aos olhos é o fato de que se $p_i \omega_i < 1$ então o i-ésimo termo será negativo. Se essa característica se repetir sempre, teremos que E será infinitamente negativo. Sabemos que graças à margem da banca, isso pode ser feito. Apesar de simples, esse resultado é bastante interessante pois nos garante que **não existe esquema vencedor que independa da banca**. Como já vimos, porém, do ponto de vista da banca é mais vantajoso prever a proporção das apostas nas escolhas do que as probabilidades em si. Embora isso dê uma "esperança" ao uso de esquemas, não anula o fato que acabamos de salientar. Primeiro, porque o sucesso do esquema continua intimamente ligado a cotações favoráveis. E, segundo, porque nada garante, de antemão, que as proporções de apostas feitas nas escolhas não estejam, no fundo, intimamente ligadas às probabilidades em si.

Em outras palavras, por mais irônico que possa parecer, o sucesso de um esquema depende, essencialmente, da banca.

2.3.3 Arbitragem

A **arbitragem (arbitrage)** é uma prática comum em economia e finanças que consiste em aproveitar as diferenças de preço entre mercados diferentes de maneira a conseguir lucro simplesmente comprando e vendendo. No caso de câmbio, por exemplo, inicia-se com uma determinada quantia de uma moeda e vai-se fazendo compras sucessivas de moedas em mercados diferentes até voltar à moeda de origem com um valor maior que o inicial [76].

No caso de um jogo de apostas, a arbitragem seria a ação de apostar numa

determinada escolha e em sua negação (usando possivelmente dutching nos dois casos) de maneira que, independente do resultado, sempre se tenha lucro. A princípio pode parecer que apenas uma banca muito mal feita permitiria tal operação, mas acontece que com o ajuste de cotações e mesmo usando bancas diferentes é possível conseguir esse objetivo.

Por exemplo, digamos que num determinado momento para um evento com duas escolhas, temos cotações de $1,25$ e $3,00$. O apostador coloca 10 de risco na segunda escolha. Seu balanço será -10 caso o resultado seja a primeira e 20 caso seja a segunda. Digamos que após um tempo, por conta de ajuste de cotações, elas passem a $1,55$ e $2,20$. Ora, se o apostador colocar agora 19 na primeira escolha, seu balanço será, respectivamente, $10,45$ e -19. Se somarmos os dois balanços, teremos $0,45$ e 1, o que significa que independente da escolha que for o resultado do evento, o apostador garantiu ter lucro. Podemos ainda melhorar a aposta, pois dependendo do resultado o lucro será maior ou menor e o ideal seria que fosse sempre o mesmo. Considerando r o risco a ser feito na segunda aposta, basta fazermos com que o balanço da primeira escolha seja igual ao da segunda, somando-se as duas apostas, ou seja:

$$1,55r - r - 10 = 20 - r$$
$$1,55r = 30$$
$$r = 19,35$$

Nesse caso, teríamos o balanço $10,64$ e $-19,35$ na segunda aposta e balanço somado de $0,64$ e $0,65$ (novamente, as diferenças acontecem por questão de arredondamento).

Mas quais seriam as condições necessárias para haver a possibilidade de uma

arbitragem? Vamos considerar sem perda de generalidade, nos cálculos a seguir, que o evento tem apenas duas escolhas.

Digamos que as duas escolhas sejam a e b e que r_1^a seja o risco na primeira escolha no primeiro momento e r_2^b o risco na segunda escolha num segundo momento. Chamamos de ω_1^a e ω_1^b as cotações no primeiro momento das escolhas a e b e ω_2^a e ω_2^b as cotações das mesmas no segundo momento. Sabemos que o balanço total tem de ser positivo (e igual independente da escolha resultado). Os balanços, somados, em cada escolha são respectivamente:

$$r_1^a\omega_1^a - r_1^a - r_2^b$$

$$r_2^b\omega_2^b - r_2^b - r_1^a$$

Igualando os dois, temos:

$$r_2^b = \frac{r_1^a\omega_1^a}{\omega_2^b}$$

Este resultado confere com a conta que fizemos no exemplo anterior. Ainda resta, porém, a questão sobre quando é possível realizar a arbitragem, já que, a princípio, apenas calculamos o risco que devemos investir no segundo momento para garantir um lucro uniforme. Isso pode ser respondido levando-se em conta que o balanço deve ser sempre positivo, pois nossa conta apenas garante que eles sejam iguais. Ou seja:

$$r_1^a\omega_1^a - r_1^a - r_2^b > 0$$

$$r_1^a(\omega_1^a - 1) - r_2^b > 0$$

$$r_1^a(\omega_1^a - 1) > r_2^b$$

$$r_1^a(\omega_1^a - 1) > \frac{r_1^a \omega_1^a}{\omega_2^b}$$

$$\omega_1^a - 1 > \frac{\omega_1^a}{\omega_2^b}$$

$$\omega_2^b > \frac{\omega_1^a}{\omega_1^a - 1}$$

No nosso exemplo anterior, portanto, a condição para a possibilidade de arbitragem seria $\omega_2^b > \frac{3}{2}$, o que significa $\omega_2^b > 1,50$, o que realmente aconteceu.

Apesar de ser uma operação que vise essencialmente ao lucro, a arbitragem também pode ser usada em situações em que o prejuízo torna-se certo. Embora pareça incoerente essa visão, em alguns casos pode ser um recurso necessário. Digamos, por exemplo, que no caso anterior o apostador tenha investido uma quantidade considerável de seu cacife na escolha com cotação $3,00$ já esperando uma futura operação de arbitragem. Suponhamos que ao invés de as cotações serem ajustadas daquela maneira tenham sido para $1,40$ e $2,50$. Conforme já vimos, a cotação de $1,40$ não é suficiente para permitir a arbitragem. O apostador, agora, pode não fazer mais nada e apenas torcer para que a escolha seja o resultado. Se não for, porém, boa parte de seu cacife terá ido pelo ralo. É um risco que, muitas vezes, não se deseja correr.

Conforme vimos, se ele fizer uma aposta de $r = \frac{10 \times 3}{1,40} = 21,43$ ele pode uniformizar seu balanço, que no caso será de $-10 + 8,57 = -1,43$ e $20 - 21,43 = -1,43$. Ou seja, ele pode escolher entre ter um prejuízo certo de $1,43$ ou manter sua aposta inicial, lucrando 20 em caso de sucesso ou perdendo 10 em caso de fracasso.

2.3.3.1 Arbitragem em Múltiplas Bancas

Uma maneira teoricamente simples de se fazer arbitragem é aproveitar cotações de diferentes bancas. Já vimos quais as condições para ser possível realizar arbitragem e obter lucro, então seria suficiente ficar verificando periodicamente as cotações de diferentes bancas e realizar as apostas assim que as condições fossem favoráveis.

Embora em teoria pareça simples, dificilmente tal operação pode ser realizada no mundo real (no caso das casas de apostas), uma vez que as casas de apostas periodicamente também verificam tais condições de maneira a evitar que isso aconteça, como se fosse uma espécie de cartelização.

Um exemplo simples de como isso pode ser feito, porém, é aproveitar o estilo de aposta leiga que se faz no dia-a-dia com os colegas. Essas apostas, em geral, são feitas na cotação de um para um, ou mais especificamente, com $\omega = 2,00$. Cada um dos apostadores coloca um valor x e o vencedor leva o total de $2x$. Pode-se, naturalmente, generalizar para n jogadores, com cotação $\omega = n$.

Sabendo de tal fato, basta encontrar um determinado evento que permita arbitragem. Conforme foi visto, basta que haja uma cotação de escolha $\omega > 2,00$ (lembrando para eventos com mais de duas escolhas, podemos fazer uso de dutching para chegarmos a tal resultado).

Então, num exemplo simples, suponhamos que num determinado evento, o time adversário daquele para o qual o apostador torce esteja cotado a $2,20$. Assim, ele aposta com seu colega 100 (em um sistema monetário arbitrátio qualquer) que o time para o qual torce vai vencer. Conforme visto na Seção 2.3.3 basta ele apostar, agora, $90,90$ contra seu time na casa de apostas e terá um balanço garantido de

$100 - 90,90 = 9,10$ caso seu time vença ou $-100 + 109,08 = 9,08$ caso perca.

2.4 Aplicações

Nesta seção veremos algumas aplicações da teoria de apostas em problemas reais que, a princípio, não têm necessariamente uma ligação explícita e clara com o ato de apostar diretamente.

Veremos, primeiro, uma analogia com nosso cotidiano para verificar como a quantificação que fazemos de tudo às vezes é tão sutil e implícita que não percebemos a não ser que olhemos com certo cuidado. Além disso, muitas vezes assumimos algum tipo de risco, no sentido das apostas, em certas escolhas de eventos que não é claro sem que tenhamos algum tipo de investigação que nos explicite tal fato.

2.4.1 Analogia do Seguro

Uma das maneiras mais simples de verificar a aplicação na vida real das apostas é usando a analogia do seguro. Consideremos, de uma maneira simples, que um seguro seja constituído de um pagamento de uma taxa (podendo ser em parcela única ou várias parcelas) de maneira a receber um determinado valor caso um fato indesejado aconteça (roubo de carro, incêndio de casa, morte etc).

Mas o que isso tem a ver com as apostas? Antes de mais nada, podemos verificar, do ponto de vista do apostador, que uma apólice de seguro nada mais é do que uma aposta na tragédia. Sim, pois o apostador investe uma quantia a uma cotação alta (já que em geral são eventos bastante raros) e caso o indesejado aconteça, ele recebe aquele prêmio.

Usando como exemplo o mundo das apostas esportivas, seria o equivalente ao apostador realizar uma aposta contra o seu próprio time, o que é justamente o contrário do que o senso comum nos leva a crer. Em geral, apostar no próprio time seria uma garantia de confiança, ou mesmo uma maneira de salientar a força do mesmo.

Acontece, porém, que para o fã de esportes, uma derrota de seu time pode significar uma perda (emocional) muito grande, que aumenta de acordo com a importância da partida (uma final, por exemplo).

Digamos que haja uma final de campeonato entre os times A e B, e que as cotações sejam $2,20$ e $1,80$. Um determinado apostador torce para o time A e, portanto, apostará no time B uma quantia r. Caso seu time vença, ele terá perdido o dinheiro, mas estará tão feliz que isso não lhe fará diferença. Do contrário, caso perca, ele terá como consolação o valor $1,80 \times r$ de retorno, feito como um seguro para o caso de derrota.

Isso nos leva a questão de quanto deveria ser o valor r a ser apostado? Como quantificar o prêmio suficiente para consolar o torcedor?

Com relação à primeira questão podemos interpretá-la como um valor que o apostador estaria disposto a pagar para ver seu time campeão. E a segunda é o valor que ele gostaria de ganhar de maneira a não ligar para o time ter perdido o campeonato. É óbvio que nem sempre tais valores serão compatíveis.

No exemplo dado acima, digamos que o apostador defina $r = 100$, porém o prêmio de consolo deveria ser 1000. Naturalmente, caso aposte 100 no time adversário, seu prêmio será de 180, muito longe do que ele pretendia. Por outro lado, para ter o prêmio de 1000 em caso de derrota, ele deveria apostar $555,56$, o que está bem

acima do máximo que ele pretende gastar.

Se analisarmos esses números por um instante, iremos perceber que, no fundo, a paixão do apostador foi responsável por uma aposta implícita no time A de seu coração. Se ele quantificou como 1000 o prêmio que deseja receber para compensar a derrota, significa que, implicitamente, foi realizada uma aposta no time A com risco de 1000. Já que o máximo que ele admite gastar para equilibrar a situação é 100, então significa que o balanço implícito da aposta é de 1100, o que nos leva à conclusão de que a aposta implícita foi feita com cotação de $1,10$. Como já foi visto na Seção 2.3.3, a cotação do time a deveria ser de 11 para permitir o lucro certo, o que está bem longe de acontecer. Por outro lado, ele pode realizar uma operação de maneira a padronizar seu prejuízo.

Conforme calculado na Seção 2.3.3, ele deveria apostar $\frac{1000\times1,10}{1,80} = 611,11$ no time B, o que significaria ter um balanço de $100 - 611,11 = -511,11$ caso o time A vença e $-1000 + 488,88 = 511,11$ caso o time B vença. Claro que o valor a ser apostado $(611,11)$ é bem maior que risco máximo inicialmente estipulado pelo apostador (100). Neste caso, o balanço do jogador seria $100-100 = 0$ caso o time A vença e $-1000+80 = -920$ caso o time B vença, ou seja, um prejuízo muito grande no segundo caso.

Claro que a análise torna-se bastante imprecisa uma vez que estamos aproveitando determinadas características para tentar quantificar sentimentos (neste exemplo, a paixão por um time de futebol). Se pensarmos com um pouco mais de calma, contudo, veremos que isso não é tão incomum no nosso dia-a-dia, uma vez que na nossa sociedade moderna a quantificação do abstrato é comumente feita através da representação monetária. Nossa força de trabalho é quantificada, nossa alimentação é quantificada, até mesmo nossos desejos são quantificados.

O que a teoria do seguro aqui apresentada demonstra é que, na verdade, usando a analogia cotidiana de uma bolsa de apostas, podemos verificar a quantificação implicíta que temos em todas as nossas ações, algumas até mesmo involuntárias. A presença do modelo de apostas na nossa vida é, portanto, muito mais forte do que podemos notar a uma primeira vista.

2.4.2 Decisão, Evolução e Competição

O modelo da teoria de apostas apresenta um campo para evolução interessante, conforme já foi superficialmente citado na Seção 2.2.5. Sob a ótica evolutiva, os indivíduos mais aptos seriam aqueles com maior capacidade de estimar com precisão as verdadeiras probabilidades de suas escolhas e, portanto, fariam as melhores apostas, maximizando seu cacife a longo prazo (supondo o uso do critério de Kelly). Aqueles com menor capacidade iriam quebrar mais cedo ou mais tarde, sendo incapazes de continuar na competição. Esse ponto de vista nos sugere que o cacife, os riscos feitos e os prêmios recebidos seriam, implicitamente, responsáveis pela aptidão (**fitness**) dos indivíduos.

A grande dificuldade, nesses casos, seria modelar o problema de acordo com a teoria de jogo de apostas, com eventos, escolhas, cacifes e riscos. Outra preocupação seria definir a reprodução em si desses indivíduos, ou seja, como novos indivíduos entrariam em cena. Claro que nada impede, contudo, que este paradigma seja usado em conjunto com computação evolutiva, aproveitando as vantagens de cada um.

Com estas preocupações em mente, precisamos delinear uma base para que possamos aplicar os conceitos desenvolvidos em um método evolutivo.

3 MODELO

Neste capítulo veremos como utilizar a teoria exposta no capítulo anterior de maneira a realizar uma busca heurística, dando origem a um método de otimização baseado em apostas.

3.1 Metaheurísticas

Por metaheurística entendemos, geralmente, um modelo ou método criado para encontrar ou selecionar heurísticas que sejam capazes de retornar boas soluções para determinados tipos de problema. Além disso, as metaheurísticas têm a capacidade de conseguirem escapar de ficarem presas em locais cujas soluções podem não ser boas no geral (ou seja, em ótimos locais), ao contrário das heurísticas.

Naturalmente que o conceito de "boas soluções" é um pouco abstrato e isso pode variar bastante dependendo do problema e das expectativas com relação ao mesmo. Por esse exato motivo, metaheurísticas são boas candidatas para aplicações em problemas considerados "difíceis", ou seja, para os quais não se conhece solução viável próximo do ótimo global cujo tempo de execução esteja dentro de um limite razoável [9]. Isso não impede, naturalmente, que soluções viáveis possam ser facilmente construídas (e para muitos problemas isso é verdadeiro), porém uma solução de qualidade (ou seja, muito perto do melhor valor possível) torna-se uma tarefa proibitiva.

Pela própria natureza de tais métodos (e dos problemas aos quais eles são aplicados) é normal esperar que as soluções encontradas não sejam necessariamente

os ótimos globais mas, sim, as melhores possíveis dentro de certas restrições (normalmente tempo de execução, mas possivelmente, também, espaço de armazenamento, quantidade de avaliações de funções etc.).

Em geral, os modelos desenvolvidos costumam se basear em analogias e metáforas do mundo real, como algoritmos genéticos [37], colônias de formigas [24], [23], *simulated annealing* [47] etc. O método aqui apresentado não foge a essa regra, já que nos baseamos nos conceitos de apostas desenvolvidos e explorados no capítulo anterior.

Além disso, também é comum que tais modelos tenham um componente estocástico importante, principalmente pelo fato de sua aplicação ser normalmente voltada a problemas de otimização [75], sendo mais comumente aplicados a problemas de otimização determinísticos com os dados de entrada previamente conhecidos. Entre outras vantagens, isso pode permitir, por exemplo, fugir de ótimos locais. Outro ponto comum é uma certa quantidade de parâmetros e variáveis que devem ser definidos ao aplicar tais abordagens e, normalmente, a atribuição de tais valores segue normas empíricas (a prática de tentativa e erro é bastante comum) para que haja um ajuste adequado (em alguns casos pode haver razões analíticas para tal, porém). Há, também, métodos baseados em memória que não exigem tentativa e erro e costumam ter desempenho reconhecidamente melhor.

De uma maneira geral, modelos metaheurísticos constituem uma área bastante prolífica em pesquisas e avanços em aplicações a problemas de otimização combinatória [7]. Há vários resultados resultados obtidos por vários modelos e em diversos tipos de aplicação de altíssimo nível e bastante inovadores, mas existem também problemas com relação à explosão de métodos e às metáforas utilizadas, algumas vezes com resultados não tão espetaculares [77]. Há outros métodos, contudo, que não seguem a tendência de metáforas específicas, como GRASP (*Greedy*

Randomized Adaptive Search Procedure) [29] e ILS (*Iterated Local Search*) [53].

Tendo tudo isso em mente, apresentamos neste capítulo um modelo heurístico baseado nos conceitos teóricos já desenvolvidos até aqui neste trabalho. Iremos, entretanto, explorar mais detalhes de metaheurísticas no Capítulo 4, e também iremos analisar e comparar aplicações das mesmas aos problemas de que trataremos no mesmo capítulo.

3.2 Objetivos

O objetivo principal da aplicação do método é achar o máximo (ou mínimo, conforme o problema) de uma determinada função qualquer $f : \mathcal{A} \to \mathcal{B}$, sendo $\mathcal{A}, \mathcal{B} \subset \mathbb{N}$. Supõe-se que o domínio da função $\mathcal{A}$ é grande o suficiente para que seja proibitivo fazer uma busca exaustiva. Em outras palavras, precisamos varrer o conjunto $\mathcal{A}$ de maneira a encontrar $x^* \in \mathcal{A}$ tal que $f(x^*) \geq f(x)$ (ou $f(x^*) \leq f(x)$), $\forall x \in \mathcal{A}$. Estamos supondo que f pode ser avaliada em tempo polinomial.

3.3 Elementos

Algo que precisamos definir, antes de mais nada, é como representar cada solução candidata. Como $x \in \mathcal{A}$ é um número natural, cada solução candidata pode ser representada como uma cadeia de N bits, o que significa que o tamanho do espaço de busca para nossa função é 2^N. Isso não significa, naturalmente, que não haja outras maneiras de representação como, por exemplo, um vetor de inteiros que poderiam representar uma permutação de nós de um grafo. De qualquer maneira, usaremos N para representar o tamanho de nossa solução candidata, o que significa o quantidade de elementos do conjunto.

O próximo conceito importante é a *máscara* M, que é um conjunto de tamanho $|M|$. Cada passo de nosso método irá lidar com a possibilidade de fazer mudanças para a melhor solução conhecida aplicando essa máscara. Por causa disso, ela pode ter a mesma forma de representação que a solução candidata mas também pode ser diferente, como uma lista de índices únicos, por exemplo.

Também haverá m aplicações diferentes da máscara, que representa todos os possíveis resultados de um evento (ou seja, no que os jogadores irão apostar). Sendo assim, se pensarmos que cada elemento em M pode ser ou não aplicado para a melhor solução atual x^*, isso significa que $m = 2^{|M|}$. Pode também ser o caso que apliquemos cada elemento de M a x^*, então nesse caso teríamos $m = |M|$.

De maneira geral, podemos pensar na máscara como uma transformação $\tau_i : \mathcal{A} \to \mathcal{A}, 0 \leq i < m$, na qual cada τ_i é definido baseado em M, e aplicamos $\tau_i(x^*), 0 \leq i < m$ para criar novas soluções candidatas que representam os possíveis resultados de um evento.

Por exemplo, digamos que temos uma cadeia de bits representando cada solução candidata, que M é um conjunto de índices da cadeia (ou seja $0 \leq M_i < N$), com $m = 2^{|M|}$ e que cada $\tau_i(x^*)$ é definido como inverter o bit M_j de x^* se, e somente se, o M_j-ésimo bit de i estiver ativado. Nesse caso, suponha que $N = 200$ e $M = \{57, 123, 190\}$, então para τ_0 não precisaríamos inverter nenhum bit, para τ_2 precisaríamos inverter o bit 123 e para τ_5 precisaríamos inverter os bits 57 e 190.

Cada uma das m aplicações da máscara será um resultado, então cada resultado é definido por τ_i. O resultado vencedor τ_j será aquele cujo $\tau_j(x^*) > \tau_i(x^*), 0 \leq i < m, i \neq j$.

Vale ressaltar que as máscaras e as transformações (aplicações das máscaras)

são responsáveis por definir as escolhas que os jogadores terão para apostar. Como as mesmas são aplicadas à melhor solução conhecida (ou mesmo a alguma solução conhecida de maneira mais genérica), elas são responsáveis por dar a "direção", por assim dizer, da busca. Sendo assim, também é possível pensar em uma variação das transformações de maneira a aplicar uma busca local, algo que pode vir a ter uma influência na qualidade dos resultados obtidos. Outra maneira de encarar tais aplicações seria aproveitar o histórico de execuções do método (memória) para que haja uma determinada alteração no emprego da transformação.

Cada jogador consiste em um algoritmo ou fórmula que decide o peso de um determinado τ_i. Isso significa que cada jogador deve ser capaz de, dada uma máscara M, e sabendo a definição de τ, calcular o peso w de cada resultado. O que isso significa, basicamente, é que haverá m soluções candidatas (possíveis resultados) que serão criadas aplicando uma transformação na melhor solução conhecida, sendo que uma delas será a vencedora (a melhor). Dados os pesos calculados $w_1, ..., w_m$ para cada combinação de m, a probabilidade estimada p^* para cada resultado é:

$$p_k^* = \frac{w_k}{\sum_{i=1}^{m} w_i} \tag{3.1}$$

A maneira de calcular os pesos pode variar e é, provavelmente, uma das mais importantes definições a se fazer quando aplicando o método para encontrar a solução para um problema. Para fins de exemplificação, vamos continuar usando a definição exemplo de τ_i que fizemos anteriormente. Uma maneira simples de definir o cálculo dos pesos para cada jogador seria ter um vetor $p1$ (com $|p1| = N$) de números reais que define a probabilidade que essa mudança de um bit em particular seja boa. Isso significa que, dados M e m, e considerando M_j para todo j como bits ativados em i e M_k para todo k como bits desativados em i, teríamos:

$$w_i = (\prod p1_{M_j}) \times (\prod 1 - p1_{M_k}) \tag{3.2}$$

Sendo assim, por exemplo, para τ_5 (bits 0 e 2 ativados, 1 desativado com $M = \{57, 123, 190\}$) teríamos $w_5 = p1_{57} \times p1_{190} \times (1 - p1_{123})$.

Cada jogador, portanto, calculará suas probabilidades estimadas para cada resultado e, então, verificam-se todas as cotações para encontrar o resultado que maximiza o ganho esperado do jogador, cuja fórmula é $p_k^* \omega_k$.

Numa definição simples e genérica, podemos ver cada apostador como sendo um vetor de números reais (entre 0 e $1,0$), sendo que cada um desses números reais representa uma probabilidade. Essa codificação permite que usemos da maneira descrita acima para o cálculo do peso. De maneira geral, a forma como esse vetor de probabilidades é definido deve ter alguma relação com o formato das máscaras de maneira que seja possível uma simples aplicação de fórmula (ou algoritmo) para calcular o peso usando as probabilidades (conforme exemplificado anteriormente).

O tamanho da aposta β seguirá o Critério de Kelly [65], conforme visto na seção 2.2.5, dado por:

$$\beta = \frac{p^* \omega - 1}{\omega - 1} \rho \tag{3.3}$$

Se 3.3 der um resultado menor que algum mínimo pré-definido β_{min}, então esse mínimo deve ser usado, a não ser que o mesmo seja maior que o cacife. Ou seja, o valor de β será:

$$\beta = min(\rho, max(\beta_{min}, \frac{p^*\omega - 1}{\omega - 1}\rho)) \tag{3.4}$$

A quantidade de jogadores é dada por P. Cada jogador π começa com cacife ρ_π. Assumimos que quando um jogador quebra, outro será criado e colocado em seu lugar. O novo jogador pode ser determinado aleatoriamente (com relação ao seu cálculo dos pesos) ou pode ser baseado nos melhores jogadores de momento (ou seja, aqueles que estão com cacife mais alto), aplicando qualquer tipo de transformação ou delta para modificar alguns valores. Possivelmente até mesmo uma combinação dos dois (aleatório ou baseado em outros).

3.4 O Método

O método é divido em duas fases (a segunda opcional). Na primeira, temos um número arbitrário de iterações (ou qualquer tipo de condição de parada desejada) . Cada iteração é um evento que consiste em m resultados, com a máscara M criada aleatoriamente. Como já foi mencionado, uma melhor solução conhecida $f(x^*)$ é mantida (e usada como base para aplicação das máscaras). Para o primeiro passo tal solução pode ser gerada totalmente aleatória ou, alternativamente, uma "boa" solução previamente conhecido pode ser utilizada como ponto de partida. A cada passo, inicialmente $\omega_i = \frac{1}{m}, \forall_i$. Outra maneira de resolver isso é pré-calcular todos os pesos dos jogadores (já que eles não serão alterados) e fazer com que todos ω_i sejam estimados baseados nisso (ou seja, na probabilidade média de cada um), o que também pode requerer ajustes.

Há um conjunto de jogadores no qual se define aleatoriamente a ordem de apostas (uma *linear shuffle* simples como mostrado no Algoritmo 1 pode ser usado no vetor de jogadores com esse fim).

Algoritmo 1: Linear shuffle simples

Crie vetor V de tamanho n tal que $V_i = i, 1 \leq i \leq n$
for $1 \leq i \leq N$ **do**
  Selecione j aleatorio tal que $1 \leq j \leq i$
  Troque V_i e V_j
end for

Calcula-se então, para cada um, as probabilidades estimadas de cada resultado (baseando-se nas regras e fórmulas). Com base nisso, seleciona-se o resultado k que dá o maior retorno esperado (desempates podem ser necessários usando-se qualquer critério desejado). O tamanho da aposta β é dada por 3.4.

Depois que cada jogador aposta, o valor de todas as cotações ω podem ser recalculadas para ajustar a qualquer tipo de tendência ou para corrigir deformações, já que como foi discutido anteriormente, a banca modifica suas estimativas com base no comportamento dos jogadores.

Quando todas as apostas tiverem sido completadas, cada resultado m é avaliado aplicando-se $f(.)$ e aquele com a melhor solução geral é escolhido como vencedor. Os jogadores que apostaram no resultado vencedor recebem seus respectivos prêmios. Se houver quaisquer jogadores quebrados após essa etapa, eles são removidos e novos são criados para ocuparem seus lugares. Conforme já discutido, existem diferentes maneiras de se gerar novos jogadores. Se solução vencedora tiver um $f(.)$ melhor que $f(x^*)$ então deve-se atualizar x^* de acordo.

Depois que todos os ajustes forem terminados, uma nova iteração se inicia, como novas máscaras. Quando um certo número de iterações tiver sido executado (ou qualquer outro critério de parada tiver sido satisfeito), a primeira fase termina.

A segunda fase é executada com um determinado número (possivelmente

pequeno) dos melhores jogadores da primeira fase. Agora, as soluções candidatas podem ser criadas usando os pesos dos melhores jogadores como base para os cálculos de maneira aleatória. Dados os exemplos anteriores, a representação de $p1$ pode ser encarada como probabilidades, o que significa que para cada $p1_i$ de um jogador, ativa-se o bit com probabilidade $p1_i$. A ideia por trás dessa abordagem é que cada jogador, a essa altura, tem uma probabilidade bem ajustada para cada bit da solução, e usá-la diretamente (de uma maneira aleatória) tende a fornecer bons resultados. Isso pode ser repetido por um número fixo de passos ou até que outra condição seja satisfeita.

O conceito desta fase é que depois de uma quantidade suficientemente grande de iterações da primeira fase, os jogadores que melhor calculam os pesos das máscaras são os que sobreviveram e, portanto, são os que melhor sabem "escolher" esses valores.

Depois de um certo número de iterações (ou outro critério como, por exemplo, tempo), a segunda fase termina e o método retorna a melhor solução encontrada.

A Figura 3.1 mostra um fluxograma que representa a lógica da primeira fase do método. Os passos 4 e 7 (selecionar máscara aleatória, aplicar transformações e atualizar a melhor solução conhecida) são as partes relativas a busca, enquanto as outras são específicas do modelo de apostas (assim como a seleção de jogadores para a segunda fase). Os passos são, respectivamente:

```
1. Selecionar solução aleatória inicial
2. O critério de parada foi satisfeito?
3. Fim
4. Selecionar máscara aleatória
5. Calcular os pesos dos jogadores para cada escolha
```

6. Calcular as apostas dos jogadores e efetivá-las
7. Aplicar transformações à melhor solução e atualizar seu valor se pertinente
8. Pagar os prêmios das apostas vencedoras dos jogadores
9. Remover jogadores que quebraram substituindo-os por novos

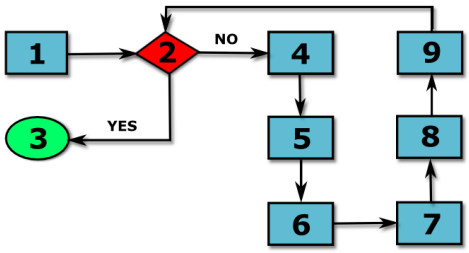

Figura 3.1: Fluxograma da lógica da primeira fase do método

3.5 Algoritmo

Os algoritmos a seguir resumem o método descrito nas seções anteriores. Estamos usando as mesmas variáveis e argumentos que foram usados ao longo do texto. Desta maneira, ω se refere às cotações, M é a máscara (sendo $|M|$ seu tamanho), m é o total de permutações da máscara, τ é a transformação aplicada para a melhor solução, N é o tamanho da solução candidata, β é uma aposta, ρ é o cacife, w é o peso de uma determinada permutação da máscara calculado pelo jogador, $p1$ é a probabilidade que um certo bit seja ativado (1), P é o conjunto de jogadores (de tamanho $|P|$), x^* é a melhor solução conhecida até o momento e $f(.)$ é a função que está sendo otimizada. Também assumimos que alguns argumentos constantes foram definidos, como o total de iterações, tamanho do conjunto de jogadores etc.

Deve-se notar que o código abaixo assume que desejamos maximizar a função $f(.)$, mas pode ser o caso que queiramos minimizar e, portanto, os devidos ajustes

devem ser feitos de acordo com isso.

O Algoritmo 2 descreve a primeira fase do método, enquanto que o Algoritmo 3 descreve uma possível maneira de se realizar a segunda fase. Finalmente, o Algoritmo 4 descreve uma maneira de se gerar novos jogadores a partir de algum outro. Vale notar que a maneira de calcular os pesos já foi explorada na seção 3.3, e basicamente se resume a um produto das probabilidades das respectivas posições do vetor de codificação do agente para cada posição ativada da máscara devido à transformação aplicada.

3.6 Criando Novos Jogadores

Um ponto importante da aplicação deste método é a renovação dos jogadores que quebram (ou seja, cujos cacifes chegam a zero) e são removidos do conjunto de agentes. Sempre que isso acontece, novos jogadores são gerados de maneira a substituir os que forma retirados. O Algoritmo 4 mostra uma maneira de criar novos agentes a partir de outros do conjunto, possivelmente entre os melhores (ou seja, com os maiores cacifes no momento).

Essa não é, naturalmente, a única maneira de gerar novos agentes. Uma outra maneira simples seria simplesmente criar jogadores aleatoriamente, o que tem o benefício de garantir uma boa diversidade. Por outro lado, utilizar agentes com bom desempenho para gerar novos pode garantir uma melhora daqueles, já que apenas são aplicados pequenos ajustes (definidos pela variável θ) aos seus valores.

Uma outra possibilidade seria usar o Algoritmo 4 mas, ao invés de se aproveitar dos melhores jogadores, utilizar codificações diferentes desses (obedecendo a algum critério que seria dependente do problema em questão). A vantagem potencial

Algoritmo 2: Primeira Fase

Selecione solução candidata aleatória e atribua a x^*
for cada iteração **do**
  Seleciona máscara aleatória M de tamanho $|M|$
  for cada jogador i do conjunto P **do**
    for j de 1 a m **do**
      Calcule o peso $w_{i,j}$
    end for
  end for
  for j de 1 a m **do**
    $\omega_j \leftarrow \frac{\sum_{i=1}^{|P|} w_{i,j}}{\sum_{k=1}^{m}\sum_{i=1}^{|P|} w_{i,k}}$
  end for
  for cada jogador i do conjunto P **do**
    for all k tal que $\omega_k > w_{i,k}$ **do**
      $p' \leftarrow \frac{1}{w_{i,k}}$
      $\omega \leftarrow \omega_k$
      $\beta_{i,k} \leftarrow min(\rho, max(\beta_{min}, \frac{p'\omega-1}{\omega-1}\rho))$
      aposte $\beta_{i,k}$ no resultado k
    end for
  end for
  $best \leftarrow 0$
  for cada máscara i em m **do**
    $t \leftarrow f(\tau_i)$
    if $t > best$ **then**
      $best \leftarrow t$
      $b \leftarrow i$
    end if
  end for
  for cada jogador i em P **do**
    if jogador apostou em b **then**
      $\rho_i = \rho_i + \beta_{i,b} \times \omega_b$
    end if
    if $\rho = 0$ **then**
      Remova jogador do conjunto P
      Crie um novo jogador e insira em P
    end if
  end for
  if $best > f(x^*)$ **then**
    $x^* \leftarrow \tau_b$
  end if
end for

Algoritmo 3: Segunda Fase

for cada iteração **do**
  for cada jogador **do**
    Crie solução $s \leftarrow 0$
    for cada bit $1 \leq i \leq N$ **do**
      Ative bit i de s com probabilidade $p1_i$
    end for
    Avalie $f(s)$
  end for
end for

Algoritmo 4: Criando um novo jogador a partir de um outro jogador

Require: $0 < \epsilon < 1$
Selecione jogador π do conjunto atual
$w' \leftarrow w_\pi$
for cada bit $0 < i \leq N$ **do**
  $\theta \leftarrow rand(-\epsilon, \epsilon)$
  $w_i \leftarrow w'_i + \theta$
  $w_i \leftarrow min(1, w_i)$
  $w_i \leftarrow max(0, w_i)$
end for

de se utilizar de tal expediente é a possibilidade de inserir uma grande diversidade ao conjunto, já que dessa maneira se evita justamente determinadas características já presentes e possivelmente dominantes no conjunto atual.

3.7 Parâmetros

Assim como todos métodos de busca heurística [59], existem alguns parâmetros que precisam ser definidos para que este método possa ser executado, e otimizá-los de acordo com o problema a que está sendo aplicado pode apresentar melhores resultados.

Dada a função $f(.)$ que estamos querendo otimizar, o tamanho do espaço de busca deve ser definido pelo tamanho de codificação N de cada solução candidata.

Para cada iteração do método, devemos definir uma máscara M (que pode ser aleatória ou obedecendo a algum tipo de critério pré-definido). O tamanho $|M|$ da máscara pode variar a cada iteração ou permanecer o mesmo.

A quantidade de soluções candidatas, cujas avaliações são executadas para cada combinação de máscara m de modo a determinar a escolha vencedora, também deve ser definida. Isso pode ser uma função do tamanho da máscara ou mesmo simplesmente uma constante, mas deve ser o mesmo tamanho para cada escolha de cada evento.

O tamanho do conjunto de jogadores, o cacife dos jogadores e aposta mínima também são outros fatores a serem pré-definidos e possivelmente permanecerem constantes ao longo da execução do algoritmo.

Além disso, as quantidades de iterações em cada fase de execução do método também devem ser definidas, ou caso seja adequado, diferentes critérios de parada (que não sejam apenas uma função do número de passos executados, podendo ser, por exemplo, tem em segundos de execução).

3.8 Análise

Faremos, nesta seção, uma análise simples e rápida da complexidade de execução no pior caso do método descrito nas seções anteriores, usando a notação O.

Escolher uma solução inicial aleatória é uma operação linear no número de bits de x^* e, portanto, este passo é $O(N)$. Não iremos considerar essa parte, porém, uma vez que concentraremos nossa análise na complexidade de execução de cada iteração do laço principal do método e, como esta etapa acontece antes disso, não será incluída na complexidade final.

Selecionar uma máscara aleatória também é linear com relação ao tamanho da mesma, o que depende de sua representação. Vimos nas seções anteriores diferentes maneiras de representar uma máscara, mas tomando como $|M|$ o tamanho da mesma podemos representar esta etapa como $O(|M|)$.

Após isso, para cada jogador do conjunto de jogadores P, calculamos o peso de cada escolha m. Conforme discutido na seção 3.3, a quantidade de escolhas pode variar e normalmente iremos ter $m = |M|$ ou $m = 2^{|M|}$. Além disso, o cálculo dos pesos pode ser feito em $O(1)$, mas como precisamos atravessar a máscara para o cálculo, essa parte levaria $O(|M|)$. Sendo assim, como temos de executar esse passo para cada jogador e para cada escolha, a complexidade total é $O(|P|m|M|)$, o que pode significar $O(|P||M|^2)$ (para $m = |M|$) ou $O(|P||M|2^{|M|})$ (para $m = 2^{|M|}$).

O cálculo das cotações é feito para cada uma das m escolhas. Temos, também, de calcular a soma de todos os pesos de cada jogador para todas as escolhas e a soma dos pesos de todos os jogadores para uma determinada escolha (naturalmente que esses valores podem ser pré-calculados uma vez e não precisam ser repetidos para cada escolha, uma vez que o valor permanece constante para cada iteração). Uma vez que calculamos a soma dos pesos para cada escolha individualmente, basta somar esses valores para termos o peso total. Baseado nisso, temos que, para cada jogador de P devemos fazer a soma dos pesos de cada escolha m, e depois disso somar todos os valores, dando uma complexidade total de $O(|P|m+m) = O(|P|m)$. Dependendo do valor de m, isso pode se traduzir em $O(|P||M|)$ ou $O(|P|2^{|M|})$.

Os cálculos das probabilidades estimadas e valores das apostas (além do ato de apostar propriamente dito) podem ser feitos em tempo constante para cada jogador. Mesmo que utilizemos um *linear shuffle* para ter uma ordem aleatória de apostas (que tem complexidade $O(|P|)$), como as demais operações são constantes para cada jogador, a complexidade total ainda é $O(|P|)$.

Aplicar as transformações τ à melhor solução conhecida deve ser uma operação polinomial, mas claramente deverá depender da complexidade de τ. Além disso, avaliar $f(.)$ também depende na complexidade da mesma, mas já assumimos que é polinomial. Para esta análise, iremos também assumir que a complexidade de pior caso das transformações é linear no número de bits de x^* e, portanto, $O(N)$. Considerando que a avaliação das transformações e da função $f(.)$ devem ocorrer para cada escolha, temos a complexidade total $O(mN + mf)$. Se assumirmos que f também é linear em N, temos $O(N|M|)$ ou $O(N2^{|M|})$, dependendo da definição de m.

Verificar quais jogadores apostaram na escolha vencedora (e lhes pagar o prêmio, aumentando seus cacifes) pode ser feito em $O(|P|)$. Se assumirmos que

cada jogador aposta no máximo uma vez em cada escolha, podemos simplesmente manter uma lista de apostas para cada escolha e iterando na vencedora para verificar jogadores e valores de apostas para pagar os prêmios. Como temos, no máximo, $|P|$ jogadores, então a complexidade será $O(|P|)$.

Finalmente temos a ação de remover jogadores que quebraram do conjunto de jogadores e repor com novos. Uma vez que a criação de novos jogadores pode ser feita a cada bit (e, portanto, $O(N)$) e que a verificação de que um jogador quebrou pode ser feita em $O(1)$, no pior caso teremos $O(|P| + |P|N) = O(|P|N)$ (caso todos os jogadores tiverem quebrado).

Juntando todas essas etapas, e usando de um abuso de notação para evitar o excesso de barras indicando tamanho de conjunto (portanto usando P para $|P|$ e M para $|M|$), temos:

$$\begin{gathered} O(M + PmM + Pm + P + NM + P + PN) \\ O(PmM + NM + PN) \\ O(PM^2 + NM + PN) \text{ ou } O(P2^M + NM + PN) \end{gathered} \tag{3.5}$$

Se assumirmos que P e N são muito maiores que M, que por sinal normalmente vai ser perto de constante de qualquer maneira (note que isso não é seguro de assumir para 2^M), podemos simplificar para:

$$O(PN) \text{ ou } O(P2^M + PN) \tag{3.6}$$

Se desejarmos, entretanto, manter a análise mais genérica, então precisamos manter a quantidade de escolhas como uma variável própria e não assumir nada com

relação ao tamanho de M. Sendo assim, nossa análise da complexidade de pior caso do método neste caso ainda é dada por:

$$O(PmM + NM + PN) \tag{3.7}$$

Vale lembrar que, mesmo mantendo a análise mais genérica aqui, ainda estamos supondo que a avaliação das transformações τ é linear com relação ao número de bits de x^*. Apesar de isso ser verdade para a grande maioria dos casos, é possível que em alguma aplicação isso não seja verificado. Nesse caso o termo NM pode assumir um peso muito maior na complexidade total.

3.9 Outras Considerações

Nesta seção iremos analisar alguns fatores que podem influenciar na aplicação do modelo, bem como casos extremos, possíveis soluções para essas situações e considerações a respeito das vantagens e desvantagens das mesmas.

3.9.1 Cacife

Dada a maneira como o método funciona, e dependendo da forma como os parâmetros sejam configurados bem como o problema em questão sendo resolvido, é possível que um jogador (ou um subconjunto de jogadores) consiga ter um crescimento exponencial de seu cacife, o que pode resultar em alguns problemas. O mais óbvio é o fato de haver um limite de precisão computacional para guardar esse valor (normalmente um tipo `double` de 64 *bits*), o que pode ocasionar em *overflow*. Outro problema decorrente desse problema é que o jogador pode, por ter conseguido

algumas poucas apostas que permitiram um crescimento exponencial, permanecer na população de agentes, sendo considero uma boa possível solução. Aproveitando-se da metáfora de apostas, seria um caso análogo a alguém que conseguiu alguns poucos ganhos muito grandes e, apesar de não fazer decisões boas, permanece com um cacife alto como se fosse um bom apostador.

Como a Equação 3.4 é baseada numa proporção do cacife, é realmente importante que β_{min} seja definido adequadamente para que o valor de β não seja particularmente "grande". Mesmo com valores bem definidos, porém, é possível que mesmo assim não seja suficiente caso o valor dos prêmios pagos seja suficientemente grande. Como a cotação é definida como o inverso da probabilidade, o problema em si não é o valor ser alto mas, sim, se for alto mesmo que, na verdade, a probabilidade seja relativamente baixa (ou seja, se o erro de estimativa for grande o suficiente).

Mesmo se todas as precauções forem tomadas ainda é possível que esse fenômeno ocorra. Uma possível solução para esses casos extremos é selecionar esses jogadores e colocá-los em uma população de jogadores separada. Usando, novamente, a metáfora de apostas, existem normalmente classes diferentes de apostadores para os mesmos eventos dependendo da quantidade apostada (*high stakes*). Neste caso em particular, basta normalizar os cacifes por algum fator e manter as apostas normalmente. Usando essa solução, caso algum jogador ultrapasse um determinado limite de cacife ele é movido para o grupo de *high stakes* e, uma vez lá, o cacife é dividido por um fator d. Uma vez que seu cacife caia abaixo de um limite, ele volta ao grupo normal de jogadores (com seu cacife multiplicado por d de volta). Por exemplo, digamos que o limite para mudar de grupo seja de 500.000.000 (quinhentos milhões). Quando $\rho > 500.000.000$ o jogador é movido para o *high stakes* e, supondo $d = 1.000.000$ (um milhão), seu cacife passa a ser $\rho' = 500$. Caso o limite inferior seja, por exemplo 200 e aconteça que $\rho' < 200$, então o jogador volta ao grupo anterior e seu novo cacife passa a ser $\rho'' = \rho' \times 1.000.000$. Note-se que, no

caso do grupo de *high stakes*, os valores de apostas (como β_{min} e β_{max}) podem ser diferentes.

Naturalmente que, se for raro um jogador ultrapassar o limite estabelecido, não haverá agentes suficientes para popular um grupo inteiro de jogadores. Nesses casos é possível apenas manter o jogador no grupo normal e usar os fatores em seu cacife para evitar que haja *overflow*, sendo importante definir os valores mínimo e máximo de apostas de acordo. Por outro lado, se houver agentes suficientes para um segundo grupo, isso pode vir a ser inclusive útil para criar uma hierarquia de agentes (quando maior o nível de cacife, supostamente melhores são os jogadores).

3.9.2 Vetores de Probabilidade

Conforme vimos na Seção 3.3, os jogadores podem ser definidos como um vetor de probabilidades (embora essa não seja a única possível). No caso da aplicação do método em sua segunda fase, conforme o Algoritmo 3, essas probabilidades têm uma grande influência na formação das soluções candidatas. Por conta disso, valores muito próximos de $0,5$ podem não ser muito interessantes, uma vez que são mais uma decisão aleatória não informada. Mesmo antes da aplicação da segunda fase, probabilidades perto de 50% são bastante neutras, então é possível que seja desejável evitar esses valores.

Uma maneira de colocar essa estratégia em prática é colocar um peso maior para probabilidades mais extremas ao criar apostadores aleatórios. Dependendo de como se deseja implementar esse dispositivo pode-se, inclusive, fazer com que essa variante só seja aplicada a uma determinada proporção de jogadores.

3.9.3 Apostador Ideal Quebrando

Vamos supor que temos um apostador ideal, ou seja, uma codificação perfeita (exata) da melhor solução para o problema sendo resolvido. Nesse caso, o vetor de probabilidades pode ser preenchido apenas como 0 e 1, o que garante que, na segunda fase, esse agente irá gerar a solução ótima.

Mesmo nesse caso, ainda assim é possível que o apostador não "sobreviva" até o fim da aplicação do método, sendo eliminado no processo por quebrar (ou seja, seu cacife ser reduzido a zero). Se pensarmos em ótimos locais *versus* globais, essa possibilidade passa a fazer mais sentido, sendo inclusive um problema comum a ser lidado em metaheurísticas. Caso a execução do método esteja explorando o hiperespaço de um ótimo local, é possível que a melhor solução (em seus subespaços, conforme são analisados pelo algoritmo) contenha mudanças que pioram a melhor solução conhecida. Dessa maneira o apostador tenderá a ir perdendo seu cacife em apostas infrutíferas até que quebre (ou até que seja salvo caso a execução saia do ótimo local).

Algumas soluções podem ser tentadas para evitar ou pelo menos diminuir a chance de que isso aconteça. Uma delas é usar heurísticas para definir as escolhas vencedoras ao invés de usar uma busca auxiliar (ou *hill climbing*) conforme definido na primeira fase do método. O problema, nesse caso, é que a heurística irá ter um peso enorme na definição final dos apostadores (e, portanto, nas soluções encontradas) e, sendo assim, terá um bom resultados apenas se for uma "boa" heurística para o problema.

Outra solução é manter não apenas uma melhor solução conhecida mas, sim, um conjunto de k melhores soluções conhecidas. Nesse caso podemos ter uma diversidade de representações do hiperespaço na esperança de não estarmos presos a

ótimos locais. Claro que é possível que cada uma dessas soluções esteja presa a um ótimo local específico. Outra desvantagem é que aumentando a dimensão de melhores soluções também aumenta-se a complexidade de aplicação das transformações das máscaras e, portanto, a própria complexidade do método.

4 OTIMIZAÇÃO COMBINATÓRIA

Neste capítulo iremos delinear a utilização do método proposto para problemas de otimização combinatória, de maneira a verificar sua utilidade. Inicialmente iremos nos concentrar no problema do caixeiro viajante assimétrico (ATSP: Asymmetric Traveling Salesman Problem) para, em seguida, analisarmos o problema da mochila binária multidimensional (MKP: Multidimensional Knapsack Problem) e, finalmente, o problema de satisfatibilidade (SAT). Todos os resultados apresentados em tabelas neste capítulo também estão reproduzidos no Apêndice A, assim como algumas outras tabelas com mais resultados.

O uso de problemas de otimização combinatória como aplicação de métodos heurísticos é bastante difundido como uma maneira de testar a usabilidade e utilidade de tais modelos. Por um lado tais problemas normalmente têm aplicações reais importantes indo além do campo teórico (como iremos expor com mais detalhes nas respectivas seções). Por outro lado, a complexidade computacional inerente aos mesmos os torna essencialmente "cobaias" perfeitas para testar eficácia de algoritmos que busquem soluções cada vez melhores. A natureza desses problemas, que geralmente envolve buscar a melhor solução possível dentro de um espaço de busca "enorme" (e por esse termo entendemos algo suficientemente grande e complexo a ponto de tornar inviável computacionalmente uma busca exaustiva), nos permite aplicar diversas estratégias diferentes em busca de melhores resultados.

Um outro fator importante é a existência dos teoremas conhecidos por *No Free Lunch* [87]. Em poucas palavras, o importante resultado desses teoremas é um determinado algoritmo que tenha um bom desempenho ao resolver uma classe de problemas certamente terá um desempenho pior que a média em outra classe (con-

siderando todas as funções objetivas possíveis e todas as métricas possíveis). Sendo assim, na média, considerando todas as classes de problemas possíveis e todas as possíveis métricas, os modelos metaheurísticos terão desempenho similar. Uma consequência direta disso é que alguma diversidade de tais métodos é importante visto que seu desempenho em diferentes classes de aplicações pode variar consideravelmente, e um modelo pode suprir com melhor desempenho em uma classe em que outros não tenham resultados tão bons. De uma maneira geral, isso garante que não existe um método único que possa ser aplicado a todos os tipos de problemas com bons resultados.

Um pouco mais formalmente, por problemas otimização combinatória, entendemos qualquer problema P que consiste em um conjunto de N variáveis e seus respectivos domínios $x_i \in D_i$, um conjunto de restrições aplicado às variáveis, e uma função objetivo $f : D_1 \times D_2 \times ... \times D_N \rightarrow \mathbb{R}$. A solução para o problema é um valor de f que seja ótimo (o maior possível ou o menor possível). Os problemas analisados neste capítulo, e aos quais aplicamos o modelo proposto (e variações), obedecem a essa definição, e sua descrição formal é dada em cada subseção respectiva.

Considerando que, em geral, os problemas de interesse para aplicação de metaheurísticas, além das características discutidas acima, também são problemas NP-completos, então como nenhum algoritmo polinomial para os mesmos é conhecido (e assumindo que $P \neq NP$), uma solução exata tem complexidade de pior caso exponencial e, portanto, a utilidade de modelos de busca heurística torna-se ainda mais acentuada.

Vários métodos metaheurísticos já foram propostos e aplicados a problemas de otimização combinatória, com resultados diversos porém muitas vezes promissores. Exemplos de tais modelos e aplicações incluem *simulated annealing* [47], [14], [78], busca tabu [35], [12], [72], algoritmos genéticos [37], [13], [19], colônias de

formigas [24], [23], [90] e *swarms* de partículas [46], [85], [43], isso sem falar nos modelos híbridos combinando diferentes abordagens dessas [8] ou mesmo heurísticas e outros métodos como programação dinâmica [6], GRASP [29] [68], [30] e ILS [64], [74]. Foge ao escopo deste trabalho fazer uma análise de cada um desses métodos, contudo a diversidade dos mesmos e o constante desenvolvimento de novas estratégias e aplicações é um grande indicativo da fertilidade e utilidade desse campo no desenvolvimento de melhores soluções para problemas complexos de otimização combinatória. As referências dadas neste parágrafo, assim como diversos resumos e estudos a esse respeito [5], [9], [7], também servem de sugestão ao leitor para procurar mais detalhes a respeito das mesmas, bem como alguns trabalhos de aplicação de tais métodos.

4.1 Calibração dos Parâmetros

O método aqui aplicado depende de alguns parâmetros distintos de maneira a alcançar seus resultados. Entre os descritos no Capítulo 3, podemos destacar a máscara e a codificação dos agentes como os mais importantes, bem como outros ainda relevantes como limites de aposta, cacife, tamanho do conjunto de jogadores e critério de parada.

Tais parâmetros foram definidos nas aplicações descritas nas seções seguintes baseados em diversos testes e resultados empíricos. Esforços foram feitos de maneira a calibrar seus valores para que os resultados obtidos fossem melhorados, não apenas numericamente mas também que executassem mais rápido.

O tamanho da máscara e a definição da quantidade de transformações são exemplos bem claros de tais esforços. Foram usadas abordagens diferentes nas aplicações dos problemas, com tamanhos distintos e transformações exponenciais e lineares

com relação ao tamanho de M. Um dos principais aprendizados nesses teste foi que a quantidade de transformações não pode ser muito grande (pelo que se entende, nos casos aqui testado, algo perto de 10 como limite). A razão para isso é que quando a quantidade de transformações cresce, a distribuição das probabilidades tende a se aproximar da média (ou seja, com m transformações, as probabilidades das escolhas tendem a $\frac{1}{m}$). Da mesma maneira, as estimativas dos jogadores tendem a ficar mais diluídas e, juntando-se ao problema das probabilidades, as escolhas passam a ficar muito mais aleatórias, tornando a aplicação do método bem menos eficaz e dificultando qualquer convergência. Além disso, como já foi visto na Seção 3.8, o aumento de M e m tem um impacto grande na complexidade geral do método, de forma que aumentá-los em demasia também traz como efeito colateral uma alta ineficiência computacional.

Nesse contexto, o tamanho do conjunto de jogadores também é um fator importante não apenas pela qualidade dos resultados computacionais mas também pela eficiência de execução do método, conforme já discutido a respeito da complexidade do mesmo. Um numero alto para o conjunto torna a execução muito mais lenta, e para tal valha a pena os resultados devem ser consideravelmente melhores. De certa maneira, contudo, o que se observou é que após determinado limite (que pode variar de acordo com o problema em questão) o método tende a não obter resultados significativamente melhores ao adicionar mais jogadores (embora fique mais lento). A razão observada para isso é que há uma quantidade significativa de agentes que quebram e são removidos do conjunto, o que acaba gerando uma quantidade alta de novos jogadores sendo adicionados. O grupo de agentes com bom desempenho não tende a aumentar linearmente no número total de jogadores, o que acaba não ocasionando grandes melhorias de resultados. Para os problemas aqui desenvolvidos e aplicados, observou-se que os melhores resultados (levando-se em conta eficiência de execução e qualidade das soluções obtidas) foram obtidos com um conjunto de cerca de 25 jogadores.

Ainda relativo ao conjunto de agentes, conforme discutido na Seção **??**, há diversas possíveis maneiras de repor agentes removidos do conjunto com novos. Para os testes a seguir, usamos uma proporção de metade totalmente aleatórios e metade usando o Algoritmo 4 para os melhores jogadores no momento. Alguns testes foram feitos para chegar a essa distribuição e, basicamente, quando a quantidade de novos agentes aleatórios é muito alta, o método tem mais dificuldades de convergir. Por outro lado, ao se utilizar o Algoritmo 4 prevalentemente, a tendencia do modelo é convergir mais rápido mas ao custo de normalmente tender para o primeiro ótimo local encontrado, uma vez que a diversidade do conjunto cai drasticamente. Por conta disso, uma distribuição mais equilibrada meio a meio obteve melhores resultados na média.

Com relação ao tamanho das apostas (mínimo e máximo) e o cacife dos jogadores, foi observado que valores muito altos de cacife para intervalos muito baixos entre os limites de extremos das apostas ocasionaram uma sobrevivência de agentes por mais tempo. Tal consequência não é boa nem ruim por si só, porém pode significar menos diversidade sendo inserida no conjunto de jogadores (uma vez que sua remoção é menos frequente) o que também pode ocasionar uma convergência bem mais lenta. Tal característica poderia significar também uma qualidade melhor de resultados, mesmo que ao custo de um maior tempo de execução, mas não foi o que aconteceu na prática. Ao calibrar os valores de cacife e apostas, chegamos a um determinado patamar ($\rho = 1000$, $\beta_{min} = 1$, $\beta_{max} = 200$) que teve os melhores resultados com eficiência de execução satisfatória para os critérios estabelecidos.

Um outro fator importante, finalmente, é o critério de parada. Decidiu-se por priorizar mais a eficiência de execução para obtenção dos resultados uma vez que vários testes de execução mais lenta não obtiveram resultados muito melhores que justificassem a troca. Estabeleceu-se, por conta disso, o limite de 200.000 iterações do método o que manteve os tempos de execução dentro de limites de segundos para

a maioria dos casos de teste.

Vale lembrar que outros parâmetros fundamentais (como, por exemplo, a codificação dos agentes) também necessitaram de extensa calibração, mas como esses casos são bem mais específicos para cada problema, iremos discuti-los nas respectivas seções de resultados de cada aplicação.

Uma possível melhora para a definição dos parâmetros deste modelo seria utilizar de memória de instâncias anteriores (uma espécie de auto-aprendizado) para permitir que o próprio algoritmo ajustasse seus valores de forma a otimizar seus resultados. Embora seja uma possibilidade com potencial e promissora, foge um pouco do escopo deste trabalho trilhar especificamente essa análise, mas não obstante continua sendo uma interessante alternativa para futuros desenvolvimentos deste método.

4.2 Testes Descartados

Ao propor e aplicar uma metodologia totalmente nova, como é o caso deste trabalho, é de se esperar que, como consequência disso haja uma necessidade muito grande de se ajustar e descobrir qual a melhor forma que o modelo se comporta através de diversas iterações de tentativa e erro.

Sendo assim, para a obtenção dos resultados descritos nas seções seguintes, diversos testes foram realizados e descartados. Além da calibração dos parâmetros do modelo conforme já discutido na Seção 4.1, que requeriram diversas execuções frustadas do método para comparação de valores, ainda houve vários outros ajustes necessários, seja na formatação das codificações dos agentes, seja tentando diferentes abordagens para atacar os problemas (como o uso de uma variante híbrida a ser

discutida mais à frente).

Tendo isso em mente, os resultados aqui apresentados representam apenas uma pequena parcela de todas as execuções e avaliações feitas ao longo desta pesquisa. Muitas execuções frustradas e ajustes de parâmetros obtiveram apenas valores não satisfatórios (ou eficiência aquém do aceitável) e tiveram de ser descartadas. Tudo isso foi responsável por consumir um enorme esforço para que pudéssemos chegar a resultados que mostrem o potencial do método, embora tal empenho seja inerentemente necessário quando se propõe uma nova forma de resolver problemas sem base em qualquer outra metodologia extensivamente estudada.

4.3 ATSP

O problema do caixeiro viajante (TSP - *Traveling Salesman Problem*) é, possivelmente, um dos problemas de otimização combinatória mais famosos, assim como um dos mais bem estudados. O problema, de uma maneira geral, consiste em dados um conjunto de cidades e as distâncias entre cada par de cidades, encontrar a menor rota possível que visite **todas** as cidades exatamente uma vez e que retorne para a cidade original de partida.

A versão de decisão do problema do caixeiro viajante é dado um valor arbitrário, determinar se é possível encontrar um percurso cuja soma das distâncias seja inferior a esse valor. Tal problema é `NP`-completo, e pode ser resolvido com programação dinâmica e complexidade $O(2^n n^2)$, onde n é o número de cidades [16].

Existem diversas aplicações importantes no mundo real do TSP, não sendo apenas importante do ponto de vista teórico, portanto [57]. Desde roteamento de veículos, agrupamento de vetores de dados, sequenciamento de DNA, produção de

microchips etc.

O TSP tem diversas variantes igualmente importantes e muitas quase que igualmente famosas à original. Para o presente trabalho, iremos considerar a versão assimétrica do problema, conhecida como ATSP. A diferença, neste caso, é que as arestas que ligam as cidades podem ter distâncias diferentes dependendo da direção da mesma.

Mais formalmente o ATSP pode ser definido como: dados um grafo direcionado $G = \{V, E\}$ e uma matriz de distâncias $D = (d_{ij})$ de dimensão $n \times n$, calcular um circuito fechado (que pode ser representado por qualquer permutação P do conjunto de vértices $\{1, ..., |V|)$ que minimiza

$$d_{S_n S_1} + \sum_{i=1}^{n-1} d_{S_i S_{i+1}} \tag{4.1}$$

Se tivermos $d_{ij} = d_{ji}, \forall i, j$, temos a variante simétrica do problema do caixeiro viajante (TSP), mas neste caso iremos tratar instâncias em que essa condição não é necessariamente satisfeita.

O TSP é um bom exemplo de problema `NP`-completo [44], [16] e, portanto, bastante adequado como um *benchmark* para a aplicação de um modelo ou técnica de otimização combinatória e busca heurística. Os exemplos usados aqui foram retirados da bem conhecida `TSPLIB` [67] que estão disponíveis livremente e são normalmente usados nesse tipo de aplicação para fins de comparação de resultados.

Por todas as características já descritas, o TSP é um problema particularmente fértil para exploração de metaheurísticas e problemas de busca heurística em geral, sendo um dos principais *benchmarks* para aplicação de tais modelos [78], [19],

[23], [85], [6].

4.3.1 Aplicação

Aplicamos o método para algumas instâncias do ATSP encontradas em [67]. Foram feitas 20 execuções independentes para cada instância, sendo que o mínimo, o máximo e a média dos resultados foram registrados, sendo mostrados na tabela 4.1.

Cada solução candidata foi codificada como um vetor de inteiros, sendo uma simples permutação do conjunto $\{1, ..., |V|\}$, o que significa que não há elementos repetidos. Sendo assim, naturalmente, nosso espaço de busca tem tamanho $|V|!$ total.

As variáveis do método foram escolhidas de maneira empírica, após uma série de testes tentando encontrar os valores que obtivessem os melhores resultados. Não foi uma busca exaustiva pela melhor configuração, mas isso é inerente à natureza de metaheurísticas.

Uma máscara de tamanho 3 ($|M| = 3$ com $1 \leq M_i \leq |V| - 4$) foi usada, e a aplicação da máscara significa que se M_i for usado então 5 elementos são removidos da solução candidata (com índices $M_i, M_i + 1, ..., M_i + 4$) e reinseridos de maneira gulosa (o que significa que reinserimos os elementos na permutação de maneira a minimizar a soma do peso acrescentado). Dado que cada elemento da máscara pode ser ou não usado para criar uma nova solução, temos $m = 2^{|M|} = 8$ possibilidades.

De maneira mais formal, a transformação $\tau_i, 0 \leq i < m$ pode ser definida conforme o Algoritmo 5 abaixo.

Algoritmo 5: Transformação τ_i para ATSP

Require: i (índice da transformação), M (máscara) e x (solução candidata)
  $y \leftarrow x$
  for cada j tal que $i \bmod 2^j = 0$ **do**
    $x' \leftarrow \{y_1, ..., y_{M_j-1}\} \cup \{y_{M_j+5}, ..., y_{|V|}\}$ // *retira* $\{y_{M_j}, ..., y_{M_j+4}\}$ *da solução*
    $best \leftarrow \emptyset$
    for k de 1 a $|x'|$ **do**
      $x'' \leftarrow \{x'_1, ..., x'_k\} \cup \{y_{M_j}, ..., y_{M_j+4}\} \cup \{x'_{k+1}, ..., x'_{|x'|}\}$
      if $f(x'') < f(best)$ **then**
        $best \leftarrow x''$
      end if
    end for
    $y \leftarrow best$
  end for
  return y

Os agentes são codificados como um vetor de números reais (de tamanho $|V|$) em que cada posição indica a probabilidade de que um determinado índice da máscara deva ser acionado. Assim sendo, por exemplo, para qualquer máscara M (lembrando que $|M| = 3$ e $m = 2^{|M|} = 8$) teremos que para uma transformação $\tau_i, 0 \leq i < 8$ a probabilidade de τ_i ser a melhor escolha para o jogador será dada pela Equação 3.2.

Além disso, foi utilizado um conjunto de 25 jogadores e 200.000 iterações do método. O cacife foi definido como 1000, a aposta mínima como 1 e a máxima como 200. Novos jogadores criados após a remoção de algum jogador do conjunto foram feitos 50% das vezes totalmente aleatórios e 50% usando o Algoritmo 4.

4.3.2 Codificação das Soluções e Transformações

A codificação para o problema do ATSP foi pensada com base em uma heurística bem simples e gulosa para modificação de soluções do caixeiro viajante. Como

temos, basicamente, uma permutação de um conjunto de inteiros, é possível fazer uma nova solução apenas mudando uma quantidade de valores dentro desse vetor. Pensando no que tal codificação representa, o que fazemos é basicamente remover arestas da solução e acrescentar outras.

Sendo assim, por exemplo, se temos como solução atual $\{A, B, C, D\}$, o que temos, na verdade, são as arestas AB, BC, CD, DA. Ao retirarmos, digamos, o vértice B da posição atual e incluí-lo no final, o que temos é uma solução $\{A, C, D, B\}$, ou seja, as arestas AC, CD, DB, BA. O que mudamos foi retirar as arestas AB, BC, DA e inserir as arestas AC, CD, BA. De qualquer maneira, o fato de retirar B da sua posição e colocar em alguma outra pode ser feito de maneira trivial, e recalcular o novo custo da solução consome tempo linear.

Um passo adiante seria, ao invés de inserir B em uma posição qualquer (ou fixa como no exemplo dado), inseri-lo de maneira gulosa, ou seja, na posição cuja solução tenha o melhor valor. Isso também pode ser feito de maneira simples. A forma mais trivial seria simplesmente tentar cada possível posição e calcular o novo custo (o que tem complexidade N^2, posto que para cada posição teríamos o cálculo linear dos pesos já descrito). Uma maneira melhor de fazer isso, porém, é calcular o peso do caminho sem o vértice removido e depois, para cada posição simplesmente calcular a diferença. Tal cálculo pode ser feito em tempo constante, posto que precisamos apenas subtrair o custo de uma aresta e somar o custo de duas. Se estamos acrescentando B entre X e Y, sendo o custo atual c, o novo custo será $c' = c - d_{XY} + d_{XB} + d_{BY}$. Logo, o processo inteiro é linear.

Definido isso tudo, fica faltando apenas decidir quantos vértices serão retirados e reinseridos. Seguindo processo semelhante de tentativa e erro definido na Seção 4.1, diversos valores foram testados. Para uma quantidade muito grande de vértices, a tendência do modelo não apenas é ter a execução mais lenta como não

obter resultados tão bons, posto que no limite passa a ser apenas uma heurística de inserção gulosa. Por outro lado, o aumento do número de vértices teve um efeito positivo nos valores menores, obtendo melhores resultados sem comprometer muito a eficiência de execução. Através desse processo, chegou-se à quantidade de cinco posições de vértices sendo movidas.

4.3.3 Resultados e Discussão

Os resultados obtidos nos testes computacionais são mostrados na Tabela 4.1. Cada instância segue a referência de nome especificada em [67], com o tamanho $|V|$ respectivo e seus valores ótimos conhecidos. Também mostrados a média obtida nas 20 execuções independentes, assim como os melhores e piores resultados obtidos nas mesmas. A coluna de percentual é dado como a razão entre a média e o ótimo. Todas as execuções foram completadas em menos de 10 segundos em um MacBook Pro 2013 com 2.3 GHz e 8 Gb RAM. Os valores médios dos tempos computacionais também estão mostrados na tabela. O código executado foi feito em `C++` e compilado usando *gcc*.

As linhas destacadas de cinza são aquelas para as quais a melhor execução do método teve um *gap* de mais de 5% com relação à solução ótima (um total de 6 entre 19). O método proposto parece promissor. Sendo baseado em critérios novos e inéditos, apresenta potencial para ser usado em problemas de otimização relevantes. Apesar de parecer mais apropriado para problemas dinâmicos ainda consegue um bom desempenho em encontrar soluções boas de maneira rápida para problemas notadamente difíceis com espaços de busca enormes, como fica demonstrado pelos resultados obtidos, que também podem ser comparados com outros trabalhos e resultados como [36], [41], [88].

Tabela 4.1: Resultados do método aplicado ao ATSP

instância	$\|V\|$	média	melhor	pior	ótimo	% média	% melhor	tempo
br17	17	39.0	39	39	39	0.00	0.00	1.49
ft53	53	7437.60	7017	7907	6905	7.71	1.62	2.25
ft70	70	40416.10	39853	40874	38673	4.51	3.05	2.63
ftv33	34	1355.00	1298	1392	1286	5.37	0.93	2.08
ftv35	36	1523.35	1490	1553	1473	3.42	1.15	1.95
ftv38	39	1564.10	1546	1598	1530	2.23	1.05	2.08
ftv44	45	1677.80	1664	1703	1613	4.02	3.16	2.14
ftv47	48	1824.55	1782	1865	1776	2.73	0.34	2.22
ftv55	56	1676.05	1609	1727	1608	4.23	0.06	2.31
ftv64	65	1946.35	1867	2027	1839	5.84	1.52	2.51
ftv70	71	2088.50	1998	2169	1950	7.10	2.46	2.61
ftv170	171	3275.05	2979	3376	2755	18.88	8.13	4.44
kro124	100	40875.40	38075	44159	36230	12.82	5.09	3.25
p43	43	5631.65	5622	5638	5620	0.21	0.04	2.11
rbg323	323	1521.00	1484	1560	1326	14.71	11.91	7.16
rbg358	358	1353.05	1324	1372	1163	16.34	13.84	7.54
rbg403	403	2733.55	2691	2759	2465	10.89	9.17	8.27
rbg443	443	3053.35	3004	3092	2720	12.26	10.44	8.96
ry48p	48	15098.20	14774	15324	14442	4.54	2.30	2.17

A Tabela 4.2 mostra a comparação dos resultados obtidos e representados na Tabela 4.1 com heurísticas dedicadas ao problema apresentadas em [36]. Tais heurísticas são identificadas como GKS (*"Greedy Karp-Steele patching"*), RPC (*"Recursive Path Contraction"*) e COP (*"Contract Or Patch"*). As linhas destacadas em cinza são aquelas em que o método aqui proposto teve um desempenho inferior no melhor caso a pelo menos uma das heurísticas comparadas. Infelizmente não é possível uma comparação direta ou mesmo indireta de tempos computacionais posto que em [36] só estão disponíveis os tempos para instâncias aleatórias e não para as instâncias da `TSPLib`. Para esses casos, o trabalho citada mostra tempos de execução abaixo de 10 segundos também para todas instâncias com menos de 1000 vértices para todas as heurísticas (embora os casos de teste não sejam, necessariamente, comparáveis, mas é possível pelo menos ter uma idéia geral de sua eficiência).

Dadas tais condições, e também o fato de que o ATSP pode não ser o melhor problema para usar de *benchmark* com uma abordagem pura como esta (uma vez que sua natureza faz com que haja soluções inválidas mesmo que a codificação seja válida, fazendo com que tenhamos de usar uma permutação de vértices), iremos também aplicar o método, nas Seções 4.4 e 4.5, a problemas mais adequado, assim como comparar também uma abordagem híbrida (usando algoritmos genéticos).

4.4 MKP

O problema da mochila (*knapsack problem*) é um outro problema clássico de otimização combinatória e que tem diversas variantes bastante estudadas [33], [16]. Em sua versão mais tradicional (também conhecida por mochila binária), a solução consiste em, dados um conjunto de itens (cada qual com um valor e um peso) e uma mochila com certa capacidade, encontrar um subconjunto dos itens que maximize soma dos valores e cuja soma dos pesos não seja superior à capacidade da mochila.

Tabela 4.2: Comparação dos resultados do método com heurísticas para o ATSP

instância	$\|V\|$	modelo	GKS	RPC	COP
br17	17	0.00	0.00	0.00	0.00
ft53	53	1.62	12.31	18.64	15.68
ft70	70	3.05	2.84	5.89	1.90
ftv33	34	0.93	8.09	21.62	9.49
ftv35	36	1.15	1.09	21.18	1.56
ftv38	39	1.05	1.05	25.69	3.59
ftv44	45	3.16	5.33	22.26	10.66
ftv47	48	0.34	1.69	28.72	8.73
ftv55	56	0.06	3.05	33.27	4.79
ftv64	65	1.52	2.61	29.09	1.96
ftv70	71	2.46	2.87	22.77	1.85
ftv170	171	8.13	1.38	25.66	3.59
kro124	100	5.09	8.69	23.06	8.79
p43	43	0.04	0.32	0.66	0.68
rbg323	323	11.91	0.00	0.53	0.00
rbg358	358	13.84	0.00	2.32	0.26
rbg403	403	9.17	0.00	0.69	0.20
rbg443	443	10.44	0.00	0.00	0.00
ry48p	48	2.30	4.52	29.50	7.97

Assim como o TSP, também é um problema NP-difícil [33].

O problema de decisão associado ao problema da mochila é de determinar se é possível, dada a capacidade da mochila, encontrar uma solução que satisfaça essa restrição e cuja soma de valores seja pelo menos igual a um valor arbitrário. Esse problema é NP-completo. Para ambos, porém, é possível uma solução pseudo-polinomial relativamente simples, usando programação dinâmica, com complexidade $O(nW)$ onde n é a quantidade de itens e W é a capacidade da mochila [16].

As aplicações em problemas reais também são bastante abundantes e vão desde como otimizar a gravação de arquivos em várias mídias, como cortar materiais de maneira menos custosa e com menos desperdício e até mesmo problemas de decisão de investimentos e mercado financeiro [45], [50].

Diversos métodos já foram aplicados ou desenvolvidos para resolver o problema da mochila, desde modelos tradicionais como programação dinâmica [2] e *branch and bound* [48], passando por algoritmos aproximativos [82] e, naturalmente, metaheurísticas como algoritmos genéticos [28], *swarms* de partículas [43], busca tabu [51], *simulated annealing* [52] etc.

Algumas variantes do problema incluem o problema da mochila multidimensional (MKP, que estudaremos nesta seção), em que há mais de uma mochila a ser preenchida (e os pesos se aplicam a diferentes mochilas), o problema da mochila multiobjetivo, em que há mais de um objetivo a ser otimizado (além de maximizar o valor, pode-se querer também minimizar um custo associado, por exemplo), o problema da mochila quadrática (QKP), em que a função objetivo é quadrática, o problema da soma do subconjunto (que basicamente é um caso especial do problema de decisão da mochila), entre outros.

Nesta seção discutimos os resultados da aplicação do método proposto no problema da mochila binária multidimensional (MKP). Esta é uma variante do problema tradicional da mochila binária, usando m dimensões.

De maneira mais formal, dados um conjunto P de n itens cada qual com lucro p_j, um conjunto C de m recursos cada qual com capacidade c_i e um conjunto W de pesos tais que cada w_{ij} representa quanto o item j consome do recurso i, o objetivo é encontrar o conjunto X de inteiros binários x_j de tamanho n tal que a Equação 4.2 abaixo seja satisfeita:

$$\begin{aligned} \text{maximizar } z &= \sum_{j=1}^{n} p_j x_j \\ \text{sujeito a } \sum_{j=1}^{n} w_{ij} x_j &\leq c_i, 1 \leq i \leq m \\ \text{sendo } x_j &\in \{0,1\}, 1 \leq j \leq n \end{aligned} \tag{4.2}$$

O problema da mochila binária tradicional é um caso específico em que $m = 1$, então não há restrições que devam ser satisfeitas. Ambas as variações, contudo, são bons exemplos de problemas NP-difíceis, e no caso específico do MKP nem mesmo um algoritmo pseudo-polinomial pode existir a não ser que P = NP [56], [54]. Assim como no caso da mochila unidimensional, para o MKP também existem diversas abordagens diferentes para atacar o problema como metaheurísticas [13], programação dinâmica aproximativa [6], computação evolutiva [66], algoritmos genéticos paralelos [18] etc.

Os exemplos usados aqui foram obtidos através da biblioteca de *benchmark* **OR Library MKP**, cujas instâncias foram estabelecidas por [13] e são mantidas para acesso público por [25]. Essas instâncias, além de n itens e m recursos, também

especificam um fator de *aperto* α, com $0 < \alpha < 1$, o qual define a maneira com a qual a capacidade é calculada, dada pela Equação 4.3 abaixo:

$$c_i = \alpha \sum_{j=1}^{n} w_{ij}, 1 \leq i \leq m \tag{4.3}$$

O fator de aperto, basicamente, significa que a capacidade de cada recurso é uma fração da soma de todos os pesos de todos os itens para aquele recurso. As instâncias, dessa maneira, ficam mais restritas à medida que α se aproxima de zero.

4.4.1 Aplicação

O método foi aplicado para instâncias do MKP encontradas em [13], [25]. Foram 20 execuções independentes para cada instância com os resultados médio, maior e menor registrados e mostrados na Tabela 4.3.

Cada solução candidata foi codificada como uma cadeia de números binários do tamanho da entrada (ou seja, igual ao número de itens n) em que cada posição indica se o item está sendo usado ou não (ou seja, basicamente representa o conjunto X). Por essa definição, podemos concluir que o espaço de busca tem tamanho igual a 2^n.

Usando apenas essa representação, todavia, podemos ter soluções inválidas (ou seja, que não satisfaçam as restrições), uma vez que não há nada que impeça que os valores dos *bits* correspondentes satisfaçam as restrições de recursos.

Para evitar que isso aconteça algumas precauções foram tomadas. A primeira

delas é que a solução inicial gerada é sempre válida (criada inserindo gulosamente itens na mochila enquanto as restrições forem satisfeitas) e pode ser vista no Algoritmo 6 abaixo.

Uma variante gulosa para o Algoritmo 6 mantém uma lista em ordem decrescente de todos os itens pelo seu lucro, e ao invés de tentar incluir os itens pela ordem, os insere de acordo com essa lista. Finalmente, uma outra variante é simplesmente percorrer aleatoriamente os itens e tentar inclui-los. Ou seja, é semelhante à variante anterior, mas ou invés de ter uma lista ordenada pelo lucro, temos uma lista de todos os itens com um *linear shuffle* (ver Algoritmo 1), que pode ser executado toda vez que for necessário criar uma nova solução.

Além disso, quando uma nova solução é criada e, ao ser verificada, constata-se que não satisfaz as restrições (ou seja, é inválida), aplicamos um algoritmo guloso que remove itens de lucro menor até que a solução seja válida novamente. Uma vez que é possível que, após a aplicação desse algoritmo, ainda haja itens que possam ser adicionados (por exemplo, se um item de baixo lucro mas com peso alto for retirado), tentamos incluir novos itens de maior lucro (também gulosamente, satisfazendo as restrições) até não ser possível inserir mais nada.

O Algoritmo 7 abaixo ilustra esse procedimento. A segunda parte do algoritmo (ou seja, continuar adicionando itens gulosamente enquanto for possível) é semelhante ao Algoritmo 6, com a diferença que os vetores x e *soma* não estão zerados (ou seja, começa-se com a solução atual, com os valores de x e *soma* atuais).

Uma máscara de tamanho 10 foi usada (ou seja, $|M| = 10$), e a aplicação da máscara significa que se M_i foi usado então o item i deve ser inserido na solução atual. Se a inserção de um item tornar a solução inválida por não satisfazer alguma restrição, aplicamos o Algoritmo 7, com a diferença de que o item recém inserido não

Algoritmo 6: Criação da solução inicial para o MKP

Require: n (número de itens), m (número de recursos), c (capacidades) e w (pesos)
for i de 1 a n **do**
  $x_i \leftarrow 0$
end for
for j de 1 a m **do**
  $soma_j \leftarrow 0$
end for
for i de 1 a n **do**
  $ok \leftarrow$ `verdadeiro`
  for j de 1 a m **do**
    $soma_j \leftarrow soma_j + w_{ij}$
    if $soma_j > c_j$ **then**
      $ok \leftarrow$ `falso`
    end if
  end for
  if $ok =$ `verdadeiro` **then**
    $x_i \leftarrow 1$
  else
    for j de 1 a m **do**
      $soma_j \leftarrow soma_j - w_{ij}$
    end for
  end if
end for
return x

Algoritmo 7: Corrigindo solução inválida para o MKP

```
Require: n, x (solução atual)
  S ← ∅
  for i de 1 a n do
    if x_i = 1 then
      S ← S ∪ {i}
    end if
  end for
  Ordena S crescente, usando o valor dos itens
  for k de 1 a |S| do
    x_k ← 0
    if solução x é válida then
      break
    end if
  end for
  aplicar algoritmo guloso para inserir quantos itens puder
  return x
```

pode ser removido (para fazer isso, basta não inseri-lo no conjunto S). Dado que cada elemento da máscara é usado uma vez para criar uma nova solução candidata, temos $m = |M| = 10$ possíveis escolhas.

A definição do vetor que representa os agentes $p1$ e o cálculo do peso para os jogadores seguiu regras simples já exemplificadas anteriormente e também na Seção 4.3.1. Isso significa que cada jogador tem um vetor de números reais que definem a probabilidade de que aquele índice em particular da máscara deveria ser usado.

Configurações semelhantes ao caso do ATSP foram utilizadas, com um conjunto de 25 jogadores e 200.000 iterações. O cacife foi definido em 1.000, a aposta mínima em 1 e houve um limite máximo de aposta em 200. A criação de um novo jogador após a remoção de algum outro do conjunto foi feita em 50% dos casos completamente aleatória e em 50% dos casos usando o Algoritmo 4.

Nesta aplicação em particular, para a segunda fase do método (Algoritmo 3 também tomamos o cuidado de não criarmos soluções inválidas. Foram usadas duas abordagens, a primeira já exemplificada no Algoritmo 3, mas com a probabilidade mudada para zero caso adicionar um item em particular fosse tornar a solução inválida. A segunda foi de ordenar os valores das probabilidades em ordem não-crescente e adicionar os itens de maneira gulosa enquanto a solução permanecer válida. O valor registrado foi o melhor das duas abordagens, e os Algoritmos 8 e 9 ilustram o que foi descrito neste parágrafo.

Algoritmo 8: Segunda Fase - Variante Um

```
for cada iteração do
  for cada jogador do
    Crie solução s ← 0
    for cada bit 1 ≤ i ≤ N do
      Ative bit i de s com probabilidade p1_i
      if s_i = 1 e solução inválida then
        s_i = 0
      end if
    end for
    Avalie f(s)
  end for
end for
```

4.4.2 Codificação das Soluções e Transformações

Para o problema da mochila, a codificação binária que foi utilizada é a mais simples e fácil de se utilizar. Também se mostrou a mais eficiente. Obviamente que o custo que se paga por tal implementação é o fato, já discutido, de ser possível criar soluções inválidas, posto que não há qualquer verificação de que os valores respeitam as restrições do problema.

O uso dos Algoritmos 6, 7, 8 e 9 têm seu custo também, já que eles são

Algoritmo 9: Segunda Fase - Variante Dois

```
S ← ∅
for i de 1 a n do
  S ← S ∪ i
end for
Ordenar S com base em p1 de forma não-crescente
for cada iteração do
  for cada jogador do
    Crie solução s ← 0
    for i de 1 a |S| do
      j ← S_i
      Ative bit j de s com probabilidade p1_j
      if s_j = 1 e solução inválida then
        s_j = 0
      end if
    end for
    Avalie f(s)
  end for
end for
```

necessários uma vez que não podemos ter soluções inválidas. Mesmo assim, porém, a facilidade de execução é cálculo das soluções ainda é mais vantajosa o suficiente para que valha a pena ter de aplicar tais algoritmos de correção.

Já a aplicação da transformação foi definida de maneira bem simples, inserindo necessariamente o item escolhido na mochila. Inicialmente os testes foram feitos com um algoritmo mais simples, que simplesmente verificava se a inclusão do item era possível e o incluía em caso positivo. Tal abordagem não obteve bons resultados, visto que muito rapidamente chegava-se a soluções que era impossível inserir qualquer novo item e que estavam longe dos ótimos globais.

A solução, nesse caso, foi forçar a inserção do item na mochila obrigatoriamente, o que, novamente, poderia torná-la inviável. Para resolver isso foi aplicado o Algoritmo 7, e essa nova abordagem obteve resultados muito melhores, já que

forçava uma mudança de direção na busca da solução e inseria mais diversidade necessariamente.

4.4.3 Resultados e Discussão

Os resultados obtidos nos testes computacionais são dados pela tabela 4.3. Em [25] há 10 instâncias para cada tipo, com $n = \{100, 250, 500\}$, $m = \{5, 10, 30\}$ e $\alpha = \{0.25, 0.50, 0.75\}$, o que significa que há 270 casos de teste no total. O resultados abaixo incluem uma instância para cada combinação (m, n, α), o que resulta em 27 casos. Como foi o caso com o problema do ATSP, também mostramos a média das 20 execuções, assim como os melhores e piores resultados. Neste caso, algumas instâncias não têm ainda o valor ótimo conhecido, por conta disso a coluna "objetivo" tem tais instâncias marcadas com um asterisco $(*)$, o que significa que ao invés do valor ótimo estamos mostrando o melhor valor conhecido até o momento. As colunas que indicam porcentagem foram calculadas usando a razão do excesso entre a média ou o melhor resultado e o ótimo. As execuções foram completadas em alguns segundos para os casos menores e até alguns minutos para os maiores, usando um MacBook Pro 2013 com 2.3 GHz e 8 Gb RAM. O código executado foi feito em `C++` e compilado usando *gcc*.

O método proposto também teve um bom desempenho neste problema. Embora não use nenhum tipo de heurística específica para o problema, nem mesmo algum tipo de otimização direcionada para o MKP, ainda assim a maioria (16 em 27) das melhores execuções tiveram desempenho dentro de 3% do valor ótimo, e apenas 6 de 27 tiveram um *gap* de mais de 5% (que são as linhas marcadas com cinza). Como esperado, as instâncias em que o fator de aperto foi menor (ou seja, $\alpha = 0,25$) foram as mais difíceis. Estes resultados também são interessantes se comparados com abordagens que usam algoritmos genéticos, inclusive paralelos, em

Tabela 4.3: Resultados do método aplicado ao MKP

m	n	α	média	melhor	pior	objetivo	% média	% melhor
5	100	0.25	22893.00	23720	22354	24381	6.10	2.71
10			21994.70	22361	21693	23064	4.64	3.14
30			20746.90	21379	20103	21946	5.46	2.96
5	250		54809.95	56061	53566	59312	7.60	5.50
10			54484.50	56022	53082	59187	7.95	5.35
30			51568.70	53628	49984	56842*	9.28	5.65
5	500		109663.95	112708	107423	120148	8.72	6.19
10			109375.60	110788	108281	117821	7.17	5.97
30			107324.00	109425	106056	116056*	7.52	5.71
5	100	0.50	41229.85	41890	40699	42757	3.57	2.03
10			40153.85	40867	39751	41395	3.00	1.28
30			39392.25	39890	38997	40767	3.37	1.25
5	250		103922.40	105196	102818	109109	4.75	3.59
10			107543.30	108770	106779	110913	3.04	1.93
30			103270.80	103999	102711	107770*	4.17	3.50
5	500		207250.45	209167	206208	218428	5.11	4.24
10			209144.70	210191	208406	217377	3.79	3.31
30			211457.00	212495	210643	218104*	3.05	2.37
5	100	0.75	58754.50	59123	58455	59822	1.78	1.17
10			56360.55	56818	56033	57375	1.77	0.97
30			56523.65	57017	56213	57494	1.69	0.83
5	250		146650.95	147483	145711	149665	2.01	1.46
10			149754.05	150523	149288	151809	1.35	0.85
30			148140.80	149029	147842	150163	1.35	0.76
5	500		288638.95	290004	287653	295828	2.43	1.97
10			299491.65	300557	298719	304387	1.61	1.26
30			297057.00	298309	297451	301675*	1.27	1.12

que o *gap* é semelhante ou mesmo maior [18]. Algumas técnicas muito específicas para este problema conseguem melhores resultados (e algumas vezes ótimos) como em [32], [13], [10], [81], [84].

4.4.4 Abordagem Híbrida

Analisamos, nesta seção, o uso de Algoritmos Genéticos [37], [58] junto com o método proposto neste trabalho num método híbrido, além dos resultados obtidos com tal aplicação. Para obter os resultados, primeiro utilizamos um algoritmo genético para convergir a uma solução para o problema do MKP e usamos a mesma como solução inicial do método proposto. Nos testes anteriores deste capítulo sempre utilizamos uma solução aleatória inicialmente, então a ideia é que usando algo mais perto do ótimo podemos ter um desempenho ainda melhor.

Para o algoritmo genético, usamos uma população de 1000 indivíduos, 2000 gerações, 20% de elitismo, 75% de cruzamento (usando o método da roleta para seleção) e 5% de novos indivíduos totalmente aleatórios. Também foi usada uma taxa de mutação de $0,01\%$ por bit para todos os novos indivíduos, com exceção daqueles selecionados por elitismo. Os parâmetros para o nosso método permaneceram em $200,000$ iterações com 25 jogadores e máscara de tamanho 10, o que permitiu que ambas as execuções tivessem aproximadamente o mesmo número de avaliações de funções. Também permitiu que o algoritmo genético executasse por tempo suficiente para convergir para um ótimo local.

A maior parte do código foi compartilhada entre ambos os métodos, incluindo o algoritmo guloso para evitar criar soluções inválidas, o que é especialmente importante para o caso de novos indivíduos criados por cruzamento, além dos totalmente aleatórios (assim como mutações inválidas). O mesmo ambiente computacional dos

exemplos anteriores foi usado, embora dessa vez tenha levado um pouco mais de tempo as execuções uma vez que o algoritmo genético foi rodado inicialmente.

Os resultados estão apresentados na Tabela 4.4 abaixo. Os valores mostrados (média, melhor e pior) são sempre os da execução híbrida (ou seja, a execução híbrida foi sempre melhor do que a execução individual em todos os casos), e as porcentagens de *gap* também são dadas com relação aos resultados híbridos. Para estes resultados há também uma coluna com o tempo médio de execução das instâncias (em segundos).

Os resultados são bastante promissores. Usando algoritmos genéticos para alimentar o método com uma solução inicial de um ótimo local fez uma diferença considerável, o que é esperado, uma vez que começão com uma solução "boa" tem um impacto positivo na qualidade das soluções em que os jogadores deverão apostar baseados nas transformações da máscara. Outro ponto interessante é o fato de que o método realmente melhora o resultado das execuções do algoritmo genético, o que é uma boa indicação de que ele pode ser bastante útil para atacar problemas difíceis em configurações híbridas.

Pela Tabela 4.4 podemos ver que nenhuma das melhores execuções teve um *gap* de 2% ou superior (o maior foi de $1,65\%$) e apenas 3 melhores das 27 execuções tiveram um *gap* maior que 1% (com apenas 6 médias também acima desse limite). Ainda houve dois casos que encontraram a solução ótima.

Podemos ver na Tabela 4.5 abaixo uma comparação entre os resultados obtidos usando uma abordagem pura com relação à híbrida (usando como parâmetros os *gaps* entre as médias e as melhores execuções), como uma vantagem clara para o método híbrido.

Tabela 4.4: Resultados do método híbrido aplicado ao MKP

m	n	α	média	melhor	pior	objetivo	% média	% melhor	tempo
5	100	0.25	24195.65	24343	24050	24381	0.76	0.15	59.66
10			22895.60	23055	22726	23064	0.73	0.03	63.69
30			21682.20	21826	21559	21946	1.20	0.54	73.29
5	250		58803.10	59038	58617	59312	0.85	0.46	141.01
10			58328.80	58594	58016	59187	1.44	1.00	140.71
30			55659.50	55901	55462	56842*	2.08	1.65	165.71
5	500		118898.25	119365	118457	120148	1.04	0.65	314.12
10			115995.45	116439	115649	117821	1.54	1.17	338.41
30			114134.45	114468	113822	116056*	1.65	1.36	385.49
5	100	0.50	42507.75	42705	42362	42757	0.58	0.12	51.74
10			41141.05	41306	40989	41395	0.61	0.21	52.69
30			40416.70	40600	40268	40767	0.85	0.40	66.72
5	250		108356.20	108629	108169	109109	0.68	0.43	113.76
10			109875.75	110423	109614	110913	0.93	0.44	127.96
30			106706.85	107039	106253	107770*	0.98	0.67	157.35
5	500		217095.15	217419	216854	218428	0.61	0.46	248.53
10			215395.95	215873	214756	217377	0.91	0.69	255.64
30			216219.10	216885	215734	218104*	0.86	0.55	310.79
5	100	0.75	59625.40	59798	59472	59822	0.32	0.04	46.09
10			57203.20	57375	57142	57375	0.29	0.00	52.15
30			57337.55	57494	57142	57494	0.27	0.00	59.34
5	250		149010.40	149393	148790	149665	0.43	0.18	99.84
10			151182.60	151531	150879	151809	0.41	0.18	110.86
30			149478.40	149855	149051	150163	0.45	0.20	141.03
5	500		294683.65	295161	294412	295828	0.38	0.22	182.94
10			303036.00	303534	302671	304387	0.44	0.28	206.78
30			300273.00	300828	299861	301675*	0.46	0.28	268.52

Já a Tabela 4.6 mostra a comparação dos melhores resultados do método híbrido com dois modelos (*branch and bound* e *branch and bound* com algoritmos genéticos) apresentados em [32]. Os resultados marcados por negrito indicam aqueles em que o modelo aqui representado consegue desempenho superior a pelo menos um dos outros dois comparados. A comparação de tempos de execução não está presente na tabela posto que em [32] os tempos são dados por gráficos de instâncias agregadas com relação à convergência dos métodos. Para problemas com $\alpha = 0.50$, $m = 30$ e $n = 100$, a convergência dos métodos de [32] é de cerca de 50 e 70 segundos, enquanto do método proposto neste trabalho é de 66.72 segundos conforme a Tabela 4.5, ficando portanto entre ambos. Já para instâncias de $\alpha = 0.75$, $m = 30$ e $n = 250$, os resultados de [32] apresentam tempo de convergência entre 45 e 130 segundos, enquanto o modelo deste trabalho fica em 110.86 segundos, novamente de acordo com a Tabela 4.5.

No Apêndice A se encontram alguns outros resultados da aplicação do método a mais instâncias do MKP.

4.5 SAT

O problema de satisfatibilidade booleana (SAT) é outro dos problemas tradicionais e famosos de computação e que tem um valor especial por ter sido o primeiro problema a ser provado NP-completo [15]. Em sua definição mais simples, o problema consiste em determinar se há uma configuração de variáveis booleanas que faça com que determinada fórmula seja satisfeita, ou seja, verdadeira. Embora o problema seja genérico, ele pode ser definido em variantes dependendo da maneira como a fórmula é definida [62].

Para o propósito deste trabalho, consideramos a variante conhecida como

Tabela 4.5: Comparação dos resultados entre os métodos puro e híbrido para o MKP

m	n	α	% média puro	% melhor puro	% média híbrido	% melhor híbrido
5	100	0.25	6.10	2.71	0.76	0.15
10			4.64	3.14	0.73	0.03
30			5.46	2.96	1.20	0.54
5	250		7.60	5.50	0.85	0.46
10			7.95	5.35	1.44	1.00
30			9.28	5.65	2.08	1.65
5	500		8.72	6.19	1.04	0.65
10			7.17	5.97	1.54	1.17
30			7.52	5.71	1.65	1.36
5	100	0.50	3.57	2.03	0.58	0.12
10			3.00	1.28	0.61	0.21
30			3.37	1.25	0.85	0.40
5	250		4.75	3.59	0.68	0.43
10			3.04	1.93	0.93	0.44
30			4.17	3.50	0.98	0.67
5	500		5.11	4.24	0.61	0.46
10			3.79	3.31	0.91	0.69
30			3.05	2.37	0.86	0.55
5	100	0.75	1.78	1.17	0.32	0.04
10			1.77	0.97	0.29	0.00
30			1.69	0.83	0.27	0.00
5	250		2.01	1.46	0.43	0.18
10			1.35	0.85	0.41	0.18
30			1.35	0.76	0.45	0.20
5	500		2.43	1.97	0.38	0.22
10			1.61	1.26	0.44	0.28
30			1.27	1.12	0.46	0.28

Tabela 4.6: Comparação dos resultados com heurísticas

m	n	α	melhor híbrido	B&B	B&B híbrido
5	100	0.25	24343	24373	24381
10			23055	23064	23064
30			**21826**	21516	21946
5	250		59038	59243	59312
10			58594	59071	59164
30			55901	56277	56796
5	500		119365	120082	120148
10			116439	117632	117741
30			114468	115154	115820
5	100	0.75	59798	59960	59965
10			57375	60633	60633
30			57494	60574	60603
5	250		149393	154654	154668
10			**151531**	149641	149704
30			**149855**	149514	149595
5	500		295161	299904	299904
10			303534	306949	307027
30			**300828**	300309	300387

3-SAT, que consiste em determinar a satisfatibilidade de uma fórmula booleana na forma normal conjuntiva (ou seja, uma conjunção de `and` lógicos em que cada cláusula é uma união de `or` lógicos) em que cada cláusula contém 3 variáveis booleanas. Tal variante do problema também é `NP`-completo [15], [16]. É possível reduzir polinomialmente a variante não restrita do SAT ao 3-SAT, de maneira que o crescimento da fórmula seja também polinomial e o resultado é satisfatível se, e somente se, a fórmula original também for [1].

Assim como os demais problemas citados neste trabalho, o SAT também tem diversas aplicações importantes em problemas do mundo real, como por exemplo inteligência artificial, criação de circuitos eletrônicos, prova automática de teorema além de várias outras [34].

Apesar de ser um problema de decisão, várias maneiras de resolver o 3-SAT usando metaheurísticas existem e já foram implementadas. A importância real (além da simbólica) torna-o um excelente candidato para testar abordagens estocásticas e heurísticas bem como um bom parâmetro de *benchmark*. Alguns do exemplos incluem algoritmos genéticos [55, 11, 26], colônias de formigas [83], busca tabu e buscas locais [61], entre outros métodos em geral [39], [40]. Há, também, um algoritmo probabilístico polinomial que que é capaz de resolver corretamente o problema com altas chances quando o mesmo é satisfatível [70].

Para a aplicação deste trabalho, iremos utilizar a variante 3-SAT considerando que cada cláusula tem exatamente 3 variáveis booleanas. Sendo assim, o objetivo do problema é dada a fórmula (em forma normal conjuntiva) f abaixo, determinar se existe alguma atribuição booleana (verdadeiro ou falso) para as variáveis x_{ij} tal que f seja verdadeira.

$$f = (x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge ... \wedge (x_{m1} \vee x_{m2} \vee x_{m3}) \tag{4.4}$$

Repare que há m diferentes cláusulas, todas com exatamente 3 variáveis. Vamos considerar que há n variáveis diferentes $x_1, ..., x_n$, sendo assim podemos assumir que $\exists k, 1 \leq k \leq n$ tal que $x_{ij} = x_k$ ou $x_{ij} = \neg x_k$, $1 \leq i \leq m, 1 \leq j \leq 3$.

Pela maneira como o problema definido podemos verificar que, para f ser satisfeita, é necessário que cada uma das m cláusulas também o sejam. E para que uma determinada cláusula seja satisfeita, basta que uma de suas 3 variáveis o seja.

4.5.1 Aplicação

O método foi aplicado para instâncias do 3-SAT encontradas na biblioteca **SATLIB** [38]. Foram feitas 100 execuções independentes para cada instância relatada. Como cada instância tem um número razoável de diferentes de testes (100 a 1000), foram escolhidos casos de teste aleatoriamente.

Neste problema, ao contrário dos anteriores, a métrica considerada foi a taxa de sucesso (ou seja, quantas execuções resultaram na resposta correta). Tais valores estão descritos na Tabela 4.7. Para este problema, essa tabela já mostra os resultados para três abordagens diferentes: o método sozinho ("puro"), algoritmos genéticos ("ag") e híbrido (ou seja, algoritmo genético seguido do método de apostas).

Ao contrário dos problemas anteriores deste capítulo, estamos aplicando o modelo proposto a um problema de decisão, então portanto a saída do algoritmo deve ser se a fórmula é satisfatível ou não. Para adaptar essa necessidade em nossa abordagem, usamos uma simples heurística para definir a função objetivo a ser

maximizada, que é basicamente quantas cláusulas foram satisfeitas para uma determinada solução candidata. Sendo assim, caso durante a execução do algoritmo seja encontrada algum x tal que $f(x) = m$, então a execução conclui que a fórmula e satisfatível, caso contrário a saída é negativa.

Cada solução candidata foi codificada como uma cadeia de *bits* de tamanho n (total de variáveis) e cada posição indica o valor booleano daquela variável (verdadeiro ou falso). Assim como no caso do MKP, temos que nosso espaço de busca para este problema é de 2^n.

Para este problema, utilizamos uma heurística conhecida para codificar as máscaras, chamada *WalkSat* [73]. Tal heurística consiste, basicamente, em selecionar uma cláusula aleatória que não seja satisfeita e escolher uma variável tal que, ao mudar seu valor, nenhuma outra cláusula previamente satisfeita torna-se falsa [73]. Caso isso não seja possível, escolhe-se uma variável aleatoriamente.

Com base nisso, uma máscara de tamanho 8 foi usada (ou seja, $|M| = 8$), e a aplicação da máscara (a transformação) significa que caso M_i seja usado, então a cláusula escolhida para aplicar a heurística *WalkSat* é a de índice M_i. Os valores de M são definidos como um subconjunto aleatório de todas as cláusulas não satisfeitas. Como cada máscara é usada separadamente, temos neste caso que $m = |M| = 8$.

Os jogadores foram codificados usando o vetor de números reais $p1$ (igualmente de tamanho n) em que cada posição indica a probabilidade de que a respectiva cláusula deva ser prioritariamente escolhida. Sendo assim, dada uma determinada escolha e (lembrando que $0 \leq e \leq m$ e que $m = |M| = 8$), temos que a probabilidade calculada pelo agente de que aquela seja a vencedora será dada pela Equação 3.2 e considerando os valores de $p1$ para cada M_i.

Além disso, como nos problemas anteriores, foi utilizado um conjunto de 25 jogadores e 200.000 iterações do método. O cacife foi definido como 1000, a aposta mínima como 1 e a máxima como 200. Novos jogadores criados após a remoção de algum jogador do conjunto foram feitos 50% das vezes totalmente aleatórios e 50% usando o Algoritmo 4.

Para o algoritmo genético, usamos uma população de 1000 indivíduos, 2000 gerações, 20% de elitismo, 75% de cruzamento (usando o método da roleta para seleção) e 5% de novos indivíduos totalmente aleatórios. Também foi usada uma taxa de mutação de $0,01\%$ por bit para todos os novos indivíduos, com exceção daqueles selecionados por elitismo. Os parâmetros para o nosso método permaneceram em $200,000$ iterações com 25 jogadores e máscara de tamanho 8, o que permitiu que ambas as execuções tivessem aproximadamente o mesmo número de avaliações de funções. Também permitiu que o algoritmo genético executasse por tempo suficiente para convergir para um ótimo local.

4.5.2 Codificação das Soluções e Transformações

Assim como no caso da Seção 4.4.2, a codificação binária para os agentes é uma escolha natural aqui. Não apenas isso, mas ao contrário do caso do MKP, neste caso não há a possibilidade de uma solução ser inválida, já que cada variável lógica pode assumir livremente ambos os valores de verdadeiro e falso, independente do estado da solução.

A aplicação de uma heurística como *WalkSat* em conjunto com a transformação das máscaras pareceu natural dadas as características do problema e a popularidade de tal heurística.

Tabela 4.7: Resultados do método aplicado ao SAT

instância	n	m	% sucesso puro	% sucesso ag	% sucesso híbrido	tempo
`uf20`	20	91	100	100	100	2.36
`uf50`	50	218	100	37	100	5.41
`uf75`	75	325	100	25	100	6.74
`uf100`	100	430	100	13	100	9.39
`uf150`	150	645	85	2	100	13.22
`uf200`	200	860	75	0	90	26.31
`uf225`	225	960	55	0	85	29.91
`uf250`	250	1065	40	0	75	34.56

4.5.3 Resultados e Discussão

Os resultados obtidos nos testes computacionais são dados pela Tabela 4.7. As execuções foram completadas em alguns segundos para os casos menores e até cerca de meio minuto para os maiores (no caso puro), usando um MacBook Pro 2013 com 2.3 GHz e 8 Gb RAM. Para a abordagem híbrida, os tempos computacionais (média) estão disponíveis na coluna "tempo" da Tabela 4.7. O código executado foi feito em `C++` e compilado usando *gcc*.

As três primeiras colunas representam as características das instâncias (o nome em [38], quantidade n de variáveis e quantidade m de cláusulas). As colunas seguintes mostram os valores obtidos de taxa de sucesso para a aplicação do método de apostas ("puro"), algoritmo genético ("ag") e ambos em conjunto ("híbrido"). Assim como no caso do MKP, o algoritmo genético é usado como uma maneira de encontrar uma solução candidata inicial melhor para o método de apostas.

Podemos notar um resultado bem melhor para o uso do método com *WalkSat* do que o desempenho do algoritmo genético, o que já era de se esperar. Apesar disso,

Tabela 4.8: Comparação dos resultados do método aplicado ao SAT

instância	n	m	híbrido	HH	GSAT+Inc	WalkSat+Inc
`uf20`	20	91	100	100	100	100
`uf50`	50	218	100	74.3	93.5	100
`uf75`	75	325	100	-	78	100
`uf100`	100	430	100	-	72.3	100
`uf150`	150	645	100	-	-	98.7
`uf200`	200	860	90	-	-	93.6
`uf225`	225	960	85	-	-	91
`uf250`	250	1065	75	-	-	87.5

o uso do algoritmo genético no método híbrido permitiu que o mesmo tivesse ganhos reais em cima da aplicação do modelo puro, o que reforça os resultados da seção anterior. O mais curioso é que mesmo quando o algoritmo genético, por si só, tem um desempenho fraco, ainda assim o fato de a solução inicial ser melhor do que a simplesmente aleatória é o suficiente para melhorar consideravelmente o desempenho do modelo de apostas.

A título de comparação, a Tabela 4.8 mostra uma comparação dos resultados obtidos e descritos na Tabela 4.7 com os resultados encontrados na literatura, em particular em dois casos de aplicações semelhantes (metaheurísticas aplicadas ao *WalkSat*) e que usaram instâncias de *benchmark* compatíveis. No caso do HH é uma heurística descrita em [3], e para os demais (*GSAT+Inc* e *WalkSat+Inc*) em [4]. Algumas instâncias não foram executadas em todos os modelos por questões de desempenho relatadas pelos mesmos. A comparação com os tempos computacionais também se torna difícil por conta da utilização de outros critérios (no caso das heurísticas, de trocas de valores de variáveis), porém o fato de para instâncias maiores elas terem sérios problemas de eficiência demonstra que o modelo deste trabalho tem uma capacidade de lidar com instâncias maiores com eficiência razoável que outras heurísticas não têm.

A comparação nos permite visualizar que o uso do método, em conjunto com outros (no caso, algoritmo genético) e com uma heurística pré-estabelecida para um determinado problema (como no caso do *WalkSat* para o SAT) nos dá resultados bem importantes e positivos, o que reforça a ideia do potencial de aplicação desse paradigma em problemas complexos de maneira satisfatória.

5 CONCLUSÃO

Este trabalho apresentou um modelo de busca heurística inédito baseado em uma teoria de apostas que foi apresentada de maneira consistente e unificada, as quais consistem na contribuição inicial do mesmo.

Do ponto de vista teórico, esta tese deixa como resultado um arcabouço teórico que agrega e formaliza vários conceitos probabilísticos usualmente necessários na área de apostas. Além disso, o método desenvolvido, também do ponto de vista teórico, pode ser considerado como um resultado em si, posto que sua aplicação pode ser analisada sob essa ótica também.

Considerando os resultados práticos, o presente trabalho mostrou como o novo modelo proposto pode ser aplicado para encontrar soluções para problemas de otimização combinatória complexos, assim como usá-lo em conjunto com outros métodos para obter resultados ainda melhores.

Ao analisar diversas abordagens para diferentes problemas, foi mostrado como os resultados podem ser refinados e melhorados, além de ter um desempenho competitivo com relação a outros modelos já bastante conceituados e amplamente estudados e usados. Com isso, temos um bom indicativo do potencial ainda maior que futuros desenvolvimentos podem vir a trazer, além de mostrar que o método proposto pode ser bastante útil em aplicações reais para solução de problemas complexos, seja utilizado em conjunto com outros modelos ou não.

Vale ressaltar, mais uma vez, que o ineditismo do presente trabalho proposto também impõe diversas dificuldades e barreiras inerentes a sua natureza. Posto que

não quaisquer outros estudos sobre os quais pode-se escorar e melhorar resultados, houve uma grande incidência de resultados frustrados e descartados, aumentando de sobremaneira o esforço necessário para chegar a valores que demonstrassem seu potencial. Conforme já discutido nas Seções 4.1 e 4.2, bem como nas respectivas seções dos problemas práticos estudados, houve diversas maneiras de calibrar, testar e executar o algoritmo de maneira a refinar e obter melhores valores. A insistência na inovação, ao contrário de adicionar uma nova perspectiva a algum tema amplamente estudado, tem esse preço e nenhuma das duas abordagens é necessariamente melhor, mas a existência de ambas é definitivamente necessária.

5.1 Trabalhos Futuros

A partir dos conceitos apresentados e discutidos aqui, os próximos passos consistem na aplicação prática do método em mais problemas de *benchmark* de otimização combinatória, e de maneira mais abrangente usando mais casos de teste e instâncias, bem como a comparação com outros métodos e modelos existentes.

Além disso, a gama de opções a partir das quais novos trabalhos podem ser desenvolvidos sobre o que foi exposto aqui é considerável.

No campo teórico, os fundamentos propostos e explorados no capítulo 2 podem ser ainda mais aprofundados de maneira a cobrir uma quantidade ainda maior de aspectos. Jogos de apostas podem ser ampliados, e estudos no sentido de verificar equilíbrios e reduções para casos de banca ideal podem auxiliar a ampliar o alcance da teoria. Um refinamento maior na questão da distinção entre o conhecimento do tipo e o conhecimento da distribuição de probabilidade do adversário também pode ser útil.

Mais uma possibilidade a ser explorada é a aplicação do modelo proposto em outras classes de problemas um pouco diferentes de otimização combinatória como, por exemplo, sistemas de reconhecimento de padrões ou de classificação.

De uma maneira mais geral, também podemos pensar em aplicações do método com variações diversas de seus parâmetros e da maneira como os mesmos são definidos e/ou tratados. Um exemplo seria considerar codificações mais complexas para os agentes, possivelmente usando algoritmos elaborados no lugar de apenas uma fórmula de cálculo de probabilidade baseada em valores reais. Outra possibilidade é explorar mais a fundo e de maneira mais forte as variações levantadas na Seção 3.9, em especial com relação ao cacife dos jogadores. Uma combinação de algumas dessas estratégias (e mesmo diferentes aplicações das mesmas) aos mesmos problemas (possivelmente os mesmo já explorados no presente trabalho) pode ser uma investigação interessante dos efeitos que tais abordagens pode ter na eficiência do modelo.

Os resultados promissores apresentados na Seção 4.4.4 também apontam para o potencial de aplicações híbridas de metaheurísticas. Assim sendo, uma possibilidade é explorar outros métodos (como Simulated Anealing e Ant Colonies) ao invés de Algoritmos Genéticos. E, além disso, continuar aplicando os modelos híbridos em diferentes problemas de aplicações reais.

5.2 Considerações Finais

Consideramos que o trabalho vem cumprindo com seus objetivos e deixa uma boa contribuição teórica, bem como um enorme potencial para aplicação prática, o qual será desenvolvido a seguir.

Acreditamos que os resultados dos desenvolvimentos propostos na tese (embasamento teórico e algoritmos) possam ser utilizados em outros contextos e sirvam de base para a elaboração de métodos computacionais heurísticos para a solução de outros problemas de otimização de interesse geral.

REFERÊNCIAS

[1] AHO, A. V.; HOPCROFT, J. E.; ULLMAN, J. **The Design and Analysis of Computer Algorithms**. 1st.ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1974.

[2] ANDONOV, R.; POIRRIEZ, V.; RAJOPADHYE, S. Unbounded knapsack problem: dynamic programming revisited. **European Journal of Operations Research**, [S.l.], v.123, n.2, p.394–407, June 2000.

[3] BADER-EL-DEN, M. B.; POLI, R. A GP-based Hyper-heuristic Framework for Evolving 3-SAT Heuristics. In: ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 9., New York, NY, USA. **Proceedings...** ACM, 2007. p.1749–1749. (GECCO '07).

[4] BADER-EL-DEN, M. B.; POLI, R. Inc*: an incremental approach for improving local search heuristics. In: EUROPEAN CONFERENCE, EVOLUTIONARY COMPUTATION IN COMBINATORIAL OPTIMIZATION, 8., Naples, Italy. **Proceedings...** Springer, 2008. p.194–205. (EvoCOP, v.4972).

[5] BAGHEL, M.; AGRAWAL, S.; SILAKARI, S. Survey of Metaheuristic Algorithms for Combinatorial Optimization. **International Journal of Computer Applications**, [S.l.], v.58, n.19, p.21–31, Nov. 2012.

[6] BERTSIMAS, D.; DEMIR, R. An approximate dynamic programming approach to multidimensional knapsack problems. **Management Science**, [S.l.], v.48, n.4, p.550–565, 2002.

[7] BIANCHI, L. et al. A survey on metaheuristics for stochastic combinatorial optimization. **Natural Computing**, [S.l.], v.8, n.2, p.239–287, 2009.

[8] BLUM, C. et al. Hybrid Metaheuristics in Combinatorial Optimization: a survey. **Appl. Soft Comput.**, Amsterdam, The Netherlands, The Netherlands, v.11, n.6, p.4135–4151, Sept. 2011.

[9] BLUM, C.; ROLI, A. Metaheuristics in Combinatorial Optimization: overview and conceptual comparison. **ACM Comput. Surv.**, New York, NY, USA, v.35, n.3, p.268–308, Sept. 2003.

[10] BOUSIER, S. et al. A multi-level search strategy for the 0-1 multidimensional knapsack problem. **Discrete Applied Mathematics**, [S.l.], v.158, n.2, p.97–109, 2010.

[11] BURJORJEE, K. M. Explaining Adaptation in Genetic Algorithms With Uniform Crossover: the hyperclimbing hypothesis. **CoRR**, [S.l.], v.abs/1204.3436, 2012.

[12] CARLTON, W.; BARNES, J. W. Solving the Traveling Salesman Problem with Time Windows Using Tabu Search. **IIE Trans**, Oxford, UK, UK, v.28, n.5, p.617–629, 1996.

[13] CHU, P. C.; BEASLEY, J. E. A genetic algorithm for the multiconstrained knapsack problem. **Journal of Heuristics**, [S.l.], n.4, p.63–86, 1998.

[14] CONNOLLY, D. T. An improved annealing scheme for the QAP. **European Journal of Operational Research**, [S.l.], v.46, n.1, p.93–100, 1990.

[15] COOK, S. A. The complexity of theorem-proving procedures. In: STOC '71: PROCEEDINGS OF THE THIRD ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, New York, NY, USA. **Anais...** ACM, 1971. p.151–158.

[16] CORMEN, T. H. et al. **Introduction to Algorithms**. 2.ed. [S.l.]: MIT Press, 2001.

[17] COTTA, C.; HEMERT, J. I. van (Ed.). **Recent Advances in Evolutionary Computation for Combinatorial Optimization**. [S.l.]: Springer, 2008. (Studies in Computational Intelligence, v.153).

[18] DANTAS, B. A.; CACERES, E. N. Implementações paralelas para o problema da mochila multidimensional usando algoritmos genéticos. **Simpósio Brasileiro de Pesquisa Operacional**, [S.l.], 2014.

[19] DEEP, K.; MEBRATHU, H. Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem. **International Journal of Combinatorial Optimization Problems and Informatics**, [S.l.], v.2, p.1–23, 2011.

[20] DEMASI, P.; SZWARCFITER, J.; CRUZ, A. Heuristics search method based on betting theory. **Simpósio Brasileiro de Pesquisa Operacional**, [S.l.], p.1948–1959, 2014.

[21] DEMASI, P.; SZWARCFITER, J.; CRUZ, A. Betting Theory and Search Heuristics Applied to Combinatrial Optimiztion. **Congresso Nacional de Matemática Aplicada e Computacional**, [S.l.], 2014.

[22] DEMASI, P.; SZWARCFITER, J.; CRUZ, A. A novel heuristic search method based on betting theory applied to combinatorial optimization. **International Transactions in Operational Research**, [S.l.], 2014. Submetido.

[23] DORIGO, M.; GAMBARDELLA, L. M. Ant Colony System: a cooperative learning approach to the traveling salesman problem. **IEEE Transactions on Evolutionary Computation**, [S.l.], v.1, p.53–66, 1997.

[24] DORIGO, M.; MANIEZZO, V.; COLORNI, A. **Positive Feedback as a Search Strategy**. [S.l.]: Politecnico di Milano, 1991. (91–016).

[25] DRAKE, J. **MKP Instances**. [Online; accessado 16-Maio-2014], `http://www.cs.nott.ac.uk/~jqd/mkp/`.

[26] EIBEN, A.; HAUW, J. V. D. Solving 3SAT with adaptive genetic algorithms. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION. **Anais...** [S.l.: s.n.], 1997.

[27] ENGELBRECHT, A. P. **Fundamentals of Computational Swarm Intelligence**. 1st.ed. [S.l.]: Wiley, 2006.

[28] FALKENAUER, E. **Genetic Algorithms and Grouping Problems**. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[29] FEO, T.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, [S.l.], n.8, p.67–71, 1989.

[30] FERREIRA, C. et al. A GRASP based approach to the generalized minimum spanning tree problem. **Expert Systems With Application**, [S.l.], v.39, n.3, p.3526–3536, 2012.

[31] FOGEL, L. J.; OWENS, A. J.; WALSH, M. J. **Artificial Intelligence through Simulated Evolution**. New York, USA: John Wiley, 1966.

[32] GALLARDO, J. E.; COTTA, C.; FERNáNDEZ, A. J. Solving the Multidimensional Knapsack Problem Using an Evolutionary Algorithm Hybridized with Branch and Bound. In: MIRA, J.; ÁLVAREZ, J. R. (Ed.). **Artificial Intelligence and Knowledge Engineering Applications**: a bioinspired approach. [S.l.]: Springer Berlin Heidelberg, 2005. p.21–30. (Lecture Notes in Computer Science, v.3562).

[33] GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability; A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman & Co., 1990.

[34] GIUNCHIGLIA, E.; TACCHELLA, A. In: GIUNCHIGLIA, E.; TACCHELLA, A. (Ed.). **Theory and Applications of Satisfiability Testing**. [S.l.]: Springer, 2004. (Lecture Notes in Computer Science, v.2919).

[35] GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers & Operations Research**, Oxford, UK, UK, v.13, n.5, p.533–549, Jan. 1986.

[36] GLOVER, F. et al. Construction heuristics for the asymmetric TSP. **European Journal of Operational Research**, [S.l.], 2000.

[37] HOLLAND, J. H. **Adaptation in natural and artificial systems**. Cambridge, MA, USA: MIT Press, 1992.

[38] HOOS, H. H. SATLIB: an online resource for research on sat. In: IN SAT 2000. **Anais...** IOS Press, 2000. p.283–292.

[39] HOOS, H. H.; O'NEILL, K. **Stochastic Local Search Methods for Dynamic SAT – An Initial Investigation**. [S.l.]: AAAI-2000 Workshop Leveraging Probability and Uncertainty in Computation, 2000.

[40] HOOS, H. H.; STüTZL, T. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. **Artificial Intelligence**, [S.l.], v.112, n.1–2, p.213–232, Aug. 1999.

[41] J. SUN Q. ZHANG, E. T. DE/EDA: a new evolutionary algorithm for global optimization. **Information Sciences 169**, [S.l.], 2005.

[42] JONG, K. D. **Evolutionary Computation**. [S.l.]: MIT Press, 2006.

[43] KANG, K. A Fast Particle Swarm Optimization Algorithm for Large Scale Multidimensional Knapsack Problem. **Journal of Computational Information Systems**, [S.l.], v.8, n.7, p.2709–2716, 2012.

[44] KARP, R. Reducibility among combinatorial problems. In: MILLER, R.; THATCHER, J. (Ed.). **Complexity of Computer Computations**. [S.l.]: Plenum Press, 1972. p.85–103.

[45] KELLERER, H.; PFERSCHY, U.; PISINGER, D. **Knapsack Problems**. [S.l.]: Springer, Berlin, Germany, 2004.

[46] KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS. **Proceedings...** [S.l.: s.n.], 1995. p.1942–1948.

[47] KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **SCIENCE**, [S.l.], v.220, n.4598, p.671–680, 1983.

[48] KOLESAR, P. J. A Branch and Bound Algorithm for the Knapsack Problem. **Management Science**, [S.l.], v.13, n.9, p.723–735, May 1967.

[49] LAARHOVEN, P. J. M. van; AARTS, E. H. L. **Simulated Annealing**: theory and applications. 1st.ed. [S.l.]: Springer, 1987. (Mathematics and Its Applications).

[50] LAGOUDAKIS, M. G. **The 0-1 Knapsack Problem – An Introductory Survey**. [S.l.]: University of Southwestern Louisiana, 1996.

[51] LI, V. C.; CURRY, G. L. Solving Multidimensional Knapsack Problems with Generalized Upper Bound Constraints Using Critical Event Tabu Search. **Comput. Oper. Res.**, Oxford, UK, UK, v.32, n.4, p.825–848, Apr. 2005.

[52] LIU, A. et al. Improved Simulated Annealing Algorithm Solving for 0/1 Knapsack Problem. In: SIXTH INTERNATIONAL CONFERENCE ON INTELLIGENT SYSTEMS DESIGN AND APPLICATIONS - VOLUME 02, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2006. p.1159–1164. (ISDA '06).

[53] LOURENCO, H.; O., M.; T., S. Iterated Local Search. In: HANDBOOK OF METAHEURISTICS. **Anais...** Kluwer Academic Publishers, 2003. p.321–353. (International Series in Operations Research and Management Science, v.57).

[54] MAGAZINE, M. J.; CHERN, M. S. A fully polynomial approximation schemes for multidimensional knapsack problem. **Mathematics of Operations Research**, [S.l.], n.9, p.244–247, 1984.

[55] MARCHIORI, E.; ROSSI, C. A Flipping Genetic Algorithm for Hard 3-SAT Problems. In: PROCEEDINGS OF THE GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE (GECCO-99). **Anais...** Morgan Kaufmann, 1999. p.393–400.

[56] MARTELLO, S.; TOTH, P. **Knapsack Problems**: algorithms and computer implementation. [S.l.]: John Wiley and Sons, 1990.

[57] MATAI, R.; SINGH, S.; MITTAL, M. L. **Traveling Salesman Problem**: an overview of applications, formulations, and solution approaches. [S.l.]: InTech, 2010.

[58] MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. 3rd.ed. [S.l.]: Springer, 1999.

[59] MICHALEWICZ, Z.; FOGEL, D. B. **How to Solve It**: modern heuristics. 2rd.ed. [S.l.]: Springer, 2004.

[60] MOORE, J. **Complete Book of Sports Betting**: a new, no-nonsense approach to sports gambling. [S.l.]: Lyle Stuart, 1996.

[61] PANKRATOV, D.; BORODIN, A. On the Relative Merits of Simple Local Search Methods for the MAX-SAT Problem. In: STRICHMAN, O.; SZEIDER, S. (Ed.). **Theory and Applications of Satisfiability Testing ¿ SAT 2010**. [S.l.]: Springer Berlin Heidelberg, 2010. p.223–236. (Lecture Notes in Computer Science, v.6175).

[62] PAPADIMITRIOU, C. H. On selecting a satisfying truth assignment. In: FOUNDATIONS OF COMPUTER SCIENCE, 1991. PROCEEDINGS., 32ND ANNUAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 1991. p.163–169.

[63] PEDERSEN, M. **Tuning & Simplifying Heuristical Optimization**. 2010. Tese (Doutorado em Ciência da Computação) — University of Southampton,.

[64] PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An Iterated Local Search heuristic for the Heterogeneous Fleet Vehicle Routing Problem. **Journal of Heuristics**, [S.l.], v.19, n.2, p.201–232, 2013.

[65] PIOTROWSKI, E.; SCHROEDER, M. **Kelly criterion revisited**: optimal bets. [S.l.]: University of Bialtystok, Department of Theoretical Physics, 2006. Departmental Working Papers 24.

[66] RAIDL, G. R.; GOTTLIEB, J. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms. A case study for the multidimensional knapsack problem. **Evolutionary Computation**, [S.l.], n.13, p.441–475, 2005.

[67] REINELT, G. TSPLIB - a Traveling Salesman Problem Library. **European Journal of Operations Research**, [S.l.], 1991.

[68] RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures. In: HANDBOOK OF METAHEURISTICS. **Anais...** Kluwer Academic Publishers, 2003. p.219–249. (International Series in Operations Research and Management Science, v.57).

[69] ROSS, S. **Introduction to Probability Models**. 9th.ed. [S.l.]: Academic Press Inc., 2006.

[70] SCHöNING, U. A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 40., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1999. p.410–. (FOCS '99).

[71] SCHWARTZ, D. **Roll the bones**: the history of gambling. [S.l.]: Penguin Group, 2007.

[72] SCRICH, C. R.; ARMENTANO, V. A.; LAGUNA, M. Tardiness minimization in a flexible job shop: a tabu search approach. **J. Intelligent Manufacturing**, [S.l.], v.15, n.1, p.103–115, 2004.

[73] SELMAN, B.; KAUTZ, H.; COHEN, B. Local Search Strategies for Satisfiability Testing. In: DIMACS SERIES IN DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE. **Anais...** [S.l.: s.n.], 1995. p.521–532.

[74] SEMAAN, G. S. et al. Um algoritmo ILS aplicado ao Problema do Caixeiro Viajante com Coleta e Entrega. **Congresso Brasileiro de Inteligência Computacional**, [S.l.], 2011.

[75] SPALL, J. **Introduction to Stochastic Search and Optimization**. [S.l.]: John Wiley & Sons, Inc., 2003.

[76] STARK, B. **Special Situation Investing**: hedging, arbitrage, and liquidation. [S.l.]: Dow-Jones Publishers, 1983.

[77] SöRENSEN, K. Metaheuristics the metaphor exposed. **International Transactions in Operational Research**, [S.l.], v.22, n.1, p.3–18, 2015.

[78] TEOH, E. J.; TANG, H.; TAN, K. C. A Columnar Competitive Model with Simulated Annealing for Solving Combinatorial Optimization Problems. In: IJCNN. **Anais...** IEEE, 2006. p.3254–3259.

[79] THORP, E. **The Mathematics of Gambling**. [S.l.]: Gambling Times Publishing, 1984.

[80] THORP, E. The Kelly Criterion in Blackjack, Sports Betting, and the Stock Market. In: INTERNATIONAL CONFERENCE ON GAMBLING AND RISK TAKING, 10. **Anais...** [S.l.: s.n.], 1997.

[81] VASQUEZ, M.; VIMONT, Y. Improved results on the 0-1 multidimensional knapsack problem. **European Journal of Operations Research**, [S.l.], v.165, n.1, p.70–81, 2005.

[82] VAZIRANI, V. V. **Approximation Algorithms**. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[83] VILLAGRA, M.; BARáN, B. Ant Colony Optimization with Adaptive Fitness Function for Satisfiability Testing. In: LEIVANT, D.; QUEIROZ, R. de (Ed.). **Logic, Language, Information and Computation**. [S.l.]: Springer Berlin Heidelberg, 2007. p.352–361. (Lecture Notes in Computer Science, v.4576).

[84] VIMONT, Y.; BOUSSIER, S.; VASQUEZ, M. Reduced cost propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. **Journal of Combinatorial Optimization**, [S.l.], v.15, n.2, p.165–178, 2008.

[85] WANG, K. P. et al. Particle swarm optimization for traveling salesman problem. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND CYBERNETICS. **Proceedings...** [S.l.: s.n.], 2003. p.1583–1585.

[86] WILSON, A. **The Casino Gambler's Guide**. [S.l.]: Harpercollins, 1970.

[87] WOLPERT, D. H.; MACREADY, W. G. **No Free Lunch Theorems for Search**. [S.l.]: Santa Fe Institute, 1995. Working Papers. (95-02-010).

[88] Y. NAGATA, D. S. A new genetic algorithm for the asymmetric traveling salesman problem. **Expert Systems with Applications**, [S.l.], 2012.

[89] YAO, K. **Weighing the Odds in Sports Betting**. [S.l.]: Pi Yee Press, 2007.

[90] YUA, B.; YANGA, Z.-Z.; YAOB, B. An improved ant colony optimization for vehicle routing problem. **European Journal of Operational Research**, [S.l.], v.196, n.1, p.171–176, 2009.

APÊNDICE A

Tabela A.1: Resultados do método aplicado ao ATSP

instância	$\|V\|$	média	melhor	pior	ótimo	% média	% melhor	tempo
br17	17	39.0	39	39	39	0.00	0.00	1.49
ft53	53	7437.60	7017	7907	6905	7.71	1.62	2.25
ft70	70	40416.10	39853	40874	38673	4.51	3.05	2.63
ftv33	34	1355.00	1298	1392	1286	5.37	0.93	2.08
ftv35	36	1523.35	1490	1553	1473	3.42	1.15	1.95
ftv38	39	1564.10	1546	1598	1530	2.23	1.05	2.08
ftv44	45	1677.80	1664	1703	1613	4.02	3.16	2.14
ftv47	48	1824.55	1782	1865	1776	2.73	0.34	2.22
ftv55	56	1676.05	1609	1727	1608	4.23	0.06	2.31
ftv64	65	1946.35	1867	2027	1839	5.84	1.52	2.51
ftv70	71	2088.50	1998	2169	1950	7.10	2.46	2.61
ftv170	171	3275.05	2979	3376	2755	18.88	8.13	4.44
kro124	100	40875.40	38075	44159	36230	12.82	5.09	3.25
p43	43	5631.65	5622	5638	5620	0.21	0.04	2.11
rbg323	323	1521.00	1484	1560	1326	14.71	11.91	7.16
rbg358	358	1353.05	1324	1372	1163	16.34	13.84	7.54
rbg403	403	2733.55	2691	2759	2465	10.89	9.17	8.27
rbg443	443	3053.35	3004	3092	2720	12.26	10.44	8.96
ry48p	48	15098.20	14774	15324	14442	4.54	2.30	2.17

Tabela A.2: Comparação dos resultados do método com heurísticas para o ATSP

instância	$\|V\|$	modelo	GKS	RPC	COP
br17	17	0.00	0.00	0.00	0.00
ft53	53	1.62	12.31	18.64	15.68
ft70	70	3.05	2.84	5.89	1.90
ftv33	34	0.93	8.09	21.62	9.49
ftv35	36	1.15	1.09	21.18	1.56
ftv38	39	1.05	1.05	25.69	3.59
ftv44	45	3.16	5.33	22.26	10.66
ftv47	48	0.34	1.69	28.72	8.73
ftv55	56	0.06	3.05	33.27	4.79
ftv64	65	1.52	2.61	29.09	1.96
ftv70	71	2.46	2.87	22.77	1.85
ftv170	171	8.13	1.38	25.66	3.59
kro124	100	5.09	8.69	23.06	8.79
p43	43	0.04	0.32	0.66	0.68
rbg323	323	11.91	0.00	0.53	0.00
rbg358	358	13.84	0.00	2.32	0.26
rbg403	403	9.17	0.00	0.69	0.20
rbg443	443	10.44	0.00	0.00	0.00
ry48p	48	2.30	4.52	29.50	7.97

Tabela A.3: Resultados do método aplicado ao MKP

m	n	α	média	melhor	pior	ótimo	% média	% melhor
5	100	0.25	22893.00	23720	22354	24381	6.10	2.71
10			21994.70	22361	21693	23064	4.64	3.14
30			20746.90	21379	20103	21946	5.46	2.96
5	250		54809.95	56061	53566	59312	7.60	5.50
10			54484.50	56022	53082	59187	7.95	5.35
30			51568.70	53628	49984	56842*	9.28	5.65
5	500		109663.95	112708	107423	120148	8.72	6.19
10			109375.60	110788	108281	117821	7.17	5.97
30			107324.00	109425	106056	116056*	7.52	5.71
5	100	0.50	41229.85	41890	40699	42757	3.57	2.03
10			40153.85	40867	39751	41395	3.00	1.28
30			39392.25	39890	38997	40767	3.37	1.25
5	250		103922.40	105196	102818	109109	4.75	3.59
10			107543.30	108770	106779	110913	3.04	1.93
30			103270.80	103999	102711	107770*	4.17	3.50
5	500		207250.45	209167	206208	218428	5.11	4.24
10			209144.70	210191	208406	217377	3.79	3.31
30			211457.00	212495	210643	218104*	3.05	2.37
5	100	0.75	58754.50	59123	58455	59822	1.78	1.17
10			56360.55	56818	56033	57375	1.77	0.97
30			56523.65	57017	56213	57494	1.69	0.83
5	250		146650.95	147483	145711	149665	2.01	1.46
10			149754.05	150523	149288	151809	1.35	0.85
30			148140.80	149029	147842	150163	1.35	0.76
5	500		288638.95	290004	287653	295828	2.43	1.97
10			299491.65	300557	298719	304387	1.61	1.26
30			297057.00	298309	297451	301675*	1.27	1.12

Tabela A.4: Resultados do método híbrido aplicado ao MKP

m	n	α	média	melhor	pior	objetivo	% média	% melhor	tempo
5	100	0.25	24195.65	24343	24050	24381	0.76	0.15	59.66
10			22895.60	23055	22726	23064	0.73	0.03	63.69
30			21682.20	21826	21559	21946	1.20	0.54	73.29
5	250		58803.10	59038	58617	59312	0.85	0.46	141.01
10			58328.80	58594	58016	59187	1.44	1.00	140.71
30			55659.50	55901	55462	56842*	2.08	1.65	165.71
5	500		118898.25	119365	118457	120148	1.04	0.65	314.12
10			115995.45	116439	115649	117821	1.54	1.17	338.41
30			114134.45	114468	113822	116056*	1.65	1.36	385.49
5	100	0.50	42507.75	42705	42362	42757	0.58	0.12	51.74
10			41141.05	41306	40989	41395	0.61	0.21	52.69
30			40416.70	40600	40268	40767	0.85	0.40	66.72
5	250		108356.20	108629	108169	109109	0.68	0.43	113.76
10			109875.75	110423	109614	110913	0.93	0.44	127.96
30			106706.85	107039	106253	107770*	0.98	0.67	157.35
5	500		217095.15	217419	216854	218428	0.61	0.46	248.53
10			215395.95	215873	214756	217377	0.91	0.69	255.64
30			216219.10	216885	215734	218104*	0.86	0.55	310.79
5	100	0.75	59625.40	59798	59472	59822	0.32	0.04	46.09
10			57203.20	57375	57142	57375	0.29	0.00	52.15
30			57337.55	57494	57142	57494	0.27	0.00	59.34
5	250		149010.40	149393	148790	149665	0.43	0.18	99.84
10			151182.60	151531	150879	151809	0.41	0.18	110.86
30			149478.40	149855	149051	150163	0.45	0.20	141.03
5	500		294683.65	295161	294412	295828	0.38	0.22	182.94
10			303036.00	303534	302671	304387	0.44	0.28	206.78
30			300273.00	300828	299861	301675*	0.46	0.28	268.52

Tabela A.5: Comparação dos resultados entre os métodos puro e híbrido para o MKP

m	n	α	% média puro	% melhor puro	% média híbrido	% melhor híbrido
5	100	0.25	6.10	2.71	0.76	0.15
10			4.64	3.14	0.73	0.03
30			5.46	2.96	1.20	0.54
5	250		7.60	5.50	0.85	0.46
10			7.95	5.35	1.44	1.00
30			9.28	5.65	2.08	1.65
5	500		8.72	6.19	1.04	0.65
10			7.17	5.97	1.54	1.17
30			7.52	5.71	1.65	1.36
5	100	0.50	3.57	2.03	0.58	0.12
10			3.00	1.28	0.61	0.21
30			3.37	1.25	0.85	0.40
5	250		4.75	3.59	0.68	0.43
10			3.04	1.93	0.93	0.44
30			4.17	3.50	0.98	0.67
5	500		5.11	4.24	0.61	0.46
10			3.79	3.31	0.91	0.69
30			3.05	2.37	0.86	0.55
5	100	0.75	1.78	1.17	0.32	0.04
10			1.77	0.97	0.29	0.00
30			1.69	0.83	0.27	0.00
5	250		2.01	1.46	0.43	0.18
10			1.35	0.85	0.41	0.18
30			1.35	0.76	0.45	0.20
5	500		2.43	1.97	0.38	0.22
10			1.61	1.26	0.44	0.28
30			1.27	1.12	0.46	0.28

Tabela A.6: Comparação dos resultados com heurísticas

m	n	α	melhor híbrido	B&B	B&B híbrido
5	100	0.25	24343	24373	24381
10			23055	23064	23064
30			**21826**	21516	21946
5	250		59038	59243	59312
10			58594	59071	59164
30			55901	56277	56796
5	500		119365	120082	120148
10			116439	117632	117741
30			114468	115154	115820
5	100	0.75	59798	59960	59965
10			57375	60633	60633
30			57494	60574	60603
5	250		149393	154654	154668
10			**151531**	149641	149704
30			**149855**	149514	149595
5	500		295161	299904	299904
10			303534	306949	307027
30			**300828**	300309	300387

Tabela A.7: Resultados da aplicação do método híbrido a outras instâncias do MKP

m	n	α	média	melhor	pior	ótimo	% média	% melhor
10	100	0.25	22405.50	22565	22167	22801	1.73	1.03
	250		57715.20	58110	57342	58781	1.81	1.14
	500		117466.75	117627	117352	119249	1.49	1.36
10	100	0.50	41992.50	42159	41758	42344	0.83	0.43
	250		107854.00	108179	107574	108717	0.79	0.49
	500		216945.50	217544	216379	219077	0.97	0.69
10	100	0.75	58765.10	58899	58654	58978	0.36	0.13
	250		148000.50	148406	147753	148772	0.51	0.24
	500		300737.25	300995	300282	302379	0.54	0.45
10	100	0.25	21820.00	21948	21693	22131	1.40	0.82
	250		57243.20	57628	56828	58097	1.46	0.80
	500		117533.00	117948	117274	119215	1.41	1.06
10	100	0.50	42099.10	42263	41769	42401	0.71	0.32
	250		107921.60	108091	107358	108932	0.92	0.77
	500		215351.25	216061	214604	217847	1.14	0.81
10	100	0.75	58048.30	58265	57932	58391	0.58	0.21
	250		151169.00	151487	150854	151909	0.48	0.27
	500		300996.50	301340	300697	302417	0.46	0.35

Tabela A.8: Resultados do método aplicado ao SAT

instância	n	m	% sucesso puro	% sucesso ag	% sucesso híbrido
`uf20`	20	91	100	100	100
`uf50`	50	218	100	37	100
`uf75`	75	325	100	25	100
`uf100`	100	430	100	13	100
`uf150`	150	645	85	2	100
`uf200`	200	860	75	0	90
`uf225`	225	960	55	0	85
`uf250`	250	1065	40	0	75

Tabela A.9: Comparação dos resultados do método aplicado ao SAT

instância	n	m	híbrido	HH	GSAT+Inc	WalkSat+Inc
uf20	20	91	100	100	100	100
uf50	50	218	100	74.3	93.5	100
uf75	75	325	100	-	78	100
uf100	100	430	100	-	72.3	100
uf150	150	645	100	-	-	98.7
uf200	200	860	90	-	-	93.6
uf225	225	960	85	-	-	91
uf250	250	1065	75	-	-	87.5