Universidade Federal do Amazonas

Faculdade de Tecnologia

Programa de Pós-Graduação em Engenharia Elétrica

Um novo método de otimização baseado em teorias de satisfatibilidade

Rodrigo Farias Araújo

Manaus – Amazonas

Março de 2017

Rodrigo Farias Araújo

Um novo método de otimização baseado em teorias de satisfatibilidade

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica na área de concentração: Automação e Controle de Sistemas.

Orientador: João Edgar Chaves Filho

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Araujo, Rodrigo Farias
A663u Um novo método de otimização baseado em teorias de satisfatibilidade / Rodrigo Farias Araujo. 2017
82 f.: il. color; 31 cm.

Orientador: João Edgar Chaves Filho
Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Amazonas.

1. otimização. 2. satisfatibilidade. 3. teoria do módulo de satisfatibilidade. 4. planejamento de caminho. I. Chaves Filho, João Edgar II. Universidade Federal do Amazonas III. Título

RODRIGO FARIAS ARAÚJO

UM NOVO MÉTODO DE OTIMIZAÇÃO BASEADO EM TEORIAS DE SATISFATIBILIDADE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica na área de concentração Controle e Automação de Sistemas.

Aprovado em 30 de março de 2017.

BANCA EXAMINADORA

Prof. Dr. João Edgar Chaves Filho, Presidente

Universidade Federal do Amazonas- UFAM

Prof. Dr. Vicente Ferreira de Lucena Junior, Membro

Universidade Federal do Amazonas- UFAM

Profª. Dra. Fabíola Guerra Nakamura, Membro

Universidade Federal do Amazonas- UFAM

À minha mãe.

Agradecimentos

Agradeço primeiramente à minha mãe e meu pai, Raimunda e Reginaldo, por me apoiarem e guiarem desde sempre. Agradeço a minha namorada Adria por sempre me incentivar e apoiar durante o mestrado.

Agradeço aos amigos e colegas da UEA e UFAM que acumulei ao longo da graduação e do mestrado, em especial ao Iury Bessa pela ajuda durante o mestrado.

Agradeço ao Prof. João Edgar por todos os ensinamentos, conselhos, orientações e suporte ao longo do mestrado. Ao Prof. Lucas Cordeiro pelo suporte e conselhos.

Agradeço finalmente a toda a equipe de professores PPGEE/UFAM que ajudaram a sedimentar essa formação.

One can achieve a great success when we stay
true to ourselves.

Friedrich Nietzche

Resumo

Este trabalho apresenta um novo método de otimização aplicado a diferentes classes de problemas, como não-convexos e convexos. A metodologia consiste na utilização do contra-exemplo gerado a partir da técnica de verificação de modelos, baseada na teoria de satisfatibilidade booleana (SAT) ou na teoria do módulo de satisfatibilidade (SMT), para guiar o processo de otimização. São desenvolvidos três algoritmos de otimização, são eles: Algoritmo Genérico, aplicado a qualquer classe de problema de otimização, neste será utilizado na otimização de funções não-convexas, Algoritmo Simplificado, empregado na otimização de funções nas quais tem-se algum conhecimento prévio, por exemplo, funções semi-definidas ou definidas positivas e Algoritmo Rápido, utilizado para otimização de funções convexas. Adicionalmente, são fornecidas as provas de convergência para os respectivos algoritmos. Os algoritmos são implementados utilizando dois verificadores de modelos, o CBMC que utiliza como *back-end* o solucionador MiniSAT baseado em SAT, e o ESBMC, que tem suporte aos solucionadores baseados em SMT, como: Z3, Boolector e MathSAT. Para avaliação de desempenho, os algoritmos são aplicados a um conjunto de trinta funções retiradas da literatura e utilizadas para teste de algoritmos de otimização, os mesmos também são comparados com algoritmos de otimização tradicionais usualmente empregados na resolução de problemas de otimização não-convexa, como: algoritmo genético, enxame de partícula, busca de padrões, recozimento simulado e programação não-linear. Através da análise dos resultados pode-se concluir que os algoritmos desenvolvidos são adequados as classes de funções para os quais foram desenvolvidos e possuem maior taxa de acerto na busca pelo valor ótimo em comparação com os outros algoritmos. Finalmente a metodologia desenvolvida é aplicada para resolver problemas de otimização no contexto de planejamento de caminhos bidimensionais para robô móveis autônomos.

Palavras-chave: otimização, satisfatibilidade, teoria do modulo de satisfatibilidade, planejamento de caminho.

Abstract

This work presents a new method of optimization applied to different classes of problems, such as non-convex and convex. The methodology consists in the use the counterexample generated from the model checking technique based on Boolean satisfiability theory (SAT) and satisfiability modulo theory (SMT), to guide the optimization process. Three algorithms of optimization are developed: Generic Algorithm, applied to any class of optimization problem, it will be used in the optimization of non-convex functions, Simplified Algorithm, used in the optimization of functions in which there is some previous knowledge, *e. g.*, semi-defined or defined positive functions and Fast Algorithm, used to optimize convex functions. In addition, convergence proofs are provided for the respective algorithms. The algorithms are implemented using two model verifiers, CBMC which uses the SAT-based MiniSAT solver as back-end, and the ESBMC, which supports SMT-based solvers, such as Z3, Boolector and MathSAT. For perfomance evaluation, the algorithms are applied to a set of thirty functions taken from the literature and used to test optimization algorithms, they are also compared with traditional optimization algorithms usually used in solving non-convex optimization problems, such as genetic algorithm, particle swarm, pattern search, simulated annealing and nonlinear programming. Through the analysis of the results it can be concluded that the developed algorithms are suitable the classes of functions for which they were developed and have a higher rate of success in the search for the optimal value in comparison with the other algorithms. Finally, the developed methodology is applied to solve optimization problems in the context of the two-dimensional path planning for autonomous mobile robots.

Keywords: optimization, satisfiability, satisfiability modulo theory, path planning.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Abreviações

BMC - *Verificação de modelos limitada - do inglês **B**ounded **M**odel **C**hecking*

CBMC - *Verificador de modelos limitado para C - do inglês **C** **B**ounded **M**odel **C**hecker*

CFO - *Otimização de força central - do inglês **C**entral **F**orce **O**ptimization*

DS - *Busca diferencial - do inglês **D**iferential **S**earch*

ESBMC - *Verificador eficiente de modelos limitado baseado em teoria do modulo de satisfabilidade - do inglês **E**fficient **S**MT-based Context-**B**ounded **M**odel **C**hecker*

FPGA - *Arranjo de portas programável em campo - do inglês **F**ield **P**rogrammable **G**ate **A**rray*

ILP - *Programação linear inteira - do inglês **I**nteger **L**inear **P**rogramming*

LEA - *Algoritmo de lista expandida - do inglês **L**ist **E**xpanding **A**lgorithm*

LP - *Programação Linear - do inglês **L**inear **P**rogramming*

NLP - *Programação Linear Não-Linear - do inglês Non-Linear **P**rogramming*

SAT - *Satisfatibilidade proposicional - do inglês Propositional **SAT**isfiability*

SMT - *Teoria do módulo de satisfatibilidade - do inglês **S**atisfability **M**odulo **T**heory*

UAV - *Veículo aéreo não tripulado - do inglês **U**nmanned **A**erial **V**ehicles*

UGV - *Veículo terrestre não tripulado - do inglês **U**nmanned **G**round **V**ehicles*

UUV - *Veículo aquático não tripulado - do inglês **U**nmanned **U**nderwater **V**ehicles*

VC - *Condição de verificação - do inglês **V**erification **C**ondition*

Capítulo 1

Introdução

Otimização é um importante tópico de pesquisa em diversas áreas, especialmente em ciência da computação e engenharia [1]. Normalmente, cientistas e engenheiros precisam encontrar parâmetros ótimos para diferentes tipos de problemas, ou seja, uma solução ótima, que minimiza o comportamento de um dado sistema ou o valor de uma determina função. Otimização é a ferramenta que distingue o olhar da engenharia sobre um problema, por esta razão, estudo anteriores mostram que o processo de otimização é uma das principais diferenças entre um projeto de engenharia e um projeto tecnológico [2].

A ciência da computação e otimização mantem uma relação estreita e caminham lado a lado em seus desenvolvimentos. Muitos avanços importantes da ciência da computação são baseadas na teoria de otimização, por exemplo, problemas de planejamento e decisão (teoria dos jogos [3]), problemas de alocação de recursos (co-projeto de *hardware/software* [4]), e aproximação e estimação computacional (análise numérica [5]) representam importantes aplicações de otimização. Por outro lado, a ciência da computação desempenha uma papel importante em estudos de otimização recentes, desenvolvendo algoritmos eficientes e fornecendo as respectivas ferramentas para suportar o gerenciamento de modelos e análise de resultados [6].

Existem várias técnicas de otimização descritas na literatura, por exemplo, simplex [7], gradiente descendente [8], e algoritmo genético [9], que são adequadas para diferentes classes de problemas de otimização, como: linear ou não-linear, contínuo ou discreto, convexo ou não-convexo, simples ou multiobjetivo. Estas técnicas são usualmente divididas em dois grandes grupos: otimização determinística ou otimização estocástica. Otimização determinística é uma abordagem clássica para algoritmos de otimização que se baseiam em cálculos e operadores

algébricos, como gradientes e Hessianas, já otimização estocástica emprega aletoriedada em seus procedimentos de busca pela solução ótima [10].

Particularmente, um problema de otimização não-convexo é um dos problemas mais complexos e desafiadores, visto que possuem diversas soluções ótimas locais. Devido a isso, vários métodos tracionais, como Newton-Raphson [1] e Gradiente Descendente [8] são ineficientes para resolver esta classe particular de problemas. Muitas heurísticas foram desenvolvidas como alternativa a esses problemas, como colônia de formigas [11] e algoritmo genético [9] que oferecem soluções mais rápidas para estes problemas, mas sacrificam a exatidão da solução, além de frequentemente ficarem presos em soluções que são mínimos locais.

Este trabalho apresenta um novo algoritmo de otimização baseado em busca, que emprega representação não-determínistica das variáveis de decisão para garantir otimização global. Esta classe de técnicas é definida neste como *otimização não-determinística*.

O método apresentado é uma técnica de otimização não-determinística baseada em satisfatibilidade proposicional e em teorias do módulo de satisfatibilidade, adequada a uma larga variedade de problemas de otimização, até mesmo não-lineares e não-convexos. A avaliação da função de custo e a busca pela solução ótima é alcançada pela execução recursiva de sucessivas verificações baseadas em proposições SAT/SMT. Em comparação com outros métodos heurísticos, que são normalmente utilizados para otimização destas classes de funções, a abordagem apresentada sempre encontra a solução ótima global.

Este trabalho também extende o apresentado por Araújo *et al.* [12] e apresenta três novos algoritmos de otimização não-determinística baseada em SAT/SMT, que melhoram o desempenho para algumas classes de funções especiais.

1.1 Trabalhos Relacionados

Solucionadores SMT têm sidos aplicados para resolver diversos tipos de problemas [13], eles são usualmente empregados para verificar a satisfatibilidade de uma fórmula lógica, retornando condições que fazem a fórmula verdadeira, se esta é satisfatível. Uma importante e recente aplicação de SMT é em otimização. Nieuwenhuis e Oliveras [14] apresentam a primeira pesquisa de aplicação de SMT na resolução de problemas de otimização. Desde então, solucionadores SMT foram usados para resolver os mais diferentes tipos de problemas de otimização, por exemplo, minimizar erros de aritmética linear de ponto-fixo em *software* de controle em-

barcado [15], minimizar o número de portas lógicas em circuitos digitais, otimizando o projeto de FPGA [16], particionamento *hardware/software* em sistemas embarcados para decidir qual a implementação mais eficiente [17–19], gerenciamento de aplicação em plataforma com multi-processador [20]. Todos esses estudos usam otimização baseada em SMT aplicada ao domínio booleano para encontrar a melhor configuração para o sistema dado um conjunto de métricas. Em particular, neste último [20] o problema é formulado como um problema otimização multi-objetivo.

Houve também um grande avanço no desenvolvimento de diferentes solucionadores especializados, como o `ABsolver` [21], que é usado para análise e verificação automática de sistemas híbridos e sistemas de controle. O `ABsolver` é uma ferramenta de otimização não-linear que aborda uma combinação de problemas com restrições booleanas e polinomiais. Similarmente, o `CalCs` [22] é também um solucionador SMT que combina restrições não-lineares convexas e booleana. Um outro solucionador que pode manipular vários tipos de funções reais não-lineares, como polinomiais, trigonométricas e exponenciais, é o `dReal` [23].

Recentemente, νZ [24] extendeu o solucionador SMT $Z3$ para resolver problemas de otimização linear, Yi li *et al.* apresentaram o algoritmo `SYMBA` [25], que é um algoritmo de otimização simbólica baseado em SMT e usa a teoria de aritmética real linear usando solucionadores SMT como $back-end$. Sebastiani e Trentin também desenvolveram o `OptiMathSAT` [26], uma ferramenta de otimização que extende o solucionador MathSAT5 SMT para permitir resolver problemas de otimização no domínio booleano, racional, inteiro ou uma combinação entre eles. O potencial das ferramentas de otimização levou ao desenvolvimento de soluções para otimização de funções sob o domínio dos números reais, que fora inicialmente proposta por Sebastiani *e*t al. [27, 28]. Sebastiani e Tomasi [28] usaram a combinação de técnicas SMT e LP para minimizar funções racionais, os autores estenderam seu trabalho com aritmética linear para o domínio racional/inteiro, combinando técnicas SMT, LP e ILP [27].

Shoukry *et al.* [29] apresentaram a prosposta de uma solução escalável para síntese de controloadores digitais e planejamento de movimento para robôs sub-atuados. Sua solução é mais flexível e permite resolver uma larga variedade de problemas, mas eles também são restritos a problemas que devem ser divididos em um parte booleana e outra convexa. Um outro exemplo de aplicação, Pavlinovic *et al.* [30] propõem uma abordagem que consideram todos os possíveis fontes de erros de tipos de varíaveis. Os autores formulam a abordagem como um problema de otimização baseado em SMT e usam o νZ para computar uma fonte de erro ótima

em um determinado programa mal-condicionado com tipos.

A maioria dos estudos anteriores relacionados com otimização baseada em SMT podem solucionar apenas problemas lineares sob o domínio inteiro, racional ou booleano, em casos específicos. Apenas poucos trabalhos são capazes de tratar problemas não-lineares [21, 29], mas também possuem restrições a funções convexas, acarretando em limitações na aplicação em engenharia. Em contrapartida, este trabalho propõe um novo método de otimização baseado em SAT/SMT para minimizar funções de custo lineares ou não-lineares, convexas ou não-convexas, contínuas ou descontínuas, com precisão ajustável das soluções. Como resultado, o método é capaz de solucionar problemas diretamente no domínio real sem o uso de qualquer outra técnica para auxiliar na manipulação do espaço de estados. Além disso, a proposta utiliza uma ferramenta de verificação de modelos como *front-end* para gerar automaticamente fórmulas SMT e SAT a partir de códigos ANSI-C, o que facilita e padroniza a representação de problemas de otimização baseado em satisfatibilidade.

1.2 Objetivos

O objetivo principal deste trabalho é utilizar teorias de satisfatibilidade como satisfatibilidade boolena (SAT) e teoria do módulo de satisfatibilidade (SMT) para resolver problemas de otimização com funções de custo não-convexas e convexas.

Esse objetivo deverá ser alcançado a partir dos seguintes objetivos específicos:

1. Desenvolver os algoritmos para resolver problemas de otimização sob o domínio real;
2. Provar matematicamente a convergência dos algoritmos;
3. Validar os algoritmos desenvolvidos em um *bechmark* de funções utilizadas para avaliação de algoritmos de otimização;
4. Utilizar diferentes verificadores de modelo e solucionadores SAT/SMT para verificação da eficácia e desempenho quanto ao tempo de otimização dos mesmos;
5. Demonstrar a viabilidade do algoritmo para geração de caminhos ótimos bidimensionais para robôs móveis presentes em ambientes com obstáculos estáticos e conhecidos.

1.3 Contribuições

As principais contribuições deste trabalho são:

- **Novos algoritmos de otimização não-determinística.** Este trabalho descreve a aplicação da teoria do satisfatibilidade para resolver o problemas de otimização de funções não-convexas e convexas. Ele apresenta três novos algoritmos de otimização baseado em SAT/SMT, chamados: algoritmo generalizado, algoritmo simplificado, algoritmo rápido. O algoritmo generalizado é adequada a qualquer classe de funções. O algoritmo simplificado é empregado a funções onde se tem alguma informação a respeito, como funções semi-definidas positivas. E o algoritmo rápido é uma alternativa aplicável a funções convexas.

- **Provas de Convergência.** São apresentadas as provas de convergência dos algoritmos propostos, omitidas em Araújo *et al.* [12].

- **Comparação do desempenho dos solucionadores SAT e SMT.** Os experimentos são avaliados com três diferentes solucionadores SMT: Z3 [31], Boolector [32], and MathSAT [33], Araújo *et al.* [12] empregou apenas Boolector para avaliação experimental. Além de um solucionador SAT, MiniSAT [34] para comparação entre satisfatibilidade boolena e teoria do módulo de satisfatibilidade. Os experimentos mostram que o solucionador escolhido pode influenciar consideravelmente na performance do método.

- **Benchmarks.** O *benchmark* de 4 funções Araújo *et al.* [12] é expandido para 30 funções extraídas da literatura [35].

- **Comparação com várias técnicas tradicionais.** A técnica proposta é comparada com algoritmo genético [9], enxame de partícula [36], busca de padrões [37], recozimento simulado [38], progamação não-linear [39], que são técnicas tradicionais de otimização empregadas em problemas de otimização de funções não-convexas.

- **Aplicação.** A metodologia proposta é aplicada na resolução de um problema de geração de trajetória bidimensional para robôs móveis.

1.4 Organização do trabalho

O restante desse trabalho é organizado da seguinte forma:

- **Capítulo 2**: é abordado o arcobouço teórico necessário para desenvolvimento da metodologia: otimização, teoria do módulo de satisfatibilidade, verificação de modelos e os verificadores de software CBMC e ESBMC;
- **Capítulo 3**: apresenta as etapas de desenvolvimento da metodologia proposta para o resolução de problemas de otimização de funções convexas e não-convexas utilizando a abordagem baseada em satisfatibilidade.
- **Capítulo 4**: apresenta os resultados experimentais da aplicação dos algoritmos desenvolvidos ao *benchmark* de funções, bem como a comparação do desempenho dos solucionadores SAT e SMT, assim como a comparação com diferentes técnicas de otimização.
- **Capítulo 5**: mostra a viabilidade da aplicação da metodologia proposta no Capítulo 3 no contexto de geração de caminhos bidimensionais para robôs móveis.
- **Capítulo 6**: apresenta as conclusões do trabalho e propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste Capítulo serão apresentados alguns conceitos fundamentais empregados no desenvolvimento da metodologia de otimização baseada em satisfatibilidade. Na Seção 2.1 é apresentado o conceito de otimização e alguns métodos disponíveis para solucionar problemas de otimização. A Seção 2.2 descreve o conceito de satisfatibilidade e as teorias empregadas na resolução desse problema, satisfatibilidade proposicional e teoria do módulo de satisfatibilidade, e também a técnica de verificação de modelos e verificação de modelos limitada baseada em satisfatibilidade. Finalmente, a Seção 2.3 descreve brevemente os verificadores de modelo: CBMC e ESBMC, a utilização deste último em diferentes aplicações.

2.1 Otimização

De acordo com Rao [40], otimização é a ação de obter o melhor resultado sob determinadas circunstâncias. No processo de projeto, construção e manutenção de qualquer sistema de engenharia, devem ser tomadas muitas decisões em vários estágios, com o objetivo final de minimizar o esforço requerido ou maximizar um benefício desejado.

Tais características, tanto esforço requerido quanto benefício desejado, podem ser expressas por funções de variáveis de decisão. Assim, otimização pode ser definida como o processo de encontrar certas condições, isto é, uma solução que otimize o comportamento de um sistema ou valor de uma dada função.

Existem vários métodos disponíveis na literatura para resolver problemas de otimização, Rao [40] lista e classifica alguns conforme a Tabela 2.1, porém a classificação não é única.

Programação Matemática	**Técnicas de Processos Estocásticos**	**Métodos Estatísticos**	**Métodos Heurísticos**
Métodos de cálculos	Processos de Markov	Análise de Regressão	Algoritmo Genético
Programação Linear	Teoria Renewal	Reconhecimento de Padrão	Recozimento Simulado
Programação não-linear	Métodos de Simulação		Enxame de Partículas
Métodos Geométricos	Teoria de Factibilidade		Lógica Fuzzy
Programação Inteira			

Tabela 2.1: Métodos de otimização.

As técnicas de programação matemática, por exemplo, simplex, gradiente descendente, Newton-Raphson, se utilizam de propriedades matemáticas para convergirem para solução ótima, no entanto, são ineficientes para classes de funções não convexas ou descontínuas.

As técnicas baseadas em processos estocásticos podem ser usadas para analisar problemas descritos por um conjunto de variáveis aleatórias, caso se conheça suas distribuições de probabilidade. Métodos estatísticos permitem analisar dados experimentais e construir modelos empíricos para obter maior precisão na representação de uma condição física.

Os métodos heurísticos de otimização, como algoritmo genético, colônia de formigas, enxame de partículas, são baseados em certas características de comportamento biológico, molecular, enxame de insetos e sistemas neurobiológicos. Tais metodologias oferecem soluções para problemas complexos com maior rapidez, entretanto, sacrificam a corretude da solução e podem ficar presos em soluções subótimas (mínimos ou máximos locais).

Um problema de otimização pode ser definido como:

Dada uma função de custo, $f : X \to \mathbb{R}$, tal que $X \subset \mathbb{R}^n$ é o vetor de variáveis de decisão $X = [x_1, x_2, ..., x_n]$ e $f(x_1, x_2, ..., x_n) \equiv f(\mathbf{x})$. Dado o subconjunto *Omega* um conjunto de restrições sob as variáveis de decisão, tal que $X \subset \Omega$. Deseja encontra o vetor de variáveis de decisão que minimiza a função de custo.

Definição 2.1. *Um problema de otimização multivariável consiste em encontrar o vetor ótimo* $\boldsymbol{x}^*$*, que minimize* f *em* Ω*.*

De acordo com a Definição 2.1, este tipo de problema de otimização pode ser escrito como:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.a.} \quad & \mathbf{x} \in \Omega. \end{aligned} \tag{2.1}$$

Este problema de otimização pode ser classificado de diferentes modos, de acordo com a função de custo f, por exemplo, linear ou não-linear, convexo ou não-convexo, contínuo ou descontínuo, discreto, inteiro, racional, booleano ou real. Dependendo da classificação, os métodos citados anteriormente podem ser menos eficientes, e alguns podem gerar soluções subótimas, ou seja, a solução não é o mínimo global de f, é apenas um mínimo local. Uma solução ótima global da função f pode se definida de acordo com Definição 2.2.

Definição 2.2. *Um vetor $\boldsymbol{x}^* \in \Omega$ é uma solução ótima global de f em Ω, se e somente se $f(\boldsymbol{x}^*) \leq f(x), \forall \boldsymbol{x} \in \Omega$.*

2.2 Satisfatibilidade

Uma lógica proposicional é definida por uma relação binária, na qual se presume que toda sentença deve ser verdadeira ou falsa. O problema de satisfatibilidade proposicional ou booleana (SAT) é um dos problemas fundamentais da ciência da computação. Foi o primeiro problema identificado como pertencente a classe de complexidade NP-completo, e consiste no problema de determinar se existe um valor para uma proposição lógica ϕ, tal que este valor satisfaça a proposição em questão, ou seja, o objetivo é decidir se uma fórmula sobre variáveis booleanas pode ser verdade utilizando conectivos lógicos, escolhendo-se os valores de suas variáveis proposicionais [41].

Assim, um solucionador SAT [34] é um algoritmo que toma como entrada uma fórmula ϕ e decide se ela é satisfatível ou insatisfatível. A fórmula ϕ é dita satisfatível se o solucionador é capaz de encontrar um condição que a torne verdadeira, neste caso o solucionador fornece atribuição às variáveis proposicionais que satisfazem a fórmula ϕ, e é dita insatisfatível se nenhuma condição a torna verdadeira.

Problemas de satisfação surgem em diversas áreas, incluindo verificação de software e hardware, geração de casos de teste, programação, planejamento, entre outros [42].

2.2.1 Teoria do Módulo de Satisfatibilidade

Em diversos tipos de problema é necessário verificar a satisfatibilidade de fórmulas considerando uma determinada teoria ou combinação de teorias. Isto constitui a teoria do módulo de satisfatibilidade (SMT), a qual verifica a satisfatibilidade de fórmulas lógicas de primeira ordem a partir de uma ou mais teorias de suporte. De modo formal, é um conjunto de sentenças sob uma assinatura $\sum$. Dada uma teoria T, dizemos que ψ é módulo satisfatível de T se $T \cup \{\psi\}$ é satisfatível. Algumas das teorias suportadas por SMT são: aritmética linear, aritmética de diferenças, aritmética não-linear, funções livres, vetores de bit, *tuples*, vetores, tipos de dados indutivos e outras teorias de primeira ordem decidíveis [41].

A fim de verificar a satisfatibilidade de uma fórmula, os solucionadores SMT [31–33] manipulam os termos em uma dada teoria de suporte, usando um procedimento de decisão. Os solucionadores SMT são construídos sobre solucionadores SAT, para melhorar o desempenho e o suporte a diferentes teorias de decisão.

2.2.2 Verificação de Modelos

De acordo com Clarke Jr. et al. [43], verificação de modelos (*Model Checking*) é uma técnica automática para verificação de propriedades de sistemas de estados finitos.

Uma das vantagens da verificação de modelos é que a verificação pode ser realizada de forma automática. O procedimento utiliza uma busca no espaço de estado do sistema para determinar se alguma propriedade é verdadeira ou falsa. Dada uma pesquisa exaustiva, o procedimento sempre irá terminar com a resposta verdadeira ou falsa para o teste da propriedade. O principal desafio na verificação de modelo é lidar com o problema de explosão no espaço de estados, visto que este pode crescer demasiadamente.

Alguns modos de representação do espaço de estado são descritos em Jhala, R. e Majumbar, R. [44]. Dentre eles podemos citar, enumerativo, no qual estados individuais são representados; simbólico, onde conjuntos de estados são representados utilizando restrições; e utilizando técnicas de abstração, que reduzem o espaço de estados em detrimento da precisão, a restrição para sistemas de espaço finito pode vir a ser a principal desvantagem.

O processo de verificação de modelos consiste em três etapas: modelagem, especificação e verificação [43]. A modelagem é a primeira etapa, e converte o sistema para um formalismo aceito pela ferramenta de verificação de modelos. Utiliza-se linguagens de programação usuais

para tal, como C, entre outras. Muitas vezes, a modelagem pode requerer o uso de uma abstração para eliminar detalhes irrelevantes ou menos importantes. Segundo Baier, C. e Katoen, J.-P. [45], qualquer verificação usando técnicas baseada em modelo é tão boa quanto o modelo do sistema.

Na segunda etapa, se faz a especificação, onde é descrito o comportamento do sistema, bem como a propriedade a ser verificada. Uma importante questão na especificação é a plenitude. A verificação de modelos fornece meios de verificar se uma dada especificação satisfaz uma propriedade do sistema, mas é impossível determinar se tal especificação cobre todas as propriedades que o sistema deveria satisfazer.

E por fim, a etapa de verificação, idealmente, é completamente automática. Nesta fase verifica-se uma dada propriedade é satisfeita, ou seja, todos os estados relevantes do sistema são verificados em busca de algum estado que viola a propriedade verificada. Na prática, frequentemente envolve a assistência humana. Tal atividade consiste na análise dos resultados da verificação. Em caso de resultado negativo para a propriedade verificada, é fornecido o caminho de execução do sistema que o levou a violar a propriedade. Vale ressaltar que, erros também podem ocorrer oriundos da modelagem incorreta do sistema ou da especificação inapropriada, gerando falso negativo.

2.2.3 Verificação de Modelos Limitada

A verificação de modelos limitada (BMC), está entre as mais populares técnicas de verificação de modelo, proposta inicialmente por Biere, A. et al. [46], representa as transições do sistema como fórmulas lógicas booleanas, representação simbólica do espaço de estado e, ao invés de explorar todos os estados exaustivamente, evolui as relações de transição até certo limite e busca simbolicamente por violações de propriedade dentro desse limite. A técnica BMC pode ser baseada em satisfabilidade booleana ou em teorias do módulo de satisfabilidade [47]. BMC baseado em satisfabilidade booleana foi utilizado para diminuir o problema da explosão do espaço de estados [48]. A ideia básica do BMC é verificar a negação de uma determinada propriedade até determinada profundidade.

Definição 2.3. *Dado um sistema de transição M, uma propriedade ϕ e um limite k, BMC desdobra o sistema k vezes e o transforma em uma condição de verificação (VC) ψ de modo*

que ψ é satisfatível se, e somente se, ϕ tem um contraexemplo de profundidade menor ou igual a k [48].

2.3 Verificadores de Modelo - *Model Checker*

Um verificador de modelos é uma ferramenta que é capaz de verificar códigos em busca de erros que podem violar as especificações de operação de um sistema. Várias ferramentas tem sido desenvolvidas nos últimos anos, em especial podemos citar, CBMC [49] e ESBMC [50], os quais serão utilizados neste trabalho para desenvolvimento da metodologia para resolução de problemas de otimização baseado em satisfatibilidade. A ideia básica é tirar vantagem da característica de verificação de programas de ambos e suas capacidades de gerarem contraexemplos, para verificar todas as possíveis soluções factíveis de um problema de otimização, para isso são utilizadas duas diretivas de programação C/C++, ASSUME e ASSERT, que são responsáveis por definir, respectivamente, as restrições do problema de otimização e a propriedade a ser verificada.

2.3.1 CBMC - *C-Bounded Model Checker*

O CBMC é uma ferramenta que implementa BMC em programas ANSI-C utilizando solucionador SAT como o MiniSAT [34]. A ferramenta possui uma biblioteca interna para operações de aritmética em ponto-fixo e ponto-flutuante e é capaz de verificar propriedades como: segurança de ponteiros, limites de arranjos, programas concorrentes e assertivas fornecidas pelo próprio usuário [51].

2.3.2 ESBMC - *Efficient SMT-Based Context-Bounded Model Checker*

Afim de lidar com o aumento da complexidade de software, solucionadores SMT podem ser utilizados como *back-ends* para resolver as condições de verificações geradas.

O ESBMC é um verificador de modelos capaz de verificar programas C/C++ sequenciais e multitarefa utilizando o *front-end* do CBMC. No entanto, no ESBMC utiliza-se solucionadores SMT, como *Z*3 [31], Boolector [32] e MathSAT [33], e tem suporte para verificação de propriedades expressas em lógica temporal linear. O problema é formulado através da construção da seguinte fórmula lógica

$$\psi_k = I(S_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1}) \wedge \overline{\phi(s_1)} \tag{2.2}$$

onde ϕ é uma propriedade e S_0 é um conjunto de estados iniciais de M, $\gamma(s_j, s_{j+1})$ é uma relação de transição de M entre o passo j e $j+1$. Assim, $I(S_0) \wedge \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1})$ representa a execução de uma transição do sistema M de tamanho i. A VC acima ψ pode ser satisfeita se e somente se, para algum $i \leq k$ existe um estado alcançável no passo i em que ϕ é violado. Se a equação (2.2) é satisfatível, isto é, retorna verdadeiro, então o solucionador SMT fornece o chamado contraexemplo.

Definição 2.4. *Um contraexemplo para uma propriedade ϕ é uma sequência de estados s_0, $s_1, \ldots, s_k$ com $s_0 \in S_0$, $s_k \in S_k$, e $\gamma(s_i, s_{i+1})$ para $0 \leq i < k$ que faz a equação (2.2) satisfatível. Se esta não é satisfatível, isto é, retorna falso, então podemos concluir que não há estados de erro em k passos ou menos.*

Em geral, os contraexemplos contêm todos os estados utilizados até a propriedade ser violada. Dessa forma, o usuário pode facilmente reproduzir a execução do sistema até o estado de erro.

Verificadores de modelo baseados em SMT podem ser usados para verificar a corretude de software ANSI-C embarcado [48], verificação de software multitarefa [18, 52, 53], ou ainda verificar modelos de software C++ [54]. Além disso, o ESBMC é utilizado para verificação de filtros e controladores digitais [55–57]. Recentemente, o ESBMC foi utilizado como ferramenta de otimização, na resolução do problema de otimização de particionamento *Hardware/Software* para sistemas embarcados [17].

2.4 Conclusão

Neste Capítulo foram apresentados os fundamentos teóricos necessários para desenvolvimento da nova metodologia de otimização. Foram explanados os conceitos e ferramentas utilizados para otimização e verificação de *software* baseado em teorias de satisfatibilidade. O Capítulo seguinte, utilizará o método de verificação baseado em satisfatibilidade para resolver problemas de otimização de funções não-convexas e convexas.

Capítulo 3

Otimização via Satisfatibilidade

Neste Capítulo será apresentado o algoritmo desenvolvido para resolução de um problema de otimização não-convexo e convexo utilizando satisfatibilidade proposicional e teoria do módulo de satisfatibilidade. A Seção 3.1 detalhará o método de resolução de problemas de otimização de funções não-convexas, nesta é proposto um algoritmo de otimização baseado em satisfatibilidade, chamado Algoritmo Genérico. Na Seção 3.2 é apresentada a prova de convergência do algoritmo, além de exemplificar como o mesmo evita mínimos locais com êxito. Um novo algoritmo, chamado Algoritmo Simplificado, é obtido a partir do anterior, para aplicação em funções custo semi-definidas ou definidas positivas. A Seção 3.3 apresenta um versão melhorada do algoritmo aplicável a otimização convexa, chamado Algoritmo Rápido, também é fornecido a prova de convergência para o mesmo. Finalmente, as conclusões do Capítulo são apresentadas na Seção 3.4.

3.1 Otimização de Problemas Não-Convexos

Muitos problemas práticos são não-convexos, e a maioria dos problemas não-convexos são difíceis de resolver, quando possíveis, com exatidão em um tempo razoável, devido a presença de múltiplas soluções ótimas locais. Métodos matemáticos, como o gradiente descendente [8] são muito ineficientes para essa classe de problemas, visto que evoluem na direção do gradiente e frequetemente são presos por estas soluções. Como alternativa, usa-se algoritmos metaheurísticos, como algoritmo genético [9], no entanto, pode-se ou não encontrar soluções adequadas.

Nesta Seção é proposto um novo método de otimização baseado em satisfatibilidade capaz de minimizar funções, lineares ou não-lineares, convexas ou não-convexas, contínuas ou descontínuas com precisão ajustável diretamente no domínio racional, sem uso de qualquer outra técnica para assistir a manipulação do espaço de estados. A metodologia proposta é eficiente na resolução de problemas não-convexos, uma vez que realiza um busca exaustiva baseada em contraexemplo para guiar o processo de otimização.

Existem duas diretivas de código na linguagem de programação C/C++, que podem ser usadas para modelar e controlar o processo de verificação, são elas: ASSUME e ASSERT. A diretiva ASSUME é responsável por definir as restrições das variáveis não-determinísticas presentes no código, a partir delas o verificador de modelos cria o espaço de estados que será verificado, nesse processo são criados todos os estados admitidos pelas restrições. Já a declaração ASSERT é usada para definir uma propriedade a ser verificada. Quando executada, está poderá reportar sucesso ou falha na afirmação da propriedade especificada. Usando estas declarações, qualquer verificador de modelos por exemplo, ESBMC [50], CBMC [49] e CPAChecker [58], pode ser aplicado para verificar restrições específicas em um problema de otimização, como descrito pela equação (2.1).

Conforme descrito no Capítulo 2, o processo de verificação de modelos consiste em três etapas: modelagem, especificação e verificação. Tais etapas serão seguidas de modo a exemplificar o método de resolução de um problema de otimização de funções não-convexas utilizando o verificador de software ESBMC e solucionadores SMT, no entanto, conforme dito anteriormente qualquer verificador de modelos pode ser empregado assim como solucionadores SAT. O ESBMC suporta também a técnica de verificação de modelos limitada, porém neste trabalho a verificação limitada será implementada pelo próprio algoritmo, o qual irá alterar as restrições definidas pelo ASSUME limitando o espaço de estados a cada execução.

Para ilustrar o método de otimização baseado em satisfatibilidade para problemas de otimização não-convexos foi escolhida a função de Himmelblau. A função de Himmelblau é uma função de duas variáveis positivas com quatro mínimos globais com valor de $f(x_1,x_2) = 0$, onde cada mínimo está localizado em um quadrante do plano (x_1,x_2), definida pela equação (3.1) e mostrada na figura 3.1.

$$f(x_1,x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{3.1}$$

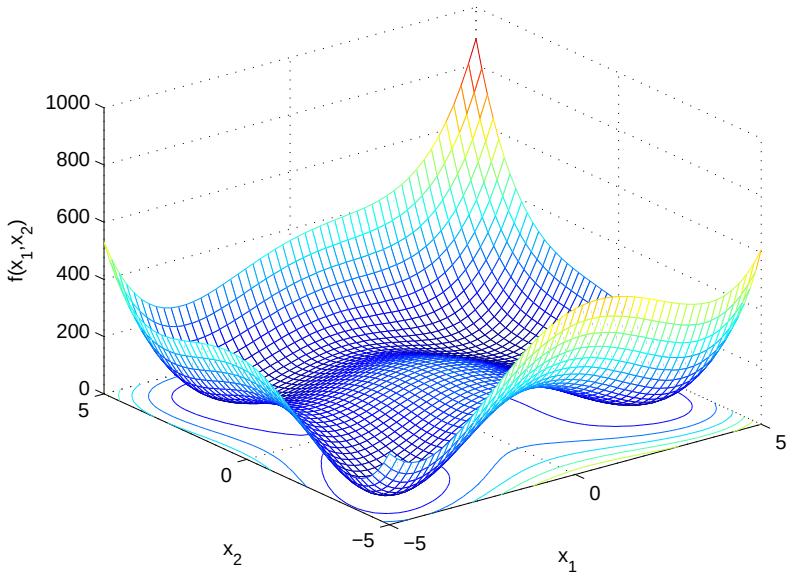


Figura 3.1: Função de Himmelblau.

3.1.1 Modelagem

O processo de modelagem do problema de otimização tem por objetivo descrever a função a ser minimizada, além de definir as restrições que o mesmo pode assumir, isto é, o conjunto Ω. Esta etapa é importante para reduzir o espaço de estados de busca e consequentemente evitar o problema de explosão comentado anteriormente no Capítulo 2. O método proposto não é eficiente para otimização de problemas sem restrições, visto que um problema com poucas restrições pode gerar um espaço de estados muito grande e aumentar demasiadamente o tempo de execução do algoritmo, já um problema com muitas restrições diminui o espaço de estados diminuindo assim o tempo de busca da solução, em contrapartida podemos deixar de fora do espaço de estados o mínimo global e gerar como solução um mínimo local. Note que as restrições devem ser escolhidas baseado no conhecimento prévio sobre o problema ao qual o algoritmo será aplicado.

Para exemplificar o algoritmo, o problema será proposto inicialmente em encontrar o mínimo global da função de Himmelblau situado no 2º quadrante, ou seja encontrar os valores das variáveis x_1 e x_2 que minimize a equação (3.1), dessa forma podemos escrever como o problema de otimização a seguir:

$$\begin{aligned} \min_{x_1,x_2} \quad & f(x_1,x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \\ \text{s.a.} \quad & x_1 \leq 0, \\ & x_2 \geq 0. \end{aligned} \tag{3.2}$$

Apesar de as restrições definirem um ponto no 2º quadrante, estas ainda geram um número infinito de pontos (x_1, x_2) pois x_1 e x_2 podem assumir valores próximos de $-\infty$ e ∞, respectivamente. Para resolver esse problema podemos retirar algumas informações da equação 3.1, por exemplo, para quaisquer valores de $x_1 < -5$ e $x_2 > 5$, a função é crescente sem a possibilidade de se encontrar um mínimo para valores além destes limites. Dessa forma, o problema de otimização anterior pode ser rescrito conforme a equação (3.3).

$$\begin{aligned} \min_{x_1,x_2} \quad & f(x_1,x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \\ \text{s.a.} \quad & -5 \leq x_1 \leq 0, \\ & 0 \leq x_2 \leq 5. \end{aligned} \tag{3.3}$$

A partir da formalização do problema pela equação (3.3), podemos codificar a etapa de modelagem, para isso devemos inicialmente declarar as variáveis a serem definidas como variáveis não-determinísticas, em seguida declarar as restrições ao ESBMC utilizando a diretiva ASSUME, assim o mesmo irá criar o espaço de estados com todas os possíveis valores que x_1 e x_2 podem assumir, e por fim descrever o modelo da função objetivo. A codificação da modelagem do problema de otimização é mostrada no código da figura 3.2.

```
1  float nondet_float();
2  int main() {
3    //definição das variáveis de decisão
4    float x1 = nondet_float();
5    float x2 = nondet_float();
6    //definição do espaço de estados
7    __ESBMC_assume((x1>=-5) && (x1<=0));
8    __ESBMC_assume((x2>=0) && (x2<=5));
9    //cálculo da função de Himmelblau (função objetivo) para cada estado
10   float fobj = (x1^2+x2-11)*(x1^2+x2-11)+(x1+x2^2-7)*(x1+x2^2-7);
11   return 0;
12 }
```

Figura 3.2: Codificação do problema de otimização dado pela equação (3.3).

3.1.2 Especificação

A próxima etapa de metodologia proposta é a especificação, onde o comportamento do sistema e a propriedade a ser verificada são descritos. Para a função de Himmelblau, o resultado desta etapa é mostrado no código da figura 3.3. Podemos perceber no código da figura 3.2 que devido as variáveis de decisão serem declaradas como tipos *float* não-determinísticos, o espaço de estados gerado pelo verificador de modelos será consideravelmente grande, pois este usará a precisão de máquina para geração dos estados, aumentando o tempo de execução do código. Para contornar este problema, a declaração será feita utilizando tipos inteiros não-determinísticos de modo a discretizar e reduzir o espaço de estados. Entretanto, isto também reduz a precisão do processo de otimização.

Para manter o compromisso entre precisão e tempo de verificação, mantendo a convergência para a solução ótima, o procedimento deverá ser executado recursivamente, onde a cada sucessiva iteração a precisão da resposta é aumentada. Uma variável inteira $p = 10^{\eta}$ é criada e ajustada iterativamente, onde η é a quantidade de casas decimais de precisão das variáveis de decisão, ou seja, $\eta \in \mathbb{Z}^{+}$ e se $\eta = 0$, $p = 1$ e apenas valores inteiros para as variáveis de decisão são verificados, se $\eta = 1$ implica que $p = 10$ e agora valores racionais com uma casa decimal para as variáveis de decisão serão verificados, até p atingir a precisão desejada para as variáveis de decisão. Nos experimentos executados neste trabalho, convencionou-se utilizar três casas decimais de precisão para as variáveis de decisão, de modo que $\eta = 3$ e $p = 1000$ na última iteração.

Além disso, uma nova restrição é inserida, de tal forma que seja gerado um novo espaço de estados menor a cada iteração de maneira que o tempo de execução do código não aumente demasiadamente quando a precisão é aumentada, mas ainda assim garantindo que a solução ótima pertença neste novo espaço de estados. A nova restrição é definida sobre o valor da função objetivo, tal que $f(\mathbf{x}^{(i)})$ é o valor da função objetivo na i-ésima iteração e deve ser menor que o valor obtido na iteração anterior, chamada de candidato a mínimo, $f_c = f(\mathbf{x}^{(i-1)})$.

Uma propriedade deve ser especificada para garantir a convergência para o valor ótimo a cada iteração. A propriedade é indicada por meio da diretiva ASSERT, que verifica se a condição l_{otimo} dada na equação (3.4) é satisfeita para todo valor da função.

$$l_{otimo} \iff f(\mathbf{x}) > f_c \tag{3.4}$$

O processo de verificação para, quando $\neg l_{otimo}$ é satisfatível, isto é, se existe algum $\mathbf{x}^{(i)}$ para o qual $f(\mathbf{x}^{(i)}) < f_c$, então um contraexemplo é mostrado com os valores $\mathbf{x}^{(i)}$. Para o exemplo da figura 3.3, $f(\mathbf{x}^{(0)})$ é arbitrariamente inicializado em 100.

```
1  int nondet_int();
2  int main() {
3    //variáveis a serem modificadas a cada execução do código para
         aumentar a precisão
4    int p = 1;
5    float f_i = 100;
6    //limites das variáveis de estados
7    int lim_inf_x1 = -5*p;
8    int lim_sup_x1 = 0*p;
9    int lim_inf_x2 = 0*p;
10   int lim_sup_x2 = 5*p;
11   //definição das variáveis de decisão
12   int x1 = nondet_int();
13   int x2 = nondet_int();
14   //definição do espaço de estados
15   __ESBMC_assume( (x1>=lim_inf_x) && (x1<=lim_sup_x) );
16   __ESBMC_assume( (x2>=lim_inf_y) && (x2<=lim_sup_y) );
17   //cálculo da função de Himmelblau (função objetivo) para cada estado
18   float X1 = (float) x1/p;
19   float X2 = (float) x2/p;
20   float fobj;
21   float fobj = (X1^2+X2-11)*(X1^2+X2-11)+(X1+X2^2-7)*(X1+X2^2-7);
22   //restrição para excluir estados onde fobj>f_(i-1)
23   __ESBMC_assume( fobj < f_i );
24   //f_i: valor candidato a mínimo da função
25   //teste da função objetivo
26   __ESBMC_assert( fobj > f_i,"" );
27   return 0;
28 }
```

Figura 3.3: Código C após etapa de especificação dada pela equação (3.4).

Quando a propriedade verificada é violada, não é garantido que o valor mínimo foi encontrado, porém o contraexemplo gerado pelo verificador informa os valores que a violaram. Assim, essas informações são utilizadas para executar novamente o código 3.3 com valores atualizados das restrições convergindo iterativamente a solução para o valor ótimo, $\mathbf{x}^*$, a cada execução.

3.1.3 Verificação

Por fim temos a etapa de verificação, como dito anteriormente, estamos resolvendo o problema de otimização a partir do código da figura 3.3, logo este deve ser executado recursivamente. Assim a cada término de execução o verificador de modelos irá retornar um contra-

exemplo, o qual deverá ser tratado para obter novos valores de **x**, para o qual o valor da função objetivo é aproximada ao valor ótimo. O código C da figura 3.3 apenas retorna VERDADEIRO como resultado da verificação se o valor candidato a mínimo encontrado na iteração anterior é o valor ótimo para a precisão especificada, definida por p, isto é, $f(\mathbf{x}^{(i-1)}) = f(\mathbf{x}^*)$. Na figura 3.4 é mostrado o contraexemplo da execução do código da figura 3.3.

```
1  Counterexample:
2  State 1 file dissertacao.c line 19 function main thread 0
3  <main invocation>
4  ------------------------------------------------------------
5    c::main::$tmp::return_value_nondet_int$1=0 (0)
6  State 2 file dissertacao.c line 19 function main thread 0
7  <main invocation>
8  ------------------------------------------------------------
9    dissertacao::main::1::x1=0 (0)
10 State 3 file dissertacao.c line 20 function main thread 0
11 <main invocation>
12 ------------------------------------------------------------
13   c::main::$tmp::return_value_nondet_int$2=2 (2)
14 State 4 file dissertacao.c line 20 function main thread 0
15 <main invocation>
16 ------------------------------------------------------------
17   dissertacao::main::1::x2=2 (2)
18 State 5 file dissertacao.c line 33 function main thread 0
19 <main invocation>
20 ------------------------------------------------------------
21   dissertacao::main::1::fobj=0f (0)
22 State 101 file dissertacao.c line 42 function main thread 0
23 <main invocation>
24 ------------------------------------------------------------
25 Violated property:
26   file dissertacao.c line 42 function main
27   assertion
28   fobj > fc
29 VERIFICATION FAILED
```

Figura 3.4: Contraexemplo do código da figura 3.3.

Note que, o estado $x_1 = 0$ e $x_2 = 2$ violou a condição definida pelo ASSERT, já que $f(0,2) = 90$, logo o valor de mínimo é $f_{min} \leq 90$, e a variável f_i no código, denotada por $f(\mathbf{x}^{(i-1)})$, pode ser atualizada para o valor de 90. A variável p que define a precisão dos valores só deverá se atualizada se a execução do código da figura 3.3 retornar VERDADEIRO, isso significa que não existe nenhum outro valor da função de custo menor, para precisão p atual das variáveis de decisão. Este processo se repete após a atualização de p até que a precisão desejada seja alcançada.

3.1.4 Algoritmo de Otimização baseado em Satisfatibilidade

Baseado na metodologia descrita anteriormente, podemos propor um algoritmo baseado em satisfatibilidade para problemas de otimização não-convexos, chamado Algoritmo Genérico 1, nele as etapas de especificação e verificação são repetidas até a solução ótima $\mathbf{x}^*$ ser encontrada. A precisão da solução ótima é definida pela variável de precisão desejada η. Conforme dito anteriormente, para $\eta = 0$ o algoritmo resulta em soluções inteiras. Solução com uma casa decimal é obtida para $\eta = 1$, duas casas decimais são alcançadas para $\eta = 2$, isto é, o número de casas decimais η para a solução calculada pode ser definido pela equação (3.5).

$$\eta = \log p \tag{3.5}$$

Algoritmo 1: Algoritmo de otimização não-convexa baseado em satisfatibilidade: Algoritmo Genérico.

Dados: Uma função de custo $f(\mathbf{x})$, um conjunto de restrições Ω e uma precisão desejada η

Resultado: O vetor de variáveis de decisão ótimo $\mathbf{x}^*$ e o valor ótimo da função de custo $f(\mathbf{x}^*)$

1 *Inicialize* $f(\boldsymbol{x}^{(0)})$ *aleatoriamente*;
2 *Inicialize a variável de precisão com* $p = 1$, $i = 1$ *e* $k = \log p$;
3 *Declare as variáveis de decisão* $\boldsymbol{x}^{(i)}$ *como variáveis inteiras não-determinísticas*;
4 **while** $k \leq \eta$ **do**
5 *Defina os limites de* $\boldsymbol{x}$ *com a diretiva* `ASSUME`*, tal que* $x \in \Omega^k$;
6 *Descreva o modelo para* $f(\boldsymbol{x})$;
7 **do**
8 *Defina a restrição* $f(\boldsymbol{x}^{(i)}) < f(\boldsymbol{x}^{(i-1)})$ *com a diretiva* `ASSUME`;
9 *Verifique a satisfatibilidade de* $\neg l_{otimo}$ *dado pela Eq.* (3.4) *com a diretiva* `ASSERT`;
10 *Atualize* $\boldsymbol{x}^* = \boldsymbol{x}^{(i)}$ *e* $f(\boldsymbol{x}^*) = f(\boldsymbol{x}^{(i)})$ *baseado no contraexemplo*;
11 *Faça* $i = i + 1$;
12 **while** $\neg l_{otimo}$ *é satisfatível*;
13 *Atualize a variável de precisão, p, e consequentemente k*;
14 **end**
15 $\boldsymbol{x}^* = \boldsymbol{x}^{(i)}$ *e* $f(\boldsymbol{x}^*) = f(\boldsymbol{x}^{(i)})$;
16 **return** $\boldsymbol{x}^*$ *e* $f(\boldsymbol{x}^*)$;

Note que o Algoritmo 1 contém dois laços após a inicialização e declaração das variáveis (linhas 1-3). A cada execução do laço externo `while` (linhas 4-14), um novo procedimento de verificação é iniciado após a atualização da precisão p (linha 13), a cada atualização de p uma

casa decimal é adicionada as variáveis de decisão, aumentando a precisão da solução, de modo que a variável auxiliar k, inicialmente é igual a zero e $k = \eta$ ao final da execução do algoritmo.

O laço interno `do-while` (linhas 7-12) corresponde as fases de especificação (linhas 8-9) e verificação (linhas 10-12), onde a função objetivo é verificada para o valor candidato a mínimo obtido na iteração anterior, $f_c = f(x^{(i-1)})$, através da verificação de satisfatibilidade de $\neg l_{otimo}$, se existe algum $f(x) \leq f_c$, retorna-se a fase de especificação. Se $\neg l_{otimo}$ é não satisfatível, o valor $f(x^{(i-1)})$ é mínimo para a precisão p, inicialmente igual a 1. O valor da precisão p é atualizado e o laço externo é repetido. O tempo de execução deste algoritmo depende de como o espaço de estados é restringido e do número de casas decimais desejado para a solução.

3.2 Prova de Convergência

Um problema de otimização genérico descrito na seção anterior é formalizado como: dado um conjunto $\Omega \subset \mathbb{R}^n$, determinar $\mathbf{x}^* \in \Omega$, tal que, $f(\mathbf{x}^*) \in \Phi$ é o menor valor de uma função f, isto é, $\min f(\mathbf{x})$, onde $\Phi \subset \mathbb{R}$ é o conjunto imagem de f, ou seja, $\Phi = Im(f)$. A abordagem soluciona o problema de otimização com η casas decimais, isto é, a solução $\mathbf{x}^*$ é um elemento de um domínio racional $\Omega^\eta \subset \Omega$ tal que $\Omega^\eta = \Omega \cap \Theta$, onde $\Theta = \{\mathbf{x} \in \mathbb{Q}^n | \mathbf{x} = k \times 10^{-\eta}, \forall k \in \mathbb{Z}\}$, ou seja, Ω^η é composto por racionais com η casas decimais de Ω, por exemplo, $\Omega^0 \subset \mathbb{Z}^n$. Então, $\mathbf{x}^{*,\eta}$ é o mínimo de uma função f em Ω^η.

Lema 3.1. *Dado Φ um conjunto finito composto por todos os valores $f(\boldsymbol{x}) < f_c$, onde $f_c \in \Phi$ é um candidato a valor mínimo e $\boldsymbol{x} \in \Omega$. O literal $\neg l_{otimo}$ (Eq. (3.4)) é não satisfatível, se e somente se f_c é o menor valor em Φ. Caso contrário, se $\neg l_{otimo}$ é satisfatível, existe algum $\boldsymbol{x}_i \in \Omega$ tal que $f(\boldsymbol{x}_i) < f_c$.*

Teorema 3.1. *Dado Φ_i o i-ésimo conjunto imagem do problema de otimização restrito por $\Phi_i = \{f(\boldsymbol{x}) < f_c^i\}$, onde $f_c^i = f(\boldsymbol{x}^{(i-1)}), \forall i > 0$, e $\Phi_0 = \Phi$. Existe um $i^* > 0$, tal que $\Phi_{i^*} = \emptyset$, e $f(\boldsymbol{x}^*) = f_c^{i^*}$.*

Demonstração. Inicialmente, o candidato a mínimo é escolhido aleatoriamente de Φ_0. Considerando o Lema 3.1, se $\neg l_{otimo}$ is satisfatível, algum $f(\mathbf{x}^0)$ (baseado no contraexemplo) é adotado como o próximo candidato, isto é, $f_c^1 = f(\mathbf{x}^{(0)})$, e todo elemento de Φ_1 é menor que f_c^1. Similarmente nas próximas iterações, enquanto $\neg l_{otimo}$ é satisfatível, $f_c^i = f(\mathbf{x}^{(i-1)})$, e todo elemento de Φ_i é menor que f_c^i, consequentemente, o número de elementos de Φ_{i+1} será sempre

menor que o número de elementos de Φ_i. Sendo Φ_0 finito, na i^*-ésima iteração, Φ_{i^*} será um conjunto vazio e $\neg l_{otimo}$ é não satisfazível, o que significa (Lema 3.1) $f(\mathbf{x}^*) = f_c^{i*}$. □

O Teorema 3.1 fornece condições suficientes para minimização global sobre um conjunto finito. Este solucionará o problema de otimização definido na seção anterior se e somente se o domínio de busca Ω^η é finito. Este de fato é finito, uma vez que, ele é definido como uma intersecção entre um conjunto limitado (Ω) e um conjunto discreto (Θ). Então, o Algoritmo 1 sempre irá fornecer o valor de mínimo $\mathbf{x}^*$ com η casas decimais.

3.2.1 Evitando de Mínimos Locais

Uma importante característica do algoritmo de otimização proposto é a prova de falha na busca pelo mínimo global. Muitos algoritmos de otimização podem ficar presos em mínimos locais e podem resolver problemas de otimização de maneira sub-ótima. Entretanto, a técnica apresentada assegura o desvio desses mínimos locais, através da verificação sucessiva de questões de satisfatibilidade. Esta propriedade é mantida para qualquer classe de funções e para qualquer valor de inicialização. Para solucionar o problema de aumento no tempo de execução a técnica propõe discretizar o espaço de estados e aumentar a precisão conforme a solução converge, conforme visto na Seção 3.1.

As figuras 3.5 e 3.6 ilustram a propriedade acima mencionada, comparando seu desempenho com outros algoritmos, algoritmo genético (*genetic algorithm* - GA) e gradiente descendente (GD). Nestas figuras, a função de Himmelblau é adaptada, para melhor visualização, para um problema de simples variável, x_1, isto é, x_2 é considerado fixo e igual a 3.131, e a função é reduzida para um plano com mínimo global em $x_1 = -2.805$ e um mínimo local com x_1 positivo.

O resultado a cada iteração são mostrados para os três métodos avaliados. Note que o método de otimização baseado em satisfatibilidade não apresenta uma trajetória contínua do ponto inicial ao ponto final, entretanto sempre alcança o mínimo global. A figura 3.5 mostra que todas as técnicas convergem para o ótimo global, no entanto, a figura 3.6 mostra que o GA e GD podem ficar presos em mínimos locais quando inicializados e diferentes pontos, próximos ao mínimo local. Em contraste, o método proposto pode ser inicializado em qualquer ponto e sempre encontrará o mínimo global para qualquer função de custo.

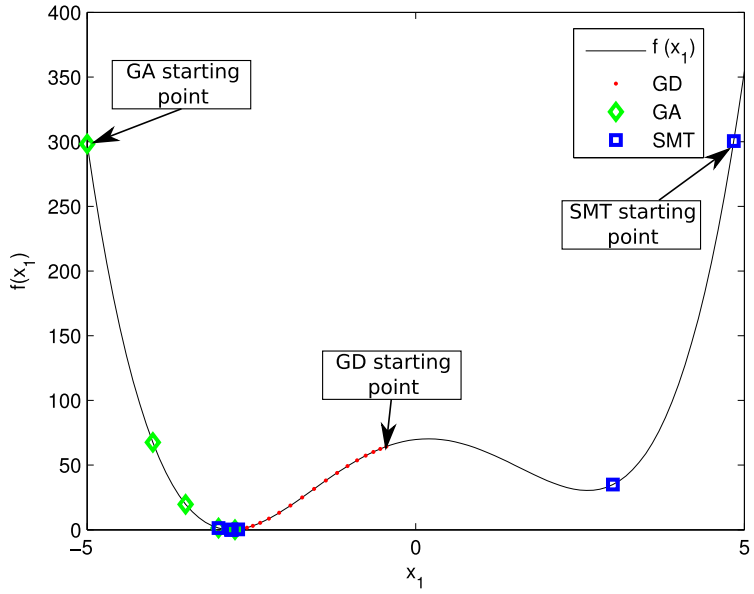


Figura 3.5: Trajetória dos algoritmos de otimização GA, GD, and SMT para o plano da função de Himmelblau em $x_2 = 3.131$. Todos os método obtêm a solução ótima.

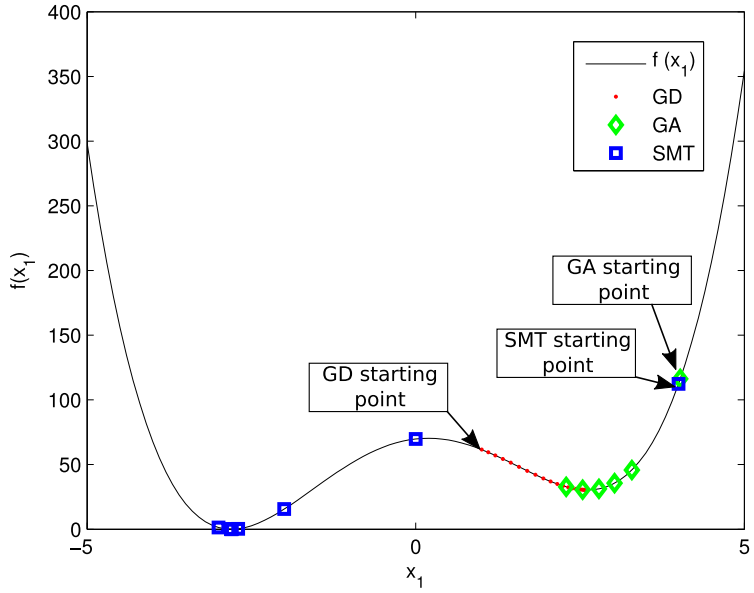


Figura 3.6: Trajetória dos algoritmos de otimização GA, GD, and SMT para o plano da função de Himmelblau em $x_2 = 3.131$. GA e GD ficam presos no mínimo local, mas SMT obtêm a solução ótima.

3.2.2 Algoritmo de Otimização Simplificado

O Algoritmo Genérico 1 é adequado a qualquer classe de função. Entretanto, existem funções em que temos algum conhecimento a priori da mesma. Como exemplo, podemos citar

funções semi-definidas ou definidas positivas, em que $f(\mathbf{x}) \geq 0$ ou $f(\mathbf{x}) > 0$, respectivamente, funções de distância ou energia são exemplos desse tipo de funções. Usando esta característica o algoritmo 1 foi modificado para esta classe de funções. Este novo algoritmo será chamado de Algoritmo Simplificado e é proposto no Algoritmo 2.

Algoritmo 2: Algoritmo de otimização semi-definido e definido positivo baseado em satisfatibilidade: Algoritmo Simplificado.

Dados: Uma função de custo $f(\mathbf{x})$, um conjunto de restrições Ω, uma precisão desejada η and uma taxa de aprendizado α

Resultado: O vetor de variáveis de decisão ótimo $\mathbf{x}^*$ e o valor ótimo da função de custo $f(\mathbf{x}^*)$

```
1  Inicialize f_m = 0;
2  Inicialize f(x^(0)) aleatoriamente;
3  Inicialize a variável de precisão com p = 1, i = 1 e k = log p;
4  Declare as variáveis de decisão x^(i) como variáveis inteiras não-determinísticas;
5  while k ≤ η do
6      Defina os limites de x com a diretiva ASSUME, tal que x ∈ Ω^k;
7      Descreva o modelo para f(x);
8      Defina δ = (f(x^(i−1)) − f_m)/α;
9      if (f(x^(i−1)) − f_m > 0.00001) then
10         do
11             Defina a restrição f(x^(i)) < f(x^(i−1)) com a diretiva ASSUME;
12             while (f_m ≤ f(x^(i−1)) do
13                 Verifique a satisfatibilidade de ¬l_otimo dado pela Eq. (3.4) com a diretiva
                    ASSERT;
14                 Faça f_m = f_m + δ;
15             end
16             Atualize x* = x^(i) e f(x*) = f(x^(i)) baseado no contraexemplo;
17             Faça i = i + 1;
18         while ¬l_otimo é satisfatível;
19     end
20     else
21         break;
22     end
23     Atualize a variável de precisão, p, e consequentemente k;
24 end
25 x* = x^(i) e f(x*) = f(x^(i));
26 return x* e f(x*);
```

Note que o Algoritmo 2 contem três laços após a inicialização e declaração das variáveis (linhas 1-4). A cada execução do laço externo `while` (linhas 5-24) são atualizados os limites e precisão, similar ao algoritmo 1. A principal diferença está na presença da condição na linha 9

que quebra o algoritmo se não é satisfeita, ou seja, não é necessário realizar novas verificações uma vez que já foi atingido o limite mínimo para o valor da função, neste caso $f(\mathbf{x}^*) = 0$.

Há ainda, uma laço adicional `while` (linhas 12-15) interno do laço `do-while` (linhas 10-18). Este é responsável por gerar múltiplas propriedade através da diretiva `ASSERT`, dentro do intervalo limitado por f_m e $f(\mathbf{x}^{(i-1)})$, este laço gera $\alpha + 1$ propriedades a partir do passo definido por δ na linha 8, α é uma nova entrada para o algoritmo e é definido como taxa de aprendizado.

Estas modificações permitem ao Algoritmo Simplificado 2 convergir mais rápido que o Algoritmo Genérico 1 quando aplicado a funções semi-definidas ou definidas positivas, já que a chance de uma verificação falhar é maior devido ao aumento de propriedades. Entretanto, se α é muito elevado, o algoritmo poderá criar muitas propriedades, o que poderia causar o efeito oposto, aumentando o tempo de execução, além de causar em alguns casos *overflow* de memória.

3.3 Otimização de Problemas Convexos

A principal característica de funções não-convexas é a presença de múltiplos mínimos locais, o que torna o problema de otimização sobre tais funções um grande desafio. No entanto, existe outra classe de funções mais simples, as chamadas funções convexas. Essas funções possuem determinadas propriedades que podem facilitar a busca por soluções ótimas. Esta Seção apresenta um algoritmo baseado em satisfatibilidade para problemas de otimização convexos. Adicionalmente, a prova de convergência é apresentada.

Um problema de otimização convexo é similar ao apresentado na equação (2.1), onde $f(\mathbf{x})$ é uma função convexa, isto é, satisfaz a equação (3.6)

$$f(\alpha x_1 + \beta x_2) \leq \alpha f(x_1) + \beta f(x_2) \tag{3.6}$$

para todo $x_i \in \mathbb{R}^n$, com $i = 1, 2$ e todo $\alpha, \beta \in \mathbb{R}$ com $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$.

Adicionalmente, o Teorema 3.2 é um importante teorema na área de otimização convexa. É usado pelo maioria dos algoritmos de otimização convexa e será usado para garantir a convergência do algoritmo de otimização convexa baseado em satisfatibilidade.

Teorema 3.2. *Um mínimo local de uma função convexa f, em um conjunto convexo, é sempre um mínimo global de f. [59].*

3.3.1 Algoritmo de Otimização Convexa

O Algoritmo 1 previamente descrito desenvolve pelo incremento da precisão das variáveis de decisão, isto é, na primeira execução do laço `while`, o mínimo global obtido é inteiro chamado $\mathbf{x}^{*,0}$, pois $p = 1$. O Algoritmo 3 é uma melhora do Algoritmo 1 para aplicação em funções convexas, este é chamado de Algoritmo Rápido.

Note que, a única diferença do Algoritmo 1 é a inserção da linha 14, a qual atualiza Ω^k antes do aumento da precisão. Um novo domínio de busca $\Omega^k \subset \Omega^\eta$ é obtido sob Ω^{k-1}, definindo Ω^k como: $\Omega^k = \Omega^\eta \cap [x^{*,k-1} - p, x^{*,k-1} + p]$, onde $x^{*,k-1}$ é a solução ótima com $k-1$ casas decimais.

Algoritmo 3: Algoritmo de otimização convexa baseado em satisfatibilidade: Algoritmo Rápido.

Dados: Uma função de custo $f(\mathbf{x})$, um conjunto de restrições Ω e uma precisão desejada η

Resultado: O vetor de variáveis de decisão ótimo $\mathbf{x}^*$ e o valor ótimo da função de custo $f(\mathbf{x}^*)$

1 *Inicialize $f(\boldsymbol{x}^{(0)})$ aleatoriamente*;
2 *Inicialize a variável de precisão com $p = 1$, $i = 1$ e $k = \log p$*;
3 *Declare as variáveis de decisão $\boldsymbol{x}^{(i)}$ como variáveis inteiras não-determinísticas*;
4 **while** $k \leq \eta$ **do**
5 *Defina os limites de $\boldsymbol{x}$ com a diretiva `ASSUME`, tal que $x \in \Omega^k$*;
6 *Descreva o modelo para $f(\boldsymbol{x})$*;
7 **do**
8 *Defina a restrição $f(\boldsymbol{x}^{(i)}) < f(\boldsymbol{x}^{(i-1)})$ com a diretiva `ASSUME`*;
9 *Verifique a satisfabilidade de $\neg l_{otimo}$ dado pela Eq.* (3.4) *com a diretiva `ASSERT`*;
10 *Atualize $\boldsymbol{x}^* = \boldsymbol{x}^{(i)}$ e $f(\boldsymbol{x}^*) = f(\boldsymbol{x}^{(i)})$ baseado no contraexemplo*;
11 *Faça $i = i + 1$*;
12 **while** *$\neg l_{otimo}$ é satisfatível*;
13 *Atualize o conjunto Ω^k*;
14 *Atualize a variável de precisão, p, e consequentemente k*;
15 **end**
16 $\boldsymbol{x}^* = \boldsymbol{x}^{(i)}$ *e* $f(\boldsymbol{x}^*) = f(\boldsymbol{x}^{(i)})$;
17 **return** $\boldsymbol{x}^*$ *e* $f(\boldsymbol{x}^*)$;

3.3.2 Prova de Convergência para o Algoritmo de Otimização Convexa

O algoritmo de otimização convexa baseado em satisfatibilidade computa iterativamente Ω^k, para $0 \geq k \leq \eta$. O Teorema 3.1 garante minimização global para um conjunto finito Ω^k. A convergência global é assegurada se e somente se, o mínimo do conjunto Ω^{k-1} está contido também em Ω^k. Isto é naturalmente assegurado pelo Algoritmo 1, pois $\Omega^1 \subset \Omega^2 ... \subset \Omega^{k-1} \subset \Omega^k$. Entretanto, o Algoritmo 3 modifica os limites do conjunto Ω^k, restringindo-o a partir da $k-1$-ésima solução.

Lema 3.2. *Seja $f : \Omega^k \to \mathbb{R}$ uma função convexa, como Ω^k é um conjunto finito, o Teorema 3.1 garante que o mínimo, $x^{*,k}$ em Ω^k é um mínimo local para a precisão p, onde $k = \log p$. Além disso, qualquer elemento $x \in \Omega^{k+1}$ fora do intervalo $[x^{*,k} - p, x^{*,k} + p]$ tem sua imagem $f(x) > f(x^{*,k})$ garantido pela equação* (3.6)*, já que f é convexa.*

O Lema 3.2 garante que a solução é um mínimo local de f, e o Teorema 3.2 assegura que este é mínimo global. De maneira que, os limites de Ω^k podem ser atualizados a cada execução do laço externo `while`, esta modificação reduz consideravelmente o espaço de estados gerado pelo verificador de modelos, o que consequentemente diminuirá o tempo de execução do algoritmo consideravelmente.

3.4 Conclusão

Neste Capítulo foram propostos novos algoritmos de otimização baseado em satisfatibilidade, os três algoritmos apresentados são adequados a várias classes de funções. O Algoritmo Genérico 1 pode ser utilizado a qualquer classe de funções e é apresentado como proposta para otimização de funções não-convexas. O Algoritmo Simplificado 2 é empregado na otimização de funções as quais se tem algum conhecimento prévio, como funções semi-definidas ou definidas positivas. Por fim, o Algoritmo Rápido 3 é proposto para otimização de funções convexas. Adicionalmente as provas de convergência para os respectivos algoritmos foram apresentadas.

Capítulo 4

Avaliação Experimental

Neste Capítulo serão apresentados os resultados da aplicação dos algoritmos propostos no Capítulo 3 aos *benchmarks* de funções utilizados para avaliação dos diferentes algoritmos de otimização. Diferentes solucionadores baseados tanto em SAT quanto SMT serão avaliados, assim como dois verificadores de modelos, CBMC e ESBMC. A Seção 4.1 descreve a configuração do computador utilizado para os testes e os *bechmarks* de funções avaliados. As Seções 4.2, 4.3 e 4.4 discutem o resultados para os grupos de funções de custo não-convexas, semi-definidas ou definidas positivas e convexas. Na Seção 4.5 a abordagem desenvolvida é comparada com outros métodos de otimização existentes. Finalmente, a Seção 4.6 apresenta as conclusões do testes efetuados.

4.1 Configuração e Bechmarks

Neste Capítulo foram realizados diversos experimentos com seguintes objetivos:

- Avaliar o desempenho dos algoritmos propostos na busca pela solução ótima dos *benchmarks*.
- Verificar a eficiência dos solucionadores baseados em satisfatibilidade proposicional ou booleana (SAT) e teoria do módulo de satisfatibilidade (SMT).
- Aplicar diferentes verificadores de modelos na implementação dos algoritmos.
- Comparar a metodologia proposta com técnicas tradicionais, como: algoritmo genético, enxame de partículas, busca de padrões, recozimento simulado e programação não-linear.

Todos os experimentos foram conduzidos em um computador utilizado para este único fim e equipado com um processador Intel Core $i7-4790$ CPU 3.60 GHz, com 16GB de memória RAM e sistema operacional Linux Ubuntu 14.10.

Os verificadores de modelos utilizados são: CBMC v4.5 para Ubuntun com suporte ao solucionador SAT MiniSAT v2.2.0 e o ESBMC v3.1.0 com suporte aos solucionadores SMT, Z3 v4.5.0, Boolector v2.2.0 e MathSAT v5.3.13.

Para avaliação das outras técnicas de otimização usadas para comparação, todos as funções foram avaliadas por meio da ferramenta de otimização do *software* MATLAB 2016b, as respectivas configurações das funções implementadas foram mantidas padrão do MATLAB. Os tempos de execução e as taxas de acertos das tabelas são relativos a média de 100 execuções para cada *benchmark* utilizado e para cada algoritmo avaliado.

4.1.1 Descrição dos Bechmarks

Um grupo de 30 funções retiradas da literatura [35], dentre funções não-convexas, convexas e semi-definidas e definidas positivas, foi utilizado para avaliação de desempenho dos algoritmos de otimização propostos. Todas as funções são multivariáveis com duas variáveis de decisão. Estas funções apresentam diferentes operadores, como por exemplo, seno, cosseno, *floor*, somatório, raiz quadrada e polinômios. E também, podem ter diferentes classificações, como: diferenciável ou não-diferenciável, separável ou não-separável, unimodal ou multimodal. A tabela 4.1 descreve os *benchmarks* de funções utilizadas nos experimentos, com seus respectivos domínios de avaliação e pontos ótimos.

A fim de avaliar os três algoritmos propostos, Genérico, Simplificado e Rápido, as funções da tabela 4.1 foram divididas em outros grupos. O Algoritmo Genérico 1 foi empregado na otimização de todos os *benchmarks* de funções. O Algoritmo Simplificado 2 foi utilizado na otimização de 15 funções selecionadas a partir do conhecimento prévio das mesmas, *benchmarks* $(\#13-\#27)$, podemos afirmar que estas funções são semi-definidas ou definidas positivas, e por fim, o Algoritmo Rápido 3 foi aplicado a 10 funções convexas dentre os *benchmarks* da tabela 4.1, *benchmarks* $(\#21-\#30)$.

Tabela 4.1: Benchmark de funções para problemas de otimização global.

#	Funções	Domínio	Ponto Ótimo
01	Engvall	$-10 \leq x_i \leq 10$	$f(1,0) = 0$
02	Tsoulos	$-1 \leq x_i \leq 1$	$f(0,0) = -2$
03	Zirilli	$-10 \leq x_i \leq 10$	$f(1.046,0) = -0.3523$
04	Step 2	$-100 \leq x_i \leq 100$	$f(0,0) = 0$
05	Scahffer 4	$-10 \leq x_i \leq 10$	$f(0,1.253) = 0.292$
06	Adjiman	$-1 \leq x_i \leq 2$	$f(2,0.10578) = -2.02181$
07	Cosine	$-1 \leq x_i \leq 1$	$f(0,0) = -0.2$
08	S2	$-5 \leq x_i \leq 5$	$f(x_1,0.7) = 2$
09	Styblinski Tang	$-5 \leq x_i \leq 5$	$f(2.903,2.903) = -78.332$
10	Trecanni	$-5 \leq x_i \leq 5$	$f(\{0,0\},\{2,0\}) = 0$
11	Ursem 1	$-3 \leq x_i \leq 3$	$f(1.697136,0) = -4.8168$
12	Branin RCOS	$-5 \leq x_i \leq 15$	$f(\{-\pi,12.275\},\{\pi,2.275\},$ $\{3\pi,2.425\}) = 0.3978873$
13	Wayburn Seader 2	$-500 \leq x_i \leq 500$	$f(\{0.2,1\},\{0.425,1\}) = 0$
14	Alpine 1	$-10 \leq x_i \leq 10$	$f(0,0) = 0$
15	Egg Crate	$-5 \leq x_i \leq 5$	$f(0,0) = 0$
16	Himmeblau	$-5 \leq x_i \leq 5$	$f(3,2) = 0$
17	Leon	$-2 \leq x_i \leq 2$	$f(1,1) = 0$
18	Price 4	$-10 \leq x_i \leq 10$	$f\{(0,0),(2,4),$ $(1.464,-2.506)\} = 0$
19	Schuwefel 2.25	$-10 \leq x_i \leq 10$	$f(1,1) = 0$
20	Sphere	$0 \leq x_i \leq 10$	$f(0,0) = 0$
21	Booth	$-10 \leq x_i \leq 10$	$f(1,3) = 0$
22	Chung	$-10 \leq x_i \leq 10$	$f(0,0) = 0$
23	Cube	$-10 \leq x_i \leq 10$	$f(1,1) = 0$
24	Dixon & Price	$-10 \leq x_i \leq 10$	$f(x_i) = 0, x_i = 2^{-((2i-2)/2i)}$
25	Power Sum	$-1 \leq x_i \leq 1$	$f(0,0) = 0$
26	Schumer	$-10 \leq x_i \leq 10$	$f(0,0) = 0$
27	Sum Square	$-10 \leq x_i \leq 10$	$f(0,0) = 0$
28	Matyas	$-10 \leq x_i \leq 10$	$f(0,0) = 0$
29	Rotated Ellipse	$-500 \leq x_i \leq 500$	$f(0,0) = 0$
30	Zettl	$-5 \leq x_i \leq 10$	$f(0.029,0) = -0.0037$

4.2 Avaliação do Algoritmo Genérico

Os resultados experimentais apresentados na tabela 4.2 são relativos a desempenho do Algoritmo Genérico 1. O tempo de CPU é medido em segundos para encontrar a solução ótima. Cada coluna da tabela informa os tempos de execução do algoritmo para cada solucionador SAT ou SMT, lembrando que o solucionador SAT, MiniSAT, é utilizado em conjunto com o verificador de modelos CBMC e os solucionadores SMT, Z3, Boolector e MathSAT, são empregados

como $back-end$ do verificador de modelos ESBMC. A solução global ótima é encontrada em 100% dos casos para todos os solucionadores SAT/SMT utilizados, considerando uma precisão de 10^{-3} para a solução.

Inicialmente, todos os experimentos foram avaliados usando variáveis do tipo *float* para efetuar os cálculos necessários da função de custo avaliada, no entanto, ocorreram problemas de *overflow* ou *underflow* nas operações de algumas funções em particular quando utilizados os solucionadores Z3 e Boolector, por exemplo, para as funções: *Wayburn Seader* 2 (#13), *Price* 4 (#18), *Chung* (#22), *Cube* (#23) e *Dixon & Price* (#24). Isto ocorre devido ao truncamento de algumas operações aritméticas, por exemplo, seno e cosseno, uma vez que para estes solucionadores o ESBMC tem apenas suporte para cálculos empregando aritmética de ponto-fixo, acarretando erros graves devido a propagação do erro a cada operação, gerando resultados incorretos.

Por esta razão, para funções onde ocorre este problema são utilizadas variáveis do tipo *double*, a fim de melhorar a precisão das operações matemáticas. Os experimentos nas quais foi aplicado este recurso são diferenciados na tabela 4.2 pelo uso do carácter asterisco (*).

Note que, o tempo de execução tende a ser maior quando é utilizado tipos de variável $double$ ao invés de $float$, além disso, este problema não ocorre quando utilizamos o solucionador MathSAT, uma vez que o ESBMC possui suporte a aritmética de ponto-flutuante para este solucionador, garantido uma melhor precisão na representação numérica.

O solucionador MiniSAT utilizado pelo verificador de modelos CBMC obteve melhor desempenho em $53,3\%$ do *benckmark* de funções, em negrito pode-se ver os melhores tempos para cada função de custo, este é baseado em satisfatibilidade proposicional (SAT), assim o espaço de estados é criado mais rapidamente do que para os solucionadores SMT, no entanto, quando aplicado em funções não-convexas, como os *benchmarks* ($\#1-\#12$), o desempenho não foi tão bom quanto os demais, uma vez que tais funções possuem operadores como seno e cosseno, que são melhor tratados pelas teorias adicionais que apenas os solucionadores SMT suportam.

A figura 4.1 mostra o somatório de todos os tempos de otimização para cada solucionador, podemos observar é que o solucionador Boolector e Z3 obtiveram os piores desempenhos gerais, pois como dito anteriormente, para algumas funções foi necessário o uso de tipos de variável *double*, o que impactou consideravelmente nos tempos totais destes solucionadores. Além disso, vemos que o solucionador MathSAT obteve o melhor desempenho geral, uma vez

Tabela 4.2: Resultados Experimentais para o Algoritmo Genérico (tempos em segundos).

#	Boolector	Z3	MathSAT	MiniSAT
1	1020	3653	**662**	692
2	**305**	9023	2865	3793
3	**383**	720	662	1088
4	3	**1**	11	4
5	**6785**	14738	33897	41350
6	**665**	2969	19313	10506
7	**393**	2358	3678	13460
8	32	13	**10**	7000
9	1330	19620	**438**	2260
10	**76**	269	2876	725
11	808	**645**	11737	16853
12	17458	25941	**3245**	4319
13	33794*	36324	37	**21**
14	**537**	6788	590	3948
15	5770	3565	**500**	8750
16	4495	11320	10	**3**
17	269	1254	4	**2**
18	16049*	110591	6	**1**
19	2972	5489	7	**3**
20	1	1	2	**<1**
21	2660	972	5	**1**
22	839*	5812*	2	**<1**
23	170779*	77684*	5	**2**
24	36337*	22626*	**8**	**8**
25	**3**	40	4	**3**
26	445	20	4	**2**
27	41	**1**	3	**1**
28	5945	5267	23	**22**
29	1210	2741	16	**12**
30	271	611	11	**6**

que o seu suporte a aritmética de ponto-flutuante o permite melhor tratamento para as funções não-convexas. Adicionalmente, foi observado que para funções que possuem mais de um mínimo global, como *Wayburn Seader* 2 (#13) com dois mínimos globais em $f(0.2.1)$ e $f(0.425,1)$, o algoritmo encontra sempre a solução com menor precisão, neste caso $f(0.2,1)$, isso é devido ao fato de o algoritmo evoluir a busca pelo mínimo aumentando a precisão das variáveis de decisão, conforme comentado no Capítulo 3. Assim, para se encontrar a outra solução devemos adicionar uma restrição ao problema através da diretiva `ASSUME`, de modo que o algoritmo exclua a solução encontrada anteriormente.

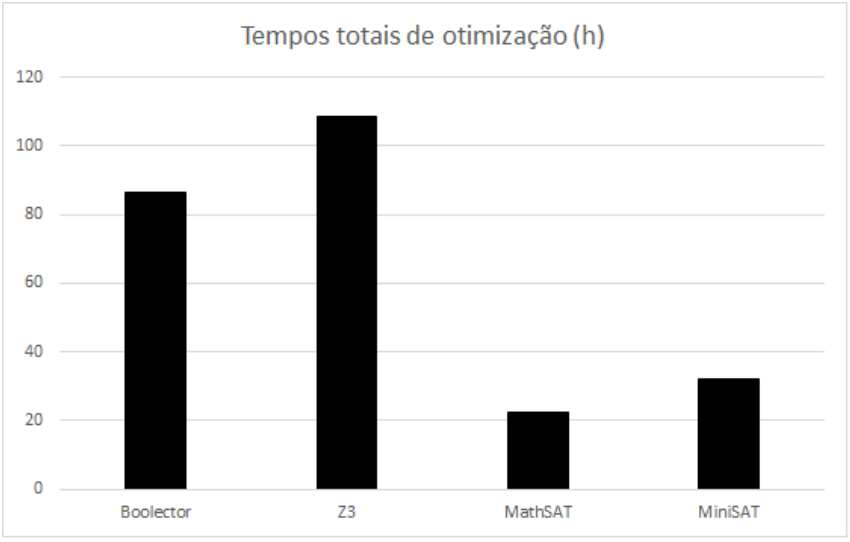


Figura 4.1: Tempos totais de otimização para o Algoritmo Genérico.

4.3 Avaliação do Algoritmo Simplificado

O Algoritmo Simplificado 2 é aplicado a funções em que tem-se algum conhecimento prévio, como por exemplo, funções semi-definida ou definida positiva, onde não há a necessidade da busca por valores de $f(x)$ negativos. A função *EggCrate* (#15) dada pela equação (4.1) tem o mínimo global em $f(0,0) = 0$.

$$f(x_1,x_2) = x_1^2 + x_2^2 + 25[\sin^2(x_1) + \sin^2(x_2)] \tag{4.1}$$

Através da análise desta função é possível afirmar que esta não assume valores negativos para qualquer valor das variáveis x_1 e x_2. Como dito anteriormente, para avaliar a efetividade do Algoritmo 2, 15 *benchmarks* foram selecionados da tabela 4.1 (#13 – #27), todos são funções semi-definidas positivas. Novamente, o algoritmo proposto obteve acerto de 100% na busca pela solução ótima, considerando a precisão da solução de 10^{-3}.

A tabela 4.3 apresenta a comparação dos tempos de execução entre os Algoritmos Genérico e Simplificado. Pode-se perceber que o Algoritmo Simplificado reduz consideravelmente o tempo de otimização, principalmente para os solucionadores Z3 e Boolector, que utilizam aritmética de ponto-fixo para as operações matemáticas, enquanto que o solucionador MathSAT, o suporte a aritmética de ponto-flutuante foi o responsável por não ocorrer uma queda maior nos tempos, uma vez que o verificador ESBMC, utiliza mais recursos para este solucionador, já para o solucionador MiniSAT utilizado pelo verificador de modelos, houve também pequena

melhora em relação ao Algoritmo Genérico.

Tabela 4.3: Resultados Experimentais para o Algoritmo Simplificado para os *benchmarks* #13 – #27, comparados com os resultados do Algoritmo Genérico (tempos em segundo).

#	Algoritmo Simplificado				Algoritmo Genérico			
	Boolector	Z3	MathSAT	MiniSAT	Boolector	Z3	MathSAT	MiniSAT
13	215	2446	30	**10**	33794*	36324	37	21
14	74	**2**	413	79	537	6788	590	3948
15	34	**2**	240	192	5770	3565	500	8750
16	**1**	**1**	6	3	4495	11320	10	3
17	1	**<1**	2	1	269	1254	4	2
18	**<1**	**<1**	5	4	16049*	110591	6	1
19	**1**	2	5	3	2972	5489	7	3
20	**<1**	**<1**	**<1**	1	1	1	2	<1
21	**<1**	**<1**	1	1	2660	972	5	1
22	**<1**	**<1**	**<1**	2	839*	5812*	2	<1
23	**<1**	**<1**	2	2	170779*	77684*	5	2
24	14	**2**	6	6	36337*	22626*	8	8
25	**<1**	**<1**	2	2	3	40	4	3
26	**<1**	**<1**	1	2	445	20	4	2
27	**<1**	**<1**	**<1**	1	41	1	3	1

Conforme mostra figura 4.2, os solucionadores Boolector e MiniSAT obtiveram os melhores desempenhos gerais, porém, o solucionador Z3 obteve o melhor desempenho para 86,6% das funções, o que impactou em seu desempenho geral foi apenas a função #13, um ajuste da taxa de aprendizado pode resolver este problema.

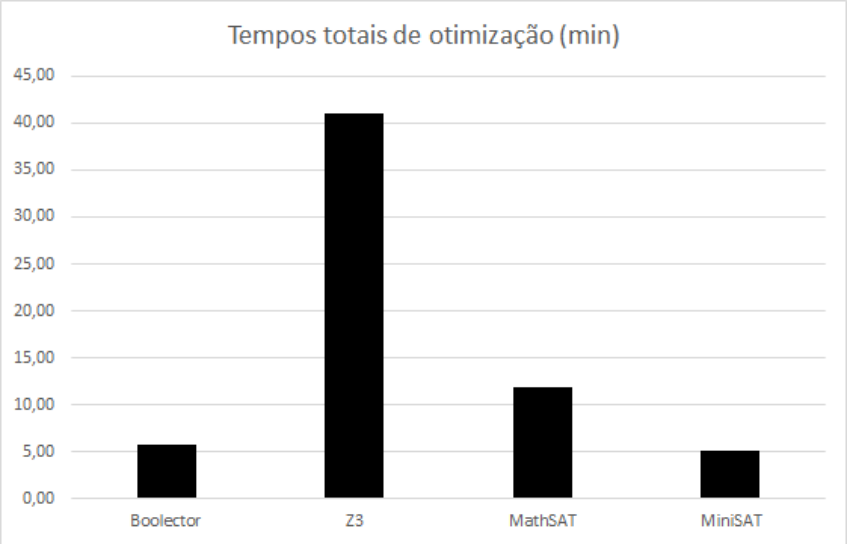


Figura 4.2: Tempos totais de otimização para o Algoritmo Simplificado.

A proposta do Algoritmo Simplificado é reduzir o tempo de verificação, para funções onde se tem algum conhecimento a priori do mínimo da função, este objetivo é alcançado aumentando-se o número de propriedades verificadas em um único processo de verificação a partir da taxa de aprendizado α. Conforme descrito no Capítulo 3, Seção 3.2.2, a variável δ gera várias propriedades para verificação a partir de α, estabelecido como uma entrada do algoritmo. No entanto, se α for muito elevado a quantidade de propriedades geradas será consequentemente muito maior, o que pode acarretar em um tempo de execução maior. A taxa de aprendizado α deve ser escolhida de forma empírica para cada função, nos experimentos da tabela 4.3 foi utilizado $\alpha = 5$ para todos os *benchmarks*.

Além disso, se analisarmos a função $S2$ a equação (4.2), podemos facilmente detectar que não há valores de $f(x)$ menores que 2 para a função, na linha 1 do Algoritmo 2, podemos atribuir $f_m = 2$. Esta pequena modificação na inicialização de f_m restringi o espaço de estados e o tempo de verificação.

$$f(x_1, x_2) = 2 + (x_2 - 0.7)^2 \tag{4.2}$$

4.4 Avaliação do Algoritmo Rápido

Os resultados experimentais para o Algoritmo Rápido 3 são apresentados na tabela 4.4. Este algoritmo é aplicável apenas para funções convexas, onde existe apenas um mínimo global, para estes problemas o espaço de estados é reduzido a cada iteração do laço `while` externo do Algoritmo 3, garantindo a presença do mínimo global no novo espaço delimitado, assim o algoritmo reduz o tempo de otimização.

Para avaliação da eficiência do Algoritmo Rápido 3, foram selecionadas 10 funções convexas, *benchmark* #21 − #30, a tabela 4.4 mostra os resultados dos tempos de otimização para comparação dos Algoritmos Genérico e Rápido.

Nota-se que o Algoritmo Rápido obteve significativo aumento de desempenho, quando é utilizado os solucionadores SMT, para aplicação aos *benchmarks* de funções convexas quando comparado ao Algoritmo Genérico. O Algoritmo Rápido é 1000 vezes mais rápido usando o solucionador SMT Boolector e 750 vezes mais rápido para o solucionador SMT Z3 em comparação do uso desses solucionadores pelo Algoritmo Genérico. No entanto, mais uma vez

o solucionador SAT MiniSAT obteve o melhor desempenho para 70% das funções conforme mostra a tabela 3 e ligeira melhora em relação ao Algoritmo Genérico.

Tabela 4.4: Resultados Experimentais para o Algoritmo Rápido para os *benchmarks* #21 – #30, comparados com os resultados do Algoritmo Genérico (tempos em segundos).

#	Algoritmo Rápido			Algoritmo Genérico		
	Boolector	Z3	MiniSAT	Boolector	Z3	MiniSAT
21	**<1**	**<1**	**<1**	2660	972	1
22	33*	26*	**<1**	839*	5812*	<1
23	43*	25	**1**	170779*	77684*	2
24	59*	10*	**6**	36337*	22626*	8
25	**1**	10	**1**	3	40	3
26	**1**	2	**1**	445	20	1
27	1	**<1**	1	41	1	1
28	7	**2**	8	5945	5267	22
29	2	**1***	4	1210	2741	12
30	63*	76	**6**	271	611	6

A figura 4.3 mostra os tempos totais de otimização, e nota-se que o solucionador MiniSAT também obteve o melhor desempenho geral para as 10 funções convexas do *benckmark*.

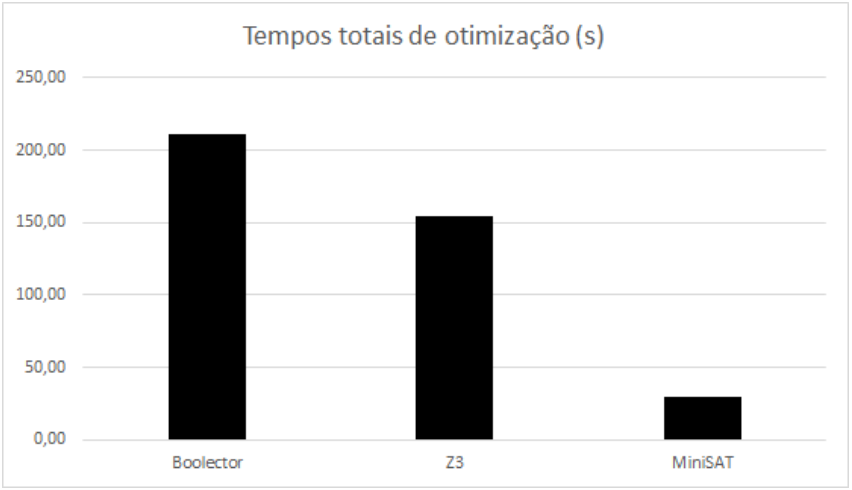


Figura 4.3: Tempos totais de otimização para o Algoritmo Rápido.

4.5 Comparação com outras Técnicas

Nesta Seção, os algoritmos propostos baseado em teorias de satisfatibilidade são comparados com outras técnicas de otimização tradicionais: algoritmo genético (*genetic algorithm*

- GA), enxame de partículas (*particle swarm* - ParSwarm), reconhecimento de padrões (*pattern search* - PatSearch), recozimento simulado (*simulated annealing* - SA) e programação não-linear (*non-linear programming* - NLP).

A tabela 4.5 mostra a taxa de acerto e tempo médio de otmização para cada função quando aplicado a técnica desenvolvida baseada em SAT/SMT e as outras técnicas (ParSwarm, GA, ParSearch, SA e NLP). Para a metodologia apresentada neste trabalho são mostrados os tempos de otimização do melhor solucionador e melhor algoritmo, a identificação é feita da seguinte forma: (1) Genérico, (2) Simplificado e (3) Rápido. Todas as outras técnicas de otimização foram executas 100 vezes usando o MATLAB, para obtenção da taxa de acerto e tempo médio de otimização mais confiável para cada função.

A taxa de acerto dos algoritmos de otimização baseado em SAT/SMT é omitida na tabela, pois a abordagem encontra a solução ótima em 100% dos experimentos. Os experimentos mostram que abordagem é superior a qualquer outra técnica de otimização avaliada, no entanto o tempo de otimização é usualmente maior.

As outras técnicas de otimização são muito sensíveis a não-convexidade, por esta razão, elas são frequentemente presas por mínimos locais, suas taxas de acertos para essa classe de funções são muito menores, *benchmarks* ($\#1 - \#12$). No entanto, apresentam melhor desempenho quando aplicadas funções convexas, devido ao fato dessas funções não apresentarem mínimos locais e apenas um mínimo global, *benchmarks* ($\#21 - \#30$).

4.6 Conclusão

Os resultados apresentados neste Capítulo mostram que a novo método de otimização baseado em teorias de satisfatibilidade, como satisfatibilidade proposicional (SAT) e teoria do módulo de satisfatibilidade (SMT), é adequado a diferentes classes de funções, em particular funções não-convexas, que são um grande desafio para algoritmos de otimização.

Os algoritmos propostos são avaliados exaustivamente usando uma grande conjunto de *benchmarks* públicos utilizados para avaliação de algoritmos de otimização. Também foi feita a avaliação dos algoritmos usando diferentes solucionadores SAT/SMT e diferentes verificadores de modelos. Além disso, os mesmos foram comparados com outras técnicas de otimização, como: algoritmo genético, enxame de partículas, busca de padrões, recozimento simulado e programação não-linear.

Tabela 4.5: Resultados Experimentais para as Técnicas Tradicionais e o melhor Algoritmo baseado em solucionadores SAT e SMT (tempos em segundos).

#	SAT	SMT	GA		ParSwarm		ParSearch		SA		NLP	
	T	T	R(%)	T	R(%)	T	R(%)	T	R(%)	T	R(%)	T
1	$661^{(1)}$	$691^{(1)}$	90	1	100	2	90	3	95	1	100	7
2	$305^{(1)}$	$3793^{(1)}$	100	9	95	1	100	3	75	9	0	6
3	$383^{(1)}$	$1088^{(1)}$	100	9	100	1	100	3	60	1	75	2
4	$1^{(1)}$	$4^{(1)}$	0	9	0	1	0	2	0	8	0	1
5	$6785^{(1)}$	$41350^{(1)}$	30	1	15	<1	0	<1	0	2	0	<1
6	$665^{(1)}$	$10506^{(1)}$	0	10	100	1	0	4	80	2	95	2
7	$393^{(1)}$	$13460^{(1)}$	100	9	100	1	95	3	95	2	15	2
8	$10^{(1)}$	$7000^{(1)}$	65	<1	100	<1	100	<1	85	1	100	<1
9	$438^{(1)}$	$2260^{(1)}$	100	9	100	1	50	3	100	1	35	2
10	$76^{(1)}$	$725^{(1)}$	0	9	0	1	0	3	0	1	0	2
11	$645^{(1)}$	$16853^{(1)}$	100	9	100	1	100	3	80	1	65	2
12	$3245^{(1)}$	$4319^{(1)}$	100	8	100	9	100	4	75	8	0	5
13	$30^{(2)}$	$10^{(2)}$	100	1	95	1	100	3	100	1	100	2
14	$2^{(2)}$	$79^{(2)}$	25	1	45	3	10	4	50	1	0	9
15	$2^{(2)}$	$192^{(2)}$	100	9	100	1	70	3	100	1	25	2
16	$1^{(2)}$	$3^{(2)}$	60	9	50	1	25	3	15	1	35	2
17	$< 1^{(2)}$	$1^{(2)}$	90	1	75	2	0	7	10	1	100	4
18	$< 1^{(2)}$	$4^{(2)}$	0	9	10	2	0	7	0	4	50	2
19	$1^{(2)}$	$3^{(2)}$	100	1	95	1	100	3	100	1	100	2
20	$< 1^{(2)}$	$1^{(2)}$	100	10	100	7	100	4	100	1	100	2
21	$< 1^{(3)}$	$< 1^{(3)}$	100	10	100	2	100	6	95	1	100	2
22	$< 1^{(2)}$	$< 1^{(3)}$	100	9	100	1	100	4	90	1	100	5
23	$< 1^{(2)}$	$1^{(3)}$	20	1	30	3	0	8	10	2	100	7
24	$1^{(3)}$	$6^{(3)}$	0	9	0	2	0	3	0	1	0	2
25	$< 1^{(2),(3)}$	$1^{(3)}$	100	9	100	1	100	3	50	1	100	2
26	$< 1^{(2),(3)}$	$1^{(3)}$	100	9	100	1	100	4	75	1	100	4
27	$< 1^{(3)}$	$1^{(3)}$	100	9	100	1	100	4	100	1	100	2
28	$2^{(3)}$	$8^{(3)}$	100	9	100	1	100	8	10	1	100	2
29	$1^{(3)}$	$4^{(3)}$	100	9	100	2	100	7	100	1	100	2
30	$63^{(3)}$	$6^{(3)}$	100	9	100	1	100	3	60	1	75	2

Os algoritmos de otimização baseados em satisfatibilidade são capazes de encontrar a solução ótima em 100% dos experimentos, quando aplicados a classe de funções especificadas para cada um. O tempo de otimização é significativamente reduzido comparado a Araújo *et al.* [12].

Capítulo 5

Estudo de Caso

Neste Capítulo, a metodologia desenvolvida no Capítulo 3 será utilizada para resolver o problema de planejamento de caminho para robôs móveis autônomos. A Seção 5.1 apresenta os tipos e principais aplicações de robôs móveis. A Seção 5.2 descreve o problema de planejamento de caminho para robôs móveis autônomos. Na Seção 5.3 é discutido sobre alguns métodos utilizados para resolução do problema de planejamento de caminho e trajetória. A Seção 5.4 mostrará a formulação deste problema como um problema de otimização e a aplicação, do método desenvolvido anteriormente, no contexto de geração de caminho bidimensional. Finalmente na Seção 5.5, resultados experimentais obtidos são apresentados.

5.1 Introdução

Robôs móveis são dispositivos de transporte automático, também são plataformas mecânicas dotadas de um sistema capaz de navegar através de um determinado ambiente de trabalho. Um robô móvel autônomo tem capacidade de alcançar um ou vários objetivos com uma intervenção muito pequena ou nenhuma do ser humano. Em especial, podem ser classificados quanto ao seu ambiente de trabalho, como:

- Terrestre - veículo terrestre não tripulado (*Unmanned Ground Vehicle* - UGV).
- Aquático - veículo aquático não tripulado (*Unmanned Underwater Vehicle* - UUV).
- Aéreo - veículo aéreo não tripulado (*Unmanned Aerial Vehicle* - UAV).

Cada tipo de robô móvel autônomo apresenta certas características que determinam sua aplicação, uma habilidade comum a todos é a capacidade de coletar informações sobre o ambiente e detectar objetos de interesse como pessoas e veículos. Possuem as mais diversas aplicações de acordo com suas características, existem hoje uma grandes variedade de robôs móveis (veículos) autônomos, capazes de executar as diferentes tarefas, eles têm sido aplicados com êxito em operações militares [60] e aplicações civis e comerciais.

Devido a simplicidade de controle, os UGVs foram os escolhidos pela NASA para exploração de outros planetas [61], *Spirit* e *Opportunity* são dois UGVs utilizados na exploração de Marte. Outras aplicações dos diferentes tipos de veículos autônomos, são: vigilância e patrulhamento [62, 63], monitoramento e reconhecimento de terrenos (mapeamento geográfico) [64–66], tratamento de imagens à velocidade [67] no caso de UAV, sensoriamento remoto [68, 69], exploração [70], transporte [71, 72], busca e salvamento em regiões perigosas [73–75].

Visto que necessitam de grande quantidade de energia para manter seu funcionamento, o planejamento de caminho com objetivo de minimizar o percurso necessário para realização de determinada tarefa, se faz um dos temas essenciais para assegurar que os robôs móveis autônomos levem a cabo navegações com sucesso.

5.2 Problema de Planejamento de Caminho

O planejamento de caminho é uma das etapas do projeto de trajetórias de robôs móveis autônomos. O objetivo fundamental de um algoritmo de planejamento de caminho é gerar os pontos necessários para guiar o robô móvel na execução da mesma, além de assegurar que o veículo será capaz de executar o caminho. Por exemplo, o raio de curvatura é uma restrição imposta em trajetórias geradas para veículos terrestres, uma vez que um automóvel típico não pode mover-se lateralmente.

Encontrar o caminho ótimo em uma região não é uma tarefa trivial, o número de atividades no ambiente e os tipos de obstáculos presentes no mesmo, refletem diretamente na complexidade do problema de planejamento de caminho. Na maioria dos casos isto requer excessivo tempo de processamento, especialmente se o número de pontos a serem visitados durante a execução da caminho for alto.

Assim, existe uma necessidade crescente em estudar e desenvolver técnicas para o pla-

nejamento de caminho ótimo, tanto do ponto de vista do comprimento da trajetória quanto do consumo de energia do sistema, visando gerar maior autonomia ao sistema.

Com base no grau de conhecimento das informações ambientais, dados codificados que definem onde há a presença de obstáculos para o sistema, e dos diferentes tipos de obstáculos, o planejamento de caminho de robôs móveis, pode ser dividido nas seguintes categorias:

1. Planejamento de caminho sob condições ambientais conhecidas com obstáculos estáticos;
2. Planejamento de caminho sob condições ambientais conhecidas com obstáculos dinâmicos;
3. Planejamento de caminho sob ambientes desconhecidos com obstáculos estáticos;
4. Planejamento de caminho sob ambientes desconhecidos com obstáculos dinâmicos.

O planejamento de caminho sob ambientes desconhecidos só pode ser executado de forma *on-line*, já que o mesmo será gerado conforme o robô móvel autônomo interage com o ambiente, sendo necessário um conjunto de sensores que captarão a informação do ambiente em tempo real, ao passo que em ambientes conhecidos o planejamento poderá se feito de maneira *off-line* visto que temos um conhecimento prévio do ambiente.

Além disso, pode haver a presença de obstáculos dinâmicos, por exemplo, outro robô móvel no ambiente, nesses casos o planejamento de caminho deve conter um módulo que deverá ser capaz de reconfigurá-lo, a fim evitar a colisão entre o robô (veículo) e o obstáculo em tempo hábil, e posteriormente deverá colocá-lo de volta ao caminho anterior. Já quando há a presença de obstáculos estáticos, estes podem ser evitados tratando-os como informações de um ambiente conhecido, reconfigurando o caminho previamente obtida.

Para o método de planejamento adotado neste trabalho, basicamente, segue-se os seguintes passos:

1. Codificação do ambiente, espaço de movimentação e obstáculos estáticos, no caso de caminho em ambientes conhecidos.
2. Utilizar um método de pesquisa de caminho, composto por pontos no espaço e suas respectivas orientações, para encontrar o que satisfaça as restrições definidas pelo problema.

O objetivo geral do processo de planejamento de caminho consiste em definir o conjunto de segmentos de reta que conectados forma o caminho ótimo, a fim de cumprir uma missão definida, respeitando as restrições ambientais e as limitações dinâmicas do robô móvel. O método

em estudo neste trabalho consiste na modelagem do problema de planejamento de caminho como um problema de otimização e resolução através de um algoritmo de otimização baseado satisfatibilidade, conforme proposto no Capítulo 3.

5.3 Trabalhos Relacionados

Diferentes abordagens foram adotadas na literatura sobre planejamento de trajetórias para robôs móveis usando técnicas de otimização, incluindo métodos probabilísticos [76–78], algoritmos evolutivos [79–81], métodos de pesquisa em grafo [82–84], métodos geométricos [85–91] ou técnicas bio-inspiradas [92, 93].

Métodos geométricos baseados em curvas de Dubins [94], B-splines e Bézier [85], [87], [79] e [88] também foram aplicados. Porém nenhum garante a otimalidade das trajetórias, visto que o primeiro usa do artifício de manter a altura constante quando realizam o trecho circular das curvas de Dubins, e os demais utilizam curvas de aproximações, B-splines e Bézier, para interporlar os *waypoints*.

Outro caminho está na criação de heurísticas para acelerar o tempo de obtenção da solução. Essas técnicas estão baseadas em métodos modernos de otimização. Dentre algumas podemos citar, algoritmos genéticos [80,81], algoritmo de enxames de partículas [95], ou ainda, ambos utilizados em conjunto para melhorar o método CFO [78]. No entanto, as heurísticas não obtem a solução ótima como o método de verificação de modelos garante, podendo na maioria dos casos gerar soluções locais ou aproximadas do ótimo. Entretanto, devemos tomar cuidado quanto ao tempo de execução, uma vez que podemos ter o espaço de estados demasiadamente grande conforme a complexidade do problema aumenta.

Métodos de pesquisa em grafo é outro ramo estudado para método de planejamento caminho, como no algoritmo Dijkstra e na teoria de campos potenciais [90], algoritmo de busca diferencial (DS) melhorado, que usa teoria quântica para resolver o problema de mínimos locais [83]. Para planejamento *online*, o algoritmo A* [82] e um novo algoritmo chamado LEA [84] tem sido utilizados. Tais métodos discretizam o ambiente, similar ao algoritmo baseado em satisfatibilidade, além disso, algoritmos de pesquisa em grafo necessitam de incremento afim de diminuirem seu tempo de busca, o que pode ser um problema também para o método baseado em satisfatibilidade.

Finalmente, os que utilizam técnicas bio-inspiradas, como os baseados na teoria Tau [92,

93] que estuda o comportamento do movimento natural quando animais (incluindo humanos) abordam um objeto fixo ou móvel. Tais técnicas podem não ser factíveis, visto que levam o sistema a atingir os limites das restrições.

Embora seja utilizada neste trabalho teorias de satisfatibilidade para o planejamento de caminho em ambiente conhecido com obstáculos estáticos, é importante mencionar o planejamento para ambientes com a presença de obstáculos dinâmicos, que também pode ser considerado um ambiente com múltiplos veículos. Alguns dos métodos utilizados para este cenário são: árvore aleatória de exploração rápida (RRT - *Rapidly-expansion Random Tree*) [76], campos potenciais [89], enxame de partículas [77] e teoria tau [93]. Nota-se que em todos estes trabalhos o planejamento é feito *online*, já que há a necessidade de reconfiguração do caminho quando houver eminente perigo de colisão com o obstáculo dinâmico.

5.4 Planejamento de Caminho

O problema descrito na Seção 5.2 tornou-se popular com o uso intensivo de vários tipos de robôs móveis autônomos. Nesta Seção a metodologia desenvolvida para resolução de problemas de otimização é aplicada no contexto de planejamento de caminho ótimo para veículos autônomos.

5.4.1 Formulação do Problema

Em um ambiente complexo sujeito a obstáculos de diferentes tipos, um veículo autônomo deve executar uma missão de deslocar-se de um ponto inicial até um ponto alvo descrevendo a menor caminho, dada pela definição 5.1.

Definição 5.1. *Caminho é o conjunto de segmentos de reta conectados sucessivamente de modo a ligar um ponto inicial a um ponto final.*

Então, o objetivo principal do algoritmo de planejamento de caminho é encontrar os n pontos que formam o caminho que conecta os pontos inicial e alvo da missão gerando o menor custo de deslocamento, ou seja, o problema de planejamento de caminho é essencialmente um problema de otimização de trajetória. Para efeito de simplificação, inicialmente, o problema será restrito ao ambiente bidimensional.

5.4.1.1 Modelagem do Ambiente

A modelagem de obstáculos é a tarefa chave para o planejamento de caminho de veículos autônomos. Dados o ponto inicial e ponto alvo definidos respectivamente como **S** e **T**. Ao mover-se do ponto inicial ao ponto alvo, um veículo autônomo poderá encontrar diferentes tipos de obstáculos de diferentes formas, como por exemplo: outros veículos, radares, terrenos elevados e construções. Como o problema foi restringindo a caminhos bidimensionais, tais obstáculos podem ser descritos como um perfil dos mesmos.

Uma forma simples de representar esses obstáculos é através de círculos de tal modo que os obstáculos fiquem inscritos nestes círculos. Assim, a parametrização dos obstáculos no ambiente pode ser feita pelos seus centros geométricos e ponto de maior distância com respeito ao centro, definindo o centro, (x_o, y_o), e raio do círculo r, respectivamente. A equação (5.1) delimita um obstáculo qualquer.

$$(x - x_o)^2 + (y - y_o)^2 = r^2 \tag{5.1}$$

Além disso, o raio r de cada obstáculo será acrescido de σ, adicionando uma margem de segurança para o desvio de cada obstáculo. Dessa forma cada obstáculo será delimitado por um círculo dado pela equação (5.2).

$$(x - x_o)^2 + (y - y_o)^2 = (r + \sigma)^2 \tag{5.2}$$

Se as coordenadas **S** e **T** são definidas por (x_1, y_1) e (x_n, y_n), podemos dividir o eixo x no intervalo de (x_1, x_n) em $n-1$ porções definidas pelas linhas $x_1, x_2, x_3, ..., x_n$, de forma análoga, o eixo y pode ser divido pelas linhas $y_1, y_2, y_3, ..., y_n$. A intersecção entre tais linhas formam os pontos discretos, coordenadas cartesianas do espaço, $P_i = (x_i, y_i)$, para $i = 1, 2, ...n-1, n$. Estes pontos conectados por um segmentos de reta caracteriza um caminho entre os pontos inicial **S** e alvo **T**. Assim, o problema de otimização é encontrar a matriz de variáveis de decisão **L** definida como, $\mathbf{L} = [P_1, P_2, ..., P_{n-1}, P_n]$. A Figura 5.1 ilustra a representação de dois obstáculos assim como um caminho qualquer entre os pontos inicial e alvo para $n = 6$.

Podemos perceber que o problema de planejamento de caminho bidimensional é um problema de otimização de $2n-4$ dimensões. Nota-se que se $n \to \infty$, o caminho torna-se uma trajetória perfeitamente suave, no entanto, a dimensão será extremamente elevada e sabe-se que quanto maior a dimensão de um problema de otimização maior a complexidade e precisão do

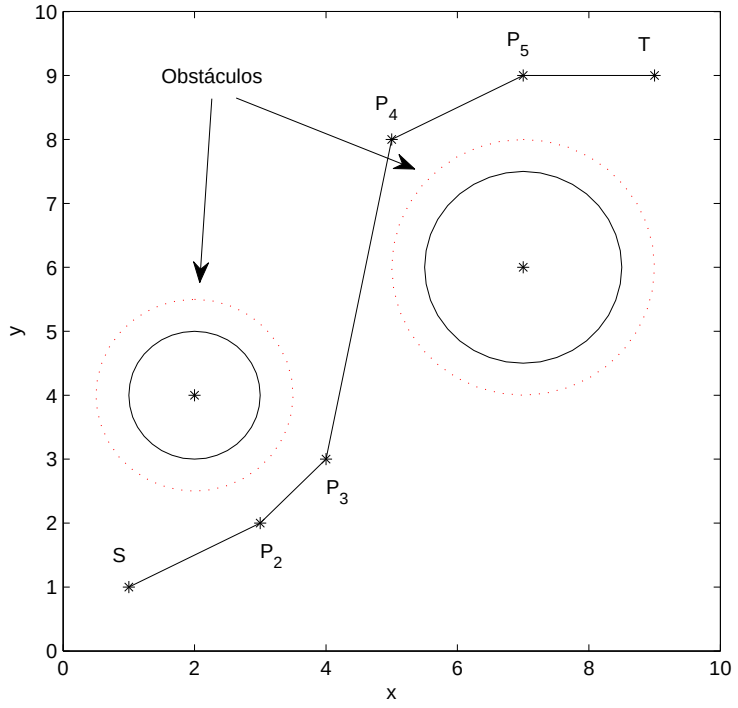


Figura 5.1: Representação do ambiente com obstáculos e um caminho para $n = 6$.

mesmo. Dessa forma, o valor de n afeta diretamente a eficiência do algoritmo, afim de controlá-lo e aumentar a eficiência da otimização, n pode ser limitado.

5.4.1.2 Função de Custo

No estudo em questão o melhor caminho ou caminho ótimo é definido como o menor caminho em termos de distância percorrida pelo veículo para realização da missão, uma vez que, um caminho menor pode ser realizado em menor tempo com o menor consumo de energia pelo veículo. Então, a função de custo, J, para uma trajetória bidimensional é definida conforme a equação (5.3):

$$J = \sum_{i=1}^{n-1} E_i \tag{5.3}$$

onde $E_i = \|(x_{i+1}, y_{i+1}) - (x_i, y_i)\|_2$ é a distância euclidiana entre dois pontos consecutivos do caminho e (x_i, y_i) denota o i-ésimo ponto P_i do caminho planejado.

A função de custo definida pela equação (5.3) é um somatório de normas euclidianas, se $n \to \infty$ a função de custo J é definida pela integral de linha do ponto P_1 até P_n.

Sabe-se que a norma euclidiana é uma função convexa, além disso, a soma de funções convexas resulta em uma função também convexa. Logo, podemos utilizar o Algoritmo 3 para resolver o problema proposto, uma vez que a função de custo é convexa.

5.4.1.3 Formulação do Problema de Otimização

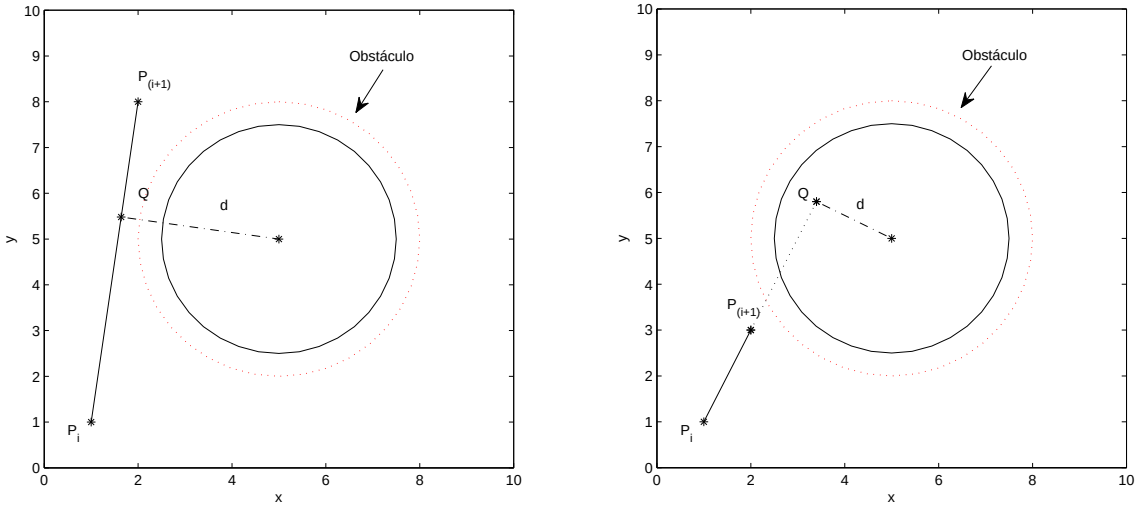
Para formulação completa do problema de planejamento de caminho como problema de otimização falta-se definir as restrições do problema. Tais restrições são sobre cada ponto P_i que compõe o caminho. Conforme dito anteriormente, o caminho é formada por $n-1$ segmentos de reta os quais conectam os n pontos. Cada segmento não pode ultrapassar os obstáculos presentes no ambiente e devem ainda estar contido dentro dos limites do ambiente de realização da missão. Assim, de acordo com a definição no Capítulo 2, Seção 2.1, o problema de otimização de trajetória pode ser definido conforme a equação (5.4)

$$\begin{aligned} \min_{L} \quad & J(L), \\ \text{s.a.} \quad & p_i(L) \notin \mathbb{O} \\ & p_i(L) \in \mathbb{E} \\ & i = 1, ..., n-1. \end{aligned} \tag{5.4}$$

onde $\mathbb{O}$ é o conjunto de pontos definido pelos obstáculos; $\mathbb{E}$ é o conjunto de pontos definido pelos limites do ambiente; n é o número de pontos que compõem o caminho; e $p_i(L)$ é todo ponto pertencente ao i-ésimo segmento de reta do caminho definido pelo vetor $\mathbf{L}$. O i-ésimo segmento de reta conecta os pontos P_i e P_{i+1} e qualquer ponto $p_i(L)$ pertencente ao mesmo pode ser encontrado pela equação (5.5).

$$p_i(L) = \gamma(P_i) + (1-\gamma)(P_{i+1}) \quad \gamma \in [0,1] \tag{5.5}$$

Todos os pontos $p_i(L)$ devem desviar de todos os obstáculos, para isso deve-se garantir que a distância do centro de qualquer obstáculo até ao i-ésimo segmento seja maior que o raio r acrescido da margem de segurança σ, referente aquele respectivo obstáculo, se o ponto Q usado para medir tal distância até o referido segmento pertence ao mesmo, conforme ilustrado na figura 5.2a, caso contrário, se Q não pertence ao segmento, figura 5.2b, basta garantir que os pontos extremos que geram o segmento estejam fora do obstáculo, assim todos os pontos pertencentes ao segmento também estarão.



(a) Caso 1 (b) Caso 2

Figura 5.2: Segmentos de reta para desvio dos obstáculos.

5.4.2 Algoritmo de Planejamento de Caminho

O Algoritmo 3 para resolução de problemas de otimização convexo descrito no Capítulo 3 será utilizado para resolver o problema descrito anteriormente. Uma das características do algoritmo de otimização baseado em satisfatibilidade proposto é discretizar o espaço de estados sucessivamente a cada iteração para obter a solução ótima com precisão ajustável. No entanto, o problema de otimização descrito na Seção anterior já é formulado de maneira discretizada, o que o torno de fácil adaptação ao algoritmo proposto.

Seguindo os passos de modelagem e especificação descritos na Seção 3.1, o problema de otimização descrito na Seção 5.4 é codificado como mostra a figura 5.3. O código é responsável por verificar se o literal J_{otimo} dado pela equação (5.6) é satisfeito para os valores J_c candidatos a ótimo, tal que $J_c = J(\mathbf{L}^{(i-1)})$, similar a equação (3.4).

$$J_{otimo} \iff J(\mathbf{L}) > J_c \tag{5.6}$$

A função `rest_pontos` é responsável por inserir as restrições sobre $n-1$ segmentos de reta que conecta os pontos P_i e P_{i+1}, de maneira que satisfaça as condições explicadas na seção anterior. Para isso é utilizada a diretiva ASSUME. Note que função deve ser executada para cada obstáculo. Sua codificação é mostrada na figura 5.4.

Se o código definido na figura 5.3 retornar FALSO, isto é, a propriedade dada por J_{otimo}

```
1  #define DIM 2    // dimensão do espaço
2  #define n 1      // números de pontos que compõe o caminho
3  #define p 1      // precisão da localização dos pontos no espaço
4  #define J_i 25   // valor da função de custo na iteração anterior
5  #define no 1     // número de obstáculos
6  // dados dos obstáculos
7  float sigma = 0.5;  // margem de segurança
8  float x0[no] = {5}; // coordenadas do centro 'x'
9  float y0[no] = {5}; // coordenadas do centro 'y'
10 float r[no] = {2.5};  // raios do obstáculos
11 int main() {
12   int i, j;
13   int A [DIM] = {1*p, 1*p};
14   int B [DIM] = {9*p, 9*p};
15   int lim [DIM][2] = { 0*p, 10*p, 0*p, 10*p};
16   // declaração dos estados, coordenadas x=x[i][0] e y=x[i][1] como não-
         determinísticos
17   for (i=0; i<n; i++)
18     for(j=0; j<DIM; j++)
19       x[i][j] = nondet_int();
20   // restrições sobre os limites do ambiente e obstáculos
21   for (i=0; i<n; i++) {
22     __ESBMC_assume ( (x[i][0] >= lim[0][0]) && (x[i][0] <= lim[0][1]) );
23     __ESBMC_assume ( (x[i][1] >= lim[1][0]) && (x[i][1] <= lim[1][1]) );
24   }
25   for (j=0; j<no; j++)
26     rest_pontos (A, 0, x0[j]*p, y0[j]*p, r[j]*p);
27   for (i=1; i<n; i++) {
28     for (j=0; j<no; j++)
29       rest_pontos (x[i-1], i, x0[j]*p, y0[j]*p, r[j]*p);
30   }
31   for (j=0; j<no; j++)
32     rest_pontos (B, n-1, x0[j]*p, y0[j]*p, r[j]*p);
33   // cálculo da função de custo
34   float Aux1[DIM], Aux2[DIM];
35   float J = 0.0;
36   for (j=0; j<DIM; j++)
37     Aux1[j] = A[j]/p;
38   for (i=0; i<n; i++) {
39     for (j=0; j<DIM; j++)
40       Aux2[j] = (float) x[i][j]/p;
41     J = J + dist(Aux1,Aux2);
42     for (j=0; j<DIM; j++)
43       Aux1[j] = Aux2[j];
44   }
45   for (j=0; j<DIM; j++)
46     Aux2[j] = B[j]/p;
47   J = J + dist(Aux1,Aux2);
48   __ESBMC_assume ( J < J_i );
49   assert ( J > J_i );
50   return 0;
51 }
```

Figura 5.3: Código C do planejamento de caminho bidimensional.

```
1  void rest_points (int P1 [DIM], int i, float x0, float y0, float r) {
2    __ESBMC_assume ( (x[i][0]-x0)*(x[i][0]-x0) + (x[i][1]-y0)*(x[i][1]-y0)
         > r*r + sigma );
3    float a, b, c;
4    if (P1[0]-x[i][0]==0){
5      a = 1;
6      b = 0;
7      c = -P1[0];
8    }
9    else{
10     a = (float) (P1[1]-x[i][1])/(P1[0]-x[i][0]);
11     b = -1;
12     c = (float) -a*P1[0]+P1[1];
13   }
14   float Py = (a*a*y0-a*b*x0-b*c)/(a*a+b*b);
15   if (((Py-x[i][1])/(P1[1]-x[i][1])>=0) && ((Py-x[i][1])/(P1[1]-x[i][1])
       <=1)) {
16     float d = (float) abs2(a*x0+b*y0+c) / sqrt2(a*a+b*b);
17     __ESBMC_assume ( d > r );
18   }
19 }
```

Figura 5.4: Código C da função `rest_pontos`.

foi violada, então, existe um $\mathbf{L}^{(i)}$ para qual $J(\mathbf{L}^{(i)}) < J_c$, a partir disso, pode-se realizar a atualização de *J_i* e executar o código novamente, até que a condição J_{otimo} seja satisfeita, assim é encontrado o vetor $\mathbf{L}$ que define o caminho ótimo bidimensional entre os pontos inicial e alvo com n pontos para uma precisão da localização dos pontos dada por p. Note que não significa que não há uma trajetória menor para um valor de n maior, de maneira que o valor de n deve ser incrementado para nova verificação. Quando o código retornar VERDADEIRO para a condição J_{otimo} após o aumento de n de maneira consecutiva, significa que este é o máximo número de pontos que gera o menor caminho para a precisão p definida.

Para continuarmos a busca por um caminho menor, a variável p deve ser atualizada até o valor máximo de η, de tal maneira que a precisão desejada da localização dos pontos no espaço no espaço é dada pela equação (3.5).

Note que para resolver o problema de otimização dado pela equação (5.4), o código da Figura 5.3 deve ser executado recursivamente, com a atualização iterativa dos valores de precisão e número de pontos que compõe o caminho. Assim, o Algoritmo 4 mostra as etapas descritas anteriormente para execução do planejamento de caminho bidimensional.

Como dito anteriormente, quanto maior o número de pontos que compõe o caminho, n, mais suave este será, no entanto, perde-se em eficiência do algoritmo pois aumenta-se a

Algoritmo 4: Algoritmo de planejamento de caminho baseado em satisfatibilidade.

input : Função de custo $J(\mathbf{L})$, um conjunto de restrições dos obstáculos $\mathbb{O}$ e um conjunto de restrições do ambiente $\mathbb{E}$, que definem Ω, e uma precisão desejada η

output: O vetor ótimo de variáveis de decisão $\mathbf{L}^*$, e o valor ótimo do valor da função objetivo $J(\mathbf{L}^*)$

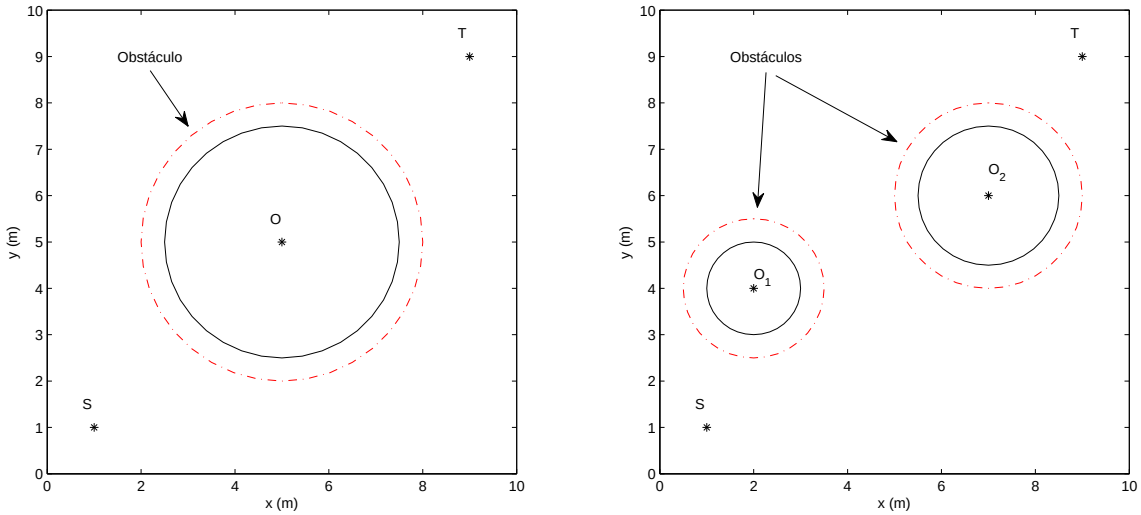
Inicialize $J(\boldsymbol{L}^{(0)})$ aleatoriamente;
Inicialize a variável de precisão com $p = 1$, $k = 0$ e $i = 1$;
Inicialize o número de pontos, $n = 1$;
Declare o vetor de variáveis de decisão $\boldsymbol{L}^i$ como variáveis inteiras não-determinísticas;
while $k \leq \eta$ **do**
    Defina os limites superior e inferior de $\boldsymbol{L}$ com a diretiva `ASSUME`, tal que $L \in \Omega^k$;
    Descreva o modelo da função objetivo $J(\boldsymbol{L})$;
    do
        do
            Defina a restrição $J(\boldsymbol{L}^{(i)}) < J(\boldsymbol{L}^{(i-1)})$ com a diretiva `ASSUME`;
            Verifique a satisfabilidade de J_{optimo} dado pela equação (5.6);
            Atualize $\boldsymbol{L}^* = \boldsymbol{L}^{(i)}$ e $J(\boldsymbol{L}^*) = J(\boldsymbol{L}^{(i)})$ baseado no contraexemplo;
            Faça $i = i + 1$;
        while $\neg J_{optimal}$ *é satisfatível*;
        if $\neg J_{optimal}$ *é não satisfatível consecutivamente* **then**
            break
        end
        else
            Atualize o número de pontos, n;
        end
    while *VERDADEIRO*;
    Faça $k = k + 1$;
    Atualize o conjunto Ω^k;
    Atualize a variável de precisão, p;
end
$\boldsymbol{L}^* = \boldsymbol{L}^{(i)}$ *e* $J(\boldsymbol{L}^*) = J(\boldsymbol{L}^{(i)})$;
return $\boldsymbol{L}^*$ *e* $J(\boldsymbol{L}^*)$;

complexidade do problema, ou seja, aumenta-se a dimensão do problema de otimização. Para o problema de caminho bidimensional a cada inserção de um ponto no mesmo, insere-se duas variáveis de decisão ao problema. No entanto, o processo descrito limita o valor de n de acordo com a precisão de casas decimais da localização dos pontos no espaço definida por η.

5.5 Avaliação Experimental

Assim como nos testes do Capítulo 3, os experimentos deste Capítulo foram realizados no mesmo computador. No entanto, para o ESBMC *v*3.1.0 foi utilizado o solucionador SMT MathSAT *v*5.3.13, pois possui suporte a aritmética de ponto flutuante necessária para o calculo de distâncias entre pontos. Para comparação foi utilizado o CBMC *v*4.5 para Ubuntun, com o solucionador SAT MiniSAT *v*2.2.0.

Inicialmente, para exemplificar o algoritmo proposto e avaliação do mesmo, foi elaborado um cenário simples, chamado **Cenário 1**, que consiste no movimento de veículo autônomo em um espaço bidimensional onde este deve desviar de apenas um obstáculo conforme mostra a figura 5.5a. Para nova avaliação, criou-se outro cenário, chamado **Cenário 2**, similar ao primeiro, mas com a presença de dois obstáculos. Este novo cenário é ilustrado na figura 5.5b.



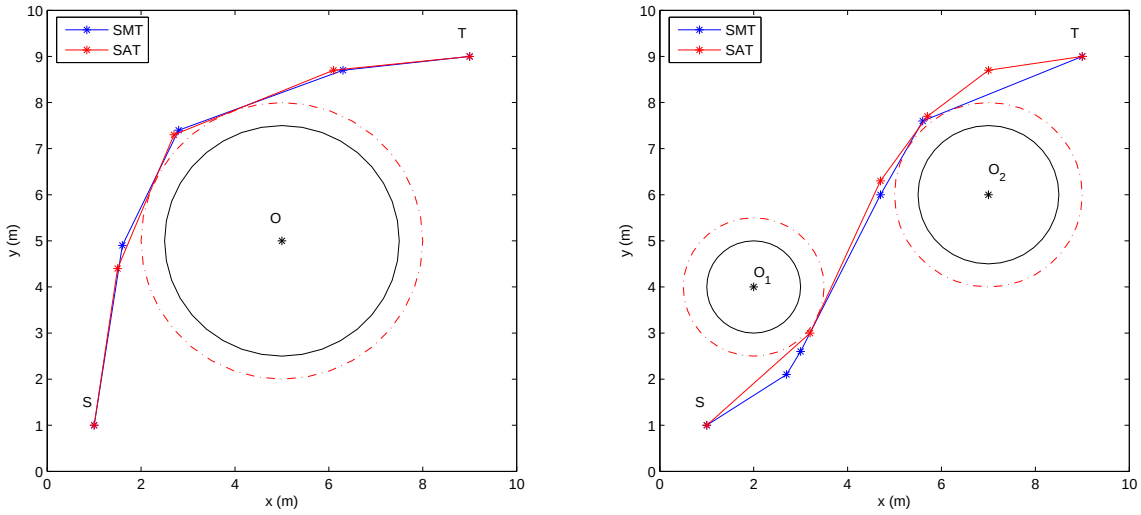
(a) **Cenário 1**. (b) **Cenário 2**.

Figura 5.5: Cenários de teste para o Algoritmo 4.

A objetivo é gerar um caminho de **S** a **T**, considerando 10^{-1}, ou seja $10cm$, a precisão na localização dos pontos que compõem o mesmo, o obstáculo é definido conforme a equação (5.2). Para o **Cenário 1**, o obstáculo é localizado com centro em $O(x_0, y_0) = (5,5)$ e raio $r = 2.5$, já para o **Cenário 2**, os obstáculos estão localizados em $O_1(x_0, y_0) = (2,4)$ de raio $r_1 = 1$ e $O_2(x_0, y_0) = (7,8)$ de raio $r_2 = 1.5$, a margem de segurança de desvio dos obstáculos para ambos os cenário é de $\sigma = 0.5$. Todas as medidas de distâncias estão em m.

O tempo de execução do algoritmo para os dois verificadores de modelos empregados

para o **Cenário 1** sofreu *timeout* de 3 dias, enquanto para o **Cenário 2** o *timeout* foi de 1 semana. Os caminho obtidos pelo Algoritmo 4 para ambos os cenários anteriores são ilustrados na figura 5.6.

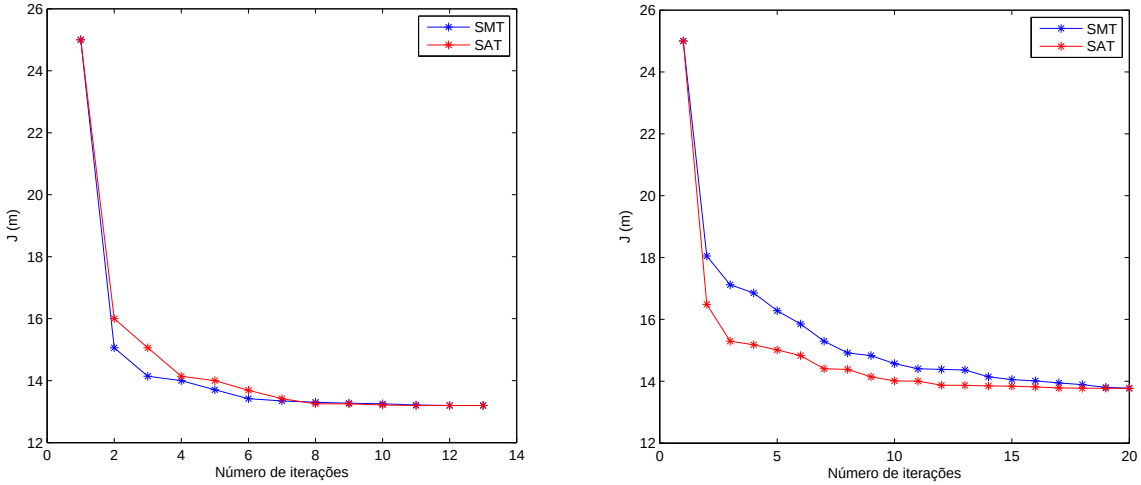


(a) Caminho gerado para o **Cenário 1**.

(b) Caminho gerado para o **Cenário 2**.

Figura 5.6: Caminhos obtidos pelo Algoritmo 4.

A figura 5.7 mostra as trajetórias de convergência da função de custo para ambos os cenários e solucionadores SAT/SMT. Nota-se que o valor da função de custo sempre diminui a cada iteração e esta convergindo para a solução ótima.



(a) Trajetória para o **Cenário 1**.

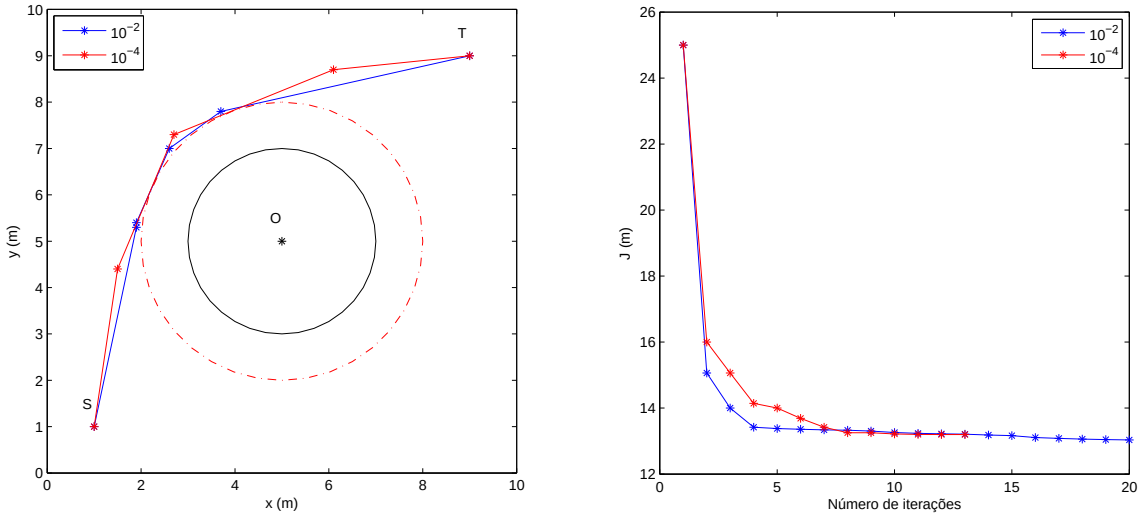
(b) Trajetória para o **Cenário 2**.

Figura 5.7: Trajetórias do valor da função de custo para os Cenários.

O problema de *timeout* para os dois cenários ocorreu devido ao fato de os pontos que compõe o caminho estarem livres para ocuparem qualquer posição no espaço, gerando uma grande quantidade de possibilidades de configuração do caminho.

Além disso, o passo da função de custo a cada iteração é de 10^{-4}, ou seja, a cada iteração a melhora na função de custo é de no mínimo $0,0001$. Assim, o algoritmo necessita de mais iterações para convergir a solução ótima e quanto mais próximo desta, cada iteração aumenta consideravelmente o tempo de convergência.

Uma solução a este problema é aumentar este passo, forçando a cada iteração uma melhora maior para a função de custo. Para comparação este passo foi fixado em 10^{-2}, que é 100 vezes maior que o anteriormente utilizado. A figura 5.8 mostra a comparação entre os caminhos obtidos e as trajetórias de convergência da função de custo para o **Cenário 1** considerando os passos citados anteriormente.



(a) Caminhos para o **Cenário 1**.

(b) Trajetórias para o **Cenário 1**.

Figura 5.8: Comparação dos caminhos e trajetórias da função de custo para os passos de 10^{-2} e 10^{-4}.

Com esta pequena modificação em apenas $4h$ chegou-se a uma solução de menor distância, com um número de iterações maior e com o número de pontos que compõem o caminho maior do que anteriormente, o que representa apenas $5,5\%$ do tempo anterior. Este passo pode ainda ser aumentado, de modo que pose-se obter soluções satisfatórias em um menor tempo. No entanto, não há a garantia que a solução ótima seja encontrada, apenas garante-se que a

solução encontrada está a uma distância referente ao passo escolhido da solução ótima, o que para muitas aplicações é aceitável.

5.6 Conclusão

Foi apresentado neste Capítulo o algoritmo de planejamento de caminho para robôs móveis, o problema é resolvido como um problema de otimização com aplicação do Algoritmo Rápido 3 proposto no Capítulo 3.

Os tempos de execução foram elevados, sofrendo *timeout*, para os dois verificadores de modelos empregados devido a complexidade do problema, isto é, a cada ponto inserido ao caminho são adicionadas duas variáveis de decisão se a trajetória é bidimensional e três variáveis de decisão para o caso tridimensional. No entanto, se $t \rightarrow \infty$, o caminho tende ao ótimo, ou seja, o menor caminho.

O problema de *timeout* foi solucionado aumentando-se o passo de melhora da função de custo, propiciando soluções mais rápidas e que muitas são satisfatórias para a aplicação. Além disso, o algoritmo desenvolve incrementando o número de pontos do caminho, logo, se $n \rightarrow \infty$, o caminho tende a uma trajetória perfeitamente suave.

Capítulo 6

Conclusão

Este trabalho possui três grandes contribuições: apresentação de uma nova metodologia para resolução de problemas otimização de funções tanto não-convexas como convexas no domínio real, comparação dos solucionadores disponíveis baseados em satisfatibilidade booleana (SAT) e teoria do módulo de satisfatibilidade (SMT), além da aplicação do método proposto no contexto de planejamento de caminho para robôs móveis autônomos.

Primeiramente, esta dissertação apresenta o potencial da verificação de modelos baseada em SAT/SMT para solucionar problemas de otimização. Algumas metodologias de otimização baseada em SAT/SMT tem sido desenvolvidas durante a última década para aplicação em diversos tipos de problemas, no entanto, a proposta apresentada é a primeira na qual é possível tratar problemas diretamente no domínio real com precisão ajustável, além de ser adequada a diferentes classes de funções.

Nesse estudo forma utilizadas as ferramentas CBMC e ESBMC para comparação dos solucionadores baseado em SAT e SMT, também pode-se verificar a viabilidade do uso de qualquer verificador de modelos na implementação da metodologia.

Finalmente, o estudo aplica a metodologia desenvolvida no contexto do planejamento de caminho bidimensional para robôs móveis autônomos. Para a avaliação de desempenho, novamente, os verificadores de modelos CBMC e ESBMC foram utilizados para comparação dos solucionadores baseados em SAT e SMT. Apesar de ocorrem *timeout* os algoritmos convergiam a solução ótima. Como alternativa ao problema foi proposto um aumento no passo mínimo de melhora da função de custo, isto diminuiu consideravelmente o tempo de otimização sem sacrificar a solução encontrada.

Não foi possível o teste do algoritmo de planejamento de caminho aplicado ao ambiente tridimensional, os testes experimentais se tornaram inviáveis devido a *overflow* de memória no computador utilizado para execução dos testes, uma vez que o problema de otimização para geração de caminho possui elevada complexidade.

6.1 Trabalhos Futuros

Um dos pontos fracos dos algoritmos baseado em satisfatibilidade é tempo de execução. Porém, ainda é possível diminuí-lo consideravelvemente. Pode-se explorar algumas soluções, como:

- uso técnicas metaheurísticas em conjunto com a metodologia desenvolvida para manipulação do espaço de estados;
- uso de outros solucionadores e verificadores de modelos;
- execução paralelizada dos algoritmos nos núcleos do processador;
- mudanças na implementação dos algoritmos;
- implementação de funções matemáticas mais eficientes;
- em particular, para o Algoritmo Simplificado 2, pode-se usar a taxa de aprendizado, α, de forma dinâmica.

Além disso, o constante avanço no desenvolvimento de solucionadores SAT/SMT também tem diminuído o tempo de verificação. Outro ponto a ser explorado, consiste na extensão da metodologia proposta para aplicação em problemas de otimização multiobjetivo.

Conforme dito anteriormente, a metodologia foi aplicada apenas no planejamento de caminhos bidimensionais para robôs móveis autônomos devido a *overflow* de memória, a aplicação em trajetórias tridimensionais pode ser alcançada expandindo a capacidade de memória do computador, mas também tem-se diminuído o uso de memória com atualização dos solucionadores e verificadores de modelos.

Para aplicação no planejamento de trajetórias suaves, a metodologia desenvolvida pode ser usada em conjunto outras técnicas matemáticas baseadas em curvas paramétricas, como: Dubins, Bézier e Splines, podendo gerar soluções muito boas e com maior rapidez.

Apêndice A

Publicações

A.1 Artigo Submetido

- **Rodrigo F. Araújo**, Higo F. Albuquerque, Iury V. Bessa, Lucas C. Cordeiro, João Edgar C. Filho. *Counterexample Guided Inductive Optimization*. **Journal Science of Computer Programming (JSCP)**.

A.2 Artigo Publicado

- **Rodrigo F. Araújo**, Iury V. Bessa, Lucas C. Cordeiro, João Edgar C. Filho. *SMT-based Verification Applied to Non-Convex Optimization Problems*. **2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)**. João Pessoa, 2016.

Referências Bibliográficas

[1] DEB, K. *Optimization for Engineering Design: Algorithms and Examples*. [S.l.]: Prentice-Hall of India, 2004. ISBN 9788120309432.

[2] GATTIE, D. K.; WICKLEIN, R. C. Curricular value and instructional needs for infusing engineering design into k-12 technology education. *Journal of Technology Education*, v. 19, n. 1, 2007.

[3] SHOHAM, Y. Computer science and game theory. *Commun. ACM*, New York, NY, USA, v. 51, n. 8, p. 74–79, Aug 2008. ISSN 0001-0782.

[4] TEICH, J. Hardware/Software Codesign: The Past, the Present, and Predicting the Future. *Proceedings of the IEEE*, v. 100, n. Special Centennial Issue, p. 1411–1430, May 2012. ISSN 0018-9219.

[5] KOWALSKI, M. A.; SIKORSKI, K. A.; STENGER, F. *Selected Topics in Approximation and Computation*. [S.l.]: Oxford University Press, 1995. I-XIV, 1-349 p. ISBN 978-0-19-508059-9.

[6] DERIGS, U. *Optimization and Operations Research*. [S.l.]: EOLSS Publications, 2009. v. 1. ISBN 9781905839483.

[7] GARFINKEL, R.; NEMHAUSER, G. *Integer Programming*. [S.l.]: Wiley, 1972. (Series in decision and control). ISBN 9780471291954.

[8] BARTHOLOMEW-BIGGS, M. The Steepest Descent Method. In: ____. *Nonlinear Optimization with Engineering Applications*. Boston, MA: Springer US, 2008. v. 19, p. 1–8. ISBN 9780387787237.

[9] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. 1st. ed. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 9780201157673.

[10] CAVAZZUTI, M. *Optimization Methods: From Theory to Design Scientific and Technological Aspects in Mechanics*. [S.l.]: Springer Berlin Heidelberg, 2012. ISBN 9783642311864.

[11] DORIGO, M.; BIRATTARI, M.; STUTZLE, T. Ant colony optimization. *IEEE Computat. Intell. Mag.*, v. 1, n. 4, p. 28–39, Nov 2006. ISSN 1556-603X.

[12] ARAÚJO, R. et al. SMT-based Verification Applied to Non-convex Optimization Problems. In: *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2016. p. 1–8.

[13] BIERE, A. Bounded model checking. In: *Handbook of Satisfiability*. [S.l.]: IOS Press, 2009. p. 457–481.

[14] NIEUWENHUIS, R.; OLIVERAS, A. On SAT Modulo Theories and Optimization Problems. In: ____. *Theory and Applications of Satisfiability Testing - SAT 2006: 9th International Conference, Seattle, WA, USA, August 12-15, 2006. Proceedings*. [S.l.]: Springer Berlin Heidelberg, 2006. p. 156–169. ISBN 9783540372073.

[15] ELDIB, H.; WANG, C. An SMT Based Method for Optimizing Arithmetic Computations in Embedded Software Code. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 33, n. 11, p. 1611–1622, Nov 2014. ISSN 02780070.

[16] ESTRADA, G. G. A Note on Designing Logical Circuits Using SAT. In: *ICES*. [S.l.]: Springer Berlin Heidelberg, 2003. p. 410–421. ISBN 978354036553.

[17] TRINDADE, A.; CORDEIRO, L. C. Applying SMT-based Verification to Hardware/-Software Partitioning in Embedded Systems. *Design Automation for Embedded Systems*, v. 20, n. 1, p. 1–19, Mar 2016.

[18] TRINDADE, A.; ISMAIL, H.; CORDEIRO, L. Applying Multi-Core Model Checking to Hardware-Software Partitioning in Embedded Systems. In: *2015 Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2015. p. 102–105.

[19] TRINDADE, A.; CORDEIRO, L. Aplicando Verificação de Modelos para o Particionamento de Hardware/Software. In: *SBESC*. [S.l.: s.n.], 2014. p. 6.

[20] COTTON, S. et al. Multi-criteria optimization for mapping programs to multi-processors. In: *SIES*. [S.l.: s.n.], 2011. p. 9–17.

[21] PISTER, M.; TAUTSCHNIG, M.; BAUER, A. Tool-support for the analysis of hybrid systems and models. *2007 10th Design, Automation and Test in Europe Conference and Exhibition*, IEEE Computer Society, p. 1–6, April 2007. ISSN 1530-1591.

[22] NUZZO, P. et al. CalCS: SMT Solving for Non-linear Convex Constraints. In: . [S.l.]: IEEE, 2010. p. 71–79. ISBN 9780983567806.

[23] GAO, S.; KONG, S.; CLARKE, E. M. dReal: An SMT Solver for Nonlinear Theories over the Reals. In: ____. *Automated Deduction – CADE-24: 24th International Conference on Automated Deduction*. [S.l.]: Springer Berlin Heidelberg, 2013. p. 208–214. ISBN 9783642385742.

[24] BJØRNER, N.; PHAN, A.-D.; FLECKENSTEIN, L. νZ - An Optimizing SMT Solver. In: *TACAS*. [S.l.]: Springer Berlin Heidelberg, 2015. p. 194–199. ISBN 9783662466810.

[25] LI, Y. et al. Symbolic optimization with SMT solvers. In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. [S.l.]: ACM, 2014. v. 49, n. 1, p. 607–618. ISBN 9781450325448. ISSN 0362-1340.

[26] SEBASTIANI, R.; TRENTIN, P. OptiMathSAT: A tool for optimization modulo theories. In: *Computer Aided Verification: 27th International Conference CAV*. [S.l.]: Springer International Publishing, 2015. p. 447–454. ISBN 9783319216904.

[27] SEBASTIANI, R.; TRENTIN, P. Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions. In: *Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. [S.l.: s.n.], 2015. p. 335–349. ISBN 9783662466810.

[28] SEBASTIANI, R.; TOMASI, S. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics, ACM TOCL*, v. 16, n. 2, p. 12:1–12:43, March 2015. ISSN 1529-3785.

[29] SHOUKRY, Y. et al. Scalable lazy SMT-based motion planning. In: *IEEE 55th Conference on Decision and Control, CDC*. [S.l.: s.n.], 2016. p. 6683–6688.

[30] PAVLINOVIC, Z.; KING, T.; WIES, T. Practical SMT-based type error localization. In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP*. [S.l.: s.n.], 2015. p. 412–423. ISBN 9781450336697.

[31] MOURA, L. D.; BJØRNER, N. Z3: An efficient SMT solver. In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. [S.l.: s.n.], 2008. p. 337–340. ISBN 9783540787990.

[32] BRUMMAYER, R.; BIERE, A. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. [S.l.: s.n.], 2009. p. 174–177. ISBN 9783642007675.

[33] CIMATTI, A. et al. The MathSAT5 SMT solver. In: *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. [S.l.: s.n.], 2013. p. 93–107. ISBN 9783642367427.

[34] EÉN, N.; SÖRENSSON, N. An extensible SAT-solver. In: *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT*. [S.l.: s.n.], 2003. v. 2919, p. 502–518. ISBN 9783540246053.

[35] JAMIL, M.; YANG, X.-S. A literature survey of benchmark functions for global optimisation problems. *IJMMNO*, v. 4, n. 2, p. 150–194, 2013.

[36] OLSSON, A. *Particle Swarm Optimization: Theory, Techniques and Applications*. [S.l.]: Nova Science Publishers, 2010. (Engineering tools, techniques and tables). ISBN 9781616685270.

[37] ALBERTO, P. et al. Pattern search methods for user-provided points: Application to molecular geometry problems. *SIAM Journal on Optimization*, SIAM, v. 14, n. 4, p. 1216–1236, 2004.

[38] LAARHOVEN, P. J. M.; AARTS, E. H. L. (Ed.). *Simulated Annealing: Theory and Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1987. ISBN 9027725136.

[39] BYRD, R. H.; GILBERT, J. C.; NOCEDAL, J. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, Springer, v. 89, n. 1, p. 149–185, 2000. ISSN 14364646.

[40] RAO, S. S. *Engineering Optimization: Theory and Practice: Fourth Edition*. [S.l.]: John Wiley and Sons, 2009. ISBN 9780470183526.

[41] MOURA, L. de; BJØRNER, N. Satisfiability modulo theories: An appetizer. In: *Formal Methods: Foundations and Applications, 12th Brazilian Symposium on Formal Methods, SBMF*. [S.l.]: Springer, 2009. v. 5902, p. 23–36. ISBN 9783642104510.

[42] BJØRNER, N.; MOURA, L. de. $z3^{10}$: Applications, enablers, challenges and directions. In: *Sixth International Workshop on Constraints in Formal Verification Grenoble, France*. [S.l.: s.n.], 2009.

[43] CLARKE JR., E. M.; GRUMBERG, O.; PELED, D. A. *Model Checking*. Cambridge, MA, USA: MIT Press, 1999. ISBN 0262032708.

[44] JHALA, R.; MAJUMDAR, R. Software model checking. *ACM Comput. Surv.*, New York, NY, USA, v. 41, n. 4, p. 21:1–21:54, out. 2009. ISSN 03600300.

[45] BAIER, C.; KATOEN, J.-P. *Principles of Model Checking*. [S.l.]: MIT Press, 2008. ISBN 9780262026499.

[46] BIERE, A. et al. Symbolic model checking without bdds. In: *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS*. [S.l.: s.n.], 1999. p. 193–207. ISBN 3540657037.

[47] BIERE, A. et al. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands: IOS Press, 2009. ISBN 9781586039295.

[48] CORDEIRO, L.; FISCHER, B.; MARQUES-SILVA, J. SMT-based bounded model checking for embedded ANSI-C software. In: *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, ASE*. [S.l.: s.n.], 2012. v. 38, n. 4, p. 137–148. ISBN 9780769538914.

[49] KROENING, D.; TAUTSCHNIG, M. CBMC – C bounded model checker. In: ____. *Tools and Algorithms for the Construction and Analysis of Systems: 20th International Conference, TACAS*. [S.l.: s.n.], 2014. p. 389–391. ISBN 9783642548628.

[50] MORSE, J. et al. ESBMC 1.22 - (competition contribution). In: *TACAS*. [S.l.: s.n.], 2014. p. 405–407.

[51] CLARKE, E.; KROENING, D.; LERDA, F. A tool for checking ANSI-C programs. In: ____. *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS*. [S.l.: s.n.], 2004. p. 168–176. ISBN 9783540247302.

[52] CORDEIRO, L. SMT-based bounded model checking for multi-threaded software in embedded systems. In: *2010 ACM/IEEE 32nd International Conference on Software Engineering, ICSE*. [S.l.: s.n.], 2010. v. 2, p. 373–376. ISBN 9781605587196.

[53] CORDEIRO, L.; FISCHER, B. Verifying multi-threaded software using SMT-based context-bounded model checking. In: *Proceedings of the 33rd International Conference on Software Engineering, ICSE*. [S.l.: s.n.], 2011. p. 331–340.

[54] RAMALHO, M. et al. SMT-based bounded model checking of C++ programs. In: *20th IEEE International Conference and Workshops on Engineering of Computer Based Systems, ECBS*. [S.l.: s.n.], 2013. p. 147–156.

[55] ABREU, R. B.; CORDEIRO, L. C.; FILHO, E. B. L. Verifying fixed-point digital filters using SMT-based bounded model checking. *CoRR*, abs/1305.2892, June 2013.

[56] BESSA, I. et al. SMT-based bounded model checking of fixed-point digital controllers. *CoRR*, abs/1403.5172, Mar 2014.

[57] ISMAIL, H. et al. Dsverifier: A bounded model checking tool for digital systems. In: *Model Checking Software - 22nd International Symposium, SPIN*. [S.l.: s.n.], 2015. p. 126–131.

[58] BEYER, D.; KEREMOGLU, M. E. CPAchecker: A tool for configurable software verification. In: ____. *Computer Aided Verification: 23rd International Conference, CAV*. [S.l.: s.n.], 2011. p. 184–190. ISBN 9783642221101.

[59] BOYD, S.; VANDENBERGHE, L. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. ISBN 0521833787.

[60] CARAFANO, J.; GUDGEL, A. The pentagon's robots: Arming the future. *The Heritage Foundation*, p. 1–6, 2007.

[61] NGUYEN-HUU, P.-N.; TITUS, J. Reliability and failure in unmanned ground vehicle (UGV). *University of Michigan*, 2016.

[62] BARTON, K.; KINGSTON, D. Systematic surveillance for UAVs: A feedforward iterative learning control approach. In: *2013 American Control Conference, ACC*. [S.l.: s.n.], 2013. p. 5917–5922. ISBN 9781479901777.

[63] LEE, K. S. et al. Autonomous patrol and surveillance system using unmanned aerial vehicles. In: *Environment and Electrical Engineering (EEEIC), IEEE 15th International Conference*. [S.l.: s.n.], 2015. p. 1291–1297.

[64] KANISTRAS, K. et al. A survey of unmanned aerial vehicles UAVs for traffic monitoring. In: *International Conference Unmanned Aircraft Systems, ICUAS*. [S.l.: s.n.], 2013. p. 221–234.

[65] HEINTZ, F.; RUDOL, P.; DOHERTY, P. From images to traffic behavior - A UAV tracking and monitoring application. In: *Information Fusion*. [S.l.: s.n.], 2007. p. 1–8.

[66] SALVO, G.; CARUSO, L.; SCORDO, A. Urban traffic analysis through an UAV. *Procedia - Social and Behavioral Sciences*, v. 111, p. 1083 – 1091, 2014. ISSN 18770428.

[67] FANG, P. et al. An improved object tracking method in UAV videos. *Procedia Engineering*, v. 15, p. 634 – 638, 2011. ISSN 18777058.

[68] GAMBA, M. T. et al. Light weight gnss-based passive radar for remote sensing UAV applications. In: *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2015 IEEE 1st International Forum on*. [S.l.: s.n.], 2015. p. 341–348.

[69] BHARDWAJ, A. et al. UAVs as remote sensing platform in glaciology: Present applications and future prospects. *Remote Sensing of Environment*, v. 175, p. 196 – 204, 2016. ISSN 00344257.

[70] SUJIT, P. B.; SOUSA, J.; PEREIRA, F. L. Coordination strategies between UAV and AUVs for ocean exploration. In: *Control Conference, ECC*. [S.l.: s.n.], 2009. p. 115–120.

[71] HARIK, E. H. C. et al. UAV-UGV cooperation for objects transportation in an industrial area. In: *International Conference on Industrial Technology, ICIT*. [S.l.: s.n.], 2015. p. 547–552.

[72] HENRIQUES, R. V. B.; ALMEIDA, M. M. de; RAFFO, G. V. Nonlinear control of a tiltrotor UAV for load transportation. *11th IFAC Symposium on Robot Control SYROCO*, v. 48, n. 19, p. 232 – 237, 2015.

[73] KARMA, S. et al. Use of unmanned vehicles in search and rescue operations in forest fires: Advantages and limitations observed in a field trial. *International Journal of Disaster Risk Reduction*, v. 13, p. 307 – 312, 2015.

[74] VARELA, G. et al. Autonomous UAV based search operations using constrained sampling evolutionary algorithms. *Neurocomputing*, v. 132, p. 54 – 67, maio 2014. ISSN 09252312.

[75] NAIDOO, Y.; STOPFORTH, R.; BRIGHT, G. Development of an UAV for search & rescue applications. In: *AFRICON*. [S.l.: s.n.], 2011. p. 1–6. ISSN 21530025.

[76] LIN, Y.; SARIPALLI, S. Path planning using 3D dubins curve for unmanned aerial vehicles. In: *International Conference on Unmanned Aircraft Systems, ICUAS*. [S.l.: s.n.], 2014. p. 296–304.

[77] ALEJO, D. et al. Collision-free trajectory planning based on maneuver selection-particle swarm optimization. In: *International Conference on Unmanned Aircraft Systems, ICUAS*. [S.l.: s.n.], 2015. p. 72–81.

[78] CHEN, Y. et al. Modified central force optimization MCFO algorithm for 3D UAV path planning. *Neurocomputing*, v. 171, p. 878 – 888, 2016. ISSN 09252312.

[79] SAHINGOZ, O. K. Flyable path planning for a multi-UAV system with genetic algorithms and bezier curves. In: *International Conference on Unmanned Aircraft Systems, ICUAS*. [S.l.: s.n.], 2013. p. 41–48.

[80] ERGEZER, H.; LEBLEBICIOGLU, M. K. 3D path planning for UAVs for maximum information collection. In: *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 79–88.

[81] OZALP, N.; SAHINGOZ, O. K. Optimal UAV path planning in a 3D threat environment by using parallel evolutionary algorithms. In: *International Conference on Unmanned Aircraft Systems, ICUAS*. [S.l.: s.n.], 2013. p. 308–317.

[82] XU, H. et al. Trajectory planning of unmanned aerial vehicle based on A* algorithm. In: *IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems, CYBER*. [S.l.: s.n.], 2014. p. 463–468.

[83] QU, X.; DUAN, H. Three dimensional trajectory planning of unmanned aerial vehicles based on quantum differential search. In: *Control Conference (CCC), 2014 33rd Chinese*. [S.l.: s.n.], 2014. p. 142–146.

[84] XIAO-DONG, C.; DE-YUN, Z.; RUO-NAN, Z. New method for UAV online path planning. In: *IEEE International Conference on Signal Processing, Communication and Computing, ICSPCC*. [S.l.: s.n.], 2013. p. 1–5.

[85] LI, W. et al. A 3D path planning approach for quadrotor UAV navigation. In: *International Conference on Information and Automation, 2015 IEEE*. [S.l.: s.n.], 2015. p. 2481–2486.

[86] LUGO-CÁRDENAS, I. et al. Dubins path generation for a fixed wing UAV. In: *International Conference on Unmanned Aircraft Systems, ICUAS*. [S.l.: s.n.], 2014. p. 339–346.

[87] NABI-ABDOLYOUSEFI, R.; BANAZADEH, A. 3D offline path planning for a surveillance aerial vehicle using B-splines. In: *Proceedings of the 2013 International Conference on Advanced Mechatronic Systems*. [S.l.: s.n.], 2013. p. 306–311. ISSN 23250682.

[88] NETO, A. A.; MACHARET, D. G.; CAMPOS, M. F. M. Feasible path planning for fixed-wing UAVs using seventh order bézier curves. *Journal of the Brazilian Computer Society*, v. 19, n. 2, p. 193–203, 2013. ISSN 1678-4804.

[89] CHEN, X.; ZHANG, J. The three-dimension path planning of UAV based on improved artificial potential field in dynamic environment. In: *5th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC*. [S.l.: s.n.], 2013. v. 2, p. 144–147.

[90] QU, Y.; ZHANG, Y.; ZHANG, Y. Optimal flight path planning for UAVs in 3D threat environment. In: *International Conference on Unmanned Aircraft Systems, ICUAS*. [S.l.: s.n.], 2014. p. 149–155.

[91] WANG, D.; ZHANG, W.; SHAN, J. Optimal trajectory planning for a quadrotor via a gauss pseudo-spectrum method. In: *Ninth International Conference on Natural Computation, ICNC*. [S.l.: s.n.], 2013. p. 1666–1670. ISSN 21579555.

[92] ZHANG, Z.; XIE, P.; MA, O. Bio-inspired trajectory generation for UAV perching. In: *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. [S.l.: s.n.], 2013. p. 997–1002. ISSN 21596247.

[93] ZUQIANG, Y.; ZHOU, F.; PING, L. A bio-inspired collision-free 4D trajectory generation method for unmanned aerial vehicles based on tau theory. In: *Control Conference (CCC), 2015 34th Chinese*. [S.l.: s.n.], 2015. p. 6961–6968.

[94] DUBINS, L. E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, Johns Hopkins University Press, v. 79, n. 3, p. 497–516, 1957.

[95] LIU, J. et al. Trajectories planning for multiple UAVs by the cooperative and competitive pso algorithm. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.: s.n.], 2015. p. 107–114.