Controlador dinâmico para o problema linear quadrático com saltos não observados

Luiz Henrique Romero

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Luiz Henrique Romero

Controlador dinâmico para o problema linear quadrático com saltos não observados

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Eduardo Fontoura Costa

USP – São Carlos
Julho de 2019

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

R763c Romero, Luiz Henrique
 Controlador dinâmico para o problema linear
 quadrático com saltos não observados / Luiz
 Henrique Romero; orientador Eduardo Fontoura Costa.
 -- São Carlos, 2019.
 147 p.

 Dissertação (Mestrado - Programa de Pós-Graduação
 em Ciências de Computação e Matemática
 Computacional) -- Instituto de Ciências Matemáticas
 e de Computação, Universidade de São Paulo, 2019.

 1. Controle Ótimo. 2. Algoritmos Genéticos. 3.
 Sistemas Dinâmicos Estocásticos. 4. Cadeias de
 Markov. I. Costa, Eduardo Fontoura, orient. II.
 Título.

Luiz Henrique Romero

**Dynamic controller for the linear quadratic jump problem
without mode observation**

Master dissertation submitted to the Institute of
Mathematics and Computer Sciences – ICMC-USP,
in partial fulfillment of the requirements for the
degree of the Master Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Eduardo Fontoura Costa

USP – São Carlos
July 2019

AGRADECIMENTOS

Ao Prof. Dr. Eduardo Fontoura Costa pela oportunidade, atenção e exímia orientação neste trabalho. Obrigado pela amizade e incentivo à pesquisa.

A minha família, pelo incentivo e amparo em momentos de dificuldade; principalmente à minha mãe, Darci, pela paciência e apoio ao longo dos meus estudos.

Aos meus amigos de Tabapuã/SP, que alegraram muitos finais de semana com poker, truco, e, principalmente, com muita conversa boa e churrasco.

A todos os professores e funcionários do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP); obrigado por toda a infraestrutura fornecida.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio financeiro.

Enfim, à todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

*“Sucesso é ir de fracasso em fracasso
sem perder o entusiasmo.”
(Winston Churchill)*

RESUMO

ROMERO, L. H. **Controlador dinâmico para o problema linear quadrático com saltos não observados**. 2019. 147 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

Os Sistemas Lineares Sujeitos a Saltos Markovianos têm sido amplamente estudados nas últimas décadas, pois fornecem modelos adequados para aplicações com mudanças bruscas de comportamento, possivelmente devido à falhas. Também é muito comum em aplicações do “mundo real” em que o chamado *estado* do sistema não seja observado de forma perfeita e instantânea. Com essa motivação, consideramos o problema linear quadrático e propomos um controlador independente da variável de salto, que é um componente de estado, o que é atraente para aplicações reais. Utilizamos dois métodos clássicos, Genético e Gradiente, e propomos dois métodos híbridos que combinam características favoráveis de ambos. Também consideramos o caso em que não observamos o estado de Markov diretamente, mas através de uma variável, um sensor, que provê informação sobre este parâmetro.

Palavras-chave: Sistemas dinâmicos estocásticos, Cadeias de Markov, Algoritmos genéticos, Controle ótimo.

ABSTRACT

ROMERO, L. H. **Dynamic controller for the linear quadratic jump problem without mode observation**. 2019. 147 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

Markov Jump Linear Systems have been extensively studied in the last decades as they provide suitable models for applications featuring abrupt changes of behaviour. It is also quite common in “real world” applications that the so called *state* of the system is not perfectly and immediately observed. With this motivation, we consider the linear quadratic jump problem and we propose a controller that is irrespective of the jump variable (a component of the state), which is appealing for real world problems. We use classical Genetic and Gradient optimization methods and we also propose two hybrid methods, combining favorable features of both of them. We also consider the case which we do not have direct access on the Markovian jump parameter, but a variable, a sensor, which provides information on this parameter.

Keywords: Stochastic dynamic systems, Markov chains, Genetic algorithms, Optimal control.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de transição entre estados de um sistema	29
Figura 2 – Fluxograma exemplificando o funcionamento de um AG	34
Figura 3 – Método da roleta para seleção de indivíduos para <i>crossover</i> de um Algoritmo Genético.	40
Figura 4 – Gráfico típico de uma derivada numérica	50
Figura 5 – Custos obtidos com o AG50 vs. custos obtidos com AG100 - populações distintas	59
Figura 6 – Estatísticas gerais para populações distintas	59
Figura 7 – Custos obtidos com o AG50 vs. custos obtidos com AG100 - população replicada	61
Figura 8 – Estatísticas gerais para população replicada	61
Figura 9 – Custos obtidos com o AG x Custos obtidos com o MG - 10 segundos (realização única)	64
Figura 10 – Estatísticas AG x MG - 10 segundos (realização única)	64
Figura 11 – Custos obtidos com o AG x Custos obtidos com o MG - 1 minuto (realização única)	66
Figura 12 – Estatísticas AG x MG - 1 minuto (realização única)	66
Figura 13 – Custos obtidos com o AG x Custos obtidos com o MG - 10 minutos (realização única)	68
Figura 14 – Estatísticas gerais AG x MG - 10 minutos (realização única)	68
Figura 15 – Custos obtidos com o AG x Custos obtidos com o MG - 10 segundos (inúmeras realizações)	71
Figura 16 – Estatísticas AG x MG - 10 segundos (inúmeras realizações)	71
Figura 17 – Custos obtidos com o AG x Custos obtidos com o MG - 1 minuto (inúmeras realizações)	73
Figura 18 – Estatísticas AG x MG - 1 minuto (inúmeras realizações)	73
Figura 19 – Custos obtidos com o AG x Custos obtidos com o MG - 10 minutos (inúmeras realizações)	75
Figura 20 – Estatísticas AG x MG - 10 minutos (inúmeras realizações)	75
Figura 21 – Custos obtidos com o AG x Custos obtidos com o AGG - 10 gerações	96
Figura 22 – Estatísticas gerais AG x AGG - 10 gerações	97
Figura 23 – Custos obtidos com o AG x Custos obtidos com o AGG - 10 segundos	99
Figura 24 – Estatísticas gerais AG x AGG - 10 segundos	99

Figura 25 – Custos obtidos com o AG x Custos obtidos com o AGG - 1 minuto	101
Figura 26 – Estatísticas gerais AG x AGG - 1 minuto	101
Figura 27 – Custos obtidos com o AG x Custos obtidos com o AGG - 10 minutos	103
Figura 28 – Estatísticas gerais AG x AGG - 10 minutos	103
Figura 29 – Custos obtidos com o AG x Custos obtidos com o AGGL - 10 segundos	106
Figura 30 – Estatísticas gerais AG x AGGL - 10 segundos	107
Figura 31 – Custos obtidos com o AG x Custos obtidos com o AGGL - 1 minuto	108
Figura 32 – Estatísticas gerais AG x AGGL - 1 minuto	109
Figura 33 – Custos obtidos com o AG x Custos obtidos com o AGGL - 10 minutos	110
Figura 34 – Estatísticas gerais AG x AGGL - 1 minuto	111
Figura 35 – Custos obtidos com o AG x Custos obtidos com o AGE - 10 segundos	115
Figura 36 – Estatísticas gerais AG x AGE - 10 segundos	115
Figura 37 – Custos obtidos com o AG x Custos obtidos com o AGE - 1 minuto	117
Figura 38 – Estatísticas gerais AG x AGE - 1 minuto	118
Figura 39 – Custos obtidos com o AG x Custos obtidos com o AGE - 10 minutos	120
Figura 40 – Estatísticas gerais AG x AGE - 10 minutos	120
Figura 41 – Esquema de uma cadeia de Markov com dois estados.	133

LISTA DE TABELAS

Tabela 1 – Estatísticas gerais para populações distintas	60
Tabela 2 – Estatísticas específicas para populações distintas	60
Tabela 3 – Estatísticas gerais para população replicada	62
Tabela 4 – Estatísticas específicas para população replicada	62
Tabela 5 – Estatísticas gerais AG x MG - 10 segundos (realização única)	65
Tabela 6 – Estatísticas específicas AG x MG - 10 segundos (realização única)	65
Tabela 7 – Estatísticas gerais AG x MG - 1 minuto (realização única)	67
Tabela 8 – Estatísticas específicas AG x MG - 1 minuto (realização única)	67
Tabela 9 – Estatísticas gerais AG x MG - 10 minutos (realização única)	69
Tabela 10 – Estatísticas específicas AG x MG - 10 minutos (realização única)	69
Tabela 11 – Estatísticas gerais AG x MG - 10 segundos (inúmeras realizações)	72
Tabela 12 – Estatísticas específicas AG x MG - 10 segundos (inúmeras realizações)	72
Tabela 13 – Estatísticas gerais AG x MG - 1 minuto (inúmeras realizações)	74
Tabela 14 – Estatísticas específicas AG x MG - 1 minuto (inúmeras realizações)	74
Tabela 15 – Estatísticas gerais AG x MG - 10 minutos (inúmeras realizações)	76
Tabela 16 – Estatísticas específicas AG x MG - 10 minutos (inúmeras realizações)	76
Tabela 17 – Exemplo de redefinição de uma cadeia de Markov com 2 estados	86
Tabela 18 – Estatísticas gerais AG x AGG - 10 gerações	97
Tabela 19 – Estatísticas específicas AG x AGG - 10 gerações	97
Tabela 20 – Estatísticas gerais AG x AGG - 10 segundos	100
Tabela 21 – Estatísticas específicas AG x AGG - 10 segundos	100
Tabela 22 – Estatísticas gerais AG x AGG - 1 minuto	102
Tabela 23 – Estatísticas específicas AG x AGG - 1 minuto	102
Tabela 24 – Estatísticas gerais AG x AGG - 10 minutos	104
Tabela 25 – Estatísticas específicas AG x AGG - 10 minutos	104
Tabela 26 – Estatísticas gerais AG x AGGL - 10 segundos	107
Tabela 27 – Estatísticas específicas AG x AGGL - 10 segundos	107
Tabela 28 – Estatísticas gerais AG x AGGL - 1 minuto	109
Tabela 29 – Estatísticas específicas AG x AGGL - 1 minuto	109
Tabela 30 – Estatísticas gerais AG x AGGL - 10 minutos	111
Tabela 31 – Estatísticas específicas AG x AGGL - 10 minutos	111
Tabela 32 – Estatísticas gerais AG x AGE - 10 segundos	116
Tabela 33 – Estatísticas específicas AG x AGE - 10 segundos	116

Tabela 34 – Estatísticas gerais AG x AGE - 1 minuto	118
Tabela 35 – Estatísticas específicas AG x AGE - 1 minuto	118
Tabela 36 – Estatísticas gerais AG x AGE - 10 minutos	121
Tabela 37 – Estatísticas específicas AG x AGE - 10 minutos	121

LISTA DE ABREVIATURAS E SIGLAS

AE	Algoritmos Evolutivos
AG	Algoritmos Genéticos
AGE	Algoritmo Genético Evolutivo
AGG	Algoritmo Genético-Gradiente
AGGL	Algoritmo Genético Guloso
CMLP	Custo Médio a Longo Prazo
EARA	Equações Algébricas de Riccati Acopladas
FDP	Função Densidade Probabilidade
MG	Método Quasi-Gradiente
SLSMs	Sistemas Lineares Sujeitos a Saltos Markovianos

SUMÁRIO

1	INTRODUÇÃO	21
1.1	Informações relevantes	25
1.2	Estrutura da dissertação	25
2	FORMULAÇÃO DO PROBLEMA	27
2.1	Problema auxiliar	27
2.1.1	<i>Cálculo da solução analítica do custo</i>	29
2.2	O problema	31
3	ALGORITMOS GENÉTICOS	33
3.1	Conceitos básicos sobre Algoritmos Genéticos	33
3.2	Algoritmo Genético para o problema proposto	36
4	O MÉTODO QUASI-GRADIENTE	45
4.1	Método Quasi-Gradiente para o problema proposto	46
5	ANÁLISE DE RESULTADOS I	55
5.1	Definição das instâncias e dos resultados	56
5.2	Influência do tamanho da população inicial nos resultados do Algoritmo Genético	57
5.3	Algoritmo Genético versus Método Quasi-Gradiente	63
5.3.1	<i>Discussões sobre o Algoritmo Genético versus Método Quasi-Gradiente</i>	76
6	MÉTODOS ALTERNATIVOS	79
6.1	Métodos Híbridos	79
6.1.1	<i>Algoritmo Genético-Gradiente</i>	79
6.1.2	<i>Algoritmo Genético Guloso</i>	82
6.2	Algoritmo Genético Evolutivo	85
6.2.1	<i>Problema de controle com observação indireta</i>	85
6.2.2	<i>Algoritmo Genético Evolutivo para o problema proposto</i>	87
7	ANÁLISE DE RESULTADOS II	95
7.1	Algoritmo Genético versus Algoritmo Genético-Gradiente	95
7.1.1	<i>Discussões sobre o Algoritmo Genético versus Algoritmo Genético-Gradiente</i>	104

7.2	Algoritmo Genético versus Algoritmo Genético Guloso	105
7.2.1	<i>Discussões sobre o Algoritmo Genético versus Algoritmo Genético Guloso</i>	<i>112</i>
7.3	Considerações sobre o Algoritmo Genético versus Algoritmo Genético-Gradiente e entre o Algoritmo Genético versus Algoritmo Genético Guloso	113
7.4	Algoritmo Genético versus Algoritmo Genético Evolutivo	113
7.4.1	<i>Discussões sobre o Algoritmo Genético versus Algoritmo Genético Evolutivo</i>	<i>121</i>
8	CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	123
REFERÊNCIAS		125
APÊNDICE A CADEIAS DE MARKOV		131
A.1	Conceitos gerais	131
A.2	Classificação dos estados em cadeias de Markov	133
A.3	Exemplo de aplicação	134
APÊNDICE B RECURSOS COMPUTACIONAIS		137
B.1	Uso dos recursos computacionais	137
APÊNDICE C GERADOR DE SLSMS		143

INTRODUÇÃO

Os Sistemas Lineares Sujeitos a Saltos Markovianos (SLSMs) vêm sendo bastante estudados nas últimas décadas, sendo alvo de pesquisa em variadas áreas do conhecimento e um campo de pesquisa em constante expansão justamente por sua capacidade de modelar fenômenos que podem apresentar mudança brusca em seu comportamento em decorrência, por exemplo, de falhas.

Os SLSMs estão presentes em diversos segmentos da sociedade, desde suspensão automotiva, até os mais complicados e críticos, como aeronaves, satélites e usinas nucleares; aliás, com relação a usinas nucleares, uma análise sobre o desastre de Chernobyl sobre o ponto de vista da teoria de controle pode ser encontrado em (STEIN, 2003).

Como outros exemplo de tais sistemas, podemos citar o trabalho de (SARIDIS, 1983) em que nos é apresentado uma abordagem para trabalhar com sistemas robóticos que requerem tomada de decisões em ambientes imprevisíveis, juntamente com o trabalho de (SIQUEIRA; TERRA, 2004); modelos macroeconômicos (VAL; BAŞAR, 1999); mecânica (SIQUEIRA; TERRA, 2004); políticas monetárias (COSTA; ARAUJO, 2008); produção de energia, no qual podemos citar o trabalho de (SWORDER; ROGERS, 1983), que se foca no problema de controle de temperatura de um sistema de geração de energia através do sol; controle de aeronaves (BLACKMORE *et al.*, 2010), (STOICA; YAESH, 2002), (TUGNAIT, 1983); modelos de transmissão em redes de telecomunicação (GONÇALVES; FIORAVANTI; GEROMEL, 2011), (GONÇALVES, 2012); controladores em rede de telecomunicação (PALMA *et al.*, 2016); entre outros.

Nas últimas décadas, o esforço em tornar os sistemas de controle mais confiáveis tem aumentado, principalmente no que diz respeito a sistemas de controle tolerantes a falhas (PATTON, 1997), (ZHANG; JIANG, 2008), (HWANG *et al.*, 2010) e (JIANG; YU, 2012). Nesse contexto, uma “ponte” entre esse tipo de sistema e SLSMs pode ser encontrada nos trabalhos de (MARITON, 1989), (SRICHANDER; WALKER, 1993) e (MUFEED; JIN; YOUNG, 2013), e

nas referências neles encontradas.

Os parâmetros dos SLSMs são governados pelo estado $\theta(t) \in \mathcal{N} = \{1, 2, \dots, N\}$ de uma cadeia de Markov, ou seja, em cada instante de tempo t , o sistema mudará seu modo de operação com base em θ . A formulação de um SLSM dada por

$$\mathbf{M} = \begin{cases} x(t+1) = A(\theta(t))x(t) + B(\theta(t))u(t) + B_w(\theta(t))w(t), \\ y(t) = C_y(\theta(t))x(t) + D_w(\theta(t))w(t), \\ z(t) = x(t)'C_z(\theta(t))x(t) + u(t)'D_u(\theta(t))u(t), \end{cases} \quad (1.1)$$

é semelhante a um sistema linear determinístico, porém o par (x, θ) representa o *estado do sistema* (enquanto em sistemas sem saltos o estado é somente o vetor x); $z(t)$ é o *custo por estágio* ou seja, o índice de desempenho para cada estágio t ; $y(t)$ é a saída observada do sistema; $u(t)$ é o controle do sistema e consiste em uma variável que podemos ajustar com base em algum critério pré-determinado; $w(t)$ é um vetor aleatório que representa o ruído aditivo atuante no sistema e $A(\theta(t))$, $B(\theta(t))$, $B_w(\theta(t))$, $C_y(\theta(t))$, $D_w(\theta(t))$, $C_z(\theta(t))$ e $D_u(\theta(t))$ são matrizes de dimensões apropriadas que representam parâmetros fixos do sistema que são indexadas por $\theta(t)$.

Esse tipo de característica, na qual a distribuição do estado do sistema no instante seguinte depende apenas do estado atual, leva o nome de *Processo Estocástico Markoviano*. Esta propriedade possui várias características interessantes e mais resultados podem ser encontrados em (COSTA; FRAGOSO; MARQUES, 2005), (COSTA; FRAGOSO; TODOROV, 2013) (DRAGAN; MOROZAN; STOICA, 2009), (VARGAS; COSTA; VAL, 2016).

Em situações em que seja imprescindível que determinado sistema opere atendendo a padrões de desempenho exigentes, e como exemplo podemos citar o funcionamento de um satélite em órbita terrestre onde se faz necessária a correção de sua trajetória devido ao seu decaimento natural ao longo do tempo, a atuação de um controlador é crucial; quanto mais o sistema desvia-se do ideal, mais “cara” é sua operação (pelo afastamento do ideal) e mais a ação de controle se faz necessária levando, também, a um controle mais “caro”; o custo desse afastamento é ligado aos termos C_z e D_u , apresentados em (1.1). Retornando ao satélite, o termo $u(t)'D_u(\theta(t))u(t)$ pode ser interpretado como a quantidade de recursos necessária para manter sua órbita, como combustível ou energia provinda de painéis solares, por exemplo; portanto, quanto menor o custo do controlador, melhor ele será. Já o termo $x(t)'C_z(\theta(t))x(t)$ é interpretado como uma penalização ao sistema afastar-se da origem (origem aqui representa o ponto de operação do satélite em um sistema de coordenadas apropriado), de forma que quanto mais próximo da origem, mais próximo do ponto ideal de operação. Há uma tendência natural que esses termos sejam conflitantes em termos da solução de controle - por exemplo, o controle nulo $u(t) = 0$ minimiza o termo $u(t)'D_u(\theta(t))u(t) = 0$, contudo, pode ter efeito “catastrófico” no outro termo, levando o satélite a cair e o termo $x(t)'C_z(\theta(t))x(t)$ tender ao infinito. Um tipo de controlador que tem sido utilizado para buscar esse compromisso é o chamado *controlador*

dinâmico que atua com o sistema (1.1):

$$\mathbf{M}_c = \begin{cases} x_c(t+1) = A_c(\theta(t))x_c(t) + B_c(\theta(t))y(t), \\ u(t) = C_c(\theta(t))x_c(t) + D_c(\theta(t))y(t). \end{cases} \quad (1.2)$$

As saídas no controlador (1.2) alimentam as entradas do sistema em (1.1), e com esse acoplamento, obtemos um “novo sistema”, com uma dinâmica que depende das matrizes de $\mathbf{M}_c$ e com um “novo custo” associado.

Quanto menor o custo, melhor o desempenho do sistema, e, portanto, o problema abordado neste trabalho trata-se de um problema de otimização. Na matemática, otimizar significa maximizar ou minimizar valores de uma função dado determinado conjunto de parâmetros. Esses problemas podem ser classificados em *determinísticos* ou *estocásticos*. Em problemas determinísticos, todas as variáveis de decisão são previamente conhecidas antes da ação de controle ser determinada. Já em problemas estocásticos, caso abordado neste trabalho, os parâmetros são aleatórios, o que faz com que as decisões a serem tomadas dependam de probabilidades (KUMAR; VARAIYA, 2015). Utilizando como exemplo o SLSM apresentado no início desse capítulo, podemos reescrever o problema, obtendo:

$$\min_{\mathbf{M}_c} \sum_{t=1}^{\tau} E[z(t)^2], \quad (1.3)$$

sujeito a (1.1), (1.2).

Como vemos, o problema de otimização consiste em minimizar o somatório do custo por estágio t , que é chamado de *problema linear quadrático com saltos nos parâmetros*. Tal problema também pode ser visto como uma variação do problema do controlador dinâmico com observação completa de estado considerado em (DRAGAN; COSTA, 2016) e detalhes serão abordados posteriormente.

Neste trabalho, enfocamos em problemas envolvendo grandes períodos de tempo, como no caso mencionado anteriormente de um satélite que deve operar ininterruptamente por tempo “infinito”, o que normalmente faz com que a função objetivo tenda ao infinito na medida em que τ cresce. Nesses casos, seguindo a literatura, é considerada a versão chamada de “Custo Médio a Longo Prazo (CMLP)”,

$$\min_{\mathbf{M}_c} \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \sum_{t=0}^{\tau} E[z(t)^2], \quad (1.4)$$

em que a função objetivo tem a interpretação da média ao longo dos estágios τ , com $\tau \rightarrow \infty$. Quando um sistema possui ruído aditivo e existe interesse em operá-lo a longo prazo, não podemos calcular o limite do somatório do custo $z(t)$ do sistema, pois $z(t)$ não tende a zero (por ação do ruído) e a soma tende ao infinito, e nesse caso faz-se necessário somar o custo no tempo e dividi-lo pelo horizonte, por isso o nome “Custo Médio a Longo Prazo”. Contudo, existem

situações em que um sistema pode possuir ruído aditivo e ser resolvido em horizonte finito sem precisar de CMLP e isso independe da observação de θ . Vale ressaltar que o interesse em CMLP está associado a problemas de horizonte grandes em sistemas estocásticos, independentemente da questão da observação de θ ; voltando ao exemplo do satélite, é obvio que se deseja operá-lo por grandes períodos de tempo, nos quais custos como (1.3) tenderiam ao infinito, com ou sem observação da variável de salto.

Os SLSMs podem ser divididos em três casos no que tange a observação da variável θ . O primeiro caso é o *caso de observação completa*, em que, em suas devidas proporções, os problemas são facilmente resolvidos através de Equações Algébricas de Riccati Acopladas (EARA) (um estudo detalhado pode ser encontrado em (DRAGAN; COSTA, 2016)), de forma que em cada instante de tempo t , o controle mudará seu modo de operação com base na variável de salto θ - o sistema mudará seu modo independentemente de observar θ ou não; já o controle tem essa capacidade somente no caso com observação. Para o segundo e terceiro casos, *caso de observação parcial* e *caso sem observação*, respectivamente, não há ainda um método “perfeito” que obtenha soluções para quaisquer problemas, pois nesses casos o problema torna-se demasiadamente complexo e perde-se unicidade de solução (OLIVEIRA; COSTA, 2015).

Considerando que em muitos problemas práticos é difícil sabermos o estado de θ , ou seja, a variável θ nem sempre é conhecida ou disponível de maneira instantânea e sem erros (talvez exceto em condições laboratoriais), faz-se necessário o estudo de novos métodos que levem em consideração esse tipo de adversidade. Portanto, neste trabalho, propomos um controlador que independe da variável θ . Assim sendo, considerando todos os elementos acima, podemos agora escrever o problema de otimização que será considerado ao longo de todo trabalho:

$$\begin{aligned} \min_{A_c, B_c, C_c, D_c} \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \sum_{t=0}^{\tau} E[z(t)^2]. \\ \text{sujeito a (1.1),} \end{aligned} \tag{1.5}$$

$$\begin{cases} x_c(t+1) = A_c x_c(t) + B_c y(t), \\ u(t) = C_c x_c(t) + D_c y(t). \end{cases}$$

Note que o controlador que compõe o problema em (1.5) é o mesmo controlador apresentado em (1.2), com a exceção que deverá operar sem a observação da variável de salto θ - o sistema ainda opera mudando seu modo de operação com base em θ , enquanto o controlador deverá atuar independentemente de tal variável.

Além disso, avançamos em relação ao estado da arte, sendo a principal contribuição a generalização do problema tratado em (DOLGOV, 2017) e a obtenção de algoritmos capazes de encontrar soluções ótimas, enquanto os algoritmos em (DOLGOV, 2017) buscam por soluções sub-ótimas. Em relação ao problema de controle ótimo resolvido em (DRAGAN; COSTA, 2016), os problemas tratados aqui são equivalentes por incluir restrições de igualdade da forma $A_c(i) = A_c$, $i \in \mathcal{N}$ e similarmente para as demais matrizes do controlador, o que inviabiliza a

caracterização da solução em termos de equações de Riccati, o que torna o problema bem mais complexo, como veremos oportunamente.

1.1 Informações relevantes

Embora uma das principais motivações do controlador abordado neste trabalho seja a relativa facilidade de implementação em problemas reais (já que não precisa de sensores para medir θ), todos os testes realizados neste trabalho foram realizados em ambiente virtual, por simulações, dada a rapidez, baixo custo e segurança.

Todos os algoritmos foram desenvolvidos utilizando o MATLAB[®] (*MATrix LABoratory*) na versão R2015a, software de alto nível voltado para análise numérica, cálculo com matrizes, etc. O MATLAB[®] foi criado no final dos anos 70 e início dos anos 80 por Clive Moler, enquanto professor de Álgebra Linear e Análise Numérica na Universidade do Novo México. O projeto, que começou apenas como passatempo, evoluiu para abranger diversos aspectos da computação científica (MOLER, 2018).

Outro adendo é o fato de que no Capítulo 3 descreveremos o funcionamento e implementação de um Algoritmo Genético, um dos métodos utilizados para a resolução do problema tratado neste trabalho. Embora o MATLAB[®] possua uma “*toolbox*” para a utilização de Algoritmos Genéticos, optamos por não utilizá-la e desenvolver o nosso próprio código.

1.2 Estrutura da dissertação

Este trabalho está organizado da seguinte forma: no Capítulo 2 introduzimos o problema estudado neste trabalho; nos Capítulos 3 e 4 apresentamos os conceitos gerais dos métodos utilizados para resolver o problema proposto, além de descrevermos como tais conceitos foram aplicados; no Capítulo 5 apresentamos uma análise dos resultados obtidos com ambos os métodos descritos nos dois capítulos anteriores; no Capítulo 6 apresentamos as motivações por estudo de métodos alternativos, derivados a partir dos métodos “originais”, além de um novo método, um Algoritmo Genético, em essência, mas com uma abordagem baseada em “níveis de transição” entre o problema auxiliar e o problema estudado neste trabalho; no Capítulo 7 apresentamos uma análise dos resultados obtidos com os métodos alternativos; por fim, no Capítulo 8 são apresentadas as conclusões e propostas para trabalhos futuros.

FORMULAÇÃO DO PROBLEMA

Essa seção descreve o problema de custo médio a longo prazo com observação da variável de salto, a qual também nos referimos como *problema auxiliar*, cuja solução já é conhecida na literatura (DRAGAN; COSTA, 2016). Optamos por apresentá-lo no texto devido ao seu na inicialização dos métodos que serão apresentados no decorrer deste trabalho, e por também ajudar na compreensão e comparação com o problema estudado nessa dissertação.

2.1 Problema auxiliar

Seja o sistema $\mathbf{G}$, extraído de (DRAGAN; COSTA, 2016),

$$\mathbf{G} = \begin{cases} x(t+1) = A(\theta(t))x(t) + B(\theta(t))u(t) + B_w(\theta(t))w(t), \\ y(t) = C_y(\theta(t))x(t) + D_w(\theta(t))w(t), \\ z(t) = C_z(\theta(t))x(t) + D_u(\theta(t))u(t), \quad t \geq 0, \end{cases} \quad (2.1)$$

em que $x(t) \in \mathbb{R}^n$ é o vetor de estado do sistema; $u(t) \in \mathbb{R}^m$ é a variável de controle; $z(t) \in \mathbb{R}^{n_y}$ é custo por estágio e $y(t) \in \mathbb{R}^{n_z}$ é o parâmetro de saída do sistema que deve ser controlado; e $\{\theta(t)\}_{t \geq 0}$ e $\{w(t)\}_{t \geq 0}$ são processos estocásticos que satisfazem as seguintes suposições:

Suposição 1. $\{\theta(t)\}_{t \geq 0}$ é uma Cadeia de Markov que pode assumir um conjunto finito de valores $\mathcal{N} = \{1, 2, \dots, N\}$ e que possui uma *matriz de probabilidade de transição* $\mathbb{P} = (p_{ij}) \in \mathbb{R}^{n \times n}$, ou seja, $p_{ij} = (\theta(t+1) = j | \theta(t) = i)$ e dada uma *distribuição inicial* $\pi_0 = (\pi_0(1), \dots, \pi_0(N))$.

Suposição 2. $\{w(t)\}_{t \geq 0}$ é uma sequência de vetores independentes e aleatórios com média zero que satisfaz:

$$E[w(t)w^T(t)] = \mathbf{W}, t \geq 0. \quad (2.2)$$

$\{w(t)\}$ é o ruído aditivo atuante no sistema.

Imagine uma antena que rastreia um satélite, rastreamento esse realizado por um servomotor. Os mecanismos têm imprecisões (folgas, desgastes), a tensão aplicada no motor sofre interferências, a antena sofre efeito do vento, etc. Essas interferências podem ter comportamento muito complexo, originando processos estocásticos muito complicados de modelar de forma bem precisa, e tratar matematicamente. Por isso, é muito comum modelar os efeitos nos sistemas simplesmente através de um termo $w(t)$ no estado, independente do processo θ ; muitas vezes, a covariância considerada é um pouco maior que a estimada, dando certa compensação a essas hipóteses simplificadoras.

Suposição 3. $\{\theta(t)\}_{t \geq 0}$ e $\{w(t)\}_{t \geq 0}$ são processos estocásticos independentes.

O custo $\mathbf{J}$, ou medida de desempenho, do sistema $\mathbf{G}$, que denotaremos $\mathbf{J}(\mathbf{G}_c)$, é dado por:

$$J(\mathbf{G}_c) = \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \sum_{t=0}^{\tau} E[z(t)^2]. \quad (2.3)$$

Dito isto, vamos considerar o seguinte controlador para o sistema apresentado em (2.1),

$$\mathbf{R}_0 = \begin{cases} x_c(t+1) = A_c(\theta(t))x_c(t) + B_c(\theta(t))y(t), \\ u(t) = C_c(\theta(t))x_c(t) + D_c(\theta(t))y(t), \end{cases} \quad (2.4)$$

que são chamados de controladores dinâmicos com observação de θ .

Controladores dinâmicos possuem uma dinâmica interna, um estado interno, que nesse contexto chamamos de x_c , como em (2.4) e que dispensam a observação direta da variável x . Como exemplo, voltemos ao satélite: se em órbita estacionária, deve manter velocidade constante para permanecer sempre em um mesmo ponto em relação à Terra; se $x(t) = [v(t) \ h(t)]'$ representa a velocidade e altitude do satélite, mas apenas a velocidade é medida através de $y(t) = v(t) + D_w(\theta(t))w(t)$, então não é possível tomar decisões com base em x , mas apenas com base na medida imprecisa $y(t)$. O controlador dinâmico faz exatamente isso e é capaz, inclusive, de estimar $x(t)$ através da variável interna $x_c(t)$; também tem a capacidade de operar de maneira mais simples, desconsiderando essa estimativa, fazendo A_c , B_c e C_c nulos, tomando decisões na forma $u(t) = D_c(\theta(t))y(t)$. Em contraste, temos os *controladores estáticos* que não possuem uma dinâmica interna, podendo, por exemplo, ser uma realimentação linear estática de estado na forma de $u(t) = Kx(t)$, como em (SILVA, 2012), que não necessitam de x . Esse tipo de controlador tem suas vantagens, como o pouco uso de memória computacional, já que não precisam armazenar diferentes matrizes, sendo portanto, mais simples do que em (2.4), mas por outro lado, levam a maiores custos (pior desempenho). Entretanto, também existem controladores estáticos na forma de $u(t) = Ky(t)$ em que não é necessária a observação de x , que, embora sejam mais simples de implementar, são extremamente complicados de se equacionar.

Podemos imaginar o controle como uma “chave” que altera o modo de operação do sistema. A Figura 1 exemplifica o funcionamento do sistema que transita entre dois modos:

Figura 1 – Exemplo de transição entre estados de um sistema

O objetivo é obter um controlador que minimize $J(\mathbf{R}_0)$, ou seja,

$$\begin{aligned} J(\tilde{\mathbf{R}}_0) &= \min_{\mathbf{R}_0} J(\mathbf{R}_0) \\ &= \min_{\mathbf{R}_0} \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \sum_{t=0}^{\tau} E[z(t)^2]. \end{aligned} \quad (2.5)$$

Note que o problema envolve

$$\begin{cases} N \text{ matrizes } A_c, \\ N \text{ matrizes } B_c, \\ N \text{ matrizes } C_c, \\ N \text{ matrizes } D_c, \end{cases} \quad (2.6)$$

que tipicamente são armazenadas na memória de um controlador (como por exemplo, um Arduino); o controlador, a cada instante de tempo, precisa receber o valor observado de $\theta(t)$, buscar as matrizes $A_c(\theta(t))$, $B_c(\theta(t))$, $C_c(\theta(t))$ e $D_c(\theta(t))$ da memória, realizar os cálculos em (2.4) para finalmente obter a decisão $u(t) = y(t)$. Há aplicações em que isso não é possível, ou por limitações de armazenamento e/ou disponibilidade de $\theta(t)$, como já comentamos anteriormente. Isso nos motiva a buscar um controlador mais simples, como será descrito.

2.1.1 Cálculo da solução analítica do custo

Seja $\mathcal{Y} = (Y_1, \dots, Y_N)$ um conjunto de matrizes tal que $\mathcal{Y}$ é a única solução da equação linear

$$\mathcal{Y} = \mathcal{L}(\mathcal{Y}) + \mathcal{B}, \quad (2.7)$$

sendo que $\mathcal{B} = (B_1, \dots, B_N) \in \mathbb{R}^{n \times n}$ é um conjunto de matrizes dado por

$$B_i = \sum_{j=1}^N p_{ji} \rho_j B_{cl,j} \mathbf{W} B_{cl,j}^T, \quad (2.8)$$

e com o operador $\mathcal{L}$ que mapeia conjunto de matrizes em conjunto de matrizes, definido de forma que

$$\mathcal{L}_i(\mathcal{Y}) = \sum_{j=1}^N p_{ji} A_{cl,j} Y_j A_{cl,j}^T, \quad 1 \leq i \leq N. \quad (2.9)$$

Note que o argumento de $\mathcal{L}$ é um conjunto de matrizes por isso não leva o “i”; ρ é a *distribuição Cesàro limite*, dada por

$$\rho = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^t \mathbb{P}^k \pi(0), \quad (2.10)$$

que pode ser interpretada como a distribuição “média a longo prazo” da cadeia de Markov quando $t \rightarrow \infty$, e seu cálculo depende de $\mathbb{P}$ e π . Por exemplo, tome $\mathbb{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ e $\pi(0) = [1 \ 0]$, de forma que $\theta(t) = 1$ (quase certamente) para t par, e $\theta(t) = 2$ (quase certamente) para t ímpar. A distribuição limite $\lim_{t \rightarrow \infty} \pi(t)$ não existe, mas a *distribuição Cesàro limite* sim, dada por $\rho = [\frac{1}{2} \ \frac{1}{2}]$. A interpretação é que se tomarmos um instante de tempo ao acaso com distribuição uniforme no intervalo $[T \ 2T]$, com T grande, então $P(\theta(t) = 1) = \frac{1}{2}$.

Em (DRAGAN; COSTA, 2016) foi mostrado o seguinte resultado.

Lema 1.

$$J(\mathbf{G}_c) = \sum_{i=1}^N \text{Tr}[C_{cl,i} \mathbf{Y}_i C_{cl,i}^T] + \sum_{j=1}^N \rho_j \text{Tr}[D_{cl,j} \mathbf{W} D_{cl,j}^T]. \quad (2.11)$$

O conjunto de matrizes de malha fechada $A_{cl,i}$, $B_{cl,i}$, $C_{cl,i}$ e $D_{cl,i}$ que aparecem nas Equações (2.11) e (2.9) são definidas por

$$A_{cl,i} = \begin{pmatrix} A_i + B_i D_{c,i} C_{y,i} & B_i C_{c,i} \\ B_{c,i} C_{y,i} & A_{c,i} \end{pmatrix}, \quad (2.12)$$

$$B_{cl,i} = \begin{pmatrix} B_{w,i} + B_i D_{c,i} D_{w,i} \\ B_{c,i} D_{w,i} \end{pmatrix}, \quad (2.13)$$

$$C_{cl,i} = \begin{pmatrix} C_{z,i} + D_{u,i} D_{c,i} C_{y,i} & D_{u,i} C_{c,i} \end{pmatrix} \text{ e} \quad (2.14)$$

$$D_{cl,i} = D_{u,i} D_{c,i} D_{w,i} \text{ com } i \in \mathcal{N}. \quad (2.15)$$

Observação 1. Computacionalmente, a implementação da Equação (2.11) merece destaque. Como vemos, tal equação faz uso das matrizes A_{cl} , B_{cl} , C_{cl} e D_{cl} bem como $\mathcal{Y}$, solução da equação linear (2.7), o que, por sua vez, depende da Equação (2.8). Como $\mathcal{Y}$ é o ponto fixo da recursão (2.7), temos basicamente duas formas de resolver - a primeira, recursivamente através de (2.7), iterando até que um número máximo de iterações seja atingido, ou o erro relativo entre duas iterações, Equação (2.16) onde Tr é a soma dos traços das matrizes que compõem a coleção $\mathcal{Y}_{t+1} - \mathcal{Y}_t$, seja menor do que 10^{-8} , por considerarmos uma medida suficientemente pequena. Segunda, transformar (2.16) em um sistema linear convencional “ $Ax = b$ ” usando operadores de Kronecker e de empilhamento de colunas de matrizes e em seguida resolvendo o sistema no MATLAB[®]. Detalhes sobre operadores de Kronecker e sobre essa implementação podem ser encontrados no Apêndice B.

Para o primeiro caso, calcularemos o *erro relativo* entre dois instantes consecutivos e se tal erro for menor do que 10^{-8} , encerraremos a execução. Neste trabalho,

$$Err = \frac{Tr(\mathcal{Y}_{t+1} - \mathcal{Y}_t)}{Tr(\mathcal{Y}_t)}, \quad (2.16)$$

em que Tr é o traço matricial.

Neste trabalho, ambas implementações foram utilizadas. Primeiramente, tentamos a solução através de um sistema linear convencional “ $Ax = b$ ” (como demonstrado no Apêndice B, Equação B.4), por não necessitar de recursão para ser resolvida, e, portanto, ser menos custosa computacionalmente. Entretanto, essa implementação pode retornar resultados negativos (custos negativos), o que, no contexto deste trabalho, não faz sentido. Caso isso ocorra, ignoraremos o resultado provindo da resolução desse sistema linear e ativaremos a função recursiva apresentada em (2.7).

Note que os custos negativos provindos através da resolução de um sistema linear não significa que a resolução de tal sistema está incorreta, mas que tais resultados, quando analisados no contexto deste trabalho (custos, ou medida de desempenho, para SLSMs), são inválidos.

2.2 O problema

Como exposto no Capítulo 1, há motivações de natureza prática para o estudo do problema considerando que θ não é observado. Por exemplo, o controlador em (2.4) pode demandar muito esforço computacional (em especial, memória) para uma determinada aplicação, sendo necessário um controlador mais simples. Também há casos em que θ não é observado ou é observado com atraso devido à comunicação entre o controlador e os sensores, ou ainda θ pode conter erros, como tipicamente em casos em que ele é estimado usando alguma técnica de estimação ou detecção (a propósito, alguns resultados recentes têm sido desenvolvidos no contexto (COSTA; FRAGOSO; TODOROV, 2013), etc.). Em razão disso, é interessante adaptar o problema apresentado na Seção 2.1 para o caso sem observação de θ .

Nesse caso, consideramos uma classe de controladores $\mathbf{R}_n \subset \mathbf{R}_o$ para que atue sem a necessidade de observação da variável θ , obtendo

$$\mathbf{R}_n = \begin{cases} x_c(t+1) = A_c x_c(t) + B_c y(t), \\ u(t) = C_c x_c(t) + D_c y(t), \end{cases} \quad (2.17)$$

em que teríamos somente uma matriz A_c , uma matriz B_c , uma matriz C_c e uma matriz D_c que minimizam o custo do sistema. Portanto, o objetivo deste trabalho é encontrar o controlador em (2.17) que minimiza o custo da Equação (2.3).

As matrizes de malha fechada A_{cl} , B_{cl} , C_{cl} e D_{cl} são obtidas de forma semelhante a (2.12)-(2.15), exceto que em todas as fórmulas retiramos os índices i das matrizes do controlador.

Equivalentemente, podemos utilizar as mesmas fórmulas e “replicar” as matrizes do controlador, por exemplo $A_c(i) = A_c$ para todo i , apenas para efeito de cálculo de malha fechada.

Embora o problema proposto se baseie na não observação dos estados de Markov, o sistema a ser controlado continuará o mesmo, assim como apresentado em (2.1), a mudança dá-se na maneira de funcionamento do controlador, que deverá operar sem a observação dos estados. Note que ainda estamos lidando com um *controlador dinâmico*.

Obviamente, nosso controlador tem grande vantagem em relação aos controladores com observação de θ ; podem ser usados em aplicações em que θ não é observado ou é observado com algum tipo de imprecisão, ou mesmo em problemas em que θ é observado, mas deseja-se um controlador menos complexo para implementação.

Analogamente, a grande vantagem em relação aos controladores estáticos é que nosso controlador os generaliza, na medida que podemos recuperá-los com escolhas particulares das matrizes A_c , B_c , C_c e D_c ; assim, temos um maior “grau de liberdade”, o que pode levar a melhor desempenho. Em resumo, nosso controlador tem complexidade e desempenho intermediários entre os controladores dinâmicos com observação de θ e os controladores estáticos sem observação de θ .

Como já dito, em situações em que a variável θ é observável, os problemas são, em suas devidas proporções, facilmente resolvidos através das EARA. Não é possível obter EARA no caso sem observação de θ , tendo em vista resultados em (VARGAS; COSTA; VAL, 2016) e (OLIVEIRA; COSTA, 2015) válidos para o controlador estático, que é um caso particular do que estudamos aqui.

O problema estudado neste trabalho difere em relação ao problema encontrado em (DOLGOV, 2017) por possuir uma matriz a mais no controlador. Em (DOLGOV, 2017), considera-se apenas

$$\mathbf{R}_{nh} = \begin{cases} x_c(t+1) = A_c x_c(t) + B_c y(t), \\ u(t) = C_c x_c(t). \end{cases} \quad (2.18)$$

A falta da matriz D_c em (DOLGOV, 2017) traz uma pequena vantagem em termos de uso de memória, mas grande desvantagem de deixar de incluir controladores estáticos como um caso particular: na nossa formulação podemos obter solução $C_c = 0$ (e D_c não trivial), o que significa que x_c é irrelevante, levando $u(t) = D_c y(t)$, recuperando, assim, o controlador estático que é mais simples de implementar.

Vale mencionar que, além da diferença de formulação explicada acima em relação a (DOLGOV, 2017), aqueles autores não buscam soluções ótimas, mas apenas minimizam um limitante superior para o custo, o que pode levar a custos muito superiores (pior desempenho).

ALGORITMOS GENÉTICOS

Objetivando soluções para o Problema, descrito na Seção 2.2, um dos métodos abordados baseia-se em Algoritmos Genéticos (AG).

Durante os últimos anos, o interesse em resolver problemas de otimização utilizando princípios baseados em evolução e em hereditariedade tem aumentado devido à ampla gama de problemas cuja solução não é trivial, seja por intratabilidade matemática ou mesmo ausência de informação acerca do problema necessários aos algoritmos “clássicos”. Visando novos métodos e técnicas para esse tipo de problema, tem-se em AGs abordagens eficientes para busca de soluções ótimas ou sub-ótimas, pois não possuem limitações encontradas em técnicas de busca tradicionais (CARVALHO, 2009).

AGs são uma classe particular de Algoritmos Evolutivos (AE), pois utilizam técnicas baseadas em seleção natural para resolver problemas de otimização (MITCHELL, 1996). O conceito teve início com o cientista da computação e psicólogo estadunidense John Henry Holland que desenvolveu sua teoria não apenas para resolver problemas de otimização, mas também para estudar sistemas auto-adaptativos em processos biológicos. Esse é um dos motivos que fazem com que AGs aproximam-se muito mais de modelos biológicos do que outras classes de AEs (AFFENZELLER *et al.*, 2009).

Os AGs possuem ampla escala de aplicação e têm obtido ótimos resultados em diversas áreas do conhecimento, entre as quais podem ser citados problemas de otimização, aprendizado de máquina, análise de modelos econômicos, problemas de engenharia e diversas aplicações na Biologia, como simulação de bactérias e sistemas imunológicos (MITCHELL, 1996).

3.1 Conceitos básicos sobre Algoritmos Genéticos

Inspirado pela Teoria da Evolução das Espécies de Charles Darwin, Holland (HOLLAND, 1975) dividiu o funcionamento dos AGs em etapas de *inicialização*, *avaliação*, *seleção*,

cruzamento (crossover), mutação, atualização e finalização.

Basicamente, o que um AG faz é criar uma população de possíveis soluções (indivíduos) para o problema e depois submetê-las aos seguintes processos, que privilegiarão os melhores resultados:

1. **Avaliação ou *fitness*:** avalia-se o quão bem os indivíduos da população respondem ao problema proposto.
2. **Seleção:** indivíduos que se saíram bem na etapa de *avaliação* são selecionados para a reprodução. A probabilidade de determinado indivíduo ser selecionado é proporcional ao seu desempenho na *avaliação*.
3. **Cruzamento ou *crossover*:** características presentes nos indivíduos selecionados na etapa de *seleção* são recombinadas, gerando, assim, novos indivíduos. Em outras palavras, “indivíduos pais/mães” geram “indivíduos filhos”. Na biologia, o *crossover* é a reprodução, característica pertinente a todos os seres vivos e fundamental para a manutenção da espécie, uma vez que, nas condições atuais, seres vivos só surgem de outros seres vivos da mesma espécie por meio da reprodução.
4. **Mutação:** características presentes nos “indivíduos filhos” são modificadas a fim de adicionar variabilidade genética à população.
5. **Avaliação:** avalia-se o quão bem os “indivíduos filhos” respondem ao problema proposto e os insere na população - e sim, a etapa de **avaliação** faz-se necessária tanto para avaliarmos a população inicial quanto para avaliarmos cada um dos indivíduos “filhos” gerados via **crossover**.
6. **Finalização:** verifica-se as condições de evolução foram atingidas, encerrando a execução do algoritmo em caso de resposta positiva ou retornando à etapa de *seleção* em caso de resposta negativa.

Figura 2 – Fluxograma exemplificando o funcionamento de um AG

A ideia básica de um AG é tratar cada possível solução do problema que está sendo estudado como “indivíduos” em uma “população” que “evoluirá” entre cada “geração” (iteração no algoritmo).

A geração de indivíduos que irão compor a população inicial requer atenção. Tais indivíduos são “soluções” possíveis para o problema considerado.

O tamanho da população também pode influenciar no desempenho de um AG; se a população for composta por poucos indivíduos, perde-se a variabilidade genética, diminuindo as chances do algoritmo encontrar uma boa solução, pois diminui-se a cobertura do espaço de busca do problema; se a população for muito grande, perde-se a eficiência, por ser necessário avaliar um grande conjunto de dados, levando à necessidade de maiores recursos computacionais.

A escolha da representação para os indivíduos também exerce papel importante. Na literatura, é comum os indivíduos serem representados por vetores binários (genótipos), onde cada elemento (0 ou 1) indica a ausência ou presença, respectivamente, de determinada característica e a recombinação entre os indivíduos constitui as características dos indivíduos, ou seja, seu *fenótipo*. Há também a representação via *ponto flutuante*, em que cada posição do vetor é composta por um número real, que representam parâmetros do problema estudado, sejam eles quais forem. Segundo (MICHALEWICS, 1996) e (DEB, 2001), a *representação binária* é fraca para problemas de otimização numérica com parâmetros reais e que requeiram precisão; além disso, descrevem simulações computacionais comparando ambas representações, ou seja, simulações comparando os resultados obtidos quando os indivíduos são representados via *ponto flutuante* e via *representação binária* (vetores binários); claro que o problema submetido ao Algoritmo Genético pode ditar o tipo de representação dos indivíduos, dado que, determinada representação pode não descrever com o maior nível de exatidão possível cada indivíduo, e isso pode exercer influência negativa nos resultados finais. Neste trabalho, utilizamos a representação via *ponto flutuante* e detalhes serão abordados posteriormente.

O *crossover*, como exposto anteriormente, é responsável por recombinar os indivíduos pais/mães e gerar novos indivíduos. Existem vários métodos utilizados para o *crossover* e uma breve descrição sobre várias técnicas empregadas na literatura pode ser encontrada em (UMBARKAR; SHETH, 2015).

A mutação amplia a variabilidade genética na população. Na *representação binária*, um *bit* é escolhido aleatoriamente e tem seu valor modificado com base em uma determinada probabilidade, geralmente baixa. Neste trabalho, a mutação será inserida em forma de ruído gaussiano nos indivíduos e detalhes serão abordados posteriormente. Os novos indivíduos serão avaliados, um *fitness* lhes será atribuído e serão inseridos na população, substituindo indivíduos menos aptos.

A *finalização*, como já exposto, verifica se as condições de evolução foram atingidas, encerrando sua execução em caso de resposta positiva e retornando à etapa de *seleção* em

caso de resposta negativa. Geralmente, a *finalização* está atrelada a um número máximo de iterações/gerações no algoritmo, ou até que alguma resposta satisfatória (de acordo com algum critério) seja encontrada.

3.2 Algoritmo Genético para o problema proposto

O AG utilizado neste trabalho é um AG considerado tradicional e segue os conceitos básicos sobre AGs, apresentadas na Seção 3.1. Contudo, é claro, tais conceitos foram adaptados para o problema proposto. O primeiro passo é selecionarmos uma representação adequada para codificar as possíveis soluções para o problema proposto, e, nesse caso, a representação via *ponto flutuante* onde cada indivíduo que compõe a população é uma vetorização criada a partir da *combinação convexa* das matrizes $A_c \in \mathbb{R}^{nc \times nc}$, $B_c \in \mathbb{R}^{nc \times ny}$, $C_c \in \mathbb{R}^{m \times nc}$ e $D_c \in \mathbb{R}^{n \times ny}$ do controlador, ou seja, **cada indivíduo que irá compor a população é um possível controlador para o problema apresentado na Seção 2.2.**

Definição 1. Considere a matriz $A^{m \times n}$. Define-se por $vec(A)$ o vetor coluna $mn \times 1$ de todos os elementos da matriz A tomados coluna por coluna, ou seja,

$$vec(A) = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \\ \vdots \\ a_{1n} \\ a_{mn} \end{bmatrix}. \quad (3.1)$$

A vetorização das matrizes não só torna os cálculos mais fáceis de serem realizados como também torna a explicação mais próxima das encontradas na literatura se montarmos a variável de decisão na forma vetorial. Portanto, como cada indivíduo na população será composto pela vetorização das matrizes A_c , B_c , C_c e D_c , considere o operador φ

$$\varphi(\mathbf{R}_o) = \left(\begin{bmatrix} vec(A_c(1)) \\ vec(B_c(1)) \\ vec(C_c(1)) \\ vec(D_c(1)) \end{bmatrix}, \dots, \begin{bmatrix} vec(A_c(N)) \\ vec(B_c(N)) \\ vec(C_c(N)) \\ vec(D_c(N)) \end{bmatrix} \right), \quad (3.2)$$

ao qual cada controlador $\mathbf{R}_o$ em (2.4) será submetido, onde N é o número total de estados θ . O operador φ é responsável por vetorizar e agrupar cada um dos conjuntos de matrizes A_c , B_c , C_c e D_c com θ observável, ou seja, matrizes que atuam no controlador apresentado em (2.4); esse operador auxiliará na criação de indivíduos para a população inicial.

De maneira análoga, considere o operador Ψ , que, ao invés de vetorizar as matrizes de controlador com θ observável, apresentado em (2.4), tem como objetivo vetorizar as matrizes do

controlador (2.17), com apenas um conjunto de matrizes A_c , B_c , C_c e D_c , ou seja um controlador independente do modo θ de operação; portanto:

$$\Psi(\mathbf{R}_n) = \begin{bmatrix} \text{vec}(A_c) \\ \text{vec}(B_c) \\ \text{vec}(C_c) \\ \text{vec}(D_c) \end{bmatrix}. \quad (3.3)$$

Estimar a população inicial requer atenção. Não podemos simplesmente “chutar” valores para as matrizes; as possibilidades são infinitas e seria raro encontrarmos resultados. Entretanto, nosso problema possui a “vantagem” de ser um problema “vizinho” de outro em que os estados de Markov são conhecidos. Portanto, resolvemos o problema com observação completa de θ usando as fórmulas de (DRAGAN; COSTA, 2016), que leva a N conjunto de matrizes $A_c(1), \dots, D_c(1), \dots, A_c(N), \dots, D_c(N)$, cada um dos quais é vetorizado de acordo com (3.2), levando a N vetores ou “indivíduos” da população inicial, e finalmente geramos outros indivíduos através de *combinações convexas* desses N vetores.

Definição 2. Dado $x_i \in \mathbb{R}^n$, $\lambda_i \in [0, 1] \mid i = \{1, 2, \dots, N\}$, tais que $\sum_{i=1}^N \lambda_i = 1$, o ponto $\sum_{i=1}^N \lambda_i x_i$ denomina-se *combinação convexa* dos pontos $x_i \in \mathbb{R}^n \mid i = \{1, 2, \dots, N\}$.

Tal definição pode ser estendida para um conjunto de pontos. O ponto

$$\begin{cases} \mathbf{b} = \lambda_1 \mathbf{x}^1 + \lambda_2 \mathbf{x}^2 + \dots + \lambda_N \mathbf{x}^N \\ \lambda_j \geq 0 (j = 1, 2, \dots, N) \end{cases} \quad (3.4)$$

é uma *combinação convexa* dos pontos $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^N$, em que

$$\lambda_1 + \lambda_2 + \dots + \lambda_N = \sum_{i=1}^N \lambda_i = 1. \quad (3.5)$$

Contudo, caso $\sum_{i=1}^N \lambda_i > 1$, o novo ponto encontrado pode estar fora da *combinação convexa*, extrapolando-a.

A partir das Definições 1 e 2, criamos cada indivíduo que irá compor a população inicial da seguinte maneira:

$$g_h' = \sum_{i=1}^N \varphi_i(\mathbf{R}_o) \lambda_i, \quad j = 1, \dots, m, \quad (3.6)$$

com h o número total de indivíduos na população e N o número total dos estados de Markov.

Os valores de λ serão gerados aleatoriamente através de uma distribuição uniforme.

Definição 3. A distribuição uniforme se caracteriza por ter uma Função Densidade Probabilidade (FDP) contínua em um intervalo fechado $[a, b]$, em que X é dado por:

$$\begin{cases} \frac{1}{b-a}, & a < X < b; \\ 0, & \text{caso contrário.} \end{cases} \quad (3.7)$$

Para facilitar, utilizamos a notação $X \sim U[a, b]$.

Analisando a Equação (3.6), notamos que ela depende da quantidade de estados de Markov do controlador original e, conseqüentemente, um valor de λ associado a cada estado; como utilizamos a função “rand” do MATLAB[®] para geração de tais valores, e tal função recebe como parâmetros a quantidade de estados de Markov, teremos $\sum_{i=1}^N \lambda_i > 1$, e o indivíduo criado extrapola o politopo gerado pelos pontos caso $N > 2$ ou a reta, caso $N = 2$. Neste trabalho, isso poderia ser interpretado como ruído e contribuiria para ampliar a variabilidade genética da população; entretanto, para a geração dos indivíduos que irão compor a população inicial, implementamos o método de modo que a garantir que $\sum_{i=1}^N \lambda_i = 1$.

Como os valores de λ na Equação (3.6) são gerados aleatoriamente com base na distribuição uniforme para cada indivíduo que irá compor a população, os indivíduos serão distintos.

Com o intuito de aumentar a variabilidade genética entre esses indivíduos, também optamos por utilizar como elemento mutagênico a distribuição normal (também conhecida como Distribuição Gaussiana), e portanto, cada indivíduo é descrito por

$$g_h = g'_h + N(\mu, \sigma^2). \quad (3.8)$$

Definição 4. A Distribuição Gaussiana é uma distribuição cuja FDP equivale a densidade normal, dado por

$$\begin{aligned} f(X) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]} \\ &= \frac{1}{\sigma\sqrt{2\pi}} e^{\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]}, -\infty < x < \infty. \end{aligned} \quad (3.9)$$

com média μ e desvio padrão σ . Para facilitar, usamos a notação $X \sim N(\mu, \sigma^2)$.

Note que a geração de um indivíduo consiste em “criar” somente uma matriz A_c , uma matriz B_c , uma matriz C_c e uma matriz D_c que são possíveis soluções para o controlador (2.17), ou seja, um controlador independente do modo, ao invés de conjuntos de matrizes A_c , B_c , C_c e D_c que atuam no controlador apresentado em (2.4), o controlador com θ observável.

A seguir, devemos obter uma população composta de possíveis soluções, ou seja, $g_h \in \mathbb{R}^{nc+ny+nc+ny}$ e submetê-las ao processo relatado na Figura 2. Entretanto, embora seja mais fácil manipularmos cada possível solução na forma vetorial, a Equação (2.11) faz uso das matrizes A_c , B_c , D_c e D_c com suas dimensões “originais”, ou seja, $A_c \in \mathbb{R}^{(nc \times nc)}$, $B_c \in \mathbb{R}^{(nc \times ny)}$, $C_c \in \mathbb{R}^{(m \times nc)}$ e $D_c \in \mathbb{R}^{(n \times ny)}$. Portanto, definimos $\Psi^{-1}(g_m)$ como

$$\Psi^{-1}(g_h) = (A_c, B_c, C_c, D_c), \quad (3.10)$$

que opera de modo inverso à Equação (3.3), ou seja, φ transforma um vetor em um controlador com as matrizes em dimensões apropriadas. A avaliação consiste em calcular o custo J a partir

da Equação (2.11) para cada controlador, ou seja, $J(\Psi^{-1}(g_1)), J(\Psi^{-1}(g_2)), \dots, J(\Psi^{-1}(g_h))$, e a solução privilegiará aqueles com o menor valor de J .

Esse tipo de método já foi estudado com alguns resultados obtidos pelo nosso grupo de pesquisa para controladores estáticos, ou seja, as matrizes A_c , B_c e C_c são nulas (SILVA, 2012).

Os resultados finais de um AG serão influenciados pelos indivíduos que irão compor a população inicial. Se a população inicial for inicializada com possíveis soluções consideradas “ruins”, a heurística pode não encontrar soluções satisfatórias, mesmo com a inserção de elementos mutagênicos nos indivíduos entre gerações, principalmente se o critério de parada do AG impedir que o algoritmo avance gerações/iterações suficientes, caso tal critério for baseado em um número máximo de gerações/iterações e não até que uma solução satisfatória seja encontrada, afinal, não espera-se que ao final da execução, uma população inicializada com “peixes” termine com “elefantes” em sua composição (neste trabalho, o critério de parada está atrelado a um tempo máximo de execução e maiores detalhes serão abordados no Capítulo 5). Do ponto científico, a vida e todos os organismos como conhecemos atualmente surgiu após bilhões de anos de evolução, contudo, como o problema neste trabalho tem a “vantagem” de inicializar a população inicial seguindo preceitos de um problema com solução conhecida, espera-se que uma solução viável (quando possível e detalhes serão abordados posteriormente) seja encontrada em muito menos tempo (no Capítulo 6 descrevemos um método que explora melhor a questão de “adaptabilidade” dos indivíduos).

Agora que já sabemos como constituir indivíduos para nossa população, como avaliá-los? A *avaliação* permite ao AG determinar a proximidade do indivíduo à solução ótima do problema. Ela é feita através da função que caracteriza o problema e tem por objetivo oferecer uma medida de aptidão que irá guiar o processo de busca, e no nosso caso, a função de custo apresentada na Seção 2.1.1, Equação (2.11). Submeteremos cada indivíduo que compõe nossa população a esta função e o inverso do resultado será o chamado *fitness* desse indivíduo, ou seja:

$$f(g_h) = 1/J(\Psi^{-1}(g_h)), \quad (3.11)$$

em que h é o número total de indivíduos na população. A opção pelo inverso dá-se pelo fato de que, em se tratando de SLSMs, quanto menor o custo, melhor o desempenho do sistema, portanto, quanto maior o resultado proveniente de cada indivíduo, menor seu *fitness* e portanto, menos apto e menor será a sua possibilidade de escolha para *crossover*.

A próxima etapa de um AG é a *seleção de indivíduos para crossover*. Existem diversas técnicas de seleção e informações detalhadas podem ser encontrados em (ALABSI; NAOUM, 2012) e (YADAV; SOHAL, 2017). Neste trabalho, foi utilizado o *método da roleta* (*Roulette Wheel Selection*), uma técnica tradicional de seleção em AGs em que todos os indivíduos que compõem a população são dispostos em uma roleta de acordo com seu *fitness*. Quanto maior o *fitness*, maior o espaço ocupado na roleta por esse indivíduo, ou seja, a probabilidade de determinado indivíduo ser selecionado é proporcional ao seu *fitness*. Giramos esta roleta virtual e

o indivíduo correspondente ao segmento selecionado é escolhido para o *crossover*. Tal processo é repetido até o número máximo de indivíduos desejado seja selecionado. Matematicamente, a probabilidade de um indivíduo i ser escolhido dá-se por

$$p_i = \frac{f(g_i)}{\sum_{j=1}^h f(g_j)}, \quad (3.12)$$

em que h representa o número total de indivíduos na população.

Contudo, vale um adendo: nada garante que os melhores indivíduos da população serão selecionados. Como estamos lidando com probabilidades, eventualmente indivíduos menos aptos serão selecionados; isto não é algo necessariamente ruim, já que contribui para a variabilidade genética da população. O método de seleção suscita uma pergunta: Então, por que não selecionar os melhores indivíduos em cada geração? A resposta é que, caso sempre os melhores indivíduos sejam selecionados, os “filhos” resultantes via *crossover* serão sempre os mesmos ou com muita pouca diferença, resultando em uma população homogênea o que, por sua vez, dificultará o processo de encontrar melhores soluções; quanto mais homogênea a população, maior a possibilidade de encontrar melhores soluções.

Figura 3 – Método da roleta para seleção de indivíduos para *crossover* de um Algoritmo Genético.

Também vale mencionar que se traçarmos um paralelo com a reprodução de espécies na natureza, um casal pode ter inúmeros filhos e/ou um mesmo indivíduo por ter filhos com parceiros(as) diferentes. Esta ideia foi levada em consideração para selecionarmos os indivíduos para *crossover*, sendo que duas diretrizes foram estabelecidas: a primeira é que determinado indivíduo deve ter um parceiro(a) de reprodução que difere de si mesmo (a não ser, claro, em reprodução assexuada onde não há troca de gametas, o que impede variabilidade genética; mas não vamos considerar esse tipo aqui); a segunda é que se determinado indivíduo for selecionado para o *crossover*, diminuiremos sua aptidão para dificultar que esse mesmo indivíduo seja selecionado novamente na mesma geração. Isso não impede a sua escolha, mas tal característica foi implementada com o intuito de ampliar a possibilidade de outros indivíduos serem escolhidos,

visando variabilidade da população. Obviamente, essa última diretriz impede a predominância de um determinado indivíduo na população, e, em se tratando de AGs, isso é algo a ser considerado. Se em uma população, determinado indivíduo possuir um *fitness* muito superior em relação aos demais indivíduos, isso fará com que seja selecionado diversas vezes para o *crossover* (uma vez que a probabilidade de seleção de indivíduos é proporcional ao *fitness*, e no caso deste trabalho, quanto maior o *fitness*, mais apto o indivíduo e maior sua probabilidade de escolha), o que, ao longo das gerações, fará com que a população atinja a homogeneidade, uma vez que todos os indivíduos se assemelharão a este indivíduo com o melhor *fitness*; isso minimizará a chance do algoritmo de encontrar diferentes soluções, uma vez que novas soluções são encontradas via *crossover* e, com uma população homogênea, os resultados serão muito parecidos, ou até mesmo iguais. Por isso optamos por diminuir em 20% a aptidão de todo indivíduo selecionado para o *crossover*; o intuito é aumentar a chance que outros indivíduos sejam escolhidos, mesmo que, no geral, o *fitness* tais indivíduos não sejam os melhores; isso nos dará “casais” heterogêneos que gerarão filhos heterogêneos contribuindo para mantermos uma população distinta, o que maximizará a possibilidade de que o algoritmo encontre um resultado satisfatório (de acordo com algum critério que dependerá do problema).

Neste trabalho, a quantidade de indivíduos selecionados para o *crossover* em cada geração é proporcional à quantidade de indivíduos na população inicial, que por sua vez, é proporcional à quantidade de indivíduos que serão substituídos por indivíduos mais aptos entre gerações; se x indivíduos serão substituídos, devemos selecionar $2x$ indivíduos para o *crossover*, já que um “casal” gerará um “filho”. Quanto maior a população inicial, maior será o valor de x , e neste trabalho, x será de 20% da população inicial. A quantidade de indivíduos que será substituída entre gerações também é importante em se tratando de AGs, novamente, visando a heterogeneidade da população. Se a quantidade de indivíduos que será substituída entre gerações for baixa, perde-se em recursos computacionais pois o algoritmo pode demorar para encontrar uma solução ótima (de acordo com algum critério); caso a quantidade de indivíduos substituída entre gerações seja muito alta, corre-se o risco de perder variabilidade genética a população tornar-se homogênea logo nas primeiras gerações/iterações do algoritmo, uma vez que indivíduos distintos e/ou com *fitness* ótimos, podem ser substituídos entre gerações devido ao alto número de “filhos” gerados via *crossover* em uma mesma geração. No contexto deste trabalho, um estudo mais detalhado poderia ser feito substituindo parcelas cada vez maiores da população e comparar os resultados finais obtidos pelo algoritmo (começar em 5% e ir aumentando gradativamente de 5% em 5, por exemplo). Entretanto, esse tipo de estudo não é o foco deste trabalho, e, portanto, optamos por substituir 20% da população entre as gerações, um valor que consideramos não muito baixo de modo que impeça que algoritmo demore para encontrar soluções, e nem muito alto para que a população torne-se homogênea logo nas primeiras iterações do algoritmo minimizando as chances que soluções ótimas (novamente, de acordo com algum critério) seja encontrada.

Observação 2. A conclusão óbvia a partir do parágrafo anterior é que quantidade de indivíduos

selecionados para o *crossover* deve ser par, e ela está correta. Contudo, isso dependerá da quantidade de indivíduos na população inicial, e se a quantidade de indivíduos selecionada para o *crossover* for ímpar, um indivíduo a mais será selecionado. Exemplificando: se a quantidade de indivíduos na população inicial for de 55 indivíduos, tomando 20%, teríamos 11 indivíduos selecionados para o *crossover*, o que impossibilitaria “casais”; nesse adicionaríamos mais um indivíduo que também seria escolhido utilizando o *método da roleta*.

Com os indivíduos selecionados, efetuaremos o *crossover* combinando os indivíduos em pares seguindo as diretrizes mencionadas no parágrafo anterior. O *crossover* também será efetuado via *combinação convexa* (UMBARKAR; SHETH, 2015) e cada “casal” gerará um “filho”. Chamaremos $\mathcal{S}$ os indivíduos selecionados para o *crossover*, que forçamos ter cardinalidade par, tendo em vista a Observação 2; chamaremos de $\mathcal{M}$ e $\mathcal{P}$ uma partição de $\mathcal{S}$, ambos com a mesma cardinalidade, que servem para designar os conjuntos de “pais” e “mães”. Além disso, vamos incluir um índice “*ell*” para marcar a geração, de forma que g_f^{ell} é o i -ésimo indivíduo da ell -ésima geração, e portanto:

$$g_f^{ell} = \lambda_f g_m^{ell} + (1 - \lambda_f) g_p^{ell}, \quad (3.13)$$

com $m \in \mathcal{M}$ e $p \in \mathcal{P}$. Varremos os conjuntos “pais” e “mães” gerando um “filho” para cada casal. A cada “filho” será atribuído ruído gaussiano (*mutação*), ou seja,

$$g_f^{ell} = g_f^{ell} + N(\mu, \sigma^2), \quad (3.14)$$

e submetido à Equação (3.10), o que, por sua vez, a Equação (2.11) e a Equação (3.11), em que um *fitness* lhe será atribuído. Entretanto, para o *crossover*, optamos por valores de λ em que $\sum_{i=1}^N \lambda > 1$, especificamente $\sum_{i=1}^N \lambda = 1,2$ (ou seja, λ_i é uma v.a $U(0, 1.2)$), e isso faz com os “filhos” gerados extrapolem o segmento de reta, visando, novamente, adicionar variabilidade genética à população.

Observação 3. Caso estivéssemos lidando com *representação binária*, existiriam diversos modos para realizar o *crossover*, como pode ser visto em (UMBARKAR; SHETH, 2015).

Observação 4. Na literatura, a *mutação* é efetuada de acordo com determinada probabilidade, geralmente, baixa (MITCHELL, 1996). Uma baixa probabilidade de mutação impede que a população fique estagnada em determinado local dentro do espaço de busca; com uma taxa muito alta, a busca pode se tornar aleatória (CARVALHO, 2009).

Neste trabalho utilizamos ruído gaussiano como elemento mutagênico em cada elemento da prole contrariando a literatura clássica sobre AGs. Entretanto, testes efetuados durante a implementação do algoritmo revelaram que essa estratégia propiciou melhores resultados por aumentar a heterogeneidade da população entre gerações, o que, por sua vez, contribuiu para que mais instâncias submetidas aos métodos (detalhes serão abordados no Capítulo 5) pudessem ser

resolvidas, e portanto optamos por inserir ruído gaussiano como elemento mutagênico em cada indivíduo “filho” resultante via *crossover*.

Claro que esse comportamento (resolução de um maior número de instâncias) aplica-se somente no SLSM estudado neste trabalho, apresentado em (2.1), e não podemos afirmar o mesmo para outra classe de sistemas. Além disso, notamos uma saturação na porcentagem de instâncias cujo método encontra solução e mais detalhes serão abordados nos Capítulos 5 e 7.

Finalmente, os novos indivíduos serão inseridos na população. Entre uma geração e outra, indivíduos deixarão de existir (menos aptos) para darem lugar à nova geração em tese, mais aptos; em tese pois nem sempre isso irá acontecer. Como estamos utilizando um método de seleção probabilístico com base no *fitness* do indivíduo, as vezes, indivíduos com um *fitness* ruim podem ser escolhidos, gerando, conseqüentemente, um “filho” ruim, ou seja, um filho menos apto com relação àqueles que serão substituídos ao final de uma geração (iteração no algoritmo). Com relação ao tamanho da população, vamos considerá-la fixa, ou seja, a quantidade de indivíduos que será inserida da população é proporcional à quantidade de indivíduos que deixarão de existir. Neste trabalho, entre uma geração e outra, 20% dos indivíduos deixarão de existir para darem lugar aos novos indivíduos, com justificativas já abordadas em parágrafos anteriores, e os indivíduos mais aptos estão sendo mantidos entre uma geração e outra, estratégia conhecida como *seleção elitista* (MICHALEWICS, 1996), (FOGEL, 1994); com isso, mesmo se o um indivíduo gerado via *crossover* for um indivíduo considerado ruim em comparação com o restante da população, não existe a chance de ele ocupar o lugar, por exemplo, do melhor indivíduo na população em determinada geração, uma vez que esse estará garantido na geração seguinte.

Usualmente o término de execução de um AG está atrelado a um número máximo de *gerações* ou ainda de uma solução considerada aceitável para o problema considerado, ou seja, todas as etapas descritas serão executadas enquanto um determinado critério não for satisfeito. Neste trabalho, executamos o AG para diferentes períodos de tempo, e maiores detalhes, assim como resultados, serão apresentados no Capítulo 5.

Também vale dizer que AGs são muito sensíveis aos parâmetros; uma pequena variação em uns desses parâmetros pode fazer com que os resultados obtidos sejam melhores/piores. Neste trabalho, buscamos atribuir valores para esses parâmetros (quantidade de indivíduos selecionada para o *crossover*, por exemplo) da forma mais “imparcial” possível, ou seja, não efetuamos diversos testes com diferentes valores buscando os melhores resultados possíveis por pensar que isso poderia levar um leitor que queira replicar o método descrito a cogitar que os resultados estariam anviados e ajustados para as instâncias submetidas ao algoritmo. Claro que um estudo mais detalhado sobre os valores que esses parâmetros podem assumir poderia ser efetuado, contudo, também não é o foco deste trabalho.

Algoritmo 1 – Pseudocódigo demonstrando o funcionamento do AG descrito nesse capítulo

- 1: Submeter o controlador em (2.4), o controlador dependente do modo θ de operação, à Equação (3.2).
 - 2: Gerar indivíduos para a população inicial de acordo com a Equação (3.6).
 - 3: Adicionar ruído (Definição 4) à cada um dos 50 elementos da população de acordo com a Equação (3.8).
 - 4: Atribuir o *fitness* a cada um dos indivíduos da população inicial de acordo com a Equação (3.11), submetendo, antes, cada indivíduo à Equação (3.10) para “desvetorizar” cada um dos indivíduos na população.
 - 5: **enquanto** condição de parada não for satisfeita - neste trabalho, tempo máximo de execução **faça**
 - 6: Selecionar 20% dos indivíduos da população, divididos em indivíduos “pais” e indivíduos “mães”, para reprodução de acordo com a Equação (3.12).
 - 7: Cruzar os indivíduos “pais” e os indivíduos “mães” (*crossover*), gerando indivíduos “filhos”, de acordo com a Equação (3.13).
 - 8: Adicionar ruído (Definição 4) à cada um dos indivíduos “filhos” gerados via *crossover* de acordo com a Equação (3.14).
 - 9: Atribuir o *fitness* à cada um dos indivíduos “filhos” gerados via *crossover* de acordo com a Equação (3.11), submetendo, antes, cada um dos indivíduos “filhos” à Equação (3.10) para “desvetorizá-los”.
 - 10: Inserir os indivíduos “filhos” na população no lugar de indivíduos, teoricamente, mais aptos. Entre cada geração 20% da população será substituída.
 - 11: **fim enquanto**
-

O MÉTODO QUASI-GRADIENTE

Outro método estudado para o problema apresentado na Seção 2.2 é baseado em gradiente. Métodos baseados em gradiente funcionam atribuindo um ponto inicial no espaço de soluções e a partir daí geram recursivamente novos pontos, até que uma solução satisfatória (de acordo com algum critério) seja obtida ou um número máximo de iterações seja atingido. A ideia básica é que dado um ponto inicial $g_0 \in \mathbb{R}^v$, uma sequência g_t é obtida iterativamente em que g_t é uma sequência finita e o último ponto obtido é tido como a solução ideal para o problema (GOUVÊA, 2016).

O procedimento utilizado por esse tipo de método consiste em iniciar a busca a partir de um ponto inicial g_0 e gerar uma sequência de possíveis soluções por meio da expressão (NOCEDAL; WRIGHT, 2006)

$$g_{t+1} = g_t + \alpha(t)\mathbf{d}(t), \quad (4.1)$$

em que $\mathbf{d}(t) \in \mathbb{R}^v$ é a direção de busca, $\alpha(t) > 0 \in \mathbb{R}$ é o tamanho do passo dado ao longo da direção $\mathbf{d}(t)$ e $g_t \in \mathbb{R}^v$ é o vetor de variáveis de decisão do problema na iteração t .

A maioria dos métodos de otimização que utilizam a Equação (4.1) determinam o passo $\alpha(t)$ e $\mathbf{d}(t)$ por meio de uma técnica de busca linear (PILLO; PALAGI, 2002). Essa técnica exige que $\mathbf{d}(t)$ seja uma direção de descida, e, considerando a Equação (2.11), temos

$$J(g_t)\mathbf{d}(t) \leq 0, \quad (4.2)$$

em que J é a função a ser minimizada. Dessa forma, se $\mathbf{d}(t)$ é uma direção de descida, então

$$J(g_{t+1}) \leq J(g_t). \quad (4.3)$$

Métodos de otimização baseados em gradientes diferenciam-se na maneira em que a direção e o tamanho dos passos são calculados na equação apresentada em (4.1), e geram uma sequência de pontos que convergem para a solução ótima (de acordo com algum critério). Porém,

na maioria dos casos isto não é possível devido a vários fatores, como: não convexidade de função objetivo, quantidade de variáveis, etc. (BORTOLIN, 2012), (OLIVEIRA; COSTA, 2015). Assim, muitos dos algoritmos são capazes apenas de encontrar uma solução ótima local, ou seja, métodos baseados em gradiente não garantem a convergência para o mínimo global da função encontrando mínimos, em muitos casos, locais.

Abordagens para problemas de controle sem observação da variável de salto utilizando métodos baseados em gradiente também foram obtidas, por exemplo, em (BORTOLIN, 2012), entretanto, para controladores estáticos.

Neste trabalho, devido as alterações no “método gradiente clássico” encontrado na literatura que serão descritas na seção a seguir, como a estimar a derivada numericamente, podemos classificar o método descrito neste capítulo como um método baseado em gradiente, e não um método gradiente “puro”, e por isso o nome Método Quasi-Gradiente (MG).

4.1 Método Quasi-Gradiente para o problema proposto

O problema a ser solucionado pelo MG é o de encontrar as matrizes A_c , B_c , C_c e D_c que atuam no controlador sem a observação da variável θ apresentado em (2.17), e o custo J desse controlador deve ser o menor possível. Portanto, como trata-se de um problema de otimização, optamos por um método baseado em gradiente.

Para este método, o modo como o ponto inicial é gerado segue a descrição apresentada na Seção 3.2 para a geração de indivíduos que compõem a população inicial no AG, ou seja, o espaço de busca é o das matrizes $A_c \in \mathbb{R}^{nc \times nc}$, $B_c \in \mathbb{R}^{nc \times ny}$, $C_c \in \mathbb{R}^{m \times nc}$ e $D_c \in \mathbb{R}^{n \times ny}$ e a variável de decisão também será montada na forma vetorial (Seção 3.2, Definição 1); assim sendo, considere o operador φ

$$\varphi(\mathbf{R}_o) = \left(\begin{bmatrix} \text{vec}(A_c(1)) \\ \text{vec}(B_c(1)) \\ \text{vec}(C_c(1)) \\ \text{vec}(D_c(1)) \end{bmatrix}, \dots, \begin{bmatrix} \text{vec}(A_c(N)) \\ \text{vec}(B_c(N)) \\ \text{vec}(C_c(N)) \\ \text{vec}(D_c(N)) \end{bmatrix} \right), \quad (4.4)$$

ao qual cada controlador $\mathbf{R}_o$ em (2.4) será submetido, onde N é o número total de estados θ . Assim como para o AG, o operador φ é responsável por vetorizar e agrupar cada um dos conjuntos de matrizes A_c , B_c , C_c e D_c com θ observável, ou seja, matrizes que atuam no controlador apresentado em (2.4), e auxiliará na criação de um ponto inicial para execução do MG.

Pelo mesmo motivo descrito na Seção 3.2, Equação (3.3) Ψ é dado por

$$\Psi(\mathbf{R}_n) = \begin{bmatrix} \text{vec}(A_c) \\ \text{vec}(B_c) \\ \text{vec}(C_c) \\ \text{vec}(D_c) \end{bmatrix}, \quad (4.5)$$

ou seja, o operador Ψ , ao invés de vetorizar as matrizes de controlador com θ observável (2.4), tem como objetivo vetorizar as matrizes do controlador (2.17), com apenas um conjunto de matrizes A_c , B_c , C_c e D_c , ou seja um controlador independente do modo θ de operação.

Alguns dos desafios mencionados na Seção 3.2 também se fazem presentes aqui, no caso, como definirmos o ponto inicial. Assim como no AG, não podemos “chutar” valores para o ponto inicial dado que as possibilidades são infinitas e não encontraríamos resultados satisfatórios. Portanto, assim como no AG, resolvemos o problema com observação completa de θ usando as fórmulas de (DRAGAN; COSTA, 2016), que leva a N conjunto de matrizes $A_c(1), \dots, D_c(1), \dots, A_c(N), \dots, D_c(N)$, cada um dos quais é vetorizado de acordo com (4.4), levando a N vetores, e finalmente geramos o ponto inicial dentro do espaço de busca através da *combinação convexa* desses N vetores (Seção 3.2, Definição 2); em consequência, a aptidão do ponto inicial também será dada de acordo com a Equação (3.6). Assim como no AG, soma-se ruído gaussiano (Definição 4) ao ponto encontrado, visando inicializar cada possível solução em locais diferentes dentro do espaço de busca (embora no MG a *combinação convexa* seja utilizada apenas na geração do ponto inicial, e, portanto, poderíamos desconsiderar a adição de ruído gaussiano, ele é necessário por motivos que serão abordados no Capítulo 5). Assim sendo, o ponto inicial para o MG é dado de maneira semelhante a (3.8):

$$g_t = g'_t + N(\mu, \sigma^2), \quad (4.6)$$

em que t representa o ponto inicial dentro do espaço de busca.

Assim como no AG, devemos avaliar o quão próximo cada um dos pontos na sequência estão para a solução ótima do problema e para o MG, também usaremos a função de custo apresentada na Seção 2.1.1, Equação (2.11), submetendo, antes, cada ponto à Equação (3.10), uma vez que a Equação (2.11) necessita das matrizes A_c , B_c , C_c e D_c em suas dimensões “originais”, e assim sendo, o custo consiste em avaliarmos $J(\Psi^{-1}(g_1)), J(\Psi^{-1}(g_2)), \dots, J(\Psi^{-1}(g_t))$. Entretanto, diferentemente do apresentado no AG, não utilizaremos o inverso do custo proveniente da Equação (2.11), uma vez que não submeteremos o custo obtido a nenhum método de escolha de possíveis soluções, tornando o inverso desnecessário.

Neste trabalho, utilizaremos o método da máxima descida (*steepest descent method*), que consiste em uma estratégia de busca que se move na direção contrária do gradiente da função objetivo no ponto atual da busca e com tamanho de passo apropriado em cada iteração (NOCEDAL; WRIGHT, 2006), ou seja, a direção da busca é dada por:

$$\mathbf{d}(t) = -\nabla(\Psi(g_t)). \quad (4.7)$$

Portanto, considerando a Equação (4.7), modificamos a Equação (4.1) e obtemos

$$\begin{aligned} g_{t+1} &= g_t + \alpha(t) - \nabla(\Psi(g_t)) \\ &= g_t - \alpha(t) \nabla(\Psi(g_t)), \end{aligned} \quad (4.8)$$

em que $\mathbf{d}(t) \in \mathbb{R}^v$ é a direção de busca, $\alpha(t) \in \mathbb{R}$ é o tamanho do passo dado ao longo da direção $\mathbf{d}(t)$, $g_t \in \mathbb{R}^v$ é o vetor de variáveis de decisão do problema na iteração t , e a cada iteração, o método faz uma busca para encontrar um passo $\alpha(t)$ que satisfaça

$$J(\Psi^{-1}(g_t - \alpha(t) \nabla(\Psi(g_t)))) \leq J(\Psi^{-1}(g_t)), \quad (4.9)$$

em que J provem da Equação (2.11), ou seja, a equação de custo que avalia quão bem determinada solução satisfaz o problema proposto. A Equação (4.9) enuncia que o custo do ponto g_{t+1} deve ser menor ou igual do que o custo do ponto g_t .

Com essa notação, inicia-se o método com um “palpite inicial” para o controlador $A_c^{(0)}$, $B_c^{(0)}$, $C_c^{(0)}$ e $D_c^{(0)}$, com que se calcula o ponto inicial g_0 e em seguida o método gera uma sequência seguindo a expressão apresentada em (4.8), **onde cada elemento da sequência é um possível controlador para o problema apresentado na Seção 2.2.**

Contudo, o principal desafio está nas derivadas. Não é possível calcularmos a derivada desta função (ou as equações se tornariam tão complexas que seria tema para outro mestrado). A solução encontrada foi estimá-las numericamente, através da *fórmula de três pontos*

$$\nabla(\Psi(g_t)) \approx \frac{1}{2h} \left[J(\Psi^{-1}(g_t(x_k + h))) - J(\Psi^{-1}(g_t(x_k - h))) \right], \quad (4.10)$$

onde x_k corresponde a cada elemento em g_t na iteração t e h é a variação. Como g_t é uma matriz, podemos acessar cada um de seus elementos utilizando a notação $g_t(x_k)$, em que g_t é a solução no instante t , e x_k um elemento em g_t .

Esta derivada é utilizada em situações em que não temos a função disponível, mas somente determinados valores referentes a esta função, para funções que não são deriváveis em todo seu domínio ou de derivação não-trivial, essa última, o caso deste trabalho. Tanto valores muito altos de h quanto valores muito baixos de h levam a erros (h alto acarreta erros denominados de truncamento, h baixo gera erros chamados de arredondamento, vide, por exemplo (BURDEN; FAIRES, 2008)), e um sério problema é determinar o que seriam esses valores altos e baixos; uma maneira de encontrar valores razoáveis de h é utilizar vários pontos e verificar os valores mais consistentes de derivada.

Exemplo 1. Vamos considerar g_0 como as matrizes $A_c^{(0)}$, $B_c^{(0)}$, $C_c^{(0)}$ e $D_c^{(0)}$ que atuam no controlador, para $g_0 \in \mathbb{R}^{nc+ny+nc+ny}$, como já descrito. Para fins didáticos, assumiremos que as matrizes $A_c^{(0)}$, $B_c^{(0)}$, $C_c^{(0)}$ e $D_c^{(0)}$ são:

$$A_c = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}; B_c = \begin{bmatrix} a_5 \\ a_6 \end{bmatrix}; C_c = \begin{bmatrix} a_7 & a_8 \end{bmatrix}; D_c = \begin{bmatrix} a_9 \end{bmatrix}. \quad (4.11)$$

Finalmente, aplicando a fórmula de três pontos:

$$\nabla(\Psi(g_0))_1 \approx \frac{1}{2 \times h} \times \underbrace{\underbrace{J}_{\text{Equação (2.11)}}}_{g_0} \begin{bmatrix} a_1 + h \\ a_3 \\ a_2 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} - \frac{1}{2 \times h} \times \underbrace{\underbrace{J}_{\text{Equação (2.11)}}}_{g_0} \begin{bmatrix} a_1 - h \\ a_3 \\ a_2 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} = \text{escalar.}$$

$$\nabla(\Psi(g_0))_2 \approx \frac{1}{2 \times h} \times \underbrace{\underbrace{J}_{\text{Equação (2.11)}}}_{g_0} \begin{bmatrix} a_1 \\ a_3 + h \\ a_2 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} - \frac{1}{2 \times h} \times \underbrace{\underbrace{J}_{\text{Equação (2.11)}}}_{g_0} \begin{bmatrix} a_1 \\ a_3 - h \\ a_2 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} = \text{escalar.}$$

$\vdots$

$$\nabla(\Psi(g_0))_9 \approx \frac{1}{2 \times h} \times \underbrace{\underbrace{J}_{\text{Equação (2.11)}}}_{g_0} \begin{bmatrix} a_1 \\ a_3 \\ a_2 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 + h \end{bmatrix} - \frac{1}{2 \times h} \times \underbrace{\underbrace{J}_{\text{Equação (2.11)}}}_{g_0} \begin{bmatrix} a_1 \\ a_3 \\ a_2 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 - h \end{bmatrix} = \text{escalar.}$$

Com o ponto original g_0 e $\nabla(\Psi(g_0))$, substituímos na expressão (4.8). O novo ponto será dado por

$$g_1 = [g_0 - \alpha \times \nabla(\Psi(g_0))] \quad (4.12)$$

e generalizando

$$g_{t+1} = [g_t - \alpha(t) \times \nabla(\Psi(g_t))]. \quad (4.13)$$

Entretanto, antes de calcularmos a derivada numericamente, uma nova questão surge: qual o valor a variável h deve assumir? Por não conhecermos o comportamento da Equação (2.11) quando uma nova possível solução é submetida à ela, não podemos afirmar o que podem ser considerados valores altos e nem o que podem ser considerados valores baixos.

Para fins didáticos, tomemos como exemplo o gráfico típico de uma derivada numérica com diferentes valores para h , exposta na Figura 4.

Figura 4 – Gráfico típico de uma derivada numérica

Em intervalos em que h é pequeno, a derivada numérica se torna “ruidosa” por erro de arredondamento; em intervalos em que h é grande, a derivada numérica apresenta erro de truncamento. Com “sorte” há uma faixa no meio em que a derivada numérica e a derivada verdadeira se aproximam, como exemplificado no gráfico da Figura 4. Como não sabemos o valor da derivada verdadeira, tomamos valores de h dentro de uma faixa em que a derivada numérica não varia muito, na esperança que corresponda a faixa em que ambas as derivadas, numérica e verdadeira, sejam próximas.

A solução utilizada visando o aprimoramento do método foi efetuarmos uma busca em h com os valores do conjunto $\mathbb{C} = \{10^{-10}, 10^{-9}, \dots, 10^9, 10^{10}\}$. Claro que poderíamos adicionar

outros elementos ao conjunto ou mudá-lo completamente, contudo, as possibilidades são infinitas e por isso optamos por valores que abrangem diversos intervalos.

Outra questão surge em decorrência a busca no valor de h . Calcularmos $\nabla(\Psi(g_t))$ para todo valor possível de h para toda instância demandaria muito esforço computacional devido as diversas varreduras necessárias. A solução encontrada foi efetuarmos pequenas perturbações em um elemento aleatório na matriz g_t que representa o ponto atual dentro do espaço de busca e encontrar o valor de h em que a derivada numérica não varie muito, ao mesmo tempo que evita desperdício de recursos computacionais, como mostra o Algoritmo 2.

Algoritmo 2 – Pseudocódigo para encontrar h que minimize erros de arredondamento

- 1: $x_k \leftarrow$ escolha aleatoriamente um elemento em g_t ;
- 2: $g_t \leftarrow$ ponto (solução) atual dentro do espaço de busca;
- 3: D ; $\leftarrow$ armazenará todos os valores da derivada para cada possível valor de h , que são elementos em $\mathbb{C}$. Uma vez que encontremos h que minimize erros, calcularemos $\nabla(\Psi(g_t))$.
- 4: **para** $h \in \mathbb{C} = \{10^{-10}, 10^{-9}, \dots, 10^9, 10^{10}\}$ **faça** ▷ Para cada valor em $\mathbb{C}$.
- 5: $D_h \approx \frac{1}{2h} \left[J(\Psi^{-1}(g_t(x_k + h))) - J(\Psi^{-1}(g_t(x_k - h))) \right]$; ▷ Enfatizando, note que teremos valores diferentes da derivada para cada possível valor para h , que são elementos em $\mathbb{C}$.
- 6: **fim para**
- 7: Após o laço de repetição acima, o valor de h será encontrado através de um deslocamento, como se segue:

$$h = [0 \ D_h] - [D_h \ 0].$$

Executaremos o processo descrito através do Exemplo 1 em que h encontrado a partir do Algoritmo 2 será aplicado para todo elemento g_t , e submeteremos $\nabla(\Psi(g_t))$ em conjunto com g_t a Equação (4.13) para encontrarmos o próximo ponto da sequência.

A Equação (4.13) nos conduz a outra alteração no método, também uma busca, mas dessa vez no tamanho do passo α . No método de gradiente “puro”, o valor de α na Equação (4.13) leva ao maior decrescimento da função objetivo J . Para simplificar e acelerar o método, consideramos apenas uma busca finita no conjunto $\mathbb{A} = \{10^{-10}, 10^{-9}, \dots, 10^0, 0, 10^1, \dots, 10^9, 10^{10}\}$, uma vez que cada avaliação da função objetivo envolve a solução de uma equação Lyapunov (2.7) (caso a resolução via sistema linear, como dito na Observação 1 e demonstrada no Apêndice B, não seja possível), de forma que a busca pelo valor ótimo para α demanda muito esforço computacional e tempo. Claro que poderíamos adicionar valores ao conjunto ou modificá-lo completamente, contudo, como as possibilidades também são infinitas e não saberíamos o que considerar, optamos por um conjunto que satisfaz diversas variabilidades.

Note que os conjuntos $\mathbb{A}$ e $\mathbb{C}$ são levemente diferentes; o conjunto $\mathbb{A}$ possui o número 0 (zero), em que o cálculo pela Equação (4.13) resulta no próprio ponto. Em princípio pode soar estranho submetermos uma função a um método cujo objetivo é encontrar um ponto do espaço de busca que decresça seu valor a cada passo, mas não o faz. Entretanto, tal particularidade

não é incomum quando aplicada em problemas do “mundo real”, em que múltiplas variáveis influenciam no resultado, e embora o conjunto $\mathbb{A}$ tenha sido criado para abranger ampla gama de possibilidades, tais possibilidades podem ser suficientes para a resolução de algumas instâncias, mas não para outras, em que o “ângulo” de descida pode assumir valores tão minúsculos/variados que os valores no conjunto $\mathbb{A}$ não englobam; além do mais, o método pode ter atingido um mínimo local da função. Portanto, quando atribuímos 0 para α garantimos que o próximo elemento da sequência seja, no mínimo, igual ao ponto atual e não maior.

O uso do elemento 0 (zero) do conjunto $\mathbb{A}$ não só pode indicar que a instância submetida ao método não possui solução, como também que a função atingiu um mínimo local dentro do espaço de busca, embora não poderemos afirmar por não conhecê-lo. Visando testar a hipótese anterior, implementamos o seguinte artifício com o intuito de movermos o ponto encontrado para outro local dentro do espaço de busca: caso o ponto g_{t+1} seja igual ao ponto g_t somaremos ruído gaussiano à $\nabla(\Psi(g_t))$ e re-efetuaremos a busca de α seguindo a Equação (4.13) com o objetivo de encontrar um novo ponto que seja menor do que o ponto anterior. Contudo, como não é garantido que esse artifício funcione e não podemos executar o algoritmo indefinidamente, a “solução” encontrada foi executarmos esse artifício um número máximo de vezes, cinco neste trabalho, em e caso de insucesso, consideramos como solução o próprio ponto, ou seja, $g_{t+1} = g_t$. A escolha por tal quantidade não se dá por nenhum motivo específico além de praticidade, já que um número alto de tentativas poderia desperdiçar recursos computacionais.

A parte a situação descrita no parágrafo anterior, esse método adiciona ruído gaussiano *somente* no ponto inicial em sua inicialização e não dos demais pontos da sequência, diferentemente do AG, que acrescenta ruído gaussiano a cada elemento da prole agindo como elemento mutagênico.

Apesar desse método possuir convergência lenta, na prática ele é muito utilizado. Isso porque o método é teoricamente simples e requer apenas o cálculo da primeira derivada da função em cada ponto g_t além de ser referência para métodos mais avançados (LUENBERGER, 1984). Claro que, dada as alterações efetuadas e descritas, a convergência mais não é garantida.

Neste trabalho, também executaremos o MG para diferentes períodos de tempo e maiores detalhes assim como resultados serão abordados no Capítulo 5.

Algoritmo 3 – Pseudocódigo demonstrando o funcionamento do MG descrito nesse capítulo

- 1: Submeter o controlador em (2.4), o controlador dependente do modo θ de operação, à Equação (4.4).
 - 2: Gerar o ponto inicial dentro do espaço de busca de acordo com a Equação (3.6).
 - 3: Adicionar ruído (Def. 4) de acordo com a Equação (4.6).
 - 4: Avaliar a aptidão do ponto gerado com relação à solução ótima do problema (controlador com θ observável) de acordo com a Equação (2.11).
 - 5: **enquanto** condição de parada não for satisfeita - neste trabalho, tempo máximo de execução **faça**
 - 6: Calcule a derivada do ponto g_t de acordo com a Equação (4.10).
 - 7: Antes de calcularmos a derivada ($\mathbf{d}(t) = -\nabla(\Psi(g_t))$) devemos encontrar a variação h de acordo com o Algoritmo 2.
 - 8: Calcular a derivada do ponto de acordo com o Exemplo 1.
 - 9: Gerar o próximo elemento da sequência (g_{t+1}) de acordo com a Equação (4.13).
 - 10: Avaliar a aptidão do novo ponto com relação à solução ótima do problema (controlador com θ observável) de acordo com a Equação (2.11).
 - 11: **se** $J(g_t) = J(g_{t+1})$ **então**:
 - 12:
 - 13: **enquanto** $J(g_t) = J(g_{t+1})$ ou número máximo de tentativas não for atingido **faça**
 - 14: Ruído será adicionado à derivada do ponto g_t :
 - $$\nabla(\Psi(g_t)) = \nabla(\Psi(g_t)) + N(\mu, \sigma^2).$$
 - 15: O próximo elemento da sequência será gerado novamente e reavaliado de acordo com as Equações (4.13) e (2.11), respectivamente.
 - 16: **fim enquanto**
 - 17: **fim se**
 - 18: **fim enquanto**
-

ANÁLISE DE RESULTADOS I

Todo algoritmo deve possuir uma condição de parada, caso contrário ficaria executando indefinidamente. Neste trabalho, estamos lidando com duas heurísticas distintas no que diz respeito à maneira com que atuam no espaço de busca, uma vez que o AG inicializa com várias soluções possíveis, enquanto o MG com apenas uma, como explanado nos Capítulos 3 e 4, respectivamente. Portanto, visando um método de avaliação que não beneficie um método em relação ao outro, optamos por avaliar os resultados provenientes de ambos os algoritmos atribuindo-lhes tempo máximo de execução. De fato, consideramos três valores de tempo máximo: 10 segundos, 1 minuto e 10 minutos. O tempo de 10 minutos foi definido em simulações preliminares (com 100 instâncias), sendo o tempo em que o AG já havia resolvido cerca de 90% das instâncias que resolveria se a ele fosse dado um tempo muito maior (de 1 hora, por exemplo).

Além disso, submetemos as instâncias aos algoritmos de duas maneiras: na primeira, mil instâncias seguindo as restrições temporais apresentadas no parágrafo anterior; na segunda, cada uma das mil instâncias foi executada dez vezes, também seguindo as mesmas restrições temporais e o motivo será explicado posteriormente.

As instâncias submetidas aos métodos foram geradas utilizando o código desenvolvido pelo nosso grupo de pesquisa e permite a inclusão de diferentes propriedades da teoria de controle, e detalhes da implementação podem ser encontrados em (SILVA, 2012) e (BORTOLIN, 2012). Algumas alterações foram efetuadas no código visando adequá-lo ao problema neste trabalho, e mais informações podem ser obtidas no Apêndice C.

Observação 5. Para simplificar a escrita, quando nos referirmos a “**realização única**” estaremos fazendo menção ao caso em que cada instância foi submetida aos métodos somente uma vez; quando nos referirmos a “**inúmeras realizações**”, estaremos fazendo menção ao caso em que cada instância foi submetida aos métodos dez vezes.

5.1 Definição das instâncias e dos resultados

A partir do gerador de instâncias desenvolvido pelo nosso grupo de pesquisa, podemos gerar infinitas instâncias, todas com custos distintos. Entretanto, neste trabalho, utilizaremos apenas instâncias que possuem “custo finito” (controle ótimo com observação de θ); as instâncias que não satisfazem essa condição foram substituídas por outras que satisfazem.

É importante ter em mente que, quando dizemos “custo finito”, estamos nos referindo a valores tangíveis. Qualquer resultado com custo maior do que 10^{100} será considerado como *não tendo solução* ou, equivalentemente, que o custo associado é infinito, já que custos tão altos sugerem fortemente que o sistema controlado tem comportamento instável, inviável na prática. Nas figuras que exibem os custos, tais custos são saturados em 10^{10} (valor máximo dos eixos, e portanto, aparecem nas bordas das figuras).

Instâncias com “custo finito” quando θ é observável podem deixar de ter “custo finito” quando θ não é observado, e esses casos também foram contabilizados nas estatísticas como ocorrências em que os métodos não conseguiram encontrar solução.

Também podemos nos deparar com instâncias que excederam o tempo máximo para geração de indivíduos suficientes para realizarmos o *crossover* no AG ou para a geração de ponto inicial no MG. Com relação ao AG, no mínimo dois indivíduos serão necessários e caso tal condição seja atendida, o algoritmo prossegue com o *crossover* (embora nesta situação a variabilidade na população seja mínima, provinda somente através de ruído gaussiano na prole). Em caso de tempo máximo excedido, esses casos também serão contabilizados nas estatísticas como ocorrências em que os métodos não conseguiram encontrar solução.

Outra questão é que não há garantias que o custo $J(\mathbf{R}_n)$, ou seja, o custo para o controlador (2.17) definido em (2.11), seja finito para um dado controlador $\mathbf{R}_n$. Levando em conta que as possibilidades de escolha das matrizes A_c , B_c , C_c e D_c são infinitas, podemos nos deparar com problemas em que a chance de encontrar matrizes levando a custo finito seja muito pequena (ou até mesmo zero). Assim, todos os problemas que não conseguimos inicializar o método de forma simples, tomando controladores seguindo o proposto na Seção 3.2, foram considerados como soluções com custo infinito associado.

A quantidade de estados de Markov para cada instância, ou seja, a quantidade de estados θ , também varia. Neste trabalho, temos instâncias com dois, três, quatro e cinco estados de Markov, ou seja, *matrizes de probabilidade de transição* 2×2 , 3×3 , 4×4 ou 5×5 ; com relação a tais matrizes, temos estados **transientes**, **recorrentes**, **absorventes** e **periódicos** e detalhes podem ser encontrados no Apêndice A.

Vale lembrar que $\mathbf{R}_0$, apresentado em (2.4), é o controlador ótimo para o problema auxiliar, com observação de θ , que tem custo menor ou igual ao caso considerado neste trabalho.

Com relação as estatísticas, o *erro padrão* é calculado utilizando a equação

$$\sigma_p = \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}, \quad (5.1)$$

em que $\hat{p}$ é o erro com relação à porcentagem referente ao caso analisado, e n é a quantidade total de instâncias analisadas, no caso, 1000 instâncias. Podemos interpretar o *erro padrão* como o quão distante a média de uma amostra está da média real. Utilizamos o *erro padrão* por não submetermos as instâncias aos métodos diversas vezes e, assim sendo, não possuímos a uma média; o objetivo com a utilização do *erro padrão* é corroborarmos a porcentagem de estatística referente a cada caso analisado.

5.2 Influência do tamanho da população inicial nos resultados do Algoritmo Genético

Como citado brevemente na Seção 3.1, o tamanho da população é uma das mais importantes escolhas enfrentadas por qualquer usuário de AGs. Se uma população é muito pequena, o AG pode convergir muito rápido; se for muito grande, o AG pode desperdiçar recursos computacionais, já que a espera para obter melhoria no *fitness* pode ser muito longa.

Na literatura, existem vários estudos sobre a influência da população no desempenho geral de um AG. Em (ROEVA; FIDANOVA; PAPRZYCKI, 2013), um problema foi proposto e o resultado final foi avaliado com base na população inicial, em que concluiu-se que quanto maior a população, melhor o *fitness*. Em (EIBEN; HINTERDING; MICHALEWICZ, 1999) é dito que na década de 80, uma população com cerca de 100 indivíduos era escolhida, contudo, sem justificativa para tal escolha. Com alguns estudos efetuados por (JONG, 1975), chegou-se à conclusão de que uma população inicial de 50 indivíduos seria suficiente para o AG ter um desempenho satisfatório, contudo, tal população pode variar dado o problema. Por outro lado, Grefenstette (GREFENSTETTE, 1986), utilizando uma abordagem um pouco diferente daquela utilizada por De Jong, chegou à conclusão que uma população inicial com 30 indivíduos seria suficiente para chegarmos a um *fitness* desejável, entretanto, a quantidade de indivíduos na população também dependeria do problema. Em (MAN; TANG; KWONG, 1996), é dito que grandes populações são compostas por 100 indivíduos e pequenas populações são compostas por 30 indivíduos e tais guias foram baseados em (JONG, 1975) e (GREFENSTETTE, 1986). Em (ZHAO *et al.*, 2011) a população inicial é de 50 indivíduos, contudo, sem justificativa. E em (GONG; SUN; MIAO, 2016) é dito que a população inicial varia de acordo com o AG utilizado.

Analisando tais estudos, uma das conclusões é que a quantidade de indivíduos na população inicial dependerá do problema que será submetido ao AG. Portanto, com o objetivo de encontrar a quantidade de indivíduos que irão compor a população inicial para o problema estudado neste trabalho e visando balancear tal quantidade com os recursos computacionais disponíveis, realizamos um teste preliminar, submetendo mil instâncias ao AG, cada uma com

tempo máximo de execução fixados em 1 minuto e com duas quantidades distintas de indivíduos na população inicial (50 e 100 indivíduos).

A princípio, a população inicial é inicializada aleatoriamente para ambos os casos, ou seja, a população inicial de ambos os casos será distinta. Como ruído gaussiano é adicionado à cada indivíduo, de acordo com (3.8), a chance de obtermos indivíduos/populações iguais é quase nula.

Para este teste preliminar, a escolha do tempo de execução de 1 minuto dá-se pelo fato de que se optássemos por executarmos apenas por 10 segundos, o AG poderia não ter tempo suficiente para gerar a população inicial com 100 indivíduos, o que faria com que o AG inicializado com 50 indivíduos obtivesse resultados superiores, comprometendo a análise. Se optássemos por executar durante 10 minutos, estaríamos desperdiçando recursos computacionais, já que, segundo testes efetuados durante a implementação do algoritmo, 1 minuto é mais do que o suficiente para a geração da população inicial e avançarmos algumas gerações.

A avaliação dos resultados foi feita utilizando estatísticas simples e são as mesmas que foram utilizadas nos trabalhos (SILVA, 2012) e (BORTOLIN, 2012), com a justificativa de estarmos trabalhando com um controlador dinâmico e ambos trabalhos apresentarem uma abordagem para controladores estáticos.

As mesmas instâncias foram submetidas ao AG com diferentes tamanhos de população, e após a execução do algoritmo, armazenamos os melhores custos obtidos para cada uma das instâncias analisadas na forma de pares ordenados $(J_{50} \ J_{100})$, em que o primeiro valor se refere ao algoritmo com população de tamanho 50 e o segundo valor ao caso com tamanho 100. Os pares de custos obtidos estão dispostos graficamente nas Figuras 5 e 6 e nas Tabelas 1 e 2.

Devido a grande diferença de valores entre os resultados obtidos após submetermos cada uma das instâncias aos métodos, os gráficos dispostos na Figura 5 e na Figura 7 estão na escala logarítmica.

Para facilitar, utilizaremos a notação AG50 para o AG com população tamanho 50, e AG100 para o AG com população tamanho 100.

Figura 5 – Custos obtidos com o AG50 vs. custos obtidos com AG100 - populações distintas

Figura 6 – Estatísticas gerais para populações distintas

Tabela 1 – Estatísticas gerais para populações distintas

	%	erro padrão
resolução SOMENTE pelo AG50	3,7%	0,5969
resolução SOMENTE pelo AG100	6,1%	0,7568
ambos os casos resolvem	30,7%	1,4586
nenhum dos casos resolve	59,5%	1,5523

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 2 – Estatísticas específicas para populações distintas

	%	erro padrão
AG50 melhor do que AG100	2,8%	0,5217
AG100 melhor do que AG50	6,8%	0,7961
sem diferença significativa entre o AG50 e o AG100	21,1%	1,2903

Analizando a Figura 5, cada ponto do gráfico representa o melhor custo para cada uma das instâncias submetidas aos métodos para θ não observável. O ponto no canto superior direito do gráfico corresponde a instâncias em que nenhum dos casos foi capaz de encontrar solução, equivalente a 59,5% do total; os pontos paralelos ao eixo das ordenadas correspondem a instâncias em que somente AG50 encontrou solução, equivalente a 6,7% do total; os pontos paralelos ao eixo das abscissas correspondem a instâncias em que somente o AG100 encontrou solução, equivalente a 6,1% do total e finalmente, no centro do gráfico estão dispostas as instâncias em que ambos os casos encontraram solução, equivalente a 30,7% do total, embora em 21,1% não exista diferença significativa entre eles.

Com relação às instâncias em que ambos os métodos encontraram solução, verificamos que o AG100 destaca-se em relação ao AG50. Esse comportamento, claro, advém da diferença entre a quantidade de indivíduos na população e também da aleatoriedade inerente ao problema estudado neste trabalho, tanto na geração da população inicial quanto na adição de ruído gaussiano via *mutação* na etapa de *crossover*.

Avaliando a porcentagem em que somente um dos casos obteve solução, verificamos que não há aumento significativo na quantidade de instâncias que possuem solução mesmo dobrando a quantidade de indivíduos na população (temos um aumento de apenas 2,4%).

A fim de corroborar os dados, efetuamos um segundo teste: executamos o AG50 replicando esses mesmos 50 indivíduos no AG100, ou seja, ambos os casos possuem os mesmos indivíduos com a diferença de que acrescentamos 50 indivíduos gerados aleatoriamente no segundo caso. Os dados obtidos estão dispostos nas Figuras 7 e 8 e nas Tabelas 3 e 4.

Figura 7 – Custos obtidos com o AG50 vs. custos obtidos com AG100 - população replicada

Figura 8 – Estatísticas gerais para população replicada

Tabela 3 – Estatísticas gerais para população replicada

	%	erro padrão
resolução SOMENTE pelo AG50	0,3%	0,1729
resolução SOMENTE pelo AG100	3,7%	0,5969
ambos os casos resolvem	33,3%	1,4903
nenhum dos casos resolve	62,7%	1,5293

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 4 – Estatísticas específicas para população replicada

	%	erro padrão
AG50 melhor do que AG100	0,2%	0,1413
AG100 melhor do que AG50	5,4%	0,7147
sem diferença significativa entre o AG50 e o AG100	27,7%	1,4152

O gráfico na Figura 7 é semelhante ao gráfico na Figura 5, o que indica que ambos os casos são equivalentes e corroborados pela Tabela 3. Podemos ver que a porcentagem de instâncias em que somente o caso com 50 indivíduos resolve diminuiu, o que era esperado, já que esses indivíduos foram replicados no AG inicializado com 100 indivíduos.

É interessante notar que no gráfico da Figura 5 alguns pontos que estão mais espalhados comparado com o gráfico da Figura 7. Isto acontece porque no gráfico da Figura 5 ambas as populações são distintas, logo, é coerente que um caso encontre resultados melhores do que outro e alguns resultados diverjam. Quando replicamos a população, a tendência é que ambos os casos encontrem os mesmos resultados e todos os pontos se agrupem, exatamente como observado.

A porcentagem de instâncias em que ambos os métodos encontraram solução mantém-se praticamente a mesma quando comparada com o teste anterior. Contudo, quando replicamos a população, notamos que a superioridade do AG100 intensifica-se, e esse comportamento é explicado pelo fato de que ambos os casos possuem os mesmos indivíduos na população, embora, devido à aleatoriedade inerente ao problema via ruído gaussiano, em algumas poucas instâncias, o método em “desvatagem” encontrará soluções melhores.

Ainda com relação à porcentagem de instâncias em que ambos os métodos encontraram solução, notamos que, embora tenha mantido-se praticamente a mesma quando comparada com o teste anterior, quando analisamos a estatística referente a porcentagem de instâncias em que os resultados obtidos não possuem diferença significativa, vemos aumento de 21,1% para 27,7%; essa diferença também advém do fato de que ambos os casos inicializaram com os mesmos indivíduos na população inicial, impedindo variabilidade na população e acarretando em resultados similares.

No que diz respeito a porcentagem de instâncias em que nenhum dos métodos foi capaz de encontrar solução, os resultados também assemelham-se, indicando que o aumento de indivíduos na população inicial não contribuiu significativamente para que mais instâncias pudessem ser resolvidas.

Embora os testes tenham apontado ligeira vantagem do AG100 na configuração de testes com 1 minuto, a diferença entre os resultados não é proporcional à quantidade de indivíduos com que os métodos foram inicializados. Avaliamos também que o AG100 pode ter o desempenho muito prejudicado nos testes com 10 segundos, além do fato de que nos testes com 10 minutos o AG50 normalmente obtém muito mais gerações que o AG100, e queremos avaliar se há uma “saturação” do resultado do AG em função do número de gerações; por esse motivo, optamos por inicializar o AG com 50 indivíduos na população.

5.3 Algoritmo Genético versus Método Quasi-Gradiente

Nesta seção comparamos os resultados obtidos pelo AG e pelo MG seguindo as diretrizes apresentadas no início desta seção. A população inicial do AG é de 50 indivíduos devido as conclusões obtidas na seção anterior.

Devido à grande diferença entre valores dos resultados provindos das instâncias submetidas aos métodos, os gráficos dispostos na Figura 9, Figura 11 e na Figura 13 estão na escala logarítmica. Todos os gráficos apresentados nesta seção e nas seções seguintes possuem a mesma interpretação dos gráficos apresentados na Seção 5.2, com exceção dos resultados e análises dos dados, claro.

Primeiramente, cada instância foi submetida somente uma vez em cada método. Após 10 segundos de execução, comparamos diretamente o desempenho de ambos os métodos para uma mesma instância e obtivemos os seguintes resultados, dispostos nas Figuras 9 e 10 e nas Tabelas 5 e 6:

Figura 9 – Custos obtidos com o AG x Custos obtidos com o MG - 10 segundos (realização única)

Figura 10 – Estatísticas AG x MG - 10 segundos (realização única)

Tabela 5 – Estatísticas gerais AG x MG - 10 segundos (realização única)

	%	erro padrão
SOMENTE o AG resolve	5,1%	0,6957
SOMENTE o MG resolve	3%	0,5394
ambos os métodos resolvem	13,3%	1,0738
nenhum dos métodos resolve	78,6%	1,2969

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 6 – Estatísticas específicas AG x MG - 10 segundos (realização única)

	%	erro padrão
AG melhor que o MG	1,9%	0,4317
MG melhor que o AG	2,8%	0,5217
sem diferença significativa entre os métodos	8,6%	0,8866

Em 78,6% dos casos, nenhum dos métodos conseguiu encontrar solução. O pouco tempo disponibilizado aos métodos impediu não somente que suas diretrizes de inicialização fossem cumpridas, mas também que avançassem gerações/iterações suficientes, contribuindo para o alto número de instâncias sem solução, já que, lembrando, instâncias que excederam seu tempo máximo de execução para a geração da população inicial/ponto inicial são contabilizadas como instâncias que não possuem solução (e no gráfico de custo, como instâncias com custo 10^{10}). Em 5,1% dos casos, somente o AG encontrou solução, e em 3% dos casos, somente o MG.

A porcentagem de instâncias em que ambos os métodos encontraram solução foi de 13,3%. Contudo, não existem diferença significativa entre os resultados encontrados pelos métodos em 8,6% dos casos analisados, e embora o AG tenha encontrado soluções para mais instâncias, as soluções obtidas pelo MG foram relativamente melhores, com 1,9% contra 2,8% respectivamente.

Reexecutamos os algoritmos aumentando do tempo máximo de execução para 1 minuto e obtivemos os resultados dispostos nas Figuras 11 e 12, e nas Tabelas 7 e 8:

Figura 11 – Custos obtidos com o AG x Custos obtidos com o MG - 1 minuto (realização única)

Figura 12 – Estatísticas AG x MG - 1 minuto (realização única)

Tabela 7 – Estatísticas gerais AG x MG - 1 minuto (realização única)

	%	erro padrão
SOMENTE o AG resolve	17,8%	1,2096
SOMENTE o MG resolve	0,6%	0,2442
ambos os métodos resolvem	16,6%	1,1766
nenhum dos métodos resolve	65%	1,5083

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 8 – Estatísticas específicas AG x MG - 1 minuto (realização única)

	%	erro padrão
AG melhor que o MG	2%	0,4427
MG melhor que o AG	4,7%	0,6693
sem diferença significativa entre os métodos	9,9%	0,9445

Após 1 minuto de execução, a porcentagem de instâncias em que os métodos não conseguiram encontrar solução corresponde a 65% dos casos. O aumento no tempo máximo de execução possibilitou não somente que ambos os métodos cumprissem suas diretrizes de inicialização, mas também que avançassem gerações/iterações e, consequentemente, que encontrassem soluções para mais instâncias.

O aumento no tempo máximo de execução também contribuiu para que a porcentagem de instâncias que ambos os métodos encontraram solução aumentasse de 13,3% para 16,6% embora em 9,9% dos casos não haja diferença significativa entre eles.

Quando analisamos o desempenho dos métodos com relação a soluções distintas, notamos que a porcentagem de instâncias que somente o AG resolveu aumentou de 5,1% para 17,8%, enquanto a porcentagem de instâncias que somente o MG resolveu diminuiu de 3% para 0,6%. Conclui-se que, com tempo suficiente, o AG é capaz de encontrar soluções para instâncias que anteriormente somente o MG havia encontrado.

Dentre as instâncias em que ambos os métodos encontraram solução, em 4,7% dos casos o MG saiu-se melhor, contra 2,8% do AG. Para esta restrição temporal, o observado no teste anterior é levemente intensificado.

Finalmente, reexecutamos os algoritmos aumentando o tempo máximo de execução para 10 minutos e obtivemos os resultados dispostos nas Figuras 13, e 14 e nas Tabelas 9 e 10:

Figura 13 – Custos obtidos com o AG x Custos obtidos com o MG - 10 minutos (realização única)

Figura 14 – Estatísticas gerais AG x MG - 10 minutos (realização única)

Tabela 9 – Estatísticas gerais AG x MG - 10 minutos (realização única)

	%	erro padrão
SOMENTE o AG resolve	18,9%	1,2381
SOMENTE o MG resolve	0,3%	0,1729
ambos os métodos resolvem	16,6%	1,1766
nenhum dos métodos resolve	64,2%	1,5160

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 10 – Estatísticas específicas AG x MG - 10 minutos (realização única)

	%	erro padrão
AG melhor que o MG	1,1%	0,3298
MG melhor que o AG	5,5%	0,7209
sem diferença significativa entre os métodos	10%	0,9487

Após 10 minutos de execução, embora a porcentagem de casos sem solução tenha diminuído de 65% para 64,2%, a diferença não seja significativa e proporcional ao tempo disponibilizado ao método.

A porcentagem de instâncias que ambos os métodos resolvem mantém-se em 16,6% e não há diferença significativa entre 10% dos casos, praticamente os mesmos resultados obtidos com a restrição temporal anterior.

Com relação aos casos que somente um dos métodos resolve, notamos que a porcentagem de instâncias que somente o AG resolveu aumentou de 17,8% para 18,9%, enquanto a porcentagem de instâncias que somente o MG resolveu diminuiu de 0,6% para 0,3%. A tendência observada no teste anterior em que o MG sobressaiu-se com as melhores soluções se mantém, mesmo que em menor proporção. Embora a porcentagem de casos em que ambos os métodos encontram solução possa ser considerada relativamente baixa, o MG saiu-se melhor que o AG devido à maneira com que atua dentro do espaço de busca.

Embora mais tempo disponibilizado aos métodos tenha permitido que mais instâncias pudessem ser resolvidas, a melhora obtida não é proporcional ao tempo gasto. Portanto, 1 minuto é tempo suficiente para os métodos obterem resultados que permitam que um usuário opte por um método ou por outro, dado que 10 segundos são insuficientes para que os métodos cumpram suas diretrizes de inicialização e/ou avancem gerações/iterações suficientes para termos certeza acerca dos resultados. Entretanto, mais tempo disponibilizado aos métodos revelou a tendência que o MG possui de encontrar melhores soluções.

A partir dos resultados obtidos, também pode-se concluir que se o período de tempo disponibilizado aos métodos é suficientemente grande, o AG consegue encontrar soluções para instâncias que anteriormente somente o MG havia encontrado e, em menor proporção, os métodos

foram capazes de encontrar solução para mais instâncias. Também verificamos que, dentre as instâncias em que o ambos os métodos foram capazes de encontrar solução, o MG foi capaz de encontrar soluções que se destacaram em relação ao AG.

Com relação ao AG, devido à adição de ruído gaussiano agindo como elemento mutagênico em cada indivíduo g_i^{ell} (Equação (3.13)) resultantes via *crossover*, é intuitivo pensar que os resultados finais obtidos pelo método foram influenciados por tal procedimento, ou melhorando-os ou piorando-os; contudo, isso não acontece com o MG, uma vez que os valores de α , definidos no conjunto $\mathbb{A}$, possui o elemento 0 (zero) e impede que o custo $J(\Psi^{-1}(g_{t+1}))$ seja maior do que $J(\Psi^{-1}(g_t))$.

Além disso, o MG pode inicializar e/ou convergir para um mínimo local que seja maior do que 10^{100} e o AG pode gerar sua população inicial somente com indivíduos considerados ruins em sua composição e não obter resultados válidos mesmo após o *crossover*, e tais decorrências entram nas estatísticas como instâncias que não possuem solução.

Ademais, estamos comparando dois métodos distintos em suas diretrizes de inicialização, uma vez que o AG inicializa com várias soluções possíveis e o MG com apenas uma e isso pode contribuir para que o AG encontre mais soluções que o MG.

Portanto, a fim de prover a oportunidade ao MG para trabalhar com várias possíveis soluções, reexecutamos os métodos submetendo cada instância aos métodos 10 vezes seguindo as restrições temporais apresentadas no início deste capítulo.

Além disso, retornando ao que foi brevemente citado na Seção 4.1, a adição de ruído gaussiano na geração do ponto inicial para o MG exerce papel muito mais significativo, uma vez que maximiza a possibilidade do método inicializar em locais distintos dentro do espaço de busca.

De maneira análoga aos gráficos anteriores, devido à grande diferença entre valores dos resultados provindos das instâncias submetidas aos métodos, os gráficos dispostos na Figura 15, Figura 17 e na Figura 19 estão na escala logarítmica.

Primeiramente, cada instância foi submetida aos métodos dez vezes durante 10 segundos, e os seguintes resultados foram obtidos, dispostos nas Figuras 15 e 16, e nas Tabelas 11 e 12:

Figura 15 – Custos obtidos com o AG x Custos obtidos com o MG - 10 segundos (inúmeras realizações)

Figura 16 – Estatísticas AG x MG - 10 segundos (inúmeras realizações)

Tabela 11 – Estatísticas gerais AG x MG - 10 segundos (inúmeras realizações)

	%	erro padrão
SOMENTE o AG resolve	6,07%	0,2388
SOMENTE o MG resolve	3,15%	0,1747
ambos os métodos resolvem	13,18%	0,3383
nenhum dos métodos resolve	77,6%	0,4169

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 12 – Estatísticas específicas AG x MG - 10 segundos (inúmeras realizações)

	%	erro padrão
AG melhor que o MG	2,53%	0,1570
MG melhor que o AG	2,22%	0,1473
sem diferença significativa entre os métodos	8,43%	0,2778

Confrontando os dados apresentados nas Tabelas 11 e 12 diretamente com os dados obtidos quando submetemos as instâncias aos métodos uma única vez com as mesmas restrições temporais, vemos que a tendência se mantém.

A porcentagem de instâncias em que ambos os métodos encontraram solução corresponde a 13,18% dos casos para **inúmeras realizações** contra 13,3% para o caso com **realização única**, aumento irrisório e corroborado pela porcentagem em que não existe diferença significativa entre ambos os métodos, 8,43% contra 8,6% respectivamente.

A porcentagem de casos em que nenhum dos métodos encontraram solução também se assemelha, com 77,6% para **inúmeras realizações** contra 78,6% para **realização única**. Mesmo para **inúmeras realizações**, o pouco tempo disponibilizado aos métodos impossibilitou que as diretrizes de inicialização dos métodos fossem cumpridas e que cálculos fossem realizados. Com relação à porcentagem de instâncias em que somente um dos métodos foi capaz de encontrar soluções, a tendência também se mantém (existe diferença, mas pouco significativa).

Com relação às instâncias em que ambos os métodos encontraram solução, os métodos também assemelham-se, exceto pelo fato de que, pela primeira vez durante os testes, o AG sobressaiu-se ao MG. Entretanto, esse resultado sofre influência do pouco tempo disponibilizado aos métodos.

Os dados mostraram que, com pouco tempo disponibilizado aos métodos, não existe diferença significativa entre submeter as instâncias aos métodos uma única vez ou inúmeras vezes. Além disso, pouco tempo de execução contribui para um número alto de instâncias sem solução.

Reexecutamos os métodos aumentando o tempo máximo de execução para 1 minuto e obtivemos os resultados dispostos nas Figuras 17 e 18, e nas Tabelas 13 e 14:

Figura 17 – Custos obtidos com o AG x Custos obtidos com o MG - 1 minuto (inúmeras realizações)

Figura 18 – Estatísticas AG x MG - 1 minuto (inúmeras realizações)

Tabela 13 – Estatísticas gerais AG x MG - 1 minuto (inúmeras realizações)

	%	erro padrão
SOMENTE o AG resolve	18,03%	1,2157
SOMENTE o MG resolve	0,57%	0,2381
ambos os métodos resolvem	16,11%	1,1625
nenhum dos métodos resolve	65,09%	1,5074

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 14 – Estatísticas específicas AG x MG - 1 minuto (inúmeras realizações)

	%	erro padrão
AG melhor que o MG	2,01%	0,4438
MG melhor do que AG	4,38%	0,6472
sem diferença significativa entre os métodos	9,72%	0,9368

O aumento no tempo máximo de execução possibilitou que os métodos encontrassem soluções para mais instâncias, diminuindo a porcentagem de 77,6% para 65,9%, uma vez que possibilitou não somente que as diretrizes de inicialização dos métodos fossem cumpridas, mas também que os métodos avançassem gerações/iterações.

Também notamos que a tendência identificada no teste anterior é intensificada. Houve aumento de 6,07% para 18,03% na porcentagem de instâncias que somente o AG resolve e a consequente diminuição de 3,15% para somente 0,57% na porcentagem de instâncias em que somente o MG resolve, confirmando que com tempo suficiente disponibilizado aos métodos, o AG é capaz de encontrar as mesmas soluções que o MG.

Também vemos aumento na porcentagem de instâncias em que ambos os métodos encontraram solução de 13,18% para 16,11%, embora a porcentagem sem diferença significativa entre as soluções também tenha aumentado de 8,43% para 9,72%. A tendência do MG em encontrar melhores soluções é intensificada com relação ao teste anterior; novamente, embora o AG tenha encontrado soluções para maior número de instâncias, as melhores soluções são encontradas pelo MG, com 2,01% contra 4,38%, respectivamente.

É interessante notar que apesar do efeito aleatório inerente à inserção de ruído gaussiano agindo como elemento mutagênico a cada indivíduo g_i^{ell} (Equação (3.13)) resultantes via *crossover*, a tendência que o AG possui em encontrar soluções para mais instâncias se mantém mesmo quando submetemos as instâncias aos métodos inúmeras vezes.

Quando comparamos os dados obtidos entre **inúmeras realizações** versus **realização única** para esta restrição temporal, vemos que os dados mantêm-se praticamente os mesmos, o que indica que não faz diferença significativa se submetemos as instâncias aos métodos inúmeras vezes ou apenas uma. A fim de confirmar essa hipótese, aumentamos o tempo máximo de

execução para 10 minutos e obtivemos os resultados dispostos nas Figuras 19 e 20, e nas Tabelas 15 e 16:

Figura 19 – Custos obtidos com o AG x Custos obtidos com o MG - 10 minutos (inúmeras realizações)

Figura 20 – Estatísticas AG x MG - 10 minutos (inúmeras realizações)

Tabela 15 – Estatísticas gerais AG x MG - 10 minutos (inúmeras realizações)

	%	erro padrão
SOMENTE o AG resolve	18,33%	0,3869
SOMENTE o MG resolve	0,42%	0,0647
ambos os métodos resolvem	16,13%	0,3678
nenhum dos métodos resolve	65,12%	0,4766

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 16 – Estatísticas específicas AG x MG - 10 minutos (inúmeras realizações)

	%	erro padrão
AG melhor que o MG	1,08%	0,1034
MG melhor do que AG	5,06%	0,2192
sem diferença significativa entre os métodos	9,99%	0,2999

Após 10 minutos de execução, os resultados pouco alteram-se.

A porcentagem de instâncias em que nenhum dos métodos encontra solução mantém-se praticamente a mesma, revelando que o aumento do tempo máximo de execução mesmo para **inúmeras realizações** não surte efeito expressivo.

O mesmo dá-se para a porcentagem de instâncias em que ambos os métodos encontraram solução que, embora tenha aumentado de 16,11% para 16,13%, é irrisório comparado com o tempo despendido e aos recursos computacionais necessários; podemos concluir o mesmo para as demais estatísticas analisadas neste trabalho.

5.3.1 Discussões sobre o Algoritmo Genético versus Método Quasi-Gradiente

Embora mais tempo disponibilizado aos métodos não contribua para que mais instâncias possam ser resolvidas, confirma a hipótese levantada no teste anterior no que diz respeito ao comportamento de ambos os algoritmos, ou seja, embora a diferença entre os dados exista, ela não é expressiva e nos leva a concluir que é indiferente submetermos as instâncias aos métodos inúmeras vezes ou apenas uma. Contudo, não podemos dizer que o tempo despendido e toda análise efetuada foi em vão, já que nos dá informações suficientes para concluir por qual dos métodos optar.

O parágrafo anterior em conjunto com os resultados obtidos para **realização única** seguindo todas as restrições temporais impostas, nos leva a confirmar que 1 minuto é tempo suficiente para os métodos encontrarem resultados que permitam optar por um ou outro, já que após 10 minutos de execução, os resultados pouco alteram-se e não faz jus ao tempo e recursos computacionais gastos.

Além disso, os dados também mostraram que embora o AG solucione maior número de instâncias, o modo como o MG atua dentro do espaço de busca faz com que se destaque no que diz respeito às melhores soluções. Isso suscita o estudo de que métodos derivados a partir do AG e do MG, que combinam a flexibilidade do AG com a rigidez matemática do MG. Nessa ideia, como o AG mostrou-se ótima opção para encontrar mais soluções, poderia ser utilizado para inicializar o método enquanto o MG para refinar os resultados. Esse tipo de algoritmo será abordado no próximo capítulo.

Com relação a saturação de resultados encontrados após 1 minuto de execução dos algoritmos, uma vez que os resultados pouco alteram-se após esse tempo, levantamos duas hipóteses: a primeira, devido a não convexidade da função objetivo (Equação (2.11)), o que pode fazer com que os algoritmos encontrem somente uma solução ótima local (OLIVEIRA; COSTA, 2015); a segunda, devido a pouca variabilidade genética da população do AG, apesar de todas as técnicas utilizadas e descritas no Capítulo 3 visando sua heterogeneidade (com uma população homogênea, diminuí-se a cobertura do espaço de busca do algoritmo, e, conseqüentemente, sua eficácia). Os dados mostraram efeito semelhante com relação ao MG, uma vez que mesmo a adição de ruído gaussiano somado à *combinação convexa* na inicialização do método para **inúmeras realizações**, o que proporciona a inicialização em diferentes locais dentro do espaço de busca, não resultou em uma diminuição significativa no número de instâncias sem solução.

Finalizando, esses resultados aplicam-se somente no modelo de SLSM estudado neste trabalho, ou seja, o sistema apresentado em (2.1), e não existem garantias que os resultados mantêm-se para diferentes SLSMs.

MÉTODOS ALTERNATIVOS

Os dados obtidos através da comparação entre o AG e o MG apresentados no Capítulo 5 mostraram que embora o AG encontre mais soluções para o problema estudado neste trabalho, muitas das soluções encontradas pelo MG são melhores. Isto motivou o estudo de métodos híbridos, que combinam a flexibilidade do AG com a rigidez matemática do MG.

6.1 Métodos Híbridos

Neste trabalho, consideramos como **métodos híbridos** os métodos que combinam o Algoritmo Genético (Capítulo 3) com o Método Gradiente (Capítulo 4).

6.1.1 Algoritmo Genético-Gradiente

O primeiro experimento com métodos híbridos foi feito entre o AG e o MG, e o “novo método”, que chamamos de Algoritmo Genético-Gradiente (AGG), não estabelece nenhuma modificação drástica em sua estrutura, ou seja, o funcionamento de ambos segue o descrito nos Capítulos 3 e 4, respectivamente. A ideia consiste em aplicar um passo do MG para cada um dos “filhos” resultantes via *crossover* em cada geração do AG.

Como os resultados entre o AG versus MG revelaram que este possui tendência em encontrar melhores resultados enquanto aquele destaca-se pela quantidade de soluções, a proposta com este “novo método” é permitir que o AG encontre o maior número de soluções possíveis enquanto o MG refine tais soluções.

Somente uma alteração foi efetuada: embora o critério de parada do MG original esteja atrelado a um tempo máximo de execução, nesse híbrido alteramos esta condição para executar com uma quantidade máxima de passos. Como o método “principal” continuará sendo o AG, a inicialização ainda contará com várias possíveis soluções dentro do espaço de busca.

Sucintamente, o AGG inicializa sua execução assim como o AG, apresentado no Capítulo 3, ou seja, com 50 indivíduos na população inicial gerados através da Equação (3.6) seguindo a definição de *combinação convexa* (Definição 2), sendo que, à cada um desses indivíduos, é atribuído um *fitness*, de acordo com a Equação (3.11). A partir daí, seleciona indivíduos “pais” e “mães” para o *crossover* com base no método da roleta, de acordo com a Equação (3.12); o *crossover* é feito de acordo com a Equação (3.13), gerando indivíduos “filhos”; em cada um desses “filhos” adicionamos ruído (Definição 4, Equação (3.14)) visando adicionar variabilidade genética à população; cada um desses “filhos” é submetido ao MG, assim como apresentado no Capítulo 4, transformado em uma “função” neste método, que aplica somente um passo à cada um desses “filhos” em cada geração/iteração do algoritmo; o *fitness*, dado pela Equação (3.11), é atribuído à cada “filho” após serem submetidos ao MG; finalizando, esses “filhos” são inseridos na população, substituindo indivíduos, em tese, menos aptos - em tese pela mesma justificativa apresentada no final do Capítulo 3.

Mas isto suscita a questão: quantos passos permitir que o MG efetue dentro do espaço de busca? Poderíamos executar vários testes com diferentes possibilidades e utilizarmos os melhores resultados, contudo, isto é inviável pela quantidade de execuções que seriam necessárias. Portanto, objetivando imparcialidade, permitiremos que o MG dê somente um passo dentro do espaço de busca. Além disso, o MG efetua varreduras nos conjuntos $\mathbb{A}$ e $\mathbb{C}$ (vide Capítulo 4), implicando em um maior esforço computacional, e, assim sendo, mais passos dentro do espaço de busca, além de maior esforço computacional, também demandaria mais tempo de execução para o algoritmo encontrar soluções consideradas satisfatórias.

Observação 6. Note que, neste método, cada indivíduo que compõe a população também é uma vetorização das (Definição 1) matrizes A_c , B_c , C_c e D_c que atuam no controlador independente de θ , apresentando em (2.17); note, também, que como cada um desses indivíduos são gerados a partir da *combinação convexa* das matrizes do controlador em (2.4), ou seja, o controlador dependente do modo θ de operação, o operador φ , apresentado em (3.2), também é necessário.

Observação 7. Note que como o AGG é um híbrido entre o AG apresentado no Capítulo 3 e o MG apresentado no Capítulo 4, e, conseqüentemente, seguem o descrito em tais capítulos, as questões concernentes a ambos os algoritmos e respectivas justificativas se fazem válidas.

Algoritmo 4 – Pseudocódigo demonstrando o funcionamento do AGG descrito nessa seção

- 1: Submeter o controlador em (2.4), o controlador dependente do modo θ de operação, à Equação (3.2).
 - 2: Gerar indivíduos para a população inicial de acordo com a Equação (3.6).
 - 3: Adicionar ruído (Definição 4) à cada um dos 50 elementos da população de acordo com a Equação (3.8).
 - 4: Atribuir o *fitness* a cada um dos indivíduos da população de acordo com a Equação (3.11), submetendo, antes, cada indivíduo à Equação (3.10) para “desvetorizar” cada um dos indivíduos na população.
 - 5: **enquanto** condição de parada não for satisfeita - neste trabalho, tempo máximo de execução **faça**
 - 6: Selecionar 20% dos indivíduos da população, divididos em indivíduos “pais” e indivíduos “mães”, para reprodução de acordo com a Equação (3.12).
 - 7: Cruzar os indivíduos “pais” e os indivíduos “mães” (*crossover*), gerando indivíduos “filhos”, de acordo com a Equação (3.13).
 - 8: Adicionar ruído (Definição 4) à cada um dos indivíduos “filhos” gerados via *crossover* de acordo com a Equação (3.14).
 - 9: **para** para cada indivíduo “filho” gerado via *crossover* **faça**
 - 10: Aplicar somente um passo do MG em cada um dos indivíduos “filhos” gerados via *crossover* exatamente como descrito no Algoritmo 3.
 - 11: **fim para**
 - 12: Atribuir o *fitness* à cada um dos indivíduos “filhos” gerados via *crossover* (após serem submetidos ao MG exatamente como descrito no Algoritmo 3) de acordo com a Equação (3.11), submetendo, antes, cada um dos indivíduos “filhos” à Equação (3.10) para “desvetorizá-los”.
 - 13: Inserir os indivíduos “filhos” na população no lugar de indivíduos, teoricamente, mais aptos. Entre cada geração 20% da população será substituída.
 - 14: **fim enquanto**
-

6.1.2 Algoritmo Genético Guloso

O Algoritmo Genético Guloso (AGGL) também funciona utilizando o AG descrito no Capítulo 3 e o MG, descrito no Capítulo 4, com leve modificação na escolha de indivíduos para *crossover* no AG. Este método, ao invés de substituir apenas uma parcela de sua população após cada geração, substitui toda a população por indivíduos, em tese, mais aptos - em tese pela mesma justificativa apresentada no final do Capítulo 3.

No AGGL, há somente uma “mãe” (o indivíduo com menor custo na população) e todos os demais são “pais” que se cruzam de com as mesmas regras do AG, ou seja, utilizando o conceito de *combinação convexa*, apresentado na Definição 2, Equação (3.13), sendo que, à cada um desses indivíduos, é atribuído um *fitness*, de acordo com a Equação (3.11). Toda população “velha” morre, os “pais” substituídos através do *crossover*, e a “mãe” através da Equação (4.13); em outras palavras, os “pais” são substituídos através do AG, e a “mãe” substituída através do MG. Com relação à Equação (3.13), a diferença dá-se na cardinalidade dos conjuntos $\mathcal{S}$, $\mathcal{M}$ e $\mathcal{P}$; o primeiro é formado por todos os indivíduos na população, o segundo contém somente o indivíduo com melhor custo na população, e o terceiro contém os indivíduos restantes. Portanto, cada novo indivíduo também é dado por

$$g_f^{ell} = \lambda_f g_m^{ell} + (1 - \lambda_f) g_p^{ell}, \quad (6.1)$$

de forma que g_f^{ell} é o i -ésimo indivíduo da ell -ésima geração, g_m^{ell} é a melhor solução e g_p^{ell} são os indivíduos “pais”, ambos na ell -ésima geração, respectivamente.

Também adicionaremos ruído gaussiano à cada “filho” gerado via *crossover* visando adicionar variabilidade genética à população de acordo com a equação

$$g_f^{ell} = g_f^{ell} + N(\mu, \sigma^2), \quad (6.2)$$

e, embora não seja comum a mutar todo indivíduo “filho”, optamos por efetuá-la dada a justificativa apresentada na Observação 4, Capítulo 3.

Sucintamente, o AGGL inicializa sua execução assim como o AG, apresentado no Capítulo 3, ou seja, com 50 indivíduos na população inicial gerados através da Equação (3.6) seguindo a definição de *combinação convexa* (Definição 2). A partir daí seleciona o indivíduo com melhor *fitness* na população, o indivíduo “mãe” (todos os demais consideramos como indivíduos “pais”); o *crossover* é feito de acordo com a Equação (3.13), gerando indivíduos “filhos”; em cada um desses “filhos” adicionamos ruído (Definição 4, Equação (3.14)) visando adicionar variabilidade genética à população; o *fitness*, dado pela Equação (3.11), é atribuído à cada “filho” que são inseridos na população, substituindo a maioria da população vigente; ao indivíduo “mãe”, aplicamos o MG, assim como apresentado no Capítulo 4, transformado em uma “função” neste método, que aplica somente um passo ao indivíduo “mãe” em cada geração/iteração do algoritmo; finalmente, o indivíduo “mãe”, após ser submetido ao MG, é inserido na população.

A motivação para tal método dá-se pelo custo computacional do MG - caro para ser usado em todo indivíduo na população. Com essa estratégia, utilizamos o MG somente em um indivíduo (a “mãe”, o melhor indivíduo na população).

Observação 8. A cada iteração do método, g_m (o indivíduo “mãe”) pode mudar, uma vez que verificamos os valores de custo para toda a nova população, adotamos o de menor valor como sendo a nova “mãe”.

Observação 9. Note que o *o método da roleta* não é utilizado neste método, uma vez que nenhum critério de seleção de indivíduos é necessário. O melhor indivíduo na população, o indivíduo “mãe”, é selecionado com base em seu *fitness*, o melhor dentre todos os indivíduos, e não probabilisticamente.

Observação 10. Neste método, e assim como em todos os métodos apresentados até o momento, cada indivíduo que compõe a população também é uma vetorização das (Definição 1) matrizes A_c , B_c , C_c e D_c que atuam no controlador independente de θ , apresentando em (2.17); note, também, que como cada um desses indivíduos são gerados a partir da *combinação convexa* das matrizes do controlador em (2.4), ou seja, o controlador dependente do modo θ de operação, o operador φ , apresentado em (3.2), também é necessário.

Observação 11. Assim como ressaltando para o AGG, note que como o AGGL é um híbrido entre o AG apresentado no Capítulo 3 e o MG apresentado no Capítulo 4, e, conseqüentemente, seguem o descrito em tais capítulos, as questões concernentes a ambos os algoritmos e respectivas justificativas se fazem válidas.

Algoritmo 5 – Pseudocódigo demonstrando o funcionamento do AGGL descrito nessa seção

- 1: Submeter o controlador em (2.4), o controlador dependente do modo θ de operação, à Equação (3.2).
 - 2: Gerar indivíduos para a população inicial de acordo com a Equação (3.6).
 - 3: Adicionar ruído (Definição 4) à cada um dos 50 elementos da população de acordo com a Equação (3.8).
 - 4: Atribuir o *fitness* a cada um dos indivíduos da população de acordo com a Equação (3.11), submetendo, antes, cada indivíduo à Equação (3.10) para “desvetorizar” cada um dos indivíduos na população.
 - 5: **enquanto** condição de parada não for satisfeita - neste trabalho, tempo máximo de execução **faça**
 - 6: Selecionar o indivíduo “mãe”, aquele com o melhor *fitness* dentro todos os indivíduos da população.
 - 7: Cruzar o indivíduo “mãe” com os demais indivíduos da população (indivíduos “pais”) de acordo com a Equação (6.1).
 - 8: Adicionar ruído gaussiano à cada um dos indivíduos “filhos” gerados via *crossover* de acordo com a Equação (6.2).
 - 9: Atribuir o *fitness* à cada um dos indivíduos “filhos” gerados via *crossover* de acordo com a Equação (3.11), submetendo, antes, cada um dos indivíduos “filhos” à Equação (3.10) para “desvetorizá-los”.
 - 10: Inserir os indivíduos “filhos” na população no lugar de indivíduos, teoricamente, mais aptos.
 - 11: Aplicar somente um passo do MG exatamente como descrito no Algoritmo 3 no indivíduo “mãe”.
 - 12: **fim enquanto**
-

6.2 Algoritmo Genético Evolutivo

O Algoritmo Genético Evolutivo (AGE) que será apresentado neste capítulo possui as mesmas etapas que compõem o AG apresentado no Capítulo 3, Seção 3.1, ou seja, as fases de *inicialização, avaliação, seleção, cruzamento (crossover), mutação, atualização e finalização* devem ser cumpridas. Entretanto, alguns conceitos foram adaptados para atender diferentes requisitos.

O AGE baseia-se na criação de um parâmetro extra, artificialmente. É um parâmetro $r(t)$ de observação indireta da variável de salto $\theta(t)$, seguindo a mesma ideia em (SILVA, 2012), em que a cada instante t , a variável $r(t)$ traz informação sobre a variável $\theta(t)$.

6.2.1 Problema de controle com observação indireta

Considere o espaço de estados $\mathfrak{X}_t = \{\theta_0, r_0, \dots, \theta_t, r_t\}$ e a variável $r(t)$, a qual é observada ao longo do tempo t e possui a seguinte distribuição condicional:

$$\begin{aligned} (1) \quad & P(r_t = i | \theta_t = i, \mathfrak{X}_{t-1}) = P(r_t = i | \theta_t = i) = c, \\ (2) \quad & P(r_t \neq i | \theta_t = i, \mathfrak{X}_{t-1}) = P(r_t \neq i | \theta_t = i) = \left(\frac{1-c}{N-1}\right). \end{aligned} \quad (6.3)$$

Voltemos ao exemplo do satélite: caso o objeto de controle seja sua velocidade, e informações sobre esse dado sejam obtidas através de sensores, o parâmetro “ c ” corresponderá a certeza com relação à informação provida desses sensores.

Como a observação de θ ocorre indiretamente através do parâmetro r , calculamos o CMLP (Equação (2.3)) considerando uma cadeia de Markov aumentada, tendo $(\theta(t), r(t))$ como estado e *probabilidade de transição*

$$\begin{aligned} & P(\theta(t+1) = i, r(t+1) = j | \theta(t) = m, r(t) = n) \\ &= P(r(t+1) = j | \theta(t+1) = i, \theta(t) = m, r(t) = n) \underbrace{P(\theta(t+1) = i | \theta(t) = m, r(t) = n)}_{p_{mi}} \\ &= P(r(t+1) = j | \theta(t+1) = i) p_{mi} \\ &= \begin{cases} p_{mi} \times c, & \text{se } i = j, \\ p_{mi} \times \left(\frac{1-c}{N-1}\right), & \text{se } i \neq j, \end{cases} \end{aligned} \quad (6.4)$$

e *probabilidade de distribuição inicial*

$$P(\theta(0) = i, r(0) = j) = \underbrace{P(\theta(0) = i | r(0) = j)}_{\hat{p}} \underbrace{P(r(0) = j)}_{\pi} = \hat{p}(i, j) \pi_i(0). \quad (6.5)$$

Exemplo 2. Como cada novo estado de Markov será formado pelo par $(\theta(t), r(t))$, a dimensão de $\mathbb{P}$ se eleva para N^2 e isso leva a necessidade de redefinirmos a *matriz de probabilidade de transição* e a *probabilidade de distribuição inicial*.

Considere uma cadeia de Markov com dois estados com a seguinte *matriz de probabilidade de transição*:

$$\mathbb{P} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}. \quad (6.6)$$

Denominaremos $s(t)$ cada novo estado de Markov formado pelo par $(\theta(t), r(t))$, portanto:

Tabela 17 – Exemplo de redefinição de uma cadeia de Markov com 2 estados

$s(t)$	$\theta(t)$	$r(t)$
1	1	1
2	1	2
3	2	1
4	2	2

Utilizando a *matriz de probabilidade de transição* “original” $\mathbb{P}$, os novos estados de Markov $s(t)$ e as Equações (6.3), redefinimos $\mathbb{P}$, que denominamos $\hat{\mathbb{P}}$, como segue:

$$\hat{\mathbb{P}} = \begin{matrix} & \begin{matrix} (\theta_t, r_t) & (1, 1) & (1, 2) & (2, 1) & (2, 2) \end{matrix} \\ \begin{matrix} (1, 1) \\ (1, 2) \\ (2, 1) \\ (2, 2) \end{matrix} & \begin{bmatrix} \frac{1}{2} \times c & \frac{1}{2} \times (\frac{1-c}{N-1}) & \frac{1}{2} \times (\frac{1-c}{N-1}) & \frac{1}{2} \times c \\ \frac{1}{2} \times c & \frac{1}{2} \times (\frac{1-c}{N-1}) & \frac{1}{2} \times (\frac{1-2}{N-1}) & \frac{1}{2} \times c \\ \frac{1}{3} \times c & \frac{1}{3} \times (\frac{1-c}{N-1}) & \frac{2}{3} \times (\frac{1-c}{N-1}) & \frac{2}{3} \times c \\ \frac{1}{3} \times c & \frac{1}{3} \times (\frac{1-c}{N-1}) & \frac{2}{3} \times (\frac{1-c}{N-1}) & \frac{2}{3} \times c \end{bmatrix} \end{matrix}. \quad (6.7)$$

Agora considere a seguinte *probabilidade de distribuição inicial*:

$$\pi(0) = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \end{bmatrix}. \quad (6.8)$$

Utilizando $\pi(0)$, os novos estados de Markov $s(t)$ e as Equações (6.3), redefinimos $\pi(0)$ como segue:

$$\begin{aligned} \hat{p}(i, j) \times \pi_i(0) &= \hat{p}(1, 1) \times \pi_1(0) = c \times \frac{1}{3}, \\ \hat{p}(i, j) \times \pi_i(0) &= \hat{p}(1, 2) \times \pi_1(0) = \left(\frac{1-c}{N-1}\right) \times \frac{1}{3}, \\ \hat{p}(i, j) \times \pi_i(0) &= \hat{p}(2, 1) \times \pi_2(0) = \left(\frac{1-C}{N-1}\right) \times \frac{2}{3}, \\ \hat{p}(i, j) \times \pi_i(0) &= \hat{p}(2, 2) \times \pi_2(0) = c \times \frac{2}{3}. \end{aligned} \quad (6.9)$$

Outra variável que deve ser redefinida é a variável ρ , que representa a *distribuição Cesàro limite* (2.10). Entretanto, nenhuma alteração deve ser feita na maneira como é calculada, uma vez que necessita apenas de $\mathbb{P}$ e $\pi(0)$ para seu cálculo. Portanto, redefinindo $\mathbb{P}$ e $\pi(0)$, consequentemente redefinimos ρ .

Observação 12. Podemos interpretar os valores de “ c ” no intervalo $[1, \frac{1}{N}]$ como estágios com diferentes *níveis de observação* para θ , com $\frac{1}{N}$ correspondente ao cenário de total incerteza, ou seja, o cenário sem observação de θ .

Portanto, como estamos lidando com uma cadeia de Markov aumentada, reescrevemos o sistema apresentado em (2.1) como

$$\hat{\mathbf{G}} = \begin{cases} x(t+1) = A(s(t))x(t) + B(s(t))u(t) + B_w(s(t))w(t), \\ y(t) = C_y(s(t))x(t) + D_w(s(t))w(t), \\ z(t) = C_z(s(t))x(t) + D_u(s(t))u(t), \quad t \geq 0, \end{cases} \quad (6.10)$$

e seu controlador em (2.4) como

$$\hat{\mathbf{R}}_0 = \begin{cases} x_c(t+1) = A_c(s(t))x_c(t) + B_c(s(t))y(t), \\ u(t) = C_c(s(t))x_c(t) + D_c(s(t))y(t). \end{cases} \quad (6.11)$$

Note que nesse “novo sistema”, $s(t)$ é definido pelos parâmetros $\theta(t)$ e $r(t)$.

Mais resultados sobre esse tipo de abordagem, onde o estado θ do sistema é observado de forma indireta, podem ser encontrados em (COSTA; FRAGOSO; TODOROV, 2015), (OLIVEIRA; COSTA; DAAFOUZ, 2018), (OLIVEIRA; COSTA, 2018), (OLIVEIRA; COSTA, 2017), entre outros.

6.2.2 Algoritmo Genético Evolutivo para o problema proposto

A proposta com relação ao AGE é criar “níveis de transição”, ou novos “ambientes”, no contexto de AGs, entre o problema original, apresentado na Seção 2.1, e o problema deste trabalho, apresentado na Seção 2.2. Tais níveis são dados pelo parâmetro “ c ”, que corresponde a certeza com relação à observação da variável θ ; se imaginarmos θ como um sensor em determinado sistema, o parâmetro “ c ” representaria a porcentagem de certeza com relação à informação provida desse sensor. Teoricamente, essa abordagem garantiria vantagens em relação ao AG tradicional descrito no Capítulo 3 por proporcionar a população, composta pelo conjunto de possíveis soluções, a oportunidade de adaptar-se a cada novo estágio, a cada novo “ambiente”, até atingirmos o cenário de não-observação de θ .

Inicializaremos o AGE com “ $c = 1$ ”, que corresponde a variáveis de salto observáveis, ou seja, θ observável, e aos poucos diminuiremos tal valor (e com isso a certeza de observação) até atingirmos total incerteza com relação à variável θ , ou seja, o cenário sem observação da variável de salto θ .

A motivação por tal abordagem baseia-se na observação de algumas espécies de animais, que migram de um ambiente para outro em busca de condições favoráveis, tais como temperatura, alimentos, entre outros. Algumas espécies de pássaros possuem rotas migratórias

intercontinentais (NEWTON, 2007), e claro que tais migrações não acontecem instantaneamente, ou seja, espécies não desaparecem de um lugar e surgem em outro “do dia para a noite”, mas gradativamente, o que proporciona para essas espécies em migração a oportunidade de adaptação em cada um dos ambientes por onde passam, com condições, climas e circunstâncias distintas.

A variável de decisão do AGE também será montada utilizando o conceito de vetorização apresentado na Seção 3.2, Definição 1. Entretanto, com relação à população inicial, as possíveis soluções não serão geradas através da *combinação convexa* (Definição 2) entre os estados do controlador com θ observável, apresentado em (2.4), uma vez que inicializaremos a execução do algoritmo desenvolvido a partir deste método com “ $c = 1$ ”, que corresponde a certeza de observação de θ , mas através da vetorização das matrizes A_c , B_c , C_c e D_c atuantes nesse controlador com a adição de ruído gaussiano (Definição 4) à cada indivíduo. Portanto, nesse método, não necessitaremos do operador ϕ , responsável por vetorizar as matrizes do controlador $\mathbf{R}_o$ em (2.4), e podemos definir os todos os indivíduos da seguinte forma:

$$\Gamma(g_h) = \begin{bmatrix} \text{vec}(A_c^1) \\ \text{vec}(B_c^1) \\ \text{vec}(C_c^1) \\ \text{vec}(D_c^1) \\ \vdots \\ \text{vec}(A_c^N) \\ \text{vec}(B_c^N) \\ \text{vec}(C_c^N) \\ \text{vec}(D_c^N) \end{bmatrix} + N(\mu, \sigma^2), \quad (6.12)$$

para $g \in \mathbb{R}^{nc+ny+nc+ny}$, h o número total de indivíduos na população, N o número de estados de Markov e com o operador Γ como responsável por efetuar a vetorização das matrizes A_c , B_c , C_c e D_c atuantes no controlador com θ observável, apresentado em (2.4). Para o AGE, esse modo de representação é essencial, uma vez que o sistema e seu controlador, em um primeiro momento, observam θ , mas existe a probabilidade “ c ” de que o valor provindo de θ esteja incorreto.

Note que a adição de ruído é necessária para garantir variabilidade genética à população, caso contrário ela seria composta por indivíduos idênticos.

Entretanto, e semelhante aos outros métodos apresentados anteriormente, a função de avaliação J , dada pela Equação (2.11) e apresentada na Seção 2.1.1, não faz uso das matrizes A_c , B_c , C_c e D_c na forma vetorizada, e portanto considere o operador Γ

$$\Gamma^{-1}(g_h) = (A_c^N, B_c^N, C_c^N, D_c^N), \quad (6.13)$$

onde N é o número de estados de Markov, que “desvetoriza” os indivíduos g_h e reagrupa o conjunto de matrizes $A_c, \dots, D_c$ para cada estado de Markov.

Cada indivíduo na população também será avaliado com base na função de custo apresentada na 2.1.1, Equação (2.11), e, portanto, $J(\Gamma^{-1}(g_1)), J(\Gamma^{-1}(g_2)), \dots, J(\Gamma^{-1}(g_h))$. Assim como

no AG tradicional descrito no Capítulo 3, os melhores indivíduos dentre todos os indivíduos na população devem possuir maior probabilidade de escolha, e assim sendo, a aptidão de cada indivíduo é obtida através da Equação

$$f(g_h) = 1/J(\Gamma^{-1}(g_h)). \quad (6.14)$$

Neste algoritmo, não é necessário retirarmos o índice i do controlador nas estruturas A_{cl} , B_{cl} , C_{cl} e D_{cl} apresentada na Seção 2.2 para comportarem somente um conjunto de matrizes A_c , B_c , C_c e D_c , dado o modo com que o AGE trabalha e a consequente a maneira como cada indivíduo é composto. Portanto, usaremos tais estruturas da maneira como foram introduzidas na apresentada na Seção 2.1.1, Eqs (2.12)-(2.15).

A próxima etapa é a *seleção de indivíduos para crossover* e também utilizaremos o *método da roleta (Roulette Wheel Selection)* pelos mesmos motivos descritos no Capítulo 3; consequentemente, a probabilidade de escolha de dá-se pela Equação

$$p_i = \frac{f(g_i)}{\sum_{j=1}^h f(g_j)}, \quad (6.15)$$

em que h representa o número total de indivíduos na população (uma representação do *método da roleta (Roulette Wheel Selection)* pode ser visto na Figura 3, disponível no Capítulo 3). Duas diretrizes foram estabelecidas na etapa de *seleção*: a primeira é que determinado indivíduo deve ter um parceiro(a) de reprodução que difere de si mesmo; a segunda é que se determinado indivíduo for selecionado para o crossover, diminuiremos sua aptidão para dificultar que esse mesmo indivíduo seja selecionado novamente na mesma geração. A justificativa para tais diretrizes são as mesmas apresentadas no Capítulo 4.

Ainda com relação à seleção de indivíduos para *crossover*, também é importante dizer que as mesmas questões concernentes ao AG descritas Seção 3.2 fazem-se presentes no AGE, como a inexistência de garantias que os melhores indivíduos serão selecionados, indivíduos devem possuir um parceiro(a) de reprodução que difere-se de si, a quantidade de indivíduos que serão selecionados em cada geração deve ser proporcional a quantidade de indivíduos que compõem a população, etc; e claro, como ambos os algoritmos assemelham-se, todas as justificativas apresentadas na Seção 3.2 para o AG também são válidas para o AGE.

Com os indivíduos selecionados, também efetuaremos o *crossover* da mesma maneira descrita da Seção 3.2, ou seja, de acordo com a Equação

$$g_f^{ell} = \lambda_f g_m^{ell} + (1 - \lambda_f) g_p^{ell}, \quad (6.16)$$

em que chamaremos $\mathcal{S}$ os indivíduos selecionados para o *crossover*, que forçamos ter cardinalidade par, tendo em vista a Observação 2; chamaremos de $\mathcal{M}$ e $\mathcal{P}$ uma partição de $\mathcal{S}$, ambos com a mesma cardinalidade, que servem para designar os conjuntos de “pais” e “mães”. O índice “ ell ” para marcar a geração, de forma que g_f^{ell} é o i -ésimo indivíduo da ell -ésima geração. Também

vale ressaltar que, assim como no *crossover* do AG apresentado no Capítulo 3, os valores de λ são tais que $\sum_{i=1}^N = 1, 2$, o que faz com que os “filhos” gerados extrapolem o segmento de reta, visando, novamente, adicionar variabilidade genética à população.

A cada “filho” será atribuído ruído gaussiano (*mutação*), ou seja,

$$g_f^{ell} = g_f^{ell} + N(\mu, \sigma^2), \quad (6.17)$$

e submetido à Equação (6.13), o que, por sua vez, à Equação (2.11) e a Equação (6.14), em que um *fitness* lhe será atribuído. Geralmente a *mutação* é efetuada de acordo com determinada probabilidade, geralmente, baixa (MITCHELL, 1996). Entretanto, optamos por aplicá-la em todo indivíduo “filho” resultante via *crossover* pelos mesmos motivos descritos na Observação 4, apresentada no Capítulo 3.

Os novos indivíduos serão inseridos na população substituindo indivíduos menos aptos. Analogamente ao AG “tradicional”, também consideraremos a população do AGE fixa, ou seja, a quantidade de indivíduos que será inserida da população é proporcional a quantidade de indivíduos que deixarão de existir. Neste trabalho, entre uma geração e outra, 20% dos indivíduos deixarão de existir para darem lugar a novos, e que os indivíduos mais aptos estão sendo mantidos entre uma geração e outra. A motivação para tal porcentagem é a mesma apresentada no Capítulo 3, ou seja, a heterogeneidade da população. Se a quantidade de indivíduos que será substituída entre gerações for baixa, perde-se em recursos computacionais pois o algoritmo pode demorar para encontrar uma solução ótima (de acordo com algum critério); caso a quantidade de indivíduos substituída entre gerações seja muito alta, corre-se o risco de perder variabilidade genética a população tornar-se homogênea logo nas primeiras gerações/iterações do algoritmo, uma vez que indivíduos distintos e/ou com *fitness* ótimos, podem ser substituídos entre gerações devido ao alto número de “filhos” gerados via *crossover* em uma mesma geração. Portanto, optamos por substituir 20% da população entre as gerações, um valor que consideramos não muito baixo de modo que impeça que algoritmo demore para encontrar soluções, e nem muito alto para que a população torne-se homogênea logo nas primeiras iterações do algoritmo minimizando as chances que soluções ótimas (novamente, de acordo com algum critério) seja encontrada.

Observação 13. Note que, embora estejamos satisfazendo todas as fases de um AG “clássico”, ou seja, *inicialização, avaliação, seleção, cruzamento (crossover), mutação, atualização e finalização*, a maneira como tais fases foram cumpridas foram adaptadas para atender a proposta de funcionamento do AGE. Apesar de tais adaptações, ainda estamos lidando com um AG, e, portanto, todas as questões e justificativas levantadas no Capítulo 3 com relação a heterogeneidade da população, quantidade de indivíduos selecionados para *crossover*, adição de ruído gaussiano atuando com elemento mutagênico em cada “filho” resultante via *crossover*, quantidade de indivíduos substituídos entre gerações, etc. também são válidas para o AGE.

Note, também, que todas as fases do AGE são influenciadas pelo valor que o parâmetro “*c*” pode assumir, uma vez que influencia na *matriz de probabilidade de transição*, na *probabilidade*

de distribuição inicial e na distribuição Cesàro limite.

A proposta em relação a utilização do AGE é fazer com que a certeza com relação à observação da variável θ diminua aos poucos, inicializando em “ $c = 1$ ” (certeza de observação) e diminuir gradativamente até atingir “ $c = \frac{1}{N}$ ” (assim a probabilidade $P(\theta = i | \hat{\theta}) = \frac{1}{N}$ para qualquer i , portanto $\hat{\theta}$ não traz qualquer informação sobre θ). Mas isso suscita a pergunta: de que maneira decrementar “ c ” a partir da certeza até atingirmos total incerteza, ou seja, o cenário sem observação da variável θ ? Para solucionar esse dilema consideramos o tempo total de execução do algoritmo e a quantidade de estados da instância analisada.

Neste trabalho, consideraremos uma malha com diferentes valores para “ c ”, e consequentemente para a variável θ ; esses valores devem estar contidos entre 1 e $\frac{1}{N}$ e assim sendo, cada elemento da malha será dado por

$$C_{s+1} = C_s - \left(\frac{1 - \frac{1}{N}}{E} \right), \quad s = 1, 2, \dots, E, \quad (6.18)$$

em que o índice s indica o elemento na malha, N é o número de de Markov e E a quantidade máxima de elementos nessa malha, ou seja, níveis de observação de θ . Abaixo, temos o trecho de código em MATLAB[®] para geração de todos os possíveis valores de observação para θ :

Código-fonte 1 – Criação de todos os possíveis valores para a parâmetro “ c ”, utilizando a linguagem de programação MATLAB[®]

```
1: vecC = 1: -(1-1/N)/E : 1/N.
```

A variável E representa quantidade de elementos na malha de “ c ”, que é definida através do tempo máximo de execução atribuído ao método e o tempo máximo atribuído a cada elemento da malha de “ c ”; $vecC$ é a malha em si, a variável que armazena todos os possíveis valores de “ c ” para cada instância (todos os possíveis valores que “ c ” poderá assumir durante a execução do algoritmo), visto que, embora a quantidade de elementos seja a mesma, o valor de cada elemento depende da quantidade de estados que cada instância analisada possui.

A transição entre os valores da malha de “ c ” acontecerá seguindo duas restrições: a primeira é que a população melhore após o “treino” em determinado “ambiente”. É intuitivo pensar que a população sempre apresentará melhora, contudo, isso nem sempre acontece dada a adição de ruído gaussiano em cada indivíduo resultante via *crossover*; a segunda é que deve existir pelo menos um indivíduo com o custo menor do que 10^{100} , uma vez que, lembrando, instâncias cujo melhor custo encontrado seja maior do que 10^{100} são contabilizadas como sem solução. Caso uma das condições não seja satisfeita e enquanto houver tempo de execução disponível, a transição entre os valores de “ c ” não acontece.

Como não existem garantias de que as restrições descritas no parágrafo anterior serão satisfeitas, implementamos o algoritmo de modo a saltar para o último elemento da malha de “ c ”

(cenário sem observação) nos últimos momentos de execução, independentemente do valor de “ c ” na etapa anterior na execução do algoritmo. Exemplo: se o tempo máximo disponibilizado ao método para execução é de 10 segundos dividido em estágios de 2 segundos, os últimos 2 segundos serão disponibilizados para o método executar no cenário sem observação, independentemente do valor de “ c ” na etapa anterior na execução do algoritmo.

A maneira como o AGE opera exige que tenhamos um conjunto de matrizes $A_c, \dots, D_c$ para cada estado de Markov. Como temos como objetivo encontrar somente um conjunto dessas matrizes para atuar no controlador (2.17) independente de θ , após a execução do algoritmo, qualquer conjunto de matrizes $A_c^1, \dots, D_c^1, \dots, A_c^N, \dots, D_c^N$ satisfaz o nosso problema, apresentado na Seção 2.2. De modo prático, a cada instante de tempo t , um conjunto de matrizes é escolhido aleatoriamente dentre um conjunto $A_c^1, \dots, D_c^1, \dots, A_c^N, \dots, D_c^N$ independentemente do modo de operação do sistema no instante t ; como a probabilidade de escolha é equiprovável, temos o cenário “ $c = \frac{1}{N}$ ”, ou seja, o cenário sem observação de variável θ , satisfazendo, assim, o nosso problema, uma vez que podemos escolher qualquer um dos conjuntos para atuar no nosso controlador em (2.17).

Algoritmo 6 – Pseudocódigo demonstrando o funcionamento do AGE descrito nesse capítulo

- 1: Gerar indivíduos para a população inicial de acordo com a Equação (6.12).
 - 2: Atribuir o *fitness* a cada um dos indivíduos da população inicial de acordo com a Equação (6.14), submetendo, antes, cada indivíduo à Equação (6.13) para “desvetorizar” cada um dos indivíduos na população.
 - 3: Definir a malha com todos os elementos de “*c*” de acordo com a Equação (6.18).
 - 4: **enquanto** condição de parada não for satisfeita - neste trabalho, tempo máximo de execução **faça**
 - 5: Alterar *matriz de probabilidade de transição*, a *probabilidade de distribuição inicial* e a *distribuição Cesàro limite* de acordo com o Exemplo 2, dado o valor do parâmetro “*c*”.
 - 6: **enquanto** condição de parada não for satisfeita - neste trabalho, tempo máximo de execução atribuído à cada estágio para o “treino” da população em determinado “ambiente” **faça**
 - 7: Selecionar 20% dos indivíduos da população, divididos em indivíduos “pais” e indivíduos “mães”, para reprodução de acordo com a Equação (6.15).
 - 8: Cruzar os indivíduos “pais” e os indivíduos “mães” (*crossover*), gerando indivíduos “filhos”, de acordo com a Equação (6.16).
 - 9: Adicionar ruído (Definição 4) à cada um dos indivíduos “filhos” gerados via *crossover* de acordo com a Equação (6.17).
 - 10: Atribuir o *fitness* à cada um dos indivíduos “filhos” gerados via *crossover* de acordo com a Equação (6.14), submetendo, antes, cada um dos indivíduos “filhos” à Equação (6.13) para “desvetorizá-los”.
 - 11: Inserir os indivíduos “filhos” na população no lugar de indivíduos, teoricamente, mais aptos. Entre cada geração 20% da população será substituída.
 - 12: **fim enquanto**
 - 13: **se** a população tenha melhorado após o “treino” em determinado “ambiente” E pelo menos um indivíduo possua custo menor do que 10^{100} **então**
 - 14: Transitar entre os valores da malha de “*c*”, diminuindo tal valor.
 - 15: Alterar *matriz de probabilidade de transição*, a *probabilidade de distribuição inicial* e a *distribuição Cesàro limite* de acordo com o Exemplo 2, dado o novo valor do parâmetro “*c*”.
 - 16: **fim se**
 - 17: **fim enquanto**
-

ANÁLISE DE RESULTADOS II

A análise efetuada a partir dos dados obtidos através da comparação entre o AG e o MG apresentados no Capítulo 5 levam a crer que melhores resultados podem ser obtidos através de métodos derivados do AG e do MG.

7.1 Algoritmo Genético versus Algoritmo Genético-Gradiente

O primeiro método proposto foi o algoritmo AGG, união entre o AG e o MG, como já descrito no Capítulo 6, Seção 6.1.1 uma vez que os dados e análises apresentados no Capítulo 5 mostraram que com tempo suficiente disponibilizado aos métodos, embora o AG seja capaz de encontrar mais soluções, muitas soluções encontradas pelo MG sobressaem-se devido ao modo como atua no espaço de busca.

Devido aos resultados obtidos entre o AG e o MG, é intuitivo pensar que a hibridização do AGG proposta seria capaz de obter melhores soluções.

Com o intuito de testar esta hipótese, antes de efetuarmos comparações seguindo as restrições temporais apresentadas no início do Capítulo 5, optamos por efetuar uma mudança nos algoritmos de modo a atrelar a condição de parada a um número máximo (10) de iterações/-gerações. Esta alteração é suficiente para que a população inicial sofra alterações e resultados comecem a surgir de modo a comprovar ou negar a hipótese; além disto, evita que tempo seja desperdiçado em execuções desnecessárias buscando a comprovação de uma ideia que pode mostrar-se falsa. Caso nossa hipótese estiver correta, quando compararmos o híbrido AGG diretamente com o AG, esse irá destacar-se.

É importante ressaltar que ambos os métodos assemelham-se em suas diretrizes de inicialização, afinal, ainda são Algoritmos Genéticos e todas as “regras” descritas no Capítulo 3 que os regem fazem-se presentes, ou seja, as fases de *inicialização*, *avaliação*, *seleção*, *cruzamento (crossover)*, *mutação*, *atualização* e *finalização* fazem-se presentes. Por este motivo,

optamos pela comparação direta entre o AG “puro” e seu híbrido e não com o MG como feito no Capítulo 5, Seção 5.3.

As conclusões obtidas com relação ao tamanho da população inicial para ambos os algoritmos mantêm-se as mesmas descritas no Capítulo 5, Seção 5.2.

Devido à grande diferença entre valores providos das instâncias submetidas aos métodos, o gráfico disposto na Figura 21 está na escala logarítmica.

Submetemos cada uma das 1000 instâncias aos métodos, e após 10 iterações/gerações obtivemos os resultados dispostos nas Figuras 21 e 22, e nas Tabelas 18 e 19:

Figura 21 – Custos obtidos com o AG x Custos obtidos com o AGG - 10 gerações

Figura 22 – Estatísticas gerais AG x AGG - 10 gerações

Tabela 18 – Estatísticas gerais AG x AGG - 10 gerações

	%	erro padrão
SOMENTE o AG resolve	4,9%	0,6826
SOMENTE o AGG resolve	3,6%	0,5891
ambos os métodos resolvem	29,8%	1,4464
nenhum dos métodos resolve	61,7%	1,5372

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 19 – Estatísticas específicas AG x AGG - 10 gerações

	%	erro padrão
AG melhor que o AGG	1,1%	0,3298
AGG melhor que o AG	14,7%	1,1198
sem diferença significativa entre os métodos	14%	1,0973

O gráfico da Figura 21 é interpretado de maneira análoga aos gráficos presentes no Capítulo 5, ou seja, o ponto no canto superior direito representa as instâncias em que nenhum dos métodos foi capaz de encontrar solução, correspondente a 61,7% dos casos. Em 4,9% dos casos, correspondentes aos pontos paralelos ao eixo das ordenadas, somente o AG conseguiu encontrar uma solução considerada válida e em 3,6% dos casos, correspondente aos pontos

paralelos ao eixo das abcissas, somente o AGG. Entre as duas faixas que cortam o gráfico estão os pontos em que não existe diferença significativa entre as soluções encontradas pelos métodos, correspondente a 14% dos casos.

A porcentagem de casos em que ambos os métodos encontraram solução corresponde a 29,8% dos casos, e entre esses casos, em 14,7% o AGG obteve melhores resultados do que o AG, confirmando o hipótese de que o método híbrido sobressai-se comparado com o AG “puro”. No gráfico da Figura 21, esse resultado pode ser observado pelos pontos acima das duas faixas que cortam o gráfico. A porcentagem de casos em que somente um dos métodos encontra solução pode ser explicada pela aleatoriedade na inicialização dos métodos e na adição de ruído gaussiano em cada um dos indivíduos da prole entre cada geração, embora ela não seja significativa, já que ambos os métodos assemelham-se. Não somente isso, tal aleatoriedade também é responsável pela porcentagem de casos em que o AG destaca-se em comparação com o AGG (pontos situados abaixo das duas faixas que cortam o gráfico).

Com a hipótese confirmada, podemos comparar o AG e o AGG seguindo as restrições temporais propostas no início do Capítulo 5 e, como confirmamos que com somente 10 gerações o AGG destaca-se, é esperado que essa tendência se mantenha.

Com relação às execuções, optamos por não submeter as instâncias aos métodos inúmeras vezes pois seria redundante, uma vez que cada um dos métodos inicializa com várias possíveis soluções e, segundo testes previamente efetuados durante a implementação dos métodos, o ganho seria irrisório comparado com o tempo de execução necessário. Contudo, mantivemos as mesmas restrições temporais já apresentadas.

De maneira análoga aos gráficos dispostos no Capítulo 5, devido à grande diferença entre valores provindos das instâncias submetidas aos métodos, os gráficos dispostos na Figura 23, Figura 25 e na Figura 27 estão na escala logarítmica.

Após 10 segundos de execução comparamos o desempenho de ambos os métodos para uma mesma instância e obtivemos resultados dispostos nas Figuras 23 e 24, e nas Tabelas 20 e 21:

Figura 23 – Custos obtidos com o AG x Custos obtidos com o AGG - 10 segundos

Figura 24 – Estatísticas gerais AG x AGG - 10 segundos

Tabela 20 – Estatísticas gerais AG x AGG - 10 segundos

	%	erro padrão
SOMENTE o AG resolve	3,4%	0,5731
SOMENTE o AGG resolve	7,2%	0,8174
ambos os métodos resolvem	15%	1,1292
nenhum dos métodos resolve	74,4%	1,3801

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 21 – Estatísticas específicas AG x AGG - 10 segundos

	%	erro padrão
AG melhor que o AGG	0,8%	0,2817
AGG melhor que o AG	2,5%	0,4937
sem diferença significativa entre os métodos	11,7%	1,0164

Em 74,4% dos casos, nenhum dos métodos foi capaz de encontrar solução, não somente devido ao pouco disponibilizado aos métodos para avançarem gerações e encontrarem soluções, mas também para cumprirem suas diretrizes de inicialização, já que, lembrando, nesta porcentagem estão inclusas instâncias em que os métodos não encontraram solução e instâncias que excederam seu tempo de execução antes que suas diretrizes de inicialização fossem cumpridas.

Dentre as instâncias em que ambos os métodos foram capazes de encontrar solução, correspondente a 15% dos casos, vemos que, como esperado, o AGG sobressai-se ao AG, com 2,5% contra 0,8%, respectivamente. Contudo, em 11% dos casos não existe diferença significativa entre os resultados encontrados pelos métodos.

Com relação às instâncias em que somente um dos métodos foi capaz de encontrar solução, vemos que o AGG destaca-se com 7,2% contra 3,4% do AG. Contudo, diferentemente da comparação entre AG versus MG em que os dados mostraram que, com tempo suficiente, o AG era capaz de obter resultados que anteriormente somente o MG tinha sido capaz de encontrar, na comparação entre o AG versus AGG, esta estatística não deve ser analisada isoladamente antes de optarmos por um dos métodos, pois ambos assemelham-se em sua inicialização, e devido à aleatoriedade em sua concepção na geração de possíveis soluções e por não conhecermos o espaço de busca, podem existir incontáveis soluções disponíveis e em cada execução, qualquer um dos métodos pode encontrar soluções que anteriormente tinham sido encontradas somente pelo outro; além disso, o pouco tempo disponibilizado aos métodos exerce influência sobre os resultados finais.

Reexecutamos os algoritmos aumentando o tempo máximo de execução para 1 minuto e obtemos os resultados dispostos nas Figuras 25 e 26, e nas Tabelas 22 e 23:

Figura 25 – Custos obtidos com o AG x Custos obtidos com o AGG - 1 minuto

Figura 26 – Estatísticas gerais AG x AGG - 1 minuto

Tabela 22 – Estatísticas gerais AG x AGG - 1 minuto

	%	erro padrão
SOMENTE o AG resolve	4,7%	0,6693
SOMENTE o AGG resolve	4%	0,6197
ambos os métodos resolvem	29,7%	1,4450
nenhum dos métodos resolve	61,6%	1,5380

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 23 – Estatísticas específicas AG x AGG - 1 minuto

	%	erro padrão
AG melhor que o AGG	2,4%	0,4840
AGG melhor que o AG	12,2%	1,0350
sem diferença significativa entre os métodos	15,1%	1,1322

Assim como na comparação entre AG e MG, mais tempo disponibilizado aos algoritmos possibilitou que os métodos encontrassem soluções para um maior número de instâncias, diminuindo o número de instâncias sem solução de 74,4% para 61,6%.

Ainda com relação as instâncias em que ambos os métodos encontraram solução, notamos que a superioridade do AGG intensifica-se e aumenta de 2,5% para 12,2% comparado com o teste anterior. Embora a superioridade do AG também tenha aumentado de 0,8% para 2,4%, a superioridade do AGG é muito mais significativa. Novamente, notamos a influência da aleatoriedade atuante no sistema, e mesmo um dos métodos possuindo uma característica que concede vantagem em relação ao outro, existirão situações que o método considerado “inferior” será capaz de encontrar melhores soluções, mesmo que ao longo prazo a superioridade de um se intensifique em relação ao outro.

Com relação às instâncias em que somente um dos métodos foi capaz de encontrar solução, notamos uma mudança em relação ao teste anterior, em que o AGG destacou-se. Após 1 minuto de execução, notamos que o AG sobressai-se ao AGG com 4,7% contra 4%, respectivamente. Embora a diferença seja pequena e não possa ser analisada isoladamente na escolha por um dos métodos, ela é um indicativo por qual deles optar, caso um usuário hipotético que deseja encontrar o controle para determinado SLSM também hipotético priorize a quantidade de soluções ao invés da qualidade.

Finalmente, visando minimizar incertezas, reexecutamos os algoritmos aumentando o tempo máximo de execução para 10 minutos e obtivemos resultados dispostos nas Figuras 27 e 28, e nas Tabelas 24 e 25:

Figura 27 – Custos obtidos com o AG x Custos obtidos com o AGG - 10 minutos

Figura 28 – Estatísticas gerais AG x AGG - 10 minutos

Tabela 24 – Estatísticas gerais AG x AGG - 10 minutos

	%	erro padrão
SOMENTE o AG resolve	4%	0,6197
SOMENTE o AGG resolve	2,6%	0,5032
ambos os métodos resolvem	31,5%	1,4689
nenhum dos métodos resolve	61,9%	1,5357

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 25 – Estatísticas específicas AG x AGG - 10 minutos

	%	erro padrão
AG melhor que o AGG	2%	0,4427
AGG melhor que o AG	15,8%	1,1534
sem diferença significativa entre os métodos	13,7%	1,0873

Após 10 minutos de execução, embora os dados alterem-se, eles não são significativos ou proporcionais ao tempo despendido em nenhum dos quesitos analisados, embora tenha permitido que certas tendências fossem ressaltadas.

A porcentagem de casos em que ambos os métodos encontraram solução aumenta de 29,7% para 31,5%, embora em 13,7% dos casos não exista diferença significativa entre eles. Também verificamos que a tendência do AGG em encontrar melhores soluções se mantém a aumenta de 12,2% para 15,8% com relação ao teste anterior; entretanto, embora a diferença não seja significativa, a superioridade do AG em relação ao AGG diminui de 2,4% para 2%.

Com relação à porcentagem de casos em que nenhum dos métodos foi capaz de encontrar soluções, os dados mantêm-se praticamente os mesmos, embora com leve aumento de 61,6% para 61,9%.

Com relação as instâncias para os quais somente um dos métodos foi capaz de encontrar solução, notamos que as porcentagens decaem com relação ao teste anterior; o AGG diminuiu de 4% para 2,6% e o AG de 4,7% para 4%. Embora essa queda não seja brusca e explicada pela aleatoriedade presente em ambos os métodos, confirmamos que, com tempo suficiente, o AG encontra maior quantidade de soluções únicas, embora as soluções provindas do AGG sejam melhores; essa estatística é a principal indicadora caso a qualidade das soluções seja mais importante do que a quantidade na escolha por um dos métodos.

7.1.1 Discussões sobre o Algoritmo Genético versus Algoritmo Genético-Gradiente

Após as execuções com diferentes restrições temporais, os dados indicam que 1 minuto é tempo suficiente para os métodos obterem resultados que permitam que um usuário opte

por um método ou por outro, dado que 10 segundos impedem que os métodos cumpram suas diretrizes de inicialização e/ou não avancem gerações suficientes para termos certeza acerca dos resultados; com 10 minutos, os recursos computacionais e tempo necessário são maiores, os resultados pouco alteram-se e não são proporcionais ao tempo gasto, embora sirvam para confirmar tendências.

Os resultados obtidos não deixam claro qual dos dois métodos é o melhor, dado que o AG destaca-se na quantidade de soluções únicas encontradas enquanto o AGG destaca-se pela qualidade de soluções. Uma interpretação é que se estivermos a procura de um método que seja capaz de encontrar soluções para maior quantidade de instâncias, deveremos optar pelo AG, e caso a escolha deva pautar-se em qualidade, o AGG. Como neste trabalho o objetivo é encontrar um controlador (2.17) que tenha o menor custo possível (Equação (2.3)), e os menores custos são encontrados pelo AGG, outra interpretação é que o AGG é o melhor método. Entretanto, é importante dizer que tais resultados podem ter sido influenciados pela complexidade do algoritmo e alguns detalhes serão abordados na Seção 7.3, em conjunto com os resultados obtidos com a comparação entre os AG versus AGGL (Seção 7.2).

Também vale ressaltar que embora mais tempo disponibilizado aos algoritmos elimina o efeito do não cumprimento das diretrizes de inicialização, o efeito da aleatoriedade (ruído) ainda faz-se presente mesmo que em menor proporção.

Assim como salientado na comparação entre o AG versus MG, notamos a saturação de resultados encontrados após 1 minuto de execução dos algoritmos para o AG versus AGG. Hipóteses para tais ocorrências foram levantadas no Capítulo 5, Seção 5.3.1, ou seja, devido a não convexidade da função objetivo (Equação (2.11)) (OLIVEIRA; COSTA, 2015), e/ou ainda, devido a pouca variabilidade genética da população do AG.

E claro, esses resultados são válidos somente para o tipo de SLSM estudados neste trabalho, apresentado em (2.1), e para controladores dinâmicos, apresentado em (2.4), e não existem garantias que os resultados mantêm-se para outros tipos de SLSMs e/ou para controladores estáticos.

7.2 Algoritmo Genético versus Algoritmo Genético Guloso

A escolha pela comparação entre esses dois métodos dá-se pelo fato de ambos assemelharem-se em suas diretrizes de inicialização, tanto na geração de possíveis soluções quanto na quantidade, uma vez que trabalham com várias possíveis soluções de uma só vez, embora diferenciam-se na maneira com que lidam com as possíveis soluções. Note que ambos são, em sua concepção, AGs, a despeito de como cumprem as fases de *inicialização*, *avaliação*, *seleção*, *cruzamento* (*crossover*), *mutação*, *atualização* e *finalização*, concernentes a AGs.

As conclusões obtidas com relação ao tamanho da população inicial mantêm-se as mesmas descritas no Capítulo 5, Seção 5.2, portanto, visando equiparar os métodos, inicializamos o AGGL com a mesma quantidade de possíveis soluções com a qual o AG inicializa.

Devido à grande diferença entre valores providos das instâncias submetidas aos métodos, os gráficos dispostos na Figura 29, Figura 31 e Figura 33 estão na escala logarítmica.

Após 10 segundos de execução comparamos o desempenho de ambos os métodos para uma mesma instância e obtivemos os resultados dispostos nas Figuras 29 e 30 e nas Tabelas 26 e 27:

Figura 29 – Custos obtidos com o AG x Custos obtidos com o AGGL - 10 segundos

Figura 30 – Estatísticas gerais AG x AGGL - 10 segundos

Tabela 26 – Estatísticas gerais AG x AGGL - 10 segundos

	%	erro padrão
SOMENTE o AG resolve	4,3%	0,6415
SOMENTE o AGGL resolve	6%	0,7510
ambos os métodos resolvem	14,1%	1,1005
nenhum dos métodos resolve	75,6%	1,3582

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 27 – Estatísticas específicas AG x AGGL - 10 segundos

	%	erro padrão
AG melhor que o AGGL	0,5%	0,2230
AGGL melhor que o AG	2,4%	0,4840
sem diferença significativa entre os métodos	11,2%	0,9973

Em 75,6% dos casos, nenhum método foi capaz de encontrar solução devido ao pouco tempo disponibilizado aos algoritmos não somente para avançarem gerações/iterações como também para cumprirem suas diretrizes de inicialização.

A porcentagem de instâncias que ambos os métodos encontraram solução corresponde a 14,1% dos casos, contudo, não existe diferença em 11,2% dos casos. A porcentagem em que um método sobressaiu-se ao outro é de 0,5% para o AG e 2,4% para o AGGL.

Com relação às instâncias em que somente um dos métodos foi capaz de encontrar solução, percebemos que o AGGL sobressai-se com 6% contra 4,3% do AG. Contudo, tal resultado é influenciado pelo pouco tempo disponibilizado aos métodos impedindo que as diretrizes de inicialização de ambos os métodos sejam cumpridas, embora estas assemelhem-se.

Reexecutamos os algoritmos aumentando o tempo máximo de execução para 1 minuto e obtivemos resultados dispostos nas Figuras 31 e 32, e nas Tabelas 28 e 29:

Figura 31 – Custos obtidos com o AG x Custos obtidos com o AGGL - 1 minuto

Figura 32 – Estatísticas gerais AG x AGGL - 1 minuto

Tabela 28 – Estatísticas gerais AG x AGGL - 1 minuto

	%	erro padrão
SOMENTE o AG resolve	3%	0,5394
SOMENTE o AGGL resolve	6,6%	0,7851
ambos os métodos resolvem	31,4%	1,4677
nenhum dos métodos resolve	59%	1,5553

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 29 – Estatísticas específicas AG x AGGL - 1 minuto

	%	erro padrão
AG melhor que o AGGL	2,5%	0,4937
AGGL melhor que o AG	13%	1,0635
sem diferença significativa entre os métodos	15,9%	1,1564

Assim como observado nas análises anteriores entre AG versus MG e/ou AG versus AGG, mais tempo disponibilizado aos algoritmos permitiu que ambos encontrassem soluções para um número maior de instâncias, uma vez que a porcentagem de instâncias que não possuem solução diminuiu de 75,6% para 59%.

Com relação à porcentagem de instâncias para os quais ambos os métodos encontraram solução, notamos que o AGGL destaca-se, obtendo melhores soluções em 13% dos casos contra 2,5% do AG, respectivamente. Embora ambos os métodos assemelhem-se em suas diretrizes de inicialização, ambos os métodos diferem-se na maneira que atuam no espaço de busca, o que favorece o AGGL.

Com relação à porcentagem de instâncias para os quais somente um dos métodos encontra solução, notamos que a tendência observada no teste anterior, em que o AGGL destacou-se, se mantém e é levemente ressaltada; embora a porcentagem de instâncias para os quais somente ao AG resolve tenha diminuído de 4,3% para 3%, a porcentagem de instâncias para os quais somente o AGGL resolve aumentou de 6% para 6,6%.

Embora a análise entre o AG versus AGG tenha revelado que 1 minuto é tempo suficiente para os algoritmos encontrarem soluções que permitam a análise e consequente opção por um ou outro método, não podemos afirmar o mesmo entre o AG versus AGGL sem os dados para avaliarmos, mesmo os métodos assemelhando-se em suas diretrizes de inicialização e possuindo características semelhantes. Portanto aumentamos o tempo máximo de execução para 10 minutos e obtivemos resultados dispostos nas Figuras 33 e 34, e nas Tabelas 30 e 31:

Figura 33 – Custos obtidos com o AG x Custos obtidos com o AGGL - 10 minutos

Figura 34 – Estatísticas gerais AG x AGGL - 1 minuto

Tabela 30 – Estatísticas gerais AG x AGGL - 10 minutos

	%	erro padrão
SOMENTE o AG resolve	1,9%	0,4317
SOMENTE o AGGL resolve	8%	0,8579
ambos os métodos resolvem	33,6%	1,4937
nenhum dos métodos resolve	56,5%	1,5677

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 31 – Estatísticas específicas AG x AGGL - 10 minutos

	%	erro padrão
AG melhor que o AGGL	1,7%	0,4088
AGGL melhor que o AG	16,9%	1,1851
sem diferença significativa entre os métodos	15%	1,1292

Após 10 minutos de execução, os dados alteram-se, embora não significativamente. A porcentagem de casos em que ambos os métodos encontraram solução aumenta de 31,4% para 33,6%, embora em 15% dos casos não exista diferença significativa entre eles.

Mais tempo disponibilizado aos algoritmos também possibilitou que os métodos encontrassem solução para um número maior de instâncias, diminuindo a porcentagem de instâncias

que não possuem solução de 59% para 56,5%, embora essa diferença não seja proporcional à quantidade de tempo disponibilizada aos métodos.

Com relação à quantidade de instâncias em que ambos os métodos encontraram solução, verificamos que a tendência do AGGL em encontrar melhores soluções é intensificada, aumentando de 13% para 16,9%. Analisando esse dado, podemos concluir que o AGGL sai-se melhor que o AG devido a maneira que atua dentro do espaço de busca.

Com relação às instâncias para os quais somente um dos métodos conseguiu encontrar solução, notamos que os dados pouco alteram-se em relação ao teste anterior; a porcentagem que somente o AG resolve decai de 3% para 1,9% e a porcentagem que somente o AGGL resolve aumenta de 6,6% para 8%. Embora a diferença entre os dados obtidos não seja notável quando comparados com o teste anterior, eles confirmam tendências que mostram que a maneira como o AGGL lida com o espaço de busca faz com que ele se destaque, tanto nas soluções encontradas quanto na qualidade de tais soluções.

7.2.1 Discussões sobre o Algoritmo Genético versus Algoritmo Genético Guloso

Assim como salientado na comparação entre o AG versus AGG, mais tempo disponibilizado aos algoritmos permitiu que mais instâncias pudessem ser resolvidas e que os métodos encontrassem melhores soluções. Entretanto, os resultados finais não são proporcionais ao tempo disponibilizado aos algoritmos, visto que os dados pouco se alteram quando comparados com o teste anterior; portanto, 1 minuto é tempo suficiente para os algoritmos obterem resultados que permitam que um usuário hipotético opte por um método ou outro, dado que 10 segundos impede que os algoritmos cumpram suas diretrizes de inicialização e/ou não avance gerações/iterações suficientes para termos certeza acerca dos resultados, e 10 minutos não produz resultados proporcionais ao tempo gasto.

Vale ressaltar que embora mais tempo disponibilizado aos métodos elimina o efeito do não cumprimento das diretrizes de inicialização, o efeito da aleatoriedade (ruído) ainda faz-se presente ainda que em menor proporção, visto que é uma característica inerente ao problema estudado neste trabalho.

Novamente, notamos a saturação de resultados encontrados após 1 minuto de execução para o AG versus AGGL, assim como observado na comparação entre o AG versus MG ou entre o AG versus AGG. As hipóteses levantadas no Capítulo 5, Seção 5.3.1, também se fazem válidas.

E claro, assim como para o AG versus MG e/ou AG versus AGGL, esses resultados são válidos somente para o tipo de SLSM estudados neste trabalho, apresentado em (2.1), e para controladores dinâmicos, apresentado em (2.4), e não existem garantias que os resultados mantêm-se para outros tipos de SLSMs e/ou para controladores estáticos.

7.3 Considerações sobre o Algoritmo Genético versus Algoritmo Genético-Gradiente e entre o Algoritmo Genético versus Algoritmo Genético Guloso

Com relação aos resultados obtidos entre o AG versus AGG e ao AG versus AGGL, notamos que o AG encontra mais resultados únicos quando comparado com o AGG, embora os métodos AG, AGG e AGGL assemelhem-se em suas diretrizes de inicialização, uma vez que trabalham com o mesmo número de soluções possíveis e lidam com tais soluções de forma análoga, através do *crossover*, etc. Entretanto, o AGG possui uma característica que, teoricamente, lhe garantiria vantagem em relação ao AG, que é aplicar um passo do MG à cada um dos “filhos” resultantes via *crossover*. Essa característica revelou-se vantajosa no quesito *qualidade* de soluções, uma vez que dentre as soluções em que ambos os algoritmos encontraram solução, o AGG destacou-se; contudo, sendo superior ao o AG, o AGG também não deveria destacar-se no quesito *quantidade* de soluções únicas encontradas? A resposta é “não” e a justificativa está na complexidade de ambos os algoritmos.

O MG executa varreduras (vide conjuntos $\mathbb{A}$ e $\mathbb{C}$) que o impedem que avance a mesma quantidade de iterações/gerações que o AG, uma vez que necessita de mais tempo e mais poder computacional, e isso pode ter contribuído no quesito *quantidade* de soluções únicas encontradas pelos métodos; por isso, na comparação entre o AG versus AGG, o AG encontra mais soluções únicas, já que avança mais gerações/iteraões que o AGG, uma vez que o AGG necessita efetuar tais varreduras, e consequentemente cálculos, para todos os “filhos” resultantes via *crossover*. Com relação ao AG versus AGGL, embora o AGGL também possua o MG em sua concepção, ele é utilizado em somente uma das soluções, aquela melhor localizada dentro do espaço de busca; como a varredura nos conjuntos $\mathbb{A}$ e $\mathbb{C}$ e cálculos são executados menos vezes, o AGGL avança quase a mesma quantidade de iterações/gerações que o AG, o que faz com que também destaque-se no quesito *quantidade* de soluções únicas encontradas.

Os testes para comprovar ou negar hipótese apresentada no parágrafo anterior não foram efetuados durante este trabalho; contudo, a análise dos algoritmos indica que tal hipótese seja verdadeira. Um possível teste poderia ser feito atribuindo tempo máximo de execução de 2 ou 3 minutos para o AGG e após, comparar os resultados obtidos com o AG.

7.4 Algoritmo Genético versus Algoritmo Genético Evolutivo

Para a primeira execução do algoritmo, o tempo máximo de execução será de 10 segundos para avaliarmos o seu desempenho nas mesmas condições impostas a todos os outros métodos analisados em capítulos anteriores. Entretanto, como o AGE deve considerar o decaimento

de certeza com relação a θ , fixaremos tempo máximo de 2 segundos para operações do AGE (*crossover*, etc) para cada elemento da malha de “ c ” (vide o que foi apresentado na Seção 6.2). A interpretação é a de dar tempo para que a população se adapte ao “novo ambiente”, que corresponde a um novo elemento da malha de “ c ”. Com tempo máximo de execução fixados em 10 segundos dividido em estágios de 2 segundos, obtemos 5 valores possíveis de observação de θ .

Optamos por dividir em estágios de 2 segundos pois, se disponibilizássemos menos tempo, o método poderia não dispor de tempo suficiente em cada possível “novo ambiente” para que a população se adaptasse, dificultando que atingíssemos o cenário sem observação (“ $c = \frac{1}{N}$ ”); caso disponibilizássemos mais tempo, a quantidade de estágios seria menor, e a diferença entre os valores com relação a θ entre um estágio e outro poderia dificultar a adaptação da população em cada “novo ambiente”, o que também dificultaria que atingíssemos o cenário sem observação, ou seja, “ $c = \frac{1}{N}$ ”.

As conclusões obtidas com relação ao tamanho da população inicial para ambos os algoritmos mantêm-se as mesmas descritas no Capítulo 5, Seção 5.2.

Também é importante evidenciar que para cada novo elemento na malha de “ c ”, a *matriz de probabilidade de transição* $\mathbb{P}$, a *distribuição inicial* $\pi(0)$ e a *distribuição Cesàro limite* devem ser redefinidas com base no novo valor do elemento da malha, como explicado no Capítulo 6, Seção 6.2.

Observação 14. Note que “ c ” ajuda na transição entre o cenário com observação (com θ observável) e o cenário sem observação (com θ não-observável). A conclusão óbvia é que “ c ” pode ser altamente reativo, ou seja, se sairmos do cenário com θ observável para o cenário com θ não-observável de maneira abrupta, o custo do sistema pode tender ao infinito. Assim sendo, ao alterarmos o valor da variável c , o custo do sistema também se alterará.

Dito isso, após 10 segundos de execução divididos entre 6 estágios de 2 segundos, obtivemos resultados dispostos nas Figuras 35 e 36 e nas Tabelas 32 e 33:

Figura 35 – Custos obtidos com o AG x Custos obtidos com o AGE - 10 segundos

Figura 36 – Estatísticas gerais AG x AGE - 10 segundos

Tabela 32 – Estatísticas gerais AG x AGE - 10 segundos

	%	erro padrão
SOMENTE o AG resolve	6,6%	0,7851
SOMENTE o AGE resolve	13,8%	1,0907
ambos os métodos resolvem	11,8%	1,0202
nenhum dos métodos resolve	67,8%	1,4776

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 33 – Estatísticas específicas AG x AGE - 10 segundos

	%	erro padrão
AG melhor que o AGE	0%	0
AGE melhor que o AG	7,3%	0,8226
sem diferença significativa entre os métodos	4,5%	0,6556

Após 10 segundos de execução, em 67,8% dos casos nenhum dos métodos foi capaz encontrar solução devido ao pouco tempo disponibilizado para aos algoritmos não somente para avançarem gerações/iterações, como também para cumprirem suas diretrizes de inicialização. Note que o pouco tempo disponibilizado aos algoritmos afeta similarmente todos dos métodos estudados neste trabalho, mantendo a porcentagem de instâncias sem solução próxima a 70%.

A porcentagem de instâncias em que ambos os métodos encontraram solução corresponde a 11,8% dos casos, e em somente 4,5% não existe diferença significativa entre eles. Comparando-os diretamente, vemos não existe casos em que o AG sobressai-se ao AGE. O modo como AGE lida com o conjunto de prováveis soluções, partindo da certeza com relação à observação de θ e diminuir aos poucos até atingir o cenário sem observação, faz com que a população adapte-se ao novo “ambiente” e que melhores soluções sejam encontradas; em contraste, o AG gera sua população inicial a partir de estados observáveis, mas busca soluções em cenário incerto, o que dificulta sua atuação dentro do espaço de busca.

Contudo, mesmo um método possuindo uma característica que teoricamente lhe concede vantagem, existirão situações que somente um dos métodos encontrará solução. Nesse cenário, vemos que o AGE destaca-se com 13,8% das soluções encontradas contra 6,6% do AG. Entretanto, esse resultado é influenciado pelo pouco tempo disponibilizado aos algoritmos para cumprirem suas diretrizes de inicialização e pela aleatoriedade inerente ao problema estudado.

É interessante notar que todos os métodos estudados sofreram com o pouco tempo disponibilizado, mantendo a porcentagem de instâncias sem solução em torno de 70% e impedindo que conclusões satisfatórias pudessem ser obtidas.

Reexecutamos o método aumentando o tempo máximo de execução para 1 minuto. Do mesmo modo que o teste anterior, dividimos o tempo total de execução em 6 estágios, mas

aumentamos o tempo de adaptação da população em cada “novo ambiente” para 10 segundos, sendo que, independentemente do valor de “ c ”, os últimos 10 segundos de execução serão destinados ao cenário sem observação (“ $c = \frac{1}{N}$ ”). A escolha por tal divisão dá-se pelos mesmos motivos apresentados no teste anterior.

Dito isso, após 1 minuto de execução divididos entre 6 estágios de 10 segundos, obtivemos os resultados dispostos nas Figuras 37 e 38 e nas Tabelas 34 e 35:

Figura 37 – Custos obtidos com o AG x Custos obtidos com o AGE - 1 minuto

Figura 38 – Estatísticas gerais AG x AGE - 1 minuto

Tabela 34 – Estatísticas gerais AG x AGE - 1 minuto

	%	erro padrão
SOMENTE o AG resolve	5,9%	0,7451
SOMENTE o AGE resolve	28,9%	1,4335
ambos os métodos resolvem	28,5%	1,4275
nenhum dos métodos resolve	36,7%	1,5242

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 35 – Estatísticas específicas AG x AGE - 1 minuto

	%	erro padrão
AG melhor que o AGE	0,2%	0,1413
AGE melhor que o AG	19,7%	1,2577
sem diferença significativa entre os métodos	8,6%	0,8866

Após 1 minuto de execução, vemos que a porcentagem de instâncias que não possuem solução diminui de 67,8% para 36,7%. Grande parte desse efeito é atribuído ao AGE e como ele lida com o espaço de busca, uma vez que parte da certeza com relação à observação de θ e “adapta” a população a cada novo “ambiente” até atingirmos o cenário sem observação, permitindo que instâncias antes sem solução pudessem ser resolvidas.

Com relação à porcentagem de instâncias que ambos os métodos encontraram solução, houve aumento de 11,8% para 28,5% com relação ao teste anterior e somente em 8,6% dos casos não existe diferença significativa entre os métodos. Com relação aos casos em que um método sobressai-se ao outro, vemos que o AG destaca-se em somente 0,2% dos casos, aumento irrisório comparado ao teste anterior, e esse comportamento também pode ser explicado pela maneira com que o AGE lida com o espaço de busca, partindo da certeza com relação à observação de θ e diminuir até atingir o cenário sem observação, o que lhe concede vantagem em relação ao AG.

Com relação as instâncias em que somente um dos métodos encontrou solução, notamos decaimento de 6,6% para 5,9% na porcentagem de instâncias em que somente o AG resolve, e aumento de 13,8% para 28,9% na porcentagem de instâncias em que somente o AGE resolve, quando comparamos com o teste anterior. Esses dados levam a conclusão de que com tempo suficiente disponibilizado aos métodos, o AGE encontra soluções que anteriormente somente o AG havia sido capaz de encontrar.

Mais tempo disponibilizado aos algoritmos revelou a tendência do AGE para encontrar melhores resultados, e além disso, permite que o método encontre soluções que anteriormente somente o AG tinha sido capaz. A fim de corroborar os resultados obtidos até o momento, além de possibilitar comparação dos métodos seguindo as mesmas restrições temporais apresentadas no início do Capítulo 5 (aos quais todos os métodos propostos neste trabalho estão sujeitos), aumentamos o tempo máximo de execução para 10 minutos. Dessa vez, como dispomos de mais tempo, optamos por dividir o tempo total em 10 estágios de 1 minuto cada, tempo mais que suficiente para que a população se adapte a cada um dos “novos ambientes”; tal divisão também permite que a diferença entre os valores dos elementos na malha de “ c ” sejam menores. Além disso, analogamente aos testes anteriores, o último minuto será destinado ao cenário sem observação (“ $c = \frac{1}{N}$ ”), independentemente do valor de “ c ” na etapa anterior na execução do algoritmo.

Dito isso, após 10 minutos de execução divididos entre 10 estágios de 1 minuto, obtivemos os resultados dispostos nas Figuras 39 e 40 e nas Tabelas 36 e 37:

Figura 39 – Custos obtidos com o AG x Custos obtidos com o AGE - 10 minutos

Figura 40 – Estatísticas gerais AG x AGE - 10 minutos

Tabela 36 – Estatísticas gerais AG x AGE - 10 minutos

	%	erro padrão
SOMENTE o AG resolve	3,4%	0,5731
SOMENTE o AGE resolve	44,4%	1,5712
ambos os métodos resolvem	32,1%	1,4763
nenhum dos métodos resolve	20,1%	1,2673

Dentre os casos em que ambos os métodos resolvem, temos:

Tabela 37 – Estatísticas específicas AG x AGE - 10 minutos

	%	erro padrão
AG melhor que o AGE	0,2%	0,1413
AGE melhor que o AG	23,1%	1,3328
sem diferença significativa entre os métodos	8,8%	0,8959

Após 10 minutos de execução, a porcentagem de instâncias sem solução diminui de 36,7% para 20,1%, devido, principalmente, à influência do AGE.

Com relação à porcentagem de instâncias em que somente um dos métodos foi capaz de encontrar solução, notamos que a diferença entre ambos é realçada, embora o aumento no tempo máximo de execução tenha favorecido o AGE; a porcentagem de instâncias que somente o AGE resolve aumenta de 28,9% para 44,4% enquanto o AG diminui de 5,9% para 3,4%. Esses dados nos levam a concluir que, com tempo suficiente disponibilizado aos algoritmos, o AGE é capaz de encontrar soluções que anteriormente somente o AG havia encontrado. Embora essa conclusão não esteja totalmente incorreta, ela é incompleta, uma vez que o que mais contribui para o aumento na porcentagem de instâncias resolvidas somente pelo AGE provém de casos anteriormente sem solução.

Também notamos aumento de 28,5% para 32,1% na porcentagem de instâncias em que ambos os métodos encontram solução, e dentre o total, não existe diferença significativa em 8,8% dos casos. O aumento no tempo máximo de execução realçou e confirmou a tendência que o AGE possui em encontrar melhores soluções, aumentando de 19,7% para 23,1%, enquanto a porcentagem em que o AG sobressai-se mantém-se em 0,2%, mesmo porcentagem obtida em relação ao teste anterior.

7.4.1 Discussões sobre o Algoritmo Genético versus Algoritmo Genético Evolutivo

O aumento do tempo máximo de execução, além de possibilitar que os métodos encontrassem soluções para mais instâncias, confirmou tendências. Entretanto, diferentemente dos outros métodos analisados neste trabalho, em que os dados permitiram concluirmos que 1 minuto

é tempo suficiente para permitir que um usuário hipotético opte por um algoritmo, não podemos afirmar o mesmo quando trata-se do AGE, uma vez que podemos classificar como significativo o decremento de instâncias sem solução entre um teste e outro. Além disso, o AGE destaca-se tanto em número de soluções distintas como em mais aptas. Portanto, podemos afirmar que um usuário hipotético deveria optar pelo uso do AGE para encontrar o controle para o SLSM com as características estudadas neste trabalho, e deveria disponibilizar ao método o tempo máximo de 10 minutos, uma vez que faz uso apropriado e dos recursos disponíveis e os resultados que obtém são proporcionais ao tempo gasto.

Diferentemente dos demais métodos analisados, o AGE obteve bons resultados após 10 minutos de execução, e, portanto, é intuitivo pensar que os resultados seriam ainda melhores caso o método dispusesse de mais tempo para avançar gerações/iterações; contudo, isso não acontece. Reexecutamos o método aumentando o tempo máximo de execução para 20 minutos e os dados pouco se alteraram (em torno de 1% para mais ou para menos em cada uma dos quesitos analisados); aumentamos para 30 minutos e os resultados foram parecidos com aqueles que obtivemos após 10 minutos e que já foram apresentados. Assim como acontece com os demais métodos, notamos uma saturação na porcentagem de instâncias em que os métodos encontram solução, com a diferença de que o AG, o MG, o AGG e o AGGL, tal saturação dá-se após 1 minutos de execução, enquanto o AGE atinge seu ápice em soluções encontradas após 10 minutos.

Analogamente a todos os métodos estudados, esses resultados são válidos somente para o tipo de SLSM estudados neste trabalho, apresentado em (2.1), e para controladores dinâmicos, apresentado em (2.4), e não existem garantias que os resultados mantêm-se para outros tipos de SLSMs e/ou para controladores estáticos.

Também vale ressaltar que divisão da malha de “c” não se dá por nenhum motivo específico além de praticidade. Nada impede de aumentarmos ou diminuirmos o número de estágios e/ou o tempo em que o algoritmo usará para “treinar” a população em cada estágio ou mesmo o tempo total de execução do algoritmo. Por exemplo: poderíamos aumentar o tempo de execução do algoritmo para 20 minutos e mantermos 10 estágios fixados em 1 minuto cada, e desse modo o algoritmo teria mais tempo para trabalhar com todos os estágios disponíveis; ou ainda, poderíamos manter os 10 minutos como tempo máximo para execução do algoritmo, mas diminuir o tempo de “treino” da população em cada estágio para 30 segundos. Enfim, diversas possibilidades surgem e, nesses casos, um estudo mais detalhado sobre o comportamento do método seria necessário. Além disso, também deve ser levado em consideração que, quanto menos estágios, maior a diferença de valores que “c” pode assumir entre estágios, e consequentemente, maior a dificuldade de adaptação da população ao “novo ambiente”; quanto mais estágios, menor a diferença entre os valores que “c” e mais fácil a adaptação da população, contudo, isso pode requerer mais poder computacional e/ou mais tempo total de execução para o algoritmo.

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste trabalho, estudamos o problema linear quadrático na forma CMLP, dado que esse tipo de problema ainda possui lacunas no cenário considerado neste trabalho - sem observação da variável de salto θ e com controlador dinâmico com o termo de realimentação de saída (aquele que envolve a matriz D_c em (2.17)). De fato, esse cenário ainda não havia sido considerado na literatura, de maneira que não havia nem sequer a formulação determinística para cálculo de custo, fornecida por nós no Lema 1.

Apresentamos dois métodos para tratar o problema da Seção 2.2, sendo eles o Algoritmo Genético (Capítulo 3) e o Método Gradiente (Capítulo 4), cujos resultados provindos a partir de análises de instâncias aleatórias geradas a partir de um gerador de SLSM desenvolvido pelo nosso grupo de pesquisa, revelaram características diferentes que foram combinadas visando soluções melhores; os dados revelaram que o Algoritmo Genético destaca-se na quantidade de soluções encontradas, enquanto o Método Gradiente destaca-se na qualidade das soluções. Além disso, analisamos os resultados restringindo os algoritmos a um tempo máximo de execução de 10 segundos, 1 minuto e 10 minutos, indicando que, para o tipo de SLSM estudado neste trabalho, 1 minuto é tempo suficiente para que os métodos encontrem soluções que permitam que um usuário hipotético opte por um método ou por outro.

Os dados provindos das análises entre o Algoritmo Genético e o Método Gradiente também nos levaram ao estudo de métodos híbridos, e com essa motivação, dois métodos foram apresentados: o Algoritmo Genético-Gradiente e o Algoritmo Genético Guloso. O primeiro consistiu em acoplarmos o MG ao AG e aplicarmos um passo do MG para cada solução provinda via *crossover* em cada geração do AG. O segundo consistiu em substituímos toda a população entre uma geração e outra, e além disso, aplicarmos o MG no indivíduo mais apto da população. Essas duas hibridizações, embora possam ser interpretadas como simples, foram suficientes para obtermos resultados superiores, vide os resultados obtidos através das comparações com o AG

“clássico” apresentado no Capítulo 3.

Além das duas hibridizações simples entre o AG e MG, propusemos um terceiro método inspirado na abordagem em (SILVA, 2012). Como explicado na Seção 6.2, ideia consiste em modificar o AG apresentado no Capítulo 3 para que inicie com uma população inicial bastante apta para o “problema auxiliar”, e que vai sendo alterada gradativamente, uma vez que, recapitulando, a proposta com o AGE, é inicializarmos o método a partir de variáveis de salto observáveis, e aos poucos diminuirmos a certeza com relação à observação de tais variáveis até atingirmos o cenário sem observação. Isso permitiu que as populações se adaptassem, evoluíssem com as mudanças no problema, e por isso o nome “Algoritmo Genético Evolutivo”. Essa capacidade mostrou-se altamente relevante nas instâncias consideradas, uma vez que o AGE obteve melhores soluções na grande maioria das vezes, vide as Tabelas 32, 33, 34, 35, 36 e 37.

Levando em conta o conjunto completo de resultados obtidos, nota-se que a inicialização dos métodos é uma questão fundamental. Indício forte disso é que o melhor método obtido é aquele que lida melhor com essa questão. O alto número de instâncias sem solução para com os métodos AG, MG, AGG e AGGL pode ser influência do seu modo de inicialização, ou seja, a *combinação convexa* das matrizes A_c , B_c , C_c e D_c atuantes no controlador apresentado em (2.4), o controlador dependente do modo θ de operação. Para trabalhos futuros, meios alternativos de inicialização para os métodos podem ser investigados, como, por exemplo, utilizar os resultados em (DOLGOV, 2017) para inicializar os métodos aqui estudados.

Ainda com relação a trabalhos futuros, há ainda muitas questões a investigar. Exemplos mais óbvios são a extensão para outras classes de sistemas (como a versão em tempo contínuo, a versão com equações algébrico-diferenciais (também chamado de sistemas singulares), com entradas exógenas, com restrições, etc.). Ainda na mesma classe de sistemas estudada aqui, julgamos que há três pontos importantes em que se pode avançar. Primeiro, as questões de existência de estabilidade da solução: é possível saber, antes de executar o algoritmo, se existe solução e se a solução estabiliza o sistema (baseando-se apenas nos dados do problema); no caso sem saltos, por exemplo, há condições chamadas de estabilizabilidade, que garantem a existência e estabilidade de soluções. Segundo, propor outros métodos, possivelmente mais específicos de controle, incluindo-se aqui a adaptação do método variacional em (BORTOLIN, 2012) e (SILVA, 2012). Terceiro, buscar formas de aproveitar melhor o “problema auxiliar” e a forma de transição entre ele e o problema original (em outras palavras, como variar “ c ” de forma a tornar os métodos que o utilizem mais eficientes. Vale ressaltar, nesse sentido, que observamos que o custo depende de “ c ” de uma forma não-linear, tendo uma alta sensibilidade a “ c ” em pequenos intervalos, intervalos esses bastante variados conforme a instância).

REFERÊNCIAS

- ABADIR, K. M.; MAGNUS, J. R. **Matrix Algebra (Econometric Exercises)**. [S.l.]: Cambridge University Press, 2005. Citado na página 139.
- AFFENZELLER, M.; WAGNER, S.; WINKLER, S.; BEHAM, A. **Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications**. [S.l.]: A Chapman and Hall Book, 2009. ISBN 978-584488-629-7. Citado na página 33.
- ALABSI, F.; NAOUM, R. Comparison of selection methods and crossover operations using steady state genetic based intrusion detection system. **Journal of Emerging Trends in Computing and Information Sciences**, v. 03, n. 07, 2012. ISSN 2079-8407. Citado na página 39.
- BLACKMORE, L.; ONO, M.; BEKTASSOV, A.; WILLIAMS, B. C. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. **IEEE Transactions on Robotics**, v. 26, n. 3, p. 502–517, 2010. Citado na página 21.
- BORTOLIN, D. C. **Métodos numéricos para o controle linear quadrático com saltos e observação parcial de estado**. Dissertação (Mestrado) — Universidade de São Paulo, São Carlos, 2012. Citado nas páginas 46, 55, 58 e 124.
- BURDEN, R. L.; FAIRES, J. D. **Análise Numérica**. 8. ed. [S.l.]: Cengage Learning, 2008. Citado na página 48.
- CARVALHO, A. P. de Leon F. de. **Algoritmos Genéticos**. [S.l.], 2009. Disponível em: <<https://conteudo.icmc.usp.br/pessoas/andre/research/genetic/>>. Acesso em: 26 fev. 2019. Citado nas páginas 33 e 42.
- ÇINLAR, E. **Introduction to Stochastic Processes**. [S.l.]: Pretince-Hall, Inc., 2013. Citado na página 135.
- COSTA, O. L. do V.; FRAGOSO, M. D.; TODOROV, M. G. A detector-based approach for the h_2 control of markov jump linear systems with partial information. **IEEE Transactions on Automatic Control**, v. 60, p. 1219–1234, 2015. Citado na página 87.
- COSTA, O. L. V.; ARAUJO, M. V. A generalized multi-period mean-variance portfolio optimization with markov switching parameters. **Automatica**, v. 44, n. 10, p. 2487–2497, 2008. Citado na página 21.
- COSTA, O. L. V.; FRAGOSO, M. D.; MARQUES, R. P. **Discrete-Time Markovian Jump Linear Systems**. New York: Springer-Verlag, 2005. Citado na página 22.
- COSTA, O. L. V.; FRAGOSO, M. D.; TODOROV, M. G. **Continuous-Time Markov Jump Linear System**. Berlin, Heidelberg: Springer, 2013. Citado nas páginas 22 e 31.
- DEB, K. **Multi-objective optimization using evolutionary algorithms**. New York: John Wiley & Sons, 2001. (Wiley-Interscience Series in Systems and Optimization). Citado na página 35.

DOLGOV, M. **Approximate Stochastic Optimal Control of Smooth Nonlinear Systems and Piecewise Linear Systems**. Tese (Doutorado) — Faculdade de Informática do Instituto de Tecnologia de Karlsruhe, St. Petersburg, 2017. Citado nas páginas 24, 32 e 124.

DRAGAN, V.; COSTA, E. F. Optimal stationary dynamic output-feedback controllers for discrete-time linear systems with markovian jumping parameters and additive white noise perturbations. **IEEE Transactions on Automatic Control**, v. 61, p. 3912–3924, 2016. Citado nas páginas 23, 24, 27, 30, 37, 47, 135 e 147.

DRAGAN, V.; MOROZAN, T.; STOICA, A.-M. **Mathematical methods in robust control of discrete-time linear stochastic systems**. [S.l.]: Springer, 2009. Citado na página 22.

EIBEN Ágoston E.; HINTERDING, R.; MICHALEWICZ, Z. Parameter control in evolutionary algorithms. **IEEE Transactions on Evolutionary Computation**, v. 3, n. 2, p. 124–141, 1999. Citado na página 57.

FOGEL, D. B. An introduction to simulated evolutionary optimization. **IEEE Transaction on Neural Networks**, v. 5, n. 1, 1994. Citado na página 43.

GONÇALVES, A. P. C. Discussion on H_∞ control for networked control systems with limited communication. **European Journal of Control**, v. 18, p. 119–121, 2012. Citado na página 21.

GONÇALVES, A. P. C.; FIORAVANTI, A. R.; GEROMEL, J. C. Markov jump linear systems and filtering through network transmitted measurements. **International Journal of Robust and Nonlinear Control**, v. 21, p. 613–624, 2011. Citado na página 21.

GONG, D.; SUN, J.; MIAO, Z. A set-based genetic algorithm for interval many-objective optimization problems. **IEEE Transactions on Evolutionary Computation**, v. 22, 2016. Citado na página 57.

GOUVÊA Érica J. C. **Métodos convergentes de otimização global baseados no vetor q-gradiente**. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2016. Citado na página 45.

GREFENSTETTE, J. J. Optimization of control parameters for genetic algorithms. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, v. 16, n. 1, p. 122–128, 1986. Citado na página 57.

HOEL, P. G.; PORT, S. C.; STONE, C. J. **Introduction to Stochastic Processes**. [S.l.]: Houghton Mifflin Company, 1972. Citado na página 135.

HOLLAND, J. H. **Adaptation in natural and artificial systems**. [S.l.]: University of Michigan Press, 1975. Citado na página 33.

HWANG, I.; KIM, S.; KIM, Y.; SEAH, C. E. A survey of fault detection, isolation, and reconfiguration methods. **International Journal of Control**, v. 18, n. 3, p. 481–498, 2010. Citado na página 21.

JIANG, J.; YU, X. Fault-tolerant control systems: A comparative study between active and passive approaches. **Annual Reviews in Control**, v. 36, n. 1, p. 60–72, 2012. Citado na página 21.

JONG, K. D. **The analysis of the behavior of a class of genetic adaptive systems**. Tese (Doutorado) — University of Michigan, Ann Harbor, 1975. Citado na página 57.

- KUMAR, P. R.; VARAIYA, P. **Stochastic systems: Estimation, identification and adaptive control**. [S.l.]: SIAM, 2015. v. 75. ISBN 978-1-611974-25-6. Citado na página 23.
- LUENBERGER, D. **Linear and nonlinear programming**. 2. ed. [S.l.]: Addison-Wesley, 1984. Citado na página 52.
- MAN, K. F.; TANG, K. S.; KWONG, S. Genetic algorithms: Concepts and applications. **IEEE Transactions on Industrial Electronics**, v. 43, n. 5, p. 519–534, 1996. Citado na página 57.
- MARITON, M. Detection delays, false alarm rates and the reconfiguration of control systems. **International Journal of Control**, v. 49, p. 981–992, 1989. Citado na página 21.
- MENDONÇA, E. B. de. **Teoria das Filas Markovianas e Aplicações**. Monografia (Graduação) — Universidade Estadual da Paraíba - Campina Grande, 2014. Citado na página 131.
- MICHALEWICS, Z. **Genetic algorithms + data structures = evolution programs**. [S.l.]: Springer, 1996. Citado nas páginas 35 e 43.
- MITCHELL, M. **An Introduction to Genetic Algorithms**. [S.l.: s.n.], 1996. ISBN 0-262-13316-4. Citado nas páginas 33, 42 e 90.
- MOLER, C. **A Brief History of MATLAB**. [S.l.], 2018. Disponível em: <<https://www.mathworks.com/company/newsletters/articles/a-brief-history-of-matlab.html>>. Acesso em: 23 fev. 2019. Citado na página 25.
- MUFEED, M.; JIN, J.; YOUNG, Z. **Active Fault Tolerant Control Systems - Stochastic Analysis and Synthesis**. [S.l.]: Springer Verlag, 2013. ISBN 3-540-00318-5. Citado na página 21.
- NEWTON, I. **The Migration Ecology of Birds**. [S.l.: s.n.], 2007. ISBN 9780125173674. Citado na página 88.
- NOCEDAL, J.; WRIGHT, S. J. **Numerical optimization**. 2a. ed. New York: Springer-Verlag, 2006. (Springer Series in Operations Research). Citado nas páginas 45 e 47.
- OLIVEIRA, A. M. de; COSTA, O. Mixed H_2/H_∞ control of hidden markov jump systems. **International Journal of Robust and Nonlinear Control**, v. 28, p. 1261–1280, 2018. Citado na página 87.
- OLIVEIRA, A. M. de; COSTA, O. L. do V. H_∞ -filtering for markov jump linear systems with partial information on the jump parameter. **IFAC Journal of Systems and Control**, v. 1, p. 13–23, 2017. Citado na página 87.
- OLIVEIRA, A. M. de; COSTA, O. L. do V.; DAAFOUZ, J. Design of stabilizing dynamic output feedback controllers for hidden markov jump linear systems. **IEEE Control Systems Letters**, v. 2, p. 278–283, 2018. Citado na página 87.
- OLIVEIRA, L. T. de; COSTA, E. F. Sobre a unicidade da solução no controle de sistemas lineares com saltos markovianos não observados. **Proceeding Series of the Brazilian Society of Applied and Computational Mathematics**, v. 3, n. 1, 2015. Citado nas páginas 24, 32, 46, 77 e 105.

- PALMA, J. M.; CARVALHO, L. D. P.; GONÇALVES, A. P. C.; GALARZA, C. E.; OLIVEIRA, A. M. de. Networked control systems application: Minimization of the global number of interactions, transmissions and receptions in multi-hop network using discrete-time markovian jump linear systems. **IEEE Latin America Transactions**, v. 14, n. 6, p. 2675–2680, 2016. Citado na página 21.
- PATTON, R. J. Fault-tolerant control systems: the 1997 situation. **IFAC SYMPOSIUM ON FAULT DETECTION SUPERVISION AND SAFETY FOR TECHNICAL PROCESSES**, p. 1033–1054, 1997. Citado na página 21.
- PILLO, G. D.; PALAGI, L. **Nonlinear programming: Introduction**. New York: Oxford University Press, 2002. Citado na página 45.
- ROEVA, O.; FIDANOVA, S.; PAPRZYCKI, M. Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. **Proceedings of the 2013 Federated Conference on Computer Science and Information Systems**, p. 371–376, 2013. Citado na página 57.
- SARIDIS, G. N. Intelligent robotic control. **IEEE Transactions on Automatic Control**, n. 28, p. 547–557, 1983. Citado na página 21.
- SCHUETTE, C.; METZNER, P. **Markov Chains and Jump Processes - An Introduction to Markov Chains and Jump Processes on Countable State Spaces**. [S.l.: s.n.], 2009. Citado na página 135.
- SILVA, C. A. **Algoritmos para o custo médio a longo prazo de sistemas com saltos Markovianos parcialmente observados**. Tese (Doutorado) — Universidade de São Paulo, São Carlos, 2012. Citado nas páginas 28, 39, 55, 58, 85 e 124.
- SIQUEIRA, A. A. G.; TERRA, M. H. Nonlinear and markovian H_∞ -controls of underactuated manipulators. **IEEE Transactions On Control Systems Technology**, v. 12, n. 06, p. 811–826, 2004. Citado na página 21.
- SRICHANDER, R.; WALKER, B. K. Stochastic stability analysis for continuous-time fault tolerant control systems. **International Journal of Control**, v. 57, n. 2, p. 433–452, 1993. Citado na página 21.
- STEIN, G. Respect the unstable. **IEEE Control Systems**, v. 23, n. 4, p. 12–25, 2003. Citado na página 21.
- STOICA, P.; YAESH, I. Jump markovian-based control of wing deployment for an uncrewed aircraft. **J. Guidance**, v. 25, n. 2, p. 407–411, 2002. Citado na página 21.
- SWORDER, D. D.; ROGERS, R. O. An lq-solution to a control problem associated with a solar thermal central receiver. **IEEE Transactions on Automatic Control**, v. 28, n. 10, p. 971–978, 1983. Citado na página 21.
- TUGNAIT, J. K. Control of stochastic systems with markov interrupted observations. **IEEE Transactions on Aerospace and Electronic Systems**, v. 19, n. 2, p. 232–239, 1983. Citado na página 21.
- UMBARKAR, A.; SHETH, P. Crossover operators in genetic algorithms: a review. **Ictact journal on soft computing**, v. 06, 2015. ISSN 2229-6956. Citado nas páginas 35 e 42.

VAL, J. B. R. do; BAŞAR, T. Receding horizon control of jump linear systems and a macroeconomic policy problem. **Journal of Economic Dynamics & Control**, v. 23, p. 1099–1131, 1999. Citado na página [21](#).

VARGAS, A. N.; COSTA, E. F.; VAL, J. B. R. do. Advances in the control of markov jump linear systems with no mode observation. In: . [S.l.]: Springer International Publishing, 2016. ISBN 978-3-319-39834-1. Citado nas páginas [22](#) e [32](#).

YADAV, S. L.; SOHAL, A. Comparative study of different selection techniques in genetic algorithm. **International Journal of Engineering, Science and Mathematics**, v. 06, 2017. ISSN 2320-0294. Citado na página [39](#).

ZHANG, Y.; JIANG, J. Bibliographical review on reconfigurable fault-tolerant control systems. **Annual Reviews in Control**, v. 32, n. 2, p. 229–252, 2008. Citado na página [21](#).

ZHAO, J.; YAO, F.; WANG, H.; MI, Y.; WANG, Y. Research on application of genetic algorithm in optimization design of transformer. **4th International Conference on Electric Utility Deregulation and Restructuring and Power Technologies**, 2011. Citado na página [57](#).

CADEIAS DE MARKOV

A.1 Conceitos gerais

Em 1907, o matemático russo Andrei Andreyevich Markov iniciou um estudo de um importante tipo de processo em que o resultado atual de um determinado experimento pode ser utilizado para determinar resultados em instantes seguintes, ou seja, as experiências anteriores não influenciam experiências futuras. Esse tipo de propriedade, também chamada de "Propriedade de Markov", caracteriza cadeia de Markov, também conhecido como "Processo Markoviano" ou "Processo de Markov".

Processos de Markov formam um tipo de processo estocástico em que a distribuição do estado seguinte $\theta(t+1)$ do processo depende somente do estado atual $\theta(t)$ e não de estados anteriores. Assumindo-se essa premissa, podemos escrever

$$P(\theta(t+1) = j | \theta(t) = i) = p_{ij}, \quad (\text{A.1})$$

onde p_{ij} é a probabilidade de transição do estado i para o estado j ; de maneira análoga, p_{ij} é a probabilidade de transição do estado i para o estado j .

Atualmente, cadeias de Markov têm sido utilizadas em diversas áreas do conhecimento, tais como ciências sociais, biológicas e administrativas; na física, são comumente utilizadas na mecânica estatística; nas ciências da informação são utilizadas no processamento de sinais, codificação de dados e até mesmo em reconhecimento de padrões; em finanças, descrevem a evolução do preço de ativos; são utilizados até mesmo na *teoria das filas*, ramo da matemática que teve início com o matemático dinamarquês Agner Krarup Erlang, visando solução para o problema de congestionamento em redes de telefonia ([MENDONÇA, 2014](#)).

Definição 1. Sejam $\theta(t) \in \mathcal{N} = \{1, 2, \dots, N\}$ os N estados possíveis de uma cadeia de Markov, a probabilidade de o sistema estar no estado θ_j em qualquer observação se no instante anterior

o sistema estava no estado θ_i , é denominado p_{ij} e é chamada *probabilidade de transição* de θ_i para o estado θ_j .

Existem vários outros conceitos inerentes a cadeias de Markov. Entre eles, e os mais relevantes para este trabalho, valem a pena ser destacadas a *matriz de probabilidade de transição*, a *distribuição de probabilidade inicial* e a *distribuição Cesàro limite*.

As probabilidades de transição entre os estados podem ser representados através de uma matriz $\mathbb{R}^{N \times N}$, onde N é a quantidade de estados da cadeia, assim, os elementos da i -ésima linha representam a probabilidade de salto para todos os estados (incluindo o salto do estado i na diagonal $\mathbb{P}$).

$$\mathbb{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & p_{22} & \cdots & p_{2N} \\ \vdots & \vdots & \cdots & \vdots \\ p_{N1} & p_{N2} & \cdots & p_{NN} \end{bmatrix} \quad (\text{A.2})$$

A *distribuição de probabilidade inicial* $\forall \theta \in \mathcal{N} = \{1, 2, \dots, N\}$ é definida por $\pi(0) = P[\theta(0) = i]$ e escrevemos $\pi(0) = [\pi_1(0), \pi_2(0), \dots, \pi_N(0)]$, tal que

$$\sum_{x=1}^N \pi_0(x) = 1. \quad (\text{A.3})$$

Dadas a *matriz de probabilidade de transição* e a *distribuição de probabilidade inicial*, podemos calcular a distribuição da cadeia em qualquer momento como segue

$$\begin{aligned} \pi_1 &= \pi_0 \mathbb{P} \\ \pi_2 &= \pi_1 \mathbb{P} \\ \pi_{t+1} &= \pi_t \mathbb{P}; \end{aligned} \quad (\text{A.4})$$

ou

$$\begin{aligned} \pi_{t+1} &= \pi_t \mathbb{P} \\ &= (\pi_{(t-1)} \mathbb{P}) \mathbb{P} \\ &= \pi_0 \mathbb{P}^t, \end{aligned} \quad (\text{A.5})$$

com $t \in \mathbb{N}$.

A transição entre estados de Markov também pode ser representado através de um grafo.

Figura 41 – Esquema de uma cadeia de Markov com dois estados.

Fonte: Elaborada pelo autor.

O estudo de cadeias de Markov envolve também, o estudo de sistemas dinâmicos que evoluem ao longo do tempo. Sendo assim, é de interesse o estudo do comportamento destas cadeias após um longo período de tempo. Tal necessidade nos dirige ao conceito da *distribuição Cesàro limite*, que pode ser interpretada distribuição "média ao longo prazo" da cadeia de Markov, é única, existe para toda cadeia e é dado por

$$\rho = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^t \mathbb{P}^k \pi(0), \quad (\text{A.6})$$

Vale frisar que ρ_j pode depender de $\pi(0)$.

A.2 Classificação dos estados em cadeias de Markov

Um estado é **transiente** se, dado um processo aleatório no estado i , tal processo pode nunca retornar para este estado. Então, o estado i é **transiente** se, e somente se, existir um estado j , onde $i \neq j$, que é acessível a partir do estado i mas não vice-versa. Em outras palavras, se o estado i é **transiente** e o processo visita este estado, existe a probabilidade que o processo irá mover-se para o estado j e nunca mais retornará para o i .

Um estado é **recorrente** se, considerando um estado i , o processo visitará outros estados e retornará para o estado i . Com isso, pode-se concluir que um estado é **recorrente** se, e somente se, ele não é **transiente**. Vale frisar que tal estado será revisitado mas não necessariamente no próximo passo do processo.

Um processo é **absorvente** se, considerando o estado i , o processo nunca irá deixar este estado. Portanto, um estado i é **absorvente** se, e somente se, $p_{ii} = 1$.

Exemplo 3. Suponha que determinada cadeia de Markov possua a seguinte *matriz de probabilidade de transição* $\mathbb{P}$.

$$\mathbb{P} = \begin{bmatrix} 0.25 & 0.75 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.35 & 0.65 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (\text{A.7})$$

O estado *quatro* é **transiente** pois se o processo estiver neste estado, existe a probabilidade que ele nunca retornará para este estado. O estado *cinco* também é **transiente** pois se o processo começar neste estado, ele imediatamente o deixa e nunca mais retornará.

Os estados *um* e *dois* são **recorrentes**, pois observando a matriz $\mathbb{P}$, nota-se que o processo começar em um destes dois estados, ele sempre retornará para tais estados.

O estado *três* é **absorvente**, pois uma vez neste estado, o processo nunca mais o deixará.

Outra propriedade que vale a pena ser mencionada é a **periodicidade**. Um estado i é considerado periódico t se o retorno a este estado é possível em $2t, 3t, \dots, \forall t \in \mathbb{Z}_+$ tal que $t > 1$.

Exemplo 4. A matriz $\mathbb{H}$ representa uma cadeia de Markov com período *três*.

$$\mathbb{H} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad (\text{A.8})$$

Se assumirmos que o processo inicia no estado *um*, observando a *matriz de probabilidade de transição* $\mathbb{H}$, nota-se que o processo partirá para o estado *dois*, depois para o estado *três* e novamente para o estado *um*, adquirindo periodicidade *três*.

Observação 15. $t = 1$ corresponde ao caso aperiódico.

A.3 Exemplo de aplicação

Exemplo 5. Uma agência de turismo vende pacotes de viagem para três destinos distintos, que chamaremos de A, B e C. No primeiro momento de vendas, as probabilidades de escolha são de 40% para o destino A, 30% para o destino B e 30% para o destino C. Em um segundo momento, das pessoas que inicialmente aderiram ao destino A, 15% migram para o destino B e 25% migram para C; das pessoas que escolheram o destino B, 45% migram para o destino A e 15% migram para C, e das pessoas que escolheram C, 20% migram para A e 25% migram para B. Com base nestes dados, determine as probabilidades de escolha para o momento *três*.

Todos os dados de que necessitamos para a resolução do problema estão no enunciado, e com as primeiras informações que foram introduzidas podemos montar o *distribuição de probabilidade inicial*,

$$\pi_0 = [0.4 \ 0.3 \ 0.3], \quad (\text{A.9})$$

sendo que a primeira posição indica a porcentagem de pessoas que, lembrando, em um primeiro momento escolhem o destino A, a segunda posição indica a porcentagem de pessoas que escolhem o destino B e finalmente, na terceira posição, a porcentagem de pessoas que escolhem o destino C.

Após, com base nos dados nos clientes que migram de um destino para outro, montamos a matriz de probabilidade de transição

$$\mathbb{P} = \begin{bmatrix} P(\theta_{t+1} = A | \theta_t = A) & 0.15 & 0.25 \\ 0.45 & P(\theta_{t+1} = B | \theta_t = B) & 0.15 \\ 0.2 & 0.25 & P(\theta_{t+1} = C | \theta_t = C) \end{bmatrix}. \quad (\text{A.10})$$

Como estamos lidando com probabilidades, cada uma das linhas na matriz $\mathbb{P}$ deve somar *um*. Logo, as probabilidades de as pessoas, em um segundo momento, escolherem o mesmo destino dá-se por

$$\begin{aligned} P(\theta_{t+1} = A | \theta_t = A) &= 1 - (P(\theta_{t+1} = B | \theta_t = A) + P(\theta_{t+1} = C | \theta_t = A)) \\ P(\theta_{t+1} = B | \theta_t = B) &= 1 - (P(\theta_{t+1} = A | \theta_t = B) + P(\theta_{t+1} = C | \theta_t = B)) \\ P(\theta_{t+1} = C | \theta_t = C) &= 1 - (P(\theta_{t+1} = A | \theta_t = C) + P(\theta_{t+1} = B | \theta_t = C)). \end{aligned} \quad (\text{A.11})$$

Então, substituindo (A.11) em (A.10), obtemos

$$\mathbb{P} = \begin{bmatrix} 0.6 & 0.15 & 0.25 \\ 0.45 & 0.4 & 0.15 \\ 0.2 & 0.25 & 0.55 \end{bmatrix}, \quad (\text{A.12})$$

e aplicando o que foi apresentado em (A.5),

$$\pi_3 = \pi_0 \mathbb{P}^3 = [0.4363 \quad 0.2429 \quad 0.3208]. \quad (\text{A.13})$$

Ou seja, no momento *três*, 43,63% das pessoas tendem a escolher o destino A, 24,29% tendem a escolher o destino B e 32,08% tendem a escolher o destino C.

É claro que existem vários problemas que podem ser resolvidos com a utilização de cadeias de Markov que são mais complexos do que este que foi apresentado, e como exemplo podemos citar o problema que motivou este trabalho (DRAGAN; COSTA, 2016). Contudo, aqui o foco não é apresentar um estudo detalhado e complexo sobre cadeias de Markov, mas sim introduzir conceitos básicos que servem como base para auxiliar no entendimento do problema original e, por sua vez, no problema tratado neste trabalho. Para se aprofundar no assunto, consultar (ÇINLAR, 2013) (SCHUETTE; METZNER, 2009) (HOEL; PORT; STONE, 1972).

RECURSOS COMPUTACIONAIS

B.1 Uso dos recursos computacionais

Embora os amplos recursos presentes nos computadores atuais (capacidade de processamento memória RAM, HDs, etc) permita a resolução de uma gama gigantesca de problemas, o bom gerenciamento de tais recursos é imprescindível à todo programador que propõe-se a usá-los. Com isto em mente, o código desenvolvido neste trabalho foi elaborado buscando não somente a resolução do problema proposto, mas também o bom uso dos recursos disponíveis.

A primeira providência foi com relação à alocação de memória. Pré-alocar memória garante execução mais rápida dos algoritmos pois o compilador não precisa se preocupar em encontrar espaços livres e contínuos na memória RAM que comportem o volume de dados proveniente das estruturas de dados (como matrizes e vetores) caso necessitem expandir suas dimensões durante tempo de execução. Pré-alocar memória é algo básico e um conceito fundamental apresentado a toda pessoa que está aprendendo a programar, contudo, em algumas linguagens de programação, pré-alocar memória é algo opcional (no MATLAB[®] por exemplo, onde todo o código deste trabalho foi desenvolvido) o que acarreta em *overhead* (em Ciência da Computação, *overhead* é qualquer processamento em excesso que seja utilizado para realizar determinada tarefa).

Outra medida diz respeito em como algumas variáveis foram representadas computacionalmente. Como todo trabalho foi desenvolvido no MATLAB[®], funções matemáticas e demais representações utilizadas neste ambiente estão otimizadas, portanto, cálculos realizados matricialmente contribuirão para o desempenho geral do algoritmo.

1. Tomemos a variável de decisão g , presente tanto no AG, Eq. (3.2), quanto no MG, Eq. (4.4). Sintetizando, tal variável é responsável por representar as possíveis soluções e que são utilizadas em vários estágios no decorrer da execução do código. Se os cálculos necessários fossem efetuados para cada matriz A_c , B_c , C_c e D_c separadamente, estaríamos

executando o mesmo trecho de código quatro vezes, um para cada matriz, já que o cálculo não difere. Portanto, a vetorização otimiza o tempo de execução e evita laços de repetição desnecessários.

2. Um conceito análogo ao apresentado anteriormente se aplica durante a realização do *crossover* no AG. Em cada geração, vários indivíduos “filhos” irão surgir, sendo que cada “filho” é o resultado entre a *combinação convexa* entre um “casal” (seguindo as diretrizes apresentadas na Seção 3.2), portanto, estes indivíduos também foram vetorizados antes de realizarmos o *crossover*. Considere as matrizes,

$$g_p^{ell} = \begin{bmatrix} p_1^{ell} \\ p_1^{ell} \\ \vdots \\ p_i^{ell} \end{bmatrix} \text{ e } g_m^{ell} = \begin{bmatrix} m_1^{ell} \\ m_2^{ell} \\ \vdots \\ m_i^{ell} \end{bmatrix}, \quad (\text{B.1})$$

sendo que $p_1, \dots, p_i$ representam indivíduos “pais”, $m_1, \dots, m_i$ representam indivíduos “mães” tal que p_i e m_i sejam distintos na ell -ésima geração. Aplicando a Definição 2 teremos:

$$g_f^{ell} = \lambda_f g_m^{ell} + (1 - \lambda_f) g_p^{ell},$$

$$\begin{bmatrix} f_1^{ell} \\ f_2^{ell} \\ \vdots \\ f_i^{ell} \end{bmatrix} = \lambda \begin{bmatrix} p_1^{ell} \\ p_2^{ell} \\ \vdots \\ p_i^{ell} \end{bmatrix} + (1 - \lambda) \begin{bmatrix} m_1^{ell} \\ m_2^{ell} \\ \vdots \\ m_i^{ell} \end{bmatrix},$$

com $m \in \mathcal{M}$ e $p \in \mathcal{P}$, em que $\mathcal{M}$ e $\mathcal{P}$ possuem a mesma cardinalidade e são partições de $\mathcal{S}$, o que, por sua vez, são os indivíduos selecionados para *crossover*, como já descrito no Capítulo 3.

Como a *combinação convexa* demonstrada acima foi executada matricialmente, não é necessário a utilização do mesmo trecho de código i vezes, uma objetivando a criação de cada “filho”, contribuindo para atenuar o tempo execução do algoritmo evitando, novamente, repetições desnecessárias.

Claro que, para ambos os casos, as dimensões das matrizes devem ser respeitadas.

Antes de inserirmos os indivíduos “filhos” na população, os mesmos devem ser “desvetorizados”, portanto

$$\zeta(g_f^{ell}) = (f_1^{ell}, f_2^{ell}, \dots, f_i^{ell}). \quad (\text{B.2})$$

Outro ponto considerado refere-se à Eq. (2.7), que desempenha papel fundamental no cálculo da solução analítica do custo, Eq. (2.11), para o controlador atuante no sistema apresentado em (2.1). Ela é o “coração” de todos os algoritmos apresentados neste trabalho, utilizada sempre que um novo controlador (matrizes A_c , B_c , C_c e D_c) for considerado, uma vez

que sua performance ante ao sistema a ser controlado deve ser avaliado. Portanto, otimizá-la contribuirá para o desempenho geral do algoritmo. Sabendo que o MATLAB[®] apresenta melhor desempenho lidando com vetores e matrizes, considere a Definição 5):

Definição 5. Considere as matrizes $A^{m \times n}$ e $B^{p \times q}$. O *Produto de Kronecker* de A por B , denotado $A \otimes B$, e dado pela matriz $mp \times nq$

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}. \quad (\text{B.3})$$

As propriedades do *Produto de Kronecker* (ABADIR; MAGNUS, 2005) considerando as matrizes $C^{p \times q}$, $D^{n \times m}$, $E^{q \times s}$ e $F^{n \times p}$, são:

1. $(\beta \otimes A) = \beta A = A\beta = (A \otimes \beta)$, onde β é um escalar;
2. $(A \otimes B)' = A' \otimes B'$;
3. $(A \otimes B) + (A \otimes C) = A \otimes (B + C)$;
4. $(A \otimes B)(D \otimes E) = (AD \otimes BE)$;
5. $\text{Tr}\{AD\} = \text{vec}(A')'\text{vec}(D) = \text{vec}(D')'\text{vec}(A)$;
6. $\text{vec}(AFB) = (B' \otimes A)\text{vec}(F)$.

Como observado na Seção 2.1.1, estamos trabalhando com matrizes, portanto, aplicando a propriedade 6 da Def. 5 na Eq. (2.7):

$$\begin{aligned} \mathcal{Y} &= \mathcal{L}(\mathcal{Y}) + \mathcal{B} \\ \text{vec}(\mathcal{Y}) &= \text{vec}(\mathcal{L}(\mathcal{Y})) + \text{vec}(\mathcal{B}) \\ &= \text{vec}\left(\sum_{j=1}^N p_{ji} A_{cl,j} Y_j A_{cl,j}^T\right) + \text{vec}(\mathcal{B}) \\ &= \text{vec}\left(\sum_{j=1}^N p_{ji} (A_{cl,j} \otimes A_{cl,j})\right) \text{vec}(Y_j) + \text{vec}(\mathcal{B}) \\ &= \begin{bmatrix} \sum_{j=1}^N p_{j1} (A_{cl,j} \otimes A_{cl,j}) \text{vec}(Y_j) + \text{vec}(B_1) \\ \sum_{j=1}^N p_{j2} (A_{cl,j} \otimes A_{cl,j}) \text{vec}(Y_j) + \text{vec}(B_2) \\ \vdots \\ \sum_{j=1}^N p_{jN} (A_{cl,j} \otimes A_{cl,j}) \text{vec}(Y_j) + \text{vec}(B_N) \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} p_{11} (A_{cl,1} \otimes A_{cl,1}) \cdots p_{N1} (A_{cl,N} \otimes A_{cl,N}) \\ \vdots \\ p_{1N} (A_{cl,1} \otimes A_{cl,1}) \cdots p_{NN} (A_{cl,N} \otimes A_{cl,N}) \end{bmatrix}}_M \times \underbrace{\begin{bmatrix} \text{vec}(Y_1) \\ \vdots \\ \text{vec}(Y_N) \end{bmatrix}}_Y + \underbrace{\begin{bmatrix} \text{vec}(B_1) \\ \vdots \\ \text{vec}(B_N) \end{bmatrix}}_{\mathcal{B}_i}. \end{aligned} \quad (\text{B.4})$$

Após a demonstração acima, temos

$$\text{vec}(\mathcal{Y}) = (M \times Y) + B_i, \quad (\text{B.5})$$

o que é semelhante ao sistema linear

$$Ax = b. \quad (\text{B.6})$$

Utilizando a função *linsolve* no MATLAB[®], podemos resolver como segue:

$$Y = \text{linsolve}((I - M), B_i), \quad (\text{B.7})$$

onde I é a matriz identidade com as mesmas dimensões da matriz M . Como o resultado proveniente da Eq. (B.7) não depende de recursões, diminuámos o tempo de execução do algoritmo, entretanto, tal equação possui um revés: as vezes, retorna resultados inválidos, como resultados negativos. Isso dá-se pelo comportamento da função e/ou as matrizes podem ser mal condicionadas (este tipo de caso não foi abordado e estudado buscando soluções); caso isto ocorra, utilizaremos a Eq. (2.7) pois, devido a seu comportamento, os resultados sempre serão positivos (note que $\mathcal{Y}$ na Eq. (2.7) faz uso de recursão).

Note também que o resultado de Y em (B.7) está na forma vetorizada. Como a Eq. (2.11) faz uso de $Y_1, \dots, Y_N$ isoladamente em não na forma vetorizada, considere o operador Ω

$$\Omega(\text{vec}(\mathcal{Y})) = (Y_1, Y_2, \dots, Y_N), \quad (\text{B.8})$$

que tem com como objetivo “desvetorizar” o conjunto de matrizes em $\text{vec}(\mathcal{Y})$.

Por último, *variáveis globais* e *variáveis locais*. Em computação, existem dois tipos principais de variáveis, as *variáveis globais* e as *variáveis locais*. *Variáveis locais*, são aquelas

1. declaradas dentro de uma função;
2. não podem ser utilizadas ou modificadas por outras funções durante a execução do algoritmo e
3. só existem enquanto a função onde foi declarada estiver sendo executada.

Em contrapartida, *variáveis globais*

1. devem ser declaradas **antes** de todas as funções do algoritmo;
2. são acessíveis por qualquer função durante toda a execução do código e podem ter seu valor modificado por estas funções e
3. existem durante toda a execução do algoritmo.

Neste trabalho, algumas variáveis foram declaradas *globais* pois tais variáveis mantêm-se as mesmas (não tem seu valor alterado) durante toda a execução do algoritmo para toda a instância e são utilizadas em vários trechos do código e não há possibilidade que seus valores sejam alterados durante a execução do algoritmo e nos levarem a um resultado final inválido. Portanto, não faz-se necessário passarmos tais variáveis como parâmetros para as funções que fazem seu uso, ocasionando, assim, na redução do uso de memória computacional.

GERADOR DE SLSMS

O gerador de SLSM “original” gera sistemas na forma de

$$\begin{cases} x(t+1) = A(\theta(t))x(t) + B(\theta(t))u(t) + B_w(\theta(t))w(t), \\ y(t) = x(t)'Q(\theta(t))x(t) + u(t)'R(\theta(t))u(t), \end{cases} \quad (\text{C.1})$$

e as alterações visaram atender sistemas na forma de

$$\begin{cases} x(t+1) = A(\theta(t))x(t) + B(\theta(t))u(t) + B_w(\theta(t))w(t), \\ y(t) = C_y(\theta(t))x(t) + D_w(\theta(t))w(t), \\ z(t) = C_z(\theta(t))x(t) + D_u(\theta(t))u(t). \end{cases} \quad (\text{C.2})$$

Portanto, a primeira mudança foi identificarmos as matrizes equivalentes, ou seja, $C_z = Q$ e $D_u = R$; tal mudança foi efetuada no código. Em seguida, foi necessário criarmos as matrizes B_w e D_w e tais matrizes foram criadas utilizando os mesmos conceitos empregados para a criação da matriz B (somente B_w aparece no código “original”); o mesmo deu-se para a criação de C_y , que segue os moldes de C_z .

Algumas matrizes tiveram suas dimensões alteradas para se atender ao sistema estudado neste trabalho, como mostra o código-fonte 2.

Código-fonte 2 – Criação das matrizes B_w e D_w no MATLAB[®]

```
1: for k=1:n_theta
2:     Bw(:,:,k) = [B_Bw(:,:,k) zeros(size(B_Bw(:,:,k),1),size(
        B_Bw(:,:,k),2))];
3:     Dw(:,:,k) = [zeros(size(B_Dw(:,:,k),1),size(B_Dw(:,:,k),2))
        B_Dw(:,:,k)];
4: end
```

Funcionalidades também foram adicionadas ao código, como uma função para criação de matrizes periódicas, como exibido no código-fonte 3.

Código-fonte 3 – Código para a criação de matrizes periódicas

```

1: function P = periodico(n_theta, periodo)
2:
3:     -----
4:     --> Nesse código, toda linha ou trecho inicializado com
        '-->' sinaliza um comentário, com o intuito de facilitar o
        entendimento de um possível futuro leitor.
5:     -----
6:
7:     --> "n_theta" dimensão da matriz.
8:     --> "periodo" periodo da matriz.
9:
10:    --> Aqui temos alguns desafios a ser considerados. O
        primeiro deles é com relação a divisão dos estados entre os
        periodos. Ex: Para 7 estados e periodo 4 podemos ter várias
        divisões possíveis, como 1 -> 3 -> 2 -> 1, 2 -> 2 -> 2 -> 1
        ou 3 -> 2 -> 1 -> 1, onde a numeração corresponde ao nú
        mero de estados em cada período. Para tal divisão usaremos
        a lógica explicada abaixo:
11:
12:    --> Vetor com todos os estados disponíveis embaralhados
        aleatoriamente. O comando "randperm" gera a quantidade de
        elementos (estados) necessários e os embaralha.
13:    vetNumeroEstados = randperm(n_theta);
14:
15:    -> Variável que corresponde ao índice na lista que
        armazenará os vetores com a quantidade de estados para cada
        período.
16:    indiceLista = 1;
17:
18:    --> Número total de elementos / Número de casas (períodos)
        disponíveis. Em um primeiro momento, "nte" e "ncd" serão a
        quantidade total de estados e o periodo da cadeia de Markov
        respectivamente. Posteriormente, tais valores serão
        alterados conforme os laço "while" abaixo for sendo
        executado.
19:    nte = n_theta;
20:    ncd = periodo;

```

```

21:
22:     --> PARTE 1 - Distribui os estados entre os períodos.
23:     while (ncd ~= 1)
24:
25:         --> Tal variável armazenará as possibilidades de
quantidade de estados que poderão compor determinado período
.
26:         x = (nte - ncd) + 1;
27:
28:         --> Vetor com a quantidade máxima de estados que poderã
o ser escolhidos para determinado período.
29:         vetQtdMaxEstados = [1:x];
30:
31:         --> Escolhe a quantidade estados escolhidos.
32:         qtdEstados = vetQtdMaxEstados(randi([1,numel(
vetQtdMaxEstados)]));
33:
34:         -- > Aloca memória correspondente com a quantidade de
estados que irão compor tal período.
35:         vetAux = zeros(size(1:qtdEstados));
36:
37:         --> O laço abaixo é responsável por escolher aleató
riamente dentre os estados disponíveis, os estados que irão
compor o período.
38:         for i=1:qtdEstados
39:             vetAux(i) = vetNumeroEstados(i);
40:         end
41:         vetNumeroEstados(1:qtdEstados) = []; --> Deleta os
estados escolhidos
42:
43:         listaPeriodos{indiceLista} = vetAux; --> A lista recebe
o conjunto de estados no período
44:         indiceLista = indiceLista + 1;
45:         clear vetAux;
46:
47:         nte = nte - qtdEstados;
48:         ncd = ncd - 1;
49:
50:         clear qtdEstados vetQtdMaxEstados;
51:     end
52:
53:     --> Quando "ncd" for igual a 1, sairemos do laço e

```

```
saberemos que ainda nos resta uma quantidade "x" de estados
mas apenas um período, portanto:
54:     listaPeriodos{indiceLista} = vetNumeroEstados;
55:     clear indiceLista;
56:
57:     --> PARTE 2 - Montagem da matriz de probabilidade
58:     P = zeros(n_theta,n_theta);
59:
60:     --> Inicializar variáveis.
61:     indice1 = 1;
62:     indice2 = indice1 + 1;
63:     for k=1:periodo
64:         vet1 = listaPeriodos{indice1};
65:         vet2 = listaPeriodos{indice2};
66:         for i=1:size(vet1,2)
67:             --> Probabilidade de transição de estados. Tal cálculo
encontra-se dentro do primeiro "for" para garantirmos que as
        probabilidades para mudança de cada estado "i" para cada
        estado "j" sejam diferentes.
68:             prob = rand(size(vet2,2),1);
69:             prob = prob / norm(prob,1);
70:             for j=1:size(vet2,2)
71:                 P(vet1(i),vet2(j)) = prob(j);
72:             end
73:             clear prob;
74:         end
75:         --> Aqui redefinimos os "indices", ou seja, apontamos
para um novo conjunto de estados na lista, tomando cuidado
para garantirmos que o último conjunto reaponte para o
        primeiro.
76:         indice1 = indice1 + 1;
77:
78:         --> Se a variável "indice1" estiver apontada para o ú
ltimo conjunto de estados, faremos a variável "indice2"
        apontar para o primeiro conjunto de estados.
79:         if(indice1 == periodo)
80:             indice2 = 1;
81:         else --> Caso contrário, somente "andamos" para frente
com os apontamentos
82:             indice2 = indice1 + 1;
83:         end
84:     end
```

85: end

Finalizando, o gerador de SLSM criava sistemas mas não resolvia as EARA para encontrar as matrizes A_c , B_c , C_c e D_c atuantes no controlador (e neste caso, um conjunto de matrizes para cada estado de θ), entretanto, dispomos do código que provê tal resolução pois este foi desenvolvido para simulação do quadrirotor, com resultados apresentados em (DRAGAN; COSTA, 2016). Portanto, transformamos este código em uma função efetuando as alterações para atender qualquer SLSM, já que todos terão suas particularidades como diferentes matrizes de transição, diferentes distribuições iniciais, diferentes *Cesàro limite*, etc; após, o acoplamos ao gerador de SLSM, logo, toda vez que um novo SLSM é gerado, suas matrizes são enviadas para esta nova função que resolverá as EARA.

Os dados que compõem os SLSMs e as EARAs, servirão como entrada para todos os métodos avaliados neste trabalho.

Observação 16. O gerador de SLSM cria sistemas com θ observável.

Observação 17. O gerador de SLSM cria *matrizes de probabilidade de transição* com estados transientes, recorrentes, absorventes e periódicos.