Programação Genética Linear com Inspiração Quântica

Douglas Mota Dias[1], **Marco Aurélio C. Pacheco**[1]

[1]Departamento de Engenharia Elétrica
Pontificia Universidade Católica do Rio Janeiro (PUC-Rio)
Rio de Janeiro – RJ – Brasil

{douglasm,marco}@ele.puc-rio.br

Abstract. *Quantum-inspired evolutionary algorithms (QIEAs) exploit principles of quantum mechanics to improve the performance of classical evolutionary algorithms. This paper presents a new model of QIEA ("Quantum-Inspired Linear Genetic Programming" – QILGP) to evolve machine code programs. Symbolic regression problems and the current more efficient model to evolve machine code (AIMGP) are used in comparative tests. QILGP outperforms it, obtaining better solutions with fewer evaluations, parameters and operators. One conclude that the quantum inspiration can be a competitive approach to evolve programs more efficiently.*

Resumo. *Algoritmos evolutivos com inspiração quântica (AEIQ) aproveitam princípios da mecânica quântica para melhorar o desempenho de algoritmos evolutivos clássicos. Este artigo apresenta um novo modelo de AEIQ ("Programação Genética Linear com Inspiração Quântica" – PGLIQ) para evoluir programas em código de máquina. Nos testes comparativos são utilizados problemas de regressão simbólica e o modelo atual mais eficiente na evolução de código de máquina (AIMGP). A PGLIQ apresenta desempenho superior, obtendo melhores soluções com menos avaliações, parâmetros e operadores. Conclui-se que a inspiração quântica pode ser uma abordagem competitiva para se evoluir programas mais eficientemente.*

1. Introdução

Em sua definição mais simples, a Programação Genética (PG) é uma técnica de Computação Evolutiva (CE) que permite a computadores resolverem problemas sem que sejam explicitamente programados para tal [Koza 1992]. Ou seja, a PG parte de uma declaração de alto nível sobre "o que se necessita ser feito" e cria automaticamente um programa de computador para resolver o problema.

Diferentes tipos de PG existentes evoluem diferentes tipos de programas e os representam por diferentes tipos de estruturas. Este artigo se baseia na PG Linear (PGL), que possui esta denominação por representar seus programas em estruturas lineares (listas). A PGL é utilizada para a evolução de programas em linguagens imperativas (p.ex. C [Brameier and Banzhaf 2007] e linguagem de montagem [Dias et al. 2006]) ou em código de máquina.

Desde sua concepção e formalização [Koza 1992], a PG vem sendo utilizada para resolver uma ampla variedade de problemas práticos, produzindo resultados competitivos

com os de especialistas humanos, incluindo novas descobertas científicas e invenções patenteáveis [Koza 2010].

Entretanto, os Algoritmos Evolutivos (AE) apresentam características que podem prejudicar seu desempenho. Como estes algoritmos necessitam avaliar diversas vezes a qualidade das soluções, problemas onde cada avaliação seja computacionalmente custosa podem tornar proibitivo seu uso. Neste sentido, os AE com inspiração quântica (AEIQ) representam um dos mais recentes avanços na CE. Estes algoritmos se baseiam em ideias inspiradas na mecânica quântica, apresentando melhor desempenho em diversos tipos de aplicações. Mais especificamente, um AEIQ usando representação binária foi utilizado com sucesso em problemas de otimização combinatória em [Han and Kim 2002], apresentando resultados superiores em relação aos AGs convencionais, por necessitar de menos avaliações para obter resultados melhores. Em [Abs da Cruz et al. 2010] é apresentado um AEIQ para otimização numérica, inspirado no princípio de múltiplos universos da física quântica e com representação de números reais, que apresenta um tempo de convergência menor para problemas *benchmark* quando comparado com algoritmos convencionais.

Sendo assim, o principal objetivo deste artigo é apresentar um novo modelo de AEIQ, inspirado no conceito de superposição de estados da mecânica quântica, para a evolução de programas, denominado "Programação Genética Linear com Inspiração Quântica" (PGLIQ). Desta forma, também se pretende investigar a capacidade deste paradigma de melhorar o desempenho da PG, mais especificamente, em problemas de regressão simbólica. Uma versão preliminar e simplificada deste modelo foi proposta pelos autores em [Dias and Pacheco 2009], apenas como prova de conceito.

Optou-se pela evolução de programas em código de máquina para o modelo por esta ser a abordagem mais rápida existente quanto à velocidade total de execução [Poli et al. 2008], uma vez que dispensa etapas como interpretação ou compilação dos indivíduos. O modelo de PGL convencional mais bem sucedido na evolução de programas em código de máquina denomina-se AIMGP (*Automatic Induction of Machine Code by Genetic Programming*) [Nordin 1998] e, portanto, é utilizado aqui como referência para fins de comparação.

2. Computação Quântica e Algoritmos com Inspiração Quântica

Em um computador clássico, o *bit* é a menor unidade de informação, podendo assumir os valores 0 ou 1. Em um computador quântico, a unidade de informação básica, chamada de *qubit*, pode assumir os estados $|0\rangle$, $|1\rangle$ ou uma superposição dos dois estados. Esta superposição de dois estados é uma combinação linear dos estados $|0\rangle$ e $|1\rangle$ e pode ser representada por $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, onde $|\psi\rangle$ é o estado do *qubit* e $|\alpha|^2$ e $|\beta|^2$ são as probabilidades de que os estados 0 ou 1 acarretem da observação (ou medição) do *qubit*, sendo $|\alpha|^2 + |\beta|^2 = 1$, com α e β números complexos.

A superposição de estados oferece aos computadores quânticos um grau de paralelismo incomparável que, se devidamente explorado, permite que estes computadores realizem tarefas impraticáveis em computadores clássicos. Um exemplo é a fatoração inteira de grandes números, onde se deseja encontrar os dois números primos cujo produto seja igual a um número fornecido. Esta tarefa pode levar milhões de anos para ser resolvida em computadores clássicos usando os algoritmos mais sofisticados conhecidos. Em

um computador quântico, no entanto, a mesma tarefa levaria apenas alguns segundos para ser concluída [Lavor et al. 2003].

A dinâmica de um sistema quântico deve levar o sistema de um estado para outro de uma forma que preserve a norma. Por sua vez, qualquer transformação unitária de um espaço de estados quânticos é uma transformação quântica legítima e vice versa. Ou seja, pode-se encarar as transformações como sendo rotações de um vetor complexo. Na computação quântica, os conjuntos de transformações primitivas de estados quânticos são denominados "portas quânticas". Para computadores clássicos existem conjuntos de portas lógicas que são universais, no sentido que qualquer computação clássica pode ser efetuada usando-se uma combinação dessas portas. De forma similar, existem conjuntos universais de portas quânticas para a computação quântica.

De fato, a maioria das abordagens da computação quântica emprega *qubits* codificados em sistemas quânticos de dois níveis. No entanto, os sistemas que codificam informação quântica normalmente têm uma estrutura física muito mais complexa, com diversos graus de liberdade diretamente acessíveis (p.ex. átomos, fótons etc.). Recentemente, começou-se a considerar sistemas quânticos multiníveis, que utilizam o *qudit* como unidade de informação, o qual pode assumir qualquer um de d valores, ou uma superposição de d estados [Lanyon et al. 2008].

Embora a computação quântica ofereça uma boa promessa em termos de capacidade de processamento, atualmente dois problemas a impedem que se torne uma ferramenta útil: a dificuldade de se implementar um computador quântico e a dificuldade de se criar algoritmos que tirem proveito.

No entanto, em [Moore and Narayanan 1995] foi proposta uma nova abordagem. Ao invés de se buscar desenvolver algoritmos para computadores quânticos ou de se tentar viabilizar o uso destes, a ideia de "computação com inspiração quântica" é apresentada. O objetivo desta nova abordagem é criar algoritmos clássicos, ou seja, executáveis em computadores clássicos, que tirem proveito de paradigmas da mecânica quântica, de forma a melhorar o desempenho dos mesmos na resolução de problemas. Ou seja, assim como os AEIQ já citados [Moore and Narayanan 1995, Han and Kim 2002, Abs da Cruz et al. 2010], o modelo aqui proposto se inspira em um fenômeno da mecânica quântica, neste caso, a superposição de estados quânticos, para melhorar o desempenho de um algoritmo clássico: a Programação Genética.

3. Programação Genética Linear com Inspiração Quântica

3.1. Plataforma

A PGLIQ evolui programas para a plataforma Intel x86 [Int 1997], utilizando algumas instruções da sua Unidade de Ponto Flutuante (*Floating Point Unit* – FPU), as quais operam com dados da memória principal (m) e/ou dos 8 registradores da FPU ($ST(i)$, onde $i = 0, 1, \ldots, 7$). Neste modelo, são utilizadas instruções para adição, subtração, multiplicação, divisão e transferência de dados, assim como instruções aritméticas e trigonométricas. O modelo representa internamente estas instruções por um "*token* de função" (TF), que pode assumir valores inteiros de 0 a $(f - 1)$, de forma a representar de maneira única cada uma das f funções da PGLIQ, as quais compõem o conjunto de funções para os estudos de caso deste artigo. A tabela 1 mostra as instruções citadas, suas respectivas

funcionalidades, o respectivo argumento de cada uma, caso haja, e o valor do *token* destas instruções.

Tabela 1. Descrição das instruções.

Instrução	Descrição	Argumento	*token*
NOP	Nenhuma operação	-	0
FADD m	$ST(0) \leftarrow ST(0) + m$	m	1
FADD ST(0), ST(i)	$ST(0) \leftarrow ST(0) + ST(i)$	i	2
FADD ST(i), ST(0)	$ST(i) \leftarrow ST(i) + ST(0)$	i	3
FSUB m	$ST(0) \leftarrow ST(0) - m$	m	4
FSUB ST(0), ST(i)	$ST(0) \leftarrow ST(0) - ST(i)$	i	5
FSUB ST(i), ST(0)	$ST(i) \leftarrow ST(i) - ST(0)$	i	6
FMUL m	$ST(0) \leftarrow ST(0) \times m$	m	7
FMUL ST(0), ST(i)	$ST(0) \leftarrow ST(0) \times ST(i)$	i	8
FMUL ST(i), ST(0)	$ST(i) \leftarrow ST(i) \times ST(0)$	i	9
FXCH ST(i)	$ST(0) \leftrightarrows ST(i)$ (troca conteúdos)	i	A
FDIV m	$ST(0) \leftarrow ST(0) \div m$	m	B
FDIV ST(0), ST(i)	$ST(0) \leftarrow ST(0) \div ST(i)$	i	C
FDIV ST(i), ST(0)	$ST(i) \leftarrow ST(i) \div ST(0)$	i	D
FABS	$ST(0) \leftarrow \lvert ST(0) \rvert$	-	E
FSQRT	$ST(0) \leftarrow \sqrt{ST(0)}$	-	F
FSIN	$ST(0) \leftarrow \operatorname{sen} ST(0)$	-	G
FCOS	$ST(0) \leftarrow \cos ST(0)$	-	H

Um programa em código de máquina evoluído pela PGLIQ representa uma solução e lê os dados de entrada na memória principal, os quais são compostos pelas variáveis de entrada do problema e, opcionalmente, constantes fornecidas pelo usuário. Estas entradas podem ser representadas por um vetor, como por exemplo:

$$I = (V[0], V[1], 1, 2, 3), \tag{1}$$

onde $V[0]$ e $V[1]$ contêm os valores das duas entradas do problema exemplificado, e onde 1, 2 e 3 são os valores das três constantes. Basicamente, o modelo AIMGP é implementado da mesma forma pelo *software* comercial Discipulus™ [Francone 2010], que é utilizado, portanto, para a execução dos experimentos comparativos deste artigo.

3.2. Representação

A PGLIQ é baseada nas seguintes entidades, da seguinte forma: o cromossomo de um "indivíduo quântico", que representa a superposição de todos os programas possíveis no espaço de busca predefinido, é observado para gerar o cromossomo de um "indivíduo clássico" (IC), a partir do qual um programa em código de máquina é finalmente gerado. Uma vez que as instruções utilizadas têm apenas um ou nenhum argumento, todas as funções têm apenas um terminal que é representado por um "*token* de terminal" (TT). No caso de uma função sem terminal, o seu *token* de terminal correspondente é ignorado pelo modelo.

Um programa é internamente representado pelo cromossomo do IC, representado por uma estrutura com $(L \times 2)$ *tokens*, conforme mostrado na porção direita da figura 1,

onde: L é o comprimento máximo do programa (em número de instruções); cada linha representa uma instrução i ($1 \leq i \leq L$) e é definida como um "gene"; a coluna da esquerda contém os valores dos *tokens* de função (TF_i) e a da direita, os valores dos seus respectivos *tokens* de terminal (TT_i). A ordem de execução do programa é da primeira até a última linha.

Apesar do cromossomo do IC possuir comprimento fixo, o programa efetivo que ele representa possui comprimento variável, assim como na PGL e na AIMGP. No caso da PGLIQ, esta variação é obtida pela inclusão da instrução `NOP` no conjunto de instruções. Ocorre que o processo de geração de código ignora qualquer gene no qual um `NOP` esteja presente, ou seja, todo gene cujo valor do seu *token* de função valha zero.

O modelo aqui apresentado é inspirado em sistemas quânticos multiníveis [Lanyon et al. 2008]. Portanto, conforme mostrado na seção 2, a unidade básica de informação adotada pela PGLIQ é o *qudit*. Esta informação pode ser descrita por um vetor de estado em um sistema mecânico quântico de d níveis, o qual equivale a um espaço vetorial d-dimensional, onde d é o número de estados nos quais o *qudit* pode ser medido. Isto é, d representa a cardinalidade do *token* que terá seu valor determinado pela observação do seu respectivo *qudit*. O estado de um *qudit* é uma superposição linear dos d estados e pode ser representado por $|\psi\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle$, onde $|\alpha_i|^2$ é a probabilidade de que o *qudit* seja observado no estado i. A normalização unitária garante: $\sum_{i=0}^{d-1} |\alpha_i|^2 = 1$.

Tome-se como exemplo um conjunto de funções composto pelas 11 primeiras instruções da tabela 1. Ao se observar um *qudit* de função cujo estado é dado pela equação 2, tal que:

$$|\psi\rangle = \frac{1}{\sqrt{5}} |0\rangle + \frac{1}{\sqrt{4}} |1\rangle + \frac{1}{\sqrt{10}} |2\rangle + \cdots + \frac{1}{\sqrt{8}} |\mathrm{A}\rangle , \tag{2}$$

a probabilidade de se medir a instrução `NOP` (estado '0') é $(1/5)^2 = 0,200$, para `FADD m` (estado '1') é $(1/4)^2 = 0,250$, para `FADD ST(i)` (estado '2') é $(1/10)^2 = 0,100$ e assim por diante, até que finalmente a probabilidade de se medir a instrução `FXCH ST(i)` (estado 'A') é $(1/5)^2 = 0,125$.

O cromossomo do IQ, denominado "cromossomo quântico", é representado por uma lista de estruturas denominadas "genes quânticos". Um gene quântico é composto por um *qudit* de função (QF), que representa a superposição de todas as funções predefinidas pelo conjunto de funções. Também possui dois *qudits* de terminal (QT), uma vez que as funções podem utilizar dois tipos diferentes de terminais. Um dos *qudits* de terminal representa os registradores (QT_{Reg}) e o outro, as posições de memória (QT_{Mem}), sendo que estes registradores e posições de memória pertencem a um conjunto de terminais predefinido. Por exemplo, a instrução `FADD ST(0),ST(i)` utiliza um *qudit* de terminal que representa a superposição dos índices i dos registradores $ST(i)$, enquanto que o *qudit* de terminal da instrução `FADD m` representa a superposição das posições m da memória. Como cada gene quântico é observado para gerar um gene do IC (uma instrução completa), o cromossomos de ambos os indivíduos, quântico (IQ) e clássico (IC), possuem o mesmo comprimento, conforme mostra a figura 1.

Pode-se ilustrar o processo de criação de um gene pela observação de um gene quântico, a partir de um exemplo baseado na equação 2 e no vetor de caso $I = (V[0], V[1], 1, 2, 3)$, definido em (1), conforme os três seguintes passos básicos:

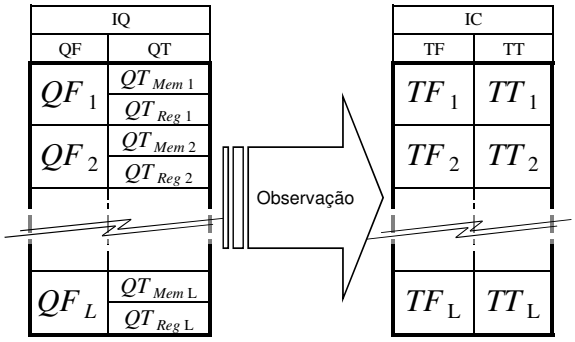

Figura 1. Criação de um indivíduo clássico (IC) pela observação de um indivíduo quântico (IQ).

1. O *qudit* de função (QF) é observado e o valor resultante (p.ex. 7) é atribuído ao *token* de função (TF) do gene.
2. O valor do *token* de função determina o *qudit* de terminal a ser observado, uma vez que cada instrução demanda um tipo diferente de terminal dentre dois: registrador ou memória.
3. O *qudit* de terminal (QT) determinado pelo valor do *token* de função é observado e o valor resultante (p.ex. 1) é atribuído ao *token* de terminal (TT) do gene.

Portanto, neste exemplo, a instrução observada é `FMUL V[1]`, uma vez que '7' é o valor do *token* de função para esta instrução (tabela 1) e '1' é o valor do *token* de terminal que representa $V[1]$ no vetor de entrada I, definido em (1). Neste caso, portanto, os valores de um *token* de terminal de '0' a '4' indicam que o argumento é $V[0]$, $V[1]$, 1, 2 ou 3, respectivamente. Entretanto, se o *token* de terminal for de uma instrução cujo argumento seja um registrador, o valor do *token* de terminal indica diretamente qual dos 8 registradores é o argumento, sendo que apenas os 4 primeiros são utilizados neste exemplo: $ST(0)$ a $ST(3)$.

Cada programa em código de máquina evoluído pela PGLIQ, assim como na AIMGP, é composto pelos três seguintes segmentos:

- Cabeçalho (*header*): inicia a FPU e carrega o valor zero em todos os registradores.
- Corpo: é o código evoluído em si (único segmento afetado pela evolução).
- Rodapé (*footer*): transfere o conteúdo de $ST(0)$ para $V[0]$ (saída predefinida do programa) e reinicia a FPU. Em seguida, executa a instrução de retorno para encerrar a execução do programa e retornar ao fluxo principal do algoritmo evolutivo.

3.3. Avaliação de um Indivíduo Clássico

O processo se inicia com a geração de um programa em código de máquina a partir do IC a ser avaliado, onde seu cromossomo é percorrido sequencialmente, gene por gene, *token* por *token* (de função e de terminal), gerando serialmente o código de máquina do corpo do programa relativo àquele IC, cuja avaliação compreende a execução do seu código de máquina para todos os casos de aptidão do problema (todos os dados de um conjunto de avaliação). Quando instruções `FDIV` incorrem em divisão por zero ou instruções `FSQRT` incorrem em raiz quadrada de número negativo, o que é detetado quando $ST[0]$ apresenta um valor inválido, atribui-se o valor nulo a $V[0]$ como resultado da avaliação daquele caso de aptidão (mesma abordagem adotada pela AIMGP).

3.4. Operador Quântico

O operador quântico aqui proposto atua diretamente nas probabilidades p_i de um *qudit*, satisfazendo a condição de normalização: $\sum_{i=0}^{d-1} |\alpha_i|^2 = 1$, onde d é a cardinalidade do *qudit* e $|\alpha_i|^2 = p_i$. Sendo assim, este operador, aqui denominado "operador P", representa a funcionalidade de uma porta quântica, efetuando rotações no vetor que representa o estado $|\psi\rangle$ de um *qudit* em um espaço vetorial d-dimensional. O operador P funciona em dois passos básicos. Primeiramente, o operador incrementa uma dada probabilidade do *qudit*, da seguinte maneira: $p_i \leftarrow p_i + s(1 - p_i)$, onde s é um parâmetro denominado "tamanho de passo", que pode assumir qualquer valor real entre 0 e 1. O segundo passo é o ajuste dos valores de todas as probabilidades do *qudit* de forma a satisfazer a condição de normalização. Ou seja, o valor da probabilidade p_i de um *qudit* é incrementado pelo operador, de um valor diretamente proporcional ao tamanho de passo s. Sendo assim, o que diferencia um indivíduo quântico de outro são os valores das probabilidades de seus *qudits*, os quais variam distintamente ao longo da evolução.

3.5. Estrutura e Algoritmo Evolutivo

O diagrama na figura 2 ilustra a estrutura do modelo e seu funcionamento básico. A PGLIQ possui uma população composta por uma população quântica e outra clássica, ambas possuindo M indivíduos. Também possui M ICs auxiliares C_i^{obs}, que resultam das observações dos IQs Q_i, onde $1 \leq i \leq M$. No exemplo da figura 2, $M = 4$.

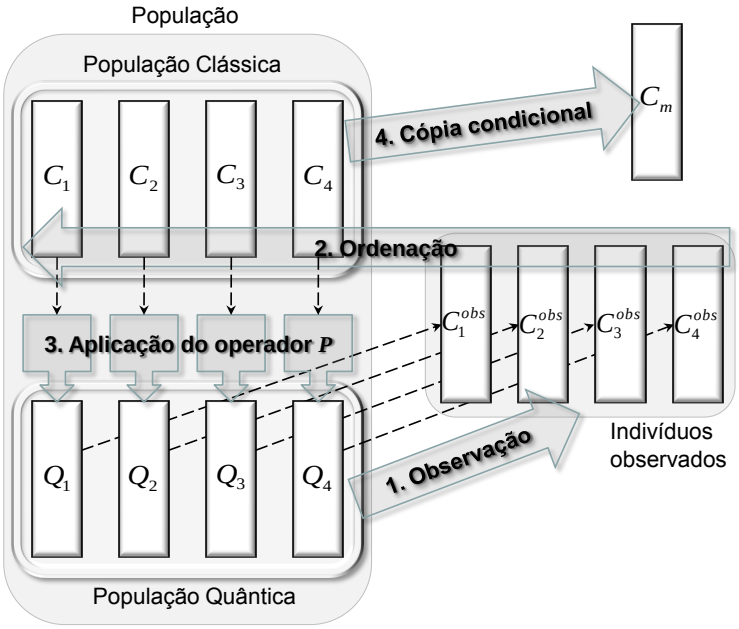

Figura 2. Diagrama descritivo básico do modelo PGLIQ.

A figura 2 também enumera as 4 etapas básicas que caracterizam uma "geração" do modelo. Na etapa 1, cada um dos M IQs é observado uma vez, resultando nos M ICs C_i^{obs}. Na etapa 2, os indivíduos da população clássica e os indivíduos observados são ordenados conjuntamente quanto às suas avaliações. Como resultado, os M melhores ICs dentre os $2M$ avaliados são mantidos na população clássica, ordenados do melhor para o pior, de C_1 para C_M. Na etapa 3, O operador P é aplicado a cada indivíduo quântico Q_i, tomando como referência seus indivíduos C_i correspondentes na população clássica. Na

aplicação do operador em um IQ, as probabilidades dos *qudits*, de função e de terminal, de cada um dos seus L genes GQ_g são ajustadas segundo seus respectivos genes GC_g de um IC (vide seção 3.4), onde $1 \leq g \leq L$. É nesta etapa que ocorre efetivamente a evolução, uma vez que, a cada nova geração, a aplicação do operador aumenta a probabilidade de que a observação dos IQs gerem ICs mais parecidos com os melhores encontrados até então. E na etapa 4, caso algum dos indivíduos da população clássica, avaliados na geração atual, seja melhor que o melhor IC avaliado até então, uma cópia do mesmo é armazenada em C_m, o melhor IC obtido até então.

Cada IQ deve ser iniciado antes de uma execução, iniciando-se cada um dos seus L genes quânticos. Cada *qudit* de terminal é iniciado pela atribuição do mesmo valor $1/t$ a todas as suas probabilidades p_i, onde $0 \leq i < t$, e t é o número de terminais representados pelo *qudit*. A iniciação de um *qudit* de função atribui um valor predeterminado $p_{0,0}$ a p_0, de forma a fornecer um controle de comprimento inicial sobre os primeiros programas evoluídos, uma vez que p_0 é a probabilidade de se observar a instrução NOP, ignorada durante a criação de um programa a partir de um IC. Assim, as demais probabilidades p_i $(1 \leq i < f)$ são iniciadas com o mesmo valor $(1 - p_{0,0})/(f - 1)$, onde f é o número de instruções representadas pelo *qudit*.

De forma a evitar uma convergência prematura, quando uma execução atinge o valor predeterminado do parâmetro $gsm_{máx}$ (número máximo de gerações sem melhoria de aptidão), as populações quântica e clássica são reiniciadas. Este processo elimina o conteúdo dos ICs e reinicia as probabilidades de todos os IQs. Em seguida, C_m é copiado para C_1 como uma semente. Finalmente, o operador P é aplicado uma vez a Q_1, tomando C_1 como referência. Entretanto, o valor de s utilizado aqui é diferente, sendo determinado pelo parâmetro s_r (tamanho do passo de reinício), cujo valor é algumas ordens de grandeza maior que o de s. Este último processo pode ser considerado uma "semeadura quântica".

4. Experimentos

Uma vez que os estudos de caso aqui abordados são de regressão simbólica, a base de dados é composta por três conjuntos. O conjunto de treinamento é o que efetivamente conduz a evolução, enquanto que o de validação serve para verificar constantemente a qualidade da generalização das melhores soluções encontradas durante a evolução. O conjunto de teste, que não é apresentado ao algoritmo durante a evolução, é aplicado ao melhor indivíduo obtido em uma execução completa do algoritmo. Cada experimento é composto por 100 execuções dos modelos, de forma a se obter a média das aptidões dos melhores indivíduos por geração, cujo valor é a média obtida por treinamento e validação, assim como as aptidões dos melhores indivíduos dentre as 100 execuções.

A aptidão de um indivíduo é determinada pela computação do seu erro absoluto médio (EAM) sobre os n casos de aptidão, definido como

$$\text{EAM} = \frac{1}{n} \sum_{i=1}^{n} |t_i - V[0]_i|, \tag{3}$$

onde t_i é o valor esperado (real) do caso i e $V[0]_i$ é o valor da saída do indivíduo para o mesmo caso (aproximação).

4.1. Configuração dos Modelos

A configuração da AIMGP adotada nos experimentos é descrita na tabela 2. Os parâmetros recebem seus valores padrão, conforme recomendado pelo manual do Discipulus™ [Francone 2010]. O número de gerações por execução e o tamanho da população foram escolhidos com base em faixas de valores típicos de experimentos com PG [Brameier and Banzhaf 2007, Poli et al. 2008].

Tabela 2. Configuração do modelo AIMGP para os experimentos.

Parâmetro	Valor
Número de gerações (G)	1.000
Tamanho da população (M)	1.000
Número de *demes*	10
Taxa de migração entre *demes*	1%
Comprimento máximo inicial do programa	20
Comprimento máximo do programa	128
Frequência de mutação	95%
– Taxa de mutação de bloco	30%
– Taxa de mutação de instrução	30%
– Taxa de mutação de dados	40%
Frequência de cruzamento	50%
– Taxa de cruzamento homólogo	95%
Número de registradores da FPU	8

Todos os estudos de caso são executados com e sem o uso de 10 *demes* (ou subpopulações) com 100 indivíduos cada (vide [Francone 2010]). Uma vez que a PGLIQ possui apenas uma população, composta por IQs e ICs, pode-se concluir que a comparação de desempenho entre os modelos é mais neutra ao se considerar a AIMGP sem *demes*, ou seja, também com uma única população.

A configuração da PGLIQ é mostrada na tabela 3, onde os parâmetros da segunda à sexta linha recebem seus melhores valores, determinados a partir de experimentos preliminares.

Tabela 3. Configuração do modelo PGLIQ para os experimentos.

Parâmetro	Valor
Número de gerações (G)	300.000 a 350.000
Tamanho da população (M)	6
Número máximo de gerações sem melhoria ($gsm_{máximo}$)	20.000
Probabilidade inicial de NOP ($p_{0,0}$)	0,9
Tamanho do passo de reinício (s_r)	1,0
Tamanho do passo do operador P (s)	0,004
Comprimento máximo do programa	128
Número de registradores da FPU	8

4.2. Resultados

O estudo de caso "Distância", baseado em um problema utilizado como *benchmark* em [Brameier and Banzhaf 2007], requer que a PG gere um programa que calcule a distância

euclideana entre dois pontos a e b no espaço tridimensional (6 variáveis de entrada), conforme mostra a equação:

$$f_{dist}\left(a_x, b_x, a_y, b_y, a_z, b_z\right) = \sqrt{\left(a_x - b_x\right)^2 + \left(a_y - b_y\right)^2 + \left(a_z - b_z\right)^2}. \quad (4)$$

Os conjuntos de dados são compostos por 300 casos de aptidão cada, onde as coordenadas dos pontos ($\in \mathbb{R}^3$) são aleatórios em $[0,\ 1]$. Diferentemente dos demais estudos de caso, este utiliza um conjunto de funções reduzido, composto pelas instruções: `FADD`, `FSUB`, `FMUL`, `FXCH`, `FABS` e `FSQRT`.

A tabela 4 mostra média e desvio padrão σ relativos às aptidões dos melhores indivíduos por geração, para as 100 execuções de cada modelo. Também mostra o número de indivíduos que, dentre as 100 execuções, representam a solução exata do problema ("Acertos"). As últimas colunas à direita ("Variação") mostram os valores percentuais das reduções dos erros médios obtidos pela PGLIQ sobre a AIMGP. A última coluna, em particular, mostra em quantas vezes o número de acertos obtidos pela PGLIQ supera os obtidos pela AIMGP.

Tabela 4. Comparação do desempenho dos modelos para "Distância".

	AIMGP			PGLIQ			Variação (%)		
Demes	Média	σ	Acertos	Média	σ	Acertos	Média	σ	Acertos
Não	0,111	0,049	2	0,068	0,047	17	-38,8	-4,0	750 (8,5 $\times$)
Sim	0,100	0,050	3	0,069	0,048	17	-30,7	-3,8	470 (5,7 $\times$)

O estudo de caso "Composição Química" tem origem em dados laboratoriais de uma composição química (saída) que, apesar do elevado custo de obtenção, são significativamente ruidosos. Portanto, o objetivo da PG é gerar um programa que relacione estes dados a 57 medições de processo de baixo custo, tais como temperaturas, pressões e vazões (entradas). Baseia-se em dados de uma aplicação industrial real da empresa Dow Chemical, tendo sido utilizado em uma competição de Regressão Simbólica no congresso "EvoStar 2010". A base de dados original [Kordon 2010] é composta por um conjunto de treinamento com 747 amostras e um de teste com 319 amostras. Para os experimentos aqui conduzidos, o conjunto de treinamento original foi dividido: 374 amostras para treinamento e 373 para validação.

A tabela 5 compara o desempenho médio dos modelos, e a tabela 6, o desempenho dos melhores indivíduos obtidos por ambos, mostrando a aptidão dos indivíduos com relação aos três conjuntos de dados. As duas últimas colunas à direita mostram que a PGLIQ é superior à AIMGP para validação e teste, apresentando soluções com erros menores em 7,1 a 11,0%.

Tabela 5. Comparação do desempenho médio para "Composição Química".

	AIMGP		PGLIQ		Variação (%)	
Demes	Média	σ	Média	σ	Média	σ
Não	0.189	0.025	0.174	0.021	-8.1	-14.7
Sim	0.185	0.023	0.176	0.021	-5.2	-11.0

Tabela 6. Comparação dos melhores indivíduos para "Composição Química".

	AIMGP			PGLIQ			Variação (%)		
Demes	Trein.	Valid.	Teste	Trein.	Valid.	Teste	Trein.	Valid.	Teste
Não	0,126	0,131	0,140	0,116	0,121	0,124	-7,9	-7,1	-11,0
Sim	0,120	0,139	0,143	0,126	0,126	0,130	1,9	-9,7	-9,2

O estudo de caso "Pontos Quânticos" se baseia em [Singulani et al. 2008], onde são utilizadas redes neurais para inferir o comportamento da altura média de pontos quânticos obtidos (saída) em função de diferentes parâmetros de síntese. Pontos quânticos são estruturas nanométricas que confinam os elétrons nas três dimensões, acarretando na quantização completa dos níveis eletrônicos de energia. O desempenho de diversos dispositivos optoeletrônicos pode ser significativamente beneficiado por esta quantização. Entretanto, seu desempenho depende fortemente do crescimento de estruturas com alta densidade de pontos, com pontos de tamanho pequeno e que apresentem baixa dispersão de tamanho dos pontos. A base de dados com os parâmetros de síntese possui 67 amostras, obtidas por experimentos realizados no laboratório de semicondutores (LABSEM) da PUC-Rio. Os 6 parâmetros (entradas) são: vazão do elemento In no reator, temperatura de crescimento, tempo de deposição, espessura da camada sobre a qual os pontos são crescidos e concentrações de In e Al da camada. Os conjuntos de treinamento, validação e teste contêm, respectivamente, 47, 14 e 6 amostras, como no trabalho de referência.

A tabela 7 compara o desempenho médio dos modelos de PG. A PGLIQ também alcança um desempenho superior ao das redes neurais de [Singulani et al. 2008] para o conjunto de teste. O melhor indivíduo evoluído pela PGLIQ apresenta erro percentual absoluto médio (MAPE) de = 5,2%, contra 8,3% da melhor rede neural, o que representa a redução do erro em 37,3%.

Tabela 7. Comparação do desempenho médio para "Pontos Quânticos" por PG.

	AIMGP		PGLIQ		Variação (%)	
Demes	Média	σ	Média	σ	Média	σ
Não	1.775	0.189	1.359	0.168	-23.5	-11.1
Sim	1.747	0.262	1.402	0.172	-19.8	-34.3

5. Conclusões e Trabalhos Futuros

Este artigo apresentou um novo AEIQ, cujo objetivo é a síntese de programas em código de máquina, o qual obteve desempenho superior ao modelo de referência em testes com problemas de regressão simbólica, ao necessitar de menos avaliações, parâmetros e operadores para alcançar resultados melhores. Estes resultados incentivam seu aperfeiçoamento e futura aplicação em outros tipos de problema. Uma possível melhoria de desempenho pode advir da inclusão de *demes* e de estratégias de migração entre eles (em desenvolvimento e com resultados promissores). É conhecido o benefício dos *demes* sobre convergência prematura e mínimos locais em AE em geral [Poli et al. 2008].

Referências

(1997). *Intel Architecture Software Developer's Manual*. Intel Corporation.

Abs da Cruz, A., Vellasco, M., and Pacheco, M. (2010). Quantum-inspired evolutionary algorithms applied to numerical optimization problems. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–6.

Brameier, M. and Banzhaf, W. (2007). *Linear Genetic Programming*. Number XVI in Genetic and Evolutionary Computation. Springer, Boston, MA, USA.

Dias, D. M. and Pacheco, M. A. C. (2009). Toward a quantum-inspired linear genetic programming model. In Tyrrell, A., editor, *IEEE Congress on Evolutionary Computation*, pages 1691–1698, Trondheim, Norway. IEEE Press.

Dias, D. M., Pacheco, M. A. C., and Amaral, J. F. M. (2006). Automatic synthesis of microcontroller assembly code through linear genetic programming. In Nedjah, N., Abraham, A., and de Macedo Mourelle, L., editors, *Genetic Systems Programming: Theory and Experiences*, volume 13 of *Studies in Computational Intelligence*, pages 195–234. Springer, Germany.

Francone, F. D. (2010). *Discipulus Owner's Manual*. Register Machine Learning Technologies. `https://www.rmltech.com/doclink/Owners%20Manual/Discipulus_Owners_Manual.pdf`.

Han, K.-H. and Kim, J.-H. (2002). Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *Evolutionary Computation, IEEE Transactions on*, 6(6):580–593.

Kordon, A. (2010). Symbolic regression competition – EvoStar Conference. `http://casnew.iti.upv.es/index.php/evocompetitions/105-symregcompetition`. Dow Chemical (USA).

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, J. R. (2010). Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284.

Lanyon, B., Barbieri, M., Almeida, M., Jennewein, T., Ralph, T., Resch, K., Pryde, G., O'Brien, J., Gilchrist, A., and White, A. (2008). Quantum computing using shortcuts through higher dimensions. *Arxiv preprint arXiv:0804.0272*.

Lavor, C., Manssur, L., and Portugal, R. (2003). Shor's algorithm for factoring large integers. *Arxiv preprint quant-ph/0303175*.

Moore, M. and Narayanan, A. (1995). Quantum-inspired computing. *Dept. Comput. Sci., Univ. Exeter*.

Nordin, P. (1998). AIMGP: A formal description. In Koza, J. R., editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA. Stanford University Bookstore.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via `http://lulu.com`. (With contributions by J. R. Koza).

Singulani, A., Vilela Neto, O., Pacheco, M., Vellasco, M., Pires, M., and Souza, P. (2008). Computational intelligence applied to the growth of quantum dots. *Journal of Crystal Growth*, 310(23):5063–5065.