UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
PROGRAMA DE MESTRADO EM ENGENHARIA ELÉTRICA
E DE COMPUTAÇÃO

PROGRAMAÇÃO GENÉTICA APLICADA À PROGRAMAÇÃO DE CONTROLADORES LÓGICO PROGRAMÁVEIS

Mestrando: Marcos Lajovic Carneiro
Orientador: Prof. Dr. Leonardo da Cunha Brito

Goiânia
2009

Marcos Lajovic Carneiro

PROGRAMAÇÃO GENÉTICA APLICADA À PROGRAMAÇÃO DE CONTROLADORES LÓGICO PROGRAMÁVEIS

Dissertação de Mestrado submetida ao Programa de Mestrado em Engenharia Elétrica e de Computação da Escola de Engenharia Elétrica e de Computação da Universidade Federal de Goiás, como parte dos requisitos para obtenção do grau de Mestre em Engenharia Elétrica e de Computação.

Área de concentração: Engenharia de Computação
Orientador: Prof. Dr. Leonardo da Cunha Brito

Escola de Engenharia Elétrica e Computação — UFG

Goiânia
2009

Universidade Federal de Goiás
Escola de Engenharia Elétrica e de Computação
Coordenação do Programa de Pós-Graduação em
Engenharia Elétrica e de Computação

FOLHA DE APROVAÇÃO

"Programação Genética Aplicada à Programação de Controladores Lógico Programáveis"

MARCOS LAJOVIC CARNEIRO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

_____

Prof. Leonardo da Cunha Brito, Dr.
Presidente da Banca

_____

Prof. Sérgio Granato de Araújo, Dr.

_____

Prof. Paulo César Miranda Machado, PhD.

_____

Prof. Maria José Dantas, Dr.

Goiânia, 29 de maio de 2009

Agradecimentos

Agradeço ao professor Sérgio Granato de Araújo e Rodrigo Pinto Lemos por me incentivarem a ingressar na carreira acadêmica e por todas as orientações durante o mestrado.

Agradeço também ao professor Weber Martins que por meio de suas aulas auxiliaram tanto nesse projeto que o considero como co-orientador dessa pesquisa. Suas aulas de metodologia científica e de métodos empíricos foram indispensáveis para que esse projeto fosse finalizado.

Finalmente agradeço ao professor Leonardo Brito pelo voto de confiança que tem feito pelo meu trabalho e pelas oportunidades que tem me oferecido.

“Evolution is Nature’s mistake. Intelligence is its insistence on making the same mistake”.

S. Lem, Golem XIV, 1981

Resumo

Essa pesquisa propõe a aplicação da técnica de inteligência artificial programação genética (PG) para facilitar o trabalho de programação de controladores lógico programáveis (CLP) através da geração automática de programas Ladder e Instruction List. A entrada de dados do sistema de automação é feita de forma leiga a partir de cenários compostos por linhas do tempo. Essas linhas do tempo demonstram graficamente os detalhes do seqüenciamento dos acionamentos das entradas e saídas do CLP permitindo a programação de sistemas que utilizam memória como os inter-travamentos e o uso de temporizadores. Como a PG é altamente dependente dos parâmetros iniciais de simulação, foram feitas milhares de simulações para determinação das melhores formas de configuração dos parâmetros de reprodução por cross-over (cruzamento) e mutação.

Abstract

This research proposes the application of an artificial intelligence technique called genetic programming (GP) to make easier the programming of programmable logical devices (PLC) by the automatic generation of Ladder and Instruction List programs. The system data input can be done by not-specialized people using scenarios composed by time lines. These time lines demonstrate graphically the sequencing details of the PLC input and output permitting the programming of systems that uses memory like inter-locking contacts and the use of timers. Since GP is great dependent of its initial simulation parameters, thousand of simulations have been done to determine the better kind of configuration of cross-over and mutation.

Sumário

Lista de Figuras

Lista de Tabelas

Siglas Utilizadas

CLP	Controlador Lógico Programável
PG	Programação Genética
PLC	Programmable Logic Controller
IEC	International Electrotechnical Commission
NASA	National Aeronautics and Space Administration
DART	Dynamic Analysis and Replanning Tool
NA	Normalmente aberto
NF	Normalmente fechado

1 Introdução

1.1.Problemática e Objetivo

O tema desta pesquisa é a programação de Controladores Lógicos Programáveis (CLP) através de técnicas de Programação Genética (PG). É investigada a geração de programas para CLP através das linguagens Ladder e Instruction List, cuja simulação e função-objetivo são definidas por cenários compostos por linhas do tempo. Estas linhas do tempo indicam a seqüência de acionamentos de botoeiras e sensores e indicam a seqüência de atuação de relés de saída.

Para o desenvolvimento do programa de simulação foram investigados os padrões definidos pela norma IEC 1131-1 (LEWIS, 1995), e os manuais do usuário de CLPs atuais dos fabricantes WEG, Siemens, Allen-Bradley e General Electronics, (WEG, 2006; WEG, 2007; WEG, 2008), (Rockwell, 2004; Rockwell, 1998), (GE, 2007), (Siemens, 2005; Siemens, 2006). A parte do código fonte do programa referente à programação genética implementada foi baseada essencialmente nos trabalhos de (KOZA, 2000) e (BANZHAF et al., 1998).

A seleção de programas pelo uso de seqüências parciais de entrada e saída receberam significativas contribuições dos trabalhos de (MANOVIT; APORNTEWAN; CHONGSTITVATANA, 1998). Foram feitos estudos baseados nos métodos de seleção proporcional (KOZA, 2000), (BANZHAF et al., 1998), (HOLLAND, 1975) e por torneio (ROMERO; MANTOVANI, 2004), esta última demonstrou ser melhor para o problema de convergência para mínimos locais.

Toda a programação foi feita em Pascal no ambiente de programação Delphi. O uso de mais de um programa para realizar diferentes tarefas, e a interação dos mesmos, pode facilitar o trabalho pelo fato de grande parte do código estar pronto e embutido nos programas, porém a correta interação entre eles se torna algo difícil.

A união dos algoritmos de simulação e dos de PG em um mesmo programa permitiu que todas as variáveis que influenciam o processo pudessem ser controladas. A programação em Delphi possui as desvantagens de ser limitado ao sistema operacional Windows e não ser de código aberto. Contudo a utilização da programação orientada a objetos e a disponibilidade de componentes gráficos agiliza o trabalho e torna o Delphi uma ferramenta para desenvolvimento rápido de programas. O uso de uma linguagem de programação genérica como o pascal permite alto grau de flexibilidade, além de gerar um programa executável em qualquer sistema Windows.

O problema abordado na pesquisa é o da programação automática de CLPs, tarefa que exige tempo, mão de obra altamente especializada e, consequentemente, cara. Sabe-se que programas Ladder e Instruction List são difíceis de depurar (UZAM; JONES, 1998), portanto, o uso de técnicas de inteligência artificial pode auxiliar o trabalho de programação minimizando a interferência humana e proporcionando como resultado soluções completas ou próximas do objetivo.

1.2. Objetivos

1.2.1. Objetivos Gerais

Com base em teorias de computação evolutiva, mais especificamente a área da programação genética (KOZA, 2000), (BANZHAF et al., 1998), e pelo fato de uma linguagem de CLP poder ser tratada como qualquer outra linguagem de programação, a utilização da programação genética e a descrição do funcionamento de um sistema de automação através de linhas do tempo são capazes gerar programas para CLP automaticamente.

Dessa forma, o objetivo geral deste projeto de pesquisa é o de contribuir para o avanço dos sistemas de automação industrial e residencial através da programação automática de CLPs. Atualmente a programação destes é feita através da criação prévia de uma rede de pétri (MORAES; CASTRUCCI, 2007) e em seguida sua conversão para Ladder, ou através de tentativa-e-erro (CARNEIRO; ARAÚJO; LEMOS, 2008).

Agregados ao objetivo geral têm-se como objetivos:

- permitir a criação de diversos programas ao mesmo tempo pela utilização de mais de um programa em execução na mesma máquina ou em mais de uma.
- solucionar problemas complexos sem o desgaste humano, facilitar o processo de programação, reduzir mão de obra e economizar recursos financeiros.

1.2.2. Objetivos Específicos

Pelo fato de algoritmos genéticos serem bastante dependentes de seus parâmetros de simulação, a pesquisa tem como objetivo específico a definição desses de forma a tornar a busca mais eficiente.

A definição das taxas de cross-over (cruzamento), mutação, reprodução, o tipo de seleção e o tamanho da população podem fazer com que o programa tenha convergência ou nunca encontre a solução (NGUYEN et al., 2008). Pretende-se tornar a busca mais eficiente através:

- da redução do número de gerações necessárias para a obtenção de respostas.
- de curvas de probabilidade de sucesso que indiquem o tipo de configuração que aumenta a probabilidade de ocorrência de uma solução ideal.

As variáveis são objetos que concretizam conceitos de modo operacional, distinguível no objeto de estudo, sendo passíveis de mensuração (MARCONI; LAKATOS, 2007). Dessa forma, as variáveis independentes analisadas na pesquisa são: as linhas do tempo em diferentes estudos de caso e os valores das taxas de cross-over e mutação.

As variáveis dependentes são os programas propriamente ditos gerados pela técnica, e o número de gerações para a obtenção das respostas. Para análise destas, foram utilizados conceitos de estatística, a produção das curvas de fitness (nível de aptidão) máximo e médio que indicam a evolução da população de programas no decorrer da simulação, gráficos de probabilidade de sucesso, probabilidade acumulada de sucesso e gráficos do número de simulações necessárias para atingir uma solução (KOZA, 2000), (MANOVIT; APORNTEWAN; CHONGSTITVATANA, 1998).

1.3. Justificativa

A cada dia os sistemas de produção estão mais automatizados. Hoje, além de automatizar tarefas mecânicas como o posicionamento de peças e ferramentas, controle de esteiras e a leitura de dados por meio de sensores também são possíveis automatizar a inteligência e a criatividade. Não se trata de imitar exatamente a forma como os seres humanos fazem, mas de utilizar formas diferentes de inteligência baseados em processos biológicos. A inteligência artificial é uma área da ciência da computação dedicada à automação do comportamento inteligente (LUGER; STUBBLEFIELD, 1997).

A inteligência artificial para o leigo é vista ainda como algo distante, e quase que uma questão de ficção científica, porém há vários anos a técnica tem sido usada para trabalhos sérios como, por exemplo, o programa Remote Agent Deep Space 1 da NASA, de 17 de maio de 1999, que se tornou o primeiro programa de planejamento autônomo de bordo a controlar o escalonamento de operações de uma nave espacial. O mesmo foi capaz de gerar planos de metas de alto nível e monitorar seu funcionamento além de detectar, diagnosticar e recuperar problemas conforme eles ocorriam.

Outro exemplo comum do poder da inteligência artificial foi o programa Deep Blue da IBM, em maio de 1997, que se tornou o primeiro programa de computador a derrotar o campeão mundial de xadrez Garry Kasparov.

O sistema ALVINN (THORPE et al., 1991) foi treinado para dirigir um automóvel e mantê-lo na pista. O mesmo foi colocado em uma mini-Van controlada por computador NAVLAB da CMU e foi utilizada para percorrer os Estados Unidos. Por um percurso de quase 4600 Km ele controlou o veículo durante 98% do tempo, com uma intervenção humana de apenas 2%.

Em 1991, durante a crise do Golfo Pérsico, as forças armadas dos Estados Unidos utilizaram a ferramenta Dynamic Analysis and Replanning Tool (DART) (CROSS; ESTRADA, 1994) que realizou o planejamento logístico de 50.000 veículos de transporte de carga aérea e de pessoal, levando em conta pontos de partida, destinos, rotas e resolução de conflitos. A técnica de planejamento com inteligência artificial permitiu a geração em algumas horas de um plano que exigiria semanas com outros métodos.

O sistema proposto neste projeto tem como objetivo facilitar a criação de programas de automação, de forma que possa ser feito por um leigo, pela inserção do comportamento do sistema na forma de linhas de tempo. Assim, qualquer pessoa que saiba como o sistema deva se comportar ao longo do tempo será capaz de criar o programa que deve ser colocado no CLP.

A geração do programa não é instantânea, exige tempo, porém é automática. No início do processo é preciso uma intervenção humana para definição das funções que o programa a ser gerado deve ser capaz de fazer e, em seguida, o processo segue sozinho. Dessa forma não é necessário ocupar um profissional durante todo o processo de programação. É possível que uma única pessoa possa dar início ao processo de geração de vários programas em pouco tempo. Com a utilização de vários computadores, cada programador pode criar vários programas ao mesmo tempo.

1.4.Resultados Obtidos na Pesquisa

A pesquisa indicou que a técnica foi capaz de gerar diferentes programas-solução para um mesmo problema. Quatro estudos de caso abordando aplicações de automação residencial e industrial indicaram a eficiência do algoritmo em produzir as soluções para todos os programas utilizando os três conjuntos de parâmetros iniciais para as simulações. Contudo, observou-se melhor desempenho nas buscas com maiores taxas de mutação. A pesquisa também demonstrou que a busca de soluções com várias simulações e com número reduzido de gerações se aproxima do caso em que a busca foi feita com poucas simulações e número elevado de gerações.

1.5.Organização da Dissertação

Esta dissertação esta organizada em sete capítulos. O capítulo 2 apresenta uma introdução à programação genética e os padrões de linguagens de CLP e faz uma revisão bibliográfica de trabalhos relacionados com o tema da pesquisa.

O capítulo 3 é apresenta a metodologia da pesquisa, seus métodos de abordagem e de procedimento. Agregado à metodologia, este capítulo apresenta a delimitação do universo pesquisado e o modelo da coleta de dados realizada para tratamento estatístico dos resultados.

No capítulo 4 são apresentados os modelos teóricos de interpretação das linguagens Ladder e Instruction List no formato em que elas são feitas através dos CLPs atuais, da norma e como foram implementadas. Também é demonstrado como foram implementados os simuladores de programas de automação nas duas linguagens estudadas e os princípios da programação genética em todos os seus detalhes.

No capítulo 5, características específicas do programa desenvolvido são apresentadas. Discutem-se detalhes do código fonte do programa, a estrutura das variáveis e os fluxogramas de simulação e programação genética. Portanto neste capítulo a abordagem é prática e próxima do sistema real desenvolvido.

No capítulo 6 são apresentados quatro estudos de caso. O primeiro é um estudo de caso referente à automação residencial para sistemas de iluminação e os três últimos estudos são referentes à automação industrial, especificamente no campo de acionamento de motores. Os quatro estudos de caso são apenas demonstrativos do uso da técnica, pois ela é genérica e pode

ser aplicada em inúmeras situações. Nesta seção são apresentados os dados coletados em gráficos, tabelas, programas de automação propriamente ditos e os cenários utilizados na obtenção destes. Ao final do capítulo é feita a análise e a comparação dos resultados obtidos.

No capítulo 7 as conclusões finais são apresentadas e as futuras perspectivas abertas pela pesquisa feita.

2 Revisão Bibliográfica de Automação e Programação Genética

As teorias de base aplicadas no projeto se concentram nas áreas de inteligência artificial e automação. Dentro da área da inteligência artificial a técnica utilizada é a da programação genética e no campo da automação foi feito um estudo das linguagens de programação de CLP com base na norma IEC 1131-3 e nos manuais de CLP dos fabricantes WEG, Siemens, Allen-Bradley e General Electronics.

2.1 Norma IEC 1131-3 e Linguagens de CLP

A norma IEC 1131-3, definida em março de 1993 pela Comissão Eletrotécnica Internacional (International Electrotechnical Commission - IEC), padroniza as linguagens de programação usadas nos CLPs (LEWIS, 1995).

A norma está dividida em cinco partes:

1. informações gerais.

2. requerimentos de equipamentos e testes.

3. linguagens de programação.

4. guia do usuário

5. especificação de mensagens de serviço.

A norma define cinco tipos de linguagem de programação: três gráficas e duas textuais. As textuais são a Instruction List (lista de instruções) e a Structured Text (texto estruturado). As gráficas são a Ladder Diagram (diagrama escada), a Function Block Diagram (diagrama de blocos funcionais) e a Sequential Function Chart - SFC (gráfico seqüencial de funções).

A norma IEC 1131-3 foi criada para padronizar o funcionamento interno dos CLPs. Atualmente observa-se que os fabricantes a utilizam apenas como uma referência de base, pois cada um utiliza nomes diferentes para instruções semelhantes em sua linguagem. Nota-se diferenças tanto para padrões de linguagem textual quanto para linguagens gráficas, como será visto adiante.

Os CLPs são computadores especializados usados em larga escala no âmbito da automação industrial. Os primeiros surgiram na década de 70 para substituir os quadros de

controle baseados em relé (MORAES; CASTRUCCI, 2007). Nessa época eles podiam apenas manipular entradas e saídas de sinais digitais e realizar funções lógicas simples. Eles consistiam em um processador de um único bit com uma memória para programa, um acumulador e um número de entradas e saídas.

Atualmente existem CLPs com diversos níveis de tecnologia. Existem CLPs pequenos e simples (WEG, 2006), (ROCKWELL, 2004), (GE, 2007), (SIEMENS, 2006) com poucas instruções, mas que permitem realizar um grande número de funções, e CLPs grandes e complexos com instruções capazes de colocar sua linguagem no nível de uma linguagem de programação tipo C e pascal, como pode ser visto no manual de operação do CLP TPW-03 da WEG (WEG, 2008), (SIEMENS, 2005). Além do controle de dados digitais também é feito o controle de variáveis de valores inteiros (Controle PID – Proporcional, Integral e Diferencial), (MORAES; CASTRUCCI, 2007).

Os CLPs mais complexos podem conter em sua linguagem: funções, procedimentos, loops, condicionais, jumps, definição de variáveis, controle de bancos de memória, comunicação de informações com computadores ou com outros CLPs.

A execução interna de um CLP é feita em três etapas distintas que, em conjunto, representam o processo chamado de SCAN (Figura 1). Na primeira etapa todos os sinais de entrada são lidos, na segunda etapa o programa é executado uma vez e na terceira etapa os resultados são escritos nas saídas. Essas três fases são repetidas contínuamente durante o modo de execução do CLP (ou modo “RUN”), (JOHANSSON; ÖHMAN, 1995). A fase de execução do programa consome um tempo variável, porém da ordem de micro-segundos, e em programas grandes e complexos em mili-segundos.

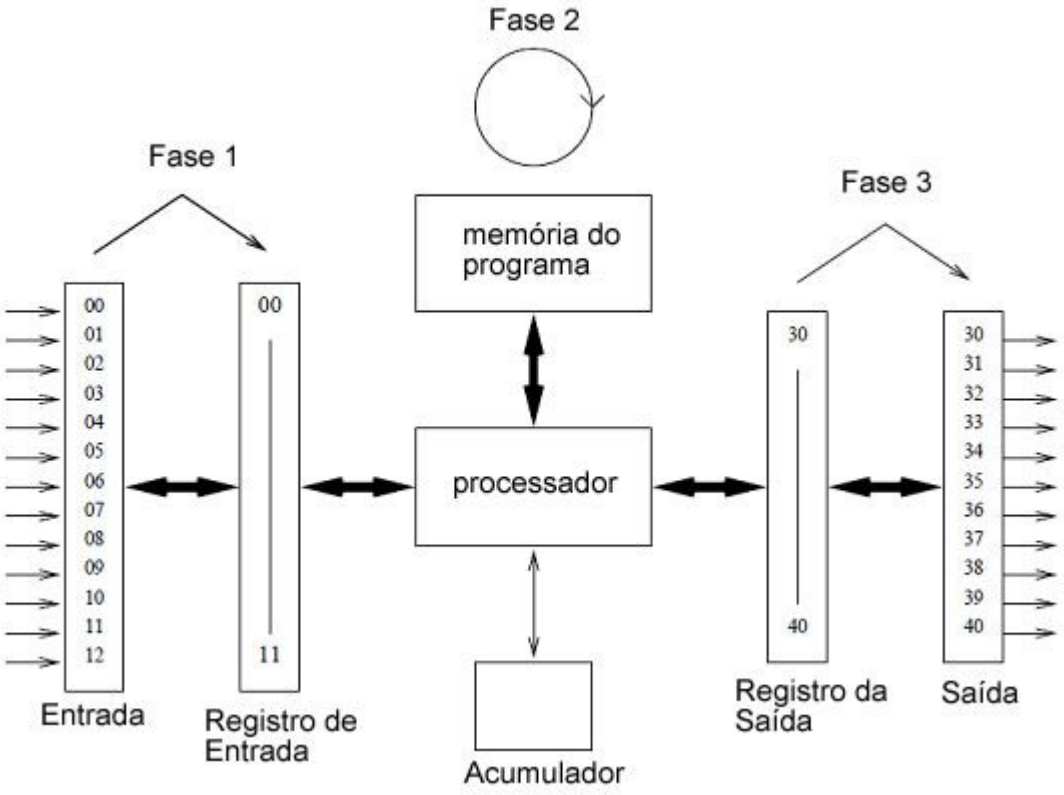


Figura 1 – Fases de execução interna do CLP

As linguagens de programação dos CLPs são definidas e exemplificadas a seguir:

a) Instruction List: linguagem de programação semelhante à linguagem assembly baseado no carregamento e armazenamento de variáveis e no uso de acumulador. Abaixo um exemplo de instruction List:

```
FUNCTION D: BOOLEAN
VAR_INPUT
        A, B, C : BOOLEAN;
END_VAR
LD A        (* Carrega A e armazena no acumulador*)
OR B        (* Or B com o acumulador e armazena no acumulador*)
AND C       (* And C com acumulador e armazena no acumulador*)
ST D        (* Armazena acumulador em D*)
END_FUNCTION
```

b) Texto estruturado: Linguagem similar ao pascal, C e Java. Possui comandos condicionais como If, For...Do, While...Do, Repeat...Until, Case. O código do exemplo anterior é apresentado abaixo em forma de texto estruturado:

```
FUNCTION D: BOOLEAN
VAR_INPUT
A, B, C: BOOLEAN;
END_VAR
D:= A OR B AND C;
END_FUNCTION
```

c) Diagrama Ladder: Linguagem gráfica que implementa a lógica de diagrama de relés (exemplo na Figura 2).

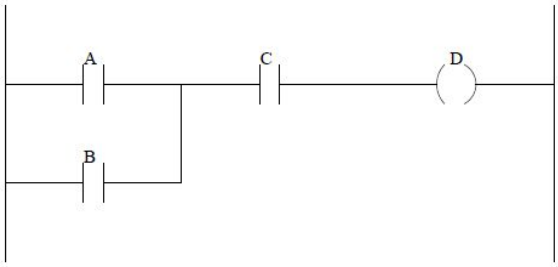


Figura 2 – Exemplo de diagrama ladder

d) Function Block Diagram: Outra representação gráfica da lógica do CLP semelhante à lógica booleana que é aplicada em sistema digitais (exemplo Figura 3):

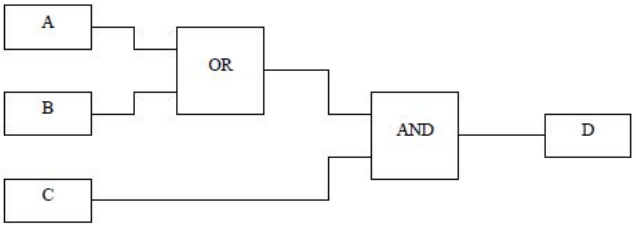


Figura 3 – Exemplo de programa FDB

e) Seqüencial Flow Charts – é uma linguagem gráfica utilizada para descrever operações seqüenciais. O processo é representado como um conjunto contínuo de etapas (na Figura 4, representado por G1, G2, G30 e G31). Cada etapa representa um estado estável. Entre cada estado existem transições (na Figura 4 representado pelas barras

horizontais entre os estados) que são ativadas através de uma condição booleana (JARGOT, 1999). Esta é a linguagem de CLP mais próxima da rede de petri.

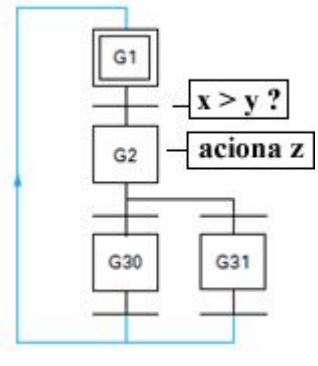


Figura 4 – Exemplo de programa SFC

As linguagens apresentadas acima estão todas em nível de implementação. É importante fazer referência às Redes de Petri (Figura 5), que são diagramas lógicos no nível de projeto baseadas em marcas, estados, transições e ações. Tanto as cinco linguagens acima quanto as Redes de Petri são formas de representação lógica, não havendo superioridade lógica entre elas. A escolha da linguagem é feita de acordo com as possibilidades do equipamento a ser utilizado e pela aptidão do programador. À priori, pode parecer que uma linguagem de texto estruturado seja muito superior a um diagrama ladder, por exemplo, porém isto não é correto, pois existem CLPs com contatos e bobinas específicas para a representação de funções, jumps, condicionais if, loops do tipo for, while, repeat, dentre outros.

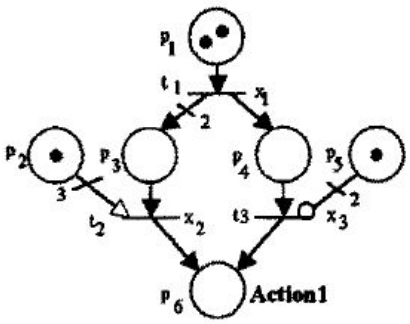


Figura 5 – Exemplo de rede de petri (UZAM; JONES, 1998)

A Rede de Petri é uma forma de representação do funcionamento do sistema de automação que mais se aproxima do pensamento humano. Ela é usada para planejamento e

sistematização do processo como uma forma intermediária para se chegar a uma das cinco linguagens acima.

A linguagem Sequencial Function Charts por ser analisada verticalmente, e pela presença de marcas e transições, é a linguagem que mais se aproxima da Rede de Petri.

2.2 Aplicações de Programação Genética em Sistemas de Automação

O uso da computação evolutiva em aplicações industriais tem sido feito em larga escala (como será visto nas referências a seguir). Encontram-se hoje diversas aplicações que se correlacionam e reforçam a funcionalidade da mesma. É apresentado a seguir a revisão de trabalhos de pesquisadores que aplicaram técnicas de computação evolutiva para solucionar problemas e para otimizar o funcionamento de um ambiente industrial.

(CHEN et al, 2004) apresenta dois problemas associados com a otimização de uma instalação industrial. O primeiro deles é referente à complexidade e a não linearidade dos processos industriais que faz com que seja impraticável uma formulação matemática analítica precisa, o que favorece o uso de uma heurística computacional. O outro problema é a prática usual de técnicos que tem como objetivo apenas manter estável a produção da fabrica, mesmo que ela seja pequena e de baixa qualidade ao invés de fazer um ajuste fino dos parâmetros de produção em suas etapas.

Foi proposto por (CHEN et al, 2004) o uso de programação genética para realizar em uma instalação industrial o procedimento de transferência de estados em multi-passos. O objetivo da técnica é o de otimizar a estratégia de controle pois nesses ambientes a instalação geralmente fornece uma resposta devagar às alterações impostas a ela e a dados de teste, especialmente em indústrias químicas e de metalurgia.

O nível de aptidão das soluções (fitness) é calculado por um algoritmo de avaliação multi-passos que fornece o valor de modelos de objetivos simples atingidos, formando no total uma avaliação sintética do processo ao todo.

As soluções encontradas tentam harmonizar os dois problemas propostos acima, de forma a escolher um ponto próximo e seguro de uma solução para ambos. O uso da otimização multi-tarefa para melhorar o desempenho global possui uma relação direta com o projeto dessa dissertação, que tem como objetivo relacionar multi-comportamentos

expressos por linhas do tempo a fim de criar um código de programação que atenda a todos os comportamentos ao mesmo tempo.

(SYSALA; DOSTÁL, 2004) realizaram um trabalho de implementação de um CLP controlado automaticamente. Um método polinomial é utilizado em conjunto com um modelo delta e com a abordagem dos pólos da função. O fato de problemas de controle industrial possuírem parâmetros que estão em constante alteração torna o uso de sistema de estimação de parâmetros e auto-controle de grande utilidade para a área.

As bibliotecas de funções de um CLP com respeito ao uso de controle PID podem ser muito limitadas. (SYSALA; DOSTÁL, 2004) implementa um tipo de função adaptativa para auto-controle do CLP. O sistema é testado com sucesso em um forno com a presença de um sensor de temperatura e um resistor de aquecimento. Esses resultados reforçam a idéia de se utilizar sistemas inteligentes para o controle de CLPs.

Outro estudo relevante para o projeto é o modelo de simulação de um sistema de redes de petri extendido e a implementação deste em linguagem ladder e instruction list proposto por (FREY, 2000; FREY; LITZ, 2000). Esse sistema chamado de Rede de Petri de Sinal Interpretado (SPIN) possui a capacidade de disparar várias transições simultaneamente, em que pode haver um disparo iterativo de transições antes que um novo estado estável seja atingido.

Esse trabalho além de expandir o conceito de redes de petri também propõe uma técnica para geração automática de códigos ladder e instruction list a partir da rede de petri. Dessa forma, se relaciona diretamente com a proposta deste trabalho, que ao invés de utilizar redes de petri, se utiliza de linhas de tempo.

(UZAM; JONES, 1998) propõem que a melhor técnica para se projetar sistema de controle a eventos discretos de alto-nível (DECS) e implementações de baixo nível são as redes de petri. Estabelecem um novo método chamado de Redes de Petri de automação (APN) para a implementação de DECSs. As redes de petri convencionais não lidam com sensores e atuadores, característica que a APN abrange. Há também uma comparação com o uso do Grafcet que também não atinge o escopo que a APN alcança.

São apresentadas também as dificuldades de uso da linguagem ladder que são difíceis de depurar e também oferecem poucos caminhos para construções estruturais para lidar com esse problema. Este estudo reforça o fato de que a criação de um programa ladder de forma indireta é bastante desejado.

Também é reforçada a idéia de que o melhor método desenvolvido até hoje para a criação de uma ladder é a prévia criação de uma rede de petri e depois sua conversão. O trabalho dessa dissertação propõe justamente uma nova técnica para que se alcance o programa ladder e sem passar pela etapa da rede de petri.

É proposta a criação da ladder a partir da rede de petri extendida utilizando-se os padrões da norma IEC 1131-3. Observa-se na prática dos fabricantes atuais que a norma é utilizada apenas como uma referência, mas não como regra, pois são observados diferentes padrões de nomes de código em linguagem instruction list, e diversidade de funções em ladder de um CLP para outro, de fabricante para fabricante.

A pesquisa de (DADONE; VANLANDINGHAM, 1998) obteve resultados próximos do objetivo deste projeto. Foi utilizado um algoritmo genético para a criação de um programa ladder para o controle de uma instalação industrial. Os indivíduos foram representados com comprimento fixo e codificação binária.

A função objetivo para atribuição do fitness dos indivíduos é específica para cada caso, como esperado. É feita uma simulação do código gerado; depois aplicado em uma instalação industrial virtual e analisado a quantidade de produto final produzido.

A pesquisa utilizou um método denominado roleta para fazer a seleção dos indivíduos, tratando-se assim do modo de seleção proporcional. É apresentado também o problema da ocorrência de simulações com convergências prematuras para mínimos locais, reforçando o uso do sistema de seleção torneio.

A solução proposta pelo autor para esse problema foi o uso do operador genético de mutação a uma baixa taxa. Foi verificado que o uso deste em grandes proporções provoca a perda de muitas soluções boas para o problema, portanto o operador genético principal deve ser o cross-over. Verificou-se também o uso de elitismo nas simulações. Um ponto importante abordado por (DADONE; VANLANDINGHAM, 1998) é o fato do algoritmo genético ser eficiente em problemas onde as soluções não são deriváveis, que é o caso da construção de um programa de CLP.

A técnica apresentada foi utilizada em um estudo de caso de uma instalação químico-industrial. A quantidade de produto final produzido foi utilizado para o cálculo do fitness dos indivíduos. Dessa forma à medida que as soluções evoluem o comportamento global da produção é otimizado.

Essa pesquisa demonstrou a possibilidade da criação de programas de CLP através da programação genética e foi a abordagem encontrada mais próxima desse projeto. O

estudo apresentado utilizou indivíduos de comprimento fixo, o que pode ter sido suficiente para solucionar o estudo de caso abordado, porém essa abordagem limita bastante o conceito de programação genética.

A criação de um programa de automação, seja em ladder ou instruction list, se enquadra no estudo de síntese de circuitos lógicos seqüências síncronos a partir de seqüências parciais de entrada e saída (MANOVIT; APORNTEWAN; CHONGSTITVATANA, 1998). A partir da função dos estados de transição desejados o algoritmo genético de (MANOVIT; APORNTEWAN; CHONGSTITVATANA, 1998) sintetizou circuitos que imitavam o comportamento desejado para as saídas em relação às entradas propostas. Foram feitos estudos de caso para circuitos lógicos como contadores, somadores seriais, divisores de freqüência, detector de módulo-5 e verificador de paridade.

Verificou-se também que o comprimento das seqüências de entradas e saídas afetam diretamente a eficiência do sintetizador. É preciso formar uma seqüência de entradas longa o suficiente para testar todos os aspectos funcionais do circuito. Se a seqüência for muito pequena ela causará ambigüidade na descrição do circuito desejado.

O tempo das simulações também deve ser considerado. Os recursos de hardware disponíveis e o formato do programa podem inviabilizar a pesquisa dentro do prazo disponível se uma simulação gastar tempo demais, visto que centenas delas precisam ser feitas. Portanto o tamanho das seqüências precisam englobar todas as funções dos circuitos mas também não podem superar limites físicos de tempo, pois quanto maior a seqüência, mais demorada ela é.

(MANOVIT; APORNTEWAN; CHONGSTITVATANA, 1998) verificam também que o crescimento das seqüências atinge uma saturação, deixando de exercer ação adicional na escolha correta do circuito. Três variáveis precisam ser ajustadas em conjunto: o tamanho da seqüência, o nível de acerto e esforço físico do sistema.

(JAFARI; SAFAVI; FADAEI, 2007) apresentam a aplicação de um algoritmo genético implementado em linguagem C para a otimização dos movimentos de um robô com três graus de liberdade. O algoritmo genético consegue alcançar melhores resultados que o Algoritmo Complexo apresentado por (PETTERSOON et al., 2004.).

Um método automático de projeto de sistemas dinâmicos baseado no algoritmo Hungaro e em programação genética (HAGP) é apresentado por (SHAOBO; GUANCI; QINGSHENG, 2008). A pesquisa investiga um método eficiente de cálculo de fitness baseado em gráficos. HAGP demonstra ser bom o suficiente para fazer modelos automáticos possíveis

e de boa qualidade através de soluções bem evoluídas com pouco esforço computacional, e rápida convergência comparado com o estado da arte de outros algoritmos.

(GALVAN-LOPEZ, 2008) estuda a implementação de um algoritmo genético com representação em grapho para solução de problemas com múltiplas saídas. Essa abordagem, chamada de múltiplas saídas interativas em arvore simples (MIOST), foi estudada primeiramente com a utilização de apenas mutação como operador genético para que suas implicações fossem analisadas.

Em seguida foram feitas simulações com mutação e o cross-over padrão do processo evolucionário. A pesquisa demonstrou bons resultados para simulações com apenas mutação, mas não demonstrou avanços quanto a resultados anteriormente apresentados com respeito ao número de saídas. Esses resultados foram justificados pelo fato da técnica não ter sido aplicada sob uma abordagem de otimização.

A programação genética foi aplicada com relevantes resultados para o conforto e bem estar em residências (HAGRAS, 2008). Os ambientes residenciais estão sendo gradualmente ocupados por dispositivos digitais que possuem capacidades computacionais e de rede. O aumento desses dispositivos começou a gerar uma necessidade de criar uma forma de inteligência para auxiliar na configuração e programação de tantos dispositivos. Através do emprego da programação genética foi possível criar uma "presença" inteligente capaz de reconhecer os usuários e automaticamente programar o ambiente para ser energeticamente eficiente, respondendo às necessidades e comportamento do usuário. Hagras afirma que os algoritmos genéticos são a técnica de computação evolucionária mais utilizada em ambientes residenciais.

(LAM, 1993) desenvolveu um controle ótimo de energia a partir de um algoritmo genético. Em seu estudo, o consumo de potência de ar condicionados foi escolhido como função objetivo a ser minimizada utilizando PG. Simulações computacionais fora utilizadas para comparar a performance de um controlador ótimo usando PG, controladores ON-OFF e controladores PI. Os resultados mostraram que a redução no consumo foi alcançada na maioria dos casos pelo controlador baseado em PG mantendo maiores níveis de conforto térmico.

(LAM, 1995) apresentou também uma metodologia que emprega um sistema classificador com PG para habilitar um controlador de ar condicionado a aprender a partir de suas experiências a melhor estratégia e controle em contraposição a um dado esquema de avaliação de performance.

(RAUSCH; KROGH, 1998) realizaram uma pesquisa a respeito de CLPs e apresentaram um simulador de programas em Ladder chamado SMV. Simulação é feita

utilizando-se codificação binária e diagramas de blocos. (LIU; DARABI, 2002) também apresenta sistemas para simulação da linguagem Ladder utilizando o método de Ramadge-Wonham. Outras pesquisas relacionadas com o desenvolvimento de simuladores de programas de CLP são apresentadas por (FREY, 2000), (FREY; LITZ, 2000), (MINAS; FREY, 2002).

A presença de um simulador de CLP no projeto dessa dissertação se torna um ponto central na pesquisa pelo fato da simulação ser o objeto gerador do fitness dos indivíduos na PG. É com os resultados apresentados pelas linhas do tempo que o algoritmo genético é capaz de selecionar as soluções para a reprodução.

2.3 Conclusão da revisão bibliográfica

Portanto, diante da revisão apresentada, conclui-se que este projeto incorporou de outros projetos as idéias de utilização da computação evolutiva para otimização de problemas de automação (SYSALA; DOSTÁL, 2004), (SYSALA; DOSTÁL, 2004). Como solução para a impossibilidade de representação matemática analítica (CHEN et al, 2004), o problema foi representado por linhas do tempo e utilizou-se PG como heurística, assim como (CHEN et al, 2004). Para que fosse possível a avaliação do nível de aptidão das soluções foi necessário criar um simulador de linguagens de CLP (FREY, 2000; FREY; LITZ, 2000), (RAUSCH; KROGH, 1998), (LIU; DARABI, 2002), (MINAS; FREY, 2002).

Percebe-se que a representação de um problema de automação com redes de petri tradicionais possui grandes limitações (UZAM; JONES, 1998), (FREY, 2000; FREY; LITZ, 2000); a utilização de linhas do tempo também visa superar estes problemas. As linhas do tempo representam também uma forma gráfica de cálculo do fitness, o que aproxima este projeto dos trabalhos de (SHAOBO; GUANCI; QINGSHENG, 2008).

A utilização da programação genética em programas com comprimento variável torna maior a quantidades de problemas resolvíveis, sendo assim uma abordagem mais flexível que de (DADONE; VANLANDINGHAM, 1998).

Para a representação de todos os comportamentos possíveis de um sistema automatizado são necessários vários cenários, cada um contendo um aspecto do ambiente. Esta é uma abordagem que se enquadra nas definições de seqüências parciais de entrada e saída (MANOVIT; APORNTEWAN; CHONGSTITVATANA, 1998).

A evolução de programas feitas através da instruction list, que é semelhante a linguagem assembly, ou seja, uma linguagem de baixo nível, se aproxima dos trabalhos de (JAFARI; SAFAVI; FADAEI, 2007) que utilizaram linguagem C (linguagem de nível intermediário).

Os resultados finais mostraram que uma quantidade elevada de mutação oferece melhores resultados, aproximando este trabalho da pesquisa de (GALVAN-LOPEZ, 2008) que utiliza apenas o operador de mutação.

O primeiro estudo de caso (sistema de iluminação de escadas) representa uma aplicação residencial, assim como os estudos de (HAGRAS, 2008) e (LAM, 1993) com algoritmos evolucionários em ar condicionados.

3. Metodologia da Pesquisa

3.1. Métodos de Abordagem e de Procedimento

Este projeto utiliza o método de abordagem **hipotético-dedutivo** pois a partir do tema escolhido um problema foi colocado e uma solução proposta, a hipótese básica. Para comprovar ou refutar tal hipótese as variáveis serão mensuradas, ou seja, dados são coletados para análise e interpretação (MARCONI; LAKATOS, 2007).

Os métodos de procedimento aplicados neste projeto são:

• **Método estatístico**: serão coletadas grandes amostras, de forma sistemática, para a obtenção de maiores certezas em relação aos fenômenos estudados. A descrição qualitativa e quantitativa das amostras será apresentada graficamente.

• **Método comparativo**: os estudos de casos feitos serão comparados uns com os outros, e com estudos das referências bibliográficas, com o objetivo de formular afirmações para os demais casos semelhantes não estudados.

Para este projeto, foram selecionadas as seguintes técnicas de coleta de dados:

• **Pesquisa bibliográfica**: verificar a evolução histórica do enfrentamento do problema (revisão bibliográfica), além do estudo em livros e artigos para aperfeiçoar a teoria de base, que sustenta a hipótese básica;

• **Pesquisa de campo quantitativa-descritiva**: investigação empírica para delineamento do problema, análise de fatos e isolamento de variáveis. A técnica será usada para a **verificação de hipóteses**, **avaliação do programa** e estudos para **a eficiência da técnica**.

3.2. Delimitação do Universo e Tratamento Estatístico

A coleta de dados será feita através da realização de simulações com parâmetros predefinidos para preencher os dados da Tabela 1. O número de simulações realizadas na segunda coluna da tabela é decrescente devido ao considerável aumento do tempo de simulação proporcional ao aumento do número de gerações a serem simuladas. Contudo foi mantido o mínimo de trinta simulações para as simulações mais longas, considerando que este número é capaz de coletar uma amostra representativa (COSTA, 2005). Será utilizado o

cálculo do esforço computacional proposto por (KOZA, 2000). Este método consiste em coletar dados estatísticos de sucesso em "**N"** execuções independentes com populações de "**M"** indivíduos cada, em "**G"** gerações.

Para cada grupo de execuções independentes com os mesmos parâmetros de cross-over, mutação e limite de gerações "**G"** é calculada a probabilidade de sucesso da simulação Y(M,i) (equação 1).

Para cada seqüência de execuções com número crescente de limite de gerações é calculada a probabilidade acumulada de sucesso para M indivíduos até a geração i, ou P(M,i) (equação 2).

$$Y(M,i)_i = \frac{Número\_de\_sucessos}{Número\_de\_execuções} \tag{1}$$

$$P(M,i)_i = \frac{Número\_de\_Sucessos\_Acumulados}{Número\_acumulado\_de\_execuções} \tag{2}$$

A probabilidade z de obtenção de sucesso até a geração i com M indivíduos em até R execuções independentes é dada pela (equação 3):

$$z = 1 - [1 - P(M,i)]^R \tag{3}$$

Será considerado z = 99%, assim como Koza em seus experimentos. Aplicando-se logarítmos à equação, obtém-se (equação 4):

$$R = R(M,i,z) = \left\lceil \frac{\log(1-z)}{\log(1-P(M,i))} \right\rceil \tag{4}$$

A equação (4) representa o número estimado de execuções independentes para sucesso com probabilidade "**z"** até a geração "**i**" com o uso de "**M"** indivíduos (equação 4). O esforço computacional é medido pela quantidade de avaliações de aptidão (número de indivíduos processados) necessárias para alcançar o sucesso. Esse número é obtido pelo produto do número de indivíduos da população pelo número de gerações avaliadas, sendo assim o esforço computacional estimado, em número de indivíduos a serem processados, para a obtenção de uma solução com probabilidade "**z"** (equação 4).

Segundo Koza, autor do método, este valor é uma estimativa pós-processamento, dependente de todo o contexto (“**M”**, “**G”** e outros parâmetros), baseada nas estatísticas coletadas, e não representa, portanto, o mínimo esforço computacional para o problema (KOZA, 2000).

Tabela 1 – Tabela para coleta de dados das simulações

Maximo de gerações	**Numero de simulações**	**Nº de sucessos**	**Nº simulações acumulado**	**Nº de sucessos acumulado**	**Y(M,i)**	**P(M,i)**	**R(z)**
25	150						
50	100						
100	80						
150	50						
200	30						
300	30						
400	30						
500	30						
Total							

Para cada estudo de caso serão representados alguns programas obtidos como soluções para os problemas apresentados em linguagem Instruction List e Ladder. Logo em seguida será feita uma discussão a respeito destes, levando em conta seus objetivos, suas particularidades.

4. Funcionamento Teórico e Simulações do Projeto

4.1. Teoria e Implementação dos Simuladores para Automação

4.1.1. Interpretação da Linguagem Ladder

A Ladder por ser uma linguagem gráfica é representada por um diagrama e não por linhas de código. Visualmente sua lógica é representada por duas barras verticais, uma à esquerda e outra à direita, e entre as mesmas existem linhas horizontais que contêm sua seqüência de contatos. Em meio a estas linhas horizontais podem existir ramificações verticais que formam uma espécie de “árvore lógica”.

Toda linha horizontal deve estar ligada a uma das barras verticais, sendo que as horizontais ligadas à barra da esquerda iniciam a lógica e as ligadas à barra da direita (ou apenas a uma bobina) finalizam a lógica.

A Ladder deve ser analisada de cima para baixo e da esquerda para a direita. Os contatos da linguagem representam as condições a serem obedecidas. Ao final de uma linha horizontal deve existir uma bobina que representa a saída de informações. Dessa forma os contatos sempre permanecem à esquerda e as bobinas à direita.

Em programas Ladder mais simples uma bobina está energizada quando existe um caminho contínuo entre ela e a barra da esquerda. Dessa forma todos os contatos devem permitir que a “energia” chegue até a bobina. Através dessa representação simples a linguagem se torna muito limitada. Atualmente os fabricantes de CLP adicionaram elementos lógicos que superaram essa limitação, porém isso fez com que a Ladder deixasse de representar um caminho elétrico entre uma bobina e a fonte para representar unicamente uma estrutura lógica, capaz de criar o comportamento desejado para o sistema de automação.

Dessa forma é comum presenciar uma bobina energizada em uma Ladder sem que exista um caminho fechado entre ela e a barra vertical da esquerda. A presença de elementos com memória dentro da lógica ladder (como por exemplo, pulsadores, bobinas

de retenção, temporizadores e contadores) exige que a lógica seja analisada através de uma linha do tempo (CARNEIRO; ARAÚJO; LEMOS, 2008).

Um contato normalmente aberto (NA) não comutado não conduz corrente, portanto, uma bobina sendo acionada através de um contato desses não está ligada. Já um contato normalmente fechado (NF) não comutado permite que exista fluxo de corrente, portanto uma bobina conectada a este contato está ativa.

Quando uma entrada é ativada o estado de um contato referente a essa entrada se torna oposto ao estado normal. Assim um contato NA ativo (comutado) é fechado e passa a conduzir corrente, e um contato NF ativo (comutado) é aberto e interrompe a corrente.

Os contatos básicos existentes em uma ladder são os contatos NA e os contatos NF. Entre os contatos com memória existem os pulsadores (quando energizados deixam passar um pulso de energia) e os elementos SET (quando recebem uma transição positiva forçam a linha a sua direita a ficar energizada) e RESET (que forçam a linha a sua direita a ficar desenergizada).

Os fabricantes tem diversificado os tipos de bobinas na programação Ladder de seus CLPs. Essas mudanças tornaram a antiga Ladder, que representava quadros de relés, em uma linguagem mais próxima das linguagem de programação de computadores, tornando a linguagem mais estruturada. O que existe de mais comum com respeito à bobinas são: as bobinas simples, as bobinas com retenção, os temporizadores, e os contadores.

A linguagem Ladder como qualquer outra linguagem possui regras que devem ser respeitadas e, quando isso não acontece, ou a ladder simplesmente não pode ser escrita ou então podem ocorrer comportamentos inesperados.

Um exemplo simples desse fato é a utilização de uma mesma bobina mais de uma vez em um programa Ladder (Figura 6).

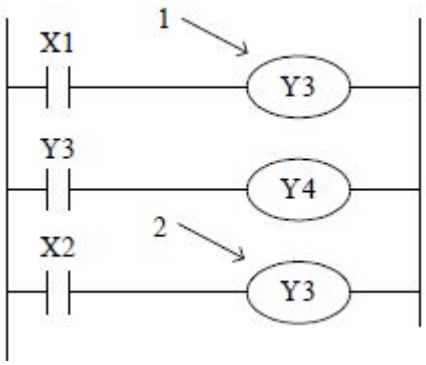


Figura 6 – Exemplo de situação causadora de "bug" em uma Ladder

Quando um CLP é colocado em modo "RUN", vários SCANs começam a ser processados. Utilizar o mesmo endereço de memória em mais de uma instrução de saída no programa não é uma prática recomendada.

O exemplo do programa mostrado na fig. 6 mostra a situação em que duas bobinas, em posições diferentes, utilizam o mesmo endereço de memória (existem duas saídas Y3). A seguinte seqüência de eventos ocorrerá quando X1 = ON e X2 = OFF:

a) A primeira linha lógica de Y3 terá seu valor de status ON porque X1 está ON. Os contatos associados com a Y3 também energizarão quando a bobina de saída Y3 energizar. Portanto, a saída Y4 também será energizada (ON).

b) A última e mais importante linha deste programa mostra o estado da entrada X2. Como X2 está desenergizada (OFF), a saída Y3 NÃO será energizada. Portanto, ao final do ciclo de varredura (SCAN), o estado da bobina Y3 que o programa irá mostrar fisicamente será OFF apesar da lógica indicar a primeira bobina Y3 como ON. Os estados finais das saídas serão então Y3 = OFF e Y4 = ON.

É preciso evitar situações onde o mesmo endereço de memória tenha sido utilizado mais de uma vez para uma instrução de saída do programa.

Quando existe repetição de um mesmo endereço em diversas saídas do programa, internamente poderá haver alteração na lógica, porém sempre a última instrução de saída com este endereço é que será utilizada para atualizar o estado da saída física.

4.1.2. Interpretação da Linguagem Instruction List

Entre os CLPs existem diferenças na nomenclatura dos comandos, como por exemplo a instrução para carregamento de contato inicial do CLP TP-02 WEG é "STR" e no TPW-03 WEG é "LD", porém a forma de análise e leitura do programa é a mesma. Para implementação dessa linguagem foram escolhidas as instruções do TPW-03 pela facilidade da linguagem exposta em seu manual e pela livre distribuição deste manual no site do fabricante.

Para a simulação de uma instruction list é preciso criar uma matriz de acumuladores. A representação de um programa de automação através de uma instruction

list não exige representação gráfica. Como não há uma correspondência geométrica entre os elementos do programa, apenas correspondência lógica entre variáveis booleanas, o processamento se torna mais rápido que o processamento de um programa representado em Ladder.

Foi utilizada uma matriz linha “Reg” de variáveis booleanas com 500 posições de memória. Utiliza-se também uma variável inteira **i** como apontador de memória para essa matriz booleana. O código de declaração dessas variáveis em pascal é apresentado abaixo:

```
Reg: array [1..500] of boolean;
i: integer;
```

No início de qualquer simulação o valor inicial desse apontador **i** se torna 250, ou seja, está indicando um elemento no meio da matriz. À medida que o programa é simulado, o apontador **i** pode ser incrementado ou decrementado de forma que valores lógicos vão ocupando a matriz **Reg** em diferentes posições de acordo com a leitura do programa instruction list.

O tamanho da matriz **Reg** escolhido é extenso o suficiente para garantir que durante a simulação de qualquer programa o apontador **i** nunca assuma um valor abaixo de zero nem acima de 500, o que provocaria erro durante as simulações.

Define-se como referência uma letra seguida de um número que indica uma entrada, saída, contato auxiliar ou temporizador, como por exemplo: I1, I2, I3, ... , para entradas; Q1, Q2, Q3, ..., para saídas; M1, M2, M3, ... , para contatos auxiliares; e T1, T2, T3 para temporizadores.

Associado a cada contato de um programa de automação, seja ele em ladder ou em instruction list, existe uma referência. A seguir são apresentados os operadores básicos da linguagem instruction list utilizadas nesse projeto e o detalhamento de como eles funcionam:

- LD

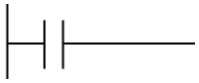


Figura 7 – Comando LD

O comando “LD” (Figura 7) representa o primeiro contato da lógica do tipo NA. O estado do contato é carregado na matriz **Reg**. Se ele estiver comutado (fechado) é carregado o valor 1, se não estiver comutado é carregado o valor 0.

- LDI

Figura 8 – Comando LDI

Primeiro contato da lógica do tipo NF (Figura 8). É carregado na matriz **Reg** o estado do contato, se ele estiver comutado (fechado) é carregado o valor 0, se não estiver comutado é carregado o valor 1.

LD e LDI também podem ser utilizados para informar o início de uma seqüência lógica dentro de uma ramificação lógica quando estiver usando as instruções ORB e ANB.

- OUT (Bobina)

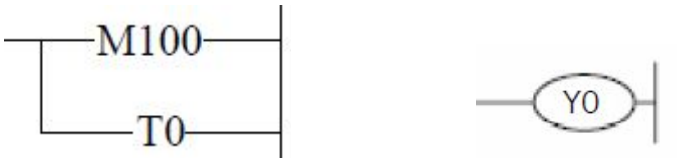


Figura 9 – Comando OUT

O comando OUT (Figura 9) finaliza um ramo lógico e armazena o resultado dos cáculos da lógica anterior. Como pode ser visto nessa figura, é possível conectar várias bobinas em paralelo.

É possível configurar uma bobina como: comum (simples), temporizadora, contadora, constante (de retenção) ou de pulso.

As bobinas comuns apenas armazenam seu estado ativo. As bobinas temporizadoras costumam ser mais complexas. De CLP para CLP existem variações. Normalmente a bobina armazena o valor de sua contagem atual, o valor limite, e o estado de sua bobina (ativo ou inativo). Alguns CLPs possuem outras variáveis que podem ativar diferentes modos de temporização como, por exemplo, contagem regressiva, continuação

da contagem após o limite ser atingido, e outras opções que variam de fabricante para fabricante.

Isso também acontece com os contadores que costumam armazenar seu valor atual, seu valor limite e o estado da bobina, além de ter outras opções dependendo do fabricante.

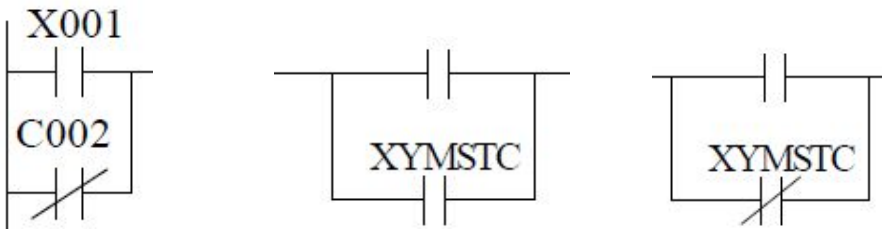
- AND e ANI

AND ANI

Figura 10 – Comandos AND e ANI

Os comandos AND e ANI (Figura 10) realizam a conexão em série dos contatos. O comando AND executa a operação lógica “and” entre a posição atual da matriz booleana **Reg** e o valor do próximo contato NA em série. O comando ANI executa a operação “and” entre o acumulador e o próximo contato NF em série. A quantidade de contatos em série não possui limite lógico, essa limitação é definida apenas pelo programa específico do CLP.

- OR e ORI



ORI OR ORI

Figura 11 – Comandos ORI, OR e ORI

O comando OR (Figura 11) gera a conexão paralela da ladder e a função lógica “or”. O comando OR executa a operação lógica “or” entre o acumulador e o contato NA em paralelo. O comando ORI executa a operação para um contato NF em paralelo. O uso

desses comandos é mostrado na Figura 12 com a representação ladder e instruction list do mesmo programa.

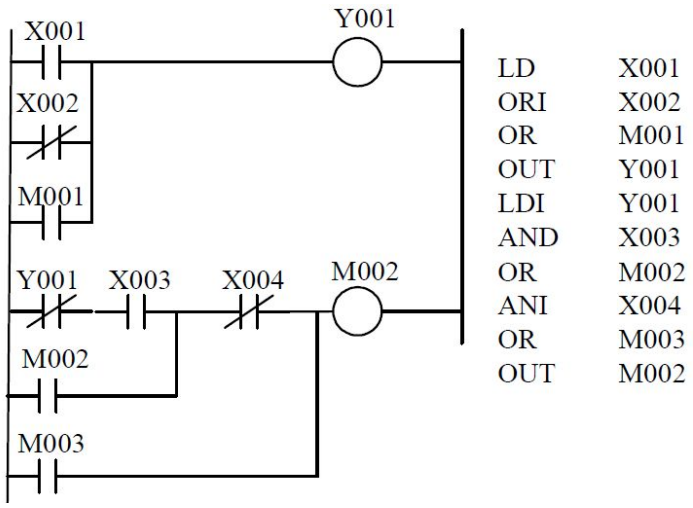


Figura 12 – Exemplo de programa com os comandos OR e ORI

- LDP e LDF

Figura 13 – Comandos LDP e LDF

LDP (Figura 13) é usado quando um contato de pulso está conectado diretamente a barra da esquerda da ladder (operação lógica inicial). Ela executa a operação lógica de pulso alto pelo período de um scan. A LDF (Figura 13) mantêm a linha à sua direita em ativo alto e realiza um pulso baixo pelo período de um scan.

- ANDP, ANDF

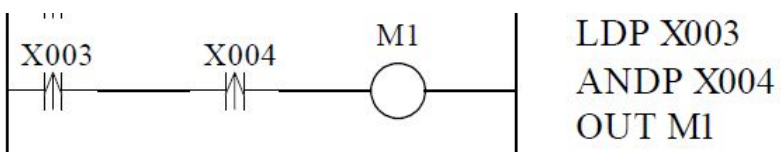


Figura 14 – Comando ANDP

A operação ANDP (Figura 14) é usada para conectar um contato de pulso alto em série com outro contato. A operação ANDF (Figura 14) é utilizada para contectar um contato de pulso baixo em série.

- ORP e ORF

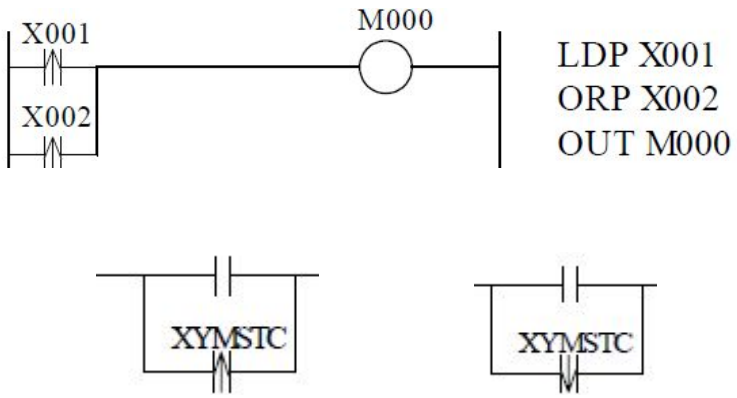


Figura 15 – Comandos ORP e ORF

A operação ORP (Figura 15) é usada para conectar em paralelo um contato de pulso alto com o contato anterior. O ORF (Figura 15) é utilizado para o contato de pulso baixo.

- ORB (Operação OR para circuitos lógicos)

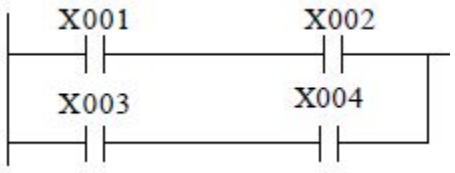


Figura 16 – Comando ORB

A instrução ORB (Figura 16) é independente e não está associada à nenhum endereço, ela é usada para conectar dois circuitos lógicos em paralelo para a construção de circuitos mais complexos. Geralmente esta operação conecta blocos de circuitos em série ao circuito paralelo precedente.

A declaração do circuito em paralelo é feita com as instruções LD ou LDI. Quando estas instruções são utilizadas, o resultado da lógica vai sendo armazenado na matriz **Reg**, na posição do apontador **i**, inicialmente. Quando LD ou LDI são utilizados pela segunda o apontador **i** é incrementado e a lógica booleana passa a ser armazenada na matriz **Reg** na

posição (**i+1)**. Ao utilizar a instrução ORB é realizado a operação lógica OR na matriz **Reg** entre as posições **i** e **i+1.** O resultado da operação é armazenado em i, e o apontador é decrementado **(i-1)**, voltando de **i+1** para **i**. Essa operação resulta no equivalente de dois circuitos em paralelo.

No caso de três circuitos em paralelo (Figura 17), o efeito é o mesmo. A lógica do primeiro circuito vai sendo armazenada na matriz **Reg** em **i.** Quando o comando LD é utilizado pela segunda vez o apontador de memória se torna **i+1**. A instrução ORB é realizada e todo conteúdo dos dois circuitos fica mantido em **i**. O terceiro circuito (terceiro uso de LD) armazena seus dados em **i+1** novamente e ao encontrar outra instrução ORB realiza novamente a operação OR entre **i** e **i+1**.

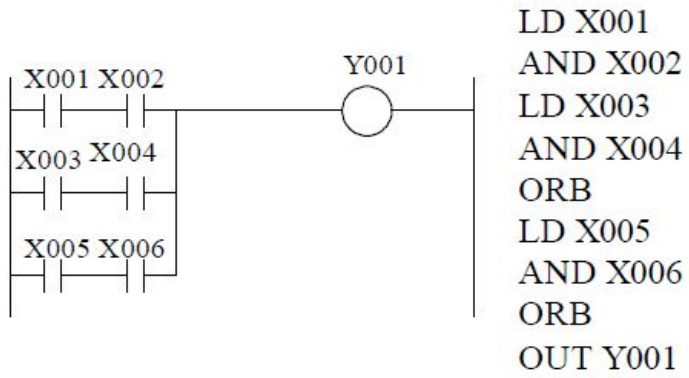


Figura 17 – Exemplo de uso do comando ORB

- ANB (Operação AND para circuitos lógicos)

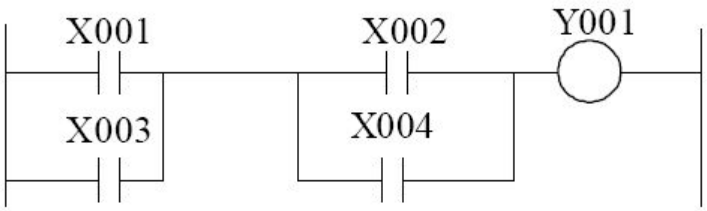


Figura 18 – Exemplo de uso do comando ANB

A instrução ANB (Figura 18) é independente e não esta associada a nenhum endereço, ela é usada para conectar dois circuitos lógicos em série para a construção de circuitos mais complexos.

A declaração do circuito em série é feita com as instruções LD ou LDI. À medida que estas instruções são usadas a lógica é armazenada em acumuladores em seqüência.

Quando LD ou LDI é utilizado o resultado da lógica vai sendo armazenado na matriz **Reg** na posição **i**, quando utilizado pela segunda vez elas são armazenadas em **i+1**. Quando utilizado pela terceira vez a lógica é armazenada em **i+2**, e assim sucessivamente. A limitação de acumuladores para realização desse tipo de operações é definida pelo fabricante. No caso do TPW-03 podem ser usados até oito registradores (até **i+7**) para armazenar o circuito lógico. No caso do projeto atual é aceito até a posição **i+249**.

Um programa que exija uma sequencia de armazenamento de memória que vá até o final dessa matriz não poderá ser implementado em um TPW-03. Em um situação dessas é preciso otimizar e rescrever o mesmo programa com uma lógica correspondente que não utilize tantos registradores.

Ao utilizar a instrução ANB é realizada a operação lógica AND entre o acumulador corrente **i** e o anterior (**i-1**) resultando assim no equivalente de dois circuitos em série.

- MPS, MRD e MPP

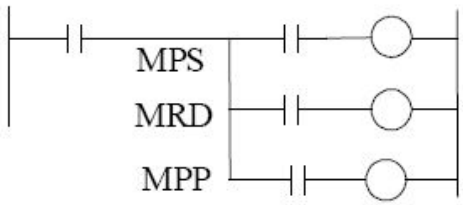


Figura 19 – Comandos MPS, MRD e MPP

As instruções MPS, MRD e MPP (Figura 19) são utilizadas para gerar ramificações à direita da linha lógica para a conexão de bobinas ou blocos de circuitos lógicos.

No momento em que MPS é usado o valor atual da lógica é armazenado. Quando MRD é encontrado o ramo do circuito é forçado a se conectar à lógica armazenada anteriormente.

A instrução MPP é usada para indicar que as ramificações chegaram ao fim. Para cada MPS usado no programa é necessário usar um MPP. No momento em que ela é encontrada o próximo ponto do circuito é conectado à lógica anterior e em seguida o registro de memória é apagado.

- SET e RST

A instrução SET (Figura 20) coloca o valor ativo em um registrador e o RST coloca o valor inativo.

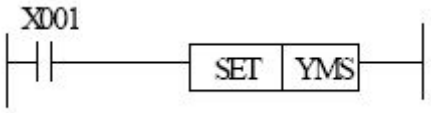


Figura 20 – Comando SET

No exemplo acima quando a entrada X001 é ativada a instrução SET é executada e seu registrador de referência (YMS) assume o valor ativo. Se a entrada X001 se tornar inativa o registrador a que o SET faz referência continuará ativo.

Figura 21 – Comando RST

O mesmo ocorre com a instrução RST (Figura 21), a diferença é que a referência assume o valor inativo.

- NOP

A instrução NOP não executa nenhuma função. Ela é mais utilizada para substituir instruções e facilitar a edição do programa.

- END

A instrução END indica o fim do programa para que o ciclo de SCAN possa voltar para o início.

4.2.1. Simulação Computacional de uma Ladder

No decorrer desta pesquisa, obteve-se como produto uma publicação relacionada a geração de programas de automação utilizando a linguagem Ladder diretamente

(CARNEIRO; ARAÚJO; LEMOS, 2008). Nesse trabalho, foi feita a criação de um simulador de programas Ladder baseado em uma matriz onde a disposição geométrica de seus dados modificavam o comportamento da simulação. A metodologia se utiliza de uma matriz de registros (Figura 22):

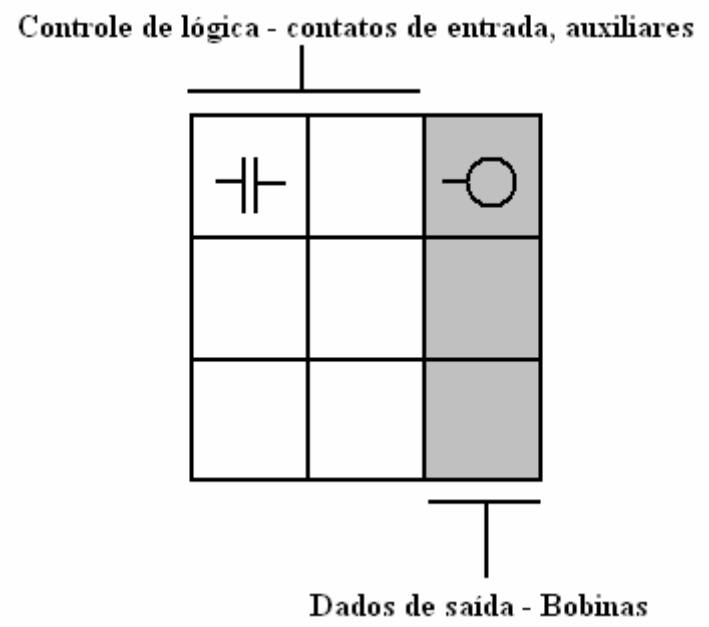


Figura 22 – Diagrama demonstrativo da matriz de registros

Nessa matriz as células da última coluna são reservadas para armazenar apenas bobinas simples, bobinas de retenção e temporizadores. As demais células podem assumir os elementos (Figura 23): contato NA, contato NF, pulsador, fio ou jump (elemento de junção ou derivação).

Figura 23 – Elementos da ladder

Cada uma das células da matriz é independente uma da outra. Elas possuem os atributos: entrada, saída, função, referência e estado (Figura 24).

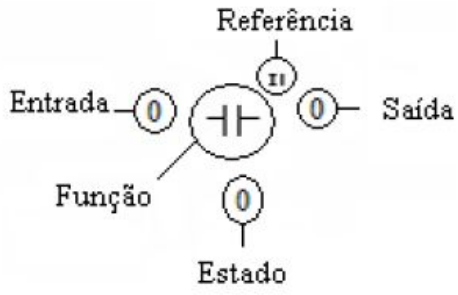


Figura 24 - Atributos de uma célula

O objetivo de cada célula é obter seu valor de entrada, verificar o estado da referência que esta associada a ela, realizar sua função e determinar o valor de sua saída. A simulação de uma ladder através dessa metodologia assume um carater mais gráfico por apresentar como resultado o diagrama Ladder diretamente, e pelo fato da disposição geométrica das celulas da matriz influenciarem na lógica da simulação.

Durante o processo de simulação da Ladder todas as entradas, saídas e estados das células são zerados. Logo em seguida as células que estão na primeira coluna da matriz adquirem inicialmente valor 1 em suas entradas. Ao longo da simulação dois processos precisam ser sincronizados, que são os ciclos de scan e os incrementos no relógio do tempo. Para cada incremento de tempo foi determinado uma quantidade de cinco scans no programa, número que foi considerado suficiente para levar os estados das células da ladder para um estado estável.

Após realizada esta etapa inicial, o programa faz uma varredura em todas as células da Ladder atualizando os estados destas de acordo com a linha de tempo da simulação. Em seguida são aplicados cinco scans e incrementado o valor do relógio do tempo. Após esta etapa os valores obtidos das saídas devem formar a linha do tempo das saídas para aquele instante.

O processo começa então a se repetir: novamente os estados das entradas são atualizados de acordo com a linha do tempo das entradas, e é realizado o ciclo de 5 scans. Logo em seguida o relógio de tempo é incrementado. Esse incremento no relógio do tempo está diretamente relacionado com o uso de temporizadores ou contadores na Ladder. Durante os ciclos de scans não há contagem de tempo pois, na prática, o scan de um CLP real leva um tempo da ordem de micro-segundos. Durantes esses scans, os temporizadores não são incrementados, apenas os atributos de entrada, estado e saída são alterados.

No momento em que o relógio de tempo do sistema é incrementado, o programa faz uma varredura pela matriz que contem a Ladder e incrementa o valor de todos os temporizadores ou contadores que devem ser incrementados, ou seja, aqueles que estão em estágio de contagem.

Outro elemento importante para o correto funcionamento da simulação é o elemento “jump” (mesmo elemento gráfico de um “fio” com a diferença de ter uma referência de posição). Ele se comporta como um fio pois apenas conduz energia, porém alem de conduzir energia de sua entrada para sua saída ele também pode energizar outros pontos da ladder que possui sua mesma referência. Dessa forma quando a simulação encontra um elemento “jump” que acabou de receber energia, uma varredura em toda a matriz é feita para alterar o valor do estado do “jump” para 1, indicando que em algum lugar do programa um “jump” com aquela referência está energizado. No momento de análise de um jump, o estado de sua saída é determinado através da verificação de sua entrada, ou de seu estado.

A simulação chega ao fim quando a linha do tempo das entradas termina. O processo de simulação consiste então em possuir uma Ladder, um conjunto de estados iniciais de entradas e saídas, e as linhas do tempo das entradas. Como resultados são obtidos as linhas do tempo das saídas.

O fitness dos indivíduos é calculado pela comparação do conjunto de linhas do tempo gerado pelo indivíduo e um conjunto de linhas do tempo padrão que determina o objetivo final a ser alcançado.

Em todos os testes realizados na publicação desse trabalho foi utilizado elitismo; cross-over com dois pontos de secção para escolha do segmento de código a ser transferido entre os indivíduos; e o operador de mutação foi aplicado em um único elemento da matriz que armazena a Ladder em cada vez que ele é executado. O operador de reprodução copia o indivíduo para próxima geração sem fazer qualquer alteração.

4.2.2. Simulação Computacional de uma Instruction List

Toda linguagem Ladder possui sua correspondente na linguagem Instruction List, e vice versa. No decorrer da pesquisa verificou-se que a simulação de uma Instruction List é

mais rápida que da Ladder pelo fato de não apresentar correspondências geométricas em seu código que exigem uma maior quantidade de loops para o reconhecimento dessas posições. A semelhança da linguagem Instruction List com à linguagem assembly já indica que ela é mais fácil de ser processada até mesmo por computadores mais antigos.

É possível criar um programa com a seguinte estrutura de variáveis para a simulação de uma instruction list, em qualquer linguagem estruturada. A seguir são apresentados seguimentos de código em pascal (linguagem utilizada na pesquisa):

a) Uma matriz de registradores para armazenamento da lógica do circuito.

Reg: array[1..500] of boolean;

Assim é possível acessar a memória de cada registrador utilizando a referência: Reg[1], Reg[2], Reg[3], ..., Reg[500]. A medida que o programa encontra as instruções LD ou LDI o índice do registrador é aumentado e a lógica é armazenada no próximo registrador da seqüência. Da mesma forma, à medida que o programa encontra as instruções ORB ou ANB, esse índice é decrementado para união de blocos lógicos.

b) Uma variável inteira para armazenar a posição do registrador corrente.

i: integer;

c) Matrizes booleanas para registrar os valores das entradas e das saídas.

Entrada: array [1..10] of boolean;
Saída: array [1..10] of boolean;

Como todos os elementos de uma Instruction List já foram apresentados, sua simulação será mais facilmente compreendida através da análise dos exemplos a seguir:

Instruction List 1

1: LD X001

```
2:  OR      X004
3:  LD      X002
4:  AND     X003
5:  LDI     X005
6:  AND     X006
7:  ORB
8:  OR      X007
9:  ANB
10: OR      X010
11: OUT     Y001
```

No início da simulação da Instruction List 1 uma posição de memória no meio da matriz booleana (i:= 250) deve ser definida como registrador corrente. Dessa forma quando o programa encontrar a referência “Reg[i]” irá armazenar o valor em “Reg[250]”.

A primeira linha do programa indica que o valor de X001 deve ser armazenado no registrador. Então:

Reg[i] := X001

Que representa o mesmo que: Reg[250]:= X001;

Em seguida é executada a operação OR entre o registrador corrente e a entrada X004:

Reg[i]:= Reg[i] OR X004;

Na linha 3 o ponteiro do registrador deve ser incrementado pois nesse momento é criado um novo bloco lógico.

i:= i+1;
Reg[i]:= X002

Assim o Reg[251] é que assume o valor. Logo após é feita a instrução AND:

Reg[i]:= Reg[i] AND X003

Na linha 4 novamente o ponteiro de registrador é incrementado:

i:= i+1;
Reg[i]:= NOT X005

O “Reg[252]” recebe o valor invertido da entrada X005. Logo após é realizado a instrução AND entre esse registrador e a entrada X006.

Reg[i]:= Reg[i] AND X006

Na linha 7 é encontrada a instrução ORB. Nesse momento o ponteiro de registrador é decrementado (i:= i-1) tornando como registrador corrente o “Reg[251]”. Essa operação é apresentada a seguir:

i:= i-1;
Reg[i]:= Reg[i] OR Reg[i+1];

Nesse momento são conectados dois blocos lógicos em paralelo, os blocos “Reg[251]” e o “Reg[252]”. A linha 8 realiza a operação OR entre o “Reg[251]” e a entrada X007. Em uma Ladder a entrada X007 é representada em paralelo com o ramo lógico anterior. A representação em ladder da operação é apresentada na Figura 25:

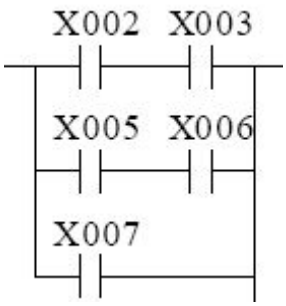


Figura 25 – Conexão dos blocos paralelos

A linha 9 representa a operação ANB que decrementa o ponteiro de registrador e executa a operação AND entre o registrador corrente e o próximo:

i:= i-1;
Reg[i]:= Reg[i] AND Reg[i+1];

Esse comando realiza a conexão em uma ladder representada na Figura 26.

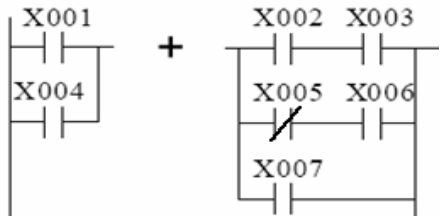


Figura 26 - Conexão de blocos em série

A seguir, na linha 10, é executada uma operação OR entre todo esse bloco lógico e a entrada X010.

Reg[i]:= Reg[i] OR X010;

E finalmente, a última linha coloca o valor final de toda lógica na saída Y001.

Saída[1]:= Reg[i];

Para finalizar a teoria de analise de uma instruction list é apresentado um segundo exemplo (Figura 27) para demonstrar a presença de complexas ramificações:

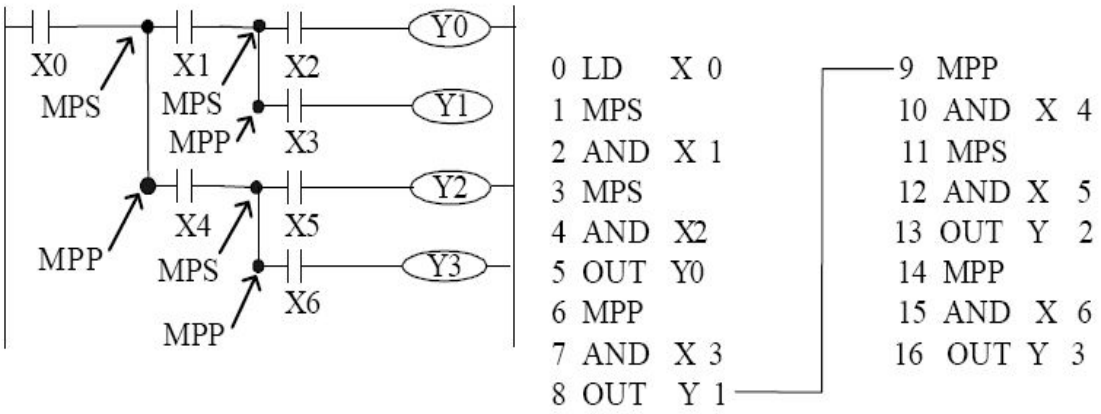


Figura 27 – Segundo exemplo de programa em instruction list
Fonte: WEG – manual de programação do TPW-03.

A simulação de uma instruction list com os comandos MPS, MRD e MPP exige a criação de uma pilha para armazenamento dos valores das ramificações dos circuitos. Uma pilha é uma lista de valores que pode receber e fornecer valores, e o valor fornecido por ela é sempre o último valor que foi colocado.

Dessa forma a linha 0 da instruction list acima representa o carregamento do Reg[250] pelo valor da entrada X0. A instrução MPS da linha 1 armazena na pilha o valor do registrador corrente (nesse caso o valor de X0).

Seguindo para a linha 2 vê-se que é executada a instrução AND entre o Reg[250] e a entrada X1.

Na linha 3 é encontrada outra instrução MPS que insere na pilha (Figura 28) o novo valor do Reg[250] que corresponde à X0 AND X1 como pode ser visto na figura a seguir:

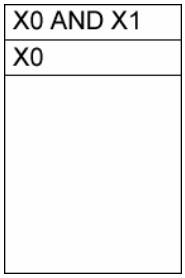


Figura 28 – Pilha de memória

Na linha 4 é executado a instrução AND entre o registrador corrente e a entrada X2 e em seguida o valor do resultado é colocado na saída Y0 pela instrução OUT da linha 5.

Encontra-se agora a instrução MPP que move o primeiro valor da pilha para o registrador corrente. Dessa forma agora o Reg[250] recebe o valor X0 AND X1, e na pilha resta apenas o valor X0.

A linha 7 executa a instrução AND com a entrada X3 e logo em seguida o valor da operação é colocando na saída Y1 (instrução OUT, linha 8).

Novamente é encontrada uma instrução MPP na linha 9. O valor da pilha é então movido para o registrador corrente (Reg[250] = X0). A linha 10 executa uma AND com a entrada X4 e a linha 11 apresenta outra instrução MPS. A pilha agora adquire o valor X0 AND X4.

A linha 12 executa uma AND entre o registrador e a entrada X5 e a linha 13 coloca o resultado na saída Y2.

A linha 14 recupera o valor X0 AND X4 da pilha e a linha 15 executa uma AND entre o registrador e a entrada X6. Logo em seguida o valor é apresentado na saída Y3 na linha 16.

Com os conceitos apresentados nos dois exemplos acima é possível criar e simular complexas redes lógicas através de uma instruction list.

4.2.3. Potencial das linguagens de programação de CLPs

A linguagem Ladder pode dar a impressão de ser limitada e às vezes nem ser considerada como uma real linguagem de programação em alguns CLPs mais simples. Porém em CLPs de médio e grande porte essa linguagem adquire o potencial semelhante a das linguagens de computador como C e pascal.

O controle do programa de um CLP não se resume em apenas contatos NA, NF, bobinas, temporizadores e contadores. Alguns CLPs (WEG, 2006; WEG, 2008), (Rockwell, 2004; Rockwell, 1998), (Siemens, 2005) possuem funções complexas exercidas por bobinas de função que exercem o efeito de comandos como While..Do, For...Next, If, Case. Também é possível montar rotinas dentro de um CLP usando uma bobina CALL (comando semelhante ao assembly) (Figura 29).

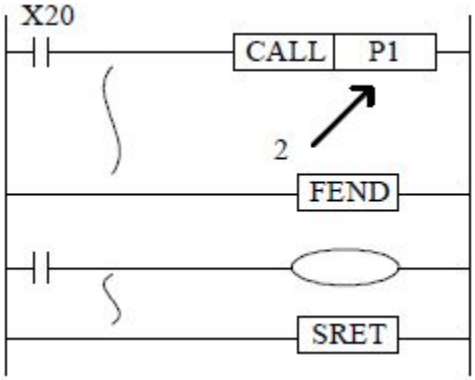


Figura 29 – Comando CALL

Além das instruções de controle de fluxo do programa é possível armazenar dados no CLP em forma de representação binária para armazenar: números em diferentes bases, números inteiros e de ponto flutuante, números em notação científica.

Também é possível realizar operações matemáticas em alguns CLPs pela presença de instruções de adição, subtração, multiplicação e divisão. Os CLPs também chegam a ter instruções matemáticas de trigonometria como seno, cosseno e tangente (WEG, 2008).

Alguns CLPs também possuem instruções mais complexas (semelhantes ao assembly) como o de rotação para direita e para esquerda de um número binário.

Dessa forma é possível perceber que um CLP realmente se comporta como um computador por possuir uma linguagem bastante flexível e não apenas uma peça seqüenciadora de contatos.

4.3. Teoria da Programação Genética

4.3.1. Princípios e Definições

Seguindo o raciocínio do projeto, após a apresentação das teorias de automação necessárias, é apresentada a seguir a parte da pesquisa referente à programação genética que é a técnica de inteligência artificial utilizada para gerar os programas de CLP.

A programação genética trata da questão de como fazer computadores resolverem problemas sem que eles sejam explicitamente programados, ou seja, sem que seja estabelecida forma, tamanho, e complexidade estrutural. Essas características devem surgir durante o processo de solução do problema como resultado das demandas dele. O tamanho, a forma e a

complexidade estrutural devem ser partes da resposta produzida pela técnica de solução do problema e não parte da questão.

Em 1950, Arthur Samuel (SAMUEL, 1963) formulou as seguintes perguntas:

"Como os computadores podem aprender a solucionar problemas sem serem explicitamente programados?"

"Em outras palavras, como fazer com que computadores façam o que deve ser feito sem que recebam a informação de como eles exatamente precisam fazer ?"

Um dos impedimentos encontrados na solução desses problemas pelos métodos de inteligência artificial conhecidos como: o aprendizado de máquina, os sistemas de auto-aperfeiçoamento, os sistemas de auto organização, as redes neurais e indução; é que os mesmos não buscam a solução do problema através da forma de programas de computadores. Esses métodos são baseados em estruturas especializadas e não em estruturas livres como as linguagens de programação de computadores.

Os programas de computador oferecem flexibilidade para realizar operações de forma hierárquica, fazer cálculos condicionais intermediários a outros cálculos, realizar iterações e recursões, utilizar variáveis de diferentes tipos e definir valores intermediários e sub-programas que podem ser subsequentemente reutilizados (KOZA, 2000).

Através dessa visão de que apenas uma estrutura livre e complexa como os programas de computador podem solucionar o problema proposto, encontra-se um novo obstáculo: o vasto espaço de soluções. O fato dos programas de computador poderem representar praticamente qualquer problema gera a questão de como encontrar a solução correta nesse imenso espaço de soluções.

Um método de busca de soluções conhecido como Hill Climbing (KOZA, 2000) parte de uma solução aleatória inicial e testa soluções na vizinhança da mesma. Quando uma dessas soluções vizinhas apresenta um melhor desempenho então uma nova posição do espaço de soluções é assumida e uma nova vizinhança de busca passa a ser vasculhada. Essa é uma forma de busca inteligente e adaptativa, ao contrário de uma busca aleatória cega que testa caso a caso todos os elementos do espaço. O problema nesta técnica é que em problemas não lineares o sistema de busca possui o defeito de acabar ficando preso em algum ponto ótimo local ao invés de convergir para um ótimo global.

O poder dos métodos de inteligência artificial costuma ser subestimado pelo meio leigo. Um exemplo a ser analisado é o de construção e correção de um programa de computador. É comum pensar que apenas uma mente humana pode realizar tal tarefa, pois a correção de erros em um programa é algo não linear e que também não é uma atividade

mecânica. Porém através da técnica da programação genética isso pode ser feito (KOZA, 2000).

O fato de a técnica ser baseada em construções aleatórias e ser predominantemente probabilística torna difícil uma prova matemática de que a solução pode ser sempre encontrada para qualquer problema. A utilização da técnica é baseada em evidências empíricas. Em contra partida, é possível fornecer um grande número de resultados e experimentos que demonstram a capacidade desta técnica em solucionar problemas.

4.3.2. Os princípios da Programação Genética e das pesquisas convencionais

Existem sete princípios comuns à maioria das pesquisas (KOZA, 2000), que são o uso de abordagens: corretas, consistentes, justificáveis, determinísticas, ordenáveis, parcimoniosas (econômicas) e decisivas. A programação genética funciona fora desses sete princípios. De forma mais aprofundada estes princípios podem ser explicados da seguinte maneira:

Correto: na matemática, na ciência e na engenharia geralmente a solução correta para o problema é o objetivo buscado, pois esta oferece uma utilidade prática. Porém é comum notar que as máquinas de cálculo introduzem imprecisões nos resultados, e durante provas analíticas de fórmulas matemáticas muitas simplificações podem ser feitas, e em cálculos de engenharia muitas aproximações são aceitáveis. Através desses fatos vê-se que apesar da resposta exata não ser encontrada, o fato de a solução estar próxima do exato já confere a ela uma função prática para a realidade. Um exemplo apresentado em (KOZA, 2000) é a seguinte solução para uma equação a segundo grau $ax^2 + bx + c = 0$:

$$X = \frac{-b + \sqrt{(b^2 - 4ac)}}{2a} + 0.000000000000001a^3bc$$

Esta fórmula representa um erro no cálculo da raiz de uma equação de segundo grau, porém é aceitável para a prática. A programação genética trabalha apenas com soluções incorretas admissíveis e apenas apresenta a correta solução analítica casualmente.

Consistente: a programação genética encoraja, preserva e usa conjuntos de abordagens inconsistentes e contraditórias em suas tentativas para solucionar os problemas. Algo que é tipicamente inaceitável para uma mente lógica na ciência convencional.

Justificável: a programação genética não produz suas respostas através de um fluxo de raciocínio lógico.

Determinístico: na programação genética tudo pode acontecer e nada é garantido, pois todos os passos chave na evolução da solução são probabilísticos.

Ordenamento: nas ciências convencionais os procedimentos são geralmente ordenados, controlados e sincronizados. A desordem é um fator central nos processos biológicos que operam na natureza, e a programação genética segue o mesmo princípio.

Parcimônia (economia): a ciência atual está em busca de soluções simples e econômicas. A programação genética produz soluções que geralmente são complexas e apresentam redundâncias.

Decisivo: é comum a busca por um ponto definitivo de uma solução, um momento em que a mesma converge e o processo se acaba. Os processos biológicos não possuem geralmente um final claramente definido, eles permanecem em constante mudança.

A natureza cria complexas estruturas ao longo do tempo sem obedecer aos sete princípios acima; nesse caso, o importante para a sobrevivência é a aptidão, ou fitness. É através desse fator que ao longo do tempo através da seleção natural que estruturas funcionalmente melhores do que as outras sobrevivem e geram seus descendentes, que por sua vez podem ser mais ou menos aptos. A natureza mantém estruturas inconsistentes e contraditórias e de fato, a manutenção da diversidade é um ingrediente importante para assegurar a futura habilidade dos indivíduos a se adaptarem a mudanças no ambiente.

Não há prova matemática no sentido de justificar a diferença de desenvolvimento entre as estruturas dos seres vivos observados hoje e seus ancestrais ou de demonstrar a seqüência de regras lógicas que foram aplicadas a estes para demonstrar os resultados a partir de suas premissas. O processo evolucionário na natureza é incerto e não determinístico, além de envolver atividades assíncronas, não coordenadas, locais e de atividade independente, sem um controle central (KOZA, 2000).

4.3.3. Funcionalidade e os Introns

Como dito acima, o que predomina na evolução é o fitness e não a parcimônia. Uma vez que a natureza encontra uma solução para o problema ela normalmente a protege, mesmo que não seja uma solução ótima. A parcimônia apenas interfere quando ela faz parte do fitness do indivíduo. Um mesmo programa de computador pode ser escrito com 10 linhas de código ou com 200 linhas. Se houver espaço suficiente, o tamanho deste não fará diferença, porém se houver um número de linhas muito limitado então a economia de espaço passa a fazer parte da definição do fitness.

O objetivo principal da técnica é obter **capacidade funcional** e é a partir daí que é possível fazer uma otimização do que já está em funcionamento. É através deste princípio que é baseado o uso do elitismo nas simulações. Em problemas muito complexos, em que o espaço de soluções é praticamente infinito e na prática existem poucas soluções possíveis, de uma geração para outra o indivíduo com maior fitness é guardado de forma irrestrita. Às vezes é preciso esperar muitas gerações até que o processo evolucionário encontre uma alternativa para o problema, portanto, perder uma informação dessas seria retroceder no tempo.

Quando uma solução é encontrada normalmente ela possui em si estruturas não utilizadas e redundantes. Exemplos matemáticos desse fato podem ser:

- Sucessivas multiplicações de um número por 1.
- Sucessivas somas com zero.
- Divisões por 1.
- Somas e subtrações de um mesmo número.
- Multiplicações e divisões pelo mesmo número.
- Operações OR e AND de uma variável com ela mesma.

Em biologia essas estruturas não usadas e sem funcionalidade aparente são conhecidas como **introns** (BANZHAF et al., 1998). Em 1994, Angeline observou nos exemplos da obra de Koza estranhos segmentos de código que podiam ser retirados sem alterar a solução. Ela foi a primeira pesquisadora a associar o conceito de “códigos extras” na programação genética ao conceito de introns da biologia (ANGELINE, 1994).

Outro fato obsevado por (TACKETT, 1994) é o fato das soluções dos indivíduos crescerem até atingirem o limite máximo permitido, efeito chamado “hitchhiking” (em português, “pegar carona”) por Tackett, ou “bloat”, inchamento. Estudos em 1995 e 1996 sugeriram que do início ao meio do processo de simulação da programação genética, de 40% a 60% de todo o código produzido é formado por introns (BANZHAF et al., 1998). Mais a frente na simulação, os introns tendem a crescer exponencialmente e passam a abranger quase toda a população, restando apenas poucos indivíduos soluções com funcionalidade útil (NORDIN et al., 1995), (NORDIN et al., 1996), (SOULE et al., 1996), (SOULE; FOSTER, 1997), (ROSCA, 1997). É importante enfatizar que essas são constatações para simulações de programação genética, pois, em casos de algoritmo genético e outras formas de computação evolucionária, o comportamento não precisa necessariamente obedecer a estas constatações.

4.3.4. Condições para a evolução

O uso freqüente dos sete princípios apresentados anteriormente geralmente são tão usados que passa-se a assumir de forma inquestionável que eles são realmente necessários para todo problema científico. A abordagem desta técnica simplesmente coloca de lado estas regras e passa a adotar os princípios reais que acontecem na natureza.

Em (KOZA, 2000) definem-se quatro condições na natureza para que a evolução ocorra:

- A entidade possui a habilidade de se reproduzir.
- Existe uma população de entidades auto reprodutoras.
- Existe variedade entre essas entidades.
- Diferenças na habilidade de sobreviver ao ambiente estão associadas com a variedade.

(BANZHAF et al., 1998) também define quatro condições para a ocorrência da evolução:

- Reprodução de indivíduos em uma população.
- Variação que afeta a maneira de sobrevivência dos indivíduos.
- Hereditariedade na reprodução.
- Recursos finitos causando a competição.

Estes são conceitos expressos na teoria da evolução e seleção natural de Charles Darwin (DARWIN, 1859). Através de um período de tempo e muitas gerações, a população como um todo passa a conter mais indivíduos cujos cromossomos são traduzidos em estruturas e comportamentos que os habilitam a melhor desempenhar suas tarefas, a reproduzir e a sobreviver.

A teoria de Darwin que surgiu em 1859 entrou para o campo da informática em 1975 por John Holland em sua obra “Adaptation in Natural and Artificial Systems” (HOLLAND, 1975). Nesse momento os primeiros conceitos de evolução e adaptabilidade em programas de computador passaram a ser implementados.

4.3.5. Implementação da programação genética

Existem diversas formas de se implementar um programa com a técnica da programação genética pois não há uma forma fixa, o importante é respeitar os princípios teóricos. Existem varias formas de se lidar com a população de programas gerados através dos parâmetros de simulação.

Dois algoritmos genéticos básicos são apresentados nas Figura 32 e Figura 33. Observa-se que a seqüência básica de funcionamento é a seguinte:

- **Passo 1 -** Criação de uma população inicial de indivíduos.
- **Passo 2 –** Cálculo do Fitness de cada indivíduo.
- **Passo 3 –** Verificar se o critério de término do algoritmo foi obtido. Se “sim” ocorre o fim do processo / se “Não” segue-se para o próximo passo.
- **Passo 4 –** Gerar nova população obedecendo aos critérios probabilísticos de reprodução. Volta ao passo 2.

Os indivíduos de uma população de programas são formados por **funções** e **terminais**. Os terminais são valores constantes, valores de entrada do programa ou funções sem argumentos. Eles são chamados de terminais, pois finalizam os ramos de um algoritmo estruturado em forma de árvore (BANZHAF et al., 1998). Os terminais são agrupados em um conjunto que o programa usará para fazer escolhas aleatórias ao montar a população inicial de indivíduos.

Exemplo de terminais seriam valores inteiros (1, 2, 3), reais (1.2, 5.3, 7.987, 2.1), booleanos (0, 1), etc.

As funções constituem os nós de um programa estruturado em árvore, servindo como meio de conexão de funções com funções, e de funções com terminais. Alguns exemplos de funções são:

- Funções booleanas: and, or, not, xor.
- Funções aritméticas: +, -, x, /.
- Funções transcendentais: funções trigonométricas e logarítmicas.
- Funções de recepção de valores
- Declarações condicionais: if, then, else, case.
- Declarações de controle de fluxo: go to, call, jump.
- Declarações de loop: while, repeat, for.
- Sub-rotinas: virar a esquerda, andar para frente, ler sensor.

Estes elementos (terminais e funções) podem ser organizados em três tipos de estrutura de programa, conhecidas como:

- Estrutura em árvore:

Organiza o código do programa graficamente em forma de nós conectados em formato de árvore (Figura 30), ou em forma de linha de texto, comumente utilizada na programação LISP.

Exemplos:

- Em linha: (A and B or (C and not (D or A))) or C

- Em gráfico é apresentado o mesmo código da linha acima (Figura 30):

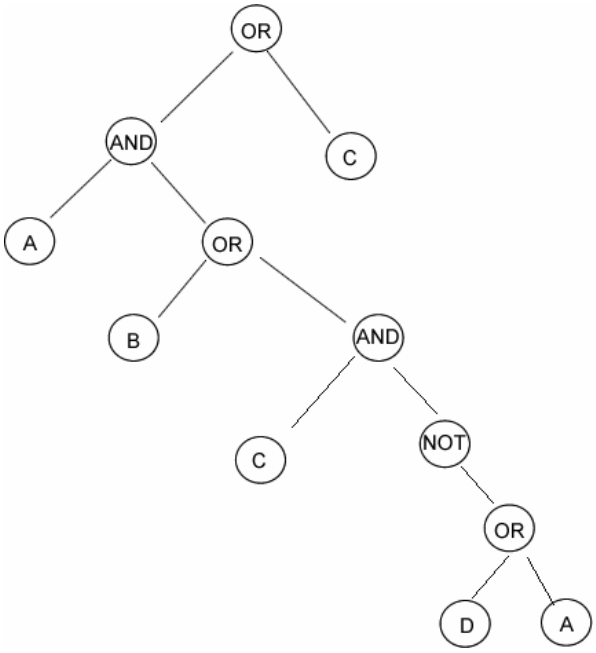


Figura 30 – Estrutura de código em árvore

- Estrutura Linear:

Organiza o código em uma corrente de instruções que são executadas da esquerda para a direita, ou de cima para baixo. O exemplo a seguir apresenta o mesmo código da Figura 30.

Exemplo:

D:= D or A

D:= not D

C:= C and D

B:= B or C

A:= A and B

A:= A or C

- Estrutura em Grapho:

A estrutura grapho é a mais complexa, chamada de PADO por (TELLER; VELOSO, 1995). Ela representa nós conectados entre si indicando o fluxo do programa, e é capaz de representar loops e recursão, algo que é feito com mais dificuldade pelos outros sistemas. Na Figura 31 é representado um programa PADO.

A leitura do programa se inicia pelo nó “Start” e quando o sistema atinge o nó “End” o processo termina. A execução do programa é feita seguindo as setas. Esse tipo de programa necessita de uma memória (RAM) para dar os valores das variáveis dos nós.

Além desta, é necessária uma memória do tipo pilha para o transporte de valores entre as variáveis. Cada nó executa uma função que lê ou escreve um valor na pilha. No programa abaixo o nó “A” recebe seu valor da memória RAM e o escreve na pilha (elemento “push” fig. 31). O nó 6 escreve seu valor na pilha. O nó X retira dois valores da pilha, os multiplica, e devolve o resultado para a pilha.

O sistema também possui uma terceira memória (memória indexa). O nó “Write” retira dois valores da pilha e escreve o primeiro valor na posição de memória indicada pelo segundo valor. O nó “Read” realiza praticamente o mesmo só que o inverso (leitura).

Cada nó do grapho deve:

- Realizar uma função relacionada com a pilha ou com a memória indexa.
- Decidir qual será o próximo nó a ser executado.

A escolha do próximo nó a ser executado é realizado através de um teste feito nos valores de memória da pilha ou da memória indexa.

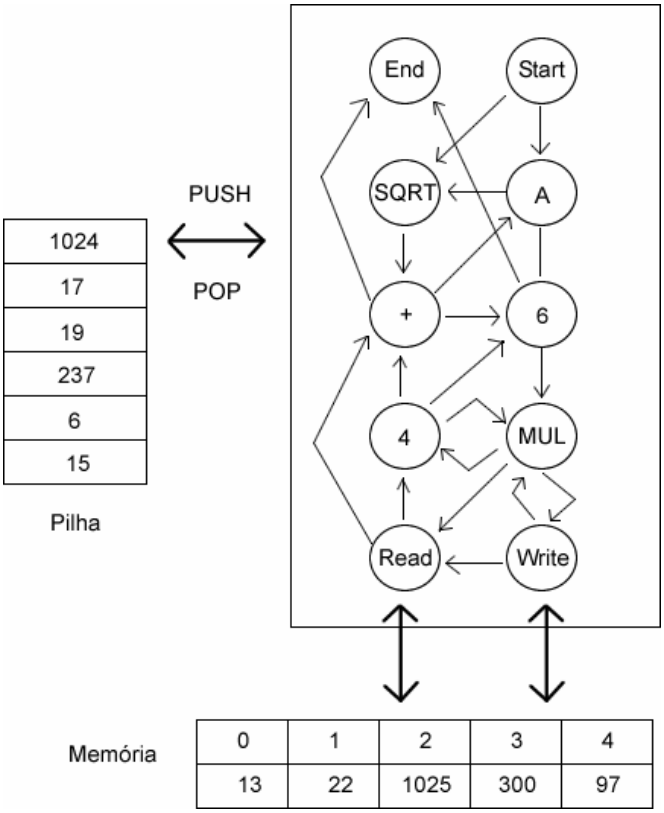


Figura 31 – Programa PADO

Essas três estruturas apresentadas são convenções que podem ser alteradas com flexibilidade de acordo com as necessidades do programa. Em um sistema PADO a velocidade do algoritmo é sacrificada, porém estruturas em árvore ou lineares não conseguem evoluir sua própria ordem de execução.

O programa desenvolvido pela pesquisa utiliza a estrutura linear, pois é a forma como uma Instruction List é representada.

Outra etapa essencial para a programação genética é o processo de reprodução. Internamente a este processo existem duas etapas básica: a seleção e o tipo de reprodução.

a) Métodos de Seleção

Os métodos de seleção mais comuns são: o **método de reprodução proporcional** ao fitness (Figura 32) e o **método de reprodução por torneio** (Figura 33).

Na **reprodução proporcional** ao fitness todos os indivíduos da população possuem a chance de se reproduzirem, e a probabilidade de isso ocorrer é proporcional ao seu número de fitness.

A desvantagem desse método é a geração de um número não inteiro de descendentes. Esse problema pode ser resolvido através de uma “roleta” ou “gráfico em pizza”. Nesta roleta cada indivíduo possui uma área proporcional ao seu fitness e um número aleatório de 0 a 100 é gerado de forma que os indivíduos são pouco a pouco escolhidos até que o número limite seja alcançado.

Outro problema é gerado pela necessidade de padronizar os problemas sendo necessário tratar diferente os problemas em que o fitness buscado é o maior possível e os problemas em que o fitness buscado seja o menor possível. Teoricamente fitness significa nível de aptidão e o número máximo é sempre o almejado, porém na prática das simulações em algumas situações o número prático que se deseja obter é o menor possível. Nesses casos é necessário fazer uma transformação matemática para que indivíduos com “pontuação” baixa se tornem indivíduos com alto fitness e os com alta “pontuação” se tornem indivíduos com baixo fitness.

Essa forma de seleção também tende a fazer o algoritmo a ficar preso em máximos locais. Nesse caso indivíduos com fitness alto, mas que ainda estão longe do fitness máximo global, reduzem muito a chance de que um indivíduo atual com fitness baixo mais com maior potencial de atingir a solução ótima se reproduza.

Essa forma de reprodução perde o poder de seletividade nas fases finais do processo pois os valores dos fitness dos indivíduos se tornam muito próximos.

Na **reprodução por torneio** um número pequeno de indivíduos (2, 3 ou 4) é escolhido aleatoriamente com a mesma probabilidade e então os indivíduos são comparados entre si e não perante a população. O indivíduo com maior fitness é escolhido. Esse método de seleção soluciona os problemas da seleção proporcional, é computacionalmente mais rápido e mais simples de ser implementado pela ausência da roleta. A abordagem nos métodos de maximização e minimização de fitness é a mesma, mudando apenas no momento da comparação dos poucos indivíduos escolhidos.

A escolha dos indivíduos deixa de ser exclusivamente pelo valor relativo entre os indivíduos da população. A sobrevivência do indivíduo mais apto permanece, mas a variabilidade aumenta pela escolha de indivíduos mais aptos em grupos mais dispersos da população.

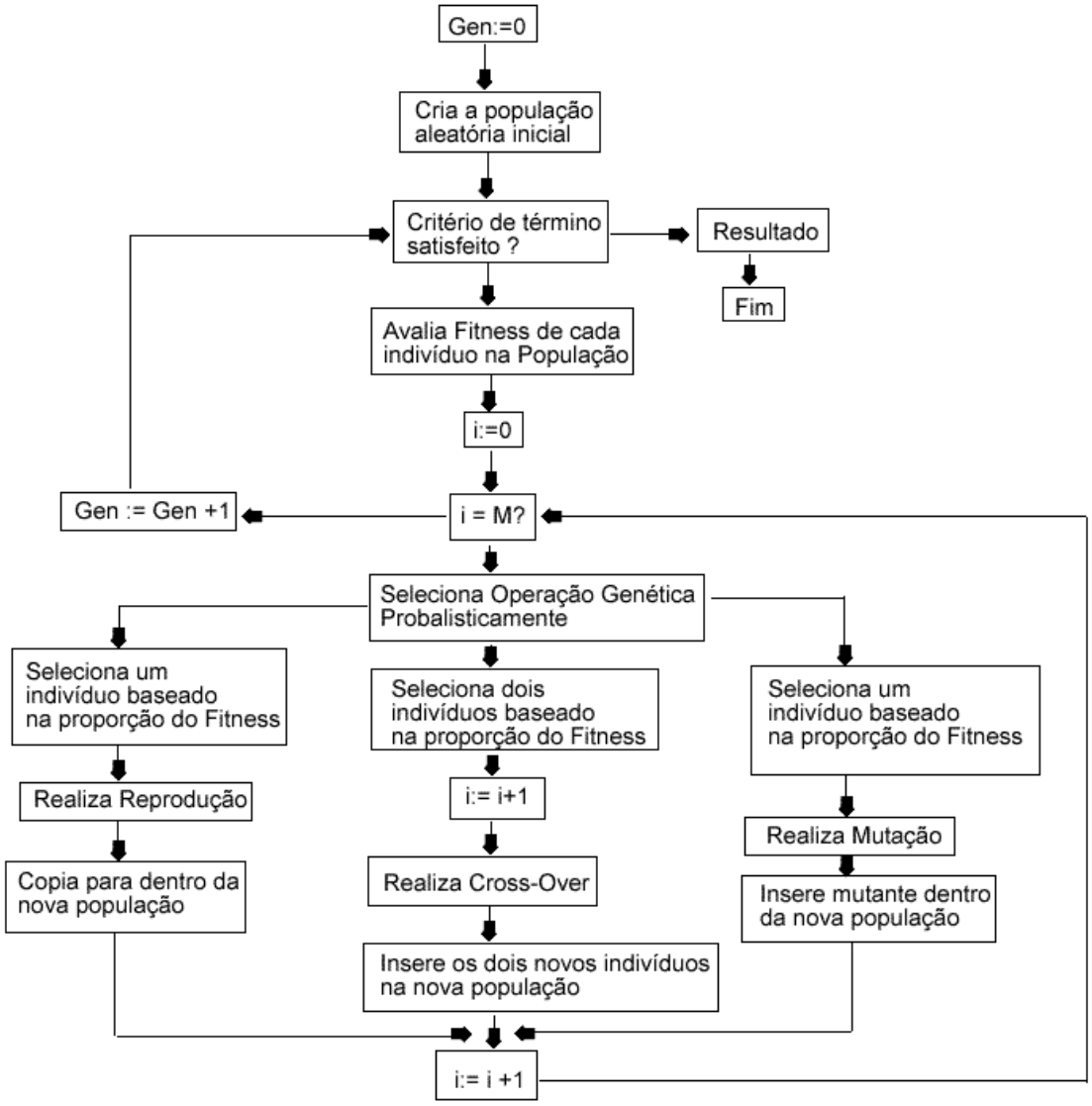


Figura 32 – Fluxograma do programa com seleção proporcional

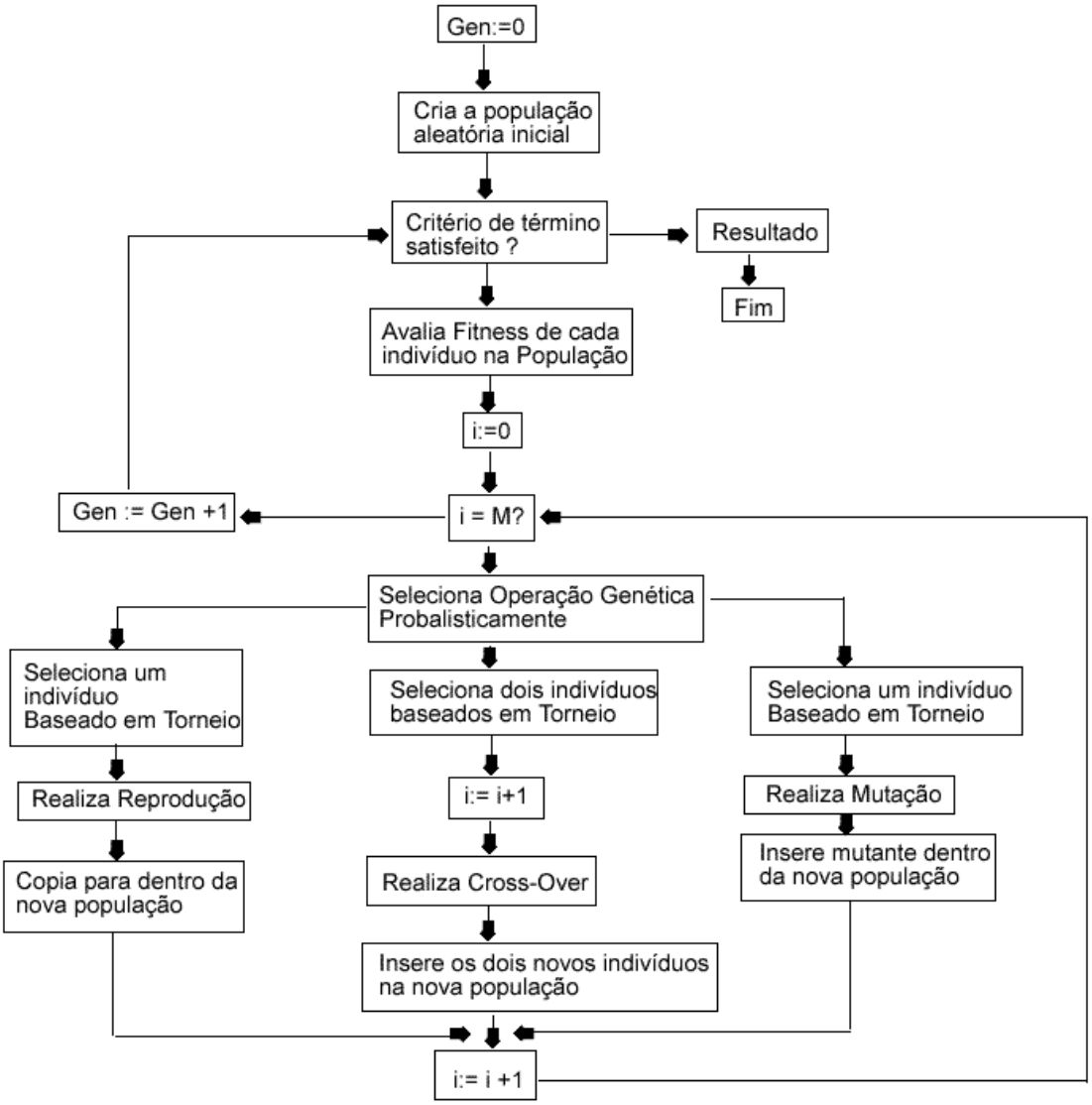


Figura 33 – Fluxograma do programa com seleção por Torneio

b) Tipos de reprodução (operadores de busca)

Outra etapa importante do algoritmo é a escolha do tipo de reprodução que eles irão sofrer, ou os operadores de busca que serão utilizados. Uma população inicializada normalmente possui um nível de fitness muito baixo, a evolução ocorre pela transformação dessa população inicial pelo uso desses operadores. Essas formas de reprodução são: o cross-over, a mutação e a reprodução.

O **cross-over** combina o material genético de dois pais pela troca de um segmento de um pelo segmento de código do outro, gerando dois indivíduos novos. A Figura 34 exemplifica a reprodução em um algoritmo com estrutura em árvore e a Figura 35 exemplifica a reprodução em uma estrutura linear.

O modelo de cross-over utilizado no programa desenvolvido nessa pesquisa é realizado em dois pontos para escolha do segmento do código a ser transferido, da mesma forma como é ilustrado na Figura 35.

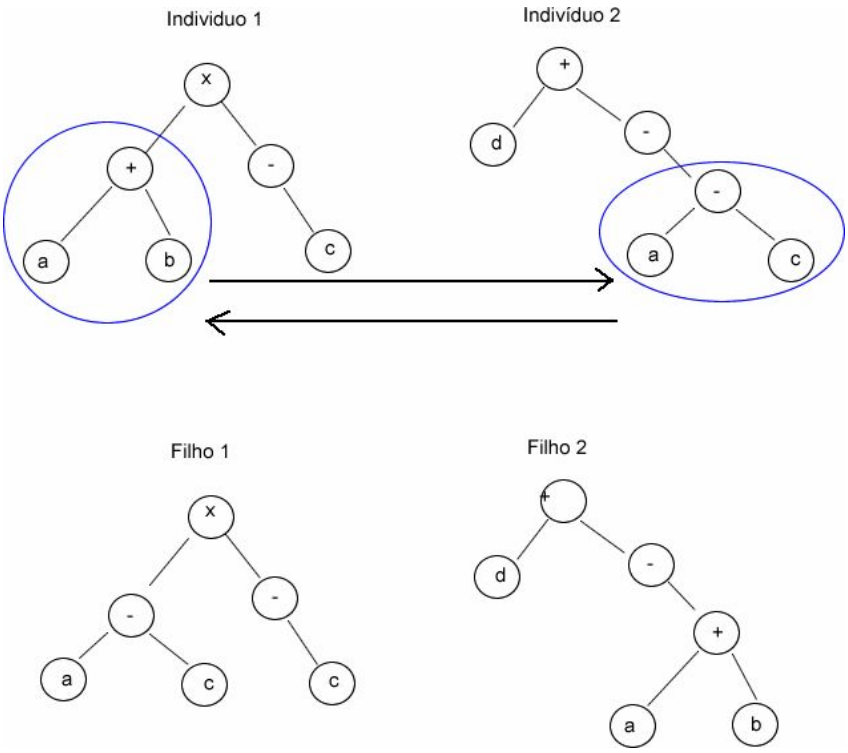


Figura 34 – Reprodução em estrutura árvore

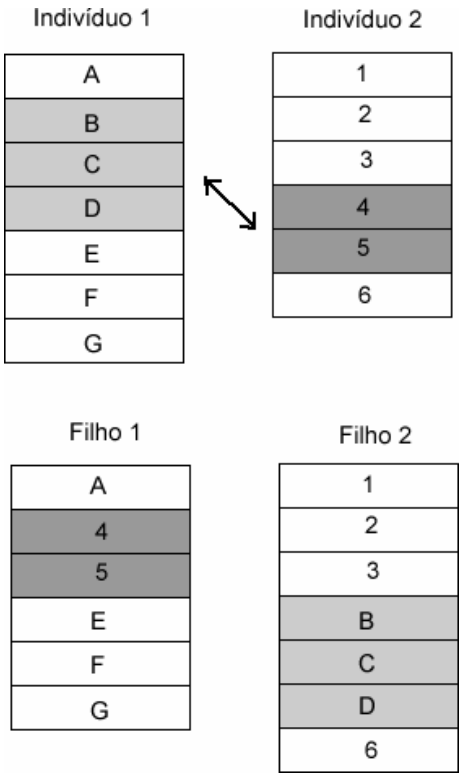


Figura 35 – Reprodução em estrutura linear

O operador genético **mutação** é realizado sobre um indivíduo apenas. Ele pode ser usado após o cross-over ter ocorrido em uma parcela dos filhos produzidos sob uma baixa probabilidade, ou ser usado separadamente sem que um cross-over tenha ocorrido.

Quando utilizado em uma estrutura em árvore um nó é escolhido aleatoriamente, podendo ser função ou terminal, e então seu valor é alterado também aleatoriamente. Em um sistema linear um exemplo de mutação pode ser visto abaixo:

Indivíduo Antes

X:= A + B

Indivíduo Depois

X:= A – B

Um processo análogo ocorre no sistema grapho.

Este operador é muito utilizado para aumentar a variabilidade de uma população, para retirar o algoritmo de algum mínimo local, ou para visitar a vizinhança de um ponto do espaço de soluções.

O modelo de mutação utilizado no projeto escolhe aleatoriamente uma linha do código da Instruction List do programa. Em seguida, aleatoriamente escolhe-se um comando dentre o seguinte conjunto {LD, LDI, LDP, LDF, ANDP, ANDF, ORB, AND, ANI, OR, ORI, ANB, OUTR, OUT}. Após a modificação do comando da linha é escolhido uma nova referência aleatória, podendo ser uma entrada, saída, contato auxiliar ou temporizador, de acordo com os elementos possíveis existentes no estudo de caso em específico.

No operador **reprodução** um indivíduo simplesmente é escolhido e copiado da mesma forma para a próxima geração. É normalmente usado em conjunto com o **Elitismo** que é a escolha do indivíduo com maior fitness na população e logo após copiado para a nova geração.

No projeto cada geração possui o tamanho M igual a 100 indivíduos. Durante a criação de uma próxima geração, são criados apenas mais 100 indivíduos.

4.3.6. População inicial e fechamento do programa

O primeiro passo de execução na programação genética é a criação de uma população inicial de indivíduos (passo genérico para todas as aplicações de PG). Nesse momento, independentemente do problema, os elementos de solução são todos escolhidos aleatoriamente e agrupados nos indivíduos gerando uma população com fitness muito baixo.

À medida que novas populações são geradas, outros códigos aparecem e são testados pelo programa. É essencial para o funcionamento da técnica que todos os indivíduos gerados, ou na população inicial ou pelos operadores genéticos, que eles sejam testáveis e que se seu código for de pouca utilidade que ele receba um baixo fitness e nunca trave o programa. Essa é a propriedade do fechamento.

Todas as soluções produzidas devem ser soluções em potencial. Durante a implementação é preciso ter o cuidado de gerar código extra que proteja e formalize a geração de indivíduos testáveis.

Para a criação de um indivíduo aleatório pelo programa da pesquisa, é levado em conta o número de saídas, entradas, contatos auxiliares e temporizadores disponíveis para a aplicação do estudo de caso em específico. Se no programa em busca existem apenas duas botoeiras disponíveis, o programa possui em seu conjunto de entradas apenas I1 e I2 (ou seja, uma variável de referência para cada botoeira).

Antes do início da busca, apesar do programa fazer uma busca genérica para qualquer tipo de indivíduo, conhece-se diversas características da futura solução, pois esta é dependente do sistema físico disponível. Dessa forma, em cada indivíduo da população inicial estão disponíveis para sua geração aleatória apenas uma quantidade de referências de saída correspondente ao número de saídas físicas que serão utilizadas.

Outra variável importante, presente nessa pesquisa, para a geração da população inicial é o tamanho do “corpo do programa”. Este número define quantas linhas de código aleatórias serão geradas para cada saída do programa. Portanto, em uma Instruction List para um sistema com duas saídas, e com “corpo de programa” igual a cinco terá todos os seus programas iniciais com doze linhas. Ou seja, cinco linhas aleatórias, mais uma linha com comando aleatório e uma referência de saída número 1; e mais 5 linhas aleatórias, e outra linha com a segunda saída.

Portanto, o corpo do programa é aleatório e seu tamanho inicial é dependente do número de saídas necessárias para a construção deste.

Esse processo aleatório se repete até que toda a população da geração inicial seja preenchida. Nos estudos apresentados foram utilizados 100 indivíduos em todas as populações.

5. Implementação do Projeto

Para a implementação deste projeto que visa a criação automática de programas de CLP através da programação genética, foi utilizado o **software Borland Delphi 7 Enterprise**. Este programa é uma plataforma de programação que utiliza a linguagem **Pascal** e foi criada com a intenção de facilitar a criação de programas.

O Delphi 7, através de sua programação orientada a objetos, proporciona ao programador blocos de código prontos em forma de objetos gráficos, como por exemplo, botões, caixas de texto, objetos para controle de gráficos e outros. Através desta plataforma de programação implementou-se em um mesmo programa o conjunto de códigos que representam o simulador dos programas de automação e o sistema de inteligência artificial.

A seguir procurou-se descrever de forma mais detalhada como o sistema foi programado com o objetivo de proporcionar para outros pesquisadores o máximo de idéias utilizadas no funcionamento do projeto. Acredita-se que com a compreensão de toda a parte teórica apresentada previamente e com as informações apresentadas a seguir seja possível a reconstrução deste projeto, ou de um similar com a mesma funcionalidade.

5.1 Matrizes das Linhas do Tempo

A peça chave deste projeto é a avaliação do nível de aptidão das soluções através das linhas do tempo. A simulação de um programa de automação exige linhas do tempo que representem o fluxo de dados ao longo do tempo que chega às entradas do CLP, e o resultado da simulação para ser descrito de forma detalhada também deve ser feito na forma de linhas do tempo.

As linhas do tempo foram representadas na memória do computador em matrizes booleanas com duas dimensões. Cada linha da matriz representa uma referência (entrada, ou uma saída, ou um contato auxiliar, ou um temporizador). Cada coluna representa o estado desta

referência no tempo. A Figura 36 é um exemplo de representação lógica de linhas do tempo que estão armazenadas em uma matriz. Na Figura 37 procurou-se representar uma matriz análoga à Figura 36 para facilitar a compreensão.

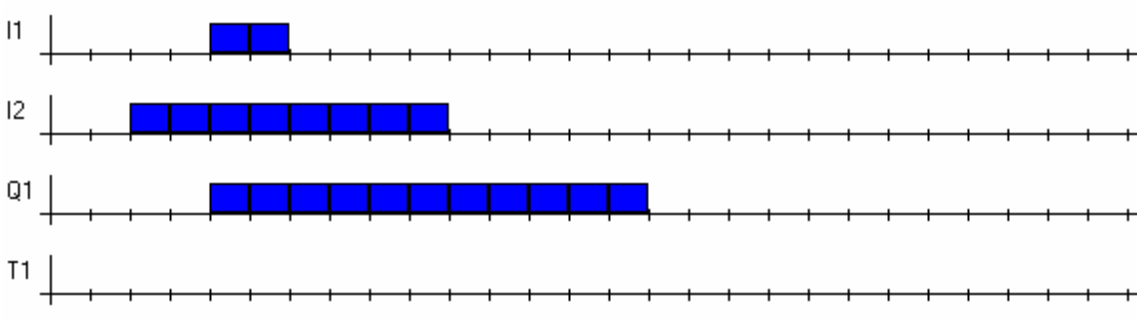


Figura 36 – Exemplo de linha do tempo teórica

0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 37 – Matriz representativa da um conjunto de linhas do tempo

Foi criado uma estrutura de dados para armazenar todas essas matrizes que representam as linhas do tempo através do seguinte código:

```
type
  LinhasDeTeste = record
    I: array [1..10,1..200] of boolean;
    Q: array [1..10,1..200] of boolean;
    M: array [1..10,1..200] of boolean;
    T: array [1..10,1..200] of boolean;
  end;
```

Esta estrutura de dados chamada de “LinhasDeTeste” armazena quatro matrizes: I, Q, M e T; Estas matrizes possuem dez linhas e duzentas colunas, sendo que cada célula da matriz representa uma variável booleana (verdadeiro ou falso). Quando uma dessas posições de memória é “verdadeira” ela indica que aquela referência naquela posição do tempo está energizada.

Cada linha da matriz “I” representa uma entrada, dessa forma o programa suporta a utilização de até dez entradas: I1, I2, I3, I4, I5, I6, I7, I8, I9, I10. E cada uma destas linhas do

tempo pode representar até duzentos estados diferentes ao longo do tempo. O mesmo raciocínio deve ser utilizado para compreender Q, M e T.

Com a utilização de uma estrutura “LinhasDeTeste” é possível simular um conjunto de situações contínuas no tempo para o sistema de automação. Passo a passo as matrizes de entrada de dados são lidas, repassadas para o simulador e os resultados deste são gravados nas matrizes Q (matrizes das saídas).

Para que varias situações possam ser simuladas criou-se um conjunto de “LinhasDeTeste“ como esta sendo representado no código abaixo:

```
var
Linha: array [1..20] of LinhasDeTeste;
```

Portanto podem ser criadas vinte situações diferentes para simular o sistema.

Em uma simulação são inseridos todos os dados das entradas “I” e são deixadas em branco (todas a variáveis em **falso** ou **0**) os dados de M, T e Q. Ao final da simulação estas últimas três matrizes estão preenchidas com os resultados.

Para calcular o fitness do programa de automação esses resultados precisam ser comparados com um conjunto de linhas do tempo que representam o comportamento desejado, ou seja, a função objetivo. Para que isso seja feito então também é criada a seguinte estrutura de dados:

```
var
CenariosCompara: array [1..20] of LinhasDeTeste;
```

A estrutura de dados “CenariosCompara” armazena então a função objetivo. Ao final da simulação de cada programa gerado pela programação genética são comparadas as matrizes “Q” e para cada igualdade é somado “um” na variável fitness deste programa.

5.2 Algoritmos da Simulação

5.2.1 Leitura e Escrita das Matrizes das Linhas do Tempo

Na Figura 38 pode ser visto o fluxograma da simulação de um programa de automação da forma como é feito pelo programa desenvolvido. O primeiro passo da simulação é a leitura dos estados iniciais de todos os contatos do programa.

A seguir, o programa é executado. Nessa execução, devido ao estado inicial das entradas pode haver a modificação ou não de alguma saída. Essa leitura do programa é chamada de SCAN. Verifica-se então se houve alteração de algum contato no programa. Se houve então é preciso repetir o SCAN, pois, a mudança de algum contato no final do programa pode exercer alguma ação em um contato no início do programa. O SCAN é realizado até que não haja nenhuma mudança de contato, ou seja, o programa está em equilíbrio. Dependendo de como o programa foi feito podem existir loops infinitos, portanto, para impedir que a presença de um loop desses trave o programa foi definido o limite de 10 SCANs.

Durante a simulação do programa de um CLP existem duas camadas de tempo a serem observadas. A primeira é o tempo gasto pelo SCAN. Em um CLP real, a execução desse SCAN possui tempo variável de acordo com o tamanho do programa salvo no CLP. Esse tempo gasto é da ordem de milissegundos, portanto, em um CLP real centenas de SCANs são realizados em um segundo.

Sabe-se então que a maior parte desses SCANs não representam qualquer mudança nas saídas do CLP. Para agilizar a simulação, executa-se o SCAN até que esse equilíbrio seja atingido ou até que um número de SCANs máximo seja feito.

Após o final do ciclo de SCANs, os contatos das saídas são atualizados e o relógio contador de tempo real é incrementado. Portanto, na simulação o tempo não passa enquanto os SCANs são executados. No momento em que esse relógio de tempo é incrementado, os tempos dos temporizadores são atualizados.

Esse processo de leitura da linha do tempo, execução dos SCANs e atualização das linhas do tempo é realizado até o final da linha do tempo. Todo esse processo representa a simulação de um cenário de testes. Cada estudo de caso possui um número pré-definido de cenários de teste. Quando existe mais de um cenário então, as variáveis são reinicializadas, e parte-se para a repetição do processo utilizando as linhas do tempo do próximo cenário.

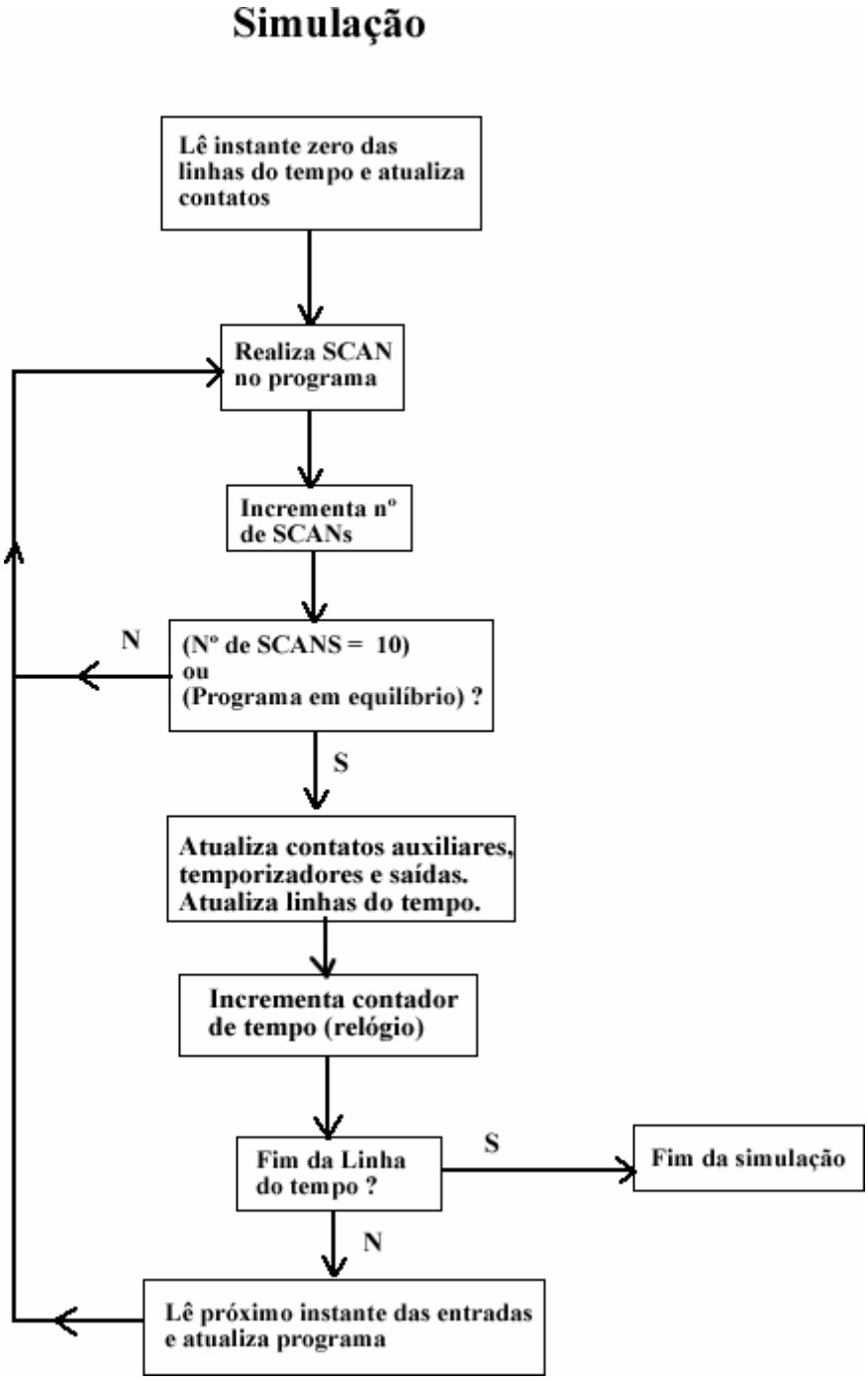


Figura 38 – Fluxograma da simulação de um programa de automação

5.2.2. Leitura do Programa (SCAN)

O fluxograma de um SCAN é apresentado na Figura 39. Esse é o principal processo executado na simulação. O processo se inicia com a leitura da primeira linha de código do

programa. Para a representação de uma Instruction List foi criada a seguinte estrutura de dados:

```
type
  instr = record
   Comand: string;
   tipo: string;
   num: integer;
   mem: string;
  end;

var
    IL: array[1..1000] of instr;
```

A estrutura de dados “instr” armazena uma linha de código Instruction List. A variável “Comand” do tipo string armazena o nome do comando (opcode), a variável “tipo” armazena o tipo do contato (I, M, Q, T), a variável inteira “num” armazena a numeração do contato (de 1 a 10), e a variável “mem” é utilizada para operações internas para comandos que possuem retenção de memória como por exemplo as saídas com retenção (OUTR), os temporizadores e os contatos de pulso.

A estrutura de dados “IL” armazena o programa Instruction List inteiro, ou seja, um conjunto de linhas. No momento em que um indivíduo da população de programas deve passar pela simulação, seu código é então copiado para “IL” e a função de simulação é executada. No programa foi definido o limite máximo de 1000 linhas para uma Instruction List. O SCAN lê cada uma das linhas do programa e executa o comando até chegar à última linha.

O seguinte bloco de código define o nome dos procedimentos chamados quando cada comando é encontrado:

```
  procedure LD(tipo: string;num: integer);
  procedure LDI(tipo: string;num: integer);
  procedure ANDP(mem_pos: string);
  procedure ANDF(num: integer; mem_pos: string);
  procedure ORB();
  procedure ANB();
```

procedure OUTR(tipo: string;num: integer;mem_pos: string);

procedure AND_(tipo: string;num: integer);

procedure ANI(tipo: string;num: integer);

procedure OR_(tipo: string;num: integer);

procedure ORI(tipo: string;num: integer);

procedure OUT_(tipo: string;num: integer;mem_pos: string);

Figura 39 – Fluxograma de um SCAN

5.3 Algoritmo da Programação Genética

O algoritmo geral da evolução dos programas é mesmo representado na Figura 33. A seguir é detalhado como cada etapa desse fluxograma foi implementada.

5.3.1 Geração da População Inicial

O armazenamento da população de programa foi feita através da seguinte estrutura de dados:

```
var
   pop1: array[1..500,1..1000] of instr;      // pais
   pop2: array[1..500,1..1000] of instr;      // filhos
```

A estrutura de dados “pop1” armazena a população inicial e a população de programas que serão simulados, e a estrutura “pop2” é utilizada apenas no momento da reprodução para armazenamento dos indivíduos filhos momentos antes dela ser passada para “pop1” para substituir a população de pais.

No início da geração aleatória de um indivíduo verifica-se primeiro a quantidade de saídas do sistema de automação. É gerado então um número de linhas aleatórias definidas pelo usuário, como pode ser visto na fig. 40 no campo “Tam. Meio do prog.”. No momento em que um conjunto de linhas do tempo é carregado (campo “Linhas de teste de comparação”, Figura 40), os números de entradas, auxiliares, saídas e temporizadores são atualizados.

Para cada “Saída – Q” é gerado um número de linhas correspondentes ao campo “Tam. Meio do prog.”, com comandos aleatórios. Esses comandos são escolhidos utilizando-se o seguinte conjunto: {LD, LDI, LDP, LDF, ANDP, ANDF, ORB, AND, ANI, OR, ORI, ANB}, ou seja, os comandos de saída não são utilizados. Após a criação deste número de linhas, é criada uma linha de saída utilizando os comandos {OUTR, OUT}. Depois que um comando é escolhido aleatoriamente é escolhido uma referência aleatória, dentre as existentes no programa. Dessa forma é criado o programa Instruction List.

Esse procedimento limita o número inicial de linhas do programa e torna esse tamanho dependente do número de saídas utilizadas.

Observa-se também que pelas estruturas “pop1” e “pop2” podem ser criados até 500 indivíduos. A coleta de dados definida pela pesquisa fixou este número em 100 indivíduos, porém este mesmo programa pode ser utilizado para pesquisas que buscam definir a ação da quantidade de indivíduos em uma população.

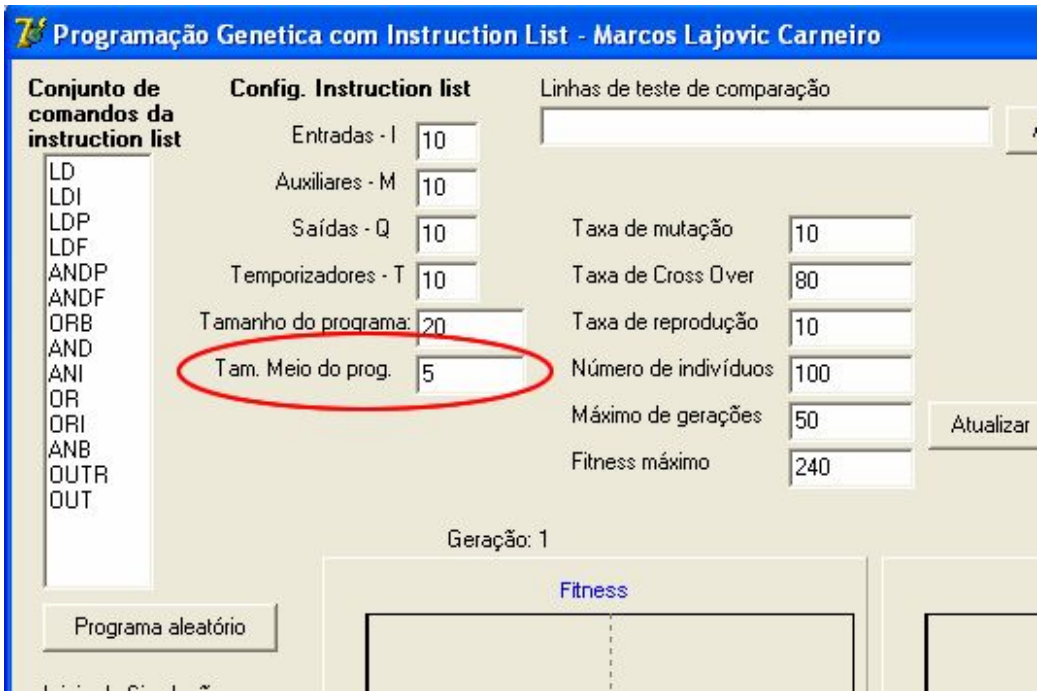


Figura 40 – Campo que define o tamanho do meio do programa

5.3.2. Carregamento dos Indivíduos no Simulador

Na Figura 33, na caixa “Avalia Fitness de cada indivíduo da População”, é definido o momento em que cada indivíduo da população, registrado em “pop1” é copiado para a matriz de simulação de Instruction List “IL”.

Nesse instante cada linha de um indivíduo definido em “pop1” é copiado para sua posição correspondente em “IL” antes da simulação.

Os indivíduos não são simulados diretamente em “pop1”, pois no momento da simulação é utilizado com campo “mem” da estrutura de dados “instr” durante o SCAN. Preferiu-se manter toda a simulação uma única posição de memória “IL” para facilitar a reinicialização de variáveis e para separar o bloco de código de simulação do bloco de código de programação genética.

5.3.3. Cálculo do Fitness (Linhas de referência)

O cálculo do fitness é feito através da comparação das linhas de tempo das saídas (Q) geradas pela simulação com linhas de tempo de referência, previamente definidas antes da busca. Essas linhas de referência são criadas pelo programador a partir da prévia noção de como o sistema automático deve se comportar.

A Figura 41 exemplifica o processo de cálculo de fitness; Antes da simulação existem as informações das entradas I1 e I2 mas não existem dados para Q1 e T1 (Figura 41a). Deseja-se adquirir um comportamento pré-definido (Figura 41b), porém a criação aleatória de indivíduos e a ação de operadores genéticos podem produzir formatos inesperados de saídas, como por exemplo, o comportamento imperfeito definido na Figura 41c.

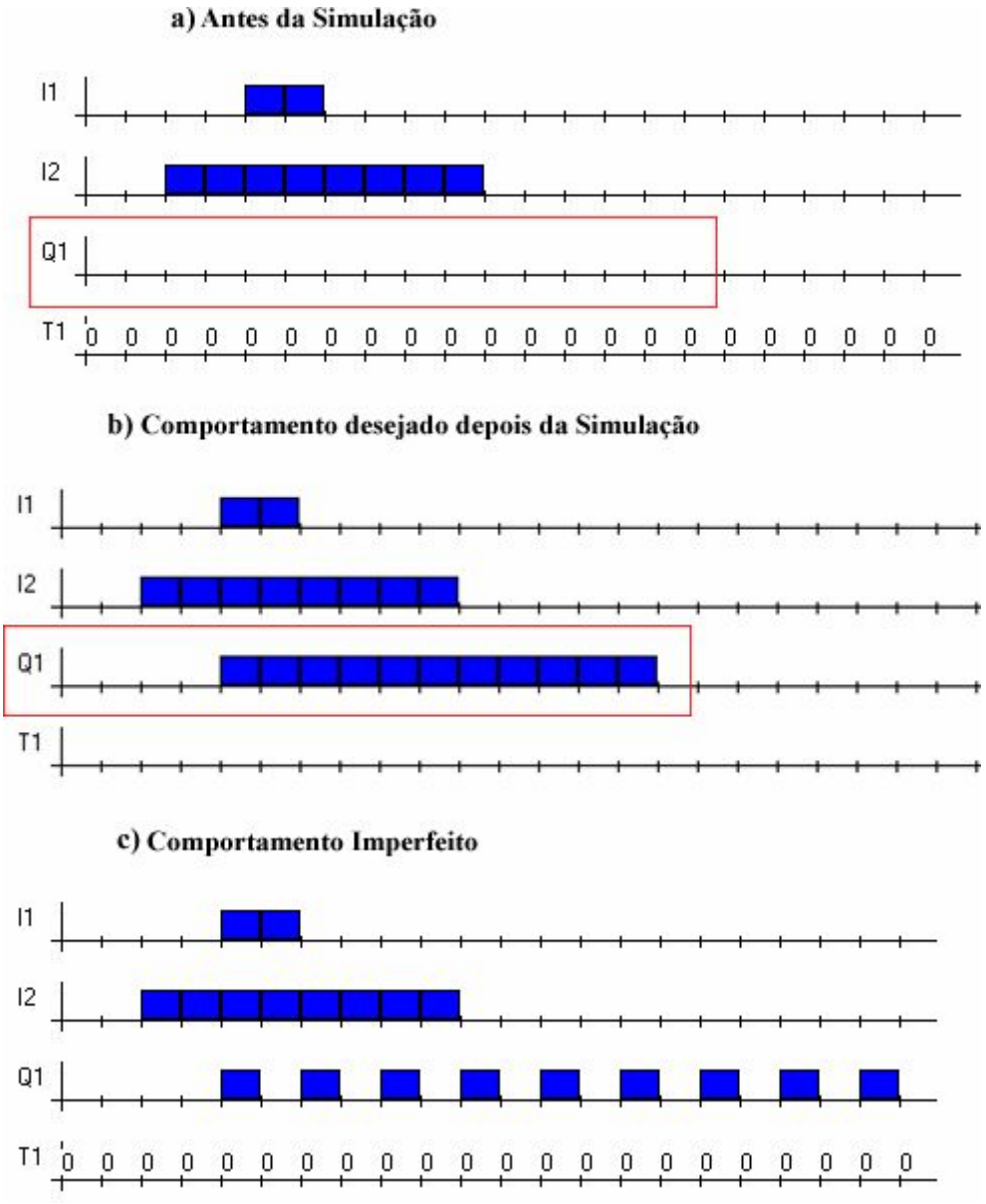


Figura 41 – Exemplificação da utilização de linhas do tempo para o cálculo do fitness

Foi criada a seguinte função para o cálculo do fitness:

```
Function TForm1.Fitness2: integer;
var
 i1,i2,i3: integer;
 k1: integer;
 Fitness: integer;
 FitMin: integer;
begin

  Fitness:= 0;
  // Compara e soma fitness
  for k1:= 1 to nCenarios do
  begin
    for i1:= 1 to nSaidas do
    begin
      for i2:= 1 to LimiteTempo do
      begin

        if Linha[k1].Q[i1,i2] = CenariosCompara[k1].Q[i1,i2] then
        Fitness := fitness+1;

      end;
    end;
  end;
  Result:= Fitness;
end;
```

A variável “Fitness” armazena o número de igualdades encontradas entre a linha Q produzida pela simulação e a linha Q previamente definida. Ao final do código o valor de fitness é repassado para “Result”. O valor de “Result” é obtido quando a função “Fitness2” é chamada e então pode ser associado a cada indivíduo.

O objetivo da busca é a de encontrar um programa que consiga realizar todos os comportamentos previamente definidos com 100% de fidelidade. No momento em que um programa apto é encontrado ele é salvo em um arquivo de texto e a busca é finalizada, como é demonstrado no critério de parada da Figura 33.

5.3.4. Seleção Torneio

A seleção torneio utilizada no programa escolhe aleatoriamente na população **quatro** programas e, em seguida, seleciona o programa com maior fitness dentre os quatro selecionados.

O código abaixo exemplifica como essa ação foi realizada em pascal:

```
Function Seleciona_Torneio(Npop:integer): integer;

Function TFormPG1.Seleciona_Torneio(Npop:integer): integer;
  var
  a1,a2,a3,a4: integer;
  b1: integer;
begin

  // ------------ Escolhe primeiro indivíduo
  // ------------ 4 individuos aleatorios
  a1:= random(Npop)+1;
  a2:= random(Npop)+1;
  a3:= random(Npop)+1;
  a4:= random(Npop)+1;

  // ------------ escolhe o maior deles

  b1:= a1;
```

```
  if Form1.IndFit[b1,2] < Form1.IndFit[a2,2] then b1:= a2;
  if Form1.IndFit[b1,2] < Form1.IndFit[a3,2] then b1:= a3;
  if Form1.IndFit[b1,2] < Form1.IndFit[a4,2] then b1:= a4;

  Result := b1;

end;
```

5.3.6. Cross-Over de 2 pontos

O operador genético cross-over utilizado no projeto retira um segmento de código do programa selecionado a partir de dois pontos (ver Figura 35). Esse segmento, por sua vez, pode estar em qualquer posição do programa.

No momento em que a população inicial é gerada o programa possui certa estrutura e organização, pois a frente de cada bloco de código com comandos de entrada e de manipulação de lógica existe um comando de saída. Com as operações do cross-over o programa perde esta estrutura e inicia-se o processo de formação de introns, de códigos não conclusivos e o programa passa a ter limite variável.

Apesar de esse operador desorganizar o código, o cross-over é responsável por buscas extensivas no espaço de soluções e ele é um operador característico dos algoritmos genéticos.

5.3.7. Mutação em 1 elemento

A mutação no código Instruction List é feito através da escolha aleatória da linha de comando. Depois de escolhida, um comando aleatório é colocado no local e uma nova referência também é escolhida (ver exemplo de mutação na **seção 4.3.5 b) Tipos de reprodução**).

6. Estudos de Caso

Neste capítulo, são apresentados os estudos de caso em que a programação genética foi utilizada para geração de programas de automação para CLP. Para cada caso é apresentado:

a) uma descrição física do sistema;

b) funcionamento do sistema

c) os cenários para simulação e cálculo de fitness compostos por linhas do tempo.

d) discussões a cerca da busca do programa e alguns programas-soluções em linguagem Ladder e instruction list obtidos pela técnica.

e) os quadros de coletas de dados das simulações (análise estatística) e os gráficos de desempenho do programa

f) as discussões acerca dos resultados obtidos e do comportamento do programa.

Foram escolhidos um caso residencial e três casos industriais que apresentassem diferenças em suas linhas do tempo. A aplicação específica de cada um em si não são o foco da pesquisa. O programa desenvolvido tem como objetivo a solução de problemas genéricos, e estes se diferenciam através de seus cenários de avaliação e não através de suas aplicações.

6.1 Estudo de Caso I – Sistema de Iluminação de Escadas

O primeiro estudo de caso é referente a um sistema de automação residencial para o controle de iluminação em escadas retirado do manual do CLP CLIC-02 da WEG, (WEG, 2008).

a) Descrição física do sistema:

As instalações físicas do sistema possuem um interruptor do tipo campainha (interruptor de pulso) no início e outro no final da escada, ambos representados por I1; lâmpadas de teto ou arandelas ao longo da escada, representadas por Q1; um ou mais sensores de presença representados por I2.

A referência T1 é utilizada para representar um temporizador internamente no CLP.

b) Funcionamento do sistema:

- Quando alguém sobe ou desce a escada, as lâmpadas precisam ser energizadas para fornecer iluminação.

- Após a saída da pessoa, o sistema de iluminação precisa ser desligado manualmente ou em cinco minutos automaticamente.

c) Cenários de teste

O primeiro cenário de teste (Figura 42) representa a situação em que as lâmpadas estão apagadas (Q1 = 0), não há ninguém na escada (I2 = 0), e nenhum interruptor foi acionado (I1 = 0). Após alguns instantes uma pessoa se aproxima da escada e é detectada pelo sensor de presença (I2 = 1). A pessoa permanece nas escadas alguns instantes e depois se retira (I2 = 0).

Essa situação representa o caso em que o local está de dia, com iluminação natural e por isso não necessita que o sistema de iluminação seja acionado.

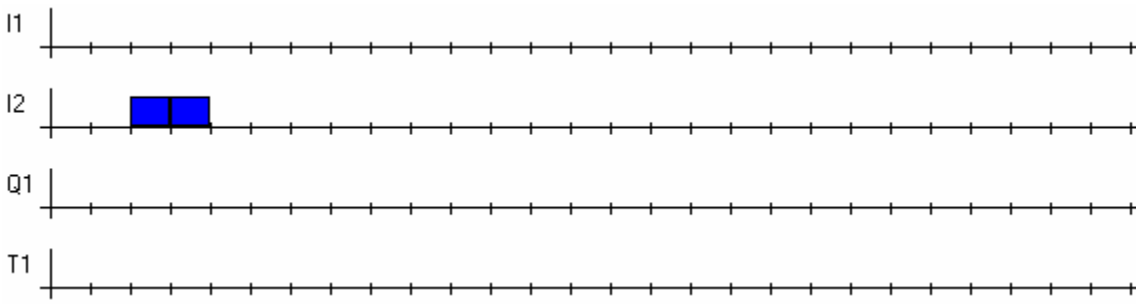


Figura 42 - Cenário 1 – Iluminação de escadas

O segundo cenário de teste (Figura 43) representa a situação em que inicialmente não há ninguém nas escadas e tudo está desligado. Após alguns instantes uma pessoa se aproxima (I2 = 1), pressiona o interruptor das lâmpadas (I1 = 1), no mesmo instante as lâmpadas se acendem (Q1 = 1) e a pessoa solta o interruptor (I1 = 0). O indivíduo permanece no local por algum tempo e depois vai embora (I2 = 0) sem apagar as lâmpadas (Q1 = 1). As lâmpadas permanecem acesas por 5 minutos e são apagadas automaticamente pelo CLP.

Essa situação representa o caso em que uma pessoa ao chegar às escadas, acende as lâmpadas, faz o percurso e depois sai do local sem apagar a iluminação. O sistema então após algum tempo desliga as lâmpadas para economia de energia.

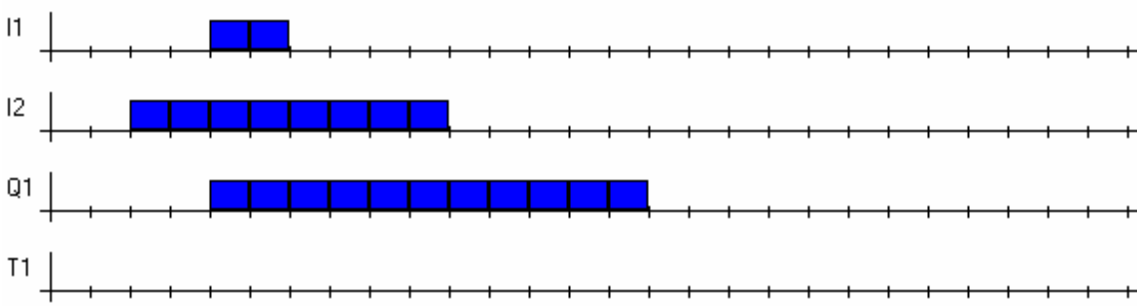


Figura 43 - Cenário 2 – Iluminação de escadas

O terceiro cenário (Figura 44) representa a situação em que uma pessoa chega até as escadas (I2 = 1), pressiona e solta o interruptor (I1 = 1, em seguida I1 = 0) e as lâmpadas se acendem (Q1 = 1). Após alguns instantes o interruptor é acionando novamente e então as lâmpadas se apagam (Q1 = 0). A pessoa ainda está presente no local, e depois de alguns instantes se retira.

Essa situação representa o caso em que uma pessoa chega até as escadas, acende a lâmpada, faz o percurso e ao final dele apaga as lâmpadas e vai embora.

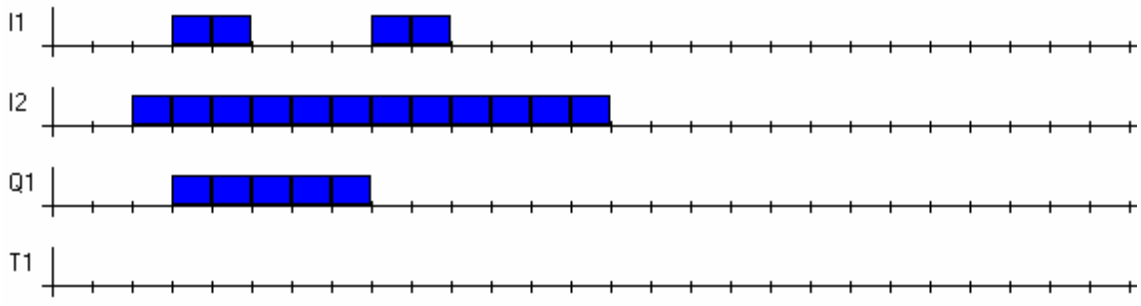


Figura 44 - Cenário 3 – Iluminação de escadas

O quarto cenário (Figura 45) representa o caso em que uma pessoa chega ao local (I2 = 1), acende as lâmpadas (Q1 = 1), permanece um tempo no local e depois se retira (I2 = 0). O indivíduo então retorna às escadas antes que os cinco minutos acabem (I2 = 1). Dessa forma as lâmpadas continuam acesas. Após algum tempo a pessoa se retira definitivamente sem apagar as luzes e após 5 minutos a iluminação é apagada.

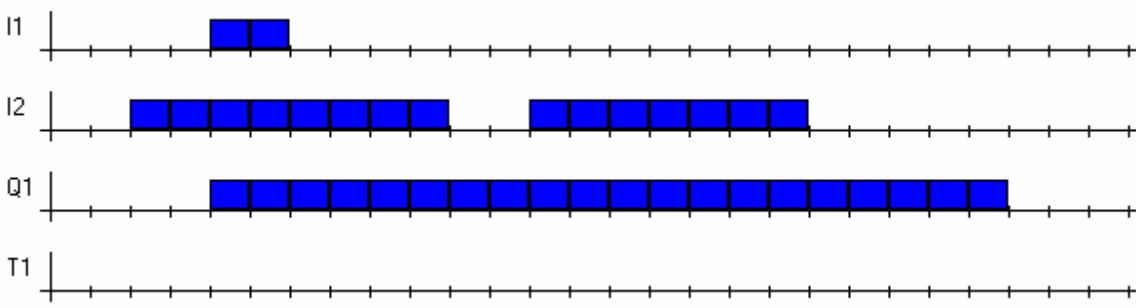


Figura 45 - Cenário 4 – Iluminação de escadas

d) Discussões iniciais e soluções encontradas

O manual do usuário do CLP CLIC 02 apresenta a seguinte solução para este problema (Figura 46):

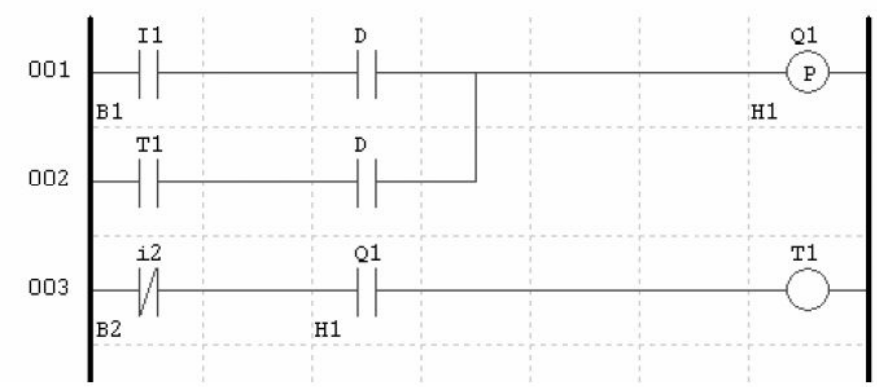


Figura 46 – Solução do manual do CLIC 02 para o sistema de iluminação

Nessa Ladder são apresentados os elementos:

I1 – Interruptor (tipo campainha).

I2 – Sensor de presença.

Q1 – Lâmpada.

T1 – Timer.

D – Elemento diferencial (envia pulso positivo de duração de um scan quando recebe uma transição positiva. Corresponde ao comando ANDP da instruction list do CLP TPW-03).

A interpretação do programa da Figura 46 é feita da seguinte forma:

Inicialmente a lâmpada está desligada. Quando o interruptor é pressionado o contato I1 fecha e chega energia até o elemento D (elemento diferencial) (linha 001). Recebendo essa transição positiva ele envia um pulso positivo durante um scan. Esse pulso chega até a lâmpada Q1 que está representada por uma bobina de pulso. Essa bobina comuta de estado (ligada para desligada, ou vice-versa) toda vez que recebe uma transição positiva.

O elemento i2 representa o sensor de presença. Percebe-se que para que o T1 (temporizador) ligue é preciso que os estados da lâmpada e do sensor ajam em conjunto (função lógica AND). Percebe-se na linha 002 que sempre que o temporizador T1 atinge um tempo pré-definido ele emite um pulso para a lâmpada Q1, comutando seu estado. Como o temporizador só pode ser ativado quando a lâmpada está ligada (contato NA, linha 003), pode-se ver que ele sempre atua no sentido de desligar a lâmpada.

Assim, estando a lâmpada ligada (contato Q1, do tipo NA, fechado), e ninguém no cômodo (contato i2, do tipo NF, fechado), o temporizador é ativado e inicia sua contagem provocando o desligamento da lâmpada quando a contagem chega ao final, o que consequentemente também inativa o temporizador.

Para demonstrar que um mesmo programa Ladder pode ser escrito de diferentes formas com a obtenção dos mesmos resultados, o programa da Figura 46 foi reescrito para o modelo de Ladder do CLP TP-02 da WEG (Figura 47).

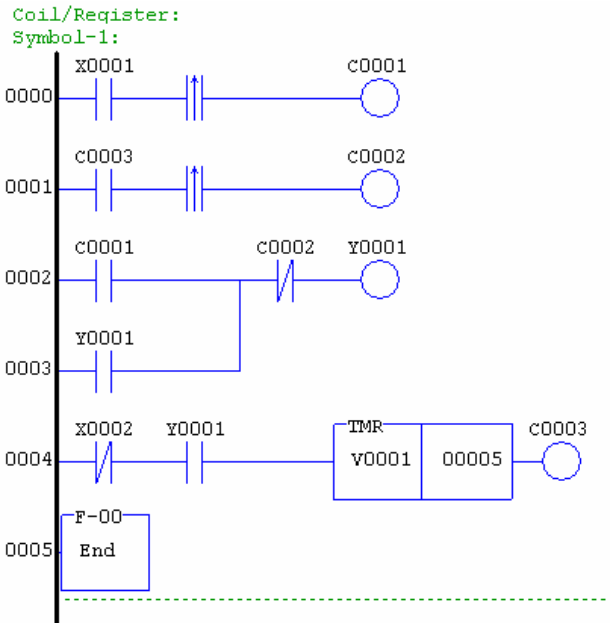


Figura 47 – Programa Ladder para o sistema de iluminação no TP-02

Percebe-se que a apesar do CLP TP-02 da WEG possuir aproximadamente 101 instruções, ao contrário das 35 instruções do CLIC-02, ele não possui a bobina de pulso, sendo

que o mesmo programa precisa ser escrito de outra forma. Essa diferença é representada pelo inter-travamento utilizado na linha 0003 da Figura 47.

A seguir o mesmo programa foi escrito em linguagem Instruction List para o CLP TP-02 (Figura 48):

```
ADDRESS     INSTRUCTION
P>0000      STR       X0001
  0001      F-05      -|^|-
  0002      OUT       C0001
  0003      STR       C0003
  0004      F-05      -|^|-
  0005      OUT       C0002
  0006      STR       C0001
  0007      OR        Y0001
  0008      AND NOT   C0002
  0009      OUT       Y0001
  0010      STR NOT   X0002
  0011      AND       Y0001
  0012      TMR       V0001
                      00005
  0014      OUT       C0003
  0015      F-00      End
  0016
```

Figura 48 – Programa de controle de iluminação em Instruction List

A conversão da Ladder da Figura 46 para a Instruction List do TPW-03 (modelo utilizado para o programa desenvolvido na pesquisa) é representado abaixo:

01:	LD	I1
02:	ANDP	
03:	LD	T1
04:	ANDP	
05:	ORB	
06:	OUTR	Q1
07:	LDI	I2
08:	AND	Q1
09:	OUT	T1

Apresenta-se a seguir sete resultados diferentes (Figura 49, Figura 50, Figura 51, Figura 52, Figura 53, Figura 54, Figura 55) obtidos através do sistema de programação genética com a utilização dos cenários apresentados.

```
1:      LD     Q1
2:      OR     I2
3:      OR     I2
4:      AND    T1
5:      OR     I1
6:      OUTR   Q1
7:      AND    I1
8:      ORB
9:      ORI    I2
10:     OUT    T1
END
```

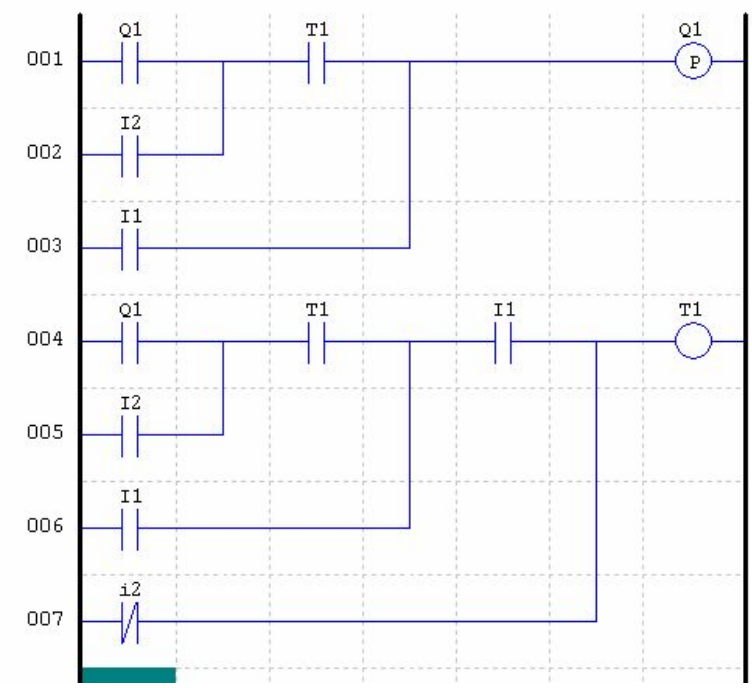


Figura 49 – Resultado 1 – Iluminação de escadas

Na Instruction List da Figura 49 percebe-se a presença de um comando AND na linha 7, logo após um comando OUTR devido as trocas feitas pelos operadores genéticos. Como o comando OUTR é um comando que finaliza uma lógica, o comando seguinte deve ser um comando de início de lógica. Esse programa quando colocado em um programador de CLP comum apresentaria erro, pois AND não é um comando de início de lógica. Nesse caso a operação AND é realizada com lógica produzida até a linha 5 da Figura 49 (linha anterior ao comando OUTR da linha 6).

O programa criado na pesquisa consegue interpretar a lógica dessa Instruction List apresentando o comportamento exatamente igual ao da Ladder da Figura 49. Esse programa em uma Instruction List para ser executável em um CLP comum deve ser representado da seguinte forma:

```
1:      LD     Q1
2:      OR     I2
3:      AND    T1
4:      OR     I1
5:      OUTR   Q1
6:      LD     Q1
7:      OR     I2
8:      AND    T1
9:      OR     I1
```

```
10:     AND   I1
11:     ORI   I2
12:     OUT   T1
END
```

A seguir são apresentados mais outros resultados:

```
1:      LDI   I2
2:      ANDP
3:      LD    I2
4:      NOP
5:      OR    Q1
6:      AND   T1
7:      OR    I1
8:      OUTR  Q1
9:      LD    Q1
10:     ANI   I2
11:     OR    I1
12:     OUT   T1
END
```

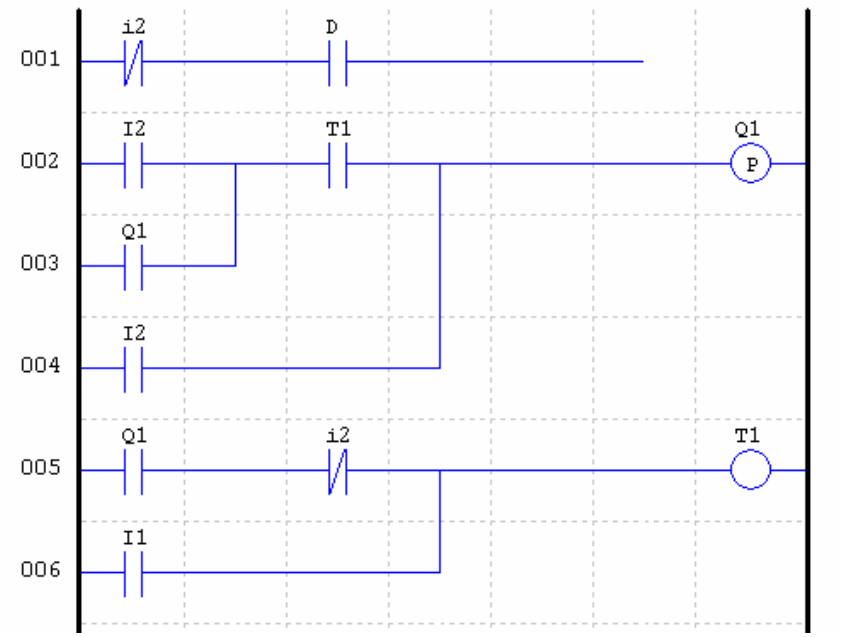


Figura 50 – Resultado 2 – Iluminação de escadas

```
1:      LD    I1
2:      NOP
3:      NOP
4:      LD    T1
5:      LD    T1
6:      ORB
7:      ORB
8:      OUTR  Q1
9:      LD    Q1
10:     ORB
11:     LD    Q1
12:     AND   Q1
13:     ANI   I2
14:     OUT   T1
END
```

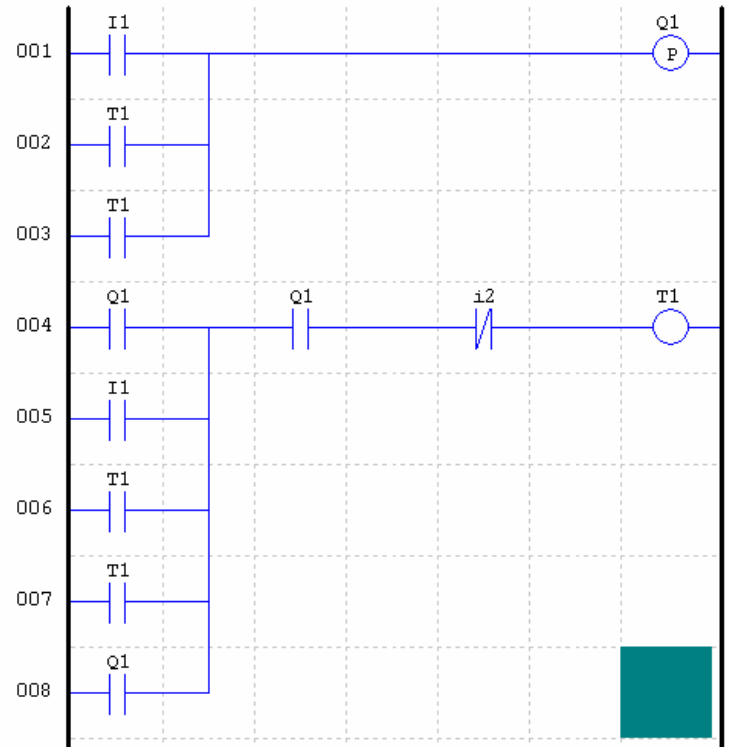


Figura 51 – Resultado 3 – Iluminação de escadas

```
1:      LD    T1
2:      ANI   I2
3:      AND   T1
4:      OR    I1
5:      NOP
6:      ORB
7:      NOP
8:      OUTR  Q1
9:      LD    I1
10:     NOP
11:     AND   Q1
12:     ORB
13:     NOP
14:     LDI   T1
15:     OUT   T1
END
```

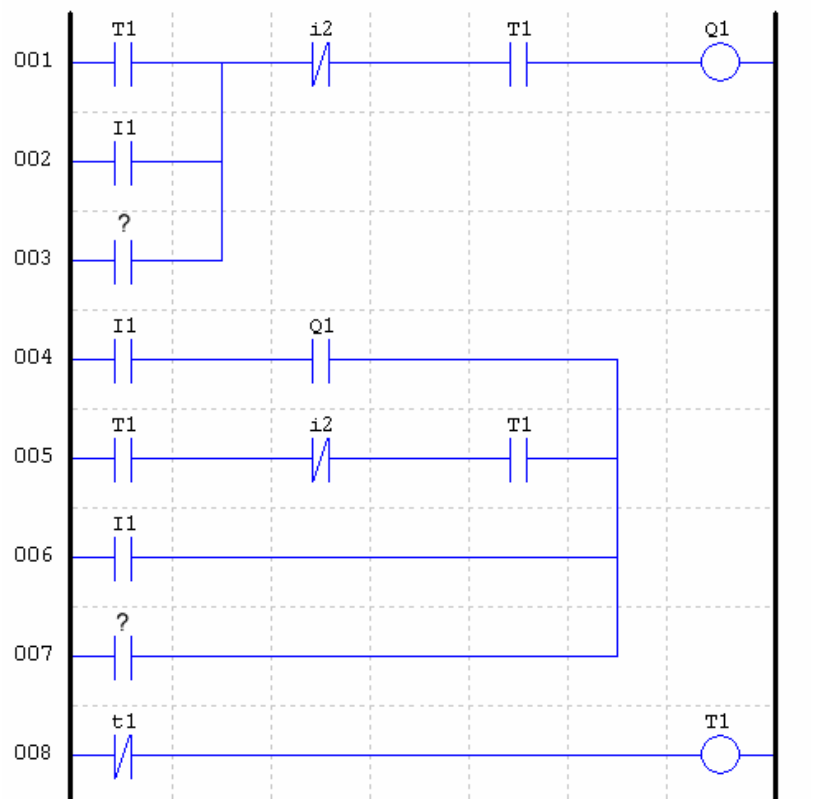


Figura 52 – Resultado 4 – Iluminação de escadas

```
1:      LD    Q1
2:      ANI   I2
3:      AND   T1
4:      OR    I1
5:      NOP
6:      ORB
7:      NOP
8:      OUTR  Q1
9:      LD    I1
10:     NOP
11:     ORI   I2
12:     ORB
13:     NOP
14:     ANI   I2
15:     ORB
16:     OUT   T1
END
```

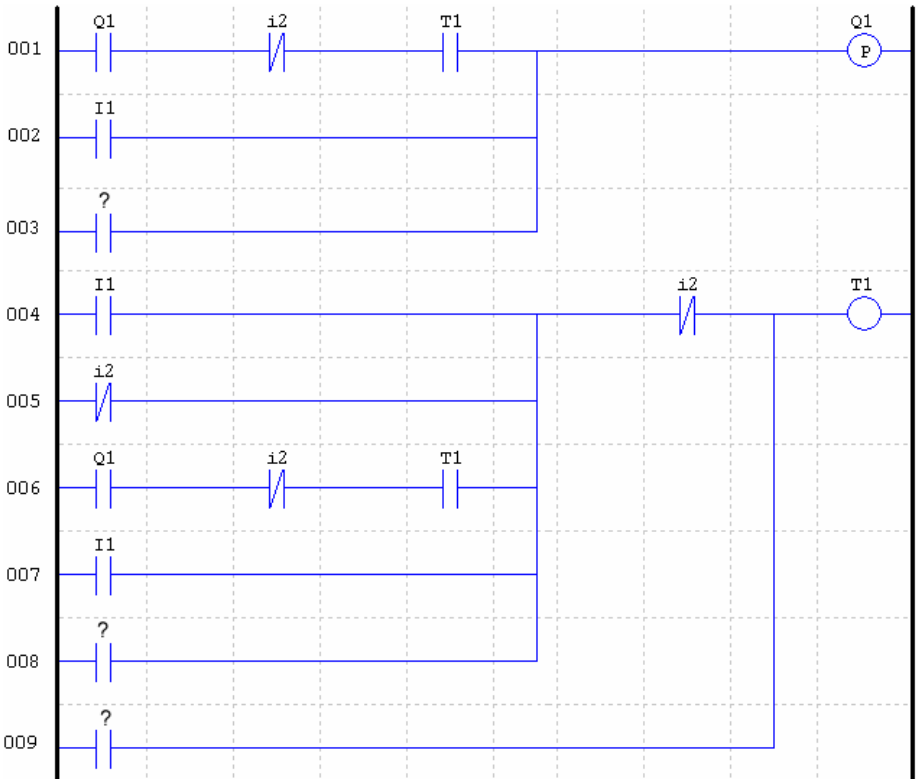


Figura 53 – Resultado 5 – Iluminação de escadas

```
1:      LD    Q1
2:      ANI   I2
3:      AND   T1
4:      OR    I1
5:      NOP
6:      ORB
7:      NOP
8:      OUTR  Q1
9:      LD    I1
10:     NOP
11:     ORI   I2
12:     ORB
13:     ANDF
14:     ORB
15:     OUT   T1
END
```

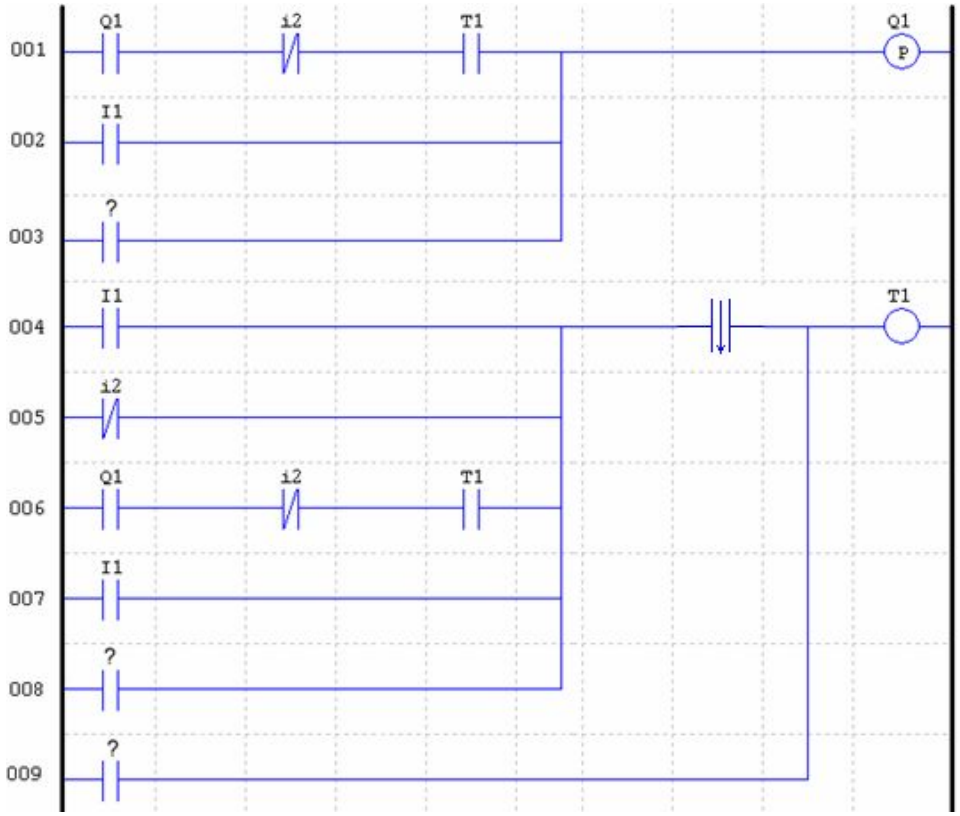


Figura 54 – Resultado 6 – Iluminação de escadas

```
1:      LD    I1
2:      OR    T1
3:      OUTR  Q1
4:      LD    Q1
5:      ANI   I2
6:      OR    T1
7:      OR    T1
8:      OR    T1
9:      OUT   T1
END
```

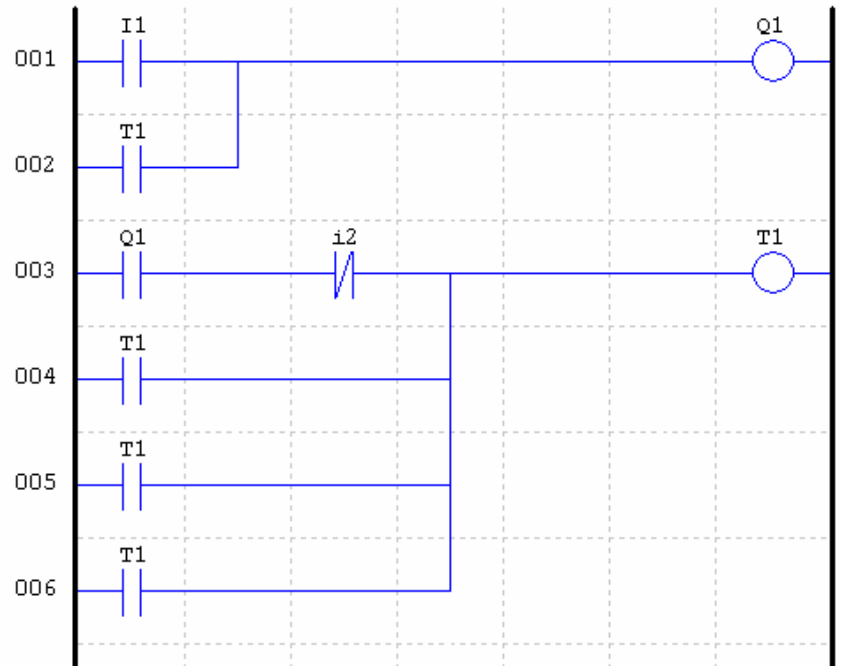


Figura 55 – Resultado 7 – Iluminação de escadas

Os sete resultados apresentados acima possuem diferenças de comprimento e forma. O programa solução da Figura 50 apresenta na linha 001 uma seção de código com lógica não conclusiva pela falta de uma bobina no final da Ladder. Esse segmento de código é um exemplo de intron. A presença desse código não-funcional não causa dano na lógica, pois ele não causa mudança na funcionalidade dos códigos seguintes. Entretanto durante a inserção desse programa em um CLP essa linha deve ser excluída.

A presença de introns demonstra o princípio da não-parcimonia na programação genética (KOZA, 2000). Depois que uma resposta com fitness máximo é encontrada, ela precisa ser analisada e seus introns excluídos.

A Figura 51 mostra códigos redundantes nas linhas 003 e 007. Na Figura 52, há um grande segmento de código não-funcional (intron) da linha 004 até a linha 007 pela falta de um comando de saída de lógica, e na linha 003 há um contato sem referência. Contatos sem referência eliminam a funcionalidade da linha de Ladder que a contém. Isso ocorre pela seqüência de código na Instruction List através do comando ORB que executa uma operação OR com a lógica do comando previamente analisado. Em alguns casos não há comando iniciando essa lógica e então o contato sem referência gerado é ignorado na simulação do programa.

Todas essas circunstâncias são estabelecidas no simulador para prevenir que o algoritmo trave. (KOZA, 2000) afirma que a estrutura do programa deve ser gerada através da programação genética automaticamente através da pressão seletiva do fitness. Com o objetivo de obter maior diversidade e número de soluções, permitiu-se que os introns fossem gerados livremente. Se por um lado eles representam código não-funcional, por outro, eles aumentam as chances de geração de novas estruturas de programas. Blocos de introns ao longo das simulações através da mudança de um único elemento podem adquirir funcionalidade e representar degraus de evolução para a população.

Portanto a presença de introns nos programas dessa pesquisa são plenamente aceitáveis. (KOZA, 2000) e (BANZHAF et al., 1998) demonstram que a presença de introns é comum em experimentos de programação genética e que durante as operações genéticas é praticamente impossível que esses códigos não surjam.

e) tabelas de coletas de dados das simulações e gráficos de desempenho.

A seguir são apresentados os resultados de 1595 simulações realizadas para o estudo de caso do sistema de iluminação de escadas. Em todas essas simulações foi utilizado elitismo, seleção por torneio, taxa de reprodução de 10% e população de 100 indivíduos.

Durante a coleta de dados foram definidas diferentes configurações de simulação para testar o desempenho do algoritmo. Definiu-se um número máximo de gerações para a simulação e um número máximo de simulações para cada situação. À medida que o número de gerações aumenta, o tempo de simulação aumenta proporcionalmente. Dessa forma definiu-se 150 simulações independentes para os casos em que o limite máximo de gerações foi fixado em 25. Esse número foi sendo reduzido gradativamente até o limite de 30 simulações independentes. Em algumas tabelas apresentam-se dados de mais de 30 simulações para simulações com número máximo de gerações de 500 por motivo de aproveitamento de simulações previamente feitas.

Para cada simulação obteve-se como resultado o sucesso (para simulações que produziram respostas 100% compatíveis com o objetivo) e o não-sucesso (para simulações que só produziram respostas abaixo de 100%). Em cada tabela foi computado o número de simulações acumuladas e o número de sucessos acumulados, referente o número de sucessos acumulados ao longo das simulações à medida que o limite de gerações foi aumentado.

Para cada linha da tabela foi calculado a probabilidade de sucesso Y(M,i) (equação 1),, a probabilidade acumulada de sucesso P(M,i) (equação 2), e o número de simulações necessárias para obtenção de um sucesso R(z) (equação 4), como já foi explicado anteriormente.

A Tabela 2 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 80%

- Mutação: 10%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 2 – Simulações com Cross-Over 80% e Mutação 10% – Iluminação de escadas

Máximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	6	150	6	4,0000%	4,0000%	113
50	100	3	250	9	3,6000%	7,6000%	58
100	80	3	330	12	3,6364%	11,2364%	39
150	50	3	380	15	3,9474%	15,1837%	28
200	30	1	410	16	3,9024%	19,0862%	22
300	30	3	440	19	4,3182%	23,4044%	17
400	30	4	470	23	4,8936%	28,2980%	14
500	30	7	500	30	6,0000%	34,2980%	11
Total	500	30					

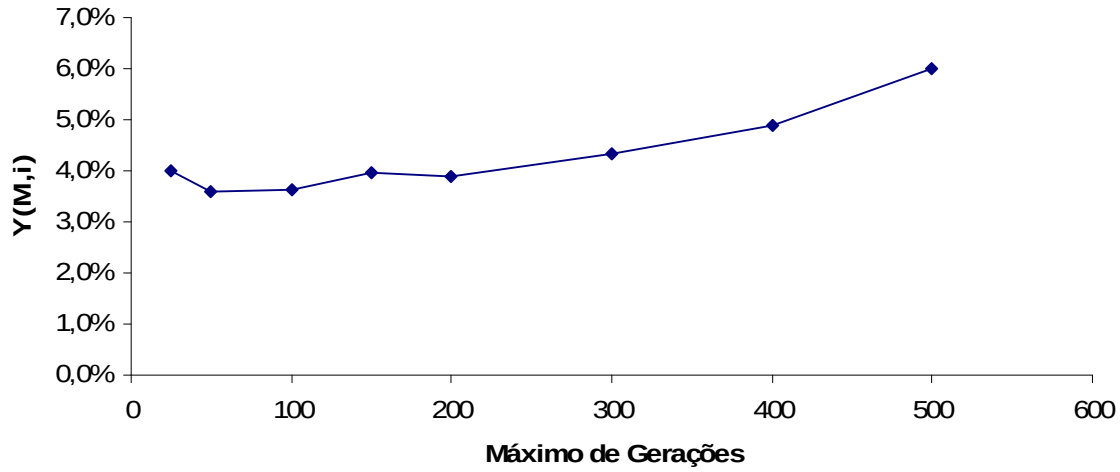


Figura 56 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 80% e mutação 10% – Iluminação de escadas

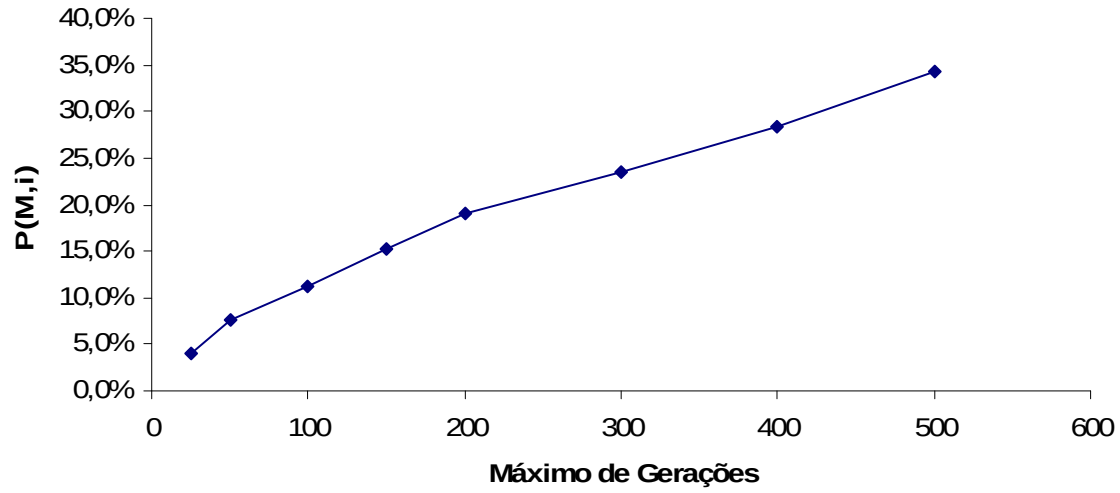


Figura 57 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 80% e mutação 10% – Iluminação de escadas

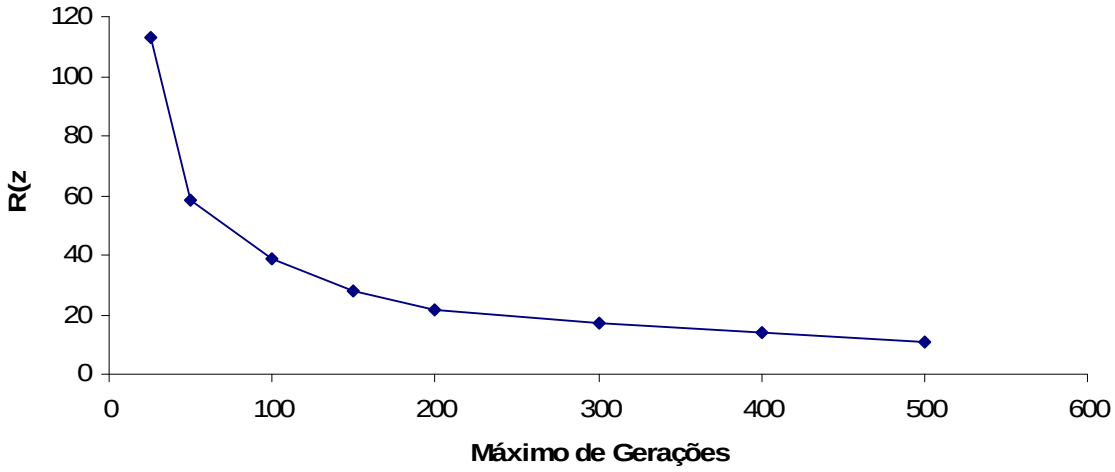


Figura 58 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 80% e mutação 10% – Iluminação de escadas

A Tabela 3 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 45%

- Mutação: 45%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 3 – Simulações com Cross-Over 45% e Mutação 45% – Iluminação de escadas

Máximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	2	150	2	1,3333%	1,3333%	343
50	100	4	250	6	2,4000%	3,7333%	121
100	80	7	330	13	3,9394%	7,6727%	58
150	50	4	380	17	4,4737%	12,1464%	36
200	30	4	410	21	5,1220%	17,2684%	24
300	30	3	440	24	5,4545%	22,7229%	18
400	30	6	470	30	6,3830%	29,1059%	13
500	30	9	500	39	7,8000%	36,9059%	10
Total	500	39					

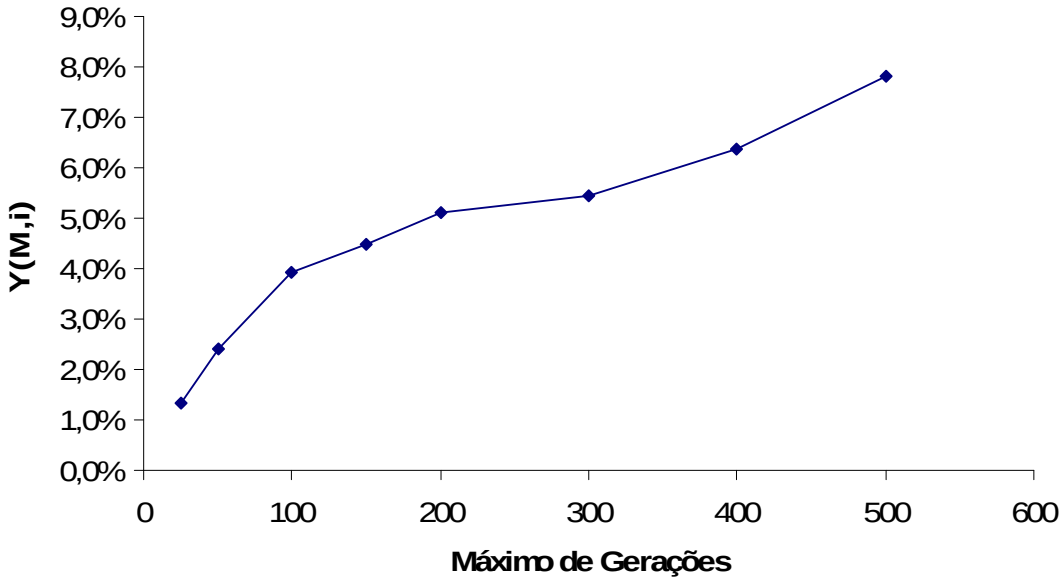


Figura 59 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Iluminação de escadas

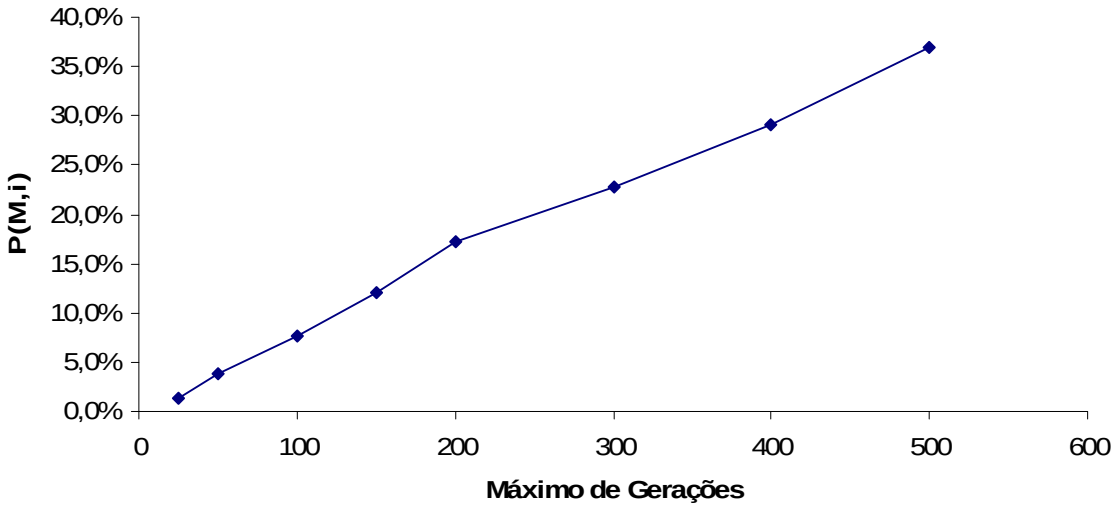


Figura 60 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Iluminação de escadas

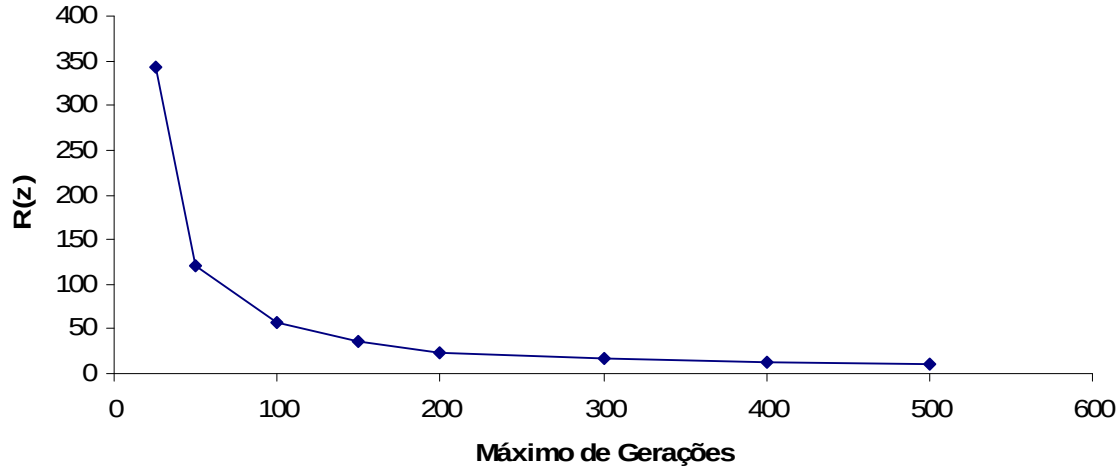


Figura 61 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 45% e mutação 45% – Iluminação de escadas

A Tabela 4 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 10%

- Mutação: 80%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 4 – Simulações com Cross-Over 10% e Mutação 80%

Máximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	1	150	1	0,6667%	0,6667%	688
50	100	2	250	3	1,2000%	1,8667%	244
100	80	4	330	7	2,1212%	3,9879%	113
150	50	7	380	14	3,6842%	7,6721%	58
200	30	8	410	22	5,3659%	13,0379%	33
300	30	8	440	30	6,8182%	19,8561%	21
400	30	8	470	38	8,0851%	27,9412%	14
500	30	11	502	50	9,9602%	37,9014%	10
Total	500	49					

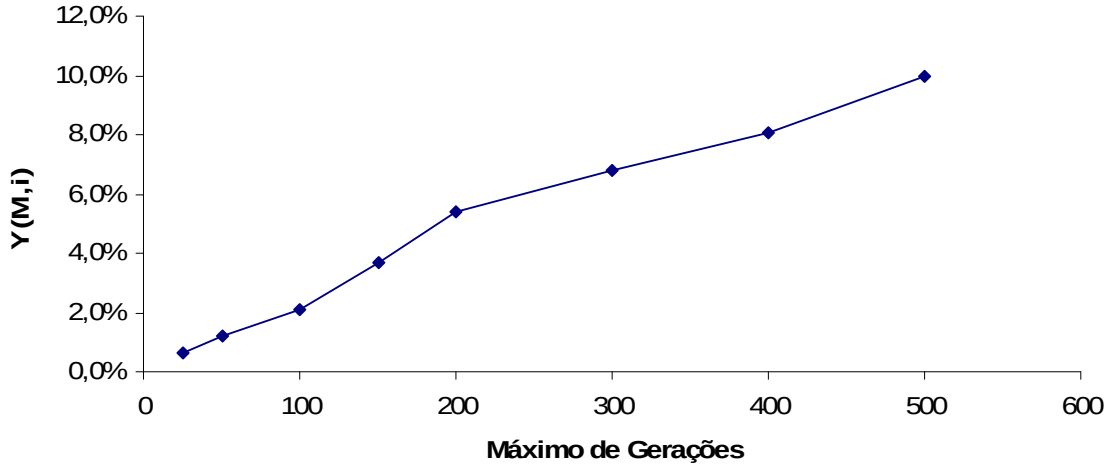


Figura 62 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Iluminação de escadas

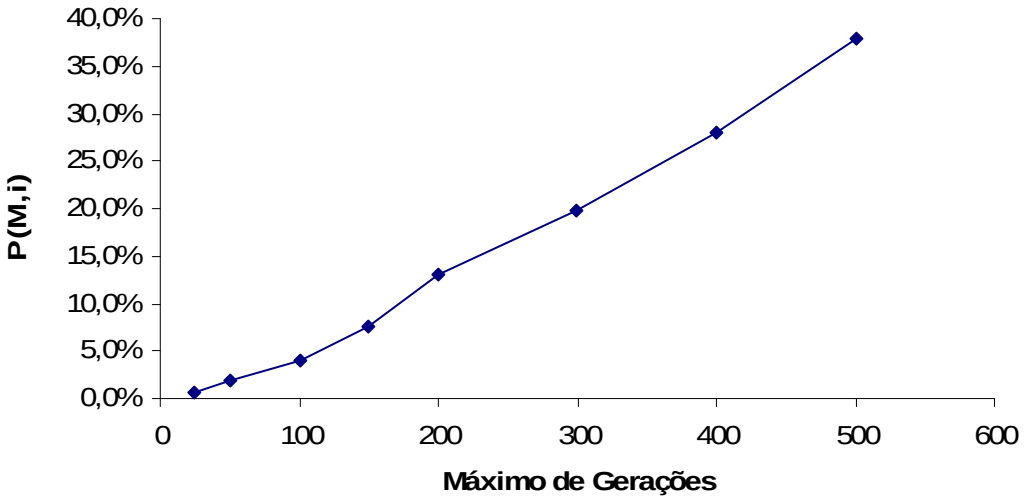


Figura 63 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Iluminação de escadas

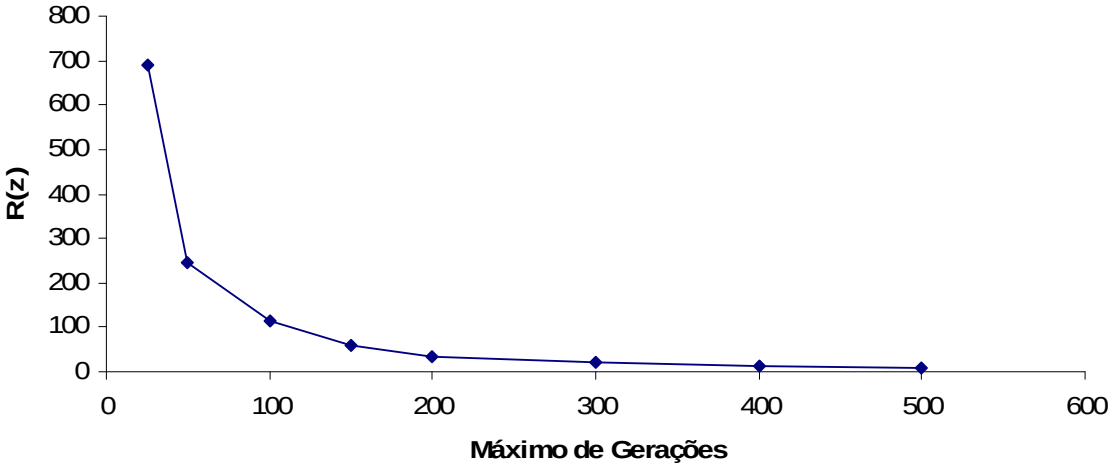


Figura 64 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 10% e mutação 80% – Iluminação de escadas

f) as discussões acerca dos resultados obtidos e do comportamento do programa.

Percebe-se na Tabela 2 que para 25 gerações, de 150 simulações apenas 6 obtiveram sucesso. Essa porcentagem corresponde a 4% das simulações. Essa porcentagem de sucesso permaneceu constante até a marca das simulações com limite máximo de 300 gerações. Foram feitas 63 coletas para simulações com máximo de 500 gerações. Nessa situação obteve-se 22,22% de sucesso nas simulações, correspondendo a 7% de sucesso acumulado em relação ao quadro geral da coleta de dados. Esse resultado reforça a idéia de que quanto mais gerações disponíveis para a evolução, maior a chance de se encontrar a solução para o problema (característico da PG).

Os gráficos da Figura 56 e da Figura 57 demonstram o crescimento da probabilidade de sucesso Y(M,i) a medida que o número de gerações aumenta. Na Figura 58, o gráfico de R(z) apresenta uma queda exponencial no número de simulações necessárias para se obter um sucesso à medida que as simulações se tornam mais longas.

Nas simulações apresentadas na Tabela 3 há um acréscimo de mutação e observou-se crescimento do número de soluções obtidas. Para simulações com 500 gerações obteve-se 30% de sucesso, e 9% de sucesso acumulado. Foram obtidas 49 soluções, o que representa um acréscimo de 27,45% de sucessos em relação aos a Tabela 2. Os gráficos da Figura 59, Figura 60 e Figura 61 permaneceram semelhantes aos gráficos do quadro anterior, indicando os resultados foram próximos, mas que também houve melhora.

Na Tabela 4 observou-se novamente uma melhora no número de soluções obtidas para simulações longas (500 gerações), subindo o nível de sucesso para 37,5%, e 38% de sucesso

acumulado. Comparando a Tabela 2, a Tabela 3 e a Tabela 4 com relação ao número de soluções obtidas com um número baixo de gerações (25) observou-se uma piora de desempenho. Para simulações com limite de 25 gerações na tabela 2 foram obtidos 6 sucessos, na tabela 3, 2 sucessos e na tabela 4, 1 sucesso. Os resultados indicam que porcentagens maiores de cross-over favorecem a obtenção de soluções com menores gerações, e porcentagens maiores de mutação, maiores sucessos com longas simulações.

O cross-over proporciona uma busca mais dispersa no espaço de soluções e a mutação uma busca nas vizinhanças dos indivíduos da população. Para esse estudo de caso observa-se que o aumento da taxa de mutação de 10% para 45% melhorou o desempenho do algoritmo, porém não foi observada mudança significativa, ou que houve uma pequena redução de desempenho, quando esse aumento passou para 80%.

Esses resultados demonstraram que a programação genética foi eficiente em encontrar programas de CLP que realizassem a tarefa de controlar o sistema de iluminação. O programa da Figura 55 foi considerado a melhor solução encontrada. Apesar de possuir algumas redundâncias, estas podem ser facilmente visualizadas e retiradas, produzindo uma solução ainda melhor. Dessa forma a técnica demonstrou ser útil para auxiliar o programador.

6.2 Estudo de Caso II – Partida Direta de Motores

O segundo estudo de caso é referente a um sistema de automação industrial simples para o acionamento direto de motores.

a) Descrição física do sistema:

A instalação física do sistema possui apenas o motor, o CLP e duas botoeiras.

b) Funcionamento do sistema:

- A botoeira I2 executa a partida direta do motor.
- A botoeira I1 retira a alimentação do motor para realizar sua parada.

c) Cenários de teste

O primeiro cenário de teste (Figura 65) representa a situação em que se nada for acionado o motor não deve entrar em movimento.

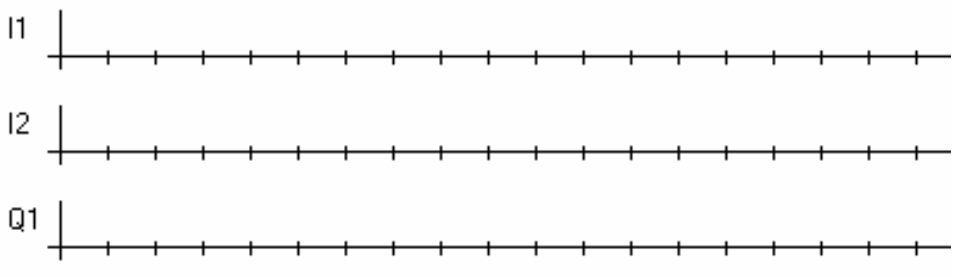


Figura 65 - Cenário 1 – Partida direta de motor

O segundo cenário de teste (Figura 66) representa a situação em que se a botoeira I2 é acionada, o motor entra em funcionamento.

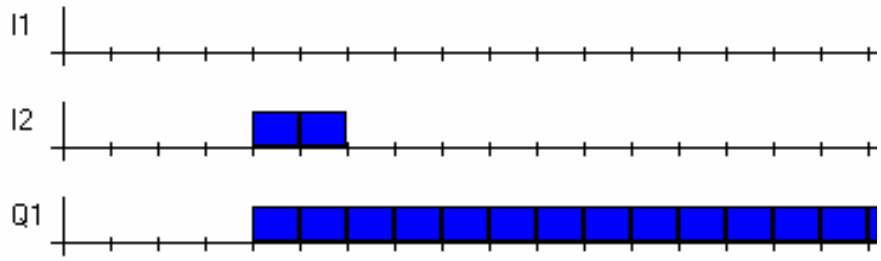


Figura 66 - Cenário 2 – Partida direta de motor

O terceiro cenário de teste (Figura 67) representa a situação em que o motor se encontra desligado, não é acionado em nenhum momento pela botoeira I2. A botoeira I1 é acionada diversas vezes, demonstrando que nessa situação ela não deve executar nenhuma função sob o motor, pois ela é a botoeira de parada.

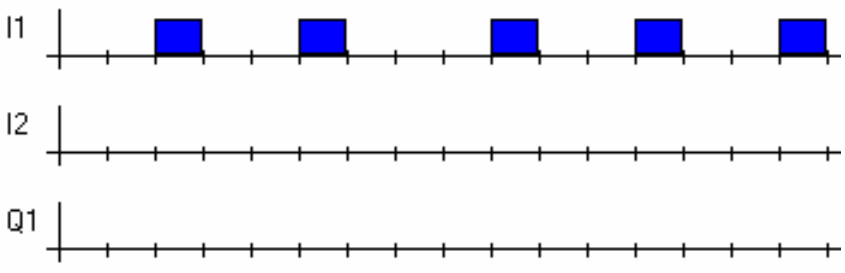


Figura 67 - Cenário 3 – Partida direta de motor

O quarto cenário de teste (Figura 68) representa a situação em que o motor é acionado (Q1 = 1) pela botoeira I2 (I2 = 1). O motor permanece rodando por algum tempo, e em seguida a botoeira I1 é ativada (I1 = 1) executando a parada do motor.

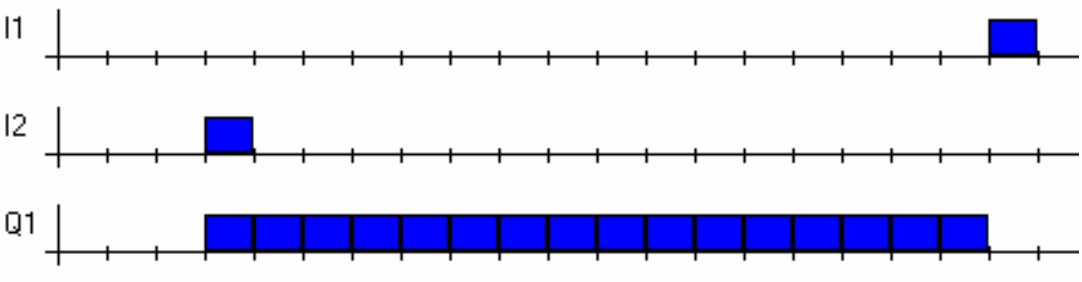


Figura 68 - Cenário 4 – Partida direta de motor

O quinto cenário de teste (Figura 69) representa a situação em que o motor é acionado (Q1 = 1) pela botoeira I2 (I2 = 1). Após o primeiro acionamento, a botoeira I2 é pressionada varias vezes de forma a demonstrar que depois que o motor está em movimento ela não exerce função alguma. Após alguns instantes o motor é parado pela botoeira I1, permanece um tempo nesse estado e novamente é acionado por I2. Após alguns instantes ele novamente é parado por I1.

O uso de situações repetidas em um mesmo cenário torna o algoritmo mais seletivo, impedindo que o programa acuse como adequada uma solução que consegue repetir o correto funcionamento do sistema apenas uma vez antes que o CLP seja resetado.

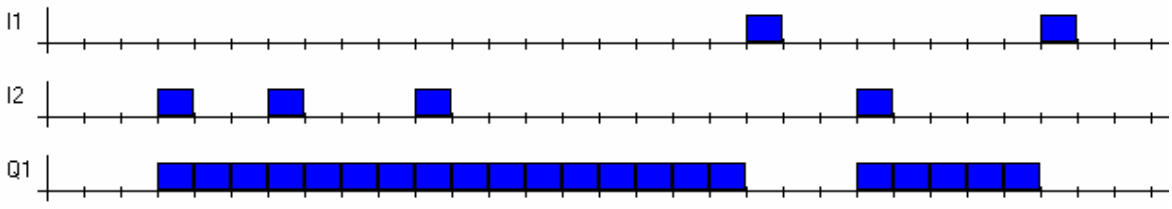


Figura 69 - Cenário 5 – Partida direta de motor

O sexto cenário de teste (Figura 70) representa a situação em que a botoeira I2 fica apertada durante muito tempo antes que seja solta novamente. Esse cenário indica que o motor parte no momento em que ela é pressionada e que o fato de mantê-la nesse estado não altera seu comportamento. Em seguida o motor é parado pela ação da botoeira I1.

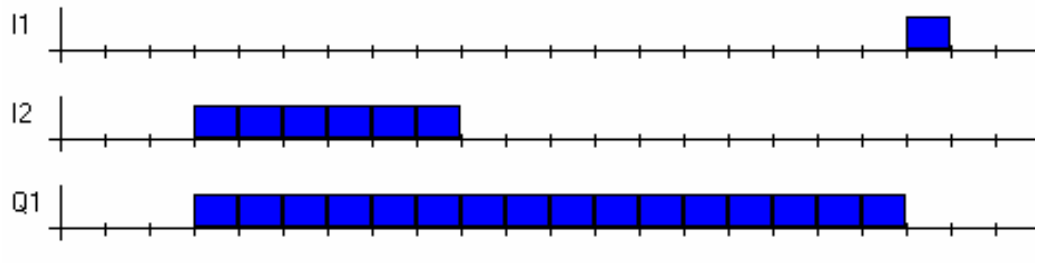


Figura 70 - Cenário 6 – Partida direta de motor

d) Discussões iniciais e soluções encontradas

Na Figura 71 é observado o esquema convencional para montagem das ligações para partida direta do motor (apenas diagrama de comando). Logo em seguida, na Figura 72, observa-se o menor programa Ladder que executa essa operação. Este foi o estudo de caso mais simples realizado com o objetivo de comparar o desempenho do algoritmo para casos simples e casos mais complexos.

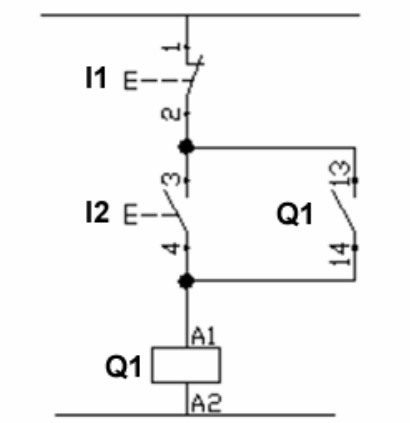


Figura 71 – Esquema convencional para partida direta de motor

Figura 72 – Programa Ladder mais simples para execução da partida direta do motor

Nessa Ladder (Figura 72) são apresentados os elementos:

I1 e I2 – Interruptores (tipo campainha ou botoeira).
Q1 – Motor.

A interpretação do programa da Figura 72 é feita da seguinte forma:

Inicialmente o motor está parado. No momento em que I2 é acionado o motor é ativa do e o contato Q1 abaixo de I2 realiza o inter-travamento do sistema, o que mantém o motor ligado depois que a botoeira I2 é solta. Quando a botoeira I1 é apertada o motor perde alimentação e o inter-travamento também é desativado.

Apresenta-se a seguir seis resultados diferentes (Figura 73, Figura 74, Figura 75, Figura 76, Figura 77 e Figura 78) obtidos através do sistema de programação genética com a utilização dos cenários apresentados na Figura 65, Figura 66, Figura 67, Figura 68, Figura 69 e Figura 70). A partir desse caso foi utilizado um software com a capacidade de desenhar diagramas Ladders maiores. Com essa mudança para a associação do diagrama Ladder com sua respectiva instructon list é preciso fazer a seguinte conversão:

- Entradas mudam de referência “I” para referência “X”.
- Saídas mudam de referência “Q” para referência “Y”.
- Temporizadores mudam de referência “T” para referência “V”.
- Contatos auxiliares mudam de referência “M” para referência “C”.

```
LD    I2
LD    I2
ANI   Q1
OR    I1
ORB   I1
ANDF
AND   Q1
OR    I2
OUTR  Q1
END
```

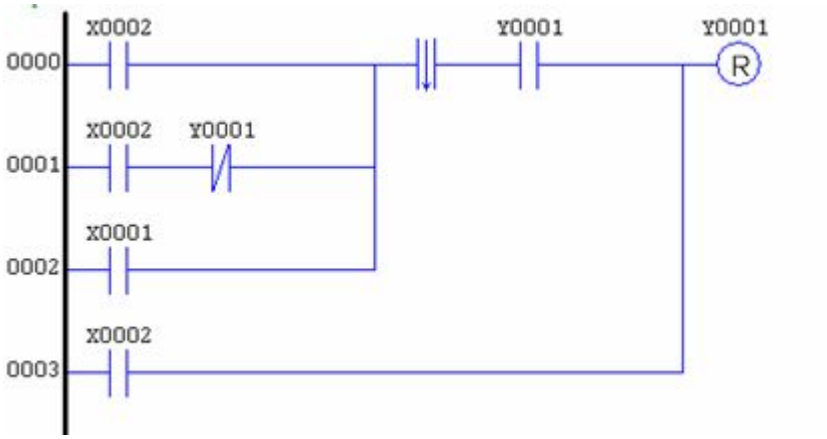


Figura 73 – Resultado 1 – Partida direta de motor

```
LDI    Q1
ANB
OR     I1
ANDF
AND    Q1
LD     I2
ORB
OUTR   Q1
END
```

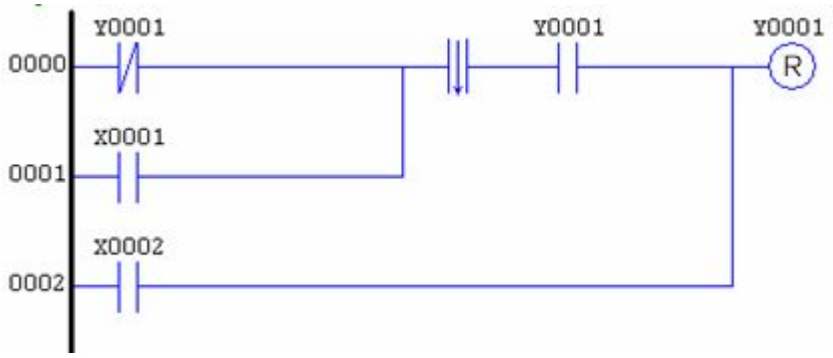


Figura 74 - Resultado 2 – Partida direta de motor

```
LDI    I2
ANB    Q1
LD     I1
LD     Q1
ANB
LDI    Q1
LD     I2
ANB    Q1
ORB    I1
OUTR   Q1
END
```

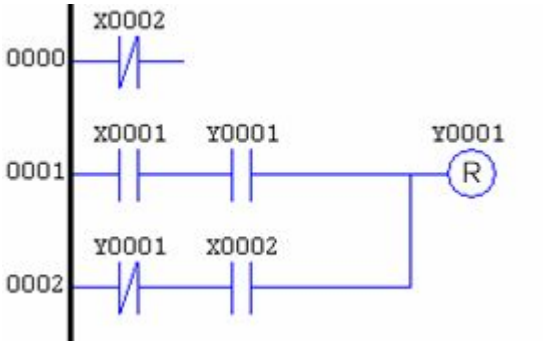


Figura 75 - Resultado 3 – Partida direta de motor

```
LDI    I2
ANDF
ANDF
ANI    I1
LD     I2
ORI    Q1
AND    I1
OR     Q1
ANI    I1
OR     I2
OUT    Q1
END
```

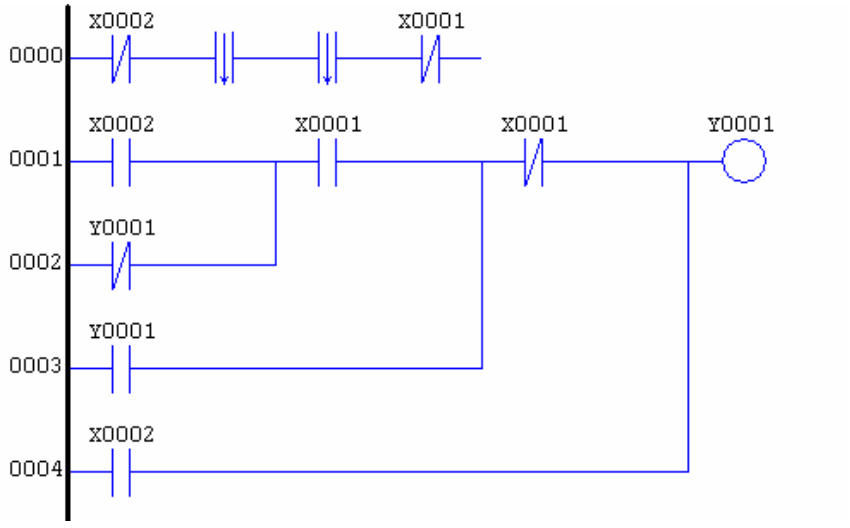


Figura 76 – Resultado 4 – Partida direta de motor

```
LD     Q1
OR     I2
ANI    I1
OUT    Q1
END  0
```

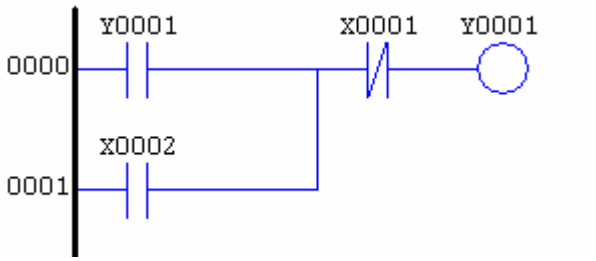


Figura 77 – Resultado 5 – Partida direta de motor

```
LDI  I1
ORI  I2
ANI  I1
NOP
AND  Q1
LD  I2
ORB
OUT  Q1
END
```

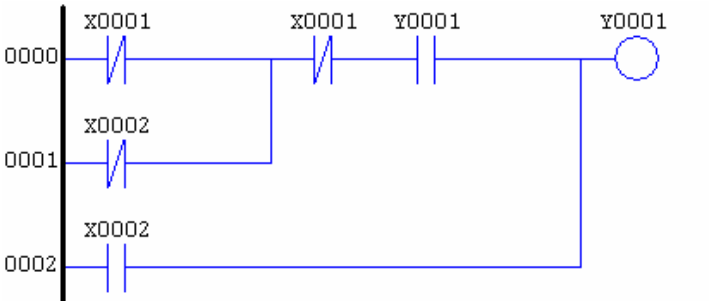


Figura 78 – Resultado 6 – Partida direta de motor

Os seis resultados apresentados, assim como os do estudo de caso anterior, também apresentaram diferenças de tamanho e formato. Novamente foi observada a presença de introns através de códigos não funcionais (linha 000 do resultado 3 e linha 000 do resultado 4).

e) tabelas de coletas de dados das simulações e gráficos de desempenho.

A seguir são apresentados os resultados de 1500 simulações realizadas para o estudo de caso do sistema de partida direta de motor. Em todas essas simulações foi utilizado elitismo, seleção por torneio e taxa de reprodução de 10%.

A Tabela 5 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 80%

- Mutação: 10%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 5 – Simulações com Cross-Over 80% e Mutação 10% – Partida direta de motor

Máximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	73	150	73	48,6667%	48,6667%	6,9
50	100	57	250	130	57,0000%	52,0000%	6,3
100	80	59	330	189	73,7500%	57,2727%	5,4
150	50	37	380	226	74,0000%	59,4737%	5,1
200	30	22	410	248	73,3333%	60,4878%	5,0
300	30	22	440	270	73,3333%	61,3636%	4,8
400	30	21	470	291	70,0000%	61,9149%	4,8
500	30	21	500	312	70,0000%	62,4000%	4,7
Total	500	312					

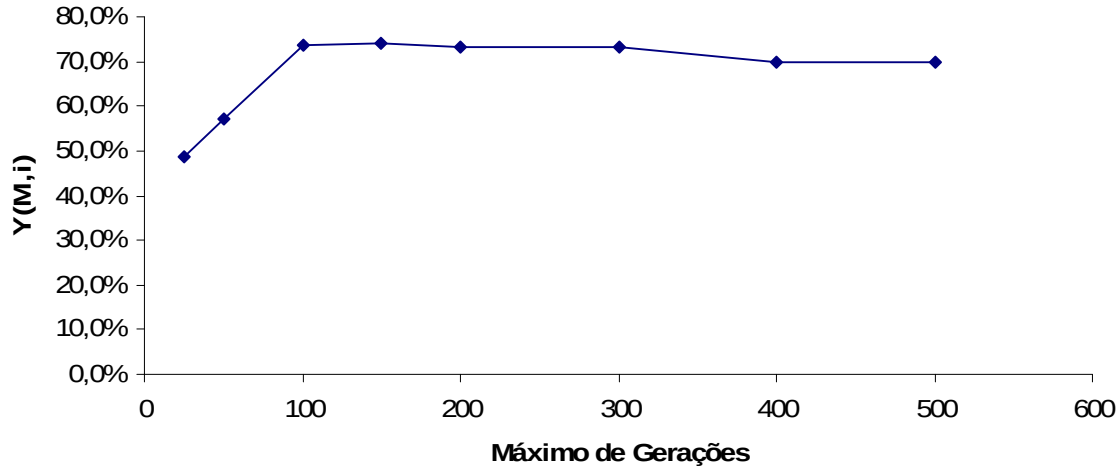


Figura 79 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 80% e mutação 10% – Partida direta de motor.

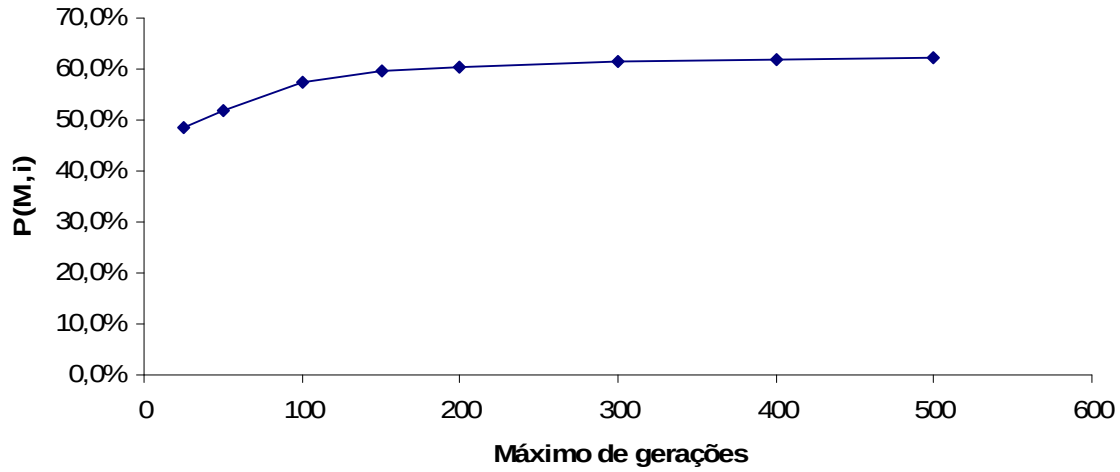


Figura 80 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 80% e mutação 10% – Partida direta de motor.

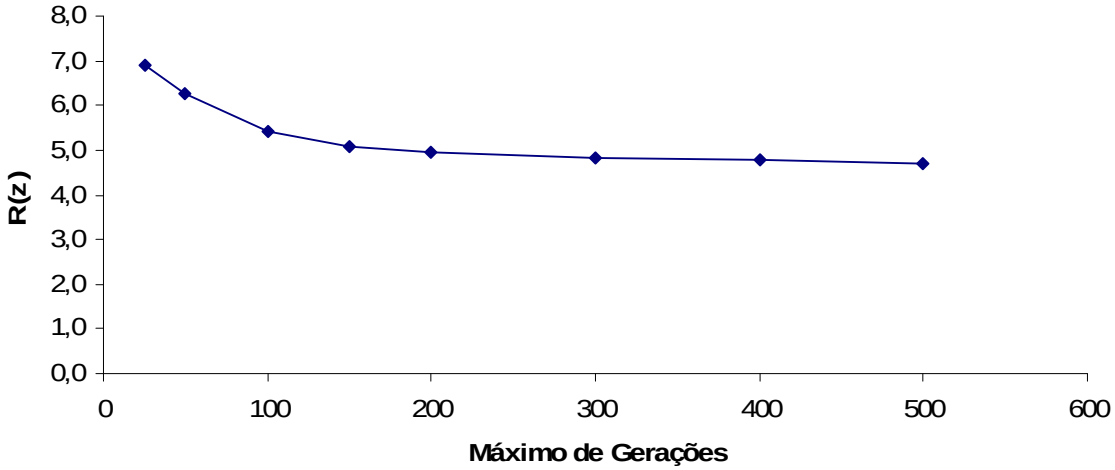


Figura 81 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 80% e mutação 10% – Partida direta de motor.

A Tabela 6 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 45%

- Mutação: 45%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 6 – Simulações com Cross-Over 45% e Mutação 45% – Partida direta de motor

Máximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	91	150	91	60,6667%	60,6667%	4,9
50	100	70	250	161	70,0000%	64,4000%	4,5
100	80	61	330	222	76,2500%	67,2727%	4,1
150	50	41	380	263	82,0000%	69,2105%	3,9
200	30	29	410	292	96,6667%	71,2195%	3,7
300	30	30	440	322	100,0000%	73,1818%	3,5
400	30	29	470	351	96,6667%	74,6809%	3,4
500	30	30	500	381	100,0000%	76,2000%	3,2
Total	500	381					

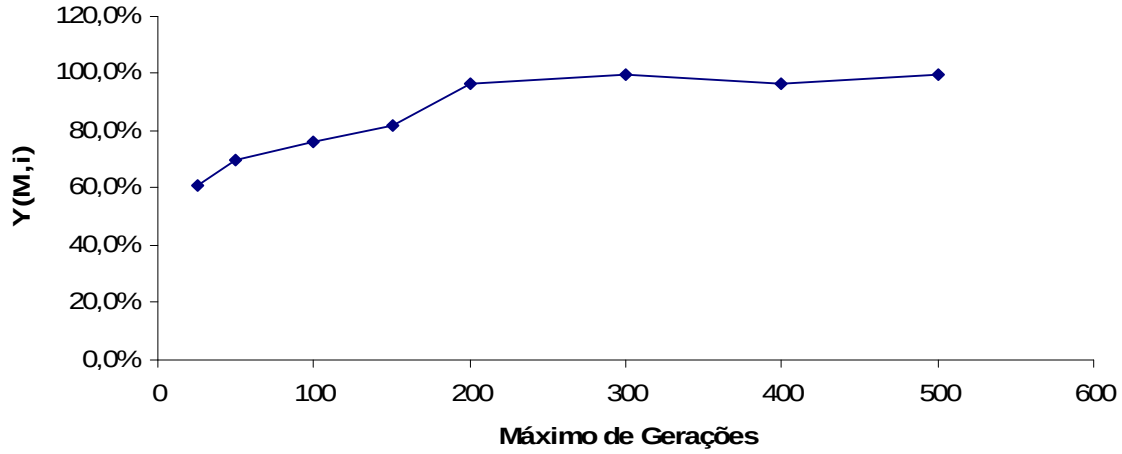


Figura 82 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Partida direta de motor

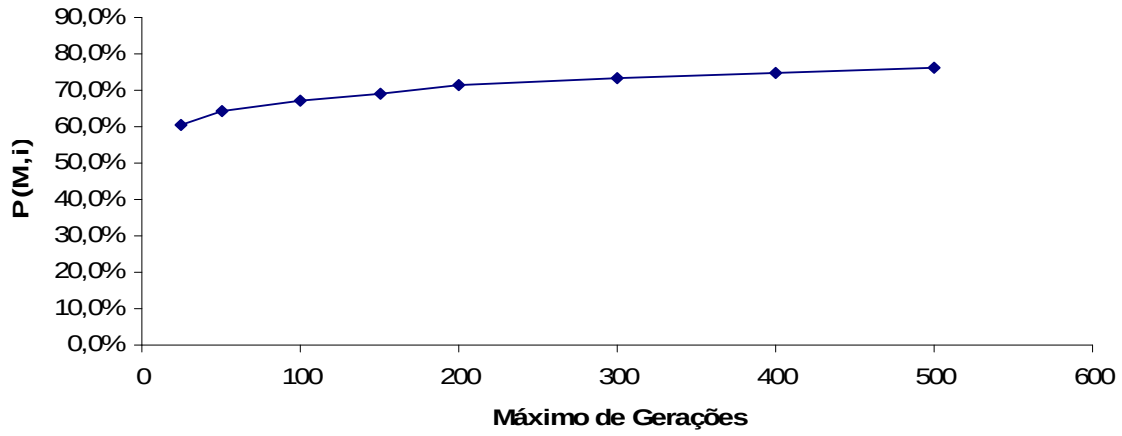


Figura 83 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Partida direta de motor.

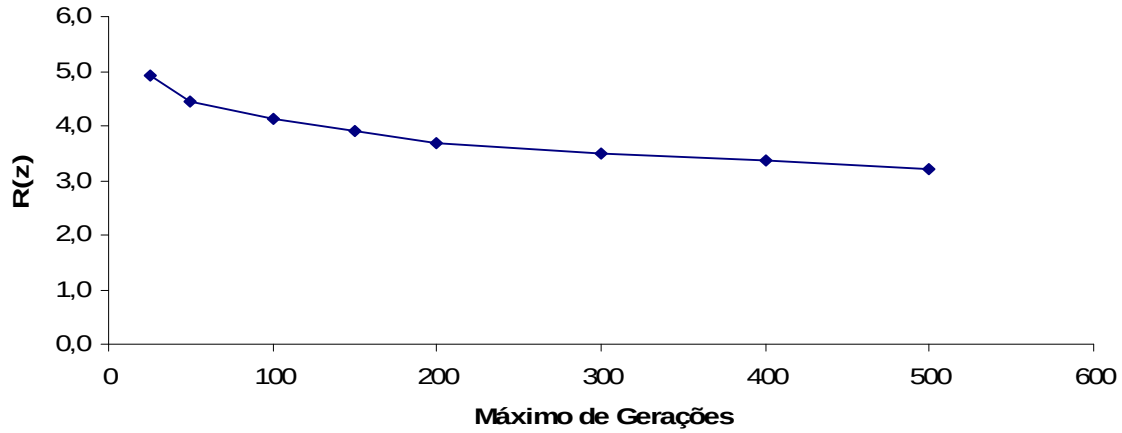


Figura 84 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 45% e mutação 45% – Partida direta de motor.

A Tabela 7 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 10%

- Mutação: 80%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 7 – Simulações com Cross-Over 10% e Mutação 80% – Partida direta de motor

Máximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	99	150	99	66,0000%	66,0000%	4,3
50	100	83	250	182	83,0000%	72,8000%	3,5
100	80	79	330	261	98,7500%	79,0909%	2,9
150	50	50	380	311	100,0000%	81,8421%	2,7
200	30	30	410	341	100,0000%	83,1707%	2,6
300	30	30	440	371	100,0000%	84,3182%	2,5
400	30	30	470	401	100,0000%	85,3191%	2,4
500	30	30	500	431	100,0000%	86,2000%	2,3
Total	500	431					

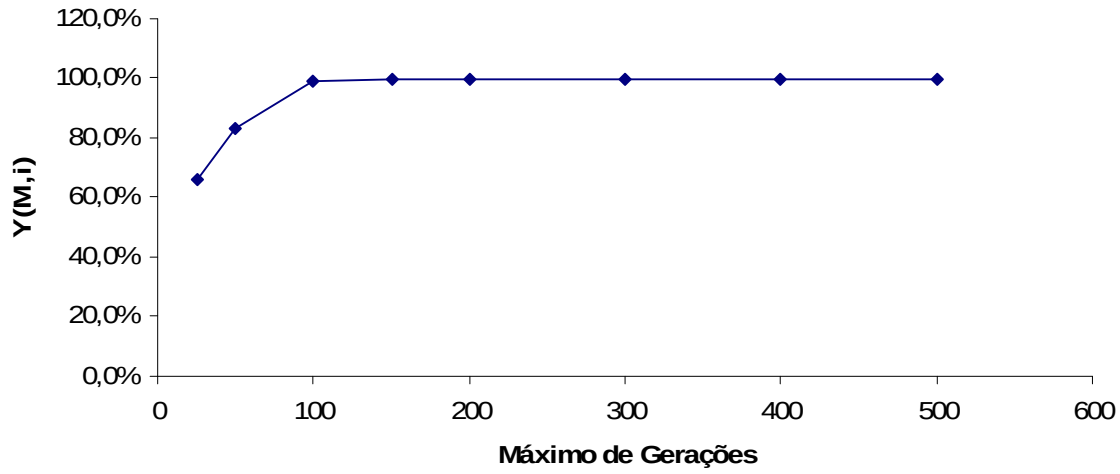


Figura 85 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Partida direta de motor

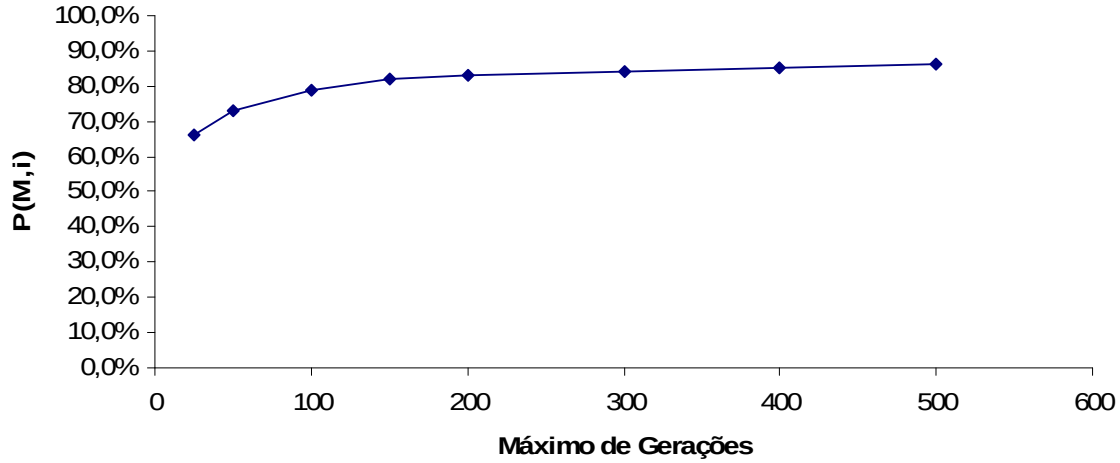


Figura 86 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Partida direta de motor.

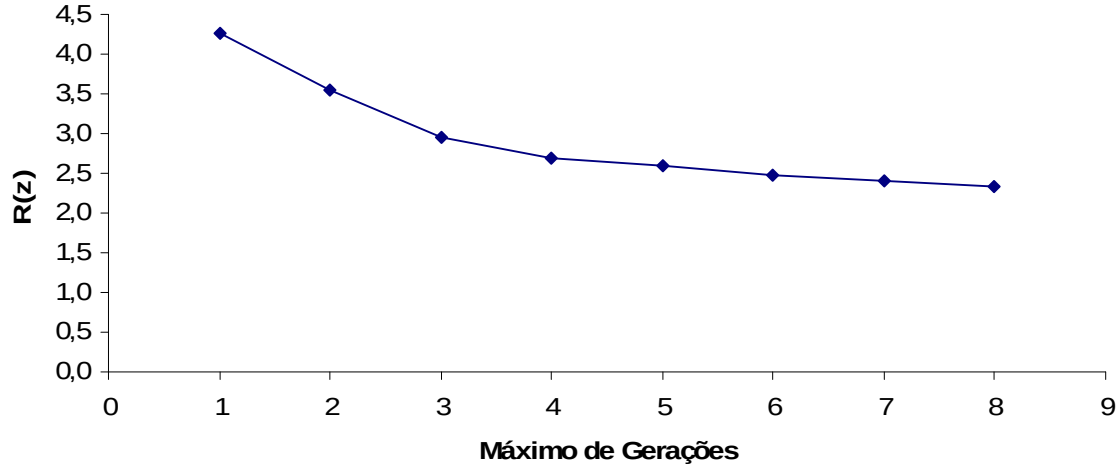


Figura 87 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 10% e mutação 80% – Partida direta de motor.

f) as discussões acerca dos resultados obtidos e do comportamento do programa.

Pela Tabela 5 observou-se que das 500 simulações, 312 obtiveram sucesso, ou seja, 63,4%. A Tabela 6 apresentou índice de 76,2% de sucesso e a (tabela 7), 86,2% de sucesso. Constata-se que pela simplicidade deste estudo de caso, os níveis de sucesso foram bastante altos. O índice R(z) da Tabela 5 indica que simulando o algoritmo de 4 a 6 vezes já é mais provável obter um sucesso.

A Figura 79 e Figura 80 indicaram que com o aumento do número de gerações o número de sucessos também aumenta, porém o nível de obtenção de soluções permaneceu estável. O gráfico de R(z) da Figura 81 permanece acima de 4 e abaixo de 7 simulações para a obtenção de um resultado.

O aumento da taxa de mutação para 45% e em seguida para 80% indicou nítida elevação de desempenho do algoritmo. Por se tratar de um programa de solução pequena, as mudanças causadas por mutação passam a ser maiores e o cross-over por não ter muita flexibilidade em um espaço reduzido de comandos acaba aproximando sua ação de um operador de mutação.

A partir das simulações com índice de mutação de 45%, na marca de 200 gerações, praticamente todas as simulações encontraram a solução ideal para o problema. Com esses parâmetros iniciais a simulação do algoritmo de 3 a 5 vezes já é o bastante para se obter uma solução com 99% de certeza com o parâmetro do máximo de gerações variando de 25 a 500 gerações.

A Tabela 7 passa a indicar um índice de sucesso próximo de 100% para todas as simulações com número máximo de gerações acima de 100. Para as três tabelas de coleta de dados os índices de R(z) permaneceram abaixo de 10 e contando dentre todas as 1500 soluções, foram obtidas 1124 soluções.

A programação genética mostrou-se eficiente na busca da solução para este estudo de caso. Pelo estudo de caso I, observa-se que a eficiência da técnica é dependente da complexidade do programa. Também foram encontrados introns nas soluções para esse problema, o que reforça a necessidade de se analisar a solução obtida antes de se transferi-la para um CLP.

6.3 Estudo de Caso III – Partida Direta de Motores com Reversão

O terceiro estudo de caso é referente a um sistema de automação industrial com um programa de automação um pouco maior e mais complexo para o acionamento direto de motores com a função de reversão do sentido de rotação.

a) Descrição física do sistema:

A instalação física do sistema possui o motor, o CLP e três botoeiras.

b) Funcionamento do sistema:

- A botoeira I2 executa a partida direta do motor no sentido horário.
- A botoeira I3 executa a partida direta do motor no sentido anti-horário.
- A botoeira I1 desliga a alimentação dos motores.
- Quando o motor está girando em um sentido a botoeira que executa a rotação no outro sentido não executa nenhum comando até que o motor tenha sido desligado da alimentação através da botoeira I1.

c) Cenários de teste

O primeiro cenário de teste (Figura 88) representa a situação em que o motor está inicialmente desligado (Q1 = 0 e Q2 = 0) e após alguns instantes a botoeira I2 é acionada. O motor então passa a girar no sentido horário (Q1 = 1). O motor deve permanecer girando se nada for feito em seguida.

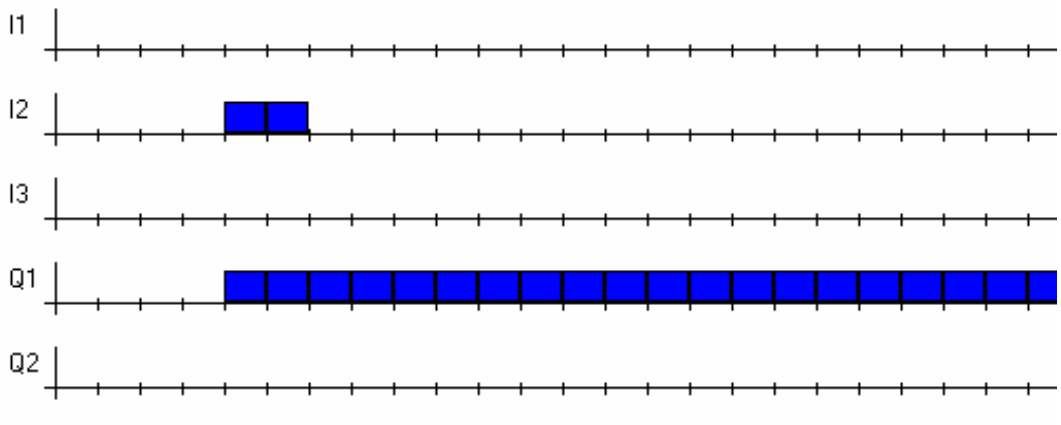


Figura 88 - Cenário 1 – Partida direta de motor com reversão

O cenário 2 (Figura 89) demonstra uma situação semelhante à anterior com a diferença da ativação ser feita pela botoeira I3 e o motor girar no sentido anti-horário.

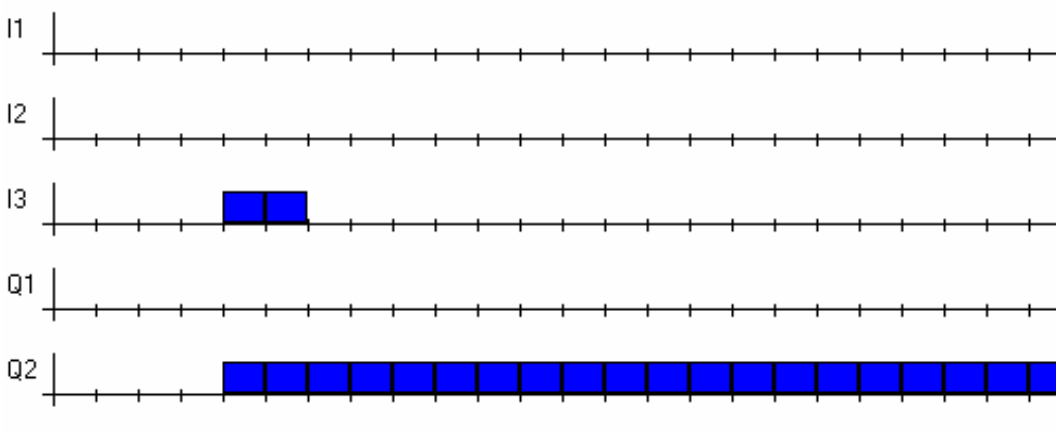


Figura 89 - Cenário 2 – Partida direta de motor com reversão

O cenário 3 (Figura 90) indica a situação em que o motor esta desligado e nenhuma botoeira é acionada. Logicamente nada deve acontecer.

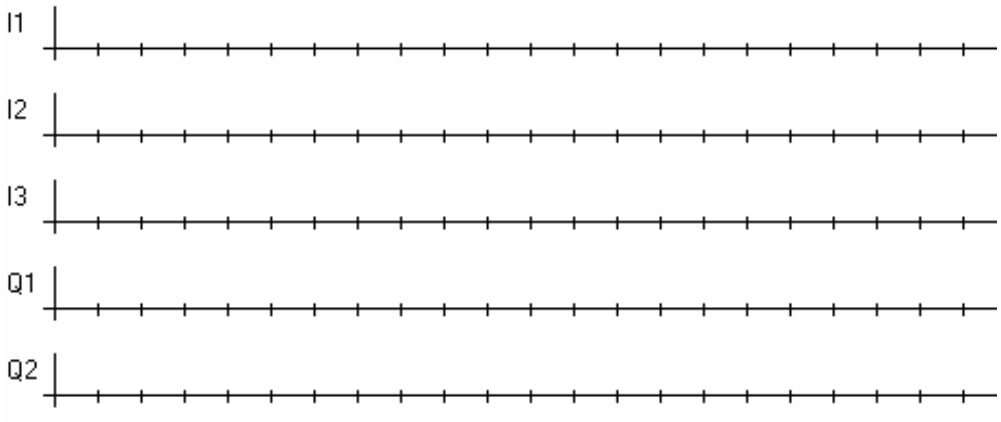


Figura 90 - Cenário 3 – Partida direta de motor com reversão

O cenário 4 (Figura 91) demonstra a situação em que o motor esta desligado e a botoeira de desligamento do motor é apertada diversas vezes. Esse cenário indica que essa botoeira não deve executar nenhuma ação em uma situação dessas.

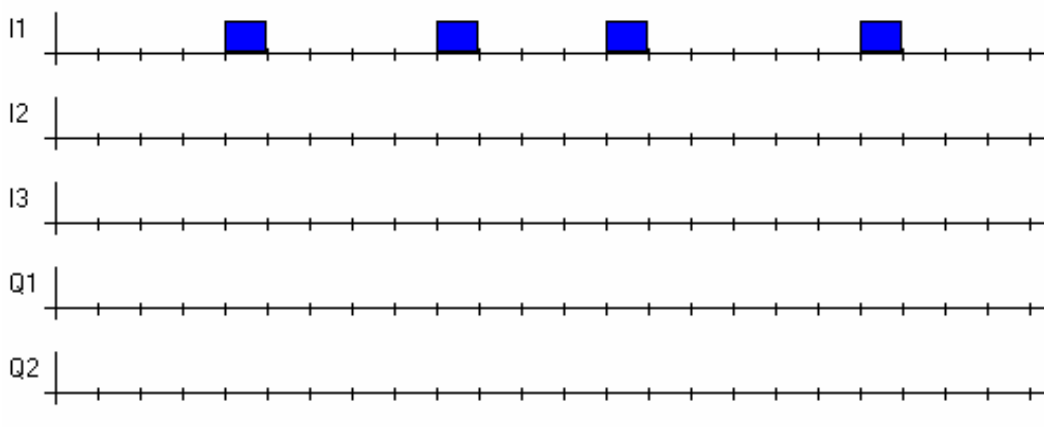


Figura 91 - Cenário 4 – Partida direta de motor com reversão

O cenário 5 (Figura 92) mostra a situação em que o motor está inicialmente parado, a botoeira I2 é acionada (I2 = 1), o motor começa a girar no sentido horário e então a botoeira I3, que ordena o giro no sentido anti-horário, é acionada diversas vezes (I3 = 1). Nessa situação o motor não deve obedecer ao comando da botoeira I3, pois isso poderia danificar o motor pela elevada corrente que passaria em seu enrolamento e pelo choque mecânico que seria produzido internamento no mesmo pela inversão súbita de sentido de rotação.

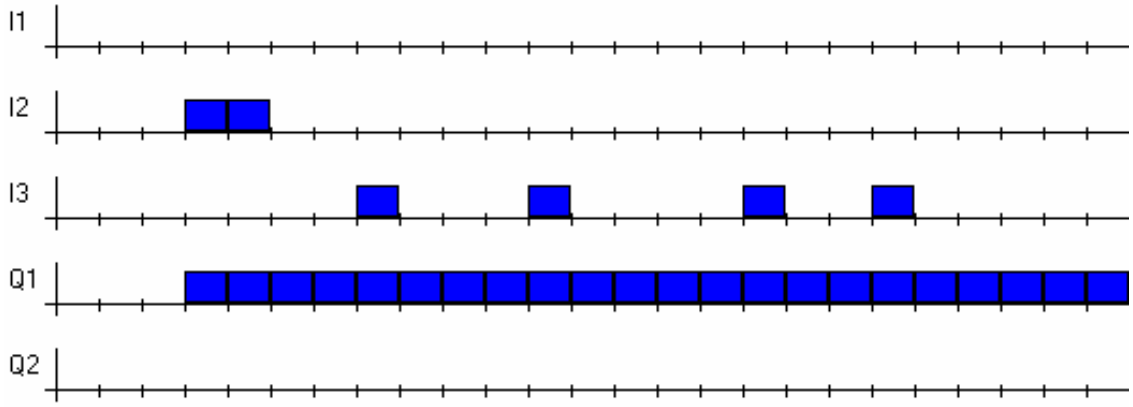


Figura 92 - Cenário 5 – Partida direta de motor com reversão

O cenário 6 (Figura 93) repete a situação anterior utilizando o sentido anti-horário de rotação. Nesse sentido o motor não deve obedecer aos comandos da botoeira I2.

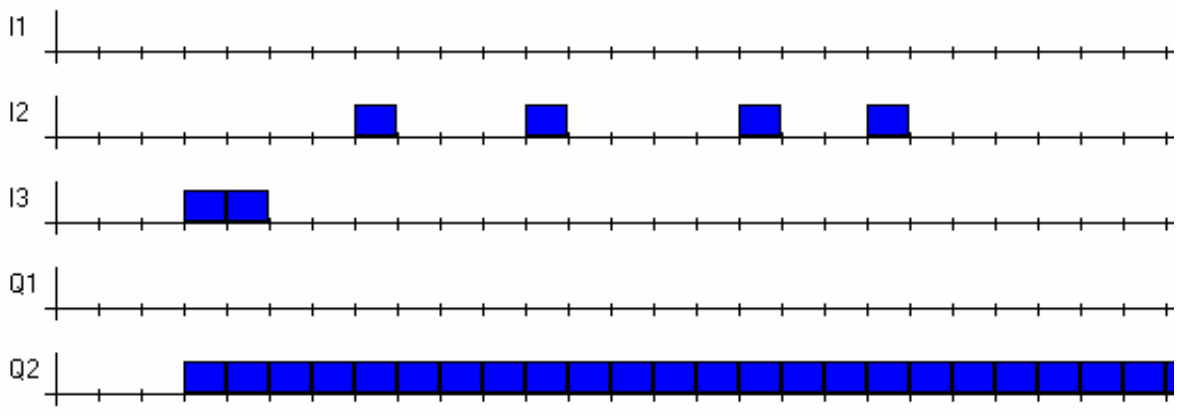


Figura 93 - Cenário 6 – Partida direta de motor com reversão

O cenário 7 (Figura 94) indica a situação em que o motor é acionado pela botoeira I2 (I2 = 1) e então começa a girar no sentido horário (Q1 = 1). Após alguns instantes o botoeira I1 é acionada (I1 = 1) para que o motor pare (Q1 = 0). Após alguns instantes a mesma situação é repetida.

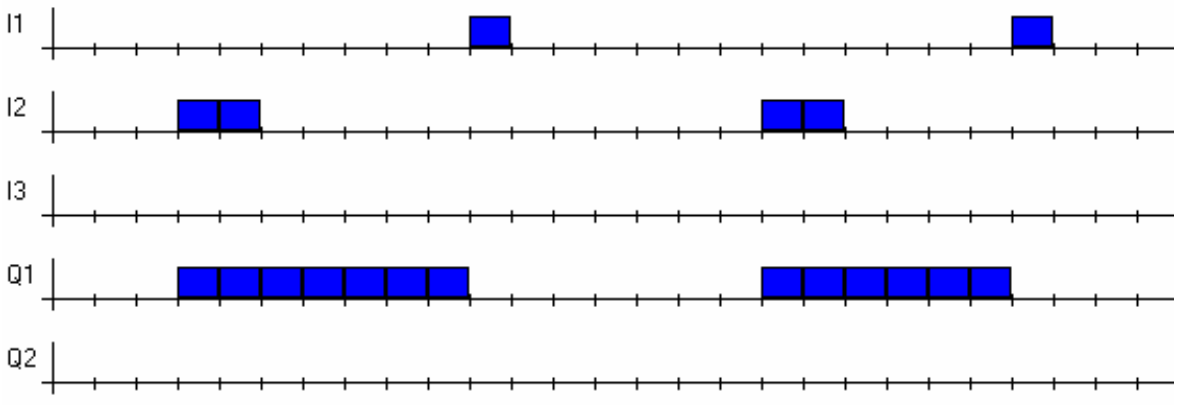


Figura 94 - Cenário 7 – Partida direta de motor com reversão

O cenário 8 (Figura 95) realiza uma ação semelhante ao do cenário 7 com a diferença do giro ser no sentido anti-horário.

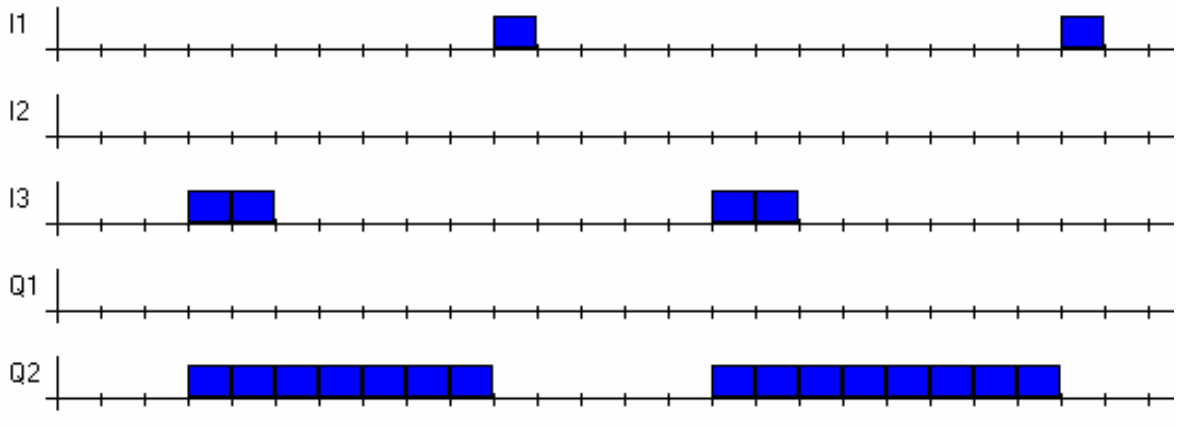


Figura 95 - Cenário 8 – Partida direta de motor com reversão

d) Discussões iniciais e soluções encontradas

Na Figura 96 é observado o esquema convencional para montagem das ligações para partida direta do motor com reversão. Na Figura 97 são mostrados os esquemas de ligação dos circuitos de comando com o CLP e do circuito de potência para alimentação do motor.

Em seguida, na Figura 98, observa-se o menor programa Ladder que executa essa operação. Este estudo de caso é mais complexo que o caso II e maior que o caso I com a simplificação de não possuir um temporizador.

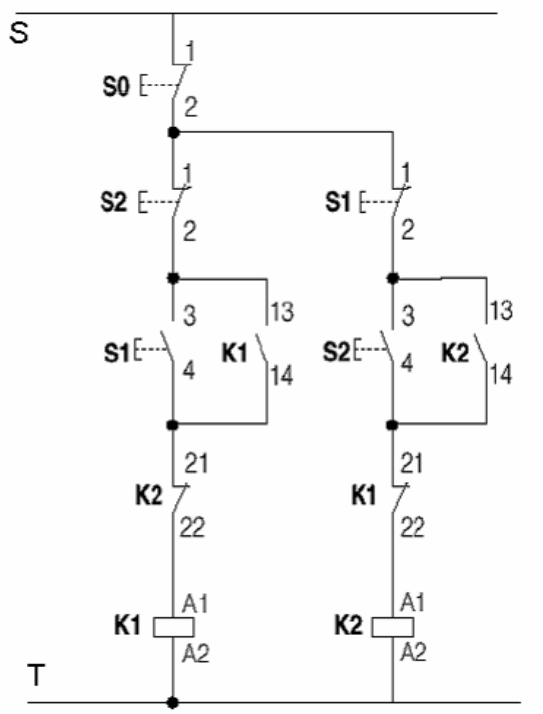


Figura 96 – Esquema convencional de partida de motor com reversão.

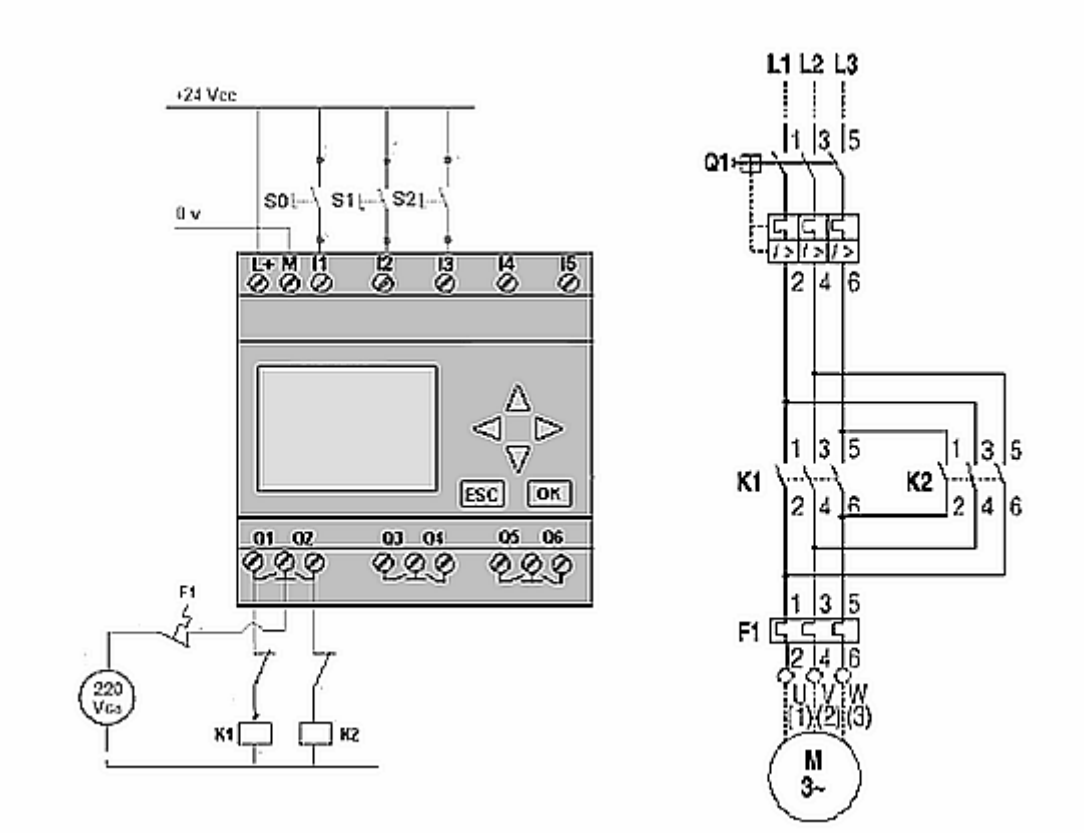


Figura 97 – Esquema de ligação do circuito de comando com o CLP e o circuito de potência para o motor.

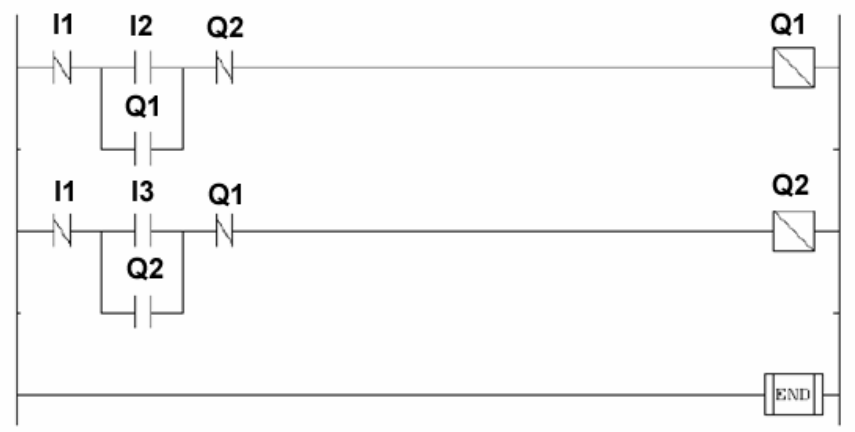


Figura 98 – Programa Ladder para controle de partida direta de um motor com reversão de sentido.

A seguir, na Figura 99, Figura 100, Figura 101 e Figura 102 são apresentados quatro soluções que evoluíram geneticamente a partir do programa desenvolvido.

Programa Instruction List do resultado 1

```
LD    I3
OR    Q1
LD    I1
ANB
OR    I2
OUTR  Q1
ANI   Q2
ORB
OUTR  Q1
ANDF
LD    I2
ANDP
LD    I2
AND   I2
LDI   Q1
ANI   Q1
ORB
AND   I3
OR    Q2
ANI   I1
OUT   Q2
END
```

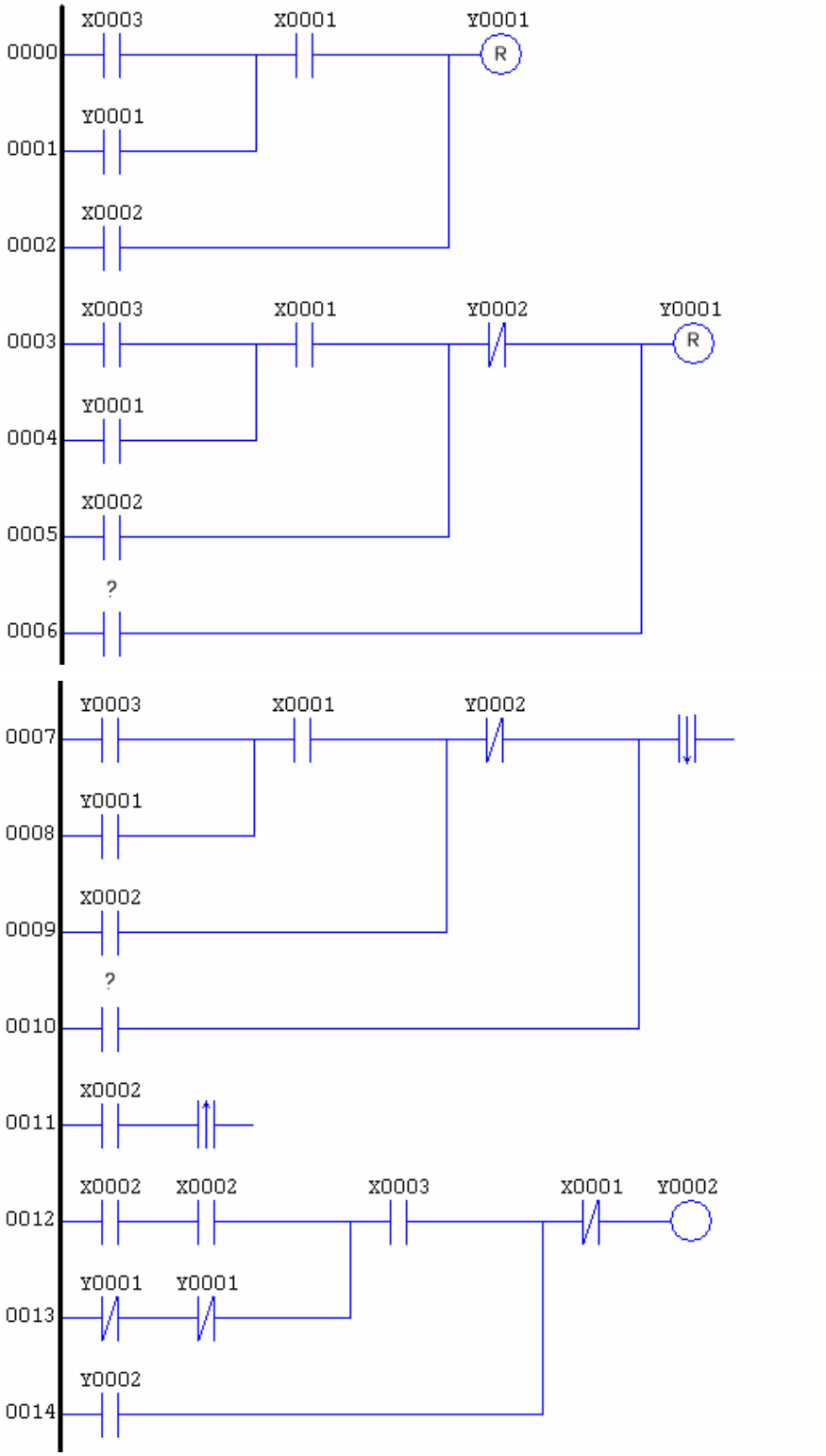


Figura 99 - Resultado 1 – Partida direta de motor com reversão

Programa Instruction List do resultado 2

```
ANDP  I3
ANDF  I2
ORB  Q2
AND  I1
LD  Q1
AND  I1
LD  I2
ORB  I1
ANDP  Q1
ANDP  0
ANI  Q2
OUTR  Q1
LDI  Q2
OR  I1
LD  I3
OR  I2
OR  Q2
OUT  Q2
ANB  Q2
ANI  Q1
OUTR  Q2
END  0
```

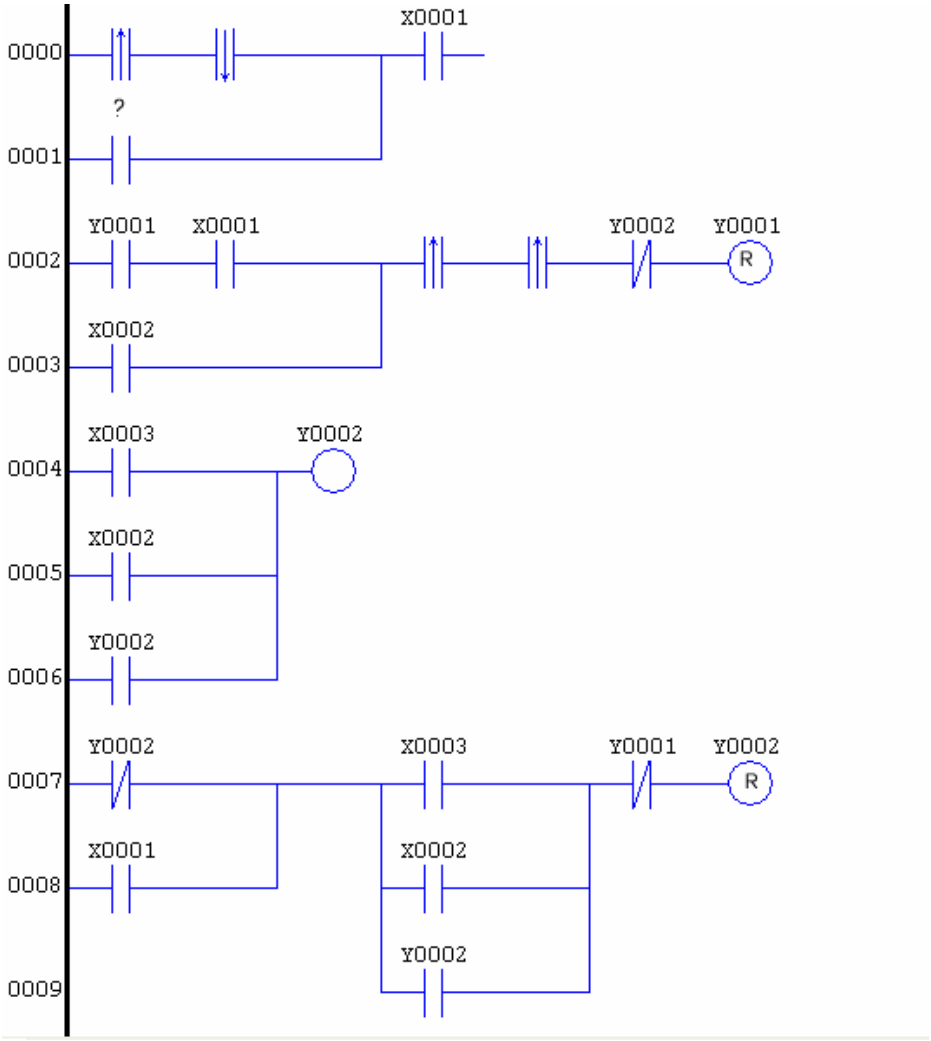


Figura 100 - Resultado 2 – Partida direta de motor com reversão

Programa Instruction List do resultado 3

OR Q2

ORI I2

ORB

ANDP

ANI Q2

LDI Q2

LD Q2

LD I1

ANB

LD I3

ORB

```
LD      I2
ANI     Q2
OUTR  Q2
OR      Q1
ANI     I1
OUT    Q1
ANDP
ORB
ANI     Q1
OUTR  Q2
END
```

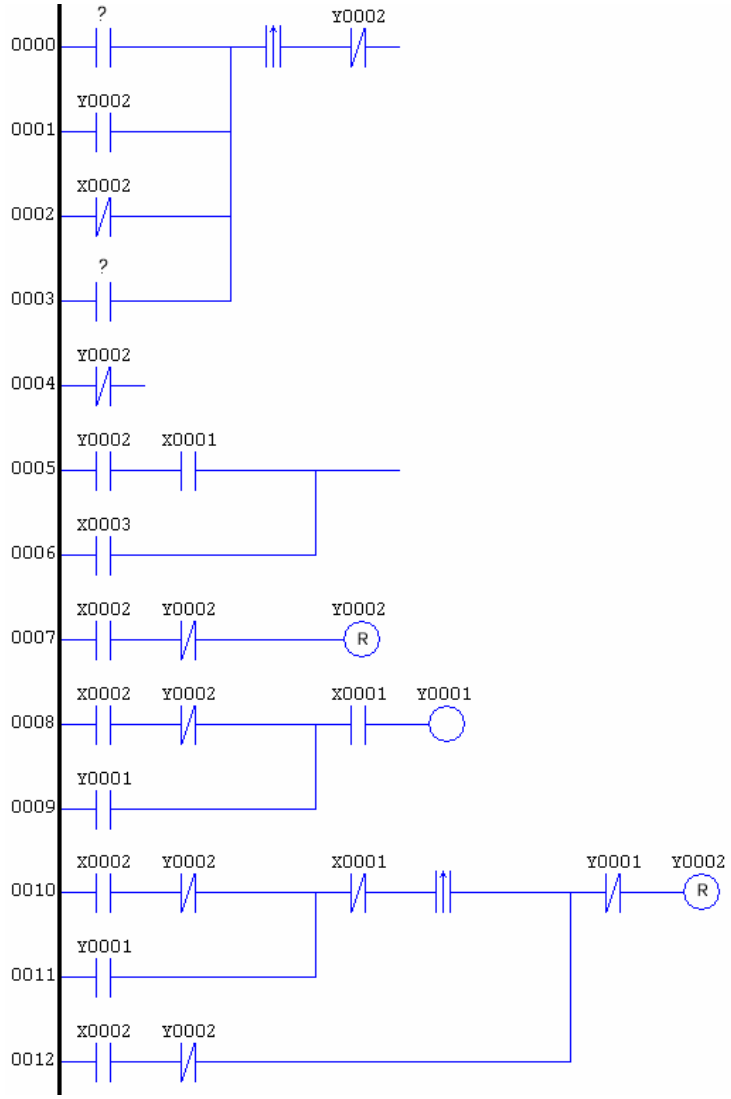


Figura 101 - Resultado 3 – Partida direta de motor com reversão

Programa Instruction List do resultado 4

```
ORI  Q2
LD  I3
ORI  I1
LD  Q2
ORI  I1
LD  I3
OUT  Q2
OR  I1
ANI  Q1
ANB
OUTR  Q2
LD  Q1
OR  Q1
AND  I1
LD  I2
LDI  Q2
ANB
ORB
ANDP
OUTR  Q1
LD  Q2
OR  Q1
END
```

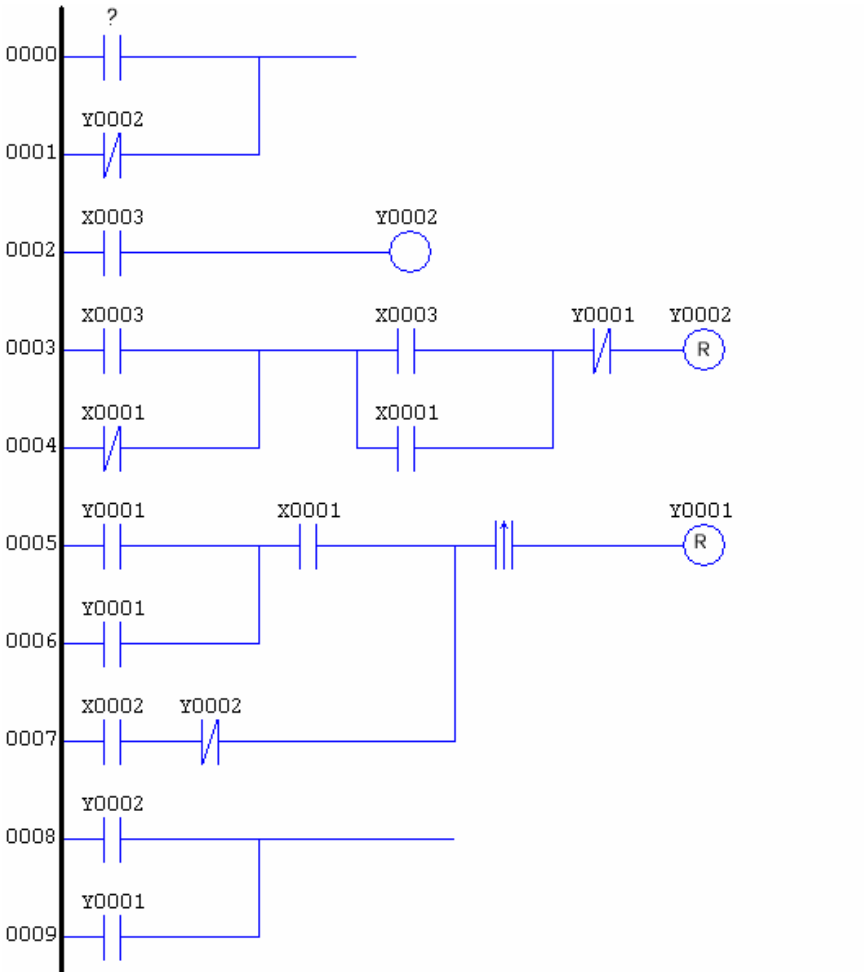


Figura 102 - Resultado 4 – Partida direta de motor com reversão

Os quatro resultados apresentados possuem complexidade maior que dos casos anteriormente estudados. Nas soluções foram encontrados segmentos maiores de introns como pode ser observado no resultado 1 entre as linhas 0007 e 0011. No resultado 1 (Figura 99) também podem ser observados contatos sem referência (linha 0006, 0010). Outro ponto a ser observado é a presença de dois conjuntos de operadores lógicos ambos ligados em bobinas com referências Y0001, que representam a rotação do motor no sentido horário. O primeiro segmento vai da linha 0000 até 0002 e o segundo segmento da linha 0003 até a linha 0006.

Pela teoria já apresentada, a utilização de uma mesma referência de saída mais de uma vez em um programa não é recomendável pelo fato das saídas do CLP serem atualizadas apenas de acordo com o resultado apresentado na última referência repetida do programa Ladder, ou Instruction List. Apesar de apenas o último contato ser utilizado para definir o estado real da saída do equipamento a presença de seguimento de código ligado a

uma referência repetida altera o comportamento do segmento de código ligado à última referência. Portanto, a repetição de referências não é recomendável, mas se o simulador indica que esse programa soluciona o problema então o segmento de código ligado à bobina repetida não pode ser retirado. A retirada do código da linha 0000 até a linha 0002 muda o comportamento geral do programa.

Um segmento de código que pode ser eliminado é o conjunto de linhas de 0007 até 0011, pois todas essas partes representam introns, ou seja, código não-funcional. Se ao final dessas linhas de código houvesse uma bobina então esses segmentos deixariam de ser introns e passariam a possuir funcionalidade.

O resultado 2 (Figura 100) possui introns nas linhas 0000 e 0001. Novamente é observada a repetição de bobinas (referência Y0002).

O resultado 3 (Figura 101) possui introns nas linhas que vão de 0000 até 0006, e repetição de referências Y0002.

O resultado 4 possui introns nas linhas 0000, 0001, 0008 e 0009. A linha 0002 pode não ser excluída pelo motivo da repetição da referência Y0002. Ela não representa um intron, pois nesse caso ela é um código funcional.

Pelos resultados apresentados pode-se concluir que apesar dos códigos obtidos possuírem maior extensão, algumas partes deles são compostas por introns e a retirada desses simplifica o programa. Observou-se também a presença da repetição de referências, o que resulta na consideração apenas da última referência do programa para a comutação da saída, porém a retirada desses segmentos produz mudança lógica podendo atrapalhar as funções do programa.

Em todos os programas, após a retirada dos códigos impróprios, observa-se que sempre restam as referências Y0001 e Y0002, que representam as saídas do CLP que executam as funções de rotação dos motores.

e) tabelas de coletas de dados das simulações e gráficos de desempenho.

A seguir são apresentados os resultados de 1500 simulações realizadas para o estudo de caso do sistema de partida direta de motor com reversão. Em todas essas simulações foi utilizado elitismo, seleção por torneio e taxa de reprodução de 10%.

A Tabela 8 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 80%

- Mutação: 10%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 8 – Simulações com Cross-Over 80% e Mutação 10% – Partida direta de motor com reversão.

Máximo de gerações	Nº de simulações	N de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	0	150	0	0,00	0,00	-
50	100	0	250	0	0,00	0,00	-
100	80	0	330	0	0,00	0,00	-
150	50	0	380	0	0,00	0,00	-
200	30	0	410	0	0,00	0,00	-
300	30	0	440	0	0,00	0,00	-
400	30	0	470	0	0,00	0,00	-
500	30	0	500	0	0,00	0,00	-
Total	500	0					

A Tabela 9 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 45%

- Mutação: 45%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 9 – Simulações com Cross-Over 45% e Mutação 45% – Partida direta de motor com reversão.

Maximo de gerações	Nº de simulações	N de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	0	150	0	0,0000%	0,0000%	
50	100	0	250	0	0,0000%	0,0000%	
100	80	0	330	0	0,0000%	0,0000%	
150	50	0	380	0	0,0000%	0,0000%	
200	30	0	410	0	0,0000%	0,0000%	
300	30	2	440	2	6,6667%	0,4545%	1011
400	30	2	470	4	6,6667%	0,8511%	539
500	30	9	500	13	30,0000%	2,6000%	175
Total	500	13					

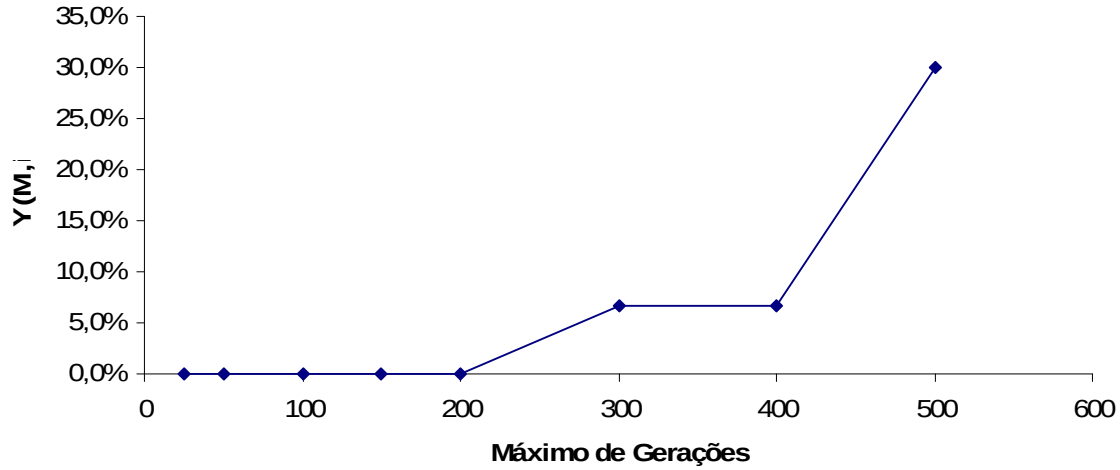


Figura 103 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Partida direta de motor com reversão.

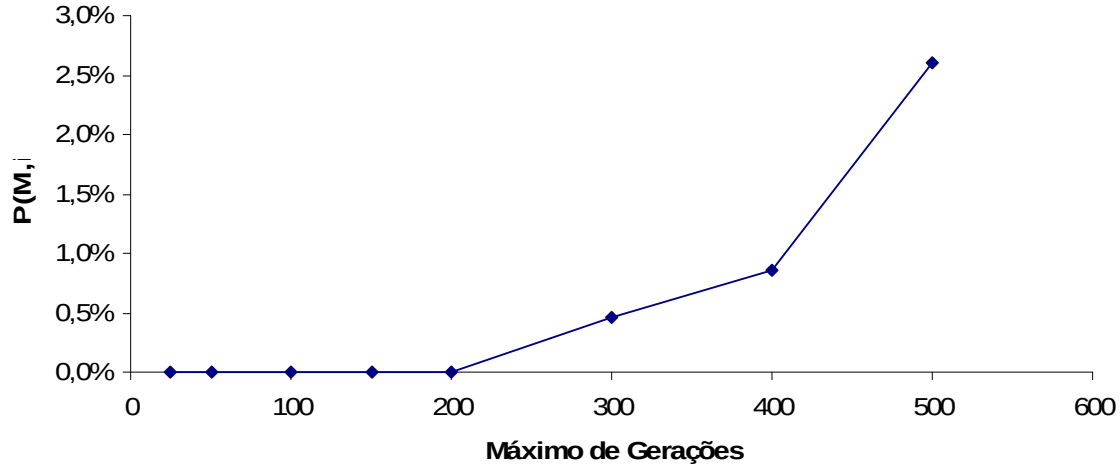


Figura 104 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Partida direta de motor com reversão.

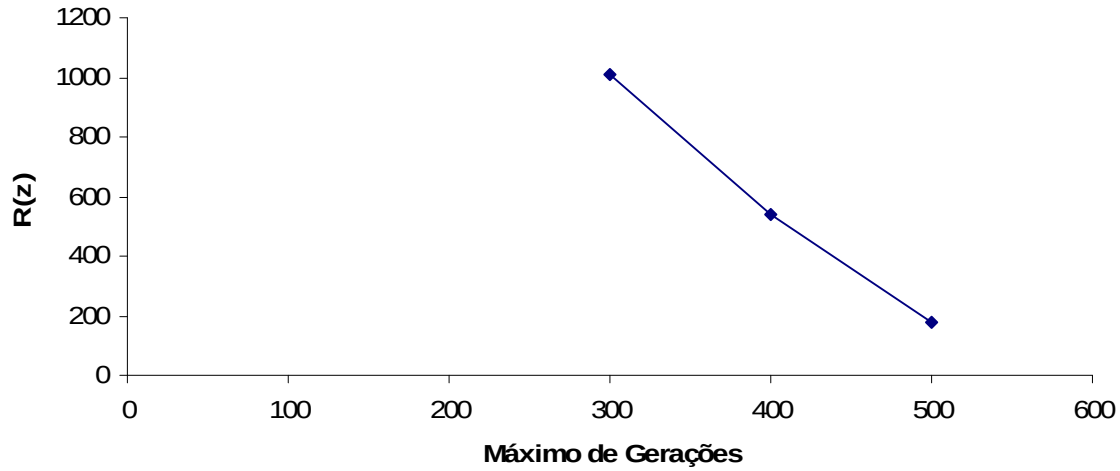


Figura 105 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 45% e mutação 45% – Partida direta de motor com reversão.

A Tabela 10 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 10%

- Mutação: 80%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 10 – Simulações com Cross-Over 10% e Mutação 80% – Partida direta de motor com reversão.

Maximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	0	150	0	0,0000%	0,0000%	
50	100	0	250	0	0,0000%	0,0000%	
100	80	2	330	2	2,5000%	0,6061%	758
150	50	4	380	6	8,0000%	1,5789%	289
200	30	2	410	8	6,6667%	1,9512%	234
300	30	11	440	19	36,6667%	4,3182%	104
400	30	11	470	30	36,6667%	6,3830%	70
500	30	14	500	44	46,6667%	8,8000%	50
Total	500	44					

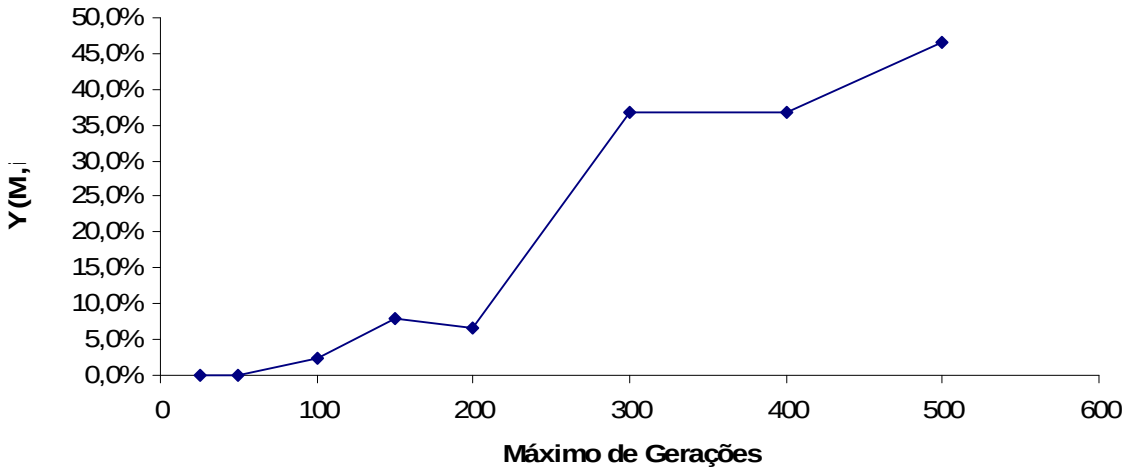


Figura 106 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Partida direta de motor com reversão.

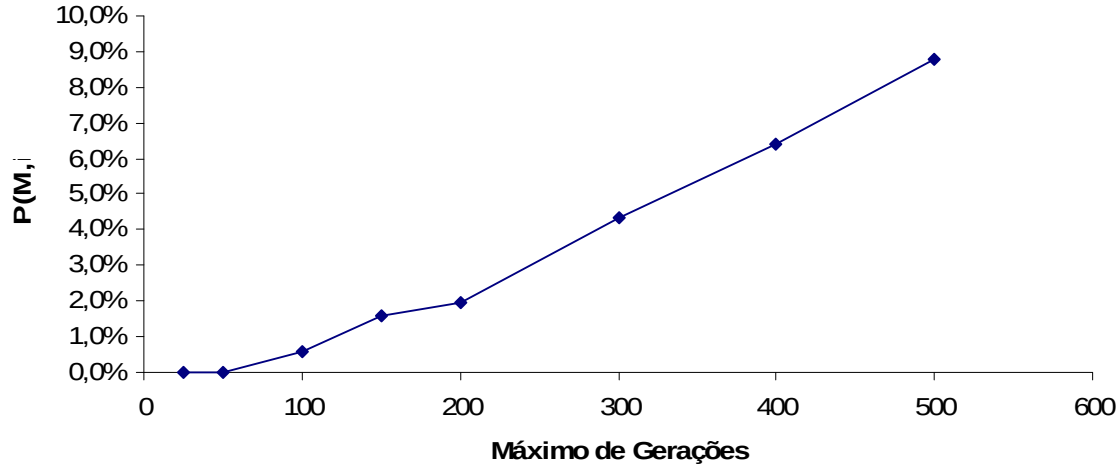


Figura 107 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Partida direta de motor com reversão.

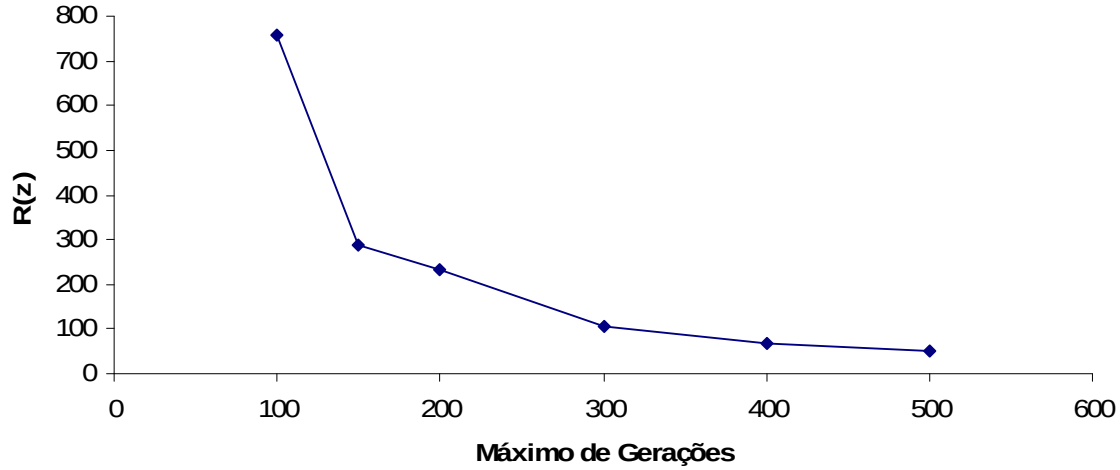


Figura 108 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 10% e mutação 80% – Partida direta de motor com reversão.

f) as discussões acerca dos resultados obtidos e do comportamento do programa.

Pela Tabela 8 observou-se que das 500 simulações, nenhuma obteve sucesso. Portanto nesse conjunto de simulações não pode ser feito o cálculo de nenhum Y(M,i), P(M,i) e R(z), nem traçar nenhum gráfico. A Tabela 9 passa a apresentar sucessos a partir de do limite de 300 gerações. Nesse nível ela demonstra duas soluções, e esse número permanece constante no limite de 400 gerações e cresce para nove no limite de 500 gerações.

Percebe-se que o processo no limite de 500 gerações já passa a ser viável, pois das 30 simulações, nove obtiveram sucesso, ou seja, 30%. Pelo fato do conjunto de simulações da

Tabela 9 não ter sido favorável, os cálculos de R(z) demonstram números de repetições bastante elevados para se obter uma solução. Isso indica que para esse estudo de caso alguns parâmetros de simulação devem ser excluídos. O gráfico de Y(M,i) da Figura 103 demonstra um salto satisfatório do número de gerações 400 para 500.

O gráfico de R(z) não possui resultados até a marca de 300 gerações. A partir desse ponto ele passa a apresentar seu comportamento decrescente padrão à medida que o número de gerações aumenta.

A Tabela 10 demonstra resultados melhores que os apresentados nas duas tabelas anteriores. Nenhum sucesso foi encontrado nos limites de geração 25 e 50, o que demonstra que a complexidade do problema exige um número maior de gerações para poder encontrar a resposta. A partir de 100 gerações o número de soluções se torna crescente, partindo de 2 soluções para 14 soluções (no limite de 500 gerações representa 46,66% se sucesso).

Os resultados desta tabela reforçam que o operador genético de mutação nesse caso se torna o principal agente na busca. Os gráficos da Figura 106, Figura 107 e Figura 108 passam a apresentam um comportamento mais próximo do padrão e os níveis de R(z) se tornam mais satisfatórios.

Esse estudo de caso deixa claro que para um problema mais extenso e de maior complexidade o operador genético de mutação, e o aumento do número de gerações na simulação se tornam indispensáveis para a viabilidade da técnica.

Apesar do número de respostas terem sido inferior ao dos estudos de caso já apresentados, foram encontradas 57 respostas. Considerando que destas, apenas uma única solução terá que ser escolhida para implementação em um sistema real considera-se que a técnica obteve eficiência satisfatória na solução do problema.

6.4 Estudo de Caso IV – Partida Estrela/Triângulo de Motores

O quarto estudo de caso é referente a um sistema de automação industrial com um programa para automação da partida estrela/triângulo de motores.

a) Descrição física do sistema:

A instalação física do sistema possui um motor, um CLP e duas botoeiras.

A finalidade de um motor em sistema estrela/triângulo é a de se reduzir a corrente de partida, ou seja, a corrente de pico no momento da partida. É necessário que o motor seja do tipo de rotor em gaiola de esquilo, e todas as pontas de ligação (seis se o motor for de seis pontas, doze se for de doze pontas) devem ser externas.

A tensão triângulo deve ser a mesma da rede (pois será nesta tensão que o motor irá trabalhar). A tensão em estrela (tensão de partida do motor) também será a mesma do triângulo, a diferença está no tipo de ligação dos terminais do motor (na ligação estrela o motor oferece mais resistência a passagem de corrente, diminuindo assim a corrente de pico).

b) Funcionamento do sistema:

- A botoeira I2 habilita o funcionamento do motor e inicia sua partida com alimentação em estrela.

- Após alguns segundos o sistema deve automaticamente trocar a topologia de alimentação do motor de estrela para triângulo. Durante essa etapa é preciso cautela para que o sistema não ative as duas formas de ligação ao mesmo tempo, o que poderia provocar um curto-circuito. O sistema deve garantir que quando a topologia estrela estiver ligada esta ligação impeça que a topologia triângulo seja conectada, e vice-versa. A comutação das formas de ligação deve ser realizada através do temporizador.

- A botoeira I1 desabilita a alimentação do motor.

c) Cenários de teste

O primeiro cenário de teste (Figura 109) indica o estado do sistema em que quando nenhuma botoeira é ativada, nenhuma função é exercida inicialmente.

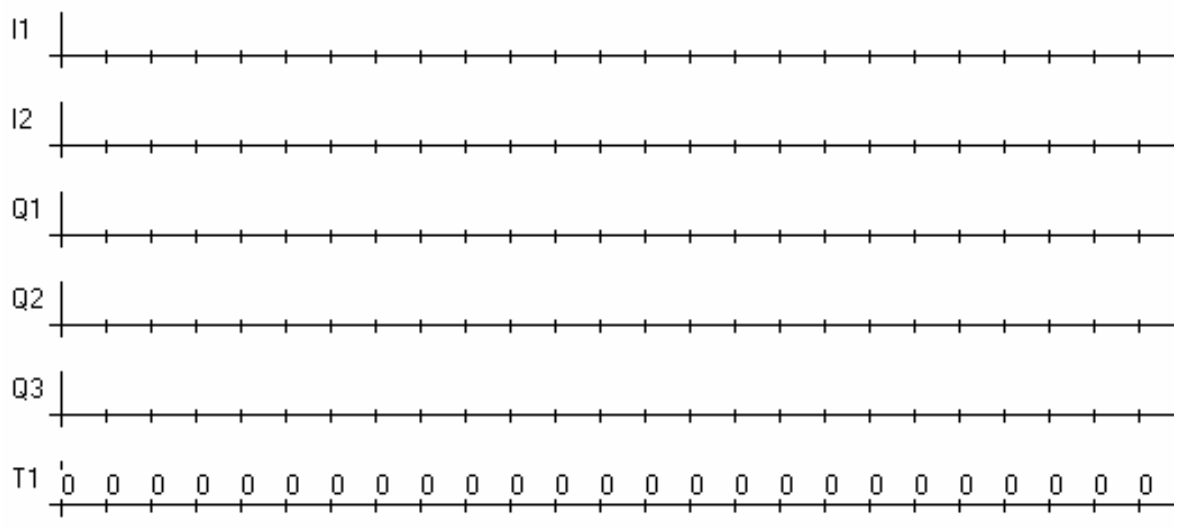


Figura 109 - Cenário 1 – Partida estrela/triângulo de motor.

O segundo cenário de teste (Figura 110) representa a situação em que a botoeira I2 é acionada (I2 = 1) e é armado um inter-travamento no sistema que inicia o processo de partida do motor. Esse inter-travamento da partida é exercido por Q1, enquanto ele estiver ativo o motor está habilitado para rodar. No instante que Q1 é ativo, a ligação alimentação do motor está em estrela (Q3 = 1). Após alguns instantes o modo de ligação estrela é desligada (Q3 = 0) e é feita a ligação em triângulo (Q2 = 1).

Quando a botoeira I1 é acionada (I1 = 1) todo o sistema deve parar, o funcionamento do motor é desabilitado e nenhuma topologia de ligação é feita no motor.

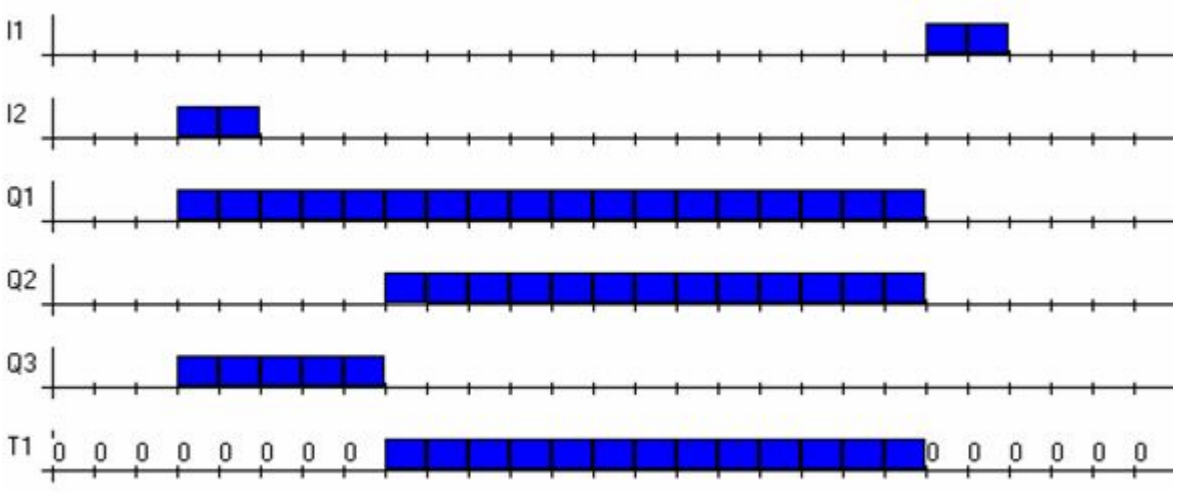


Figura 110 - Cenário 2 – Partida estrela/triângulo de motor.

O terceiro cenário de teste (Figura 111) representa a situação em que o motor recebe a partida e permanece rodando indefinidamente e I1 não é ativado em nenhum instante.

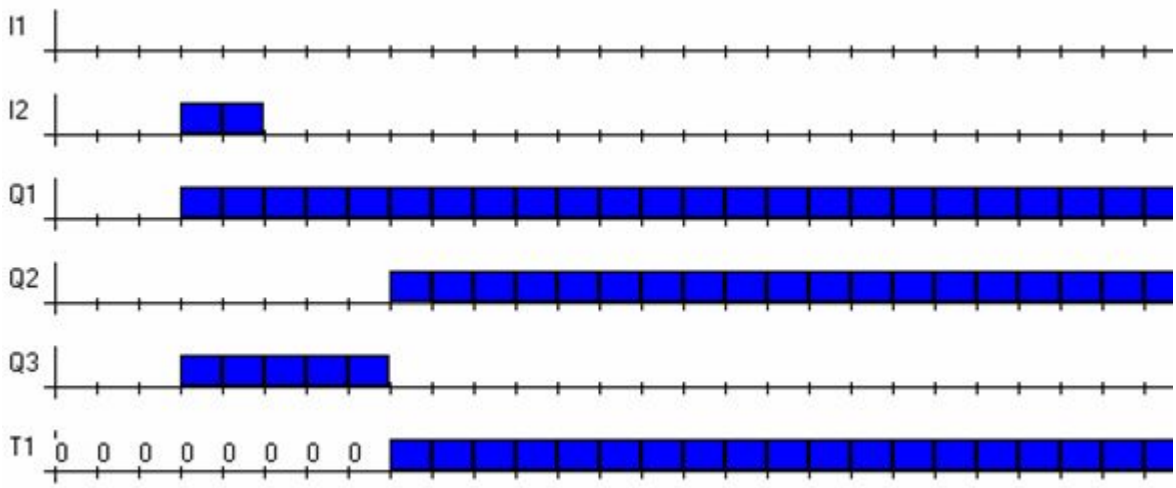


Figura 111 - Cenário 3 – Partida estrela/triângulo de motor.

O quarto cenário de teste (Figura 112) representa a situação do segundo cenário repetida duas vezes para garantir que o sistema funcione mais de uma vez. Esse tipo de cenário impede que seja escolhido um programa que ao final de sua execução armazene uma memória que impeça que ele seja executado novamente sem que o CLP seja resetado.

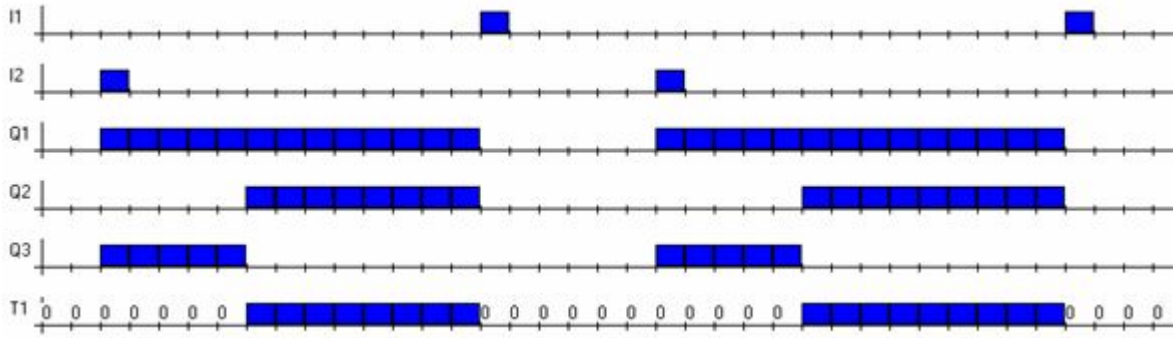


Figura 112 - Cenário 4 – Partida estrela/triângulo de motor.

d) Discussões iniciais e soluções encontradas

Na Figura 113 é observado o esquema convencional para montagem das ligações para partida estrela/triângulo do motor.

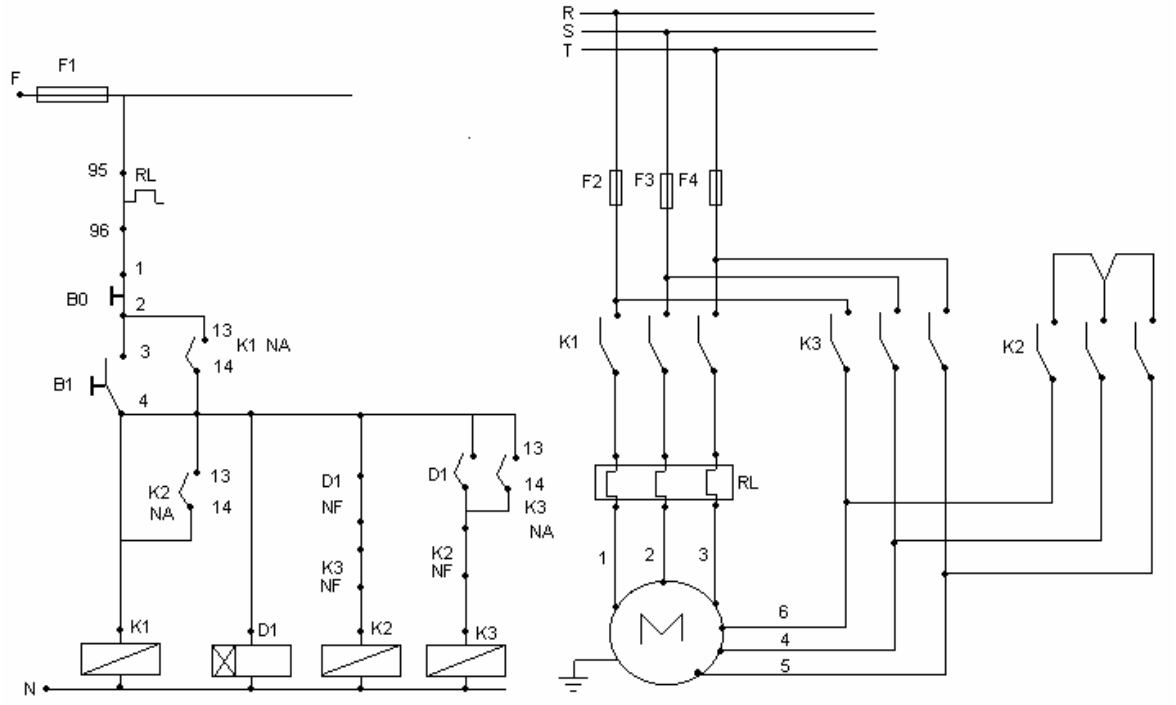


Figura 113 – Esquema de ligação convencional do sistema de controle e do sistema de alimentação do motor na partida estrela/triângulo

Em seguida, na Figura 114, observa-se o menor programa Ladder que executa essa operação. Este estudo de caso possui maior complexidade pela necessidade dos inter-travamentos do contato de habilitação do motor, do inter-travamento entre as ligações estrela/triângulo que devem impedir que elas sejam habilitadas ao mesmo tempo e do uso de um temporizador.

Outro fator que aumenta a dificuldade da busca são as poucas possibilidades de acionamento do sistema. Com a utilização de poucos cenários o cálculo do fitness é prejudicado pela redução da capacidade do algoritmo em diferenciar um programa do outro. Nesse caso existem poucos degraus de evolução entre as soluções da população inicial com fitness baixo e o indivíduo ideal para a solução do problema. Dessa forma a evolução dos programas não é feita aos poucos, pois para que haja um aumento no fitness é preciso que um conjunto maior de comandos no programa estejam integrados ao mesmo tempo no sentido de exercer a função do programa.

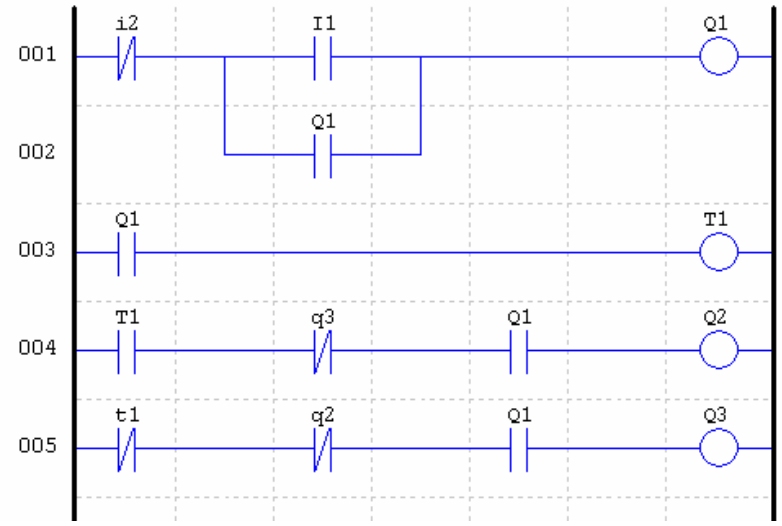


Figura 114 – Esquema de ligação convencional do sistema de controle e do sistema de alimentação do motor na partida estrela/triângulo

A seguir, na Figura 115, Figura 116 e Figura 117 são apresentadas três soluções que evoluíram geneticamente a partir do programa desenvolvido.

Programa Instruction List do resultado 1

LDI T1
LD I2
OR Q1
OUTR T1
ORI I1
OUT T1
LD Q1
AND T1
LD I1
OUTR T1
OR Q3
ANDF
ANI I2
OUT T1

```
LDI  Q3
ANDP
ANB
OUT  Q2
OR  I2
OUTR  Q3
OUT  Q2
LD  Q1
AND  T1
OUT  Q2
ANI  I1
ANI  Q3
OUT  Q2
ANI  Q1
AND  I2
OR  I2
OUT  Q3
ANDP
LDI  T1
ORB
AND  Q1
OR  Q3
OUTR  Q1
OUT  Q3
OUTR  Q1
ANI  Q3
LD  Q3
ANB
END
```

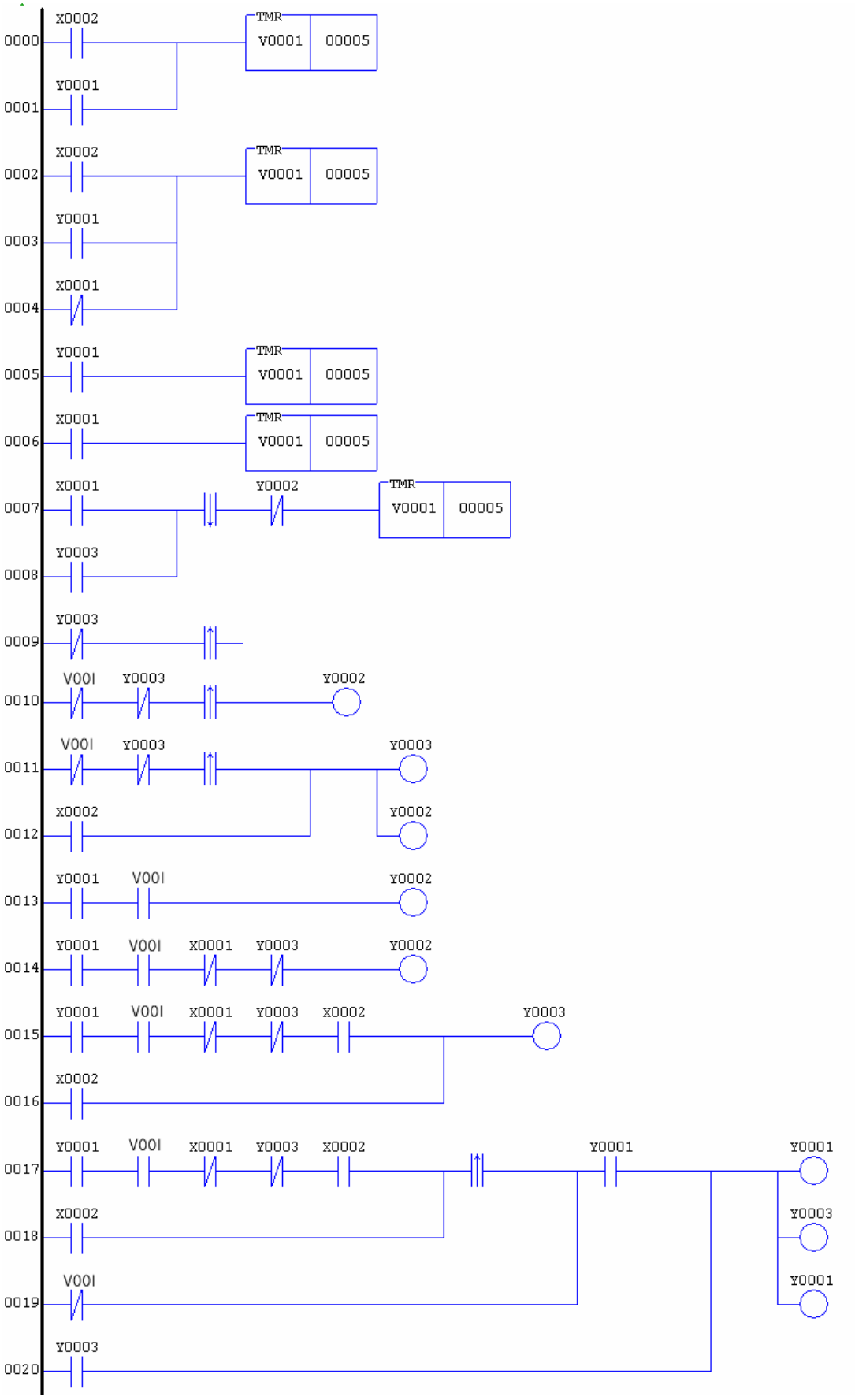


Figura 115 - Resultado 1 – Partida estrela/triângulo do motor.

Programa Instruction List do resultado 2

```
ANDP
ANI  T1
ORI  Q2
ORB
ANDP
LD  T1
OUT  Q1
OUT  Q2
ORB
ANDF
ANI  Q3
ORI  Q1
LD  I1
OUTR  Q1
OR  I2
OUTR  Q1
ORB
ANB
ANDF
ORI  Q2
AND  Q1
ANDF
ANDF
AND  Q1
ANI  I1
OR  Q1
OUT  T1
ANI  T1
ANI  T1
OUT  Q3
AND  I1
```

```
ANI  Q3
ANDP
LDI  I1
LD  I2
OR  I2
ORB
ANDP
OR  T1
AND  Q2
OUT  Q2
END
```

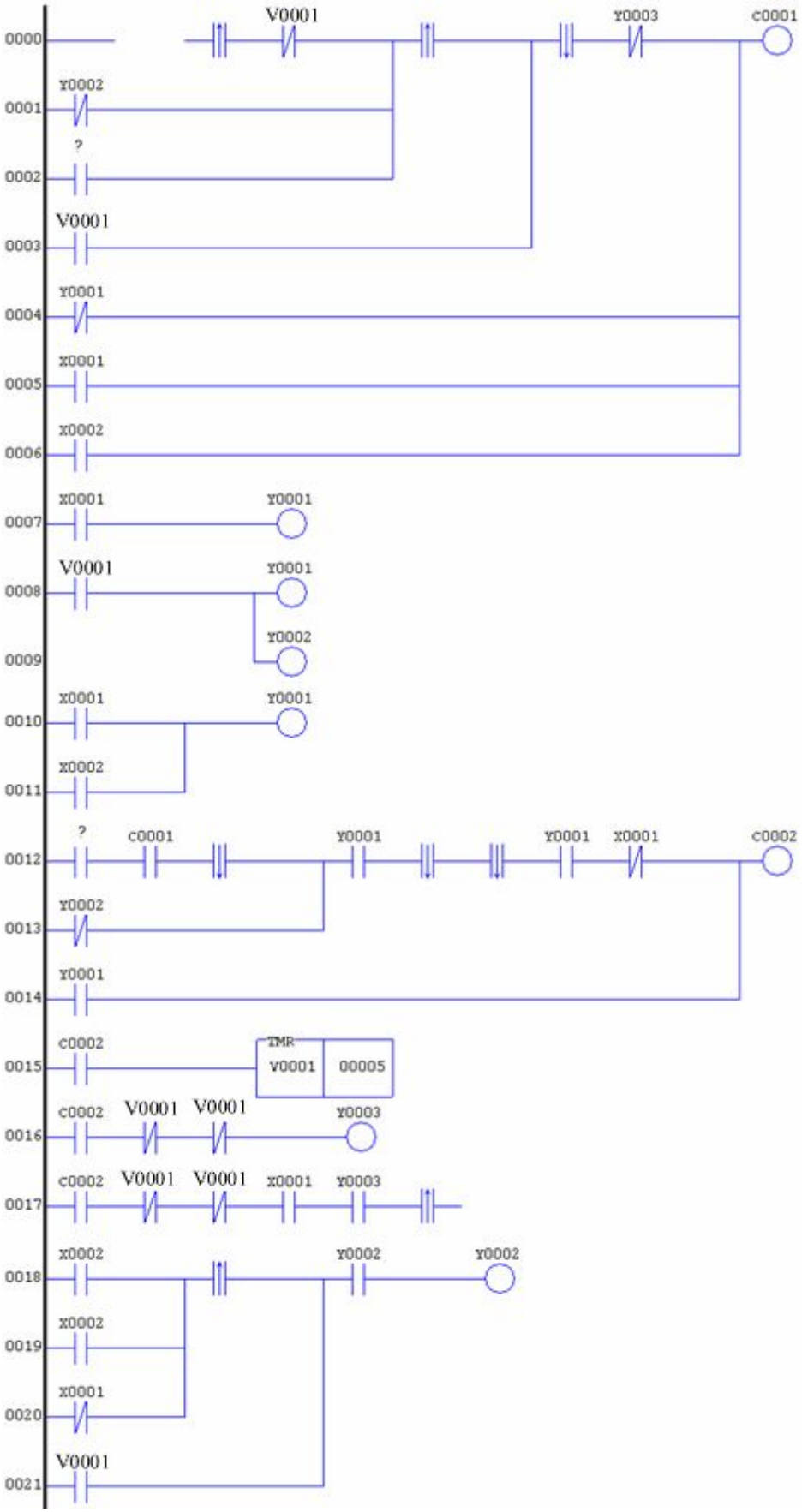


Figura 116 - Resultado 2 – Partida estrela/triângulo do motor.

Programa Instruction List do resultado 3

```
OR  I1
ANI  Q3
AND  I2
LD  I1
OR  I2
OR  I1
OUTR  Q1
LD  I2
OUTR  T1
OR  Q3
ANDP
ANDF
ANDF
AND  I1
LD  Q2
ANB
ANB
LDI  I1
AND  T1
OUT  Q2
LDI  I1
LD  Q1
OR  I2
OUT  T1
ANB
ANI  T1
AND  Q3
OR  I2
OUT  Q3
OR  I1
```

```
OUTR  Q1
ORI  I1
ANDP
LD  Q2
AND  Q3
LD  I1
OUTR  T1
ANDF
LDI  Q3
ORB  Q3
ANI  I1
LDI  Q2
END  0
```

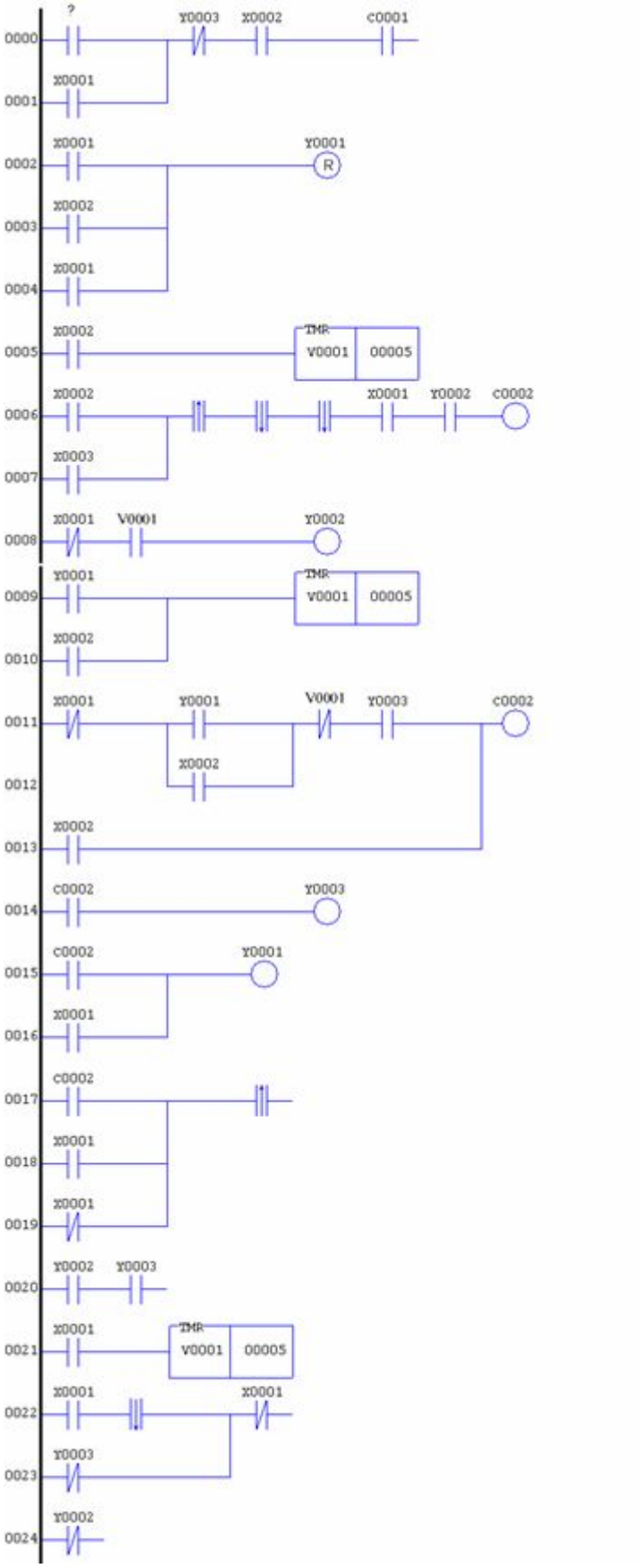


Figura 117 - Resultado 3 – Partida estrela/triângulo do motor.

Os três programas Ladder apresentados foram os mais complexos apresentados até agora. Como esperado, o programa demonstrou mais dificuldade em elaborar soluções para o problema devido às poucas opções de simulação.

Foram encontrados extensos segmentos de introns, como podem ser visto nas linhas 0009 do resultado 1 (Figura 115); Linhas 0000, 0002, 0012, 0017 do resultado 2 (Figura 116); e linhas 0000, 0001, 0017, 0018, 0019, 0020, 0022, 0023 e 0024 do resultado 3 (Figura 117).

Novamente verificou-se a questão das referências de saída repetidas. No resultado 1 (Figura 115) observam-se diversos temporizadores ligados às linhas que vão de 0000 até 0006. Contudo apenas o último temporizador (linhas 0007 e 0008) é considerado para a lógica final. Também é observada repetição nas saídas Y0002 e Y0003.

No resultado 2 (Figura 116), no início na linha 0000 observa-se uma descontinuidade na linha. Logo em seguida é visto um contato de pulso que é representado em Instruciton List por LDP. Essa quebra ocorreu pela existência de um comando ANDP ao invés de um comando LDP. Como o comando ANDP inicia o programa não existe contato anterior, dessa forma a operação ANDP não possui um resultado inicial para iniciar sua lógica fazendo com que o segmento da linha 0000 até o final do contato NF V001 nunca entre em funcionamento, pois o pulso nunca é provocado. O restante da linha 0000 é aproveitado pela presença das linhas 0001 e 0002 que fornecem sua lógica inicial. Nesse resultado não foi verificado repetição de temporizadores.

Observando a instruction list que resulta a Figura 116 observa-se que não existe o contato auxiliar C002 presente na Ladder. Durante a conversão da Instruction List para Ladder verificou-se a necessidade de utilizar esse contato auxiliar para permitir uma melhor visualização do processo, pois a Ladder resultante sem o contato auxiliar C002 seria de difícil visualização pelo seu tamanho.

O resultado 3 (Figura 117) apresenta as mesmas características dos resultados 1 e 2: presença de introns e repetição de referências de saída.

e) tabelas de coletas de dados das simulações e gráficos de desempenho.

A seguir são apresentados os resultados de 1500 simulações realizadas para o estudo de caso do sistema de partida estrela/triângulo de motores. Em todas essas simulações foi utilizado elitismo, seleção por torneio e taxa de reprodução de 10%.

A Tabela 11 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 80%

- Mutação: 10%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 11 – Simulações com Cross-Over 80% e Mutação 10% – Partida estrela/triângulo de motor.

Maximo de gerações	Nº de simulações	N de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	0	150	0	0,00	0,00	-
50	100	0	250	0	0,00	0,00	-
100	80	0	330	0	0,00	0,00	-
150	50	0	380	0	0,00	0,00	-
200	30	0	410	0	0,00	0,00	-
300	30	0	440	0	0,00	0,00	-
400	30	0	470	0	0,00	0,00	-
500	30	0	500	0	0,00	0,00	-
Total	500	0					

A Tabela 12 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 45%

- Mutação: 45%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 12 – Simulações com Cross-Over 45% e Mutação 45% – Partida estrela/triângulo de motor.

Maximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	0	150	0	0,0000%	0,0000%	
50	100	0	250	0	0,0000%	0,0000%	
100	80	0	330	0	0,0000%	0,0000%	
150	50	0	380	0	0,0000%	0,0000%	
200	30	0	410	0	0,0000%	0,0000%	
300	30	0	440	0	0,0000%	0,0000%	
400	30	4	470	4	13,3333%	0,8511%	539
500	30	3	500	7	10,0000%	1,4000%	327
Total	500	7					

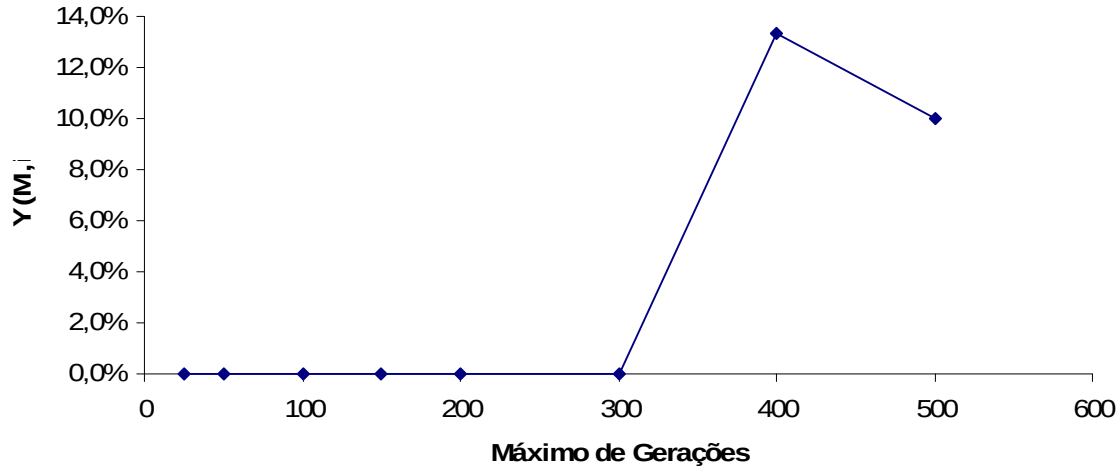


Figura 118 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Partida estrela/triângulo de motor.

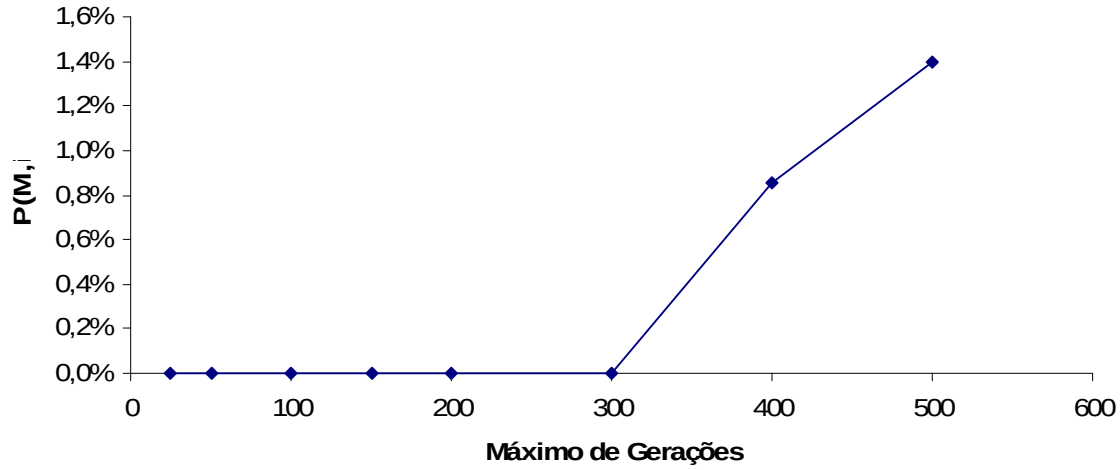


Figura 119 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 45% e mutação 45% – Partida estrela/triângulo de motor.

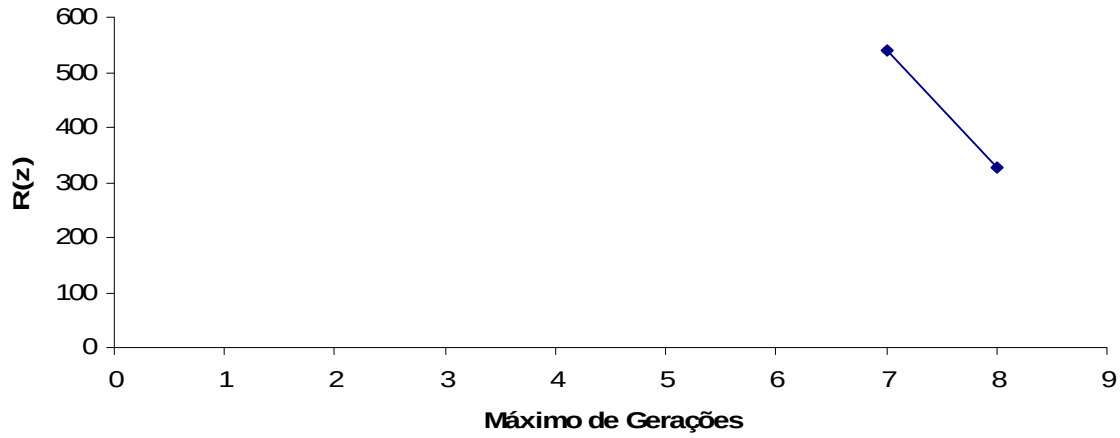


Figura 120 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 45% e mutação 45% – Partida estrela/triângulo de motor.

A Tabela 13 representa os resultados de simulações com os seguintes parâmetros:

- Cross-over: 10%

- Mutação: 80%

- Reprodução: 10%

- Elitismo

- Seleção por torneio

- População de 100 indivíduos

Tabela 13 – Simulações com Cross-Over 10% e Mutação 80% – Partida estrela/triângulo de motor.

Maximo de gerações	Nº de simulações	Nº de sucessos	Nº simulações acumulado	Nº de sucesso acumulado	Y(M,i)	P(M,i)	R(z)
25	150	0	150	0	0,0000%	0,0000%	
50	100	0	250	0	0,0000%	0,0000%	
100	80	1	330	1	1,2500%	0,3030%	1517
150	50	0	380	1	0,0000%	0,2632%	1748
200	30	2	410	3	6,6667%	0,7317%	627
300	30	2	440	5	6,6667%	1,1364%	403
400	30	6	470	11	20,0000%	2,3404%	194
500	30	5	500	16	16,6667%	3,2000%	142
Total	500	16					

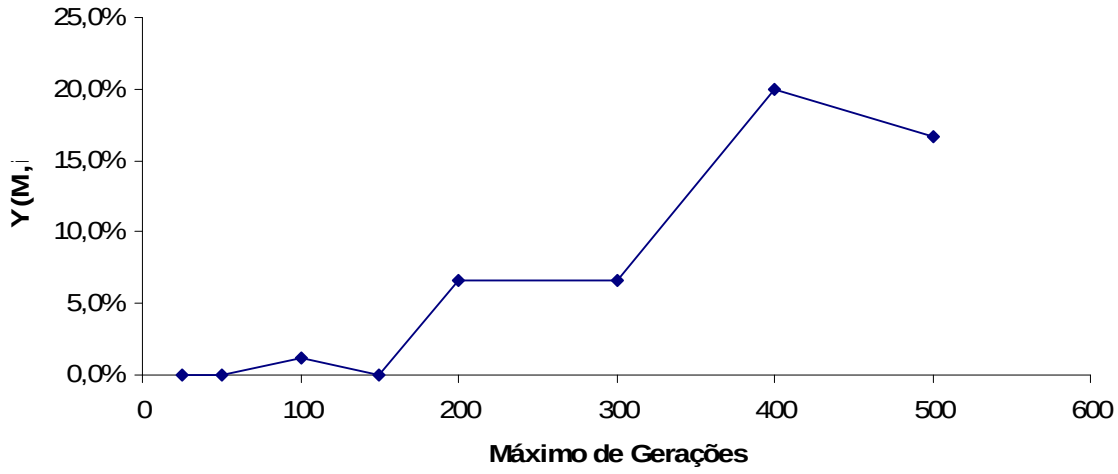


Figura 121 – Probabilidade de sucesso Y(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Partida estrela/triângulo de motor.

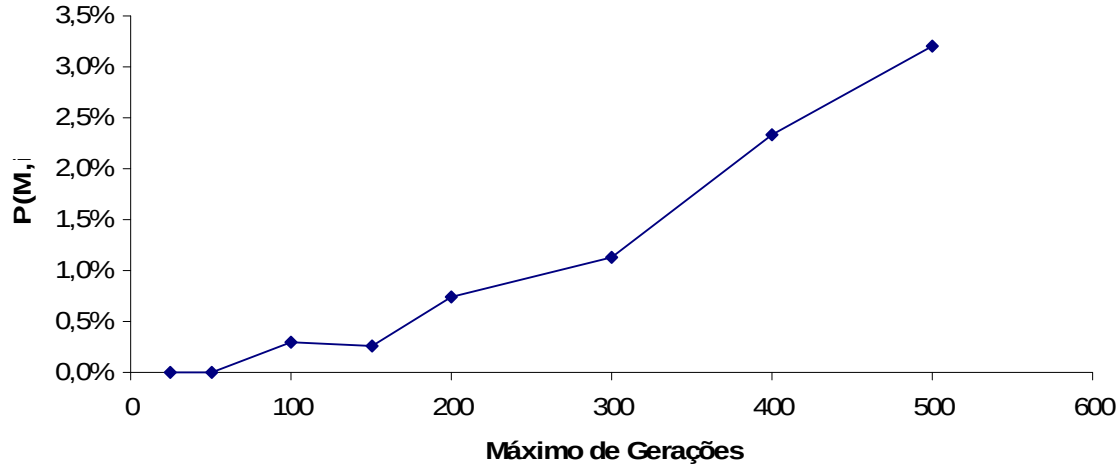


Figura 122 – Probabilidade acumulada de sucesso P(M,i) por número máximo de gerações para cross-over 10% e mutação 80% – Partida estrela/triângulo de motor.

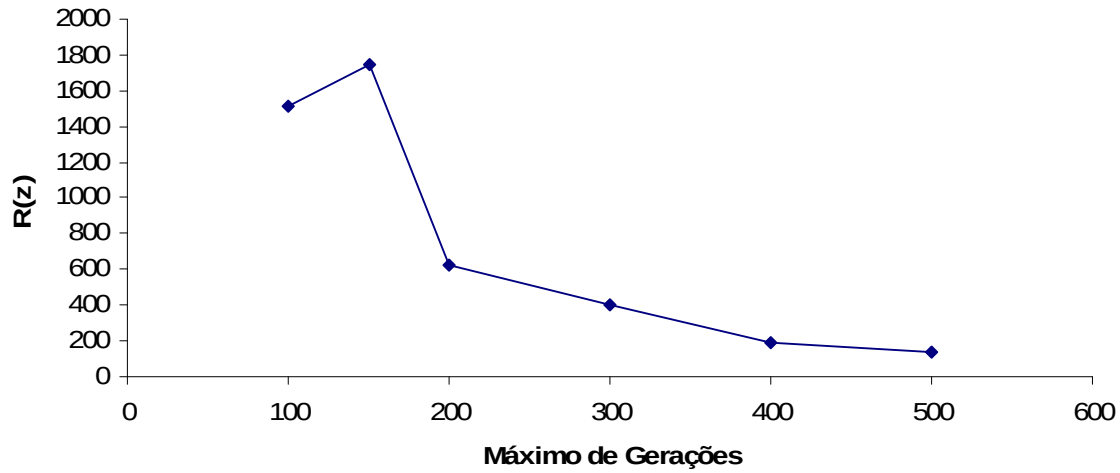


Figura 123 – Número de simulações necessárias para se obter um sucesso R(z) por número máximo de gerações para cross-over 10% e mutação 80% – Partida estrela/triângulo de motor.

f) as discussões acerca dos resultados obtidos e do comportamento do programa.

Esse estudo de caso apresentou resultados semelhantes ao estudo de caso III. Pela Tabela 11 observou-se que das 500 simulações, nenhuma obteve sucesso. A Tabela 12 mostra que até o limite de 300 gerações nenhuma simulação obteve sucesso. O algoritmo começa a conseguir solucionar o problema a partir da configuração de 45% de cross-over, 45% de mutação e limite de gerações igual a 400. Esses resultados reforçam a idéia de que para a dada complexidade do problema são necessárias simulações mais longas.

Apesar da Tabela 13 demonstrar poucas soluções, seus resultados são melhores. No limite máximo de 100 gerações uma solução já pode ser encontrada e esse número cresce gradativamente até o limite de 500 gerações. Percebe-se uma melhora sensível no índice R(z) (Tabela 13) que passa de 1517 simulações para 141 nas últimas simulações.

Ao todo foram encontradas 23 soluções para o problema. As simulações novamente demonstraram a importância do operador genético de mutação. Os resultados das simulações com parâmetros de 45% de cross-over, 45% de mutação e limite de gerações 400 foram próximos aos resultados das simulações com 10% de cross-over e 80% de mutação. Isso demonstra que para problemas com complexidade maior, apesar dos parâmetros de simulação poderem mudar a capacidade de se encontrar soluções, o limite de gerações também é importante. Para todos os parâmetros utilizados nas Tabela 11, Tabela 12 e Tabela 13, em nenhum caso foram encontrados soluções em simulações que utilizaram limite máximo de 25 e 50 gerações. Apesar de poucos dados para plotagem, os gráficos da Figura 118 até Figura 123, apresentaram comportamento padrão. Uma exceção foi encontrada no comportamento inicial da Figura 123. O gráfico de R(z) normalmente apresenta um decaimento exponencial em todo o seu comprimento. Observa-se que inicialmente ele sobe para depois apresentar um comportamento normal. Isso é devido ao conjunto de simulações com limite de 150 gerações, que no caso da presente coleta de dados não produziu nenhum resultado. Com o aumento no número de tentativas, e a permanência do mesmo número de sucessos a probabilidade se reduz. Esse comportamento anômalo ocorreu pelo fato da eficiência do algoritmo não ter se estabilizado ainda até a geração 150. Antes disso a ocorrência de sucesso ainda tinha uma forte ação do acaso, e a obtenção de sucesso ainda era improvável mesmo com a obtenção de uma solução no grupo de testes com limite de 100 gerações.

O programa demonstrou menor eficiência na solução desse problema para o quadro de simulações que foram feitas, porém um número satisfatório de soluções foi encontrado.

7 Conclusões

A seguir, as conclusões finais do trabalho são apresentadas a partir da sobreposição dos gráficos previamente apresentados e são comparados os desempenhos entre os estudos de casos, comparados os desempenhos entre as configurações em cada caso e comparações de R(z) entre as configurações de parâmetros.

7.1 Comparações de desempenho entre os estudos de caso

Na Figura 124, observa-se as porcentagens de sucesso de cada estudo de caso, para os parâmetros com cross-over 80%, mutação 10% e reprodução 10% (configuração 1). Como apresentado anteriormente, com esses parâmetros os estudos de caso III e IV não apresentaram nenhum sucesso. Observa-se uma crescente porcentagem de sucesso do caso II entre os limites de geração 25 e 100, e em seguida seu desempenho se estabiliza entre 70% e 73%. O caso I apresentou desempenho crescente ao longo e todo o intervalo.

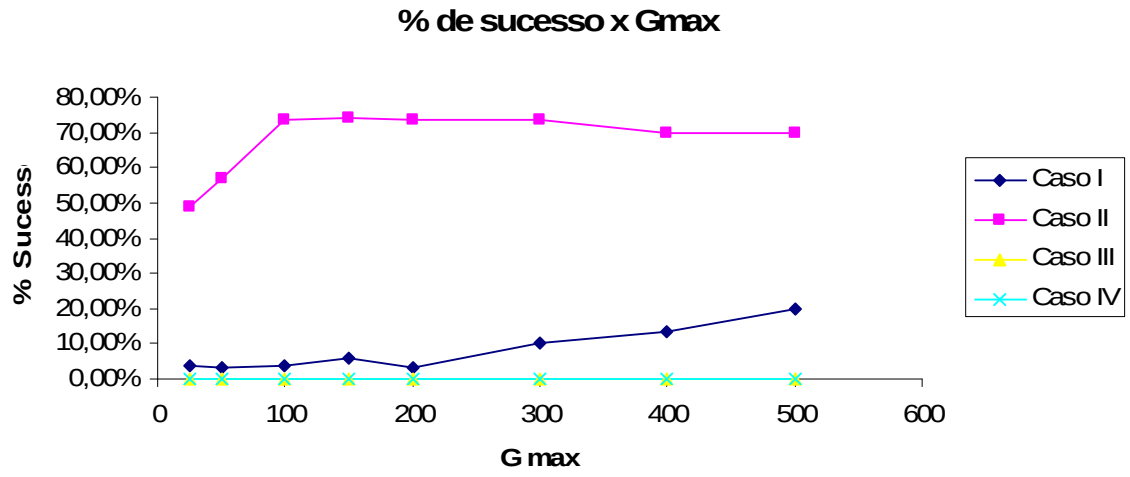


Figura 124 – Porcentagem de sucesso x Gmax entre os casos para configuração 1.

Na Figura 125, observa-se as porcentagens de sucesso de cada estudo de caso para os parâmetros com cross-over 45%, mutação 45% e reprodução 10% (configuração 2). Com o aumento do nível de mutação observa-se uma melhora no desempenho geral para todos os casos. Os casos III e IV que até então não haviam produzido nenhuma solução passaram a obter um sucesso crescente no final de suas respectivas curvas. O caso I também apresentou melhora de desempenho, e o caso II que havia se estabilizado em 70% na geração 100, passou a atingir um nível de sucesso de praticamente 100% a partir da geração 200.

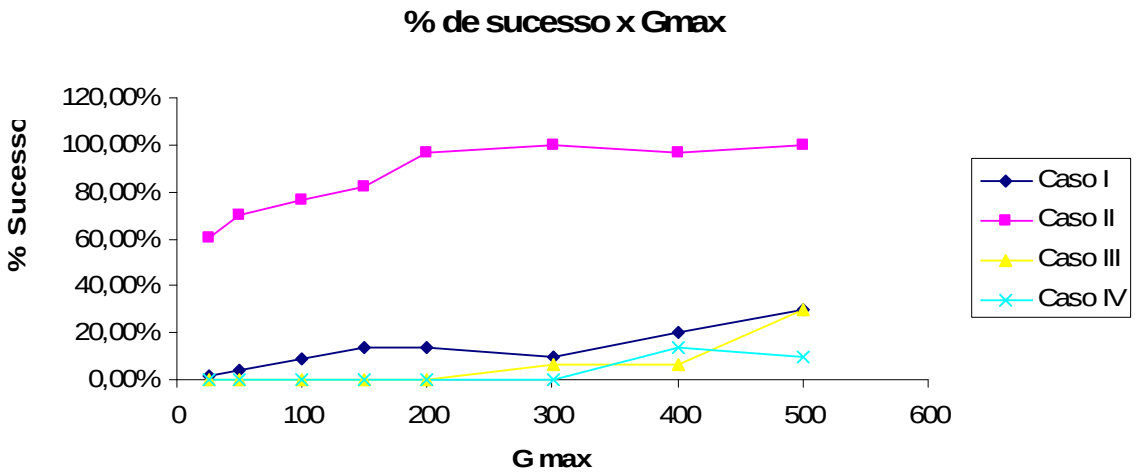


Figura 125 – Porcentagem de sucesso x Gmax entre os casos para configuração 2.

Na Figura 126, observa-se as porcentagens de sucesso de cada estudo de caso para os parâmetros com cross-over 10%, mutação 80% e reprodução 10% (configuração 3). Novamente houve melhora de desempenho para todos os casos. Os estudos I, III e IV apresentaram melhora, sendo que o estudo III que não havia produzido soluções na configuração 1, chegou a ultrapassar o desempenho do estudo de caso I. O estudo de caso II passou a atingir um nível de sucesso de 100% com menor tempo, a partir da geração 100.

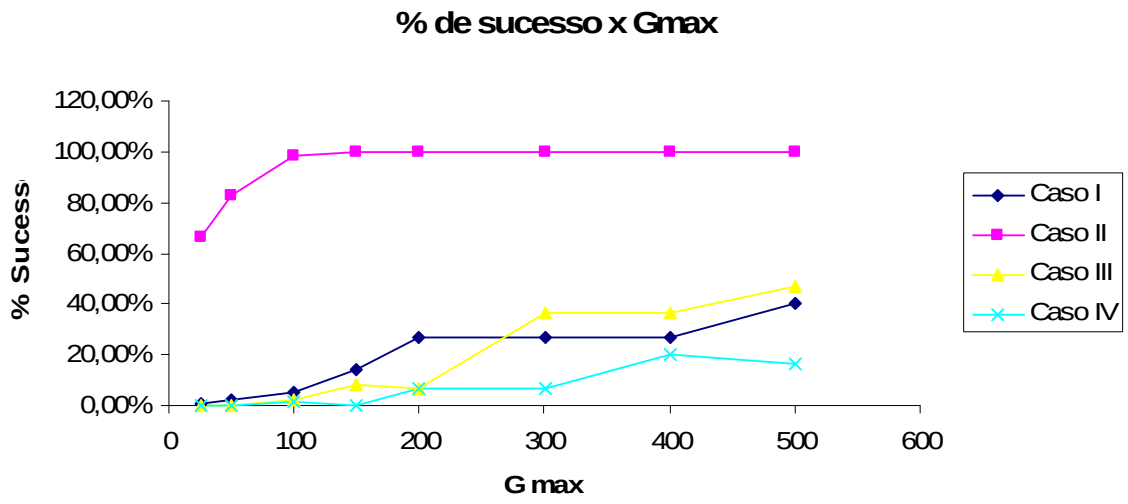


Figura 126 – Porcentagem de sucesso x Gmax entre os casos para configuração 3.

A partir dos resultados apresentados, é preciso saber que para as configurações 1 e 2, em que os casos III e IV apresentaram desempenho muito pequeno ou nulo que durante as simulações que as soluções evoluíram e chegaram a atingir níveis elevados de 70% a 90%, porém não atingiram o 100%. Isso é devido ao uso elevado do cross-over, que realiza uma busca extensiva. Quando a solução está próxima de 100%, os indivíduos que possuem nível de aptidão máximo estão na vizinhança, portanto, é necessária uma taxa maior de busca intensiva (mutação) para que elas sejam encontradas.

7.2 Comparações de probabilidade de sucesso entre as configurações de parâmetros

É apresentada a seguir uma comparação de desempenho, para cada caso, entre as três configurações de parâmetros pesquisados.

Na Figura 127 observa-se que no caso I, a configuração 3, com maior taxa de mutação, apresentou melhor desempenho que as outras configurações após 150 gerações. Observa-se também que para simulações mais curtas, no início do gráfico, a configuração 1 (com maior taxa de cross-over) apresentou melhor desempenho. Esse comportamento ocorre pelo fato do cross-over representar uma busca extensiva e a mutação uma busca intensiva. Dessa forma o cross-over é capaz de produzir avanços maiores de evolução em

menor tempo, pois procura pontos mais distantes no espaço de soluções e o operador mutação realiza buscas na vizinhança dos melhores indivíduos da população.

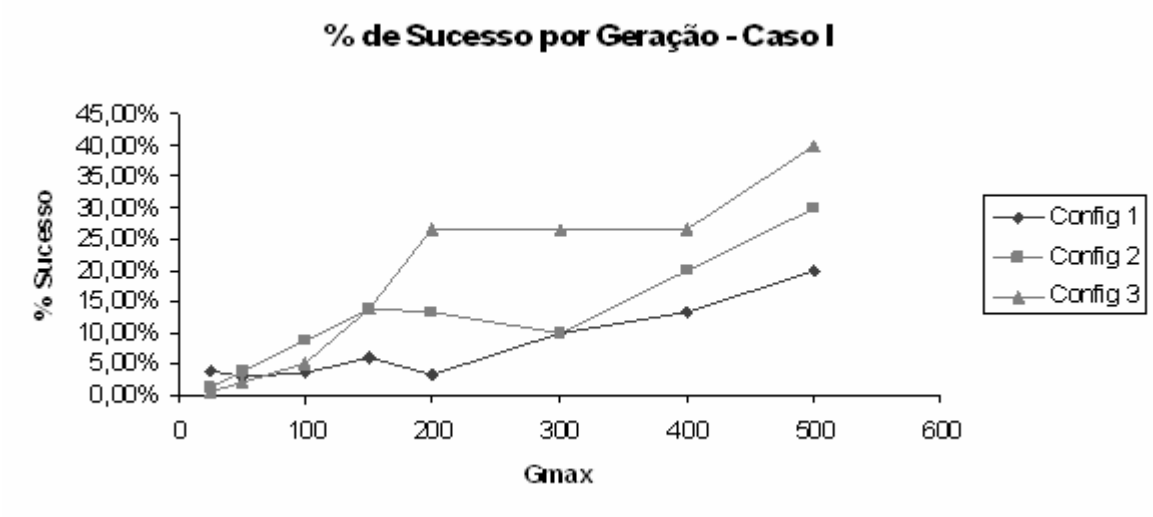


Figura 127 – Comparação entre configurações para porcentagem de sucesso por geração para caso I.

A Figura 128, a seguir, demonstra novamente o melhor desempenho da configuração 3. É possível observar uma melhora significante da configuração 1 para a configuração 3 no estudo de caso II. A Figura 129 e a Figura 130 novamente demonstra o melhor desempenho da configuração com maior mutação para os estudos de caso III e IV.

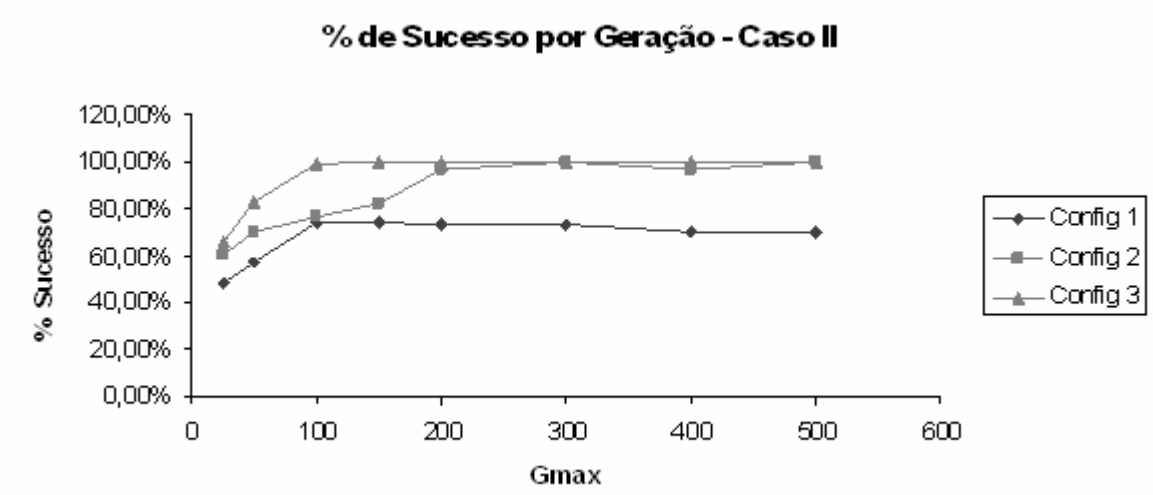


Figura 128 – Comparação entre configurações para porcentagem de sucesso por geração para caso II.

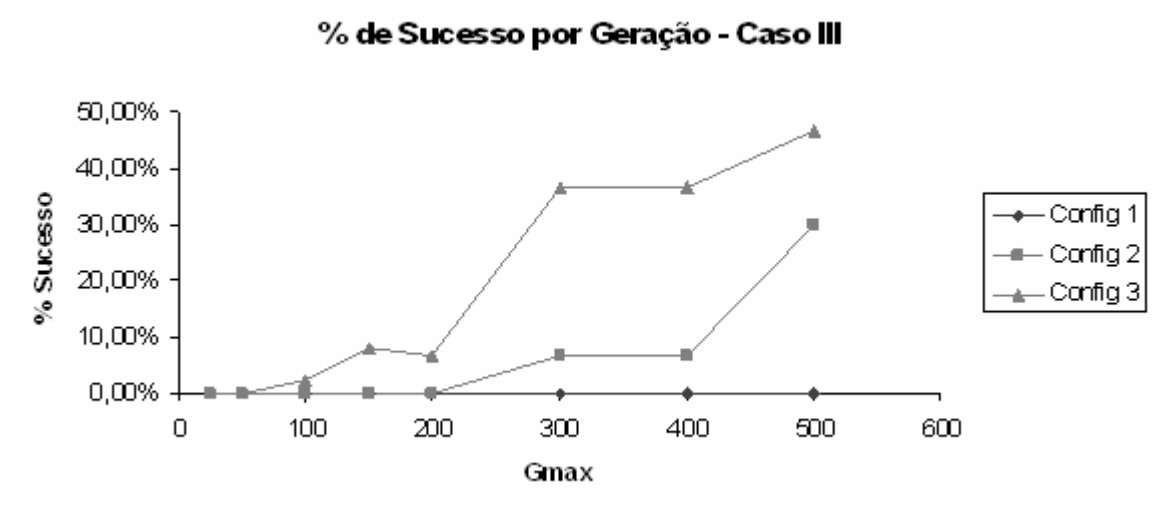


Figura 129 – Comparação entre configurações para porcentagem de sucesso por geração para caso III.

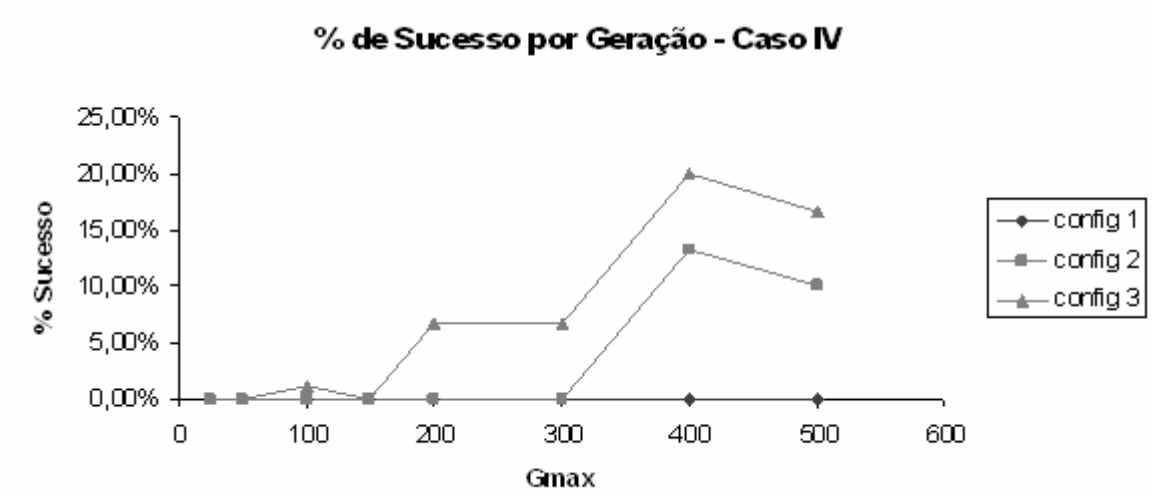


Figura 130 – Comparação entre configurações para porcentagem de sucesso por geração para caso IV.

7.3 Comparações de R(z) entre as configurações de parâmetros

A seguir são comparados os índices R(z), em cada caso, entre as três configurações de parâmetros apresentados.

Observa-se nas Figura 132, Figura 133, Figura 134 que a configuração 3 apresentou melhor desempenho ao longo de toda a curva, ou seja, para a configuração com maior mutação foram necessárias menos simulações para obtenção de mais respostas. Observa-se

na Figura 131 uma diferença nesse comportamento. Como foi visto na Figura 127, para simulações mais curtas o operador cross-over possui melhor eficiência. O caso I, apresentou melhor desempenho da configuração 1 para simulações mais curtas até a marca de 200 gerações, como pode ser observado na Figura 131.

Observa-se que em programas com lógica mais entrelaçada que pequenas mudanças podem causar significativas mudanças no seu nível de aptidão. Nesses casos o uso da mutação é mais vantajoso, pois produz mudanças graduais e evita que uma lógica mais entrelaçada seja quebrada. Já em programas mais modulares, com partes mais independentes, é possível aplicar taxas maiores de cross-over pelo fato de blocos maiores de código poderem ser transportados de um indivíduo para o outro com um risco menor de produzir um efeito destrutivo em sua lógica.

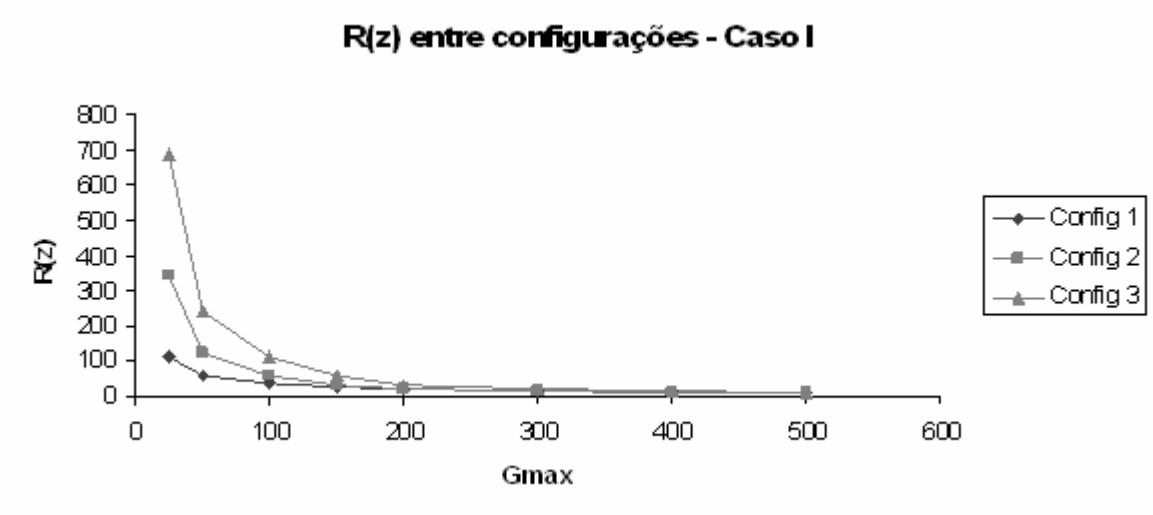


Figura 131 – Comparação de índices R(z) entre configurações para caso I.

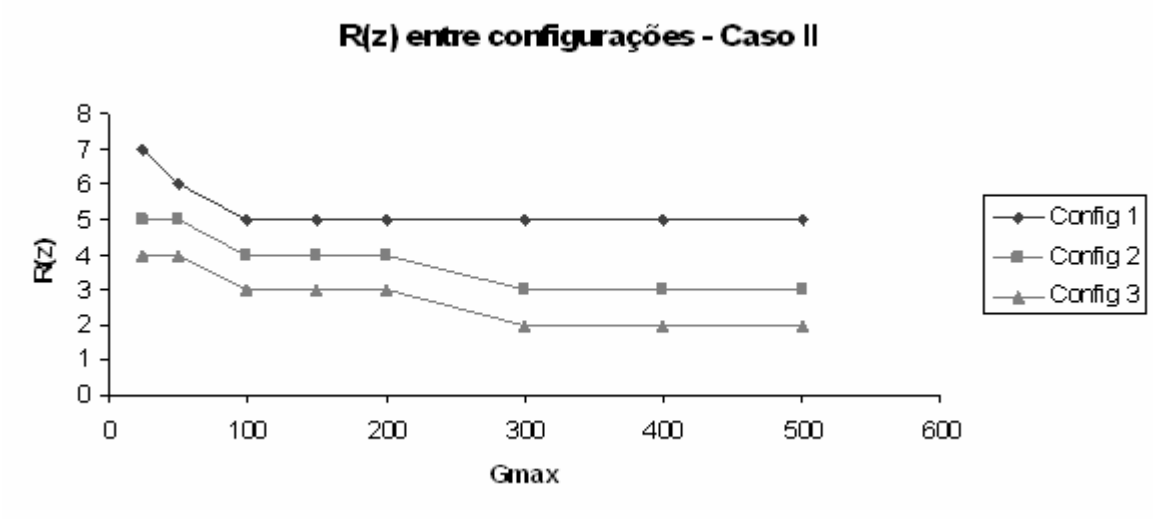


Figura 132 – Comparação de índices R(z) entre configurações para caso II.

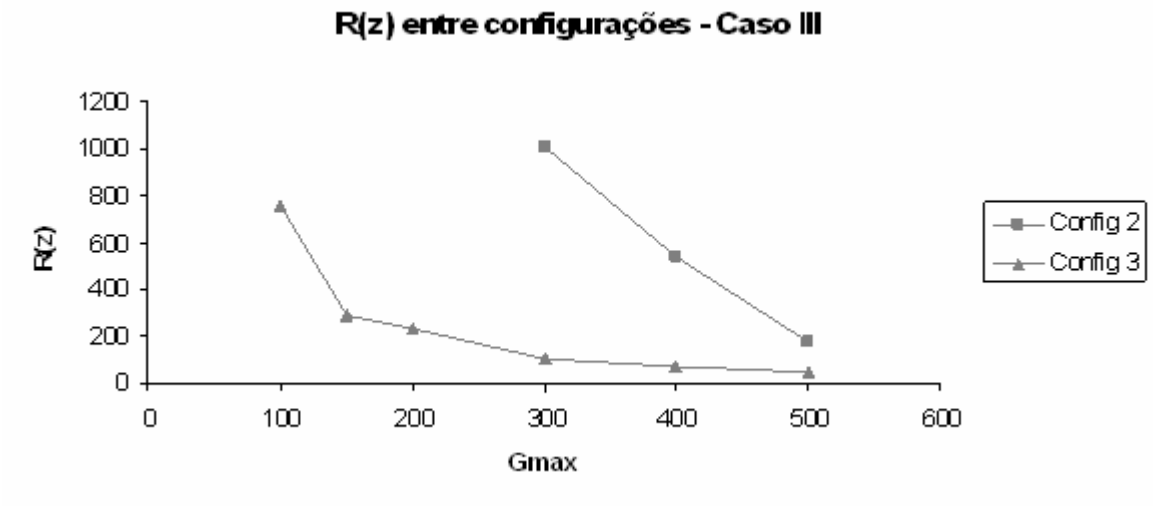


Figura 133 – Comparação de índices R(z) entre configurações para caso III.

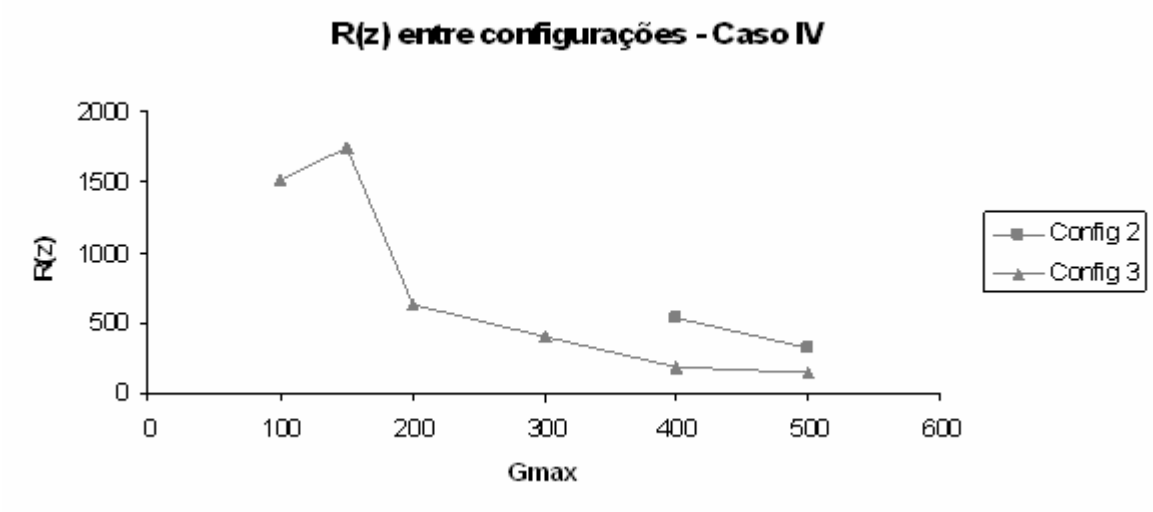


Figura 134 – Comparação de índices R(z) entre configurações para caso IV.

7.4 Considerações Finais

A motivação original da pesquisa foi a criação de um programa capaz de gerar programas para centrais de automação (CLP) de forma automática para facilitar o trabalho de programadores. Todavia, testar o sistema de programação genética desenvolvido para esse fim tornou-se, mais do que um meio para a obtenção de um resultado, mas um objetivo em si. As dificuldades encontradas foram principalmente com relação ao desenvolvimento de um simulador de programas que fosse capaz de ignorar erros de sintaxe aproveitando os segmentos de código com valor lógico. Outro problema encontrado foi a necessidade de fazer um número muito elevado de simulações para que alguns gráficos pudessem ser construídos de forma a facilitar a visão global do processo. Todo o sistema é estocástico, porém com a repetição do processo foi possível determinar a probabilidade com que o sucesso ocorre de forma dependente dos parâmetros iniciais das simulações.

A partir das mais de 6000 simulações feitas é possível afirmar que simulações com 10% de cross-over, 80% de mutação, 10% de reprodução, seleção por torneio e população de 100 indivíduos são mais favoráveis na programação genética aplicada à criação de programas de CLP. Conclui-se também que é melhor deixar com que o algoritmo realize buscas acima de 400 gerações para que um número maior de soluções seja encontrado. Esse número revelou ser pequeno, pois é comum em programação genética a execução de milhares de gerações. Dessa forma, se o quadro visualizado pela pesquisa (de 25 a 500

gerações) já demonstra resultados satisfatórios e grande número de soluções, espera-se que para simulações mais longas ainda mais soluções sejam encontradas.

Observou-se também que a complexidade do problema de automação afeta o desempenho do algoritmo tornando necessário o aumento no número de gerações e das taxas de mutação. Outro fator importante a ser observado é a capacidade de simulação fornecido pelos cenários. Quanto maior o número de possibilidades de simulação do programa, maior é a seletividade do algoritmo. Com poucas situações de teste um número maior de soluções na simulação possui a mesma numeração de fitness apesar de cada programa possuir uma estrutura de comandos diferente da outra. Isso reduz a capacidade de selecionar os melhores e a busca se torna mais cega pois indivíduos diferentes são tratados como iguais.

A partir dos programas apresentados como resultado observou-se grande diversidade de estruturas nos programas Ladder construídos. Observou-se também que antes da utilização final dos resultados para um sistema prático de automação é necessária uma análise preliminar dos programas gerados para a retirada dos introns que são facilmente visualizados através de redundâncias e códigos sem lógica conclusiva.

Como o objetivo era a criação de um sistema que gerasse programas para CLP, pode-se afirmar que os resultados são promissores, pois o programa criado é capaz de encontrar conjuntos de códigos Instruction List e consequentemente seu correspondente em Ladder para a solução de diversos problemas. Por outro lado, os resultados obtidos com o modelo criado mostram que melhorias podem ser feitas para que a eficiência do processo melhore.

Apesar do foco da dissertação ser a análise da heurística da programação genética, o simulador criado para viabilizar o sistema ocupou considerável tempo na pesquisa. Considera-se que o ponto de maior dificuldade seja justamente este sistema de simulação que exige informações de diversas matrizes booleanas que representam as linhas do tempo que compõem os cenários e os programas Instruction List; e que gera diversas outras linhas do tempo para o cálculo final do fitness. Considerando que todo esse processo é executado para cada indivíduo de uma população de programas, em cada geração, repetidas vezes, esse sistema se torna um método que necessita de computadores com processadores rápidos e considerável memória disponível para viabilidade da técnica.

Os estudos de caso se concentraram em um sistema de iluminação e em três sistemas diferentes para controle de motores. Os exemplos apresentados não possuem

grande diversidade quanto a seu campo de aplicação (acionamento de motores), mas sim na diferença entre seus cenários. O sistema de simulação e programação genética não consegue visualizar para que tipo de aplicação a busca está sendo executada. Todo o processo visualiza apenas as linhas do tempo, e simula e seleciona os indivíduos unicamente a partir destas.

7.3 Perspectivas

Algumas questões que não foram abordadas durante a pesquisa devido a extensão que elas exigem podem ser usadas para a realização de futuros trabalhos. Essas questões são citadas a seguir:

- Determinação do impacto na eficiência do algoritmo quando são utilizados diferentes tamanhos de população, ou quantidade variável de indivíduos durante as simulações (abordagem multi-objetivo).
- A abordagem realizada nesse trabalho teve como um dos objetivos revelar a importância do número de gerações na simulação, e tornou fixo os tipos de operadores genéticos, limitando o cross-over como um sistema de troca de gens em dois pontos, e o tipo de mutação em um sistema que altera apenas um comando aleatoriamente quando ele é aplicado. Em pesquisas futuras, o número de gerações poderia se tornar uma variável de controle, fixando ela em um número muito elevado, para que diferentes tipos de cross-over e mutação fossem testados.
- Após a indicação de uma solução para o problema seria relevante a criação de um filtro automático que retirasse as redundâncias e os introns da resposta e apresentasse diretamente a Ladder final para ser implementada no CLP, além da criação de um sistema que impeça a geração de bobinas com referências repetidas.
- O sistema de programação genética criado pode ser aplicado em diversos tipos de casos. O sistema pode ser utilizado para abordar um número maior de casos com objetivo de ampliar a confiança na técnica e torná-la cada vez mais viável para a real utilização em sistemas comerciais.
- Acredita-se que seja possível a utilização de outras heurísticas na solução dos problemas de automação. A aplicação de outros métodos seria relevante para a comparação da eficiência de uma em relação à outra.

Referências

Angeline, P. J., “Genetic programming and emergent intelligence”. In Kinnear, Jr., K. E., editor, Advances in Genetic programming, chapter 4, pages 75-98. MIT Press, Cambridge, MA. (1994)

Banzhaf, W., Nordin, P., Keller R. E., Francone F. D. “Genetic Programming, On the automatic Evolution of Computer programs and its applications”. Morgan Kauffmann Publishers Inc., San Francisco, Califórnia. (1998)

Chen X., Gui W., Wang Y., Cen L., "Multi-step optimal control of complexprocess: a genetic programming strategy and its application",Engineering Applications of Artificial Intelligence 17, 491–500. (2004)

Costa, S. F., “Introdução Ilustrada à Estatística”. 4º Edição, São Paulo, Editora Harbra, 398 p. (2005)

Cross, S., Estrada, R., “DART: an example of accelerated evolutionary development”. Advanced Research Projects Agency, Grenoble, France. (1994)

Carneiro, M. L., “Programação Genética Aplicada à Linguagem Ladder”. Induscon – VIII Conferência Internacional de Aplicações Industriais, Poços de Caldas, Minas Gerais, Brasil, (2008).

Dadone P., VanLandingham H. F., "PLC Implementation of a Genetic Algorithm for Controller Optimization",Proc. 3rd International Conference of Computational Intelligence and Neuroscience (JCIS’98/CI&N’98), 2, 91-94, Raleigh, NC, October (1998)

Darwin, C., “On the Origin of Species by Means of Natural Selection”. John Murray. (1859)

Frey G., "Automatic Implementation of Petri Net Based Control Algorithms on PLC", Proceedings of the American Control Conference, Chicago, Illinois, June. (2000)

Frey G., Litz L.,"Formal methods in PLC programming", IEEE. (2000)

Galvan-Lopez, E., "Efficient Graph-based Genetic Programming Representation with Multiple Outputs". International Journal of Automation and Computing, Colchester, UK. (2008)

GE Fanuc Automation, "DURUS Controllers System Manual GFK-2470". GE Fanuc Automation North America, Inc. (2007)

Hagras, H., "Employing Computational Intelligence to Generate More Intelligent and Energy Efficient Living Spaces". International Journal of Automation and Computing, Colchester, UK. (2008)

Holland, J. H. “Adaptation in Natural and Artificial Systems”. University of Michigan Press. (1975), IEEE Mec & Rob conference, Neworland, L.A., USA. (2004)

Jafari, A., Safavi, M., Fadaei, A., "A Genetic Algorithm to Optimum Dynamic Performance of Industrial Robots in the Conceptual Design Phase".Proceedings of the 2007 IEEE 10th International Conference on Rehabilitation Robotics, June 12-15, Noordwijk, The Netherlands (2007)

Jargot, P., “Langages de programmation pour API. Norme IEC 1131-3”. Techniques de L’Ingénieur, traité Informatique industrielle, CJ International. (1999)

Johansson S., Öhman M., "Prototype implementation of the PLC standard IEC 1131-3", Department of Automatic Control, Lund Institute of Technology, December. (1995)

Koza, J. R. “Genetic Programming, On the programming of computers by means of natural selection”. The MIT Press, Cambridge, Massachussetts, London, England. (2000)

Lam, H. N., "Stochastic Modelling and Genetic Algorithmbased Optimal Control of Air Conditioning Systems". In Proceedings of the 3rd International Conference of the

International Building Performance Simulation Association, Adelaide, Australia, pp. 435–441. (1993)

Lam., H. N., "Intelligent Computer Control of Air Conditioning Systems Based on Genetic Algorithm and Classifier System". In Proceedings of the 4th International Conference of the International Building Performance Simulation Association on Building Simulation, Madison, Wisconsin, USA, pp. 151–157. (1995)

Lewis, R. W. “Programming Industrial Control System Using IEC 1131-3”. IEEE Control. (1995)

Liu J., Darabi H., "Ladder Logic Implementation of Ramadge-Wonham Supervisory Controller", Proceedings of the Sixth International Workshop on Discrete Event Systems, IEEE. (2002)

Luger, G., Stubblefield, W. A., “Artificial Intelligence, Structures and Strategies for Complex Problem Solving”. Third Edition, Addison Wesley Longman, Inc, Massachussets, USA. (1997)

Manovit, C., Aporntewan, C., Chongstitvatana P. “Synthesis of synchronous sequential logic circuits from partial input/output sequence”. In Proceedings of Int. Conference on Evolvable Systems (ICES'98), pages 98-105. (1998)

Marconi, M. De A., Lakatos, E. M., “Fundamentos de Metodologia Científica”. 6º ed, 4º reimpressão, editora Atlas, São Paulo. (2007)

Minas M., Frey G., "Visual PLC-Programming using Signal Interpreted Petri Nets", Proceedings of the American Control Conference, Anchorage, AK May 8-10. (2002)

Moraes, C. C. de, Castrucci, P. de L., “Engenharia de Automação Industrial”. 2º Ed., LTC, Rio de Janeiro. (2007)

Nguyen, H. H., Uraikul, V., Chan C. W., Toniwachwuthikul, P., "A comparison of automation techniques for optimization of compressor scheduling". Advances in Engineering Software 39 (2008) 178–188. (2008)

Nordin, P., Francone, F., Banzhaf, W., “Explicity defined introns and destructive crossover in genetic programming”. In Angeline, P. J. and Kinnear, Jr., K. E., editors, Advances in Genetic Programming 2, chapter 6, pages 111-134. MIT Press, Cambridge, MA. (1996)

Nordin, P., Francone, F., Banzhaf, W., “Explicity defined introns and destructive crossover in genetic programming”. In Rosca, J. P., editor, Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, pages 6-22, Tahoe City, CA. (1995)

Rausch M., Krogh B. H."Formal Verification of PLC Programs". Proceedings of the American Control Conference, Philadelphia, Pennsylvania, junho. (1998)

Rockwell Automation, "MicroLogix 1000 Programmable Controllers, User Manual". Allen-Bradley publication, Bulletin 1761 Series July. (1998)

Rockwell Automation, "Pico GFX-70 Controllers, User Manual". Allen-Bradley publication, Bulletin 1760 Series, April. (2004)

Romero, R., Mantovani, J. R. S. “Introdução a Metaheurísticas, Anais do 3º Congresso Temático de Dinâmica e Controle da SBMAC, UNESP, Campus de Ilha Solteira, Brasil”. (2004)

Rosca, J. P., “Analysis of complexity drift in genetic programming”. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., Riolo, R. L., editors, Genetic Proramming 1997: Proceedings of the Second Annual Conference, pages 286-294, Stanford University, CA. Morgan Kaufmann, San Francisco, CA. (1997)

Samuel, A., “Some studies in machine learning using the game of checkers”. In Feigenbaum, E., Feldman, J., editors, Computers and Thought. McGraw-Hill, New York. (1963)

Shaobo, L., Guanci, Y., Qingsheng, X., "Automatic Design Method of Dynamic Systems Based on Hungarian Algorithm and Genetic Programming". International Conference on Wireless Communications, Networking and Mobile Computing, IEEE, Guiyang, China. (2008)

Siemens, "LOGO!, Manual de instruções". Siemens Simatic, manual number A5E00380839-02, edition 5. (2006)

Siemens, "S7-200 Programmable Controller System Manual". Siemens Simatic, manual number 6ES7298-8FA24-8BH0, edition 8. (2005)

Soule, T., Foster, J. A., “Code size and depth flows in genetic programming”. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, Genetic Programming 1997: Proceedings of the Second Annual Conference, pages 313-320, Standford University, Stanford, CA. Morgan Kaufmann, San Francisco, CA. (1997)

Soule, T., Foster, J. A., Dickinson, J., “Code growth in genetic programming”. In Koza, J. R., Goldberg, D. E., Fogel, D. B., Riolo, R. L., editors, Genetic Programming 1996: Proceedings of the First Annual Conference, pages 215-223, Stanford University, CA. MIT Press, Cambridge, MA. (1996)

Sysala T., Dostal P., "Adaptive Controller Implementation into PLC library", IEEE International Symposium on Computer Aided Control System Design, Taipei, Taiwan, September 24. (2004)

Tackett, W. A., “Recombination, Selection, and the Genetic Construction of Computer Programs”. PhD thesis, University of Southern California, Department of Electrical Engineering Systems. (1994)

Teller, A., Veloso, M., “A controlled experiment: Evolution for learning difficult image classification”. In Seventh Portuguese Conference On Artificial Intelligence, volume 990 of Lecture notes in Computer Science, pages 165-176, Funchal, Madeira Island, Portugal. Springer-Verlag, Berlin, Germany. (1995)

Thorpe, C., Herbert, M., Kanade, T., Shafer, S., “Toward autonomous driving: the CMU Navlab. I. Perception”. IEEE Expert, Volume 6, Issue 4, Pages 31-42. (1991)

Uzam M., Jones A. H., "Discrete Event Control System Design Using Automation Petri Nets and their Ladder Diagram Implementation", The international Journal of Advanced Manufacturing Technology, n14:716-728. (1998)

WEG Automação S.A.”Manual de Instalação – TPW03 Controlador Programável”. Av. Pref. Waldemar Grubba, 3000, Jaraguá do Sul, Santa Catarina (SC), Brasil. (2008)

WEG Automação S.A.”Manual do Usuário – CLIC 02 Micro Controlador Programável”. Av. Pref. Waldemar Grubba, 3000, Jaraguá do Sul, Santa Catarina (SC), Brasil. (2006)

WEG Automação S.A.”Manual do Usuário – TP-02 Controlador Programável”. Av. Pref. Waldemar Grubba, 3000, Jaraguá do Sul, Santa Catarina (SC), Brasil. (2007)

ANEXO I – Fotos do Programa Desenvolvido

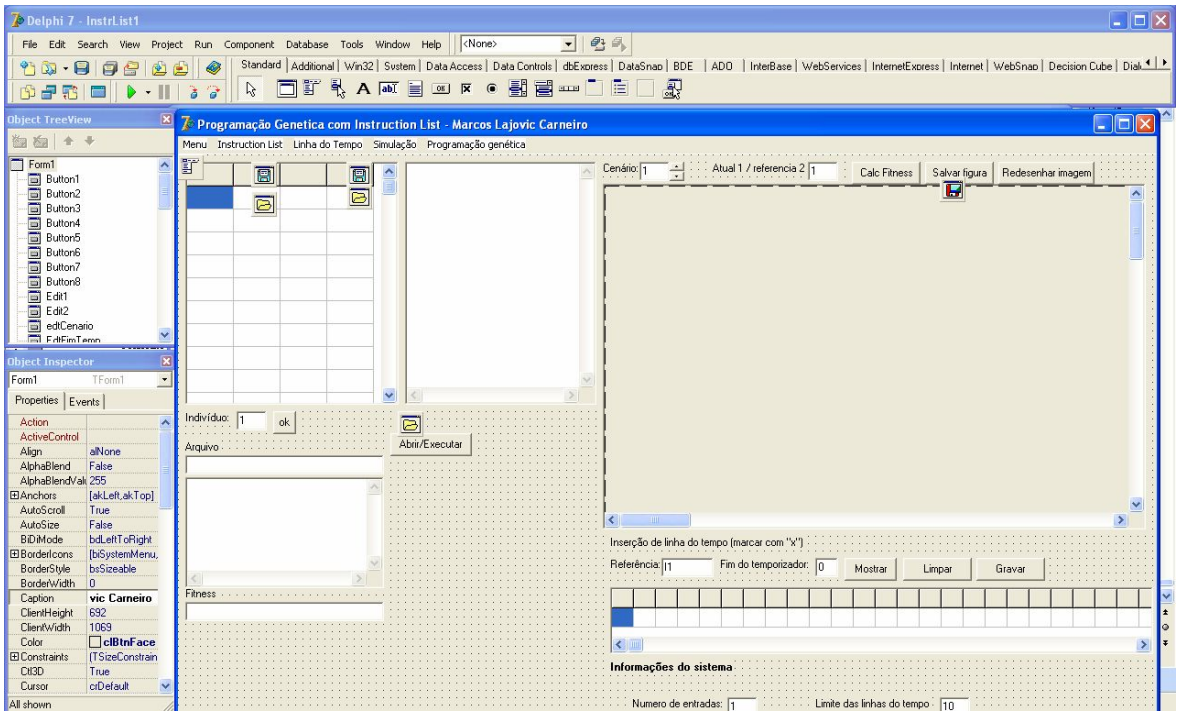


Figura 135 – Programa em desenvolvimento

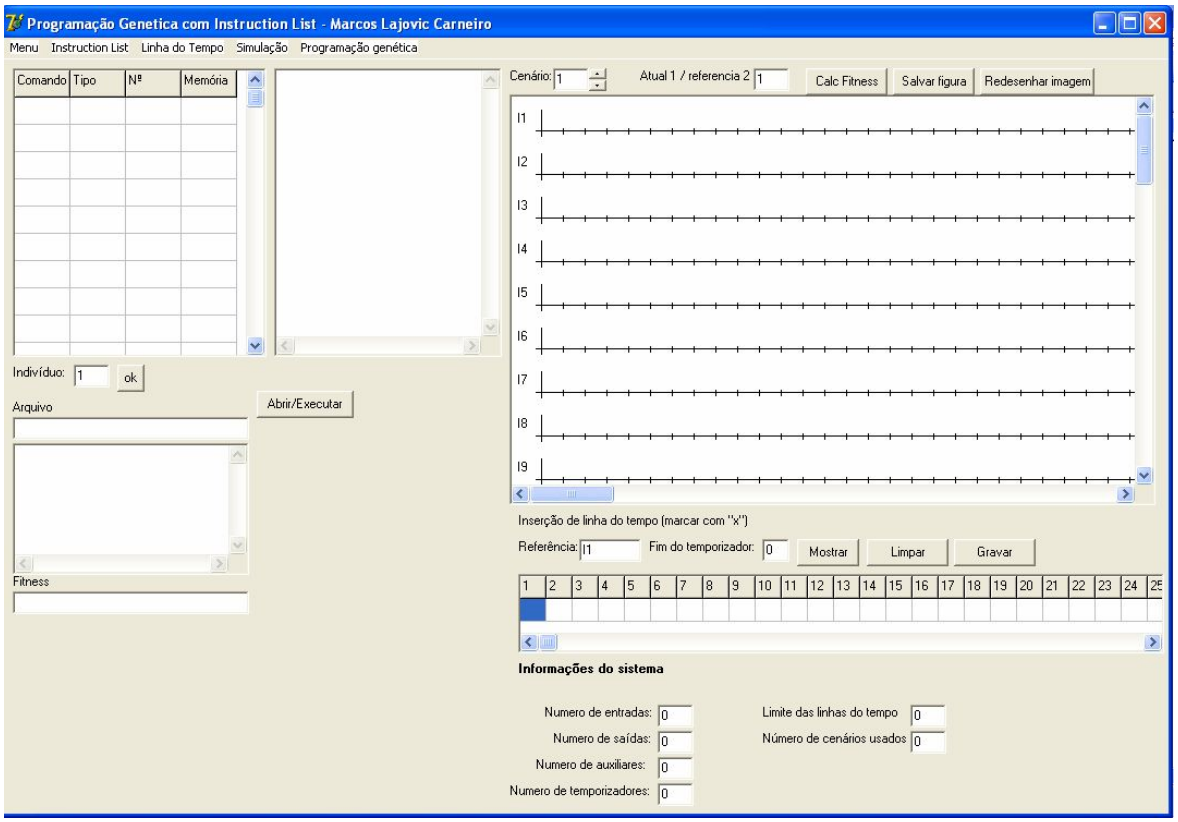


Figura 136 – Tela de Simulação

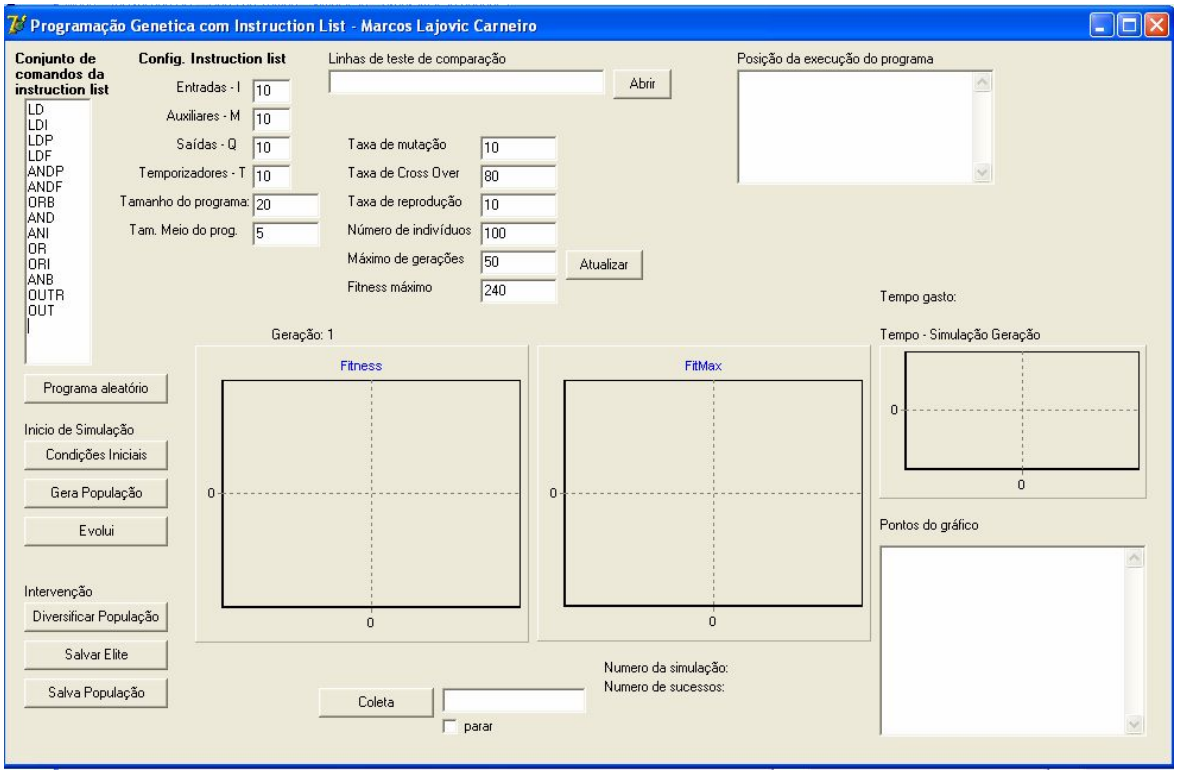


Figura 137 – Tela de Programação Genética

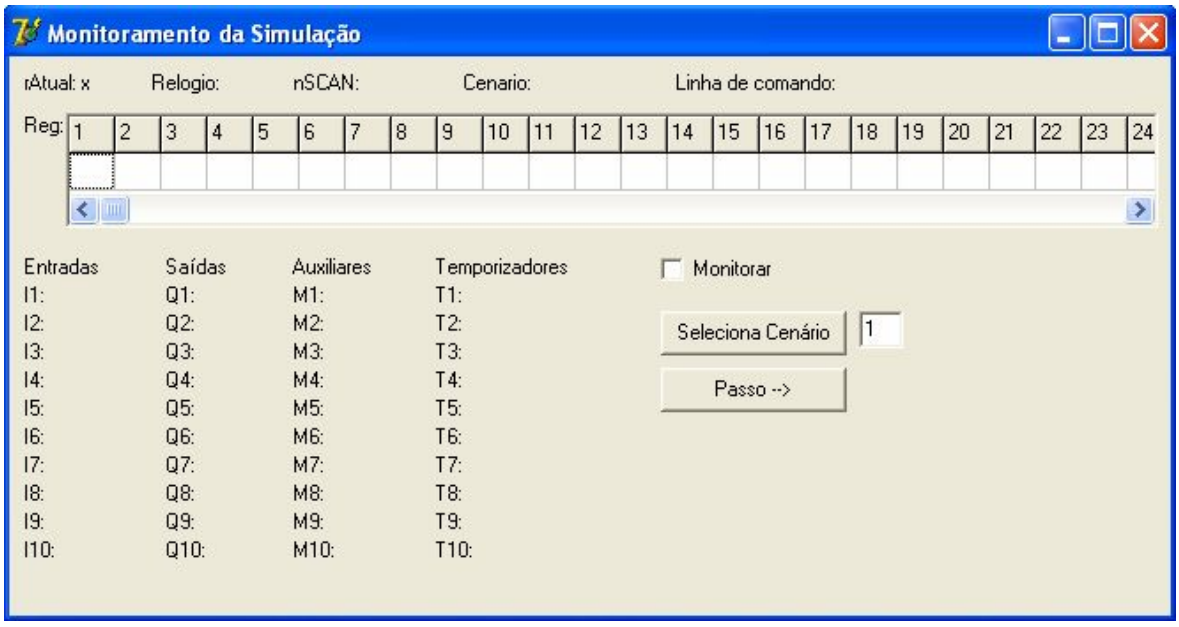


Figura 138 – Tela de Depuração da Simulação

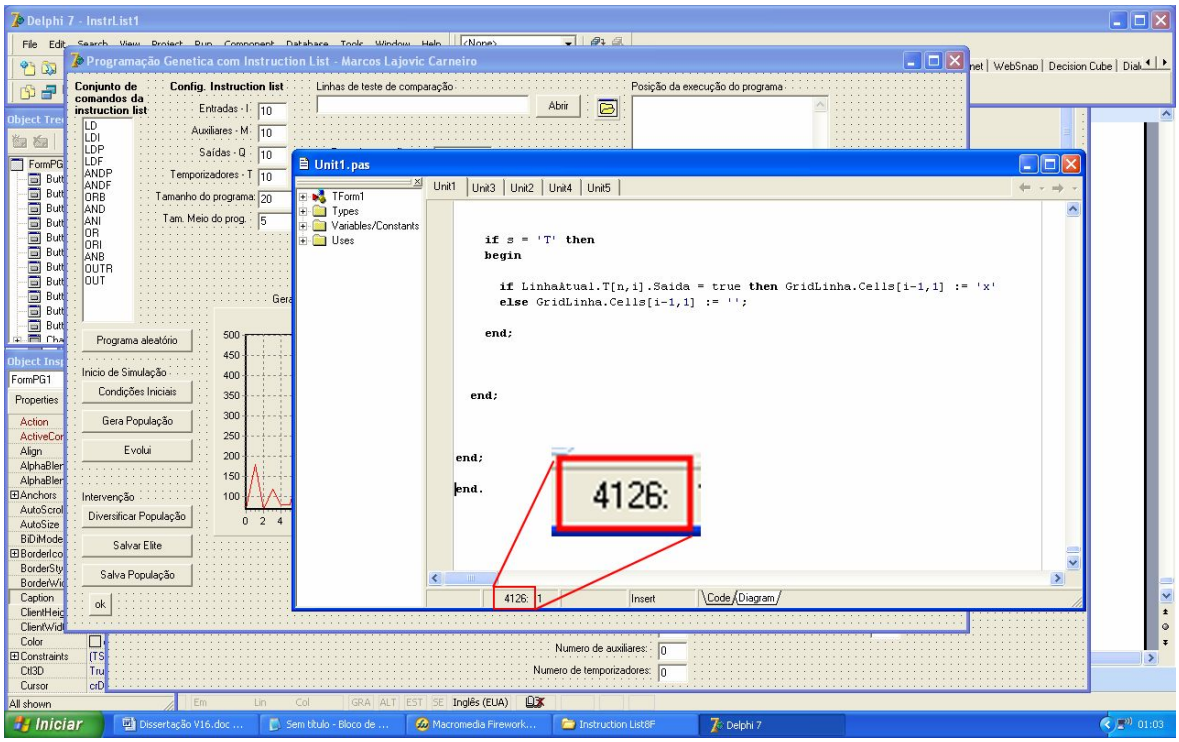


Figura 139 – Número de linhas de código da unidade de Simulação

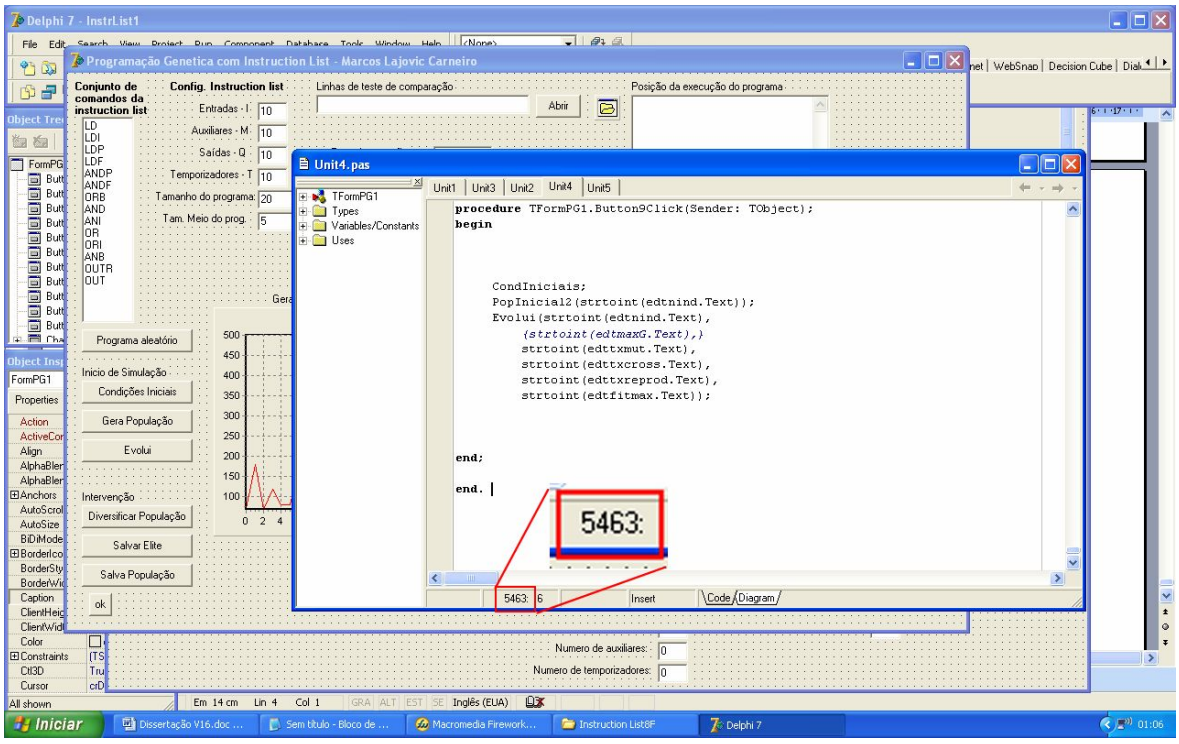


Figura 140 – Número de linhas de código da unidade de Programação Genética