

EDSON MARTINS LECHETA

**ALGORITMOS GENÉTICOS PARA PLANEJAMENTO
EM INTELIGÊNCIA ARTIFICIAL**

Dissertação apresentada como requisito
parcial à obtenção do grau de Mestre.
Curso de Pós-Graduação em Informática,
Setor de Ciências Exatas,
Universidade Federal do Paraná.
Orientador: Prof.Dr. Marcos Alexandre Castilho

CURITIBA

2004

EDSON MARTINS LECHETA

**ALGORITMOS GENÉTICOS PARA PLANEJAMENTO
EM INTELIGÊNCIA ARTIFICIAL**

Dissertação apresentada como requisito
parcial à obtenção do grau de Mestre.
Curso de Pós-Graduação em Informática,
Setor de Ciências Exatas,
Universidade Federal do Paraná.
Orientador: Prof.Dr. Marcos Alexandre Castilho

CURITIBA

2004

SUMÁRIO

LISTA DE FIGURAS	III
LISTA DE TABELAS	V
EPÍGRAFE	VII
AGRADECIMENTOS	VIII
RESUMO	IX
ABSTRACT	X
1. INTRODUÇÃO.....	1
2. PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL.....	5
2.1. Modelagem de Planejamento	10
2.1.1. Representação de Domínios.....	12
2.1.2. Representação de Problemas	16
2.2. Planejadores e Competições	17
3. ALGORITMOS GENÉTICOS	27
3.1. Histórico.....	27
3.2. Problemas de Busca	29
3.3. Características do AG.....	31
3.4. Funcionamento.....	32
3.5. Operadores Genéticos	35
3.6. Aplicações.....	37
4. ABORDAGEM GENÉTICA NO PLANEJAMENTO	40
4.1. Proposta de Modelagem.....	40
4.2. Experimentos Realizados.....	46
4.3. Conclusões e Motivações para Mudança	56
5. UMA NOVA ABORDAGEM GENÉTICA.....	58
5.1. Tipagem e Instanciação	59
5.1.1. Inferência de Tipos de Parâmetros das Ações	63
5.1.2. Inferência de Tipos de Objetos do Problema	64
5.1.3. Instanciação das Ações	67
5.2. Novo Modelo Genético para Planejamento	68
5.3. Resultados e Adaptações no Modelo	75
6. CONCLUSÕES E TRABALHOS FUTUROS.....	80
APÊNDICE	83
Apêndice A - GAPlan – Um Planejador Genético	83
Apêndice B - Testes e Resultados Gráficos	92
REFERÊNCIAS BIBLIOGRÁFICAS	110

LISTA DE FIGURAS

Figura 1 - <i>Anomalia de Sussman no Mundo de Blocos.</i>	- Pg. 7
Figura 2 - <i>Arquivo em PDDL para descrição de domínio Mundo de Blocos.</i>	- Pg. 15
Figura 3 - <i>Arquivo em PDDL com o problema aplicado ao domínio Mundo de Blocos.</i>	- Pg. 16
Figura 4 - <i>Cenários de Cubo de Rubik (Cubo Mágico) e Sokoban</i>	- Pg. 26
Figura 5 - <i>Princípio básico de um AG.</i>	- Pg. 33
Figura 6 - <i>Pseudo-código para um AG.</i>	- Pg. 34
Figura 7 - <i>Operação Genética de Mutação.</i>	- Pg. 35
Figura 8 - <i>Operação Genética de Cruzamento (crossover).</i>	- Pg. 36
Figura 9 - <i>Representação de um cromossomo para um plano de até 20 ações</i>	- Pg. 41
Figura 10 - <i>Operador de Cruzamento (crossover) de 2 pontos.</i>	- Pg. 42
Figura 11 - <i>Operador aplicado de Mutação.</i>	- Pg. 42
Figura 12 - <i>Representação de um cromossomo binário paralelo equivalente.</i>	- Pg. 44
Figura 13 - <i>Gráficos da tabela 8. Curvas de fitness Máximo, Médio e Mínimo das populações.</i>	- Pg. 48
Figura 14 - <i>Gráficos da tabela 9.</i>	- Pg. 49
Figura 15 - <i>Gráficos da tabela 10.</i>	- Pg. 50
Figura 16 - <i>Gráficos da tabela 11.</i>	- Pg. 52
Figura 17 - <i>Gráficos do teste 28, encontrando a solução após 1539 gerações.</i>	- Pg. 53
Figura 18 - <i>Gráficos da tabela 12.</i>	- Pg. 54
Figura 19 - <i>Gráficos da tabela 13.</i>	- Pg. 55
Figura 20 - <i>Ação descrita em PDDL Não-Tipado para o domínio Logistics.</i>	- Pg. 60
Figura 21 - <i>Arquivo em PDDL com o Domínio Gripper Não-Tipado.</i>	- Pg. 61
Figura 22 - <i>Arquivo em PDDL com o Domínio Gripper Tipado.</i>	- Pg. 62
Figura 23 - <i>Arquivo em PDDL com o Problema Gripper de 4 bolas.</i>	- Pg. 63
Figura 24 - <i>Pseudocódigo para descobrir o TIPO de parâmetro das ações.</i>	- Pg. 64
Figura 25 - <i>Tipagem de Parâmetros no domínio Gripper com STRIPS-PDDL.</i>	- Pg. 64
Figura 26 - <i>Pseudocódigo para descobrir o TIPO dos objetos.</i>	- Pg. 65
Figura 27 - <i>Tipagem de Objetos no Problema 01 do Gripper não-tipado.</i>	- Pg. 66
Figura 28 - <i>Instanciação de Objetos Parametrizados em Ações .</i>	- Pg. 67
Figura 29 - <i>Pseudocódigo para instanciar ações.</i>	- Pg. 68
Figura 30 - <i>Representação de um cromossomo para um plano de até 20 ações.</i>	- Pg. 70
Figura 31 - <i>Representação do processo genético de Seleção por Torneio Probabilístico (Roleta).</i>	- Pg. 71
Figura 32 - <i>Tela 'Splash' enquanto inicializa e abre conexão com banco de dados.</i>	- Pg. 83

Figura 33 - <i>Tela inicial do programa – Guia Parâmetros Genéticos.</i>	- Pg. 84
Figura 34 - <i>Tela de escolha de Domínio e Problema na Gui a planejamento.</i>	- Pg. 86
Figura 35 - <i>Tela de Acompanhamento da Execução do GAPLAN.</i>	- Pg. 87
Figura 36 - <i>Tela para Configuração de Bateria de Testes.</i>	- Pg. 88
Figura 37 - <i>Tela para visualização e Análise dos Resultados.</i>	- Pg. 89
Figura 38 - <i>Exemplo de tela de visualização de Relatórios.</i>	- Pg. 91
Figura 39 - <i>Primeiros testes aplicados ao programa. Teste Nr. 7.</i>	- Pg. 92
Figura 40 - <i>Teste Nr. 52 Problema 4-0 do Mundo de Blocos.</i>	- Pg. 93
Figura 41 - <i>Arquivo PDDL com Problema 5-0 do Mundo de Blocos.</i>	- Pg. 93
Figura 42 - <i>Teste Nr. 76 – Solução encontrada para o problema 5-0 do Mundo de Blocos.</i>	- Pg. 94
Figura 43 - <i>Teste Nr.92 – Configurando Pesos para cálculo de Fitness .</i>	- Pg. 95
Figura 44 - <i>Primeira Bateria de Testes – Configuração Padrão de Testes .</i>	- Pg. 97
Figura 45 - <i>Testes Nr.284 e 323 – Controle de Estabilidade do AG unifica o funcionamento.</i>	- Pg. 97
Figura 46 - <i>Teste Nr.330 – Alteração do percentual de Executabilidade de um cromossomo .</i>	- Pg. 98
Figura 47 - <i>Testes Nr.337 e 338 – Diferença no desempenho com pouca variação de parâmetros.</i>	- Pg. 99
Figura 48 - <i>Testes Nr.379 e 391 – Diferença de desempenho com pouca variação de parâmetros.</i>	- Pg. 100
Figura 49 - <i>Arquivo PDDL com o problema 02 do domínio Gripper.</i>	- Pg. 100
Figura 50 - <i>Testes Nr.418 e 423 – Tamanho do cromossomo insuficiente. Testes infrutíferos.</i>	- Pg. 101
Figura 51 - <i>Testes Nr.430 e 437 – Menor plano e menor tempo da bateria.</i>	- Pg. 102
Figura 52 - <i>Testes Nr.444 e 453 – Sem o controle de estabilidade, poucos casos de sucesso.</i>	- Pg. 103
Figura 53 - <i>Testes Nr.460 e 469 – Menor tempo e menor plano da bateria.</i>	- Pg. 104
Figura 54 - <i>Tst. 500/515 – Com controle de estabilidade, parâmetros diferentes, resultados idênticos.</i>	- Pg. 106
Figura 55 - <i>Teste Nr.519 – Plano extenso para fator de executabilidade 90%.</i>	- Pg. 107
Figura 56 - <i>Testes Nr.523 e 535 – Mais rápido e menor plano, respectivamente, da bateria.</i>	- Pg. 108
Figura 57 - <i>Testes Nr.547 e 552 – Potencial evolutivo, mais gerações e cromossomos maiores.</i>	- Pg. 109

LISTA DE TABELAS

Tabela 1	- <i>Participantes da competição AIPS – 1998 [37].</i>	– Pg. 20
Tabela 2	- <i>Resultados da 1ª. Etapa da competição de 1998 [37].</i>	– Pg. 20
Tabela 3	- <i>Resultados da 2ª. Etapa da competição de 1998 [37].</i>	– Pg. 20
Tabela 4	- <i>Competidores da AIPS – 2000 [3].</i>	– Pg. 21
Tabela 5	- <i>Técnicas aplicadas pelos competidores da AIPS – 2000 [3].</i>	– Pg. 22
Tabela 6	- <i>Competidores da AIPS – 2002 [2].</i>	– Pg. 24
Tabela 7	- <i>Variação de Pesos conforme Gerações.</i>	– Pg. 45
Tabela 8	- <i>Primeiros experimentos.</i>	– Pg. 47
Tabela 9	- <i>Resultados após implementação de pesos variáveis.</i>	– Pg. 48
Tabela 10	- <i>Ajustes de parâmetros genéticos.</i>	– Pg. 49
Tabela 11	- <i>Mais ajustes de parâmetros genéticos.</i>	– Pg. 51
Tabela 12	- <i>Testes em outros domínios de planejamento.</i>	– Pg. 54
Tabela 13	- <i>Testes com Fator de correção e reavaliação do plano.</i>	– Pg. 55
Tabela 14	- <i>Controle de Estabilidade desativa controles conforme andamento das gerações.</i>	– Pg. 73
Tabela 15	- <i>Resumo das principais características e potencialidades do GAPlan.</i>	– Pg. 78

EPÍGRAFE

Um dia você aprende

Depois de algum tempo você aprende a diferença, a sutil diferença, entre dar a mão e acorrentar uma alma. E você aprende que amar não significa apoiar-se, e que companhia nem sempre significa segurança. E começa a aprender que beijos não são contratos e presentes não são promessas. E começa a aceitar suas derrotas com a cabeça erguida e olhos adiante, com a graça de um adulto e não com a tristeza de uma criança. E aprende a construir todas as suas estradas no hoje, porque o terreno do amanhã é incerto demais para os planos, e o futuro tem o costume de cair em meio ao vão. Depois de um tempo você aprende que o sol queima se ficar exposto por muito tempo. E aprende que não importa o quanto você se importe, algumas pessoas simplesmente não se importam. E aceita que não importa quão boa seja uma pessoa, ela vai feri-lo de vez em quando e você precisa perdoá-la, por isso. Aprende que falar pode aliviar dores emocionais. Descobre que se levam anos para se construir confiança e apenas segundos para destruí-la, e que você pode fazer coisas em um instante, das quais se arrepende pelo resto da vida. Aprende que verdadeiras amizades continuam a crescer mesmo a longas distâncias. E o que importa não é o que você tem na vida, mas quem você é na vida. E que bons amigos são a família que nos permitiram escolher. Aprende que não temos que mudar de amigos se compreendemos que os amigos mudam, percebe que seu melhor amigo e você podem fazer qualquer coisa, ou nada, e terem bons momentos juntos. Descobre que as pessoas com quem você mais se importa na vida são tomadas de você muito depressa, por isso sempre devemos deixar as pessoas que amamos com palavras amorosas, pode ser a última vez que as vejamos. Aprende que as circunstâncias e os ambientes têm influência sobre nós, mas nós somos responsáveis por nós mesmos. Começa a aprender que não se deve comparar com os outros, mas com o melhor que você mesmo pode ser. Descobre que se leva muito tempo para se tornar a pessoa que quer ser, e que o tempo é curto. Aprende que não importa aonde já chegou, mas onde está indo, mas se você não sabe para onde está indo, qualquer lugar serve. Aprende que, ou você controla seus atos ou eles o controlarão, e que ser flexível não significa ser fraco ou não ter personalidade, pois não importa quão delicada e frágil seja uma situação, sempre existem dois lados. Aprende que heróis são pessoas que fizeram o que era necessário fazer, enfrentando as consequências. Aprende que paciência requer muita prática. Descobre que algumas vezes a pessoa que você espera que o chute quando você cai é uma das poucas que o ajudam a levantar-se. Aprende que maturidade tem mais a ver com os tipos de experiência que se teve e o que você aprendeu com elas do que com quantos aniversários você celebrou. Aprende que há mais dos seus pais em você do que você supunha. Aprende que nunca se deve dizer a uma criança que sonhos são bobagens, poucas coisas são tão humilhantes e seria uma tragédia se ela acreditasse nisso. Aprende que quando está com raiva tem o direito de estar com raiva, mas isso não lhe dá o direito de ser cruel. Descobre que só porque alguém não o ama do jeito que você quer que ame, não significa que esse alguém não o ama, pois existem pessoas que nos amam, mas simplesmente não sabem como demonstrar isso. Aprende que nem sempre é suficiente ser perdoado por alguém, algumas vezes você tem que aprender a perdoar-se a si mesmo. Aprende que com a mesma severidade com que julga, você será em algum momento condenado. Aprende que não importa em quantos pedaços seu coração foi partido, o mundo não pára para que você o conserte. Aprende que o tempo não é algo que possa voltar para trás. Portanto, plante seu jardim e decore sua alma, ao invés de esperar que alguém lhe traga flores. E você aprende que realmente pode suportar... que realmente é forte, e que pode ir muito mais longe depois de pensar que não se pode mais. E que realmente a vida tem valor e que você tem valor diante da vida! Nossas dúvidas são traidoras e nos fazem perder o bem que poderíamos conquistar, se não fosse o medo de tentar.

Sir William Shakespeare

AGRADECIMENTOS

Em primeiro lugar, sinto que preciso agradecer ao maior responsável pela minha caminhada, a quem chamo de Deus. Agradeço-lhe pelo sopro de vida, no começo de tudo, e também por ter influenciado as minhas escolhas de vida para que o destino pudesse escrever os passos da minha trajetória.

Sou eternamente grato aos meus pais, José e Cyrce, por honrarem aos seus compromissos de garantir aos filhos uma educação reta e de qualidade, forjando o caráter e formatando a nossa personalidade. Quero um registro de agradecimento especial ao meu querido pai, que durante o desenvolvimento deste trabalho deixou de estar presente fisicamente entre nós, mas continua sempre ao meu lado iluminando o meu caminho.

Agradeço a minha querida esposa Ione pelo carinho, amor e dedicação que demonstra no dia-a-dia e que me servem de complemento, inspiração e força para continuar a busca por melhores dias, e de forma distinta, à minha amada filha Nikole, cujo carinho, amor e dedicação estão sempre à espera do mesmo complemento, inspiração e força para modelar seu aprendizado da vida. Vocês são responsáveis por eu ser uma pessoa melhor e mais feliz a cada dia.

Aos verdadeiros amigos agradeço pelo agradável convívio, pelo respeito, camaradagem, confiança e amizade que me fazem acreditar que existem pessoas especiais que o destino nos impõe para suavizar nossas quedas, enxugar nossas lágrimas, apoiar nossa recuperação, estar ao lado nas derrotas e conquistas, desinteressadamente, apenas por que, independentemente do tempo que estivermos distante, é sempre bom um novo encontro. A estes, sequer preciso nomear ou quantificar, pois saberão intuitivamente a quem me dirijo.

Além da amizade, agradeço a motivação, inspiração, apoio e exemplo do professor Dr. Oge Marques Filho, que leciona hoje na Universidade de Atlanta – USA.

Agradeço a todos os professores do departamento de informática da UFPR, com quem tive o privilégio de estudar, ampliar e especializar meus conhecimentos. Aos que farão parte da banca avaliadora, agradeço pelas importantes contribuições ao trabalho que certamente serão acrescentadas. Agradeço ao esforço e apoio de co-orientação do professor Fabiano da Silva.

Por fim, de forma muito especial, agradeço à paciência, dedicação, apoio, insistência, persistência, colaboração e amizade, dedicados ao árduo trabalho de orientação do professor Marcos Alexandre Castilho.

RESUMO

Este trabalho apresenta uma revisão bibliográfica atualizada sobre duas grandes áreas da Inteligência Artificial: Planejamento e Algoritmos Genéticos. A pesquisa se estende pela criação de modelos genéticos implementados em um sistema planejador dedicado à resolução de uma conhecida classe de problemas de planejamento, usando bibliotecas de código de domínio público em ambas as áreas. Uma análise dos resultados motivou a remodelagem e nova implementação, alterando a plataforma e o sistema operacional e reescrevendo o código sem o uso das bibliotecas de domínio público. O modelo foi readaptado e ajustado conforme os resultados dos novos testes, para o uso de novos operadores genéticos não-convencionais, permitindo alcançar novas conclusões sobre a abordagem original proposta. Outra importante contribuição apresenta uma forma alternativa e simples de inferir tipos de objetos e parâmetros na linguagem PDDL para instanciar corretamente as ações e reduzir drasticamente o espaço de busca.

ABSTRACT

This work presents an up-to-dated literature review about two Artificial Intelligence areas: Planning and Genetic Algorithms. The investigation evolved to create genetic models implemented into a planning system applied to solve a known class of planning problems that uses public-domain code librarians for both areas. One of the resulting analysis has motivated the remodeling and a new implementation, which was accomplished in this work through changing the platform and the operating system by rewriting the code without the use of the public domain librarians. The model was readapted and adjusted in accordance with new test results, for to be used with non-conventional new genetic operators, allowing new conclusions about the proposed approach. Another important contribution presents an alternative and simple way to infer kinds of objects and parameters in the PDDL language to correctly instantiate the actions and to reduce drastically the search space.

1. INTRODUÇÃO

Desde os primórdios o homem luta para construir máquinas capazes de diminuir o seu desgaste físico e intelectual na execução de tarefas repetitivas ou que exijam muito esforço. Inventou a roda, criou ferramentas, descobriu o fogo e construiu a alavanca. Há 300 anos, no tempo de Isaac Newton¹, a vida era curta e brutal. Sua expectativa de vida era de 30 anos, as famílias tinham dez filhos, em média para que dois ou três sobrevivessem, a maioria das pessoas era analfabeta e não tinha livros. A escola era para poucos. Quase todos passavam a vida em torno do local onde haviam nascido e não se aventuravam a mais de dois ou três quilômetros. Pouco antes do sol nascer já estavam trabalhando, principalmente no campo, e assim que o sol se punha já estavam na cama. Quase não havia distração, lazer ou esporte. A fome era crônica, as doenças crônicas comuns. A ciência, como a conhecemos, não existia e as leis da natureza estavam cobertas de mistério, medo e superstição. E, no entanto, Newton foi capaz de fazer o que fez.

A mecânica de Newton permitiu a criação de máquinas poderosas e, finalmente, o motor a vapor que provocou a Revolução Industrial e mudou o mundo.

Segundo o artigo [5], o século XIX foi de intensa descoberta científica, tirando muitas pessoas da miséria e da ignorância, enriquecendo a vida, ampliando o conhecimento, derrubando as dinastias feudais. No fim do século XX a ciência já conhecia os mistérios do átomo, da célula, do chip eletrônico. Essas três descobertas desencadeadas pela revolução quântica, a revolução do DNA² e a revolução da informática abriram ao conhecimento do homem as leis básicas da matéria, da vida e da computação, abriram o futuro.

Os três temas principais do século passado foram o átomo, o gene e o computador. Os deste século serão uma extensão natural desses temas, levados às últimas conseqüências. A revolução quântica permitirá, no futuro, um dia, talvez ainda neste século, o controle da matéria. A revolução biomolecular tornará o homem capaz de modificar o próprio homem através da engenharia genética, e de criar vida, de sintetizar e inventar novas drogas, novas terapias, novas formas de vida. A revolução da informática

¹ Matemático e físico inglês, considerado um dos maiores cientistas da história, fez contribuições importantíssimas em vários campos da ciência. Newton foi um dos inventores, junto com o alemão Wilhelm Leibniz, do cálculo estudado na matemática. Ele também solucionou os mistérios da luz e ótica, e formulou as três leis de movimento, criando a partir delas a famosa Lei da Gravidade, inspirado pela queda de uma maçã de uma árvore.

² DNA. É o ácido desoxirribonucleico, que é composto por bases ou nucleosídeos: adenina, timina, e citosina e guanina. A adenina sempre se liga com a timina e a citosina com a guanina. O DNA humano tem 3,1 bilhões de pares de bases. É no DNA que estão os genes.

que já globalizou o mundo e está transformando o mundo a partir das comunicações, vai continuar mudando nosso modo de vida e um dia, talvez ainda no século XXI, permitirá introduzir inteligência artificial em qualquer lugar do planeta.

Estamos na fronteira de mais uma revolução do conhecimento, mais profunda e mais completa do que as do século passado, porque o conhecimento humano duplica a cada dez anos e na segunda metade do século XX foi gerado mais conhecimento científico do que em toda a história humana somada.

A capacidade dos computadores duplica, no máximo a cada 18 meses. A rede internacional de computadores duplica de tamanho e alcance a cada 12 meses. Nos últimos dois anos o número de seqüências de DNA sob investigação triplicou. A mídia registra, quase diariamente, avanços em computação, telecomunicações, biotecnologia, microbiologia. As mudanças que nos provocam são quantitativas e qualitativas [5].

Mesmo com a atual capacidade de processamento, alguns problemas computacionais ainda aguardam pelas próximas fases evolutivas do hardware para o surgimento de soluções práticas viáveis. Na tentativa de construir máquinas que imitam o comportamento humano, a área de Inteligência Artificial necessita de técnicas especiais para o tratamento dos impasses de processamento gerados pela grande complexidade computacional e pelo conhecido problema da explosão combinatória em situações onde o processamento do cérebro humano se destaca, como as encontradas no planejamento de ações para a solução de problemas genéricos.

Planejar em Inteligência Artificial, significa encontrar a combinação correta de uma seqüência de ações parametrizadas que conduzem os agentes de um estado inicial fornecido para um estado final desejado. O problema do planejamento em Inteligência Artificial é antigo e importante [18], mas ficou estagnado por cerca de 20 anos no infrutífero método de buscas exaustivas.

Na verdade, este é um problema de complexidade reconhecidamente difícil, tendo sido provado ser PSPACE-completo [14]. O espaço de busca, além de ser exponencial, geralmente apresenta um crescimento extremamente rápido, o que leva a problemas derivados, como por exemplo, o de falta de memória.

Vários trabalhos recentes na literatura científica da área estão direcionados ao estudo e tratamento do problema do planejamento, em especial o uso de Funções Heurísticas Clássicas [9],[10],[36], da Teoria dos Grafos [11], de Programação Inteira [8],[27], de Redes de Petri [49], de Satisfação de Restrições [6],[45],[46] e de Prova de Teoremas [25],[26]. O problema passou a ser considerado multidisciplinar, ganhando uma

nova dinâmica. Foram criadas competições internacionais entre sistemas planejadores para concentrar os avanços e melhor acompanhar os resultados.

Anteriormente, pequenos problemas levavam horas ou dias [55] para serem resolvidos e tratavam geralmente de problemas meramente conceituais. Com o uso destas novas técnicas, alguns problemas reais passaram a ser tratados e grandes instâncias destes passaram a ser resolvidos em um tempo muito menor, sendo possível gerar soluções em poucos segundos ou milissegundos.

Com os novos métodos, surgiram pesquisas direcionadas à solução de alguns problemas mais complexos de planejamento, enquanto que outros ainda permanecem à espera de novas abordagens técnicas e pela evolução da área. Apesar de alguns destes novos algoritmos serem geralmente muito rápidos e capazes de lidar com grandes instâncias de problemas em domínios específicos, existem ainda alguns problemas que não são tratados, principalmente aqueles para os quais a árvore de busca no espaço de estados é extremamente grande.

A grande diversidade de domínios que podem ser tratados amplia o grau de complexidade do sistema de planejamento para novas fronteiras que vão além da complexidade já citada inerente aos problemas propriamente ditos. Um planejador, cujo objetivo é simular o comportamento humano, não pode depender especificamente de um domínio para funcionar. Precisa resolver os problemas genéricos em qualquer domínio, como acreditamos que acontece com o sistema humano de resolução de problemas genéricos. Diversas técnicas de Inteligência Artificial trabalham para minimizar os efeitos deste acúmulo de complexidade e tornar tratável este tipo de problema.

Uma técnica ainda pouco abordada em Planejamento é a Computação Evolutiva, em especial os Algoritmos Genéticos [22] (AG's), sendo que esta abordagem é o tema principal deste trabalho de pesquisa. Alguns trabalhos podem ser encontrados utilizando "Programação Genética" em planejadores direcionados fundamentalmente para problemas específicos no domínio de navegação e movimentação de robôs, como *Sinergy* [39], [40] e *GenPlan* [56], bem como outros autores que direcionaram o mesmo problema com robôs ao uso de Algoritmos Genéticos, como o *Ariadne's Clew Algorithm* [32], [33], [34], e outros trabalhos semelhantes encontrados em [17], [57] e [7], mas que diferem completamente da nossa abordagem e não podem ser consideradas como um tratamento para o problema do planejamento, pois são completamente dependentes do domínio.

O tratamento de situações genéricas de Planejamento a partir de Algoritmos Genéticos se mostra bastante interessante no cenário atual da área, pois apresenta um grande espaço de busca, o que é uma característica dos problemas tratados de forma satisfatória por Algoritmos Genéticos. Outro fator relevante é o relaxamento da restrição de otimização da solução, presente em vários planejadores atuais, o que facilita a busca pela solução no AG.

Ao unirmos estas duas sub-áreas da Inteligência Artificial, nosso principal objetivo foi estudar e investigar as técnicas de Computação Evolutiva, em particular os Algoritmos Genéticos para a resolução do problema genérico de Planejamento, desde a melhor maneira de representação e modelagem do estado inicial e ações para o problema. Neste sentido, trata-se de uma abordagem inédita. Apresentamos duas abordagens genéticas para o problema de planejamento em Inteligência Artificial e seus resultados.

O texto do trabalho está organizado da seguinte maneira: o próximo capítulo contém a fundamentação teórica dos problemas de Planejamento, apresenta a base da evolução e o Estado da Arte na área, enquanto que no capítulo 3 está o histórico, os conceitos e o funcionamento dos Algoritmos Genéticos.

No capítulo 4 apresentamos uma abordagem genética ao problema do planejamento, incluindo a modelagem do problema, os resultados dos experimentos aplicados a conhecidos domínios do problema de Planejamento.

No capítulo 5, com base na análise dos resultados e gráficos produzidos pela modelagem original, apresentamos uma mudança importante na modelagem e na forma de implementação, reescrevendo o código e substituindo o uso das bibliotecas de domínio público com a apresentação de resultados mais satisfatórios e mais próximos do ideal. Uma seção deste capítulo descreve uma forma simplificada para a inferência de tipos de objetos e parâmetros para a correta instanciação das ações e conseqüente redução do espaço de busca.

As conclusões com as principais contribuições do trabalho e direcionamento para futuros desenvolvimentos estão apresentadas no capítulo 6. O leitor interessado encontrará nos apêndices uma descrição formal da interface visual desenvolvida e o detalhamento dos principais testes aplicados ao programa que fundamentam as conclusões deste trabalho de pesquisa.

2. PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

O desenvolvimento de solucionadores de problemas gerais tem sido um dos principais objetivos da área de Inteligência Artificial. O desafio consiste em modelar o comportamento dos humanos que, segundo acredita-se, resolvem problemas genéricos, isto é, adaptam-se aos problemas para obterem a melhor solução. Um programa de computador que aceite descrições de alto nível dos problemas e processe automaticamente a sua solução pode ser considerado como um resolvidor geral de problemas [42]. O escopo pode ser definido por meio de uma classe de modelos matemáticos que define os tipos de problemas, as formas de solução e quais soluções são melhores ou ótimas. Um sistema de Planejamento em Inteligência Artificial, dentro desta perspectiva, é um resolvidor de problemas que possui uma representação conveniente e uma solução efetiva para uma certa classe de modelos matemáticos.

O problema de Planejamento é o de encontrar uma seqüência de ações, chamada *plano*, que resolva um determinado problema proposto. Mais especificamente, um Sistema de Planejamento é aquele que recebe como entrada uma descrição de um estado inicial do mundo, de um objetivo e de um conjunto de ações que podem ser aplicadas e gera como saída um plano, que é a seqüência de ações que transforma o estado inicial no estado final.

Na metodologia aceita atualmente pela comunidade que estuda este problema, um planejador deve ser capaz de ler um problema de planejamento descrito no formato PDDL³ [20], detalhado na próxima seção, que consiste de uma descrição do comportamento das ações e uma descrição do estado inicial e final. Deve retornar dentro de um limite de tempo previamente especificado e consumindo, no máximo, uma certa quantidade de memória, uma seqüência de ações que, quando executadas numa dada ordem sobre o estado inicial, transforma-o no estado final. Qualquer seqüência de ações que faça isto é considerada um plano.

Infelizmente, não existe um tamanho fixo para o plano. Na maioria das situações, é suposto que o planejador deve retornar um plano ótimo, isto é, um plano de tamanho mínimo. Por outro lado, como isto é difícil de acontecer, apenas encontrar um plano é, em geral, suficiente. Entretanto, se dois planejadores encontram dois planos não ótimos, o

³ PDDL – Planning Domain Definition Language. Linguagem de Definição de Domínios de Planejamento.

menor plano é considerado o melhor. Neste trabalho, consideramos que qualquer plano que resolva o problema é uma solução.

Notamos que a formulação do problema de planejamento, como definida acima, é muito abstrata. De fato, ela realmente especifica uma classe de problemas de planejamento parametrizados por linguagens usadas para representar o mundo, objetivos e ações. Neste trabalho, estaremos interessados em uma classe de problemas conhecida como planejamento clássico.

Formalmente, segundo Newell & Simon [41], [42], [48], o planejamento clássico pode ser entendido em termos de modelos determinísticos de estados que é uma classe que inclui problemas de decisões seqüenciais com ações determinísticas e informação completa, caracterizados pelos seguintes elementos:

- ? S1 - um espaço de estados S , finito e não vazio;
- ? S2 - um estado inicial $s_0 \in S$;
- ? S3 - um conjunto não-vazio de estados-meta $S_G \subseteq S$;
- ? S4 - um conjunto de ações $A(s) \subseteq A$ aplicáveis para $s \in S$;
- ? S5 - uma função de transição $f(a, s)$, para $a \in A(s)$;
- ? S6 - custo das ações $c(a, s) > 0$.

Uma solução para modelos deste tipo é uma seqüência de ações a_0, a_1, \dots, a_n que gera uma trajetória de estados $s_0, s_1=f(s_0, a_0), \dots, s_{n+1}=f(s_n, a_n)$, tal que cada ação a_i é aplicável em s_i e s_{n+1} é o estado-meta, isto é, $a_i \in A(s_i)$ e $s_{n+1} \in S_G$. A solução é ótima quando o custo total é mínimo, dado pela soma de todos os custos parciais. No planejamento clássico, é assumido que todos os custos parciais são iguais e portanto, o plano ótimo é aquele que possui o comprimento mínimo.

A primeira proposta de solução para o problema de planejamento clássico, integrando um sistema formal e um algoritmo correspondente, foi apresentada por Fikes e Nilsson em 1971, o sistema *STRIPS*⁴ [18]. O algoritmo proposto trata o problema do planejamento como um problema de busca tradicional. Em outras palavras, o algoritmo procura por uma solução utilizando uma busca em árvore de estados do mundo de tamanho exponencial. É um algoritmo simples e trata na prática um número pequeno de problemas.

A grande relevância do trabalho de Fikes e Nilsson está no sistema formal proposto, que permite representar ações e estados de maneira simples, produzindo uma

⁴ STRIPS (STanford Research Institute Problem Solver) – O programa original STRIPS foi projetado para controlar Shakey, um robô que vagava pelas salas do Stanford Research Institute, no início da década de 70.

independência entre a linguagem e o algoritmo que resolve o problema. Este formalismo é chamado STRIPS.

O formalismo STRIPS requer uma representação das variáveis e axiomas como descritores de estados e um conjunto de ações proposicionais aplicado como operadores em listas de precondições, adições e exclusões. Os estados do mundo são representados por conjunções de literais instanciadas, ou seja, predicados aplicados sobre constantes.

Por exemplo, consideremos o clássico domínio do “*Mundo de Blocos*”, onde os blocos estão dispostos sobre uma mesa e um braço mecânico deve movê-los de uma configuração para outra, respeitando algumas regras, que são melhor descritas na seção 2.1.1.1. O estado inicial, na Figura 1 abaixo, que ilustra uma situação conhecida como “*Anomalia de Sussman*” [51], pode ser descrito por:

$$\text{sobre}(C, A) \text{ ? } \text{sobre}(A, \text{Mesa}) \text{ ? } \text{sobre}(B, \text{Mesa}) \text{ ? } \text{livre}(B) \text{ ? } \text{livre}(C)$$

e o estado-meta por:

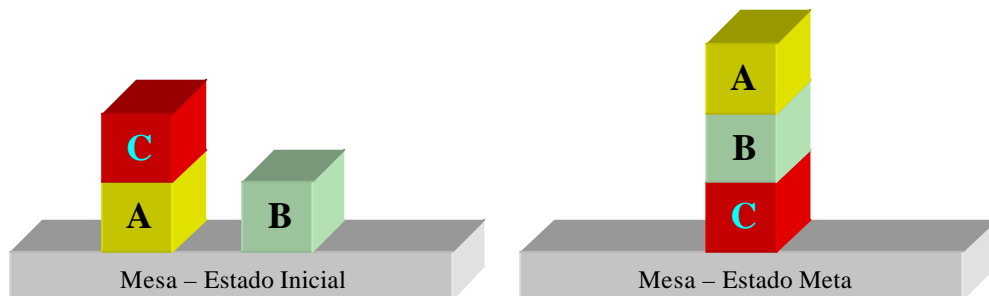
$$\text{sobre}(C, \text{Mesa}) \text{ ? } \text{sobre}(A, B) \text{ ? } \text{sobre}(B, C)$$


Figura 1. Anomalia de Sussman no Mundo de Blocos.

Por ser requerido que a descrição do estado inicial seja completa, todas as formulações e axiomas não listados explicitamente na descrição do problema são assumidas como falsas (isto é chamado “*Closed Word Assumption*” – *Hipótese do Mundo Fechado* [44] - usada para tratar o “*Frame Problem*” – *Problema do Quadro* [35]). Isto significa que ($? \text{sobre}(A, C)$) e ($? \text{livre}(A)$) estão assumidos implicitamente na descrição do estado inicial, bem como um grupo de outros literais negativos.

A representação em STRIPS restringe o tipo dos estados-meta que podem ser especificados para aqueles compostos por conjunções de literais positivos, que em conjunto com a Hipótese do Mundo Fechado para a definição do estado inicial, são dois dos três componentes que definem um problema de planejamento. O terceiro é denominado de Teoria do Domínio que é a descrição formal das ações que estão disponíveis para o agente.

As ações são representadas por três partes: descrição, precondições e efeitos. A descrição define o nome e os parâmetros da ação. A precondição é uma conjunção de literais positivos que devem ser verdadeiros para que a ação possa ser aplicada. Um efeito de ação, por outro lado, é uma conjunção que pode incluir tanto literais positivos como negativos. Eles descrevem as alterações que devem ser realizadas sobre um estado do mundo a fim de obter um outro. Por exemplo, podemos definir a ação de mover o bloco C de cima do bloco A para cima da mesa, como segue:

? Descrição: *mover(C, A, Mesa);*
 ? Pré-Condição: *sobre(C, A) ? livre(C);*
 ? Efeito: *sobre(C, Mesa) ? ? sobre(C, A) ? livre(A).*

As ações podem ser executadas apenas quando suas precondições são verdadeiras; neste caso, só podemos mover o bloco C de A para a Mesa se C estiver sobre A e C não tiver nada sobre ele. Quando a ação é executada, ela muda a descrição do mundo da seguinte maneira. Todos os literais positivos na conjunção de efeitos (chamados de ações *Add-List*) são adicionados na descrição do estado, enquanto que todos os literais negativos (chamados de ações *Del-List*) são removidos. Por exemplo, executando esta mesma ação, *mover(C, A, Mesa)*, sobre o estado inicial descrito no problema original da Anomalia de Sussman, obteríamos um estado no qual,

sobre(A, Mesa) ? sobre(B, Mesa) ? sobre(C, Mesa) ? livre(A) ? livre(B) ? livre(C)

são verdadeiras, e todas as outras formulações atômicas são falsas.

Isto conclui a descrição das entradas de um sistema de planejamento: uma descrição do estado inicial, uma descrição do estado-meta e a teoria do domínio, que é um conjunto de descrições de ações. Por exemplo, para o problema da Anomalia de

Sussman, poderíamos esperar que o resultado da execução de um sistema planejador fosse:

```
mover(C, A, Mesa)
mover(B, Mesa, C)
mover(A, Mesa, B)
```

No entanto, dependendo do algoritmo que busca uma solução, isto pode ser difícil de se obter, exatamente como aconteceu com o problema em questão que não foi resolvido por nenhum planejador da época, basicamente porque usavam métodos de busca heurística que não conseguiam vencer os mínimos locais como:

```
sobre(C, Mesa) ? sobre(A, B)
```

Mais formalmente, um planejador é uma tupla $\langle A, O, I, G \rangle$ onde:

- ? A representa o conjunto de todos os átomos (variáveis booleanas e axiomas como descritores de estados);
- ? O representa o conjunto de todos os operadores (ações proposicionais);
- ? $I ? A$ representa a situação inicial;
- ? $G ? A$ representa a situação-meta.

Operadores $o ? O$ são representados por três listas:

- ? a lista de Precondição, $Prec(o) ? A$;
- ? a lista de Adição, $Add(o) ? A$;
- ? a lista de Deleção, $Del(o) ? A$.

Intuitivamente, $Prec(o)$ especifica os átomos que devem ser verdadeiros para que " o " seja aplicável, $Add(o)$ especifica os átomos que passam a ser verdadeiros após a execução de " o " e $Del(o)$ especifica os átomos que passam a ser falsos após a execução de " o ". Um problema na semântica *STRIPS* determina um modelo de estados $S(P)$, pela tupla $P = \langle A, O, I, G \rangle$, onde:

- ? os estados $s ? S$ são coleções de átomos;
- ? o estado inicial s_0 é I ;

- ? os estados-meta $s \in S_G$ são tais que $G \models s$;
- ? as ações $a \in A(s)$ são operadores $o \in O$ tal que $Prec(o) \models s$;
- ? a função de transição f mapeia estados s em estados $s' = f(s,a)$, tal que $s' = s - Del(a) + Add(a)$ para $a \in A(s)$;
- ? os custos das ações $c(a, s)$ são todos iguais a 1.

A solução (ótima) do problema de planejamento P é a solução (ótima) do modelo de estados $S(P)$.

Dezenas de sistemas planejadores foram propostas nos últimos 10 anos. A comunidade entendeu que era preciso estabelecer um padrão e criar uma nova metodologia para avaliar e comparar cada um deles com os outros, além de estabelecer um ponto de verificação sobre o andamento das novas técnicas aplicadas na área. Com isto, surgem a partir de 1998 as competições entre planejadores, cujo principal objetivo não é o de estabelecer vencedores ou perdedores, mas sim estabelecer os tais padrões e acompanhar o desenvolvimento e evolução dos sistemas aplicados ao planejamento em Inteligência Artificial.

Na próxima seção apresentaremos a metodologia atualmente empregada para tanto enquanto na seção 2.2 abordaremos a evolução da área através do acompanhamento das competições entre os sistemas planejadores.

2.1. MODELAGEM DE PLANEJAMENTO

Uma notação universal que foi criada para descrever os problemas apresentados na competição de planejadores ocorrida na Conferência bienal AIPS⁵ de 1998 chama-se PDDL, Linguagem de Definição de Domínios de Planejamento. Estas competições estão detalhadas a seguir na próxima seção.

A linguagem PDDL foi projetada para ser uma especificação neutra dos problemas de planejamento. Neutra significa não ser favorável a nenhum sistema de planejamento. Problemas e domínios descritos de uma maneira universal podem ser submetidos a qualquer planejador que entenda a linguagem de descrição. A linguagem PDDL descende, além do STRIPS original, de várias outras notações: ADL [43], formalismo

⁵ AIPS – International Conference on Artificial Intelligence Planning Systems. Conferência Internacional sobre Sistemas de Planejamento em Inteligência Artificial.

SIPE-2 [58], formalismo Prodigy-4.0 [13], formalismo UMCP [16], formalismo Unpop [36], e mais diretamente do formalismo UCPOP [4].

Em PDDL podemos encontrar uma família de linguagens capazes de adaptar os planejadores a diferentes capacidades. Caracteriza-se, principalmente, por representar ações no estilo do formalismo STRIPS, efeitos condicionais, quantificação universal sobre universos dinâmicos (i.e., criação e destruição de objetos), axiomas de domínio sobre teorias estratificadas, especificação de restrições de segurança, de hierarquia de ações compostas de sub-ações e sub-objetivos e gerenciamento de múltiplos problemas em múltiplos domínios utilizando diferentes subconjuntos de características da linguagem.

PDDL pretende expressar a “física” de um domínio, composta por quais predicados existem, quais ações são possíveis, qual o significado da estrutura de ações compostas, e quais os efeitos das ações. A linguagem está dividida em subconjuntos de características, chamadas de requerimentos. Todo domínio definido com o uso desta linguagem deve declarar quais requerimentos assume, adotando o requerimento `:strips` quando nada for declarado, o qual indica que a semântica das linguagens irá considerar o mundo como um conjunto de situações e estados, onde cada estado é especificado pelo estabelecimento de uma lista de todos os predicados que são verdadeiros.

Sua sintaxe é limpa e precisamente definida dentro do manual. As semânticas da versão 1.2, contudo, são informais e parecem estar distribuídas ao longo dos manuais próprios das linguagens e sistemas que a PDDL substitui, como UCPOP, processadores da linguagem PDDL e o interpretador LISP. Apesar do fato da sintaxe da PDDL ser similar ao LISP parecer um argumento secundário, o significado de muitas de suas funções primitivas é dado em termos de funções do LISP. Com o aumento da complexidade da linguagem, então as semânticas da linguagem natural se tornam menos óbvias. Todas as extensões da linguagem precisam estar corretamente definidas, pois quando dois sistemas usarem estas extensões, devem fazê-lo de maneira uniforme, para preservar o padrão estabelecido.

PDDL é uma linguagem muito expressiva para uma grande variedade de aplicações de planejamento, o que tem sido mostrado pela variedade de domínios de problemas usados em competições e em conjuntos avaliativos. Além disso, a habilidade para alterar algumas suposições do ambiente está presente, apesar das semânticas de algumas destas extensões não estarem claras. A seguir, na seção 2.1.1, apresentamos a forma de representação de Domínios com a linguagem PDDL e na seção 2.1.2

representamos os Problemas em PDDL. Domínios e Problemas compõem a base de trabalho dos sistemas planejadores.

2.1.1. Representação de Domínios

Domínios são os cenários utilizados como protótipos na representação dos elementos que compõem um modelo proposto para uma determinada interpretação do mundo. Uma definição de domínio deve conter além da identificação deste domínio, as cláusulas que definem os seus requisitos, como por exemplo o formalismo STRIPS, as cláusulas que definem os seus predicados e uma lista de variáveis aplicadas a cada um, como a igualdade (“=”), por exemplo, que é um predicado pré-definido que aceita dois argumentos de qualquer tipo, e as cláusulas que definem as ações válidas, subdivididas em outras cláusulas para definir os parâmetros, as pré-condições e os efeitos de cada ação.

Em PDDL, um domínio é um “modelo” no sentido em que se tem uma representação que pode ser usada para executar operações. Uma definição de domínio está estruturada dentro de componentes através de palavras-chave, como `:constants` `:actions`, etc. Uma palavra-chave especial é `:requirements` que indica ao processo qual a mistura de características de PDDL que será usada na definição do domínio.

Como veremos na próxima seção, existe uma grande variedade de domínios estudados na área de Planejamento em IA. Na continuação da seção atual, apresentaremos uma descrição sucinta e introdutória de um subconjunto dos principais domínios que serviram como base para o estudo dos problemas de planejamento tratados neste trabalho, quais sejam, o domínio do Mundo de Blocos, o Gripper e o Logistics.

2.1.1.1. Domínio Blocksworld-3ops

O domínio do Mundo de Blocos tem sua origem que remonta aos anais de Inteligência Artificial, parecendo ter sido primeiramente mencionado na tese de Terry Winograd, em 1972 e no artigo de planejamento de Gerald Sussman, de 1974.

É um domínio clássico não-tipado em STRIPS, onde blocos empilháveis precisam ser remontados sobre uma mesa com espaço ilimitado. A representação usa 3 operadores, movendo um bloco da mesa para outro bloco, um bloco de outro bloco para a mesa, ou um bloco de um bloco para outro bloco.

Semanticamente, a representação não usa um braço de robô, diferentemente da representação abaixo para 4 operadores. O estado inicial especifica um estado completo, os estados de objetivos especificam apenas as relações requeridas entre quaisquer dois blocos. O número de blocos é o parâmetro utilizado.

É um domínio cuja característica do espaço de estados é a de ser realmente muito grande, e que em geral, poucos planejadores conseguem resolver os problemas com 10 ou mais blocos, devido à explosão combinatória no espaço de busca.

2.1.1.2.Domínio Blocksworld-4ops

Usado na competição AIPS-2000, é idêntico ao domínio com 3 operadores, exceto por usar um braço de robô para empilhar um bloco sobre outro bloco, desempilhar um bloco de outro bloco, soltar um bloco ou pegar um bloco.

Também usa como parâmetro o número de blocos e possui a mesma característica de explosão combinatória que o item anterior.

2.1.1.3.Domínio Gripper

Domínio não-tipado e baseado em STRIPS, foi usado na competição AIPS-1998, sugerido por Jana Koehler.

Descreve um robô com algumas mãos com 'garras' (*gripper*), para o transporte de um determinado número de bolas de um quarto A para um quarto B. Utiliza como parâmetro o número de bolas. O estado inicial posiciona todas as bolas no quarto A e requer o objetivo final de encontrar todas as bolas no quarto B. O número de combinações de ações possíveis, dependente direto do parâmetro de número de bolas, facilmente se configura em explosão combinatória.

Este domínio apresenta ainda características de leve paralelismo, isto é, parte das suas ações pode acontecer simultaneamente, ao contrário do que acontece no domínio do mundo de blocos, onde as ações precisam ser executadas sequencialmente. Por exemplo, a ação de pegar uma bola com a garra esquerda pode acontecer ao mesmo tempo em que ocorre a ação de soltar uma bola com a garra direita. Esta característica geralmente não é levada em consideração pela maioria dos planejadores que mesmo assim, conseguem um bom desempenho na solução de problemas para este domínio com um número elevado de bolas.

2.1.1.4.Domínio Logistics

Clássico domínio de planejamento, não-tipado e baseado em STRIPS. Foi utilizado em ambas competições de AIPS-1998 e AIPS-2000. A versão de 98 foi criada por Bart Selman e Henry Kautz. Em 2000, usou-se a versão de Manuela Veloso.

Este domínio descreve o transporte de pacotes dentro de cidades através de caminhões, e entre cidades através de aviões. Os locais dentro da cidade estão diretamente conectados, isto é, os caminhões podem se mover entre quaisquer dois lugares, e da mesma forma, com aviões e cidades. Em cada cidade, existe exatamente um caminhão e um lugar que serve como um aeroporto. Utiliza os parâmetros de número de cidades, tamanho de cada uma, isto é, número de localidades dentro da cidade, número de pacotes e de aviões.

No seu estado inicial, posiciona aleatoriamente os caminhões dentro das suas cidades, posiciona os aviões aleatoriamente nos aeroportos. Distribui os locais de partida e objetivos de chegada dos pacotes aleatoriamente em todos os locais.

É um domínio que envolve um maior número de parâmetros, de variáveis, de ações e de predicados e por este motivo é considerado um domínio bastante complexo, onde poucos planejadores obtêm sucesso. A falta de tipagem para os objetos gera um número muito grande de instanciações inválidas das ações, comprometendo o espaço de busca, mas a característica mais marcante neste domínio é a possibilidade de paralelismo de ações, sendo que a maioria delas pode acontecer concorrentemente, como por exemplo, a ação de carregar um caminhão em determinado lugar pode acontecer ao mesmo tempo em que chega algum avião na cidade ou que outro caminhão está sendo carregado em outro local da cidade.

Percebemos que planejadores aceitos nas competições precisam ser sistemas que atuam independentemente dos domínios, e como esta importante característica de paralelismo acontece apenas em determinados e poucos domínios, acontece da maioria dos sistemas não estar preparada para tratar ou tirar vantagens sobre isto.

2.1.1.5.Exemplos de representação de Domínios

A Figura 2 ilustra o cenário do tradicional domínio do mundo de blocos descrito em PDDL. Destacamos algumas características interessantes tais como o fato de que as

declarativas da notação de pré-condições, efeitos, expressões lógicas e objetos nomeados dentro do modelo, que correspondem diretamente aos objetos nomeados no domínio, tornam possível o raciocínio sobre a notação.

```

////////////////////////////////////
;;; 4 Op-blocks world
////////////////////////////////////

(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x))

  (:action pick-up
    :parameters (?x)
    :precondition (and (block ?x) (clear ?x) (ontable ?x) (handempty))
    :effect
    (and (not (ontable ?x))
         (not (clear ?x))
         (not (handempty))
         (holding ?x)))

  (:action put-down
    :parameters (?x)
    :precondition (and (block ?x) (holding ?x))
    :effect
    (and (not (holding ?x))
         (clear ?x)
         (handempty)
         (ontable ?x)))

  (:action stack
    :parameters (?x ?y)
    :precondition (and (block ?x) (block ?y) (holding ?x) (clear ?y))
    :effect
    (and (not (holding ?x))
         (not (clear ?y))
         (clear ?x)
         (handempty)
         (on ?x ?y)))

  (:action unstack
    :parameters (?x ?y)
    :precondition (and (block ?x) (block ?y) (on ?x ?y) (clear ?x)
                      (handempty))
    :effect
    (and (holding ?x)
         (clear ?y)
         (not (clear ?x))
         (not (handempty))
         (not (on ?x ?y)))))

```

Figura 2. Arquivo em PDDL para descrição de domínio Mundo de Blocos.

O campo `:predicates` consiste de uma lista de declarações de predicados. Para cada predicado, é especificada uma lista de variáveis (talvez tipadas) para declarar a sua ocorrência, e talvez também o tipo dos seus argumentos.

A linguagem PDDL possui características para estruturar hierarquicamente as ações, mas não possui características suficientes para fornecer estruturas para objetos ou

estados. Além disso, a linguagem precisa de estruturas para configurar os critérios de consistência interna como completude ou validação dos estados do mundo ou ações.

A seguir, apresentamos a representação dos problemas de planejamento aplicados nestes domínios, representados na linguagem PDDL.

2.1.2. Representação de Problemas

Achar um plano é o problema que um sistema planejador tenta resolver. É definido com respeito a um domínio e especifica uma situação inicial e um objetivo a ser alcançado.

A definição de uma situação inicial deve especificar uma lista de fórmulas atômicas inicialmente verdadeiras. As fórmulas atômicas negativas são assumidas como falsas pela suposição do mundo fechado.

A figura abaixo ilustra a definição do cenário de um problema aplicado ao domínio do mundo de blocos, representado na linguagem PDDL, onde existem quatro blocos inicialmente desempilhados e o objetivo é encontrar um plano de ações que alcance o objetivo de empilhar os blocos em determinada ordem:

```
(define (problem BLOCKS-4-0)
(:domain BLOCKS)
(:objects D B A C)
(:init (BLOCK D) (BLOCK B) (BLOCK A) (BLOCK C) (CLEAR C) (CLEAR A) (CLEAR B)
(CLEAR D) (ONTABLE C) (ONTABLE A)
(ONTABLE B) (ONTABLE D) (HANDEEMPTY))
(:goal (AND (ON D C) (ON C B) (ON B A))))
```

Figura 3. Arquivo em PDDL com o problema aplicado ao domínio Mundo de Blocos.

O campo `:objects` é requerido e lista os objetos que existem no problema, os quais poderiam ser um super conjunto daqueles que aparecem na situação inicial. Se o tipo estiver definido, isto será uma lista de objetos tipados, caso contrário são representações simbólicas dos objetos sem obedecer a um tipo específico.

O campo `:goal` de uma definição de problema é uma fórmula. Uma solução para um problema é uma série de ações tais que a sua seqüência seja possível de iniciar, a partir de uma dada situação inicial, e que a fórmula representada em `:goal` seja verdadeira após a execução desta seqüência de ações.

Na próxima seção, passaremos a acompanhar melhor a evolução da área caracterizada pelas competições de planejadores.

2.2. PLANEJADORES E COMPETIÇÕES

Por cerca de mais de 20 anos o problema de planejamento foi tratado basicamente por estas abordagens: procedimentos de buscas baseados em *STRIPS* e procedimentos de provas de teoremas, ambas ineficientes em termos práticos, caindo em buscas exaustivas e explosões combinatórias, uma vez que a complexidade do problema de planejamento em *STRIPS* é *PSPACE-Completo* [14],[15], isto é, requer um espaço de busca polinomial.

Com o surgimento de métodos rápidos para tratamento de problemas de satisfabilidade (SAT) [45], [46], a abordagem baseada em prova de teoremas foi retomada. Em 1992, Kautz e Selman apresentaram uma tradução da representação *STRIPS* para cálculo proposicional, resultando em um planejador extremamente rápido, o *Satplan* [25],[26]. Este método aplica os recentes algoritmos para satisfabilidade, que utilizam resolvidores e são extremamente eficientes. Possuem a grande vantagem de permitir a substituição de um resolvidor por outro melhor adaptado ou de desempenho melhor.

Mas a grande revolução veio três anos depois quando Blum e Furst apresentaram o *Graphplan* [11]. Eles inovam ao mostrar que, a partir de uma descrição *STRIPS*, pode-se construir um grafo que reduz sensivelmente o espaço de busca para o problema. Neste grafo reduzido, mesmo uma busca exaustiva pode ser suficiente para encontrar uma solução.

Recentemente, a técnica de *PI* (Programação Inteira) também se mostrou promissora na solução de problemas de planejamento, através de sua modelagem como um conjunto de restrições sobre variáveis inteiras. Bockmayr e Dimopoulos [8] usaram variáveis inteiras 0-1 de modo similar à abordagem SAT e examinaram o efeito de adicionar restrições redundantes ao problema de *PI*. Vossen e colegas [53], [54], discutem a importância de encontrar a representação correta do problema de planejamento em termos de um problema de *PI*. Eles mostraram várias formulações possíveis e suas vantagens. Kautz e Walser [27] tratam da solução dos problemas de *PI* resultantes usando algoritmos de busca local inteira.

Apesar dos bons resultados alcançados na aplicação de *PI*, esta área ainda está aberta para novas investigações, explorando novas possibilidades do uso de métodos para *PI* na solução de problemas de planejamento em IA.

A questão de tratar planejamento como sistemas de restrições também foi considerada por Van Beek e Chen [6], que propõem tratar ambas as abordagens *SAT* e *PI* como um caso particular de outra técnica básica de IA: a técnica de Satisfação de Restrições – *CSP* (*Constraint Satisfaction Problem*). Eles mostraram como a sua abordagem é melhor em termos de tempo de processamento e utilização de memória. Infelizmente, o algoritmo não é aceito pela comunidade, pois utiliza restrições dependentes do domínio e em geral, existe um consenso que os planejadores não podem ser dependentes de domínio.

Outra contribuição recente apresenta-se no trabalho de Silva [49], que mostra uma forma de definir o problema de planejamento como um problema de alcançabilidade em redes de Petri [38], integrando o planejamento baseado em *STRIPS*, *Graphplan*, redes de *Petri* e *PI*, abrindo caminho para uma melhor compreensão de como todos estes temas se relacionam.

A partir do *Satplan* e do *Graphplan*, houve uma grande motivação na comunidade por novas pesquisas em planejamento, até então estagnadas, como nos mostra Weld [55] em sua excelente revisão do estado da arte. A comunidade sentiu a necessidade de acompanhar a evolução destes conhecimentos nas Conferências bianuais internacionais sobre Planejamento e Escalonamento em Inteligência Artificial, AIPS, que reúnem os principais avanços da área.

Até a quarta conferência AIPS, que aconteceu em 1998, nunca havia ocorrido uma competição no campo de planejamento automatizado. A mais ampla definição de planejamento é o raciocínio sobre o comportamento de agentes e como existem muitos tipos de agentes, técnicas de raciocínio, e maneiras de combinar inferências sobre planos e execuções de planos, o campo é muito amplo englobando tudo desde escalonamento de fábrica até programação de robôs. Algumas áreas de aplicação são de interesse prático imediato, enquanto outras ainda são muito abstratas.

Muitos subcampos de Inteligência Artificial têm usado as competições como forma de medir o progresso e as direções das pesquisas. Numa competição, os pesquisadores executam seus programas sobre um conjunto de problemas comuns, ao mesmo tempo, com pequenos ajustes, e os resultados são comparados. Existem muitos propósitos para este exercício: permite uma comparação significativa entre programas, pode fornecer uma

indicação do progresso geral do campo e fornece um conjunto de problemas de testes e medições para que outros possam usar e comparar seus próprios sistemas com o estado da arte.

Um dos principais objetivos destas competições é fornecer um fórum para comparações empíricas entre sistemas de planejamento, realçar os desafios para a comunidade na forma de problemas que testam o limite das capacidades atuais, propor novas direções para as pesquisas e fornecer um núcleo de avaliações de problemas comuns e um formalismo de representação que pode auxiliar na comparação e evolução dos sistemas de planejamento.

Apesar da série possuir um estilo de competitividade (sistemas individuais são identificados pelo desempenho excepcional no próprio evento), o foco está na coleção de dados e nas apresentações, com as interpretações dos resultados sendo abrandada. O objetivo real da competição é tornar disponível para a comunidade a maior quantidade de dados possível sobre as técnicas aplicadas e os resultados obtidos.

Um exemplo disto é a idéia de construir um grafo para reduzir o espaço de busca, que foi bem aproveitada por Kautz e Selman, que mostraram que o grafo pode ser traduzido para uma instância *SAT* muito menor que a gerada pelo *Satplan*. O algoritmo resultante, *Blackbox* [28], obteve grande destaque na primeira competição de planejadores, como pode ser visto abaixo. Outro competidor de destaque, o *HSP* (*Heuristic Search Planning*) de Bonet e Geffner [9], [21], também resgata antigas técnicas de IA no contexto do planejamento, como por exemplo o uso de funções heurísticas sobre o espaço das buscas. Por estas características, é que o estado da arte em planejamento para Inteligência Artificial pode ser mais bem acompanhado pelos avanços conquistados pelos competidores das Conferências AIPS.

O mesmo grupo que criou a linguagem PDDL, liderado por Drew McDermott definiu com os competidores da AIPS-98 duas trilhas de domínios em STRIPS e em ADL, nenhuma das quais capazes de manipular problemas com números. A trilha em ADL, por tratar de domínios tipados, está fora do escopo deste trabalho.

Inscreveram-se oito participantes para esta competição, mas apenas cinco terminaram suas adaptações para o uso da linguagem PDDL. Estes estão caracterizados na Tabela 1 a seguir:

Planejador	Trilha	Equipe
BLACKBOX	Strips	Henry Kautz e Bart Selman da AT&T Labs e Universidade de Cornell – Estados Unidos [28]
HSP	Strips	Héctor Geffner e Blai Bonet da Universidade Simon Bolívar – Venezuela [9]
IPP	Strips / ADL	Jana Kohler da Universidade de Freiburg – Alemanha [24]
STAN	Strips	Derek Long e Maria Fox da Universidade de Durham – Inglaterra [31]
SGP	ADL	Corin Anderson da Universidade de Washington – Estados Unidos [47]

Tabela 1. Participantes da competição AIPS – 1998 [37].

Todos os planejadores foram sistemas baseados em GRAPHPLAN [12] e foram escritos em C/C++ exceto o HSP utilizou uma abordagem baseada em busca heurística e o SGP que foi escrito em Lisp.

A competição foi dividida em duas etapas sendo que na primeira, um total de 88 problemas que foram resolvidos por pelo menos um planejador. O HSP resolveu mais problemas que todos os outros sistemas e encontrou a menor solução com maior frequência. O BLACKBOX obteve o menor tempo médio nos problemas que tentou, mas o IPP obteve o menor tempo de solução num maior número de problemas. O STAN ficou em segundo lugar no tempo de solução e em segundo lugar geral.

Planejador	Tempo Médio (ms)	Problemas Resolvidos	Mais Rápido	Menor Plano
BLACKBOX	1498	63	16	55
HSP	35483	82	19	61
IPP	7408	63	29	49
STAN	55413	64	24	47

Tabela 2. Resultados da 1ª. Etapa da competição de 1998 [37].

Na segunda etapa, um total de 12 problemas que foram resolvidos por pelo menos um planejador e os resultados foram os seguintes:

Planejador	Tempo Médio (ms)	Problemas Resolvidos	Mais Rápido	Menor Plano
BLACKBOX	2464	8	3	6
HSP	25875	9	1	5
IPP	17375	11	3	8
STAN	1334	7	5	4

Tabela 3. Resultados da 2ª. Etapa da competição de 1998 [37].

A quinta Conferência AIPS, organizada por Fahiem Bacchus [1] em 2000 continuou o trabalho da Conferência de 98 com a segunda competição internacional que reuniu 15 competidores e foi ampliada para incluir sistemas de planejamento completamente automatizados e também aqueles com controles manuais para receber informações específicas sobre um determinado domínio. Ambas as trilhas, STRIPS e ADL, foram usadas, mas nenhuma extensão foi feita à linguagem. Os competidores da AIPS-00 foram:

	Planejador	Equipe	Universidade	País
1	FF	Joerg Hoffmann	Albert Ludwigs	Alemanha
2	GRT	Ioannis Refanidis, Ioannis Vlahavas e Dimitris Vraskas	Aristotle	Grécia
3	SYSTEM R	Fangzhen Lin	Hong Kong Science and Technology	China
4	MIPS	Stefan Edelkamp e Malte Helmert	Freiburg	Alemanha
5	IPP	Jana Koehler, Joerg Hoffmann e Michael Brener	(Schindler Lifts) Freiburg	Suíça Alemanha
6	HSP2	Hector Geffner e Blai Bonet	Simon Bolivar	Venezuela
7	PropPlan	Michael Fourman	Edinburgh	Inglaterra
8	STAN	Derek Long e Maria Fox	Durham	Inglaterra
9	BLACKBOX	Henry Kautz, Bart Selman e Yi-Cheng Huang	Whashington, Cornell	USA
10	ALTALT	Biplav Srivastava, Terry Zimmerman, BinhMinh Do, XuanLong Nguyen, Zaiqing Nie, Ullas Nambiar e Romeo Sanchez	Arizona State	USA
11	TOKEN PLAN	Yannick Meiller e Patrick Fabiani	ONERA – Center of Toulouse	França
12	BDDPLAN	Hans-Peter Störr	Dresden University of Technology	Alemanha
13	SHOP	Dana Nau, Hector Munoz-Avila, Yue (Jason) Cao, e Amnon Lotem	Maryland	USA
14	PBR	Jose Luis Ambite, Craig Knoblock e Steve Minton	Southern California Information Sciences Institute	USA
15	TAL PLANNER	Jonas Kvarnstrom, Patrick Doherty, e Patrik Haslum	Linkoping	Suécia

Tabela 4. Competidores da AIPS – 2000 [3].

As principais técnicas utilizadas por estes planejadores podem ser resumidas na Tabela 5 onde as colunas numeradas correspondem aos planejadores enumerados na Tabela 4:

Técnica Aplicada	Planejadores														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Buscas guiadas Heuristicamente	X	X				X		X		X					
Diagrama de Decisão Binária (BDD)				X			X					X			
Regressão – progressão			X												
Baseados em GRAPHPLAN					X			X		X					
Baseados em SAT									X						
Redes Petri											X				
Rede de Tarefas Hierárquicas (HTN)													X		
Reescrita de Plano														X	
Controle de Busca por Lógica Temporal															X

Tabela 5. Técnicas aplicadas pelos competidores da AIPS – 2000 [3].

No domínio de Mundo de Blocos, os testes provaram ser de 14 ou 15 blocos o limite para um desempenho aceitável, entretanto os planejadores System R, FF e HSP2 conseguiram bom desempenho nos testes com até 50 blocos. O HSP2 gera planos muito longos algumas vezes, o FF [23] gera os menores planos e o System R [30] faz planos de bons tamanhos em tempos razoáveis, sendo capaz de gerar planos de quase 200 passos em cerca de 900 segundos.

Os planejadores controlados manualmente com informações específicas do domínio conseguiram um desempenho muito bom no Mundo de Blocos. Todos os planejadores geraram planos de tamanhos similares, como nos problemas para 500 blocos e tamanho aproximado de 2000 passos. O TALPLANNER foi capaz de resolver estes grandes problemas em 1,5 segundos.

No domínio de Logística, o FF e o MIPS foram os mais rápidos, mas o STAN gerou os menores e mais consistentes planos. Neste domínio, os planos contendo mais de 200 passos foram gerados em cerca de 100 segundos. Os planejadores manipulados com informações do domínio geraram planos de tamanhos similares, mas o TALPLANNER teve um diferencial significativo em tempo de planejamento.

O domínio FreeCell demonstrou ser o mais desafiador dos domínios apresentados onde curiosamente, os planejadores manipulados não conseguiram uma diferença muito significativa em relação aos planejadores automáticos. STAN, FF e MIPS foram os automáticos de melhor desempenho em termos de tempo, mas o STAN gerou planos muito extensos e o MIPS gerou os menores planos. Novamente, o TALPLANNER destacou-se entre os planejadores manipulados com informações do domínio, sendo o

mais rápido entre eles, porém, sem muita diferença do planejador automático mais rápido e ainda gerou planos muito longos, na ordem de 5000 passos.

Para o domínio de Elevadores, todos os problemas foram muito fáceis. A dificuldade estava na representação de todas as características deste domínio. A versão mais complexa incluiu uma restrição de não carregar mais do que seis passageiros ao mesmo tempo no elevador. Apenas dois planejadores foram capazes de gerar planos corretos para este domínio: TALPLANNER e PROPPLAN.

Foram concedidos dois prêmios separados, com cortesias da Cel Corporation (CelCorp) que direcionou um prêmio em dinheiro para o melhor desempenho do planejador FF e da empresa Schindler Lifts Ltd. que ofereceu um prêmio para o melhor desempenho no domínio de Elevadores para o planejador TALPLANNER.

Receberam ainda uma Menção de Honra pelo bom desempenho apresentado os planejadores STAN, HSP2, MIPS, e SYSTEM R. Na categoria de desempenho excepcional foram classificados os planejadores FF e TALPLANNER.

Em 2002, a competição foi organizada por Derek Long e Maria Fox [2]. O evento atraiu 14 competidores e focou em planejamento sobre domínios temporais e métricos. PDDL não foi expressivo o suficiente para permitir a modelagem de ações duráveis e consumo contínuo de recursos, então foi estendida para atender a estas características. A linguagem resultante da competição, PDDL+, consistiu em cinco níveis de potência expressiva, dos quais os três primeiros, em conjunto com a linguagem PDDL 2.1, foram usados na competição. O nível 1 foi o planejamento ADL como os anteriores, nível 2 adicionou variáveis numéricas e o nível 3 adicionou construções duráveis.

A competição apresentou desafios novos e importantes para a área de planejamento em Inteligência Artificial, e obteve como resultado principal a constatação de que existem atualmente diversos sistemas planejadores capazes de tratar de problemas muito complexos e com características numéricas e temporais.

Os domínios aplicados em 2002 estavam divididos em três grandes grupos:

- 1) Domínios relacionados a problemas de Transporte – Depots, DriverLog e ZenoTravel;
- 2) Domínios relacionados a aplicações Espaciais – Satellite e Rovers; e
- 3) Domínios relacionados a Outras aplicações – FreeCell, Settlers e UMTranslog-2.

Para os dois primeiros grupos foram ainda apresentadas variações sobre o domínio, que incluíam o uso de problemas em STRIPS, numéricos, temporizados-simples,

temporizados e, em alguns casos, problemas mais complexos que geralmente, combinavam tempo e números. Estas variações foram suficientemente grandes para afetar a natureza dos problemas originais, sendo consideradas como novos domínios separados dos originais.

Houve um julgamento qualitativo baseado na cobertura, isto é, na quantidade de problemas que foram tentados, a taxa de planos de sucesso nestes problemas e a qualidade da solução gerada. Também foi considerada a velocidade dos planejadores. Foram favorecidas as combinações altas de coberturas e altas taxas de sucesso, qualitativamente ponderados nos limites entre alta cobertura e taxa moderada e boa cobertura com altas taxas de sucesso.

A Tabela 6 abaixo resume os resultados apresentados pelos competidores de 2002, nestes aspectos da avaliação:

	Planejador	Problemas Resolvidos	Problemas Tentados	Taxa (%) Sucesso	Capacidades
1	FF	237 (+70)	284 (+76)	83 (85)	Strips, Numeric, HardNumeric
2	LPG	372	428	87	Strips, Numeric, HardNumeric, SimpleTime, Time
3	MIPS	331	508	65	Strips, Numeric, HardNumeric, SimpleTime, Time, Complex
4	SHOP2	899	904	99	Strips, Numeric, HardNumeric, SimpleTime, Time, Complex
5	Sapa	80	122	66	Time, Complex
6	SemSyn	11	144	8	Strips, Numeric
7	Simplanner	91	122	75	Strips
8	Stella	50	102	49	Strips
9	TALPlanner	610	610	100	Strips, SimpleTime, Time
10	TLPlan	894	894	100	Strips, Numeric, HardNumeric, SimpleTime, Time, Complex
11	TP4	26	204	13	Numeric, SimpleTime, Time, Complex
12	TPSYS	14	120	12	SimpleTime, Time
13	VHPOP	122	244	50	Strips, SimpleTime
15	IxTeT	9 *	*	*	*

Tabela 6. Competidores da AIPS – 2002 [2].

O sistema FF tentou 76 problemas adicionais direcionados aos planejadores de entradas manuais e resolveu 70 deles com sucesso. O planejador IxTeT resolveu 9 problemas com planos aceitos pelo validador e, posteriormente, tentou outros 10 problemas, produzindo planos que não puderam ser validados, além de necessitar de recodificação manual para atender às instâncias PDDL dos problemas e domínios.

Na categoria “*desempenho diferenciado de primeira ordem*”, foram identificados os sistemas *LPG*⁶, completamente automatizado e *TLPlan*⁷, codificado manualmente. Na categoria “*desempenho distinto*”, o sistema automatizado *MIPS*⁸ e o codificado *SHOP2*⁹

⁶ LPG, de Alfonso Gerevini, Ivan Serina e equipe, Universidade de Brescia, Itália. (www.dur.ac.uk/d.p.long/IPC/LPG.html)

⁷ TLPlan, de Fahiem Bacchus e Michael Ady, Universidade de Toronto, Canadá. (www.dur.ac.uk/d.p.long/IPC/TLPlan.html)

⁸ MIPS, de Stefan Edelkamp, Universidade de Freiburg, Alemanha. (www.dur.ac.uk/d.p.long/IPC/MIPS.html)

⁹ SHOP2, de Dana Nau e equipe, Universidade de Maryland, Estados Unidos. (www.dur.ac.uk/d.p.long/IPC/SHOP.html)

receberam a distinção. O sistema de destaque na categoria “*recém-chegados*” foi o *VHPOP*¹⁰.

Houve ainda o destaque para os planejadores *FF* e *TALPlanner* que não participaram em todos os domínios e, com isto, obtiveram uma queda na avaliação geral, mas foram de desempenho excepcional nos domínios em que participaram.

A quarta competição AIPS, IPC-4 [3], acontecerá em 2004 e será construída sobre os esforços anteriores, em particular, sobre a linguagem PDDL 2.1 e suas extensões. O evento deverá ser ampliado e revisado em vários aspectos e abordagens. O aspecto mais importante será a separação das partes da competição que focalizam diferentes formalismos de planejamento: além da continuação das clássicas competições anteriores, existirá uma parte para planejamento probabilístico e, talvez, uma parte para planejamento não-determinístico. Nestas duas últimas partes, o objetivo principal do evento será a introdução de uma linguagem de representação comum para os respectivos campos, e para estabelecer os primeiros testes de desempenho. A parte probabilística será organizada por Michael Littman, e ainda não existe a designação do organizador da parte não-determinística. A parte clássica será organizada por Stefan Edelkamp e Jörg Hoffmann.

Na prática, as competições auxiliam e motivam a ampliação do conhecimento. Novas técnicas são apresentadas e resultados surpreendentes marcam o notável desenvolvimento da área. Graças a estes avanços, é que o Planejamento em Inteligência Artificial pode ser aplicado com grande destaque na automatização de controles robotizados, nos transportes e logística, montagem automática, escalonamento distribuído, gestão de fluxos de trabalho e recursos das empresas, missões militares aéreas, navais e espaciais, jogos de robôs, entre outros.

Mesmo apesar de todo este avanço na área de Planejamento em Inteligência Artificial, muito ainda resta a ser feito, sobretudo em cenários com grande explosão combinatorial como o cubo de Rubik¹¹ (também conhecido como cubo mágico) ou problemas como Sokoban¹², conforme ilustração da Figura 4. Nestes cenários, as atuais técnicas de planejamento em Inteligência Artificial ainda precisam ser melhoradas.

¹⁰ VHPOP, de Håkan Younes, Univ. Carnegie Mellon, Estados Unidos. (www.dur.ac.uk/d.p.long/IPC/VHPOP.html)

¹¹ O Cubo de Rubik, nasceu em 1974 em Budapest, capital da Hungria. Seu idealizador e criador foi Erno Rubik, professor de design de interiores da Academia de artes e trabalhos manuais de Budapest. O objetivo do cenário é, através de movimentos de rotação sob um eixo central, obter todas as faces do cubo com cores homogêneas.

¹² O Sokoban é um jogo para um único agente, inventado no Japão em 1982. Consiste num armazém fechado pelo qual existem espalhados vários caixotes. O objetivo do agente é arrumar esses caixotes num conjunto de posições pré-determinadas com círculos dentro do armazém. Nada indica no início em qual posição-objetivo deve ficar cada caixote.

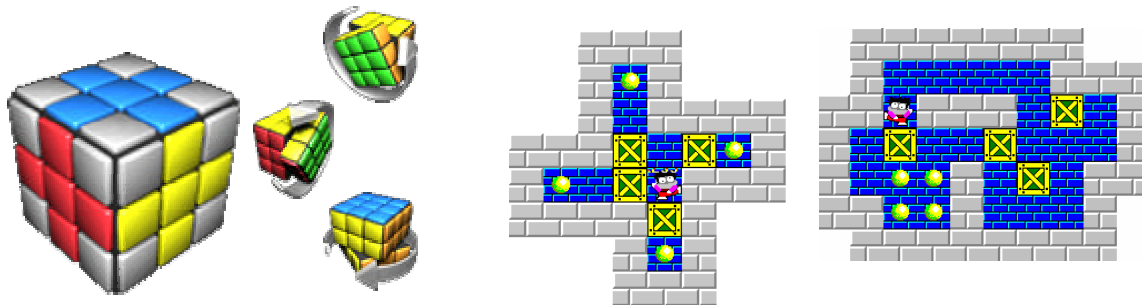


Figura 4. Cenários de Cubo de Rubik (Cubo Mágico) e Sokoban

Uma técnica ainda pouco abordada em Planejamento é a Computação Evolutiva, em especial os Algoritmos Genéticos, sendo que esta abordagem é a principal contribuição deste trabalho de pesquisa e mesmo apesar de envolver a criação de um planejador genético, o objetivo não é de participar deste tipo de competições, pelo menos nesta fase de estudos, onde os esforços estarão aplicados no aprendizado das técnicas de Algoritmos Genéticos direcionados para a resolução de problemas de planejamento em Inteligência Artificial.

Uma das principais características dos Algoritmos Genéticos, conforme seu detalhamento no próximo capítulo, é a capacidade de busca em espaços muito grandes, onde apesar de não poder garantir a solução ótima, possui mecanismos que possibilitam escapar de mínimos locais e tendem a evoluir as soluções parciais para a busca da melhor solução. No próximo capítulo analisaremos esta classe de algoritmos e no seguinte, veremos como aplicá-las aos problemas de planejamento.

3. ALGORITMOS GENÉTICOS

IA atualmente engloba uma grande variedade de subcampos, de áreas de propósito geral, como a percepção e o raciocínio lógico, para tarefas específicas como jogar xadrez, provar teoremas matemáticos, diagnosticar doenças. Um dos mais recentes subcampos da IA, é o de Algoritmos Genéticos ou Computação Evolutiva, mas que aproveita conceitos bastante conhecidos sobre a evolução de seres vivos. Até meados do século XIX, os naturalistas acreditavam que, entre os seres vivos, cada espécie havia sido criada separadamente, por um ser supremo ou através de geração espontânea. O trabalho do naturalista Carolus Linnaeus sobre a classificação biológica de organismos despertou o interesse pela similaridade entre certas espécies, levando a acreditar na existência de uma certa relação entre elas. Outros trabalhos influenciaram os naturalistas em direção à teoria da seleção natural, tais como os de Jean Baptiste Lamarck, que sugeriu uma teoria evolucionária no “uso e desuso” de órgãos, e de Thomas Robert Malthus, que propôs que fatores ambientais tais como doenças e carência de alimentos, limitavam o crescimento da população. Tais teorias fundamentam a técnica de Computação Evolutiva.

3.1. HISTÓRICO

Depois de mais de 20 anos de observações e experimentos, Charles Darwin apresentou em 1858 sua teoria da evolução através da seleção natural, simultaneamente com outro naturalista inglês Alfred Russel Wallace. No ano seguinte, Darwin publica o seu “*On the Origin of Species by Means of Natural Selection*”, com a sua teoria completa, sustentada por muitas evidências colhidas durante suas viagens a bordo do Beagle. Este trabalho influenciou muito o futuro, não apenas da Biologia, Botânica e Zoologia, mas também, foi de grande influência sobre o pensamento religioso, filosófico, político e econômico da época. A teoria da evolução e a computação nasceram, praticamente, na mesma época: Charles Babbage, um dos fundadores da computação moderna e amigo pessoal de Darwin desenvolveu sua máquina analítica em 1833. Ambos, provavelmente, estariam surpresos e orgulhosos com a ligação entre estas duas áreas.

Por volta de 1900, o trabalho de Gregor Mendel, desenvolvido em 1865, sobre os princípios básicos de herança genética, foi redescoberto pelos cientistas e teve grande influência sobre os futuros trabalhos relacionados à evolução. A moderna teoria da

evolução combina a genética e as idéias de Darwin e Wallace sobre a seleção natural, criando o princípio básico de Genética Populacional: a variabilidade entre indivíduos em uma população de organismos que se reproduzem sexualmente é produzida pela mutação e pela recombinação genética.

Este princípio foi desenvolvido durante os anos 30 e 40, por biólogos e matemáticos de importantes centros de pesquisa. Nos anos 50 e 60, muitos biólogos começaram a desenvolver simulações computacionais de sistemas genéticos. Vários cientistas da computação estudaram sistemas evolucionários com a idéia de que a evolução poderia ser usada como uma ferramenta de otimização para problemas de engenharia. Os sistemas desenvolvidos pretendiam criar uma população de candidatos à solução para um dado problema. Box (1957), Friedman (1959) e Baricelli (1967) desenvolveram algoritmos inspirados na evolução para problemas de otimização e aprendizagem de máquina. Entretanto, seus trabalhos não possuíam qualquer tipo de atenção às estratégias de evolução, programação evolucionária e algoritmos genéticos atuais. Os algoritmos genéticos foram inventados por John Holland nos anos 60 e desenvolvidos por Holland e seus alunos na Universidade de Michigan em meados de 1970. O principal objetivo de Holland não foi desenvolver algoritmos para solucionar problemas específicos, mas dedicar-se ao estudo formal do fenômeno de evolução, como ocorre na natureza, e desenvolver maneiras de importá-lo aos sistemas de computação.

John Holland, em meados dos anos 70, acreditou que as peculiaridades da Evolução Natural poderiam ser implementadas de forma algorítmica a fim de alcançar uma versão computacional dos processos de evolução. Holland foi gradualmente refinando suas idéias e em 1975 publicou o seu livro "*Adaptation in Natural and Artificial Systems*", considerado hoje como a bíblia dos Algoritmos Genéticos.

Este algoritmo poderia solucionar qualquer problema que apresentasse as mesmas características da evolução. O sistema trabalhava com uma população de algumas cadeias de bits (0's e 1's) denominadas cromossomos¹³. Semelhante à natureza, o sistema evoluiu até o melhor cromossomo, para atender um problema específico, mesmo sem saber que tipo de problema estava sendo solucionado. A solução foi encontrada de um modo automático e não-supervisionado. As únicas informações dadas ao sistema foram os ajustes de cada cromossomo produzido por ele.

¹³ Cromossomo. Geneticamente, são pacotes de [DNA](#). O homem tem 46 cromossomos (22 pares mais os cromossomos sexuais X e Y). Todas as 100 trilhões de [células](#) humanas contêm um conjunto completo de cromossomos, menos as células sexuais, que só têm a metade, e as [hemácias](#), que não têm qualquer [cromossomo](#).

A capacidade de reprodução é o principal fator de evolução dos seres vivos. Por evolução entende-se, a adaptação de um indivíduo ao meio ambiente. Os Algoritmos Genéticos, ou AG's, são sistemas adaptativos por excelência, ou seja, adaptam-se a determinadas condições que resultam na solução de um problema específico. Os AG's foram inventados numa tentativa de adaptar os processos observados na evolução natural. Na adaptação por reprodução, incorre-se no fato dos seres vivos modificarem-se à medida que vão se reproduzindo (Darwinismo). A adaptação não ocorre de forma direta, ou seja, os indivíduos se modificam de acordo com as necessidades e essa modificação passa diretamente para os descendentes (Lamarckismo).

O princípio adotado pelos AG's é o processo de evolução por seleção natural, ou seja, os indivíduos mais adaptados ao meio ambiente têm mais chances de sobreviver. Evolução biológica não possui memória. Todavia ela sabe como produzir seres vivos que se adaptam melhor ao seu meio ambiente. Este conhecimento está armazenado no grupo genético do indivíduo, responsável pelas suas características. A próxima seção apresenta o AG com as características de processo de busca no espaço.

3.2. PROBLEMAS DE BUSCA

Os propósitos, a estrutura geral e os princípios de operação dos algoritmos evolutivos têm uma estrutura básica comum: realizam reprodução, impõem variações aleatórias, promovem competição e executam seleção de indivíduos de uma dada população. Sempre que estes quatro processos estiverem presentes, seja na natureza ou em uma simulação computacional, a evolução é o produto resultante. Em particular, ao tratarmos técnicas para a solução de problemas de otimização, vinculando o uso de computação evolutiva, o problema a ser resolvido faz o papel do ambiente, e cada indivíduo da população é associado a uma solução-candidata.

Sendo assim, um indivíduo estará mais adaptado ao ambiente, sempre que ele corresponder a uma solução mais eficaz para o problema. Com a evolução, espera-se a cada geração ir obtendo soluções candidatas mais e mais eficazes, embora não exista a garantia de se chegar à solução ótima ao final do processo evolutivo. Neste contexto, um algoritmo evolutivo exerce o papel de um processo poderoso de busca iterativa e em paralelo, adequada para o tratamento de problemas de otimização, caracterizados por uma explosão combinatória de possibilidades, ausência de diferenciabilidade do critério de otimalidade ou multimodalidade.

Na verdade, todo problema suficientemente complexo, a ponto de dificultar a produção de uma formulação matemática abrangente e o atendimento de requisitos básicos de tratabilidade por ferramentas convencionais, se transforma em um candidato para ser abordado a partir da computação evolutiva, já que a aplicação de técnicas de solução conhecidas, dedicadas e capazes de garantir a obtenção de uma solução ótima, não é possível nestes casos. Toda tarefa de busca e otimização possui vários componentes, entre eles:

- o espaço de busca, onde são consideradas todas as possibilidades de solução de um determinado problema;
- a função de avaliação (ou função de custo - *fitness*), uma maneira de avaliar os membros do espaço de busca.

Existem muitos métodos de busca e funções de avaliação. As técnicas de busca e otimização tradicionais iniciam-se com um único candidato que, iterativamente, é manipulado utilizando algumas heurísticas estáticas diretamente associadas ao problema a ser solucionado. Geralmente, estes processos heurísticos não podem ser representados diretamente de forma algorítmica e sua simulação em computadores pode ser muito complexa. Apesar destes métodos não serem suficientemente robustos, isto não implica que eles sejam inúteis. Na prática, eles são amplamente utilizados, com sucesso, em inúmeras aplicações. Por outro lado, as técnicas de computação evolucionária operam sobre uma população de candidatos em paralelo. Assim, elas podem fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões.

Os Algoritmos Genéticos (AGs) diferem dos métodos tradicionais de busca e otimização, principalmente em quatro aspectos:

1. AGs trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros.
2. AGs trabalham com uma população de soluções e não com uma solução única.
3. AGs utilizam informações de custo ou recompensa e não derivadas ou outro conhecimento auxiliar.
4. AGs utilizam regras de transição probabilísticas e não determinísticas.

Algoritmos Genéticos são muito eficientes para busca de soluções ótimas, ou aproximadamente ótimas em uma grande variedade de problemas, pois não impõem muitas das limitações encontradas nos métodos de busca tradicionais. Além de ser uma

estratégia de gerar-e-testar muito elegante, por serem baseados na evolução biológica, são capazes de identificar e explorar fatores ambientais e convergir para soluções ótimas, ou aproximadamente ótimas em níveis globais. "Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes": este é o conceito básico da evolução genética biológica. A área biológica mais proximamente ligada aos Algoritmos Genéticos é a Genética Populacional.

Os pesquisadores referem-se a "algoritmos genéticos" ou a "um algoritmo genético" e não "ao algoritmo genético", pois AGs são uma classe de procedimentos com muitos passos separados, e cada um destes passos possui muitas variações possíveis.

Na próxima seção, apresentamos uma análise resumida das características destes algoritmos. Alguns conceitos básicos são necessários e podem ser naturalmente expostos explicando o seu funcionamento básico, conforme a seção 3.4.

3.3. CARACTERÍSTICAS DO AG

As principais características dos Algoritmos Genéticos são:

- ? Algoritmos Genéticos são algoritmos de otimização global, baseados nos mecanismos de seleção natural e da genética que empregam uma estratégia de busca paralela e estruturada, mas aleatória, que é voltada em direção ao reforço da busca de pontos de "alta aptidão", ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos);
- ? Apesar de aleatórios, eles não configuram caminhadas aleatórias não direcionadas, pois exploram informações históricas para encontrar novos pontos de busca onde são esperados melhores desempenhos. Isto é feito através de processos iterativos, onde cada iteração é chamada de geração;
- ? Durante cada iteração, os princípios genéticos de seleção e reprodução são aplicados a uma população de candidatos que pode variar, dependendo da complexidade do problema e dos recursos computacionais disponíveis.

3.4. FUNCIONAMENTO

Inicialmente, é gerada uma população formada por um conjunto aleatório de indivíduos que podem ser vistos como possíveis soluções do problema. Durante o processo evolutivo, esta população é avaliada: cada indivíduo recebe uma nota, ou índice, refletindo sua habilidade de adaptação a determinado ambiente.

Uma porcentagem dos mais adaptados é mantida, enquanto os outros são descartados (*Darwinismo*). Os membros mantidos pela seleção podem sofrer modificações em suas características fundamentais através de mutações e cruzamento (*crossover*) ou recombinação genética gerando descendentes para a próxima geração. Este processo, chamado de reprodução, é repetido até que uma solução satisfatória seja encontrada.

Embora possam parecer simplistas do ponto de vista biológico, estes algoritmos são suficientemente complexos para fornecer mecanismos de busca adaptativa, poderosos e robustos.

Através da seleção, determinam-se quais indivíduos conseguirão se reproduzir, gerando um número determinado de descendentes para a próxima geração, com uma probabilidade determinada pelo seu índice de aptidão. Em outras palavras, os indivíduos com maior adaptação relativa têm maiores chances de se reproduzir.

O ponto de partida para a utilização de AG's, como ferramenta para solução de problemas, é a representação destes problemas de maneira que os AG's possam trabalhar adequadamente sobre eles. A maioria das representações é genotípica, utilizam vetores de tamanho finito em um alfabeto finito. Tradicionalmente, os indivíduos são representados genotipicamente por vetores binários, onde cada elemento de um vetor denota a presença (1) ou ausência (0) de uma determinada característica: o seu genótipo. Os elementos podem ser combinados formando as características reais do indivíduo, ou o seu fenótipo.

O AG repete os quatro passos seguintes até encontrar uma solução.

1. *Avaliação*: Cada elemento da população é classificado conforme o valor da função de avaliação. Decide se o melhor elemento pode ser aceito como uma solução; se sim, encerra.
2. *Seleção*: Usa a função de avaliação para definir a distribuição de probabilidade sobre a população. Seleciona um par de elementos aleatoriamente, de acordo com a distribuição de probabilidade.

3. *Reprodução*: Produz um novo elemento a partir de cada par usando os operadores genéticos.
4. *Substituição*: Substitui os elementos da população inicial pelos melhores novos elementos produzidos no passo três.

Teoricamente, esta representação é independente do problema, pois uma vez encontrada a representação em vetores binários, as operações-padrão podem ser utilizadas, facilitando o seu emprego em diferentes classes de problemas. A utilização de representações em níveis de abstração mais altos tem sido investigada. Como estas representações são mais fenotípicas, facilitariam sua utilização em determinados ambientes, onde essa transformação "fenótipo - genótipo" é muito complexa. Neste caso, precisam ser criados os operadores específicos para utilizar estas representações.

O princípio básico do funcionamento dos AGs é que um critério de seleção vai fazer com que, depois de muitas gerações, o conjunto inicial de indivíduos gere indivíduos mais aptos. A maioria dos métodos de seleção é projetada para escolher preferencialmente indivíduos com maiores notas de aptidão, embora não exclusivamente, a fim de manter a diversidade da população. A Figura 5, a seguir, representa o princípio básico do funcionamento de um Algoritmo Genético.

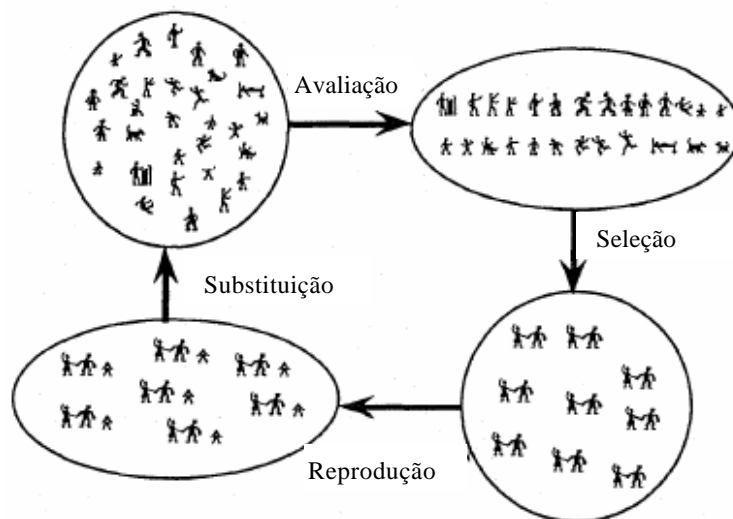


Figura 5. Princípio básico de um AG.

Um método de seleção muito utilizado é o Método da Roleta, onde indivíduos de uma geração são escolhidos para fazer parte da próxima, através de um sorteio de roleta. Neste método, cada indivíduo da população é representado na roleta, proporcionalmente

ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos de aptidão mais baixa é dada uma porção relativamente menor. Finalmente, é girada a roleta por um determinado número de vezes, dependendo do tamanho da população, e são escolhidos aqueles sorteados como indivíduos que participarão da próxima geração.

Um conjunto de operações é necessário para que, dada uma população, se consiga gerar populações sucessivas que (espera-se) melhorem sua aptidão com o tempo. Estes operadores são: cruzamento (*crossover*) e mutação. Eles são utilizados para assegurar que a nova geração seja totalmente nova, mas possui, de alguma forma, características de seus pais, ou seja, a população se diversifica e mantém características de adaptação adquiridas pelas gerações anteriores.

Para prevenir que os melhores indivíduos não desapareçam da população pela manipulação dos operadores genéticos, eles podem ser automaticamente colocados na próxima geração, através da reprodução elitista. Esse ciclo é repetido um determinado número de vezes. Abaixo é mostrado um exemplo de algoritmo genético.

```
Procedimento AG
{ t = 0;
  inicia_população (P, t);
  avaliação (P, t);
  repita até (t = d)
    { t = t + 1;
      seleção_dos_pais (P, t);
      recombinação (P, t);
      mutação (P, t);
      avaliação (P, t);
      sobrevivem (P, t);
    }
}
```

Figura 6. Pseudo-código para um AG.

onde:

t - tempo atual;

d - tempo determinado para finalizar o algoritmo;

P – população.

Durante este processo, os melhores indivíduos, assim como alguns dados estatísticos, podem ser coletados e armazenados para avaliação. Estes algoritmos, apesar de serem computacionalmente muito simples, são bastante poderosos. Além disso, eles não são limitados por suposições sobre o espaço de busca relativas à continuidade, existência de derivadas, etc. Buscas em problemas reais são repletas de descontinuidades, ruídos e outros problemas. Métodos que dependam fortemente de restrições de continuidade e existência de derivadas são adequados apenas para problemas sobre um domínio limitado.

Apresentamos na próxima seção a descrição e o funcionamento dos principais operadores genéticos convencionais que tornam o AG potencialmente adequado para o tratamento dos problemas de planejamento e finalizamos o capítulo com a seção 3.6 que apresenta uma coletânea de aplicações que já se utilizam de Algoritmos Genéticos.

3.5. OPERADORES GENÉTICOS

O princípio básico dos operadores genéticos é transformar a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório. Os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores.

O operador de mutação é necessário para a introdução e manutenção da diversidade genética da população, alterando arbitrariamente um ou mais componentes de uma estrutura escolhida, como é ilustrado na Figura 7, a seguir, fornecendo assim, meios para introdução de novos elementos na população.

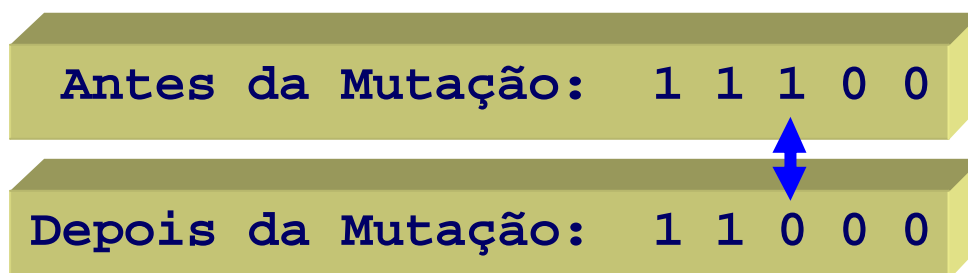


Figura 7. Operação Genética de Mutação.

Desta forma, a mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca nunca será zero, além de contornar o problema de mínimos locais, pois com este mecanismo, altera-se levemente a direção da busca. O operador de

mutação é aplicado aos indivíduos com uma probabilidade dada pela taxa de mutação P_m ; geralmente se utiliza uma taxa de mutação pequena, pois é um operador genético secundário.

O cruzamento é o operador responsável pela recombinação de características dos pais durante a reprodução, permitindo que as próximas gerações herdem essas características. Ele é considerado o operador genético predominante, por isso é aplicado com probabilidade dada pela taxa de *crossover*, que deve ser maior que a taxa de mutação.

Este operador pode, ainda, ser utilizado de várias maneiras; as mais comuns são:

- ? *um-ponto*: um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais serão trocadas. As informações anteriores a este ponto, em um dos pais, são ligadas às informações posteriores a este ponto no outro pai, como é mostrado no exemplo da Figura 8 abaixo;
- ? *multi-pontos*: é uma generalização desta idéia de troca de material genético através de pontos, onde muitos pontos de cruzamento podem ser utilizados;
- ? *uniforme*: não utiliza pontos de cruzamento, mas determina, através de um parâmetro global, qual a probabilidade de cada variável ser trocada entre os pais.

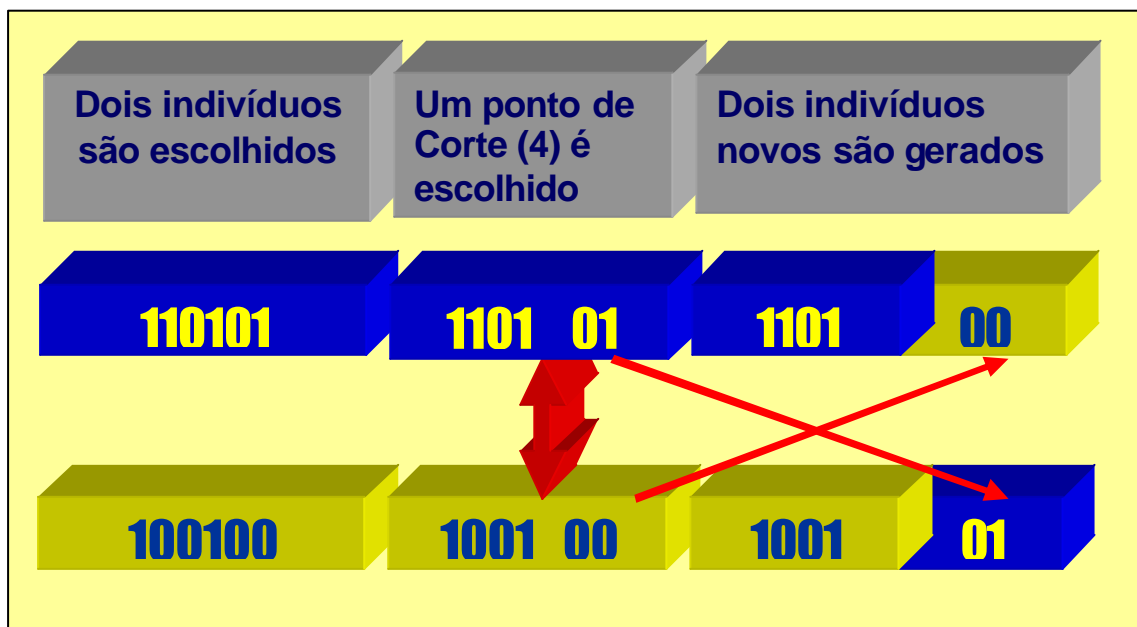


Figura 8. Operação Genética de Cruzamento (*crossover*).

É importante também, analisar de que maneira alguns parâmetros influem no comportamento dos Algoritmos Genéticos, para que se possa estabelecê-los conforme as necessidades do problema e dos recursos disponíveis:

- ? *Tamanho da População.* O tamanho da população afeta o desempenho global e a eficiência dos AGs. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.
- ? *Taxa de Cruzamento.* Quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, estruturas com boas aptidões poderão ser retiradas mais rapidamente. A maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.
- ? *Taxa de Mutação.* Uma baixa taxa de mutação previne que uma dada posição fique estagnada em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Com uma taxa muito alta a busca se torna essencialmente aleatória.
- ? *Intervalo de Geração.* Controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

3.6. APLICAÇÕES

A literatura existente sobre o assunto Algoritmos Genéticos está cada vez maior, principalmente pela divulgação ampla de estudos e pesquisas relacionadas ao assunto, na rede mundial de computadores, a Internet, dentre as quais, no estado da arte destacam-se:

- ✍ AG's têm sido usados em uma grande variedade de problemas de otimização, incluindo a de problemas numéricos e combinatoriais como o escalonamento de trabalho e o desenvolvimento de layout para circuitos eletrônicos.
- ✍ AG's usados para desenvolver programas de computador para tarefas específicas e outras estruturas computacionais, tais como, autômatos celulares e redes de ordenação estão no campo da Programação Automática.
- ✍ AG's são usados para muitas aplicações em aprendizagem de máquina, incluindo classificação e predição de tarefas, tais como: previsão do tempo ou estrutura de proteínas. Usados também para desenvolver aspectos particulares de sistemas de aprendizagem, tais como: pesos para redes neurais, regras de sistemas de aprendizagem classificadores ou sistemas de produção simbólicos, e sensores para robôs.
- ✍ AG's usados em Economia para modelar processos de inovação, o desenvolvimento de estratégias de lances, e na predição de mercados econômicos emergentes.
- ✍ AG's usados para modelar vários aspectos dos sistemas imunológicos naturais, inclusive a mutação somática durante a vida do indivíduo e o descobrimento de famílias de multi-genes durante o tempo evolucionário pertencem ao campo de Imuno-Sistemas.
- ✍ AG's usados em Ecologia para modelar fenômenos como competição entre nichos biológicos, co-evolução Parasito-hospedeira, simbiose, e fluxo de recursos.
- ✍ AG's usados para estudar questões da genética de populações, tais como: em quais condições um gene é viável, em termos evolucionários, para ser recombinado?
- ✍ AG's usados na Evolução e Aprendizagem para estudar como indivíduos aprendem, e a evolução das espécies (efeitos).
- ✍ AG's no desenvolvimento de aeronaves;
- ✍ Antecipação dinâmica da rota em redes de telecomunicações;
- ✍ AG's aplicados à geração de trajetórias para robôs;
- ✍ AG's que modelam sistemas dinâmicos não-lineares aplicados em modelos de segurança;

- ✍ AG's aplicados em sistemas de aquisição de estratégias;
- ✍ Utilização de AG's para diagnóstico de múltiplas falhas;
- ✍ AG's para a análise de DNA;
- ✍ AG's para o processamento de informações de sonares;
- ✍ Otimização econômica no gerenciamento de rios usando AG's;
- ✍ AG's aplicados ao problema do caixeiro viajante e ao escalonamento de seqüências.

As técnicas de computação evolucionária operam sobre uma população de candidatos em paralelo. Assim, elas podem fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões.

Há o sacrifício da generalidade para um ganho em desempenho, caracterizando o método de solução como forte. Perante um método forte de solução, a computação evolutiva não é competitiva e, portanto não deve ser considerada como alternativa. Mas a quantidade de problemas de interesse que hoje admitem uma solução a partir de métodos fortes é desprezível diante da quantidade de problemas para os quais não existe ainda uma solução dedicada. E é este o reino da computação evolutiva, tão vasto que as suas fronteiras se estendem além de qualquer horizonte que possa ser vislumbrado.

Como dissemos na introdução, existem várias razões para pensarmos que os algoritmos genéticos aplicam-se bem aos problemas de planejamento, embora não seja uma tarefa trivial elaborar e codificar o AG. No próximo capítulo, apresentamos uma abordagem genética ao problema de planejamento em Inteligência Artificial, que descreve esta problemática.

4. ABORDAGEM GENÉTICA NO PLANEJAMENTO

Neste capítulo, descrevemos uma forma de adaptar os Algoritmos Genéticos para o problema do planejamento em IA. A próxima seção apresenta a nossa proposta de modelagem do problema para o tratamento genético implementado e apresentado na seção 4.2 que traz a análise dos resultados e gráficos gerados pela implementação. Com base nas análises feitas sobre estes resultados e gráficos, apresentamos na seção 4.3 as motivações que nos levaram a propor mudanças estruturais importantes para o estudo do caso.

4.1. PROPOSTA DE MODELAGEM

No início da execução de um planejador os arquivos com a descrição do domínio e do problema [37] são lidos e uma estrutura de representação é construída na memória, contendo: o estado inicial, as constantes do problema e uma lista com todas as ações possíveis para o domínio. Sendo qualquer seqüência de ações desta lista um plano em potencial e, portanto, uma possível solução para o problema.

O AG faz uma busca por uma seqüência de ações da lista que resolva o problema de planejamento. Assim a codificação mais adequada é que cada gene represente uma ação do plano e o cromossomo represente um plano propriamente dito.

Cada ação instanciada da lista será representada por um número inteiro, assim, tem-se cada gene como um número inteiro entre um e o número de ações possíveis.

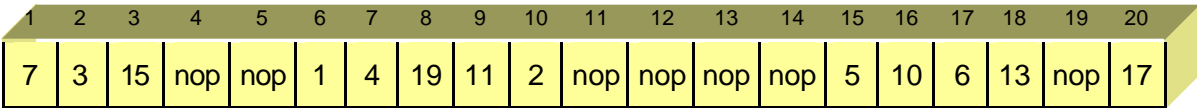
O problema de considerar um cromossomo (ou indivíduo do AG) como um plano e, portanto uma possível solução para o problema é que, previamente não se conhece o número de ações contidas no plano que resolvem o problema, e que as bibliotecas para AG, disponibilizadas pela comunidade de software livre, usam cromossomos de tamanho fixo.

A solução para esta situação inclui uma nova ação na lista de possíveis ações, chamada “NOP”, que seria aplicável a qualquer instante num plano e não teria efeitos sobre o estado do mundo. Com o seu uso, pode-se aumentar o número de ações de um plano indefinidamente, mantendo suas características originais. O problema de planos de tamanho variável é evitado pela definição de uma cota superior para o tamanho dos

planos e a inclusão de ações NOP's neles, até que todos tenham o mesmo tamanho, resultando num AG com cromossomos de tamanho fixo, onde um gene com o valor inteiro zero representa a ação NOP.

A cota superior para o tamanho do plano é determinada empiricamente, podendo ser incrementada, caso nenhuma solução seja encontrada. Esta abordagem também é utilizada em outros planejadores recentes [11], [28], [50] e causa dificuldades no tratamento do tamanho ideal para o plano, visto que podem existir muitas ações NOP entre as boas ações que formam um bom plano em potencial.

A codificação propõe o uso de cromossomos de números inteiros como genes. Este genótipo é mapeado para um fenótipo que representa uma sequência de ações através da remoção dos genes de valor zero e do uso dos seus valores como índices do vetor de ações possíveis do problema de planejamento. A Figura 9, abaixo, demonstra a representação utilizada para um cromossomo que simboliza um plano de até 20 ações. Os índices das posições do vetor indicam a ordem de execução das ações (genes), que aparecem numeradas com inteiros para indicar o número da ação ou "nop" quando a ação é nula.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
7	3	15	nop	nop	1	4	19	11	2	nop	nop	nop	nop	5	10	6	13	nop	17

Figura 9. Representação de um cromossomo para um plano de até 20 ações

Quanto à questão da geração da população inicial, consideramos para a codificação a geração aleatória e os valores dos genes escolhidos, aleatoriamente, entre zero e o número de ações possíveis para o problema.

Nesta primeira abordagem, a biblioteca genética pública PGAPack [29] foi utilizada para o controle genético do algoritmo desenvolvido em linguagem C para o sistema operacional Unix, enquanto um algoritmo desenvolvido em linguagem C++ para as competições [19] faz o controle do algoritmo de planejamento.

A fase de seleção de indivíduos para a população escolhe os cromossomos como "strings" com base nos seus valores de avaliação. Nesta fase estamos usando o método conhecido como Torneio Probabilístico [22], [29].

O operador convencional de Cruzamento estabelece um ponto de corte aleatório de cromossomos pais, gerando uma recombinação das partes em cromossomos filhos. Usamos o padrão de Cruzamento em dois pontos aleatórios, como mostra a Figura 10.

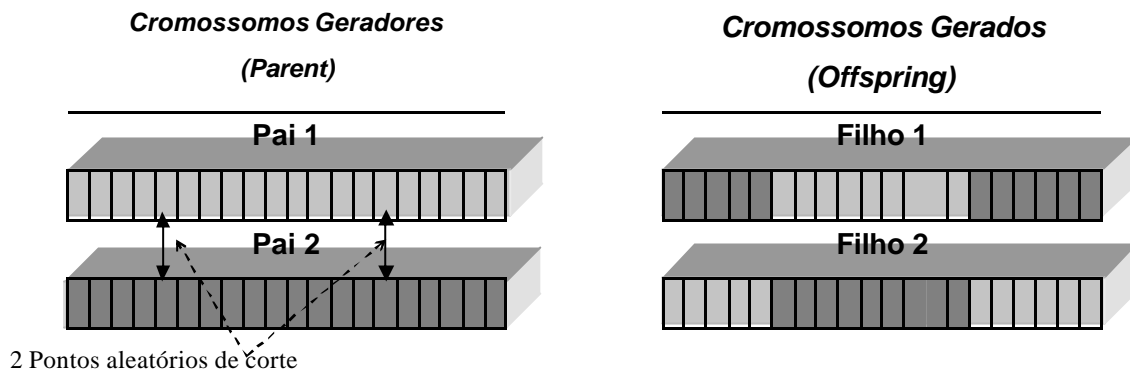


Figura 10. Operador de Cruzamento (crossover) de 2 pontos.

O parâmetro de probabilidade de Mutação determina a taxa percentual de troca de valor de um gene inteiro entre os limites determinados na inicialização, escolhendo randomicamente um valor para o gene selecionado uniformemente, como mostra a Figura 11, abaixo.

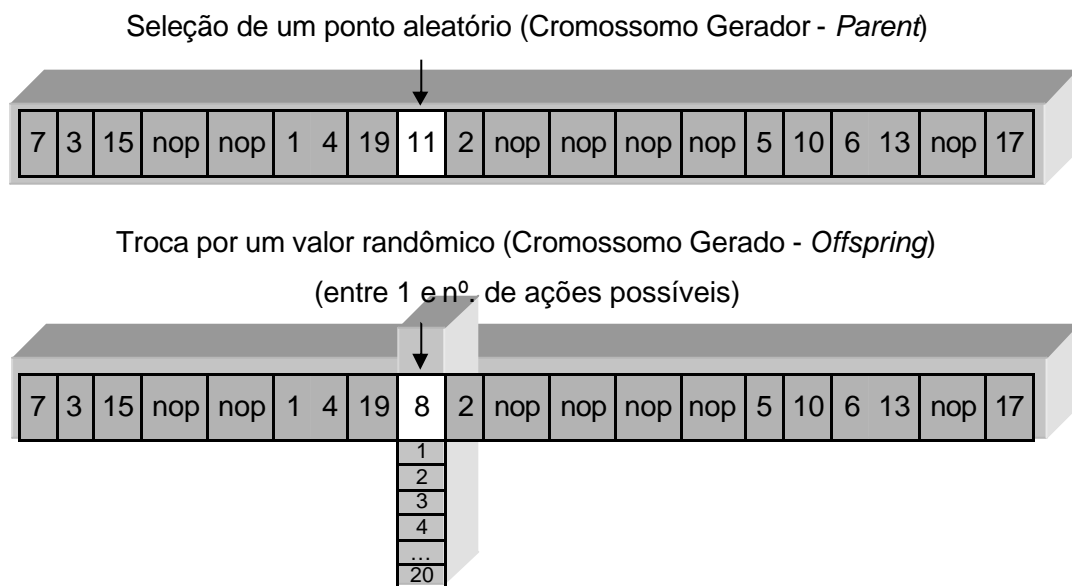


Figura 11. Operador aplicado de Mutação.

Por padrão, PGAPack aplica a Mutação apenas para os indivíduos que não foram utilizados no operador de Cruzamento. Alteramos este comportamento para que a Mutação seja aplicada sobre ambos os cromossomos selecionados para o Cruzamento, possibilitando um maior controle sobre os indivíduos nos quais age este operador.

Optamos por manter a possibilidade de indivíduos duplicados nas populações por considerarmos que devido ao amplo espaço de busca não perderíamos diversidade e também, optamos por alterar o comportamento padrão de um AG quanto à substituição da população, GRGA (*Generational Replacement Genetic Algorithm*) [29], para o padrão SSGA (*Steady-State Genetic Algorithm*) [29], que tipicamente troca apenas um percentual da população, no caso 90%, a cada geração, onde os 10% restantes de indivíduos melhores classificados no rank são simplesmente copiados para a nova geração.

Nos AG's, cada string é avaliada e recebe um valor real de retorno, chamado valor de avaliação (*fitness*), que é uma medida relativa ao resto da população, de quão bem este cromossomo satisfaz a métrica de um problema específico. Tradicionalmente o AG assume que valores de avaliação são positivos e monotonicamente crescentes na medida em que melhor satisfazem a métrica. A correta definição de um ou mais critérios para a avaliação desta métrica está diretamente relacionada com o sucesso do AG. Estes valores são, então, mapeados internamente para um valor de avaliação que usa o método de Normalização Linear [22],[29] dada por $K - (rank(p) * C)$ onde K é a constante $\frac{1}{N} * C$ e C é a constante σ , onde N representa o tamanho da população e σ é o desvio padrão da função de avaliação. O $rank(p)$ é o índice da string p na lista classificada internamente.

Para auxiliar o AG no processo de busca por uma solução, além dos operadores clássicos de mutação e cruzamento, foram estudados e desenvolvidos alguns operadores específicos para o problema do planejamento, conforme descrição a seguir.

A função de avaliação, que tem por objetivo medir a qualidade de cada indivíduo em relação ao problema de planejamento, cria um plano com o cromossomo avaliado, ao mesmo tempo em que devolve algumas métricas importantes para a avaliação final do indivíduo, com relação ao plano criado, como:

- ? o número da Primeira Ação Executável do plano (PAE);
- ? o número da Última Ação Executável do plano (UAE);
- ? o número de Ações Executáveis (AE);
- ? o número de Ações Consecutivas Executáveis (ACE);
- ? o número de Ações Válidas (AV);
- ? o número de Objetivos Alcançados (OA);
- ? o número Total de Objetivos (TO);
- ? a representação binária do cromossomo em relação a executabilidade, excluídos os NOPs (Figura 12);

? a representação do cromossomo de inteiros gerada pelo AG (Figura 12).

Para facilitar a análise da executabilidade das ações de um plano representado pelo cromossomo de inteiros, o procedimento cria um cromossomo paralelo de binários, onde o bit “1” representa uma ação executável e o bit “0” uma ação que não é executável nas suas respectivas posições, eliminadas as ações nulas “nop”, como exemplificado na representação abaixo.

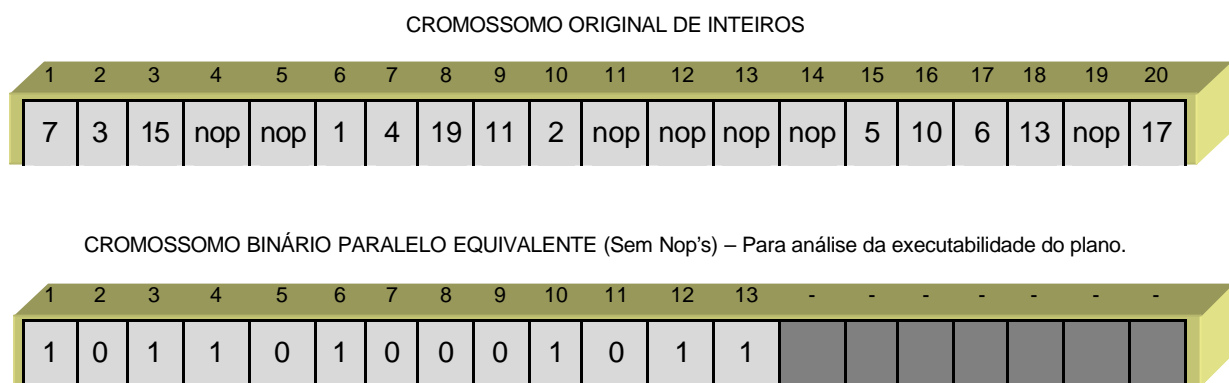


Figura 12. Representação de um cromossomo binário paralelo equivalente.

Sobre todos estes resultados, implementamos alguns tratamentos especiais, com o objetivo de controlar a evolução dos indivíduos e maximizar seu potencial de gerar filhos mais adaptados ao ambiente-problema, tais como:

1. *Ações Válidas*: - verificamos se o número de Ações Válidas (ΔV) do cromossomo avaliado for igual a zero, encerramos imediatamente a sua avaliação, retornando um valor de avaliação igual a zero;
2. *Blocos Contínuos de Ações Executáveis*: - contamos o número de blocos contínuos de Ações Executáveis (ΔE), para valorizar a avaliação do indivíduo que produzir o menor número destes blocos, indicando um cromossomo com ações mais coesas;
3. *Ações NOP's*: - este parâmetro determina a probabilidade de execução da rotina que passa pela representação do cromossomo de inteiros, substituindo randomicamente as ações NOP por outras aleatórias, produzindo um indivíduo com menos ações NOP e, conseqüentemente, maior valor de avaliação;

4. *Fator de Correção de Compactação*: - este parâmetro determina a probabilidade de executar o procedimento que reformula o cromossomo binário, para que as Ações Executáveis (AE), representadas por 1 no cromossomo binário, sejam agrupadas a partir de um “meio-cromossomo” aleatório, para as pontas, deixando um só bloco de Ações Executáveis, agrupando a representação de Ações Não-Executáveis, zero binário, ora nas extremidades, ora em apenas uma das pontas do cromossomo. Toda inversão feita sobre a representação binária implica numa inversão na representação de inteiros, produzindo um novo indivíduo com boa avaliação e grande potencial de gerar filhos interessantes para o problema;
5. *Distribuição de Pesos*: - na medida em que as gerações vão sendo avaliadas, são atribuídos alguns Pesos, que permitem um balanceamento das verificações na medida em que as gerações evoluem, conforme a distribuição abaixo indicada:

% Gerações	Peso 1 (k1) %	Peso 2 (k2) %	Peso 3 (k3) %
Geração <= 30	10	40	50
30 < Geração <= 40	20	40	40
40 < Geração <= 50	30	30	40
50 < Geração <= 70	40	30	30
70 < Geração <= 80	50	20	30
80 < Geração	60	20	20

Tabela 7. Variação de Pesos conforme Gerações.

6. *Cálculo do Valor de Retorno*: - a primeira parte do cálculo valoriza a quantidade de objetivos alcançados (OA) em relação ao total de objetivos (TO), com Peso 1 (k1), que avalia esta medida conforme a evolução das gerações, da seguinte maneira:

$$\text{valor} = k1 * (OA / TO)$$

A segunda parte valoriza o percentual de ações executáveis (AE) em relação ao número de ações válidas (AV) e de ações executáveis (AE) subtraídas as ações consecutivas executáveis (ACE) sobre o número de ações válidas (AV), com Peso 2 (k2), que diminui em função do avanço das gerações, da seguinte maneira:

$$\text{valor} += k2 * ((AE / AV) * ((AE - ACE) / AV))$$

A terceira e última parte valoriza o menor percentual de blocos de ações executáveis (BL) em relação ao número de ações válidas (AV), com Peso 3 (k_3), que diminui em função do avanço das gerações, da seguinte maneira:

$$\text{valor} += k3 * (1 - (BL / AV))$$

Com base nisto, a função deverá fornecer um *índice de qualidade* para uma possível solução, conforme o número de ações executáveis desta solução, isto é, quanto mais ações executáveis de um plano um indivíduo possuir e quanto mais organizadas estiverem as ações no cromossomo, maior será a qualidade deste indivíduo, representada por um maior valor de avaliação, que teoricamente terá maiores chances de se manter para as próximas gerações, bem como gerar filhos com bom potencial avaliativo. Ao passo que os indivíduos que forem mal avaliados nesta função serão automaticamente penalizados no processo evolutivo, exceto pelo percentual de aleatoriedade, não devendo aparecer na geração seguinte e conseqüentemente sendo abandonados pelo processo evolutivo. A aleatoriedade permite manter uma boa diversidade. Com isto, obtemos como resultado a evolução esperada das populações com indivíduos cada vez mais adaptados ao problema proposto, como se espera, a caminho da solução adequada ao problema.

Adicionamos, ainda, um segundo critério de parada do algoritmo, que pode acontecer antes do término das gerações e que verifica se os objetivos do problema já foram alcançados, causando a interrupção imediata do algoritmo e apresentação gráfica do resultado, pois, neste caso, uma solução foi obtida. Um gráfico é montado também ao final do número de gerações solicitadas, para o caso da solução não ter sido encontrada, com o objetivo de permitir uma análise visual da evolução das gerações.

4.2. EXPERIMENTOS REALIZADOS

Para o atendimento dos objetivos propostos, os esforços do trabalho foram direcionados para os seguintes pontos:

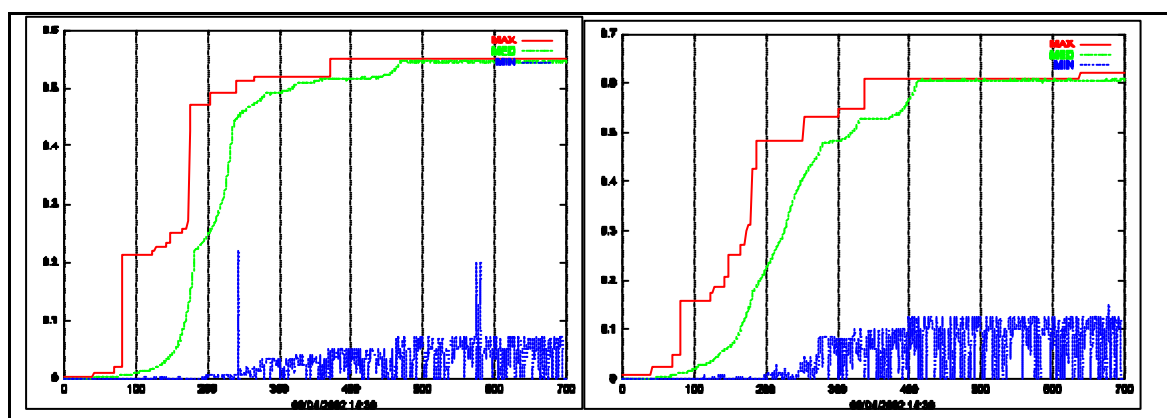
- ? Encontrar funções de avaliação (*fitness*) mais adequadas para o problema do planejamento em IA;
- ? Identificar os parâmetros genéticos e métodos que produzem os melhores resultados na execução do algoritmo;
- ? Identificar e implementar novos operadores para tratamento dos cromossomos.

Nas tabelas, a seguir, será demonstrado o número seqüencial do teste (N^o), o valor dos Pesos utilizados multiplicados por 10 ($K1$, $K2$ e $K3$), os parâmetros de tamanho da população ($PpSize$), número máximo de gerações ($MaxGen$), percentual da população de melhores indivíduos mantidos para as gerações seguintes ($Repl.$), permissão de cromossomos duplicados (Dup), taxa percentual para operação de Cruzamento ($Cross$), taxa percentual para operação de Mutação ($Mut.$) e tamanho do cromossomo ($Size$). O argumento *TEMPO*, dado em segundos, refere-se à medição do tempo de processamento necessário para a apresentação dos resultados de cada teste.

A Tabela 8 abaixo resume os principais resultados obtidos a partir das execuções parametrizadas de testes sobre o domínio *Gripper* [37], nas primeiras implementações, ainda com pesos fixos e sem os tratamentos especiais e resultados pouco expressivos:

Nº	Pesos			Parâmetros Gerais							TEMPO (s)
	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	
1	8	2	-	2000	700	N	S	0,7	0,03	11	73,425
2	5	5	-	2000	700	N	S	0,7	0,03	11	72,197
3	5	5	-	500	1200	N	S	0,7	0,03	11	51,358
4	5	3	2	5000	1000	N	S	0,7	0,03	11	414,127
5	5	4	1	2000	2000	N	S	0,7	0,03	11	325,328

Tabela 8. Primeiros experimentos.



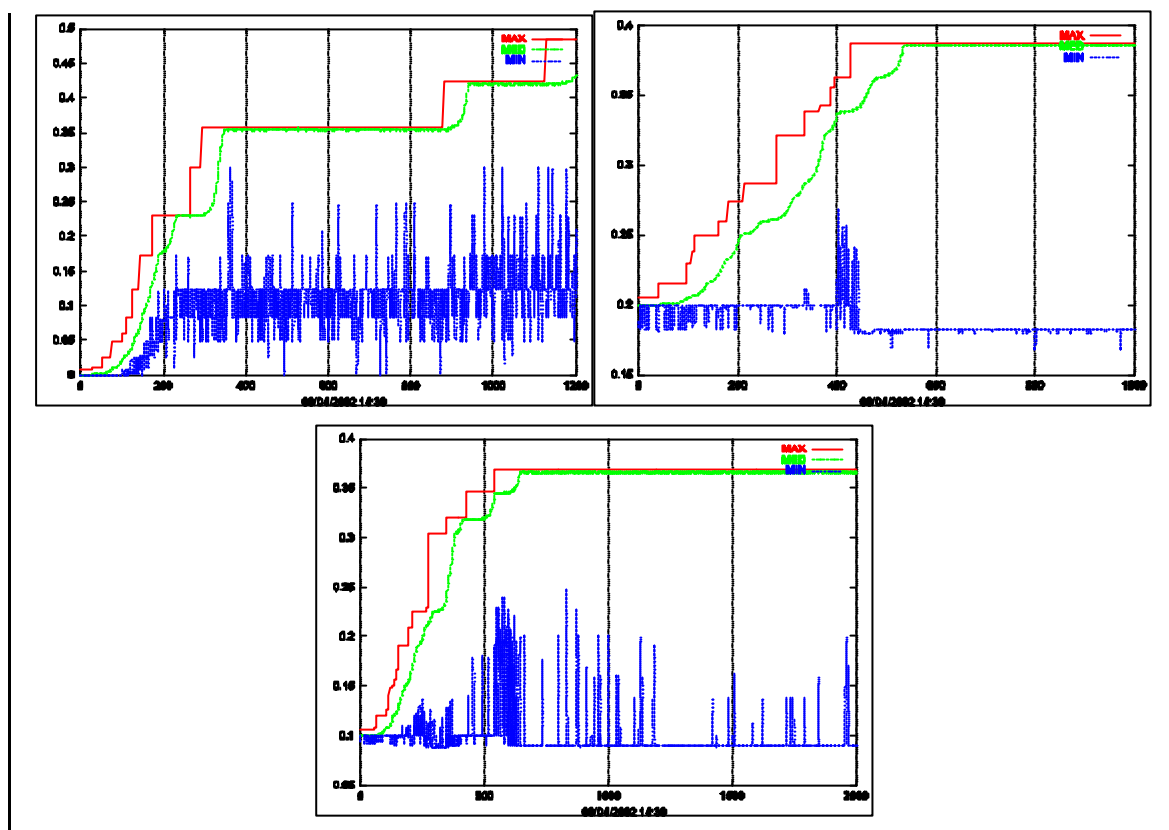


Figura 13. Gráficos da tabela 8. Curvas de fitness Máximo, Médio e Mínimo das populações.

Com a implementação dos pesos variáveis, como descrito na Tabela 7, e a possibilidade de manter um percentual da população anterior e impedir a ocorrência de indivíduos duplicados, alterando-se apenas um parâmetro na linha de comando, obtivemos um acréscimo substancial no tempo de processamento, conforme demonstra a Tabela 9, a seguir, cujos resultados demonstravam com clareza a ocorrência do processo evolutivo, mas ainda insuficiente para determinar uma solução:

Nº	Pesos			Parâmetros Gerais							TEMPO (s)
	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	
6	2-8	8-2	-	2000	1000	N	S	0,7	0,03	11	426,475
7	1-6	5-0	4	2000	1000	N	S	0,7	0,03	11	239,744
8	1-6	5-0	4	2000	1000	50%	N	0,7	0,03	11	975,944

Tabela 9. Resultados após implementação de pesos variáveis.

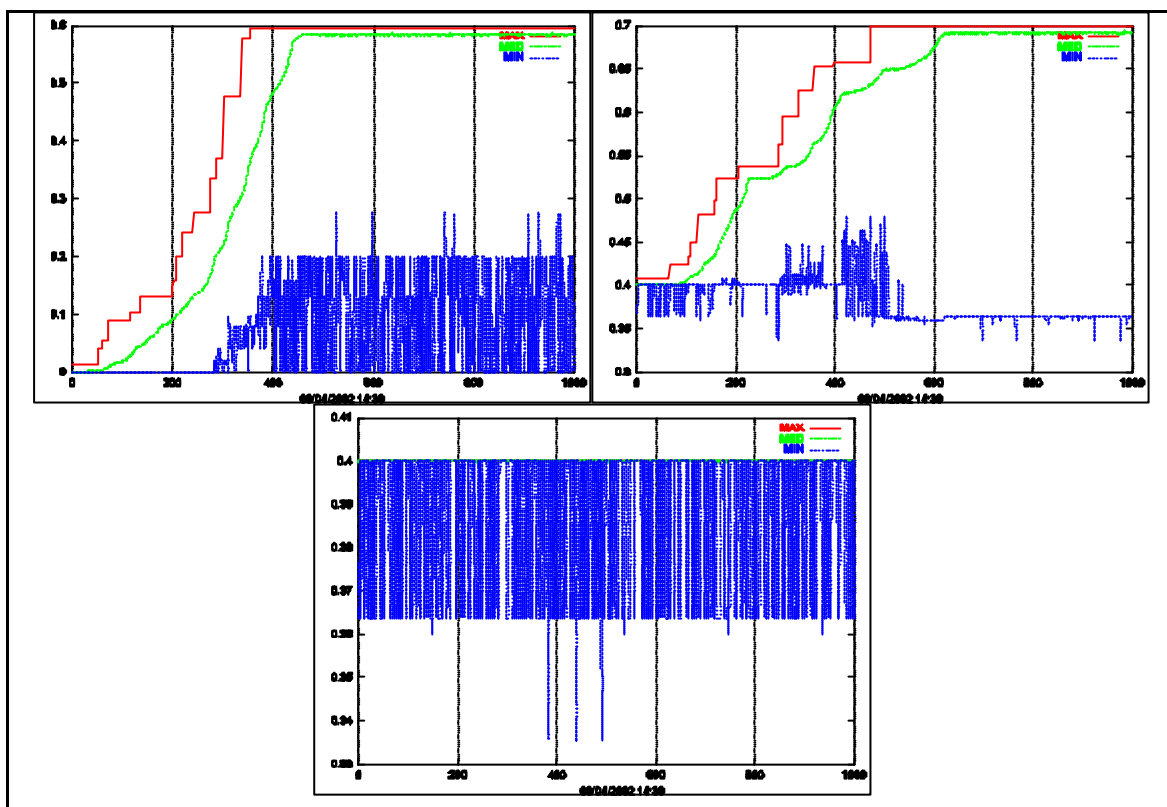


Figura 14. Gráficos da tabela 9.

Vários testes foram feitos para a adequação das taxas de cruzamento, mutação, de substituição da população, variação dos pesos e tamanho do cromossomo, numa espécie de ajuste fino dos parâmetros, seguidos de análise de resultados e comportamentos, como demonstra a Tabela 10 a seguir e seus gráficos, que nos permitem concluir que o comportamento evolutivo do AG responde prontamente à aposição dos parâmetros genéticos, indicando um bom funcionamento do algoritmo nesta parte:

Nº	Pesos			Parâmetros Gerais							TEMPO (s)
	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	
9	1-6	5-0	4	2000	1000	10%	N	0,7	0,03	11	892,471
10	1-6	5-0	4	2000	1000	2	S	0,7	0,03	11	3808,289
11	1-6	5-0	4	2000	1000	10%	S	0,7	0,1	11	1783,931
12	1-6	5-0	4	2000	1000	10%	S	0,7	0,3	11	2044,761
13	1-6	5-0	4	2000	1000	10%	S	0,7	0,1	11	1937,648
14	1-6	5-0	4	500	1000	N	S	0,7	0,1	11	34,794

Tabela 10. Ajustes de parâmetros genéticos.

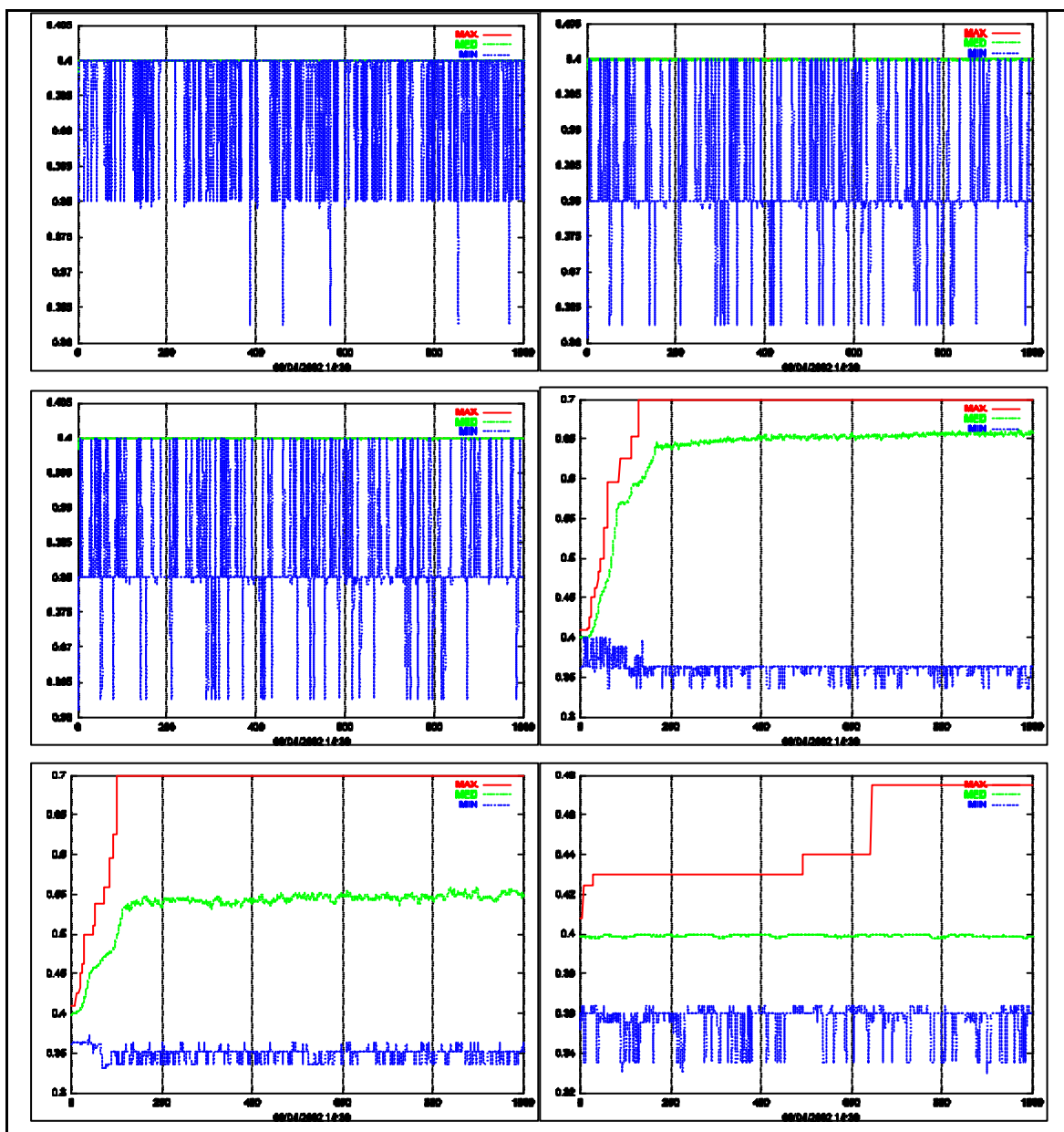
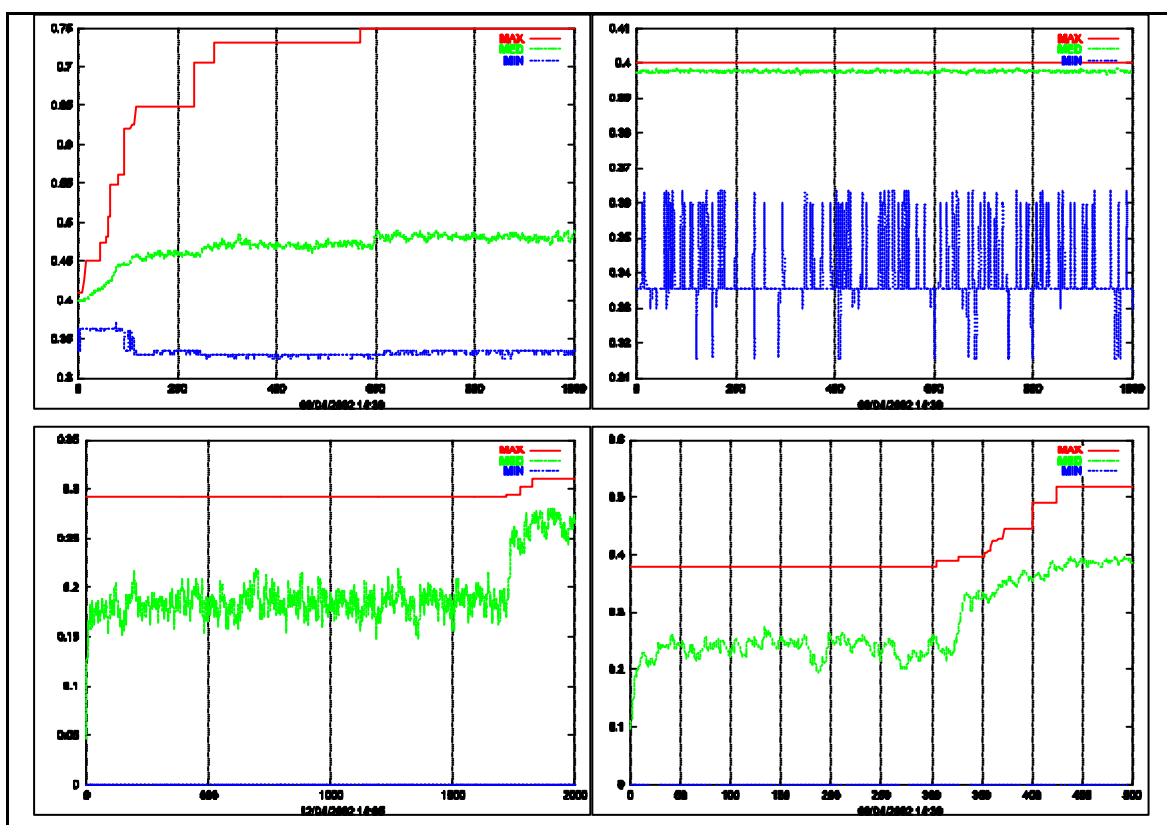


Figura 15. Gráficos da tabela 10.

O aumento do tamanho do cromossomo de 11 para 20 ações permitiu uma maior liberdade de ocorrência de ações nulas sem o prejuízo das executáveis, e com isto a elaboração de planos mais próximos dos ideais. Os ajustes de parâmetros genéticos e dos parâmetros especiais como a distribuição dos pesos, levaram a uma configuração ideal no teste de número 28 que, na geração 1539, encontrou um plano com a solução do problema, como demonstra a Tabela 11 a seguir e seus gráficos :

Nº	Pesos			Parâmetros Gerais							TEMPO (s)
	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	
15	1-6	5-0	4	500	1000	N	S	0,7	0,03	20	55,564
16	1-6	5-0	4	500	1000	N	S	0,7	0,3	20	57,327
17	1-6	5-0	4	500	1000	10%	S	0,7	0,3	20	58,271
18	1-6	5-0	4	500	500	10%	S	0,7	0,1	20	241,114
19	2-7	6-1	2	300	800	10%	S	0,7	0,1	20	236,353
20	2-7	6-1	2	500	1000	10%	S	0,7	0,1	20	478,107
21	2	5	3	500	2000	10%	S	0,7	0,1	20	974,305
22	1-6	6-1	3	1000	2000	10%	S	0,7	0,1	20	1915,955
23	1-6	6-1	3	1000	20000	10%	S	0,7	0,1	20	Reseted by Peer
24	1-6	6-1	3	500	2000	10%	S	0,9	0,05	20	1185,787
25	2	5	3	500	2000	10%	S	0,7	0,1	20	1840,151
26	2	5	3	100	2000	10%	S	0,7	0,3	20	189,076
27	2	5	3	300	2000	10%	S	0,7	0,1	20	553,893
28	2	5	3	500	2000	10%	S	0,7	0,1	20	Solução 1539

Tabela 11. Mais ajustes de parâmetros genéticos.



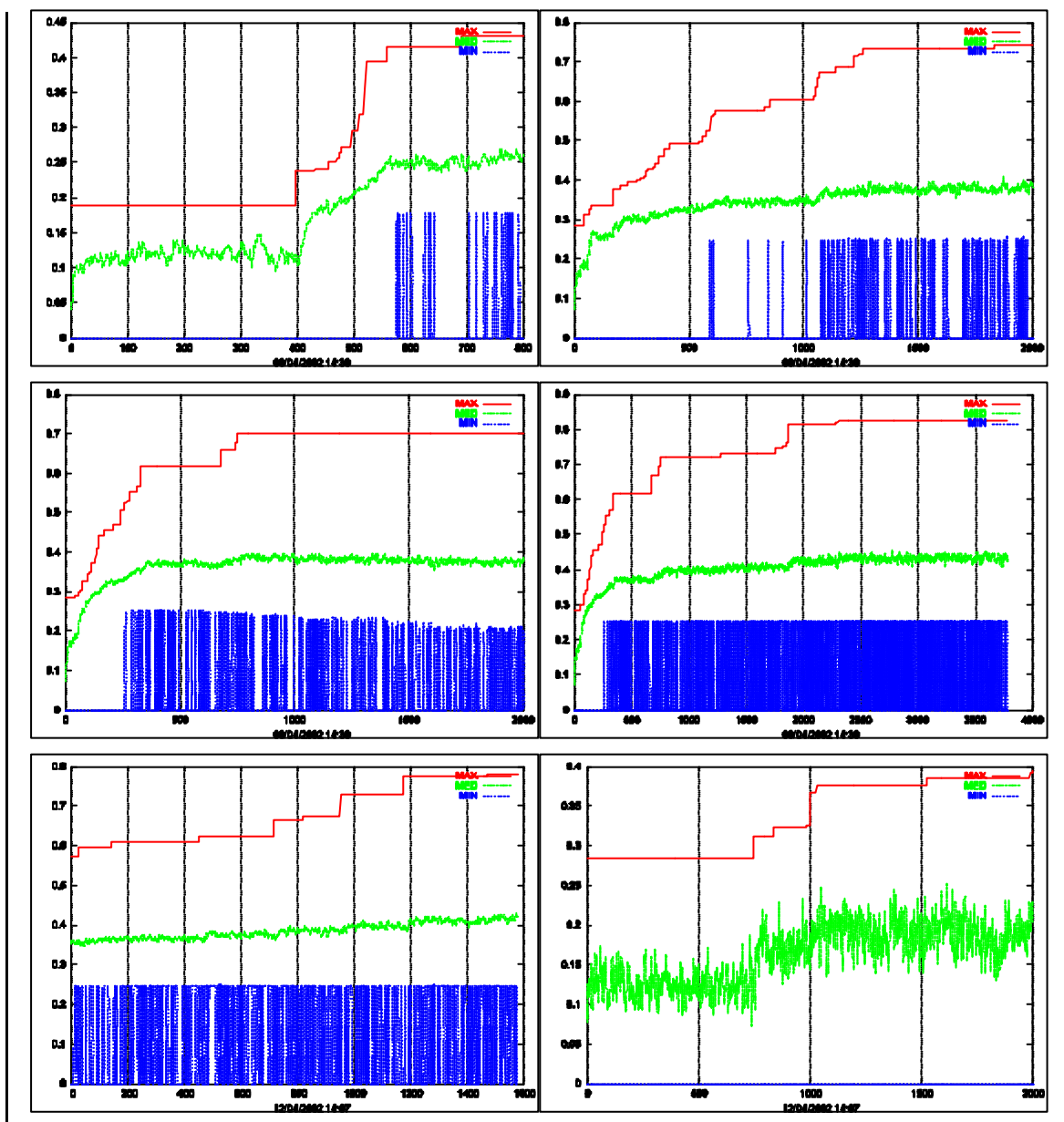


Figura 16. Gráficos da tabela 11.

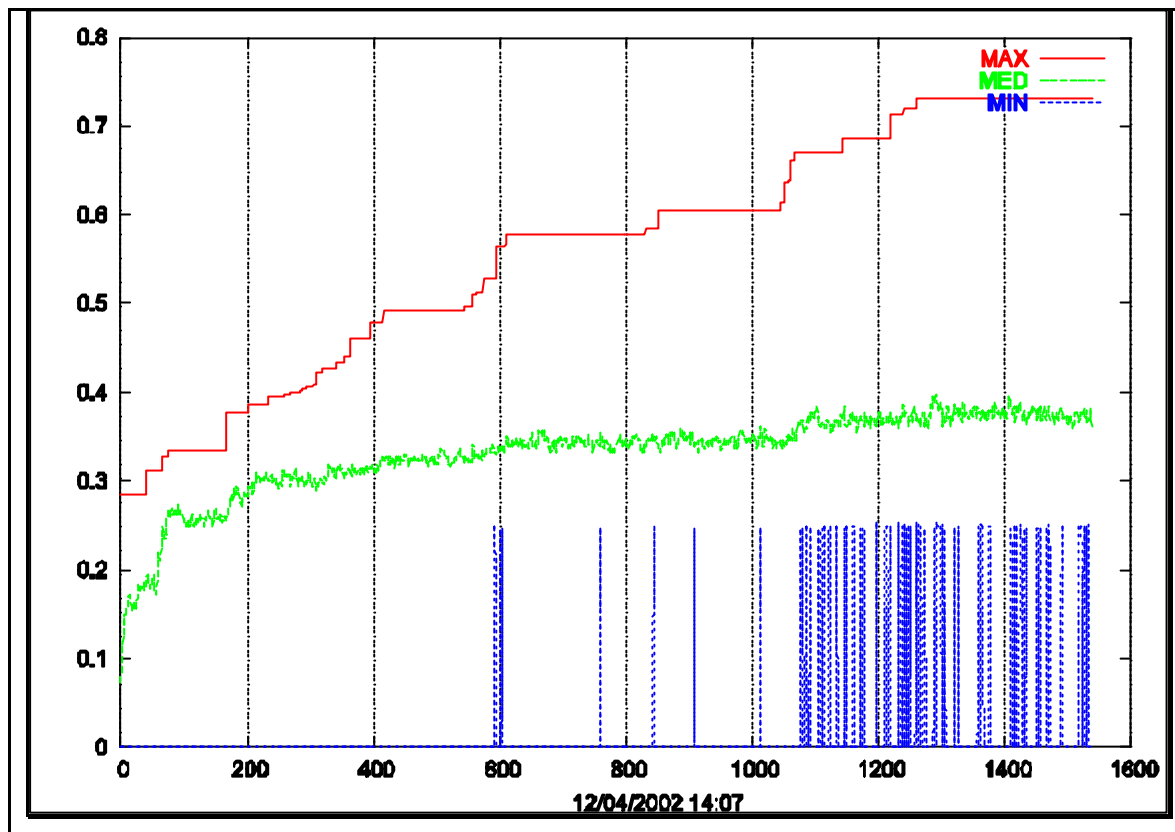


Figura 17. Gráficos do teste 28, encontrando a solução após 1539 gerações.

A partir do momento onde uma solução válida foi encontrada, gráfico acima, implementamos o tratamento de ações NOP's e testamos novamente a solução com os parâmetros usados em outros domínios, como demonstra a Tabela 12 a seguir, com resultados infrutíferos, que nos levaram a considerar a criação de outras operações genéticas especiais para o tratamento dos cromossomos:

Problema 01 – Domínio Gripper											
Nº	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	TEMPO
29	1-6	6-1	3	3000	500	10%	S	0,7	0,1	20	1471,811
30	1-6	6-1	3	750	2000	10%	S	0,7	0,1	20	1381,636
31	2	5	3	500	2000	10%	S	0,7	0,1	20	573,619
Problema 02 – Domínio Gripper											
Nº	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	TEMPO
1	1-6	6-1	3	500	2000	10%	S	0,7	0,1	20	1069,593
Problema 03 – Domínio Gripper											
Nº	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	TEMPO
1	1-6	6-1	3	500	2000	10%	S	0,7	0,1	20	1150,594

Problema 04 – Domínio Gripper											
Nº	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	TEMPO
1	1-6	6-1	3	500	2000	10%	S	0,7	0,1	20	1454,053
Problema 01 – Domínio Logistics											
Nº	K1	K2	K3	PpSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	TEMPO
1	2	5	3	500	2000	10%	S	0,7	0,1	20	1377,342
2	2	5	3	500	1000	10%	S	0,8	0,2	20	747,047
3	1-6	6-1	3	1000	500	10%	S	0,8	0,1	20	742,872

Tabela 12. Testes em outros domínios de planejamento.

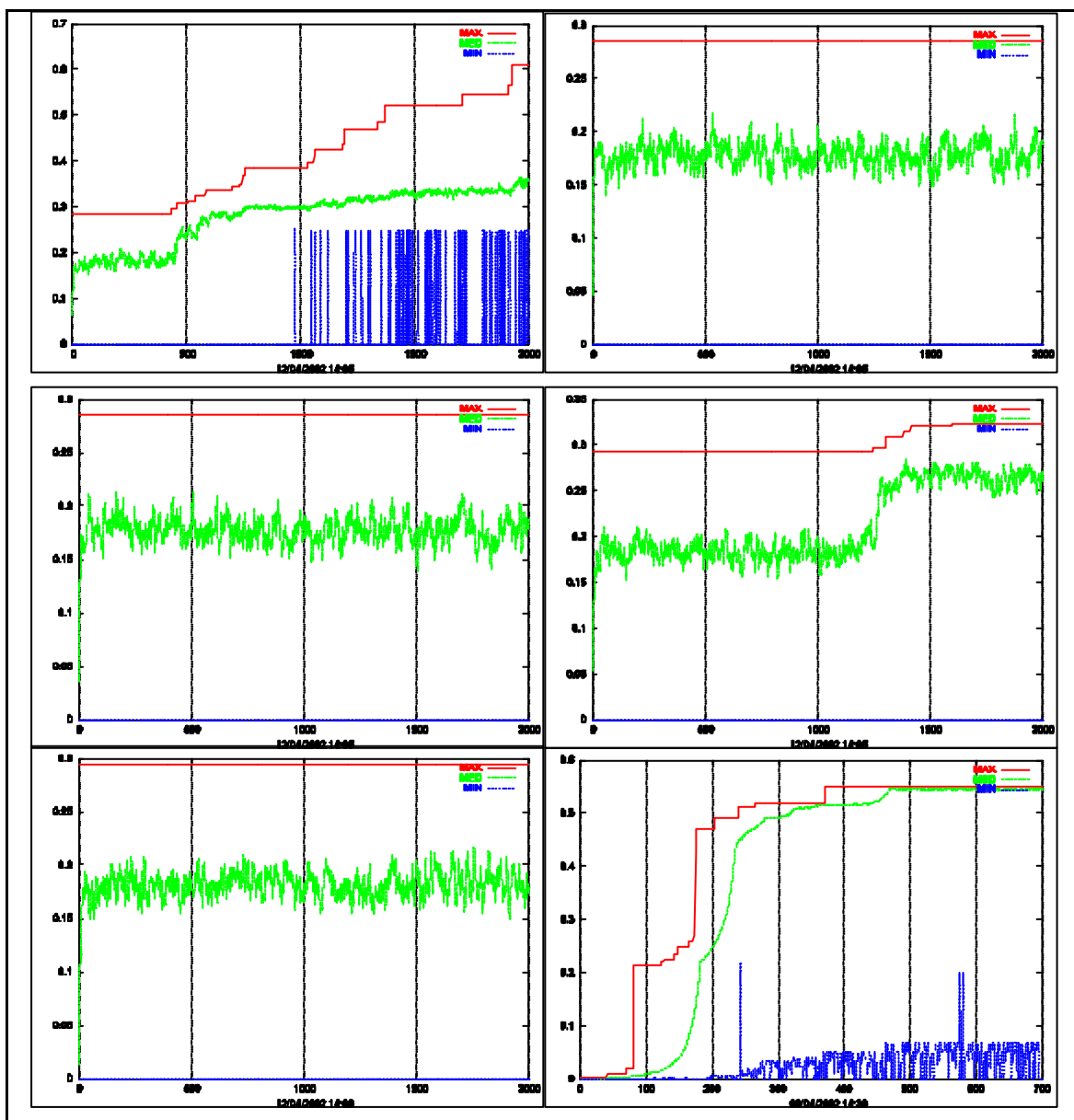


Figura 18. Gráficos da tabela 12.

Criamos o fator de correção da compactação que trata-se de um valor percentual que determina a frequência em que o programa reconstrói o cromossomo binário com a eliminação de NOPs a partir de posição aleatória central e reavaliação deste plano após a correção. Voltando ao domínio *Gripper* e alterando a forma de evolução dos pesos com o passar das gerações, implementando o fator de correção da compactação, obtivemos uma pequena melhora no comportamento evolutivo, como mostra a Tabela 13, abaixo. Os dois últimos testes deste grupo, implementam fatores variáveis de correção da compactação e de tratamento de ações NOP's, de acordo com o número de gerações avaliadas, com impacto direto sobre o tempo de processamento:

Nr.	K1	K2	K3	PopSize	MaxGen	Repl.	Dup	Cross	Mut.	Size	Reav	TEMPO
1	1-6	6-1	3	500	2000	10	S	0,7	0,1	20	Com	983,937
2	2	5	3	500	2000	10	S	0,7	0,1	20	Com	881,961
3	1-6	6-1	3	500	2000	10	S	0,7	0,1	20	Com	823,435
4	1-6	4-2	5-2	500	2000	10	S	0,7	0,1	20	Com	734,279
5	1-6	4-2	5-2	500	2000	10	S	0,7	0,1	20	Sem	361,392
6	1-6	4-2	5-2	500	2000	10	S	0,7	0,1	20	Sem	310,716
7	1-6	4-2	5-2	500	2000	10	S	0,7	0,1	20	Sem	293,944

Tabela 13. Testes com Fator de correção e reavaliação do plano.

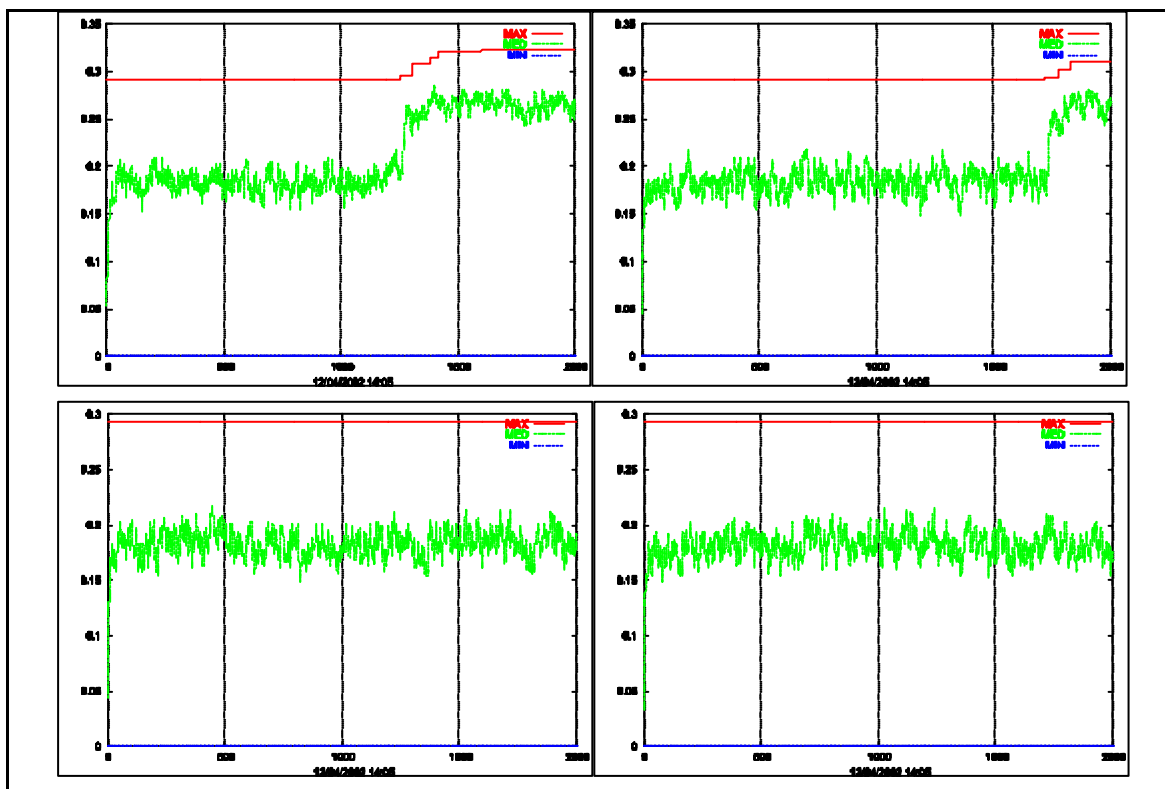


Figura 19. Gráficos da tabela 13.

4.3. CONCLUSÕES E MOTIVAÇÕES PARA MUDANÇA

A partir dos resultados destes testes, identificamos que as soluções não estavam evoluindo como esperado e que os tempos de execução estavam muito altos, dificultando a execução de uma bateria de testes para a identificação de problemas. Da mesma forma, precisávamos implementar novas operações genéticas especiais para tentar alcançar o resultado esperado, mas ao mesmo tempo tornando o processamento do programa mais pesado e, a cada vez, dificultando ainda mais a obtenção da praticidade dos testes.

Aprendemos com os resultados desta fase que o comportamento evolutivo é promissor no tratamento do problema de planejamento em IA, mas que se torna imprescindível um aprimoramento na forma de como está sendo tratado, isto é, através dos códigos prontos de bibliotecas de domínio público, tanto de abordagem genética quanto à de planejamento, visto que não temos o completo acesso sobre todas as operações de código às quais o programa está se submetendo.

A biblioteca de planejamento faz a leitura do domínio e do problema, escritos em PDDL, e gera uma combinação muito grande de ações instanciadas, visto que não existe o tratamento para a definição dos tipos de objetos parametrizados das ações. Isto ocasiona um incremento no espaço de busca com ações que não são válidas, isto é, parametrizadas com objetos de tipos errados, e que acabam precisando ser tratadas pelo programa como um todo, ocupando espaço e tempo preciosos para o alcance da resolução do problema. Acreditamos que o espaço de busca será grandemente otimizado, isto é, reduzido, com o correto tratamento dos tipos de objetos parametrizáveis para a correta instanciação e geração apenas de ações que sejam válidas.

A biblioteca genética não permite o tratamento de cromossomos de tamanhos variáveis de inteiros com um controle paralelo de cromossomos binários equivalentes e sempre gera uma população inicial aleatória, conforme nossa proposta de modelagem, além de não permitir a implementação de qualquer código intermediário que não pertença à função de avaliação ou ao início e fim de uma geração.

Uma nova abordagem requer que o código pronto das bibliotecas seja re-escrito para uma adaptação ao novo ambiente operacional, devendo contemplar todas as operações necessárias para o bom desempenho do programa. Apesar de complexa e muito trabalhosa, esta opção nos permitirá o total controle sobre as operações realizadas

pelo programa, como esperamos que aconteça no caso da tipagem dos parâmetros para a redução do espaço de busca.

A concepção da modelagem inicial do problema de planejamento tratado por um algoritmo genético será aproveitada, visto que demonstrou ser de boa adaptabilidade, bem como a representação de ações NOP's que não alteram a situação do mundo, os pesos variáveis conforme a evolução das gerações, os operadores genéticos especiais para a heurística genética de planejamento e uma reformulação dos procedimentos que antes eram automáticos pertencentes ao código das bibliotecas públicas.

Nesta última observação, consta o maior peso desta reformulação, devido ao fato de que os procedimentos de planejamento, que estavam sendo feitos pelas bibliotecas de domínio público, como a leitura de uma entrada no formato PDDL, o tratamento e a validação das ações no plano, a verificação dos planos, as consistências e, fundamentalmente, o problema do tratamento de ações não instanciadas, que não podiam ser tratadas com o uso das bibliotecas, serão desenvolvidos, recodificados e testados no novo ambiente.

Na fase que antecede a execução, o sistema deverá fazer a leitura de arquivos em PDDL contendo domínios e ações, inferir os tipos de dados e instanciar as ações, reduzindo o espaço de busca pelo uso de ações válidas com os argumentos de tipos apropriados e implementar o modelo genético proposto. Deverá este ainda, permitir a visualização gráfica dos resultados parciais e finais.

No próximo capítulo, faremos o detalhamento desta reformulação nos procedimentos adotados até o momento, nas linguagens e no ambiente operacional, através de uma implementação visual do sistema, para o atendimento aos objetivos deste trabalho de pesquisa.

5. UMA NOVA ABORDAGEM GENÉTICA

Pelos resultados apresentados na primeira etapa e pela dificuldade de visualização da execução de passos intermediários, geridos pelas eficientes porém encapsuladas bibliotecas de domínio público tanto genéticas quanto de planejamento, optamos pela total remodelação do processo.

Foram revisados os parâmetros genéticos e foi reescrito todo o código do programa. Passamos a tratar desde a leitura dos arquivos escritos em PDDL, sua interpretação com a definição dos tipos das ações com os objetos parametrizáveis dos problemas, a instanciação apenas de todas as ações válidas, a verificação de estados do mundo, a simulação da execução das ações e a correspondente atualização desses estados.

Na parte genética, ao invés do uso da biblioteca de domínio público, desenvolvemos todo o código para gerar uma população inicial aleatória, evoluir seletivamente através dos operadores básicos de cruzamento e mutação, criar e adaptar uma função de avaliação mais apropriada para os problemas e controlar todo o processo por código próprio, permitindo sua manutenção, independente do instante do processo genético em que se encontra. Todos estes recursos, na medida do possível, foram parametrizados e são controlados por uma semente de números aleatórios para que seja possível repetir um mesmo teste com a passagem dos parâmetros iguais.

Também optamos pelo armazenamento dos dados em sistema de Banco de Dados para posterior recuperação das informações e guarda da memória dos testes efetuados, bem como a geração dos fundamentais gráficos e relatórios para a análise dos comportamentos genéticos e ajustes dos parâmetros.

Por último, houve uma readequação do sistema para que o mesmo pudesse trabalhar com baterias de testes ao invés de testes isolados, facilitando a elaboração do banco de testes para análise e automatizando o processo para a investigação dos resultados.

O principal objetivo deste trabalho foi o de aplicar uma técnica promissora, de Algoritmos Genéticos, para a resolução de um dos mais importantes problemas da área de Inteligência Artificial que é o problema de Planejamento. Em tempo algum procuramos obter um desempenho comparável aos sistemas competidores vistos na seção 2.2 e todos os esforços foram direcionados ao aprendizado e entendimento das técnicas

envolvidas e comprovação do que teoricamente imaginávamos com alto potencial de adequabilidade.

A partir da modelagem apresentada no capítulo anterior, optamos por fazer uma série de testes capazes de demonstrar na prática todos os conceitos e fundamentos teóricos estudados. O crescimento do trabalho, como produto final, ficou evidenciado na medida em que uma série de testes era executada e seus resultados levavam ao entendimento de ajustes e adaptações no código para a melhoria dos resultados. Tal melhoria, que por vezes não se confirmava, obrigava a uma readaptação do código ou a criação de operadores genéticos especiais, conforme será visto adiante neste capítulo.

Neste sentido, a nova abordagem independente das bibliotecas públicas e fechadas, garantindo o completo domínio de todo o código escrito e a opção pela plataforma visual nos trouxe uma maior sensibilidade aos resultados apresentados, permitindo uma manutenção pontual de problemas e inserção de controles para o acompanhamento das funcionalidades propostas.

Neste capítulo, na seção 5.2, descrevemos a nova abordagem genética para o problema do planejamento, com a implantação de um sistema, batizado como *GAPlan* – Genetic Algorithm for Planning, desenvolvido em linguagem visual para o ambiente Windows. A interface implementada está descrita como Apêndice A - *GAPlan* – Um Planejador Genético ao final deste trabalho. Outra contribuição está definida na próxima seção que detalha a maneira de encontrar e inferir os tipos de objetos parametrizáveis, com o objetivo de instanciar apenas ações com objetos de tipos válidos e obter a conseqüente redução no espaço de busca. Por fim, na seção 5.3 que encerra o capítulo, serão apresentados os principais resultados, que balizaram novas adaptações de código, até chegarmos à versão atual do programa que implementa importantes diferenciais em relação à modelagem original.

5.1. TIPAGEM E INSTANCIAÇÃO

Em PDDL, a representação de domínios e problemas é feita de maneira organizada e simplificada, mas nem sempre direta. No início das definições de domínios estão especificados os predicados com seus parâmetros. Nas definições das ações, estão especificados os parâmetros de entrada, as pré-condições e os efeitos. Diferentemente do tratamento dado a este assunto pelas bibliotecas de domínio público utilizadas até o momento, a nossa proposta de remodelagem consiste em definir uma fase de pré-

processamento sobre os domínios não-tipados para o tratamento e inferência dos tipos, gerando apenas instanciações válidas e otimizando o espaço de busca. Para tanto, o programa que faz a leitura e interpretação deste formato, necessita instanciar as ações com os objetos de tipos apropriados para que sejam criadas instâncias válidas das ações.

O tradicional domínio do mundo de blocos, conforme Figura 2 não é um bom exemplo para o que pretendemos demonstrar, visto que possui um número pequeno de predicados e seus parâmetros só podem ser de um tipo: “bloco”. A versão mais básica das ações pegar, soltar, empilhar e desempilhar praticamente esgota todas as possibilidades de validação desde que obedecidas as pré-condições da ação. Portanto, neste domínio, uma versão tipada reduz com pouca ou nenhuma vantagem o espaço de busca em relação à sua versão não-tipada.

Já o domínio Logistics, ao contrário do mundo de blocos, possui muitos predicados e seus parâmetros podem ser de vários tipos, por exemplo caminhão, avião, aeroporto e cidade. Quando é feita uma instanciação de uma ação básica, por exemplo, na Figura 20 abaixo, a ação “load-airplane” que recebe os parâmetros “?obj”, “?airplane” e “?loc”, mas que não conhece os tipos de objetos e portanto, na divisão do arranjo combinatorial ocasionará uma instanciação errada de objetos que não obedecem aos tipos esperados.

```
(:action load-airplane
:parameters
  (?obj
   ?airplane
   ?loc)
:precondition
  (and (package ?obj) (airplane ?airplane) (location ?loc)
        (at ?obj ?loc) (at ?airplane ?loc))
:effect
  (and (not (at ?obj ?loc)) (in ?obj ?airplane)))
```

Figura 20. Ação descrita em PDDL Não-Tipado para o domínio Logistics.

O problema acontece porque na versão não-tipada, que é a mais utilizada pela comunidade, não existe uma referência direta para dizer que um bloco é do tipo “bloco” e só pode ser movido de um local para outro, ou que um avião é do tipo “avião” e que pode ser carregado, descarregado e voar de uma “cidade” para outra. Isto é, nada existe para garantir que uma cidade e um aeroporto são do tipo “local” e que um avião e um caminhão são os meios de transporte. Sem estas definições, o programa faz instanciações equivocadas como por exemplo, carregar uma “cidade” em um “avião” ou fazer um “aeroporto” voar de um “caminhão” para outro.

Para facilitar o entendimento geral do problema, optamos por exemplificar sobre um domínio de complexidade intermediária, o domínio Gripper, cujos objetos podem ser

bolas, garras ou quartos e que a falta da definição de tipos pode gerar instanciações claramente inválidas como “mover quarto A para quarto B”, ou ainda “pegar quarto B na garra esquerda com a bola 3”.

```
(define (domain gripper-strips)
  (:predicates (room ?r)
    (ball ?b)
    (gripper ?g)
    (at-robby ?r)
    (at ?b ?r)
    (free ?g)
    (carry ?o ?g))

  (:action move
    :parameters (?from ?to)
    :precondition (and (room ?from) (room ?to) (at-robby ?from))
    :effect (and (at-robby ?to)
      (not (at-robby ?from))))

  (:action pick
    :parameters (?obj ?room ?gripper)
    :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
      (at ?obj ?room) (at-robby ?room) (free ?gripper))
    :effect (and (carry ?obj ?gripper)
      (not (at ?obj ?room))
      (not (free ?gripper))))

  (:action drop
    :parameters (?obj ?room ?gripper)
    :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
      (carry ?obj ?gripper) (at-robby ?room))
    :effect (and (at ?obj ?room)
      (free ?gripper)
      (not (carry ?obj ?gripper)))))
```

Figura 21. Arquivo em PDDL com o Domínio Gripper *Não-Tipado*.

Ao compararmos uma definição não-tipada, conforme Figura 21 acima, com a mesma definição tipada, Figura 22 a seguir, de um domínio escrito em PDDL, verificamos que as diferenças estão nas sentenças que definem as constantes e os tipos dos objetos no início da definição do domínio e estão dentro de cada ação, onde os parâmetros são identificados com seus respectivos tipos. Uma consequência direta desta falta de definições na versão não-tipada é que, as pré-condições das ações precisam, na maioria das vezes, garantir que determinado parâmetro seja de determinado tipo, isto é, trata tardiamente o problema da instanciação, pois para chegar até ali, o programa já gastou tempo e processamento para criar uma ação inválida que vai ser rejeitada pela pré-condição.


```

(define (domain gripper-typed)
  (:requirements :typing)
  (:types room ball gripper)
  (:constants left right - gripper)
  (:predicates (at-robby ?r - room)
    (at ?b - ball ?r - room)
    (free ?g - gripper)
    (carry ?o - ball ?g - gripper))

  (:action move
    :parameters (?from ?to - room)
    :precondition (at-robby ?from)
    :effect (and (at-robby ?to)
      (not (at-robby ?from))))

  (:action pick
    :parameters (?obj - ball ?room - room ?gripper - gripper)
    :precondition (and (at ?obj ?room) (at-robby ?room) (free ?gripper))
    :effect (and (carry ?obj ?gripper)
      (not (at ?obj ?room))
      (not (free ?gripper))))

  (:action drop
    :parameters (?obj - ball ?room - room ?gripper - gripper)
    :precondition (and (carry ?obj ?gripper) (at-robby ?room))
    :effect (and (at ?obj ?room)
      (free ?gripper)
      (not (carry ?obj ?gripper)))))

```

Figura 22. Arquivo em PDDL com o Domínio Gripper *Tipado*.

A versão tipada normalmente exige na instrução “(:requirements:” a especificação “ADL” ou “typing”, indicando que estas construções suportam tipos e quantificações. Os tipos são indicados por hífen e aparecem após o objeto que eles qualificam. As variáveis são indicadas pelo caractere de interrogação. Em sua especificação completa, a linguagem PDDL possui suporte à representação de números e quantidades, expressões aritméticas e equações, termos não-atômicos, quantificadores, definições dependentes do contexto, fluentes¹⁴ e planejamento hierárquico através do encadeamento de chamadas a ações), mas a representação que requer STRIPS em PDDL não possui este suporte e mesmo apesar disto, consegue representar bem os domínios.

Além de representar domínios, a linguagem PDDL permite a definição de problemas. Na análise do arquivo de problema escrito em PDDL, conforme Figura 23 a seguir, após a busca dos objetos pela instrução “(: OBJECT” e carga de todos os objetos em string plana de caracteres, isto é, retirados os caracteres de controles especiais, os objetos são tratados individualmente para encontrar seus nomes que serão posteriormente aplicados na tipagem das ações.

¹⁴ Fluentes são generalizações naturais de efeitos tradicionais; ao invés de especificar como os valores mudam, permitem especificar como os termos mudam.

Estudamos as características comuns dos arquivos de vários domínios e de diversos problemas escritos em PDDL com o requerimento de STRIPS, isto é, domínios não-tipados. Percebemos que ambos os arquivos nos dão ótimas pistas para a inferência de tipos, sendo necessário definir os tipos para todos os parâmetros de todas as ações e, também, definir os tipos dos objetos declarados no arquivo de problemas.

```
(define (problem strips-gripper-x-1)
  (:domain gripper-strips)
  (:objects rooma roomb ball4 ball3 ball2 ball1 left right)
  (:init (room rooma)
        (room roomb)
        (ball ball4)
        (ball ball3)
        (ball ball2)
        (ball ball1)
        (at-robby rooma)
        (gripper left)
        (gripper right)
        (at ball4 rooma)
        (at ball3 rooma)
        (at ball2 rooma)
        (at ball1 rooma)
        (free left)
        (free right))
  (:goal (and (at ball4 roomb)
              (at ball3 roomb)
              (at ball2 roomb)
              (at ball1 roomb))))
```

Figura 23. Arquivo em PDDL com o Problema Gripper de 4 bolas.

5.1.1. Inferência de Tipos de Parâmetros das Ações

Para inferir os tipos dos parâmetros das ações definidas no domínio, percebemos que toda ação possui uma definição clara das pré-condições necessárias para que seja uma ação válida e que estas definições começam sempre obrigando que determinado parâmetro seja de determinado tipo.

A partir desta observação, fizemos a separação dos termos unários do início das pré-condições para cada ação, isolando os tipos e os parâmetros, obedecendo ao pseudocódigo ilustrado na Figura 24, que, em seguida, alimenta uma mesma estrutura de dados, conforme a ilustração da Figura 25 onde podemos perceber a separação dos procedimentos, primeiramente isolando as ações e identificando os seus parâmetros, visto que cada ação pode ter um número diferente de parâmetros.

```

/* PSEUDOCÓDIGO - Descobre o Tipo dos Parâmetros das Ações */
Para cada ação ACT do domínio Faça
  Se ACT = VAZIO Então Abandone FimSe
  Para cada parâmetro PAR da ação Faça
    Se PAR = VAZIO Então Abandone FimSe
    Se pré-condições PRE = VAZIO Então Abandone FimSe
    Para cada PRE Faça
      Se PRE = VAZIO Então Abandone FimSe
      Se PAR = PRE Então TIPO = PRE FimSe
    FimPara
  FimPara
FimPara

```

Figura 24. Pseudocódigo para descobrir o TIPO de parâmetro das ações.



Figura 25. Tipagem de Parâmetros no domínio Gripper com STRIPS-PDDL.

Para cada parâmetro, buscamos um predicado unário na sentença de pré-condições, identificando o primeiro argumento unário como sendo o tipo do parâmetro. Se, porventura houvesse mais de um predicado unário com o mesmo parâmetro, considerávamos o primeiro como a definição do tipo. Por fim, armazenamos os tipos de parâmetros na respectiva estrutura de dados para o posterior confronto com os objetos a serem instanciados, como veremos adiante.

5.1.2. Inferência de Tipos de Objetos do Problema

Com a definição dos tipos de parâmetros das ações vistos na seção anterior, o domínio fica preparado para receber os objetos do problema, mas para a correta

instanciação, ainda é necessário conhecer o tipo do objeto que pretende ocupar o espaço do parâmetro. Para tanto, estudamos os arquivos de problemas em PDDL não-tipado, e percebemos que além da lista de todos os objetos utilizados constar diretamente em instrução própria para este fim, a situação inicial do mundo nos indica os tipos de objetos em seus argumentos unários e não-repetitivos.

Criamos então uma rotina para a definição dos tipos com base nestas observações, segundo o pseudocódigo apresentado na Figura 26, e ilustrado na sequência pela Figura 27 onde percebemos claramente a separação dos procedimentos, isto é, onde para cada objeto obtido da lista de objetos, ocorre uma busca por sentenças unárias na declaração de Situação Inicial do mundo. Ali, o arquivo em PDDL não-tipado apresenta sentenças que informam ao mundo que cada objeto listado pertence a uma categoria.

Verificamos em nosso estudo que estas categorias eram equivalentes aos tipos definidos para os parâmetros e passamos a considerá-las como sendo os próprios tipos dos objetos.

```

/* PSEUDOCÓDIGO - Descobre o Tipo dos Objetos do Problema */
Para cada objeto OBJ do problema Faça
  Se OBJ = VAZIO Então Abandone FimSe
  REPETE = FALSO
  Para cada situação inicial SIT Faça
    Se SIT = VAZIO Então Abandone FimSe
    Se SIT = UNÁRIO E NÃO REPETE Então
      TIPO = SIT
      REPETE = VERDADEIRO
    FimSe
  FimPara
FimPara

```

Figura 26. Pseudocódigo para descobrir o TIPO dos objetos.

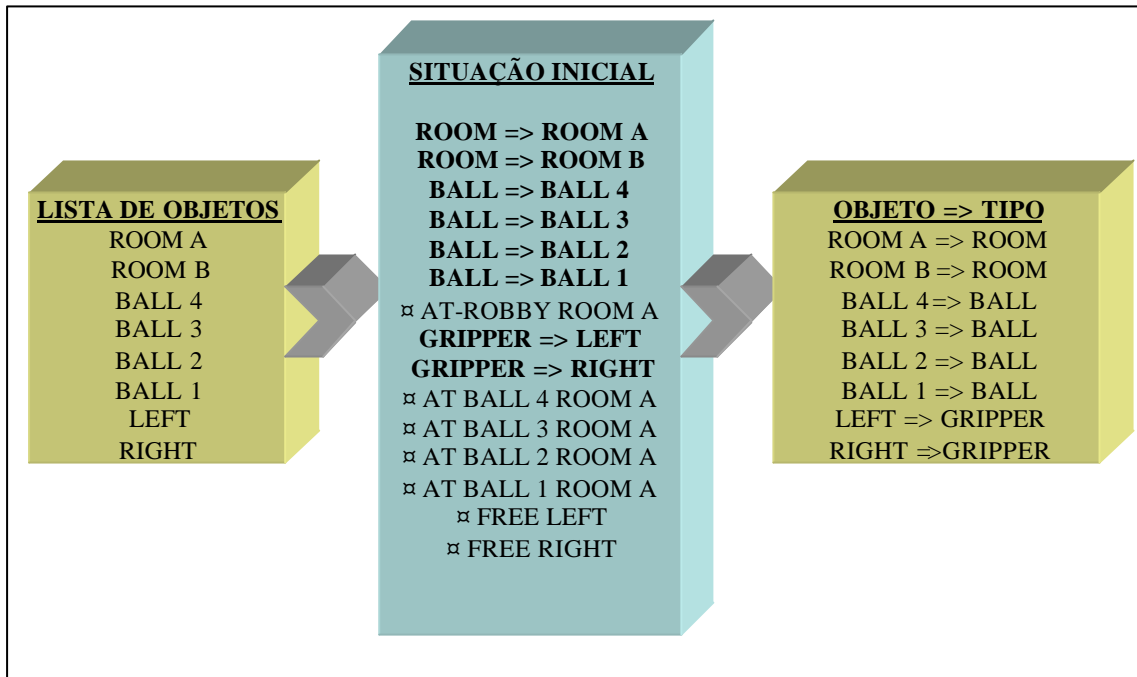


Figura 27. Tipagem de Objetos no Problema 01 do Gripper não-tipado.

Nesta situação, percebemos que algumas sentenças da Situação Inicial do Mundo não eram unárias, isto é, possuíam mais do que um argumento, devendo ser descartadas pois não interferiam na inferência dos tipos.

Em alguns domínios, como o Gripper por exemplo, também encontramos ambigüidade em sentenças como “Gripper Left” e “Free Left”, que indicam que no estado inicial, o objeto LEFT é uma garra e está livre. Sendo um conceito completamente válido, mas que precisou ser desconsiderado para evitar erro na interpretação do tipo do objeto.

Para estes casos, consideramos apenas as primeiras sentenças unárias válidas e, se determinado objeto já estivesse com o tipo inferido, as demais sentenças para aquele objeto seriam desconsideradas.

Com isto, evitamos em cerca de 95%, os casos em que alguma definição de sentença não apropriada possa aparecer antes, como por exemplo se no domínio Gripper, a sentença “Free Left” fosse definida antes da sentença “Gripper Left”, quando estaríamos inferindo que o objeto LEFT seria do tipo FREE ao invés de ser do tipo GRIPPER, que é o tipo correto. Para os casos identificados, foram feitas as inversões da ordem das sentenças no arquivo PDDL e o problema foi solucionado.

5.1.3. Instanciação das Ações

Agora, o trabalho de instanciação das ações ficou especializado com o tratamento dos tipos corretos, ocasionando a entrada de parâmetros apenas com os objetos dos tipos inferidos e evitando com isto a criação de ações inválidas.

Para instanciar as ações com objetos dos tipos apropriados, são analisadas separadamente as estruturas de dados do domínio e do problema que, a partir das inferências, já possui o tipo armazenado de cada parâmetro das ações e de cada objeto do problema, conforme ilustração abaixo:

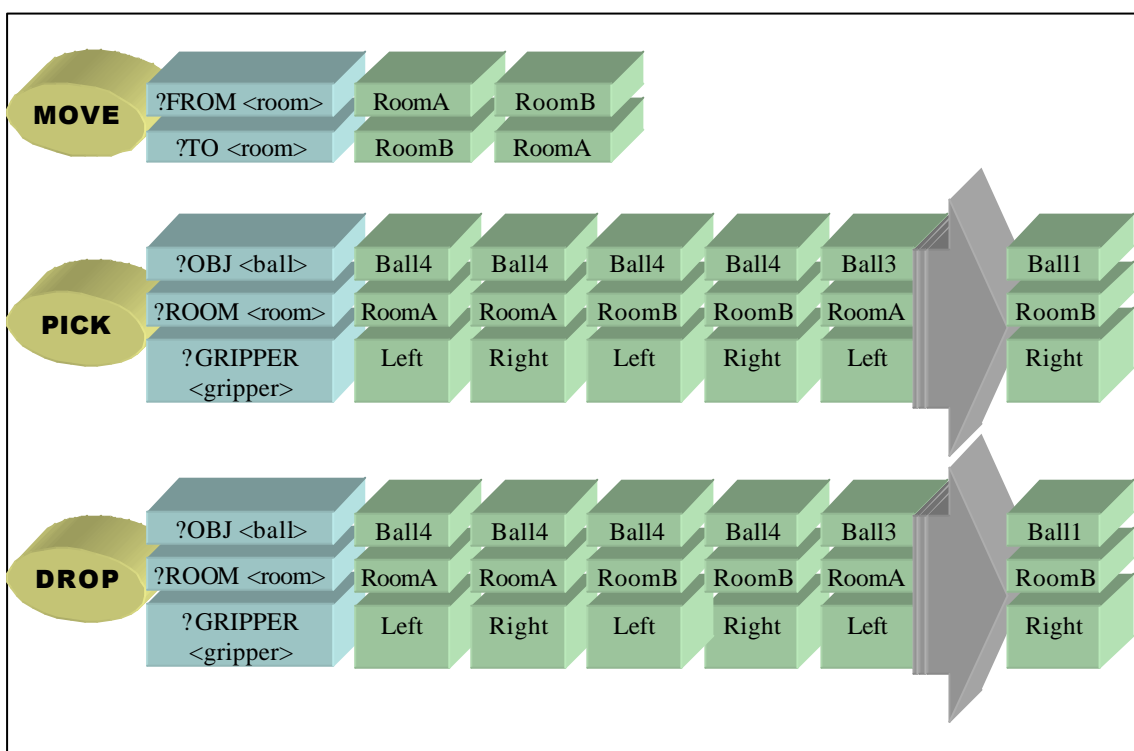


Figura 28. Instancição de Objetos Parametrizados em Ações.

Uma rotina então, conforme o pseudocódigo a seguir, unifica os processos de identificação dos tipos, fazendo a verificação do tipo do parâmetro da ação, com o tipo do objeto do problema, permitindo desta forma instanciar as ações com os objetos de tipos apropriados para o tipo do parâmetro da ação. Um cuidado adicional implementa uma rotina que armazena as informações em estruturas próprias para a elaboração da rotina que efetivamente fará a instancição, visto que o processo precisa evitar instancições

inválidas mesmo utilizando os tipos corretos, que podem acontecer em exemplos como no domínio Gripper, na ação MOVE, onde os seus dois parâmetros recebem objetos do tipo ROOM, poderia criar instâncias inválidas como MOVE ROOMA ROOMA ou MOVE ROOMB ROOMB.

```

/* PSEUDOCÓDIGO - Instanciar Ações */
Para cada ação ACT do domínio Faça
  Se ACT = VAZIO Então Abandone FimSe
  Para cada parâmetro PAR Faça
    Se PAR = VAZIO Então Abandone FimSe
    Para cada objeto OBJ Faça
      Se OBJ = VAZIO Então Abandone FimSe
      Se tipoPAR = tipoOBJ Então
        Instancia PAR com OBJ
      FimSe
    FimPara
  FimPara
FimPara

```

Figura 29. Pseudocódigo para instanciar ações.

Todas as ações instanciadas são listadas, armazenadas e recebem um importante número seqüencial. Toda ação numerada é uma ação válida e este objetivo reduz, dependendo do domínio, em muito o espaço das buscas para a composição de planos bem sucedidos no problema de planejamento em IA.

Ao final destas rotinas, será conhecido o número total de ações válidas instanciadas, que será determinante dos limites da variação aleatória do valor inteiro a ser usado nos cromossomos. Este valor será usado para o estabelecimento da correlação entre o número da ação e a sua efetiva representatividade no plano. São carregadas algumas estruturas adicionais que facilitarão o tratamento das ações através deste mesmo número.

Encerrada esta fase obrigatória de pré-processamento com a instanciación de ações com os tipos inferidos, gerando apenas ações válidas, o programa estará pronto para iniciar com os procedimentos genéticos remodelados que serão descritos na próxima seção.

5.2. NOVO MODELO GENÉTICO PARA PLANEJAMENTO

Na primeira abordagem, não existia consistência no processo de instanciación de ações, o que gerava um sério problema, ampliando exponencialmente o espaço de busca

das soluções e muito tempo de processamento era gasto para, no final do processo, identificar as ações como inválidas. Também, utilizávamos parte de um código encapsulado para a validação dos planos. O desempenho obtido desde então, não foi satisfatório, apesar da teoria levar com indícios a um caminho promissor.

Nesta nova abordagem, todos os procedimentos foram previstos e codificados no programa, desde a leitura e tradução dos arquivos PDDL, até o comportamento genético com o uso de operadores básicos e implantação de operadores especiais, como detalhado a seguir.

A mudança fundamental no procedimento genético aconteceu pela substituição do uso da biblioteca de domínio público PGAPACK, pela codificação própria do algoritmo genético em todas as suas etapas e procedimentos.

A partir do fato das ações estarem instanciadas com os objetos de tipos apropriados, constatamos a existência apenas de ações válidas e, portanto, seria desnecessário o controle de ações nulas ou NOP's, que não tinham efeitos sobre a situação do mundo, conforme modelagem anterior. Agora, todas as ações são operacionalizáveis, apesar de ainda poderem não ser executáveis em determinados momentos do plano, pois esta análise depende da análise da situação atual do mundo. Isto significa que uma ação pode aparecer no gene de número 10 e ser executável e reaparecer no gene de número 20, no qual através das alterações do mundo pelas ações antecessoras, poderá não atender às suas próprias pré-condições e será considerada como não-executável naquela posição.

O tamanho do cromossomo permanece fixo para manter o mesmo funcionamento e uma estrutura geral semelhante à utilizada na versão anterior. Mesmo sendo fixo, está parametrizável para ser possível alterar o tamanho ao refazer um teste. Mesmo utilizando apenas ações válidas, surge a necessidade do acompanhamento de cada gene, representado pelo número inteiro aleatório, entre um e o número de ações instanciadas, que é o índice do vetor de ações instanciadas, para identificar sua situação com relação a executabilidade, conforme a posição do gene no cromossomo.

A solução para esta situação inclui a criação de um cromossomo binário, paralelo ao de inteiros, para representar a condição de executabilidade de cada gene, no qual um valor zero (0) indica que a ação correspondente ao mesmo índice no cromossomo de inteiros não é executável naquela posição e, portanto não tem efeitos sobre o mundo, o que equivale a um NOP, e que o valor um (1) representa uma ação que é executável naquela posição e, conseqüentemente afetará a situação atual do mundo.

Para descobrir se uma ação é executável ou não, o programa simula a execução do plano, a partir do cromossomo aleatório e da situação inicial do mundo, proposta pelo arquivo PDDL. Quando uma ação é executável, significa que suas pré-condições podem ser atendidas pela situação atual do mundo e, então executa a rotina de alteração do mundo atual através dos seus efeitos, criando listas de inclusões e exclusões de estados, para que a próxima ação seja verificada nesta nova situação do mundo.

Pelo fato do número inteiro ser gerado aleatoriamente, como forma de evitar uma configuração de muitas ações não-executáveis, foi criado um controle denominado *fator de executabilidade*, também chamado no programa, de *fator de compactação* por compactar e agrupar os blocos de ações executáveis, que pode estar habilitado ou não na tela de Parâmetros Genéticos, com um percentual que pode ser configurado a cada execução, mas que parte de um padrão de 50%, isto é, garante que, pelo menos a metade das ações geradas em cada cromossomo seja executável. Este parâmetro pode receber valores entre 0 e 100 e força o programa a gerar, nestes percentuais, os genes executáveis. Isto tem um peso direto na eficiência da rotina, visto que um percentual muito alto pode forçar o programa a repetir a geração e teste das ações até que encontre uma ação executável.

A Figura 30, a seguir, demonstra as representações utilizadas para um cromossomo que simboliza um plano com tamanho de até 20 ações. Os índices das posições do vetor indicam a ordem de execução das ações (genes), que aparecem numeradas com os valores inteiros para indicar o número da ação no vetor de ações instanciadas. Em paralelo existe o cromossomo de binários que indica quando a ação correspondente é (1) ou não (0) executável.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
7	1	2	23	7	12	1	25	30	2	15	4	1	1	22	6	33	2	1	22
1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	1

Figura 30. Representação de um cromossomo para um plano de até 20 ações.

Quanto à questão da geração da população inicial, consideramos para a codificação, a geração aleatória e os valores dos genes escolhidos aleatoriamente entre um e o número de ações válidas instanciadas para o problema, não sendo, portanto fruto

de operações genéticas, obedecendo aos parâmetros configurados para a execução, tais como tamanho do cromossomo, da população e número de ações instanciadas.

Ainda na população inicial aleatória, cada indivíduo gerado, representado por um cromossomo de inteiros e outro paralelo de binários, recebe uma pontuação referente à avaliação de sua adequabilidade ao problema, conforme detalhamento apresentado mais adiante, conhecido como *valor de fitness*. Este valor classifica os indivíduos de uma população para a aplicação dos critérios genéticos da teoria evolucionista, onde os indivíduos melhor adequados tendem a permanecer nas próximas gerações, conforme foi detalhado no capítulo 3.

Ao final da geração inicial, são armazenados os valores do maior, do médio e do menor *fitness* daquela população. Estes dados ficam armazenados em banco de dados próprio, para futuras consultas e relatórios ou, apenas, em memória e tela para acompanhamento da evolução.

A fase genética de *Seleção* de indivíduos para a população escolhe os cromossomos como números inteiros que indicam sua posição seqüencial na população, com base nos seus valores de *fitness*. Nesta fase estamos usando o método conhecido como “Roleta” ou ainda como “Torneio Probabilístico” [22],[29]. Através de sorteio aleatório da roleta, separamos um, entre os valores limites da somatória de *fitness*, enquanto é executado um laço para soma parcial dos valores de *fitness* de cada indivíduo, até ultrapassar o valor selecionado ou encontrar o final da população, conforme ilustração da Figura 31, abaixo. Conforme o critério escolhido para elitismo, que pode ser configurado na tela de parâmetros iniciais, se solicitar por seleção elitista, garantirá que o indivíduo selecionado tenha, no mínimo, um valor de *fitness* maior do que a metade do valor médio da população.

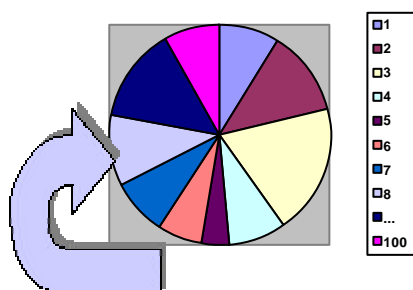


Figura 31. Representação do processo genético de Seleção por Torneio Probabilístico (Roleta).

Por considerarmos que a operação de *Cruzamento* em dois pontos não oferece uma vantagem significativa para o problema em estudo, optamos pelo uso do operador convencional de Cruzamento que estabelece um ponto de corte aleatório de cromossomos pais, gerando uma recombinação das partes em cromossomos filhos, conforme ilustrado na figura Figura 8 deste trabalho. A taxa percentual deste operador é configurada na tela de parâmetros iniciais e direciona o número de operações de cruzamento que acontecem em cada geração. Na mesma tela, podemos optar por manter os pais cujas avaliações forem superiores a dos filhos gerados pelo cruzamento, como uma forma de aumentarmos o critério de elitismo.

Permanece o operador de *Mutação* convencional, aplicado aos indivíduos envolvidos na operação de cruzamento, com a diferença que os valores possíveis para a mutação estão delimitados por um (1) e pelo número de ações instanciadas. Sua taxa percentual é igualmente parametrizável e determina o número de mutações que acontecerão a cada geração.

Criamos um operador genético especial chamado operador de *Inversão*, que permite inverter um cromossomo dentro de uma taxa percentual informada na tela de parâmetros genéticos. Esta operação permite uma nova avaliação de um cromossomo gerado no processo normal, onde a ordem invertida dos genes possa ser mais bem aproveitada na análise de executabilidade. Demonstrou não ser um operador relevante para os resultados analisados e foi mantido como um recurso opcional.

Ao final do processo de geração da população, se estiver marcado para verificar a estabilidade do AG, o programa executa uma rotina que testa se não existe uma diferença entre os máximos encontrados nas gerações de até um valor informado na tela de parâmetros, para as últimas 5 gerações, o que configura uma situação de estabilidade do AG e torna o programa mais flexível, procurando ampliar o fator de cruzamento em mais 1%, até o limite de 100%, retornando a crescer imediatamente a partir de 50%, e o de mutação e inversão em mais 3%, que crescem até o limite de 50%, quando automaticamente retornam para 1% e recomeçam a crescer.

Esta técnica pretende ampliar a diversidade biológica da população que se tornou estável, devido ao fato do programa ficar preso a um máximo local dentro do espaço de busca de estados. Com o passar das gerações, esta condição também atua desligando alguns controles do processamento elitista, como mostrado na Tabela 14 a seguir.

Geração	Controles que são Desativados
50%	Manter Pai2 melhor do que Filho1
60%	Manter Pai1 melhor do que Filho2
70%	Manter Pai2 melhor do que Filho2
80%	Manter Pai1 melhor do que Filho1
90%	Usar Estratégia Elitista

Tabela 14. Controle de Estabilidade desativa controles conforme andamento das gerações.

Se ocorrer uma evolução significativa, como por exemplo, alcançar mais um objetivo proposto pelo problema, o programa identifica que saiu da condição de estabilidade e, automaticamente os controles são reconduzidos aos seus valores originais parametrizados.

Ao solicitar o *Uso de uma Estratégia Elitista*, no parâmetro da tela inicial, o programa passa a garantir que o melhor indivíduo de uma geração seja copiado para a próxima geração. Também garante que o processo de Seleção escolha um indivíduo com valor de avaliação maior do que a metade da média dos máximos. Ainda garante, dentro de um valor percentual interno do máximo *fitness*, que se o filho de um cruzamento gerar avaliação igual a zero, seja substituído imediatamente pelo melhor indivíduo da geração anterior. O uso desta estratégia elitista demonstrou através dos testes, ser de grande relevância para os resultados.

Para o cálculo do valor de *fitness* de um indivíduo, o programa aplica três controles interdependentes que possibilitam a valorização de três fatores importantes da adequabilidade ao problema. São eles:

- ? O primeiro fator, chamado de *Peso 1* ou *P1*, valoriza o cromossomo que possui o maior intervalo entre ações executáveis, isto é, aquele cuja primeira ação executável está nos primeiros genes do cromossomo e que a última ação executável posiciona-se nos últimos genes, indicando um bom potencial para operações de cruzamentos. É importante lembrar que este cálculo somente acontece quando o número de ações executáveis daquele cromossomo é maior do que zero. Por padrão, é atribuído um valor fixo em 10 (dez) para este peso, podendo ser alterado para qualquer outro valor fixo ou variável. A seguinte expressão é aplicada, onde *P1*, *P2* e *P3* representam os valores de *Peso 1*, *Peso 2* e *Peso 3* respectivamente, *NUlt* e *NPri* representam o número da última e da primeira ação executável, e *Nex* o número total de ações executáveis.

$$(P1 / (P1 + P2 + P3)) * ((NUlt - NPri + 1) / Nex)$$

- ? O segundo fator, chamado de *Peso 2* ou *P2*, valoriza o cromossomo que possui o maior número de ações executáveis, isto é, aquele que, independente do Fator de Executabilidade, gerou o maior número de ações ordenadas executáveis em relação ao tamanho total do cromossomo, possuindo grande potencial de evolução por cruzamento ou mutação. Por padrão, é atribuído um valor fixo em 20 (vinte) para este peso, podendo ser alterado para qualquer outro valor fixo ou variável. Na sua expressão de representatividade abaixo, LChrom representa o tamanho total do cromossomo.

$$(P2 / (P1 + P2 + P3)) * (Nex / LChrom)$$

- ? O terceiro fator, chamado de *Peso 3* ou *P3*, valoriza o cromossomo que possui o maior número de objetivos alcançados em relação ao número total de objetivos, isto é, aquele que, independentemente do tamanho do plano gerado, alcançou o maior número de objetivos no estado final do mundo, possuindo grande potencial de evolução por ser um ótimo pai na operação de cruzamento e ter grande potencial para que, através de mutação possa alcançar outros objetivos. Por ser considerado o peso mais importante, recebe como padrão, um valor fixo em 70 (setenta), podendo ser alterado para qualquer outro valor fixo ou variável. Na expressão abaixo, Nob representa o número de objetivos alcançados por aquele cromossomo, e TObj o número do total de objetivos a alcançar para o problema em curso.

$$(P3 / (P1 + P2 + P3)) * (Nob / TObj)$$

Para que o valor resultante das expressões acima acumuladas fique entre os limites de zero (0) e um (1), como é praxe em algoritmos genéticos, e considerando a possibilidade de configuração de pesos variáveis, é necessário um procedimento matemático para a Normalização dos Pesos, de maneira linear, de acordo com o andamento das gerações. Para tanto, criamos a expressão abaixo que é aplicada a todo o

peso variável, onde *PesoIni* e *PesoFim* indicam os valores limites para o peso, *Gerac* representa a geração atual e *GenSize* é o número total de gerações.

$$\text{PesoIni} * (1 - \text{Gerac} / \text{GenSize}) + \text{PesoFim} * (\text{Gerac} / \text{GenSize})$$

O valor de *fitness* final, calculado para cada indivíduo é, portanto a soma dos três fatores calculados e normalizados conforme expressões acima. Com base nele, todo o programa funciona de maneira a manter os melhores valores e descartar os piores para as gerações seguintes. O resultado de todo este processo chama-se *Evolução*, que é mais bem visualizada através da apresentação dos principais resultados de testes aplicados e readaptações no código, detalhados na próxima seção.

5.3. RESULTADOS E ADAPTAÇÕES NO MODELO

De um total de cerca de 900 testes executados, cerca de 600 estão armazenados no banco de dados. Em certos casos, foi necessário interromper o teste para ajustar parâmetro ou código. Em outros casos, o teste foi interrompido por problemas de travamento do computador por esgotamento dos recursos de memória. Entretanto, a grande maioria dos testes armazenados é relevante para o estudo proposto e encontra-se detalhada no apêndice B.

Todos os testes foram executados em um computador Pentium IV, 1.6 Ghz, com 512 MB de memória RAM e discos que permitem o armazenamento de até 120 GB, à rotação de 7200 rpm. A partir do teste armazenado sob o número 76 é que foram iniciadas as baterias automáticas de testes e, a partir do número 333 a configuração para uso de memória virtual pelo Windows foi alterada, de automática, para um total, em disco, de 12 GB manuais, significando que o sistema passou a não precisar dedicar tempo de processamento para gerenciar memória, reduzindo os casos de travamento por esgotamento de memória.

Também como medida paliativa de controle e gerenciamento de memória, foi baixado pela Internet¹⁵ e instalado o pacote shareware do programa gerenciador de memória '*MemDefrag 2*', que permite entre outras tarefas, liberar espaço ocioso de

¹⁵ Site para download: http://www.brothersoft.com/Utilities/Optimize/Utilities/MemDefrag_17590.html

memória. Todos os demais testes, a partir de então, foram executados sob esta configuração que nos permite uma análise sobre o desempenho e funcionalidade do Algoritmo Genético aplicado aos problemas gerais de Planejamento.

A proposta de modelagem genética inicial aplicada a problemas de planejamento, apresentada na seção 4.1, mostrou que a aplicação de operadores genéticos convencionais sobre o modelo é insuficiente para a obtenção dos resultados esperados na geração dos planos que resolvem a conhecida e genérica classe de problemas de planejamento, conforme justifica a análise dos resultados desta implementação, melhor detalhada na seção 4.3. Este conhecimento serviu como base para a elaboração da nova implementação de modelagem genética, proposta pela seção anterior, em cujos testes pudemos identificar algumas novas situações-problema que direcionaram as alterações e adaptações do modelo, bem como a criação de novos operadores genéticos não-convencionais.

A mudança de linguagem e de ambiente operacional obrigou a primeira adaptação do modelo, para que todo o tratamento feito pelo código das bibliotecas de domínio público fosse agora elaborado e tratado por código próprio. Neste processo de adaptação do código, à medida que a remodelagem básica estava sendo concluída, eram executados alguns testes para o reconhecimento do comportamento genético sobre os problemas e domínios já trabalhados na versão anterior. A análise dos resultados obtidos até então nos levou a adaptar o modelo às necessidades identificadas pelos testes.

Por se tratar de uma geração aleatória de genes que representam as ações, acontece em grande parte dos casos que o gene não represente uma ação que possa ser executada naquela posição do plano. Criamos um controle para garantir uma taxa percentual de ações executáveis sobre um plano. O sistema passou a verificar as pré-condições de todas as ações geradas para alimentar a um vetor paralelo de executabilidade, cuja análise vai gerar um índice importante de avaliação do plano.

Um plano com muitas ações não-executáveis torna-se um plano mal avaliado para o sistema, mas que possui ainda um bom potencial genético, visto que uma operação básica de mutação ou cruzamento pode representar uma completa remodelação do estado do mundo e tornar aquele plano mais bem avaliado.

A informação na tela de parâmetros genéricos, ativa ou desativa este controle. Quando ativado, permite fornecer a taxa percentual em que o mesmo deve ocorrer. Ao colocar uma taxa muito baixa, o comportamento do programa não é muito alterado e não são verificadas variações significativas de desempenho. Com taxas muito altas, o

desempenho cai visivelmente, devido ao fato de necessitar gerar tantas quantas forem as ações equivalentes à taxa e testá-las quanto a sua executabilidade, para garantir que sejam executáveis dentro do percentual solicitado.

Concluimos que o tamanho do cromossomo é um dos fatores de importância fundamental para a solução do problema, visto que se for subestimado não permite a criação de planos extensos o suficiente para obter o estado-meta, e se for superestimado criará planos muito extensos que, por vezes estarão com muitas ações repetitivas e de efeitos inversos, apenas para preencher o tamanho do plano. Identificamos este comportamento indesejado na geração de planos muito longos, quando ações aleatórias acontecem para desfazer um objetivo que já estava alcançado. Também, um alto percentual solicitado no controle de taxa de executabilidade obriga, por vezes, que o plano crie esse tipo de ações indesejadas para o plano.

Um exemplo disto acontece quando, na situação inicial do Gripper, o robô se encontra no RoomA, uma ação “MOVE RoomA RoomB” acontece e muda o estado do mundo fazendo com que o robô esteja presente no RoomB. Esta é a pré-condição necessária para acontecer a ação de retorno “MOVE RoomB RoomA” que, por sua vez, faz com que o robô esteja no RoomA. E assim sucessivamente, um plano válido pode ficar movendo o robô de um quarto para outro, isto é, um plano pode ser gerado com um número muito grande de ações do tipo que reverte os efeitos de outra ação. O mesmo exemplo pode ser aplicado a outras ações do domínio, como pegar e soltar uma bola no domínio Gripper, carregar e descarregar um caminhão no domínio Logistics e assim por diante. Acreditamos que um sistema planejador precisa evitar a ocorrência deste tipo de ações e que este problema precisa ser tratado e resolvido para as próximas versões deste planejador, não existindo até o momento, entretanto, uma solução trivial para o mesmo.

Procuramos ainda algumas formas alternativas de tratamento de problemas com novas adaptações no código. Uma tentativa neste sentido pode ser exemplificada com o que batizamos de análise reversa do plano, onde a última ação acontece por primeiro, a penúltima acontece por segundo e assim sucessivamente até que a primeira ação seja analisada por último no plano, com as mesmas medidas de qualidade obtidas na avaliação seqüencial normal dos planos. Com este controle, passamos a ter uma forma secundária de analisar um mesmo plano. Entretanto, foi observado durante os testes que, este fator não afetou os resultados de maneira significativa, mas optamos por manter o controle implementado de maneira opcional.

Um grande potencial de adaptação aos problemas encontrados foi implementado no código com o uso de cromossomos de tamanho variável, sendo que o sistema foi adaptado para este suporte e ainda, para permitir que o tamanho seja definido na medida em que se vai necessitando ampliar o plano representado pelo cromossomo.

A Tabela 15 apresenta um resumo das principais características e potencialidades da versão mais atual do nosso planejador genético. Todos os parâmetros genéticos estão descritos nesta tabela que apresenta também um valor padrão de inicialização que pode ser alterado no programa pelas guias da tela, vistos na Figura 33.

Característica	Descrição	Padrão
Nº. de Gerações	Cada geração apresenta os resultados do processo genético evolutivo. Representa o número de evoluções pretendidas	100
Tamanho da População	Uma geração é representada por uma população com um determinado número de indivíduos. Representa a diversidade	100
Nº. de Semente Randômica	A semente randômica comanda a geração interna de números aleatórios e permite repetir os resultados obtidos	5
Tamanho do Cromossomo	Um indivíduo é representado por um cromossomo, cujo tamanho inicial é fornecido e adaptado durante a execução	30
Taxa % de Cruzamento	Operador genético convencional de Cruzamento acontece para grande parte da população, determinada por esta taxa	80%
Taxa % de Mutação	Operador genético convencional de Mutação garante maior diversidade e acontece nos indivíduos segundo esta taxa	3%
Percentual de Inversão	Parte dos cromossomos são gerados no sentido inverso do original, conforme esta taxa percentual	3%
Estratégia Elitista	Possibilidade de direcionamento de resultados evolutivos, desconsiderando os piores resultados	Sim
Fator de Executabilidade	Taxa percentual que indica o número médio de ações executáveis que serão geradas nos planos aleatórios	50%
Controle de Estabilidade	Este controle identifica a situação de estabilidade do AG por diferença de taxa média e promove alterações automáticas em parâmetros genéticos para escapar de máximos locais	Não 0,1
Análise Reversa	Análise reversa e cálculo de adaptabilidade de um cromossomo acontecem a um percentual da população indicado nesta taxa	Sim 30%
Após o Cruzamento	Estratégia elitista para controle da evolução gerada pelo operador convencional de cruzamento. Permite manter os pais melhor avaliados que os filhos gerados na operação.	Mantém Pai1 Melhor que Filho1
Peso 1	Valoriza o cromossomo que possui o maior intervalo entre ações executáveis, pode ser fixo ou variável	Fixo 10
Peso 2	Valoriza o cromossomo que possui o maior número de ações executáveis, pode ser fixo ou variável	Fixo 20
Peso 3	Valoriza aquele que possui o maior número de objetivos alcançados em relação ao número total. Fixo ou variável.	Fixo 70

Tabela 15. *Resumo das principais características e potencialidades do GAPlan.*

Todas as adaptações representam importantes avanços no produto desenvolvido, mesmo apesar de não demonstrarem ganhos aparentes de desempenho. O

funcionamento fica mais flexível e tende a ser válido para uma quantidade maior de domínios e problemas. No próximo capítulo, abordamos as principais contribuições obtidas como fruto do desenvolvimento deste trabalho, bem como fazemos as indicações e orientações para a continuidade em trabalhos futuros relacionados aos temas ora abordados.

6. CONCLUSÕES E TRABALHOS FUTUROS

A computação evolutiva representa a iniciativa de implementar em computadores as regras simples do processo evolutivo onde os operadores genéticos são responsáveis pela implementação dos processos de busca e aplicam variações aleatórias aos indivíduos de uma população, sujeitos ao processo de seleção natural sob recursos limitados. Os exemplares localizados em regiões promissoras do espaço de busca representam os indivíduos mais adaptados que sobrevivem e se reproduzem, propagando seu material genético às próximas gerações.

Assumindo que o tamanho da população é fixo ao longo das gerações, então os indivíduos mais adaptados da geração atual terão maior chance de transmitir seu código genético para a próxima geração. Existe, portanto, um compromisso entre mecanismos de exploração global do espaço de busca (manutenção de diversidade na população) e de exploração local das regiões promissoras detectadas (aplicação de operadores genéticos em conjunto com processos de seleção baseadas no nível de adaptação de cada indivíduo).

Existe uma preocupação constante em mostrar que já foram criados algoritmos superiores aos algoritmos evolutivos para resolver certos tipos de problemas. No entanto, para se chegar a estas soluções dedicadas e muito eficientes, muitos esforços foram devotados e as técnicas até então elaboradas estão fortemente vinculadas a conhecimentos específicos associados a cada problema, ou seja, representam soluções dedicadas.

Nossa preocupação nunca foi em relação ao desempenho, visto que optamos por uma plataforma de desenvolvimento em linguagem visual de programação, que permitisse o completo entendimento do processo. Foi, portanto, mais importante entender, visualizar, quebrar barreiras, implementar e poder testar o funcionamento de nosso planejador genético do que a preocupação com o desempenho de competições de planejadores. Um trabalho futuro que certamente renderá os frutos da excelência deve buscar a implantação de recursos para a melhoria do desempenho sobre as plataformas operacionais de competição.

As importantes contribuições deste trabalho passam pela apresentação de um resumo do estado da arte sobre duas das mais importantes áreas da Inteligência Artificial, a proposta de união destas áreas de maneira inédita, o entendimento e a reescrita do código das bibliotecas de domínio público, a apresentação de uma maneira mais

simplificada de inferir tipos de parâmetros e objetos e para instanciar ações. Por fim, ressalta-se o desenvolvimento em duas plataformas e modelagens distintas de sistemas capazes de resolver alguns problemas de planejamento utilizando a técnica de algoritmos genéticos, conforme proposta original. Ressaltamos o valor da pesquisa que, mesmo sem obter sucesso para todos os casos, abre os caminhos para o desenvolvimento da área e direciona para os novos trabalhos a serem desenvolvidos seguindo a mesma proposta.

Os resultados obtidos indicam um caminho promissor com um bom potencial de obtenção de resultados positivos quando aplicados ao planejamento. Entretanto, as diversas tentativas de adaptação do modelo e de ajustes de um enorme número de parâmetros trouxeram fatores impeditivos do alcance do sucesso das implementações. Concluimos ainda pela necessidade de mais trabalho de pesquisa para a consolidação da proposta, mas com grande parte do caminho já trilhada.

O processo genético evolutivo aplicado a problemas simples dos domínios mais conhecidos alcança os resultados esperados, mas sobre um complexo espaço de estados, necessita de mais ajustes de parâmetros e do desenvolvimento de operadores auxiliares pouco triviais. A complexidade computacional do conjunto analisado abrange além da complexidade do problema de planejamento a capacidade de reconhecer e tratar diversos domínios, aliada ao ajuste correto de um grande número de parâmetros genéticos necessários para o funcionamento do modelo.

Estudos paralelos apontam para a aplicação de técnicas como a de Grafo de Planos na geração da população inicial do AG, fazendo com que a busca aconteça sobre um espaço ainda mais reduzido e direcionado desde a primeira população. Um modelo que implemente este Grafo de Planos e apresente novos operadores genéticos representa certamente uma boa proposta de trabalho futuro em continuidade a esta pesquisa.

Apesar de, atualmente, não se mostrar competitivo no tempo de processamento, em relação aos planejadores atuais, demonstrou ser um processo de fácil adaptação à diversidade de domínios e problemas, devendo ser mais bem aproveitado em situações-problema onde os demais sistemas de planejamento não obtiveram sucesso.

Algumas propostas de trabalhos futuros podem aproveitar o atual ponto de desenvolvimento e conclusões para dar continuidade ao processo, principalmente em:

- ? gerar a população inicial pré-direcionada por outra técnica;
- ? definir nova função de fitness e criar outros operadores genéticos especiais;
- ? fornecer suporte a números e equações matemáticas em PDDL;

- ? suportar ações com medidas temporais e planejamento hierárquico;
- ? migrar para plataforma de competição buscando a melhoria do desempenho.

APÊNDICE

Neste anexo, apresentamos as principais características de manipulação do sistema GAPlan com o objetivo de que um usuário comum seja capaz de executar e obter seus resultados de maneira mais direcionada e fácil. A próxima seção define a interface utilizada com suas nuances e particularidades de uma aplicação visual em plataforma gráfica. Os tópicos e telas são apresentados em seqüência padrão de execução para a obtenção dos gráficos e resultados para análise. Os principais resultados que fundamentaram as adaptações do modelo são apresentados na seção B deste apêndice.

APÊNDICE A - GAPLAN – UM PLANEJADOR GENÉTICO

O novo programa foi desenvolvido na linguagem de programação *MS-Visual Basic 6.0 – Professional*, o mecanismo de banco de dados é o *MS-JET OLEDB.4.0* com acesso a bases com *ADO – Activex Data Objects*, e gerador de relatórios *Crystal Reports 8.0*. Os gráficos em tela são apresentados pelo uso do controle *MSChart*. O ambiente operacional utilizado foi o *Windows XP – Professional Edition*.



Figura 32. Tela ¹⁶ 'Splash' enquanto inicializa e abre conexão com banco de dados.

Ao ser iniciado, devido à demora nos procedimentos de inicialização de variáveis, criação de estruturas e, principalmente, na criação da conexão aos serviços de banco de

¹⁶ A imagem ao fundo da tela ilustra a famosa obra do pintor, escultor e artista gráfico catalão surrealista Salvador Dalí: "A Persistência da Memória", de 1931.

dados, visto que a tabela com as gerações armazena até o momento, algo em torno de 30 mil registros, é exibida uma tela meramente introdutória e informativa, conforme pode ser vista na Figura 32.

Automaticamente, ao final das inicializações, o programa apresenta tela principal com uma divisão em cinco partes, como guias de tabulação. As divisões das partes são as guias para Parâmetros Genéticos, Planejamento, Execução, Bateria de Testes e Resultados, sendo que a primeira começa aberta. A parametrização dos elementos variáveis passou a ser representada em tela respectiva para configuração dos valores, ao invés de linha de comando usada anteriormente, conforme demonstra a Figura 33, a seguir. Uma configuração inicial, com os dados mais utilizados, é iniciada como padrão.

Estão disponíveis para uso imediato o botão de relatórios e as demais guias, que apesar disto, fazem sentido apenas para a configuração de um novo teste ou baterias, exceto a opção de visualização de resultados para consulta e visualização gráfica dos dados armazenados até aquele momento no banco de dados do sistema.

Figura 33. Tela inicial do programa – Guia Parâmetros Genéticos.

Os parâmetros foram separados em grupos menores para a organização da tela. No grupo de Parâmetros Maiores, estão posicionados os campos para a entrada dos dados de Tamanho da População, Número de Gerações e Semente Randômica. Como Parâmetros Menores estão os campos para Tamanho do Cromossomo, Percentuais de Cruzamento, Mutação e de Inversão, que é um operador genético especial e que será detalhado na próxima seção deste capítulo. Algumas variáveis qualitativas estão previstas para a escolha do Método de Geração da População Inicial, do Método da Seleção para Reprodução, da Seleção para o Cruzamento e uso de Sobre-Seleção, ainda se pretende usar Medida de Ajuste Opcional e se pretende que algumas rotinas genéticas trabalhem com uma Estratégia Elitista. Algumas variáveis qualitativas ainda não foram implantadas, como Sobre-Seleção e Medida de Ajuste Opcional, bem como as variações para os outros controles deste grupo, entretanto existe o espaço preparado para uma futura implementação de melhorias e possibilidades de maiores ajustes ao programa. Quanto ao grupo de Cálculo, existe a seleção do tipo de cruzamento uniforme de um ponto, visto que a opção para cruzamento uniforme de dois pontos está desabilitada e a possibilidade de aumentar o rigor do parâmetro elitista, onde podemos direcionar os resultados das operações de cruzamento, como em manter os pais, em caso dos filhos resultantes de um cruzamento não produzirem maiores valores de avaliação.

Em Parâmetros Especiais, podemos ativar e desativar o Fator de Compactação do Cromossomo que é declarado na forma de percentual, isto é, entre 0 e 100, e indicam que serão geradas ações executáveis no cromossomo com este fator de ocorrência. Por exemplo, o padrão é ativado em 50% o que significa que os cromossomos deverão possuir, no mínimo, 50% de ações executáveis em sua composição. É também referenciado neste trabalho como Fator de Executabilidade. Também é possível habilitar ou desabilitar a Análise de Estabilidade do AG, que significa para o programa poder identificar quando as gerações não estão conseguindo evoluir, e o valor informado para este parâmetro serve para identificar a diferença entre os valores de melhor avaliação que não mudam há mais de 5 gerações. Um *flag booleano* de estabilidade controla a partir da primeira situação de estabilidade, uma geração elitista a outra não, incrementando o percentual de cruzamento em 1% e mutação em 3% a cada geração, numa tentativa de aumentar a diversidade biológica da população que ficou estável e, com isto, tentar escapar no espaço de busca de um máximo local. Os Pesos, K1, K2 e K3, podem ser fixos ou variáveis. Se forem fixos, precisam ser informados apenas os primeiros valores de cada grupo e se forem variáveis, também devem ser informados os outros valores do

grupo, indicando os iniciais e finais para cada peso conforme o andamento percentual das gerações.

No subgrupo de Memória do Teste é possível indicar se pretende que os dados do teste sejam armazenados em Banco de Dados e se permite a execução de músicas ao fundo que podem ser escolhidas na lista abaixo disponibilizada.

A próxima guia de tabulação permite ao usuário selecionar um Domínio e um Problema de Planejamento, conforme mostra a Figura 34, a seguir.

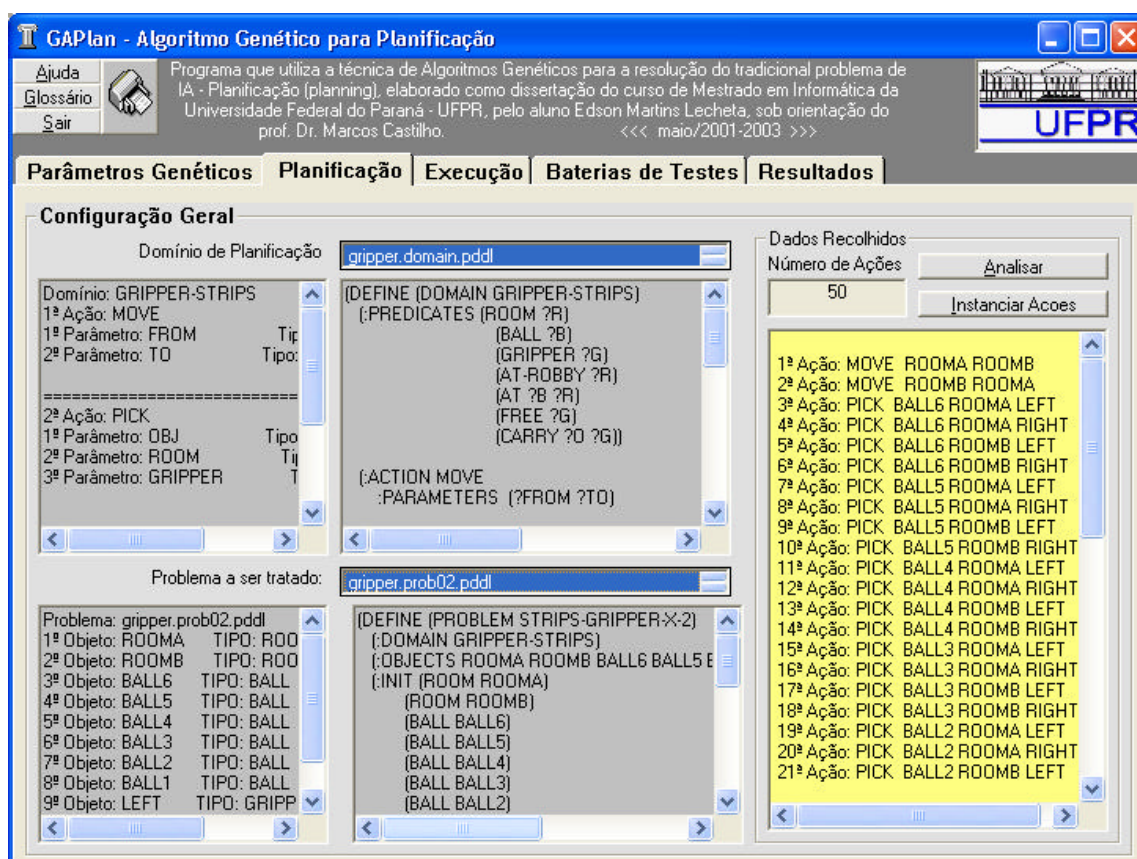


Figura 34. Tela de escolha de Domínio e Problema na Gui a planejamento.

Ao selecionar um domínio na lista de Domínios de Planejamento, o programa carrega automaticamente o primeiro arquivo de problema na lista de Problema a ser Tratado, podendo o usuário selecionar um problema diferente dentre os que estiverem listados neste controle. Importante perceber que na medida em que o domínio é selecionado, automaticamente inicia o processo de reconhecimento deste arquivo para o trabalho de identificação dos tipos de dados dos parâmetros das ações. Para poder completar esta análise, é necessário que exista um arquivo de problema também selecionado. Pelo arquivo de problemas, o programa identifica na seção “(: INIT” o tipo

dos objetos e através da identificação dos tipos de parâmetros das ações do domínio, automaticamente são instanciadas as ações com os tipos apropriados, gerando um rol apenas de ações instanciadas possíveis de serem executadas. Esta operação está detalhada no item 5.1 deste trabalho.

Além do vetor de ações instanciadas produzido nesta fase, outro controle fundamental é o número de ações instanciadas, mostrado nesta tela, que norteará todo o processo genético limitando os valores inteiros de genes entre 1 (um) e este número.

Configurados os Parâmetros Genéticos e feita a escolha do Domínio e Problema de Planejamento, o programa já está preparado a esta altura para executar o procedimento genético conforme descrito a seguir e que pode ser acompanhado pelos painéis informativos vistos na Figura 35, a seguir.

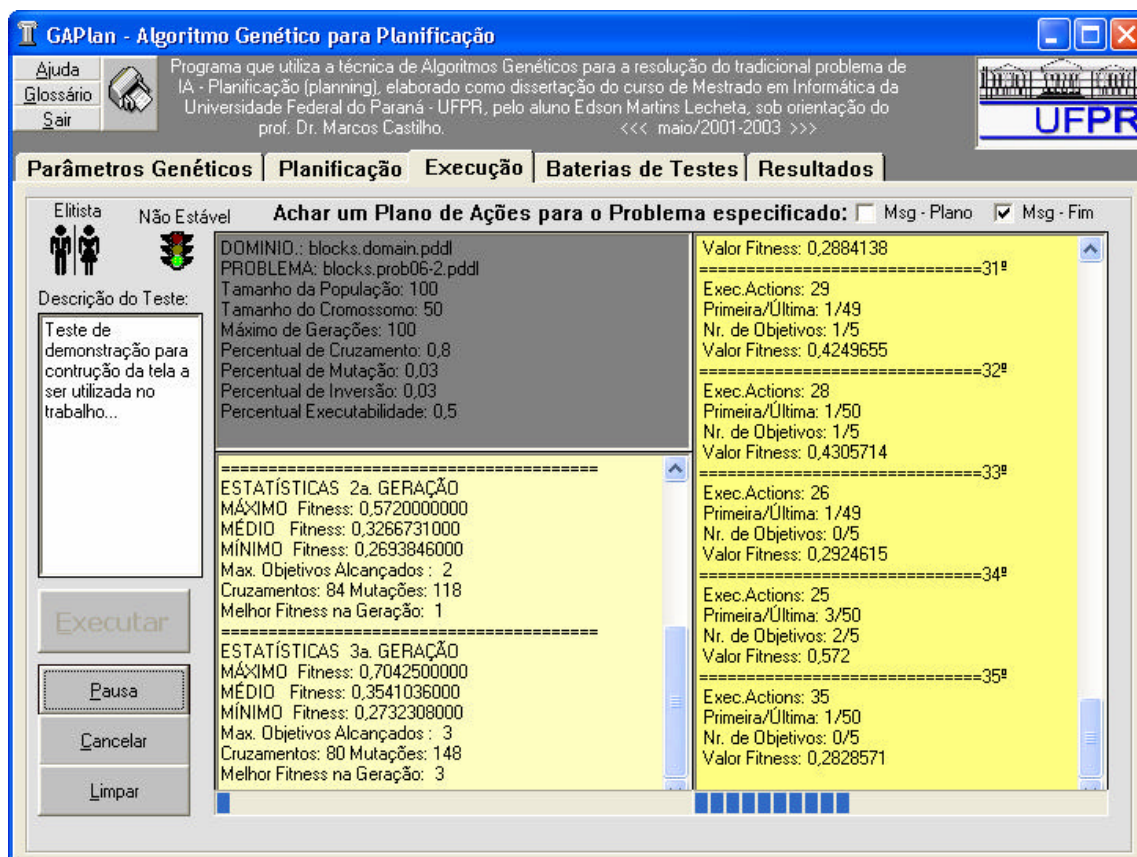


Figura 35. Tela de Acompanhamento da Execução do GAPLAN.

Existe um campo para a digitação das observações finais sobre o teste, que podem, mais tarde, identificar seu objetivo. O botão Executar dá início ao procedimento genético descrito na próxima seção deste capítulo, mas que pode ser observado pelas

informações sobre cada indivíduo, mostradas no painel à direita da tela, resumo das informações de cada geração, no painel à esquerda na parte de baixo e configurações gerais no painel logo acima deste. Alguns controles em botões que podem ser acionados durante a execução do AG são os botões de Pausa que interrompe imediatamente o programa e mostra uma mensagem na tela que permanecerá até que o usuário pressione o botão Ok desta caixa de mensagem, retornando do ponto onde parou para dar continuidade ao procedimento completo do AG. O Botão Cancelar solicita confirmação do usuário podendo haver apenas a pausa, no caso de não cancelar, ou o efetivo cancelamento do processo ao final da geração que estiver em curso. O botão Limpar serve para reiniciar alguns controles internos e também alguns controles de tela para, provavelmente, uma nova execução. Nesta tela existem ainda os indicadores gráficos que são atualizados durante a execução e servem para identificar a situação de estabilidade do AG e se está executando processos elitistas.

Pode-se optar pela execução de uma bateria de testes, configurados conforme a tela mostrada na Figura 36, a seguir.

GAPlan - Algoritmo Genético para Planificação

Ajuda Glossário Sair

Programa que utiliza a técnica de Algoritmos Genéticos para a resolução do tradicional problema de IA - Planificação (planning), elaborado como dissertação do curso de Mestrado em Informática da Universidade Federal do Paraná - UFPR, pelo aluno Edson Martins Lecheta, sob orientação do prof. Dr. Marcos Castilho. <<< maio/2001-2003 >>>

Parâmetros Genéticos | Planificação | Execução | Baterias de Testes | Resultados

Domínio de Planificação: gripper.domain.pddl
 Problema a ser tratado: gripper.problem02.pddl

Descrição do Teste: Bateria de Testes:
 Problema: gripper.problem02.pddl
 Nr. Testes: 20
 A partir de: 365

Gerar Dados Executar Testes Parar Testes

Nr.	Ger	Pop	Cron	Cros	Mut	Inv	Exec	P1F1	P2F2	P1F2	P2F2	Elit	p1Fx	p1I	p1F	p2Fx	p2I	p2F	p3Fx	p3I	p3F
366	100	100	20	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
367	100	100	21	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
368	100	100	22	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
369	100	100	23	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
370	100	100	24	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
371	100	100	25	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
372	100	100	26	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
373	100	100	27	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
374	100	100	28	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
375	100	100	29	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
376	100	100	30	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
377	100	100	31	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
378	100	100	32	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
379	100	100	33	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
380	100	100	34	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
381	100	100	35	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
382	100	100	36	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
383	100	100	37	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
384	100	100	38	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70
385	100	100	39	0,8	0,03	0,03	0,5	True	False	False	False	True	True	10	10	True	20	20	True	70	70

Figura 36. Tela para Configuração de Bateria de Testes.

Nesta tela repete-se a informação sobre o Domínio e o Problema de Planejamento previamente escolhidos e podemos solicitar ao programa, através do botão 'Gerar Dados', para que sejam gerados os dados automaticamente para o número de testes informados em 'Nr. de Testes a Gerar', com numeração iniciada no primeiro número a partir do informado no campo 'A partir de:'.

Com a mesma configuração em todos os testes gerados, o usuário pode selecionar o campo desejado na grade e começar a personalizar a bateria, alterando os dados que melhor lhe convier, sendo também permitido completar a descrição automática da bateria de teste, que se repetirá na caixa de descrição da tela de acompanhamento de execução, para onde, aliás, o programa altera a exibição com o objetivo de mostrar ao usuário a execução da bateria, depois de pressionado o botão 'Executar Testes'. Também é possível encerrar uma bateria em execução, através do pressionamento do botão 'Parar Testes'.

Como resultado de uma execução do AG podemos obter um entre dois resultados possíveis: encontrar o plano ou não. Quando um plano resolve o problema, seus dados são armazenados para conferência posterior e uma música de fundo alerta ao usuário que naquele instante foi encontrada uma solução.

Para o caso de não encontrar um plano que resolva o problema especificado, o AG segue até a última geração configurada e, sem alarde, aguarda a mudança da guia para a apresentação dos resultados, ou reconfiguração de novo teste ou bateria.

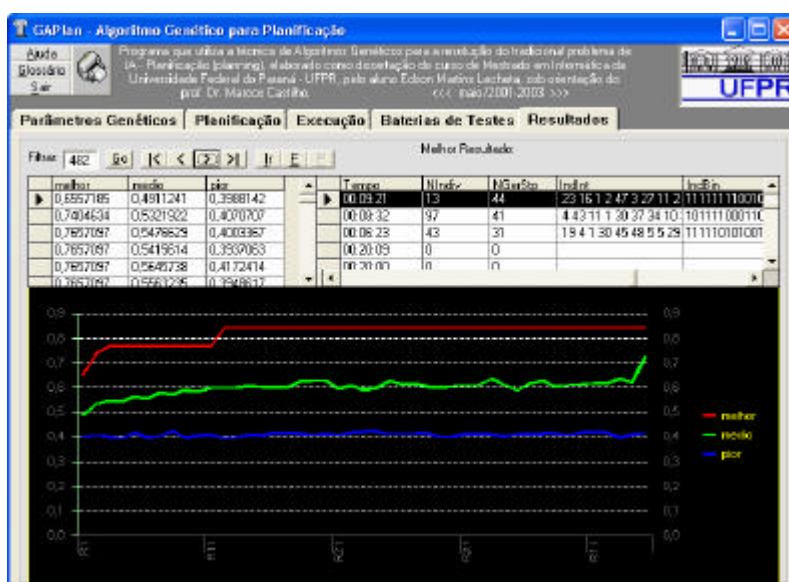


Figura 37. Tela para visualização e Análise dos Resultados.

Sendo uma execução simples pode-se, igualmente, seguir para a próxima guia dos resultados, mas em se tratando de uma bateria de testes, o próximo teste configurado já inicia automaticamente, sem parar, mesmo que tenha obtido sucesso no anterior, enquanto ainda toca a música de identificação de sucesso, já prossegue com o próximo teste. Ao final da bateria, segue-se para a guia dos resultados que, conforme pode ser visto na Figura 37, facilita a análise dos resultados pela apresentação dos dados, de maneira tabular e, de um gráfico que demonstra a curva do desenvolvimento do teste.

Os dados mostrados na tabela à esquerda representam as informações de cada geração de um determinado teste, que é reconhecido por seu número e, que compõem efetivamente o gráfico. À direita está a tabela com os dados gerais armazenados sobre as configurações de cada teste. Basta um duplo clique sobre o número do teste para que ele se torne o gráfico corrente. Esta operação também é facilitada pela opção de filtro pelo número do teste desejado ou com os botões de navegação sobre o banco de dados, permitindo uma fácil e rápida localização dos dados.

Todos os dados de todos os testes podem ser visualizados e, quando necessário, impressos, através da opção do botão de relatório, situado no canto superior esquerdo da tela, com a figura de uma impressora. O relatório é mostrado em ordem decrescente de número de testes para que os últimos sejam apresentados por primeiro, mas com controles que permitem a navegação por todas as páginas do relatório e a possibilidade de impressão de apenas algum intervalo de páginas. O relatório mostra, além dos dados tabulares, igualmente um gráfico correspondente aos testes, e ainda adiciona uma nova linha neste gráfico, que representa uma linha de análise de tendência polinomial dos valores médios, visto que esta informação pode ser interpretada como a existência ou não de evolução durante o processo do AG sobre aquelas gerações.

A ênfase no relatório está sobre a Situação Final do Mundo e o Plano Gerado, sendo apresentado na forma do cromossomo de inteiros, seu equivalente paralelo binário e, suas ações executáveis traduzidas em ações descritivas que permitem a análise imediata da validade e qualidade do plano gerado, conforme mostra a Figura 38, a seguir.