An Agent Program Capable of Applying Local Search Strategies in the State Space of Well Defined Problems

Thayanne França[1], Raimundo J. C. F. Junior[1], Jherson H. A. Pereira[3], Francisca R. de V. Silveira[2], Lidio M. L. de Campos[3], Thelmo P. Araújo[1], Gustavo A. L. de Campos[1]

[1]Ciência da Computação – Universidade Estadual do Ceará (UECE)
Av. Dr. Silas Munguba, 1700 – Itaperi – 60.714-903 – Fortaleza – CE

[2] Ciência da Computação – Instituto Federal de Educação –
Ciência e Tecnologia doCeará (IFCE) – Campus Aracati
CE 040 km 137,1 – Bairro Aeroporto – Aracati – CE

[3]Faculdade de Computação – Instituto de Ciências exatas e Naturais –
Universidade Federal do Pará (UFPA)
Rua Augusto Corrêa,01 - Guamá – 660751-110 – Belém – PA

jherson_k-f@hotmail.com,{raiferrojunior, thayanne.f.silva}@gmail.com,
raquel.silveira@hotmail.com, lidio@ufpa.br, {gustavo,
thelmo}@larces.uece.br

Abstract. *Classical models of agents for solving well-defined problems are widely used in the literature but are limited to systematic search strategies in order to find the solutions. However, these strategies are not suited for all types of application. This work presents an adaption of classical models of agents for local search strategies. One agent system for neural network automatic design is used to show the feasibility of the proposal. The results are promising, since the model found satisfactory solutions for the proposed problems.*

Resumo. *Os modelos clássicos de agentes de resolução de problemas bem definidos, apesar de serem largamente usados, se limitam ao uso de estratégias de buscas sistemáticas para encontrar soluções. Entretanto, essas estratégias não são adequadas para todos os tipos de aplicação. Este trabalho apresenta uma adaptação dos modelos clássicos de agente para o uso de estratégias de busca local. Um sistema de agentes para o projeto automático de redes neurais foi utilizado para demonstrar a aplicabilidade da extensão proposta. O modelo atingiu as expectativas, encontrando soluções satisfatórias para os problemas propostos.*

1. Introdução

Na maioria das discussões teóricas, a resolução de problemas em computador é caracterizada como um processo de busca através de uma árvore, mais precisamente um grafo direcionado, cujos 'nós' contém informações sobre estados, ou situações, e cujos 'ramos' contém informações sobre operações, ou ações, que transformam as situações

umas nas outras. Em geral, estes grafos contêm um nó de partida e um ou mais nós do tipo meta. Neste cenário, um agente de resolução de problemas deve ser capaz de encontrar uma sequência de ações que transformem uma situação inicial em uma situação desejada, ou seja, um caminho do nó de partida para algum nó meta [Simon 1977].

Grande parte das estratégias de busca sistemática, inclusive aquelas que possuem um poder heurístico razoável, implementa um ciclo de três ações principais, repetido até uma solução ser encontrada ou até um número predeterminado de repetições serem realizadas: (1) selecionar e remover o primeiro nó em uma lista de tentativas; (2) selecionar e aplicar *n* ações possíveis no estado descrito no nó selecionado; (3) avaliar os *n* estados gerados e armazenar os nós correspondentes em ordem de valores na lista de tentativas. Por exemplo, os programas de agentes artificiais de resolução de problemas bem definidos, isto é, que ocorrem em ambientes de tarefas observáveis e determinísticos, encapsula exatamente esta ideia [Russell e Norvig 2010].

O programa do agente de resolução de problemas desenvolvido e, atualmente, propagado nas salas de aula de, pelo menos, 750 universidades em 85 países, foi concebido para encontrar soluções que são sequências de ações, geradas a partir de um processo de busca sistemática no espaço de estados do problema. Dois tipos de buscas podem ser definidos para o agente realizar esse processo: as buscas sem informação, que vasculham exaustivamente o espaço de estados em comprimento ou largura, e as buscas com informação, que também vasculham exaustivamente o espaço de estados, mas com o auxílio de uma heurística que visa selecionar porções do espaço que estejam na direção dos estados desejados.

A maneira como foi descrito o programa do agente de resolução de problemas permite que o mesmo seja adaptado a executar um processo de busca com ou sem informação, dependendo da maneira como o projetista conceber a estrutura de dados que implementa a lista de tentativas, utilizada no ciclo de ações nas estratégias de busca sistemática, ou seja, como uma pilha, uma fila ou como uma fila de prioridades. Essa capacidade de adaptação do programa à implementação de diversas estratégias é interessante por dar flexibilidade ao projeto do agente sem a preocupação inicial com a escolha da estratégia, o que poderá ser feito a partir de experimentos com as diversas formas de implementar lista de tentativas.

Diferentemente das estratégias sistemáticas informadas, que exploram sistematicamente o espaço de estados do problema, armazenando um ou mais caminhos na memória e registrando em cada ponto ao longo de um caminho os nós que foram explorados e os que não foram, o agente de resolução de problemas também poderia empregar alguma estratégia de busca local [Michiels 2007]. Neste caso, o agente não necessitará armazenar caminhos na memória durante o processo de solução. O agente necessitará de pouquíssima memória e, frequentemente, conseguirá encontrar soluções satisfatórias em grandes espaços de estados ou em espaços de estados contínuos (infinitos), para os quais as estratégias de busca sistemáticas são inadequadas.

Ocorre que, a versão atual do programa de agentes artificiais de resolução de problemas não foi concebida para a execução de estratégias de busca local. Ainda cabem adaptações no ciclo de ações e nas estruturas de dados implementadas no programa, visando incorporar os princípios subjacentes na maioria destas estratégias.

Este trabalho consiste em uma contribuição que visa preencher esta lacuna, ou seja, nos referenciais teóricos sobre sistemas de resolução de problemas. Mais especificamente, o trabalho apresenta uma nova descrição que pode ser adequada à concepção de programas de agentes artificiais capazes de realizar busca local nos espaços de estados de problemas bem definidos.

Assim como na versão desenvolvida para busca sistemática, a ideia é que esta nova versão também possa ser adaptada a implementar as operações de avaliação, seleção e modificação de estados na maioria das estratégias de busca local, inclusive aquelas que são baseadas em populações, como é o caso de diversas meta-heurísticas bastante utilizadas na resolução de problemas. A descrição desta nova versão é realizada em mais quatro seções. A Seção 2 apresenta o programa concebido para busca sistemática. A Seção 3 apresenta a versão do programa concebido para busca local. A Seção 4 apresenta duas aplicações da nova versão do programa resolvendo problemas de otimização. A Seção 5 apresenta algumas considerações finais.

2. Agente de Resolução de Problemas com Busca Sistemática

A maioria dos procedimentos de resolução de problemas emprega alguma variação do ciclo de ações descrito na primeira seção, visando realizar buscas sistemáticas no espaço de estados do problema. Em relação à arquitetura do agente de resolução de problemas discutido neste trabalho, o formalismo para descrever o processo de tomada de decisão de programas de agentes artificiais para resolver problemas foi sintetizado a partir dos trabalhos sobre agentes inteligentes em [Russell e Norvig 2010] e em [Wooldridge 2002]. A Figura 1 ilustra o esqueleto para a concepção destes programas. Esta seção descreve como este esqueleto pode ser adaptado para programar agentes artificiais capazes de resolver problemas empregando estratégias de busca sistemática.

A Figura 1(a) apresenta o diagrama esquemático e o esqueleto do programa, descrito como um sistema de processamento de informação decomposto em três subsistemas, isto é, representado por três funções: *ver*, *próximo* e *ação*. A função *ver* representa o subsistema de percepção do agente, mapeia uma percepção do ambiente em uma descrição de estado correspondente, *ver:* $P \rightarrow S$. A função '*próximo*' representa o subsistema de atualização de estado interno do agente, mapeia a descrição do estado produzido pela função *ver*, e uma descrição de estado mais completa, mantida internamente pelo agente, em uma nova descrição de estado, *próximo: Estado* $\times S \rightarrow$ *Estado*. A função '*ação*' representa o subsistema de tomada de decisão do agente, mapeia uma descrição de estado interno em uma ação, *ação: Estado* $\rightarrow A$.

O subsistema de atualização de estado interno e o subsistema de tomada de decisão consideram informações sobre o efeito das ações do agente nos estados do ambiente, ou seja, descrito em termos de duas funções: a função *Ações: Estado* $\rightarrow 2^A$, mapeia um estado interno em um conjunto de ações possíveis de serem executadas no estado; e a função *RESULTADO: Estado* $\times A \rightarrow$ *Estado*, mapeia um estado interno e uma ação em um novo estado interno. O subsistema de tomada de decisão considera informações sobre o objetivo do agente (estados desejados), descrito em termos de uma função *Teste_Objetivo: Estado* $\rightarrow$ *{V, F}*, que mapeia um estado interno em um valor

verdade. Quando verdadeiro, o agente sabe que pode encerrar o processo de busca, pois gerou um estado desejado.

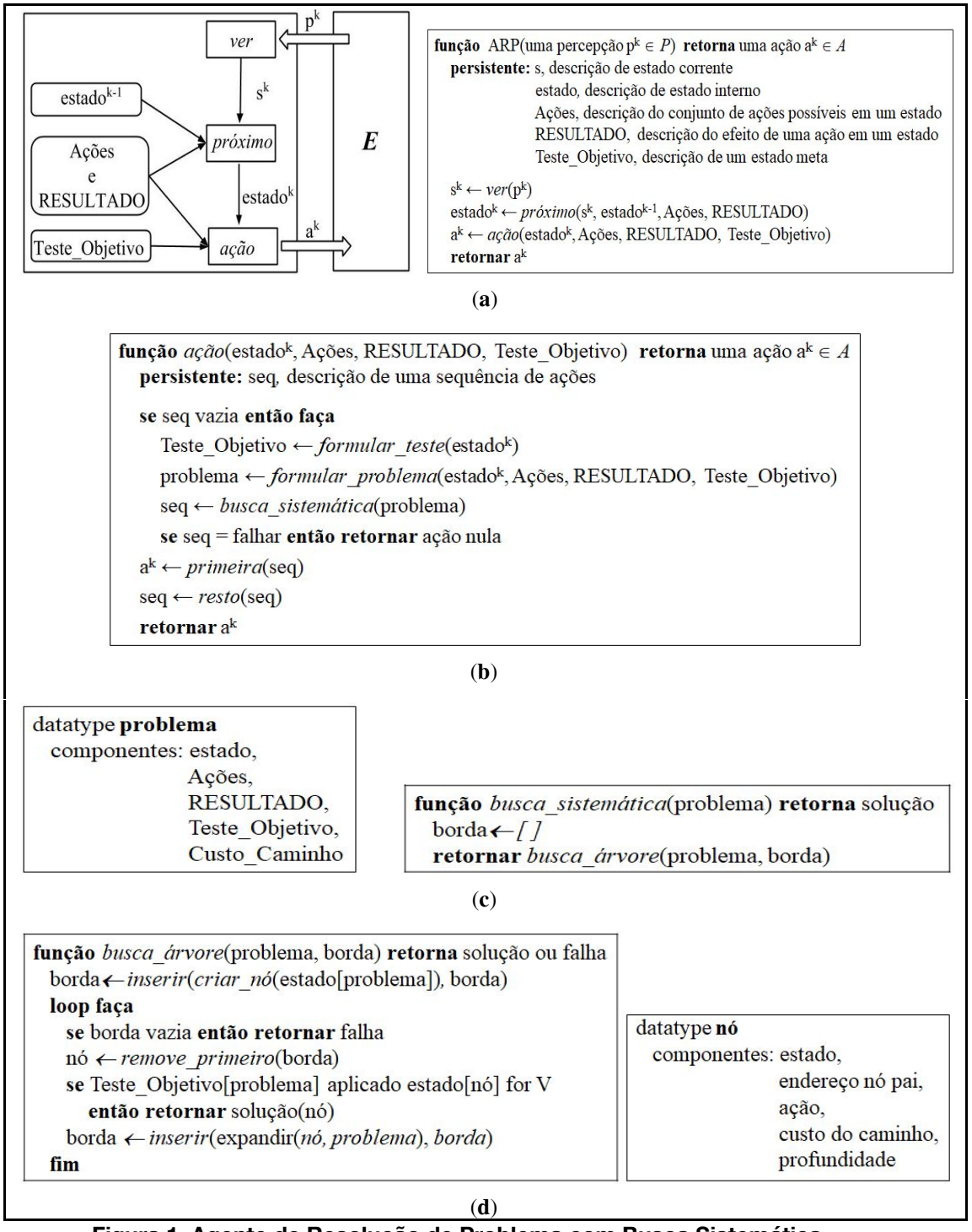


Figura 1. Agente de Resolução de Problema com Busca Sistemática

A Figura 1(b) apresenta a especificação do subsistema de tomada de decisão para o caso em que o agente emprega alguma estratégia de busca sistemática. Inicialmente, a variável 'seq', que representa uma sequência de ações é vazia. A estrutura de dados 'problema' contém todos os componentes que definem o problema do agente, ela pode ser representada por uma estrutura de dados da forma descrita na

Figura 1(c). A função *Custo_Caminho: Estado* × *A* × *Estado* → R^+ atribuí um custo numérico a cada caminho. Em problemas bem definidos, assume-se que o custo de caminho pode ser descrito como a soma dos custos das ações individuais ao longo do caminho.

O objeto 'nó' pode ser implementado por meio de uma estrutura de dados da forma apresentada na Figura 1(d), em que a variável 'estado' consiste em uma descrição do estado corrente no nó, 'nó pai' identifica o endereço do nó pai na árvore de busca, 'ação' identifica a ação que deu origem ao estado descrito no nó, 'profundidade' é a profundidade do nó na árvore, e 'custo_caminho' é o custo do caminho que liga o nó raiz ao nó sobre consideração. Considerando os esquemas de representação das informações disponíveis na definição de um problema, o esquema de busca em árvore do agente pode ser programado tomando como base as funções: *busca_sistemática* e *busca_árvore* (ou em grafo), descritas na Figura 1(c) e 1(d).

A variável 'borda' nestas funções representa uma lista que serve para armazenar os nós que ainda não foram expandidos. Esta lista é inicializada com o nó que contém a descrição do estado inicial do problema. O programa realiza repetidamente um ciclo de três ações principais: (1) remove o primeiro nó da borda, (2) testa para ver se o estado descrito no nó removido é um estado meta e, caso não seja, (3) ramifica este nó, aplicando todas as ações possíveis no estado representado no nó, gera novos nós e os inseri na lista 'borda' de acordo com alguma ordem, no início ou no final. Este ciclo ações continua até que o nó removido contenha um estado desejado ou que todo o espaço de estados tenha sido percorrido.

A ordem e a maneira como os nós gerados pela função '*busca_árvore*' são inseridos em 'borda' é determinada pela estratégia de busca sobre consideração. Existem duas grandes classes de estratégias de busca sistemática, ou seja: as Desinformadas ou Cegas, e as Informadas ou Heurísticas. As estratégias Desinformadas, para decidir qual será o próximo nó a ser ramificado na árvore, só fazem uso das informações disponíveis na definição do problema. Elas partem do nó que representa o estado inicial do problema e ramificam sistematicamente a árvore de busca até uma solução ou uma condição de parada ser encontrada. As estratégias de busca Informadas fazem uso de informações extras como, por exemplo, de funções para avalição do estado em um nó para selecionar o próximo nó. Estas funções adicionam conhecimento a respeito dos problemas nas estratégias de busca Desinformadas.

Por exemplo, as estratégias de busca cega Largura e Profundidade encontram soluções de maneira diferentes, que são também diferentes de Custo Uniforme e das estratégias de busca Heurística. A estratégia de busca em Largura simplesmente insere os nós que foram ramificados no final de 'borda', ou seja, a lista funciona como uma fila. A estratégia Profundidade insere os novos nós no início da borda, ou seja, a lista funciona como uma pilha. A estratégia de busca Custo Uniforme considera o custo associado ao nó como sendo o valor do custo do caminho do nó raiz até o nó sobre consideração, e ordena todos os nós previamente inseridos em 'borda'. As estratégias Heurísticas como, por exemplo, Gulosa pela Melhor Escolha e a busca A*, consideram o valor de função avaliação heurística associado ao nó. Em ambos os casos a lista 'borda' funciona como uma fila de prioridades.

Entre os algoritmos que encontram soluções ótimas a busca em grafo *A** é considerada otimamente eficiente, ou seja, nenhum outro algoritmo ótimo tem a garantia de expandir um número de nós menor que *A**. Mesmo assim, *A** não é a resposta para todos os problemas, pois armazena muitos nós, podendo esgotar a memória bem antes de esgotar o tempo. Em todo caso, o uso de uma boa função heurística ainda proporciona enorme economia em comparação com a busca sem informação. Alguns algoritmos desenvolvidos recentemente superam o problema de espaço, sem sacrificar o caráter ótimo ou a completeza, a um custo pequeno no tempo de execução.

3. Agente de Resolução de Problemas com Busca Local

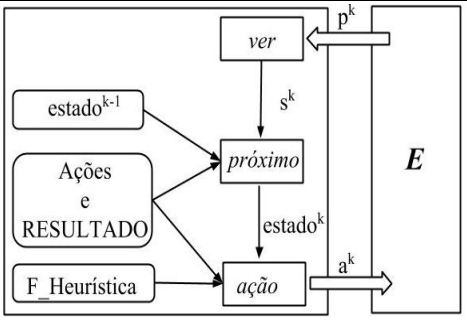
O agente de resolução de problemas com busca local não explora sistematicamente o espaço de estados do problema, nem armazena um ou mais caminhos na memória, nem registra em cada ponto ao longo de um caminho os nós que foram explorados, e os que não foram. Diferentemente da busca sistemática, em que o agente representa o problema como um problema de busca em uma árvore ou em um grafo direcionado, o problema do agente consiste em encontrar o melhor estado de acordo com a função avaliação heurística. Diz-se que a busca é local pois ela escolhe o próximo estado sem decidir com antecedência para onde irá em seguida [Michiels 2007]. Por exemplo, vale considerar algumas destas estratégias locais bastante utilizadas em diversos sistemas de resolução de problemas: Subida de Encosta, Feixe Local, Têmpera Simulada e Algoritmo Genético [Russell e Norvig 2010].

A Figura 2 ilustra o esqueleto adaptado do agente de resolução de problemas. Esta adaptação visa inserir no programa descrito na Figura 1 os princípios presentes na maioria das estratégias de busca local. A Figura 2(a) apresenta o diagrama esquemático e o esqueleto do programa adaptado. A única alteração com relação à Figura 1(a) diz respeito à inserção da informação sobre a função avaliação 'F_Heurística', no lugar da informação sobre o objetivo 'Teste_Objetivo' no subsistema de tomada de decisão do agente, especificado na Figura 2(b). A função avaliação, *Função_Heurística: Estado* $\rightarrow R^+$, atribuí um valor de custo estimado a um estado. Inicialmente, a variável 'solução' para o problema é vazia. Além da informação sobre um estado inicial e o modelo de transição de estados, a estrutura de dados 'problema' contém a informação sobre a função avaliação.

A Figura 2(c) descreve a função '*busca_local*'. Esta função deve ser adaptada para funcionar de acordo alguma implementação de busca local específica, conforme o projetista do agente desejar, tomando como base aquelas estratégias que estão disponíveis na literatura sobre o assunto. A informação na estrutura de dados 'nó corrente' agrega somente a descrição do estado no nó e o valor da função heurística aplicada ao estado. Esta variável é inicializada com a informação que descreve o estado inicial corrente do problema e seu valor de avaliação, obtidos respectivamente por meio das funções '*gerar_solução*' e '*avaliar*'.

A função '*busca_local*' opera usando somente o estado corrente e, em geral, produz movimentos para vizinhos deste estado, utilizando as informações sobre o estado corrente e sobre o efeito das ações neste estado. A função considera a existência de um contador de iterações 't' e de um teste, *teste*, que serve para verificar a satisfação da

condição de parada do processo de busca. Esta condição é descrita por meio de uma proposição envolvendo informações sobre seus parâmetros como, por exemplo: o número máximo de iterações permitido para a estratégia e um valor de avaliação satisfatória para uma solução.



função ARP(uma percepção $p^k \in P$) **retorna** uma ação $a^k \in A$
persistente: s, descrição de estado corrente
estado, descrição de estado interno
Ações, descrição do conjunto de ações possíveis em um estado
RESULTADO, descrição do efeito de uma ação em um estado
F_Heurística, descrição de uma heurística para avaliar estados
$s^k \leftarrow$ *ver*(p^k)
$estado^k \leftarrow$ *próximo*(s^k, $estado^{k-1}$, Ações, RESULTADO)
$a^k \leftarrow$ *ação*($estado^k$, Ações, RESULTADO, F_Heurística)
retornar a^k

(**a**)

função *ação*($estado^k$, Ações, RESULTADO, F_Heurística) **retorna** uma solução
persistente: solução, descrição de uma solução, estado ou sequência de ações
se solução vazia **então faça**
Função_Heurística ← *formular_heurística*($estado^k$)
problema ← *formular_problema*($estado^k$, Ações, RESULTADO, Função_Heurística)
solução ← *busca_local*(problema, parâmetros_estratégia)
se solução = falhar **então retornar** solução nula
retornar solução

(**b**)

função *busca_local*(problema, parâmetros) **retorna** uma solução
t ← 1
$solução^t \leftarrow$ *gerar_solução*(estado[problema], parâmetros)
avaliação ← *avaliar*($solução^t$, F_Heurística[problema])
nó ← *criar_nó*($solução^t$, avaliação),
loop faça
se nó vazio **então retornar** falha
se *teste*(t, nó[avaliação])
então retornar solução(atual)
$solução^{t+1} \leftarrow$ *modificar*(solução[nó], avaliação[nó], problema, parâmetros)
avaliação ← *avaliar*($solução^{t+1}$, F_Heurística[problema])
nó ← *atualizar*($solução^{t+1}$, avaliação),
t ← t+1
end

(**c**)

função *subida_de_encosta*(problema) **retorna** um estado que é um máximo local
...
nó_corrente ← *criar_nó*(estado[problema]);
repita
nó_vizinho ← *sucessor de corrente com valor mais alto*
se *valor[nó_vizinho]* ≤ *valor[nó_corrente]* **então retorne** estado[nó_corrente]
nó_corrente ← nó_vizinho

gerar_solução
avaliar e *modificar*
teste
atualizar

(**d**)

Figura 2. Agente de Resolução de Problema com Busca Local

A função '*modificar*' é invocada repetidamente para transformar uma solução corrente candidata em uma nova solução candidata, conforme os valores de avaliação e o modelo de transição de estados indicarem. O processo é encerrado quando uma solução candidata for ótima e/ou a condição de parada em '*teste*' for satisfeita. A função '*avaliar*' considera uma solução candidata corrente e atribuí um valor de avaliação de custo/lucro estimado da solução. Esta função decide se aceita a nova solução candidata como solução corrente e, caso positivo, armazena suas informações na estrutura de dados 'nó corrente'. Ao final do processo, se não retornou falha antes, a função '*busca_local*' retorna uma solução satisfatória para o problema.

Vale ressaltar, a estratégia em '*busca_local*' encapsula, na função tomada de decisão do agente, a ideia de uma busca local orientada por valor de função avaliação heurística, ou seja, uma adaptação da técnica de programação gerar-e-testar com o objetivo de encontrar uma solução satisfatória [Simon 1977]. Mais especificamente, em problemas em que o objetivo do agente consiste em encontrar a melhor solução candidata de acordo com uma função avaliação, e que a solução desejada pode ser obtida a partir da transformação de uma solução candidata inicial dada, pode-se dizer que a estratégia em '*busca local*' na figura consiste em uma adaptação da técnica modificar-testar.

A Figura 2(d) apresenta o pseudocódigo da estratégia de busca local Subida de Encosta. As partes do código que foram grifadas identificam as quatro principais ações presentes na maioria dos algoritmos de busca local, ou seja, no caso do programa proposto a repetição de um ciclo de quatro ações: (1) gerar solução inicial candidata ou modificar solução candidata corrente; (2) avaliar solução candidata; (3) testar e ver se a candidata é uma solução em uma iteração corrente; (4) atualizar solução candidata corrente. Assim, da mesma forma que Subida de Encosta, várias outras estratégias de busca local podem ser utilizadas como referencial para concretizar o esqueleto na Figura 2. A seção a seguir apresenta um exemplo considerando os algoritmos genéticos.

4. Aplicação Exemplo

O agente de resolução de problemas pode empregar as estratégias locais em diversas situações. Por exemplo, para resolver problemas em que os caminhos até os estados desejados são irrelevantes, ou seja, o que é importante na solução é a "configuração" final e não a ordem em que os objetos são acrescentados. Pode, também, empregar estas estratégias em situações em que os estados desejados não foram definidos, como é o caso dos problemas de otimização, em que se busca encontrar o melhor estado de acordo com uma função objetivo. O agente pode não ter conhecimento a respeito de um teste objetivo e de uma função custo do caminho até um nó, que estão disponíveis na definição dos problemas para os quais as estratégias sistemáticas são normalmente empregadas.

Esta subseção apresenta um exemplo de aplicação do esqueleto do agente que realiza busca local para resolver um problema de otimização, ou seja, uma aplicação para resolver o problema de projeto automático de redes neurais artificiais [De Campos 2016]. A adaptação do programa de agente de resolução de problemas considerou os princípios empregados nos algoritmos genéticos descrito na Figura 3. Conforme pode

ser percebido, as operações no esqueleto (*gerar*, *avaliar*, *modificar*, *testar*, *atualizar*) foram implementadas considerando como fundamento os operadores genéticos '*seleção de pares*', '*cruzamento*' e '*mutação*'.

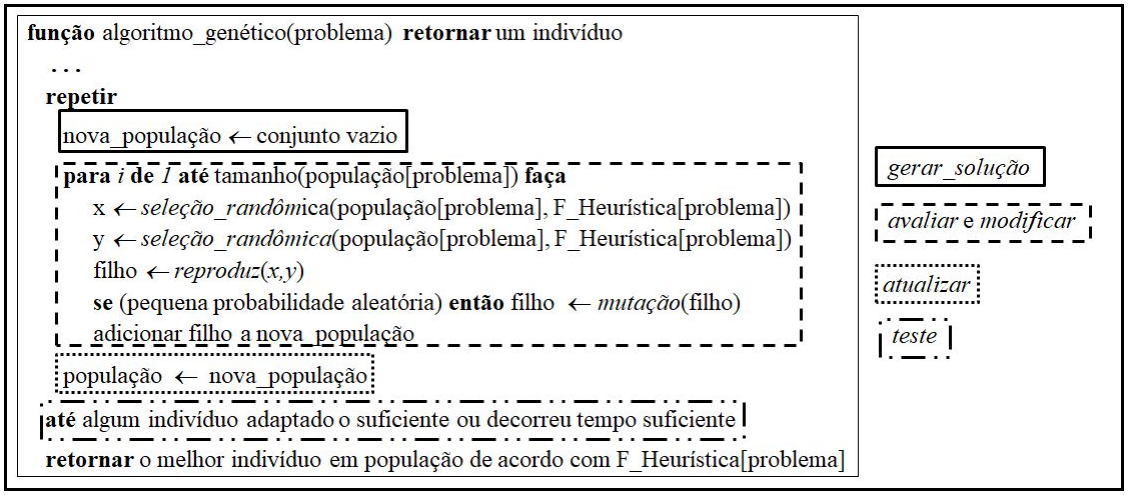


Figura 3. Um Algoritmo Genético Popular

O problema de projeto automático de rede neural artificial (RNA) pode ser formulado como um problema de otimização, em que se deseja encontrar uma topologia de RNA que seja capaz de minimizar uma função avaliação que, a princípio, consiste no erro quadrático médio entre valores desejados nas saídas dos neurônios da RNA, em um conjunto de observações, e valores calculados nestas saídas, quando são apresentados valores correspondentes aos desejados nas entradas dos neurônios da RNA [Yao 1987]. A topologia de uma RNA pode ser descrita como um conjunto finito de neurônios, representados por 'nós' em um grafo, e um conjunto finito de conexões entre neurônios, ou seja, 'arcos' direcionados no grafo.

Por exemplo uma RNA *feed-forward* pode ser representada por um grafo acíclico, enquanto que uma RNA recorrente por um grafo cíclico. Uma camada de entrada é formada por um conjunto de unidades de entrada, ou seja, um subconjunto de 'nós' do grafo. A camada de saída é formada por um conjunto de unidades de saída, ou seja, um subconjunto de 'nós' do grafo. Em uma rede *feed-forward*, uma camada intermediária 'k' (k> 1) é formada por um conjunto de nós, em que para cada 'nó' no conjunto existe um caminho de comprimento 'k-1' entre alguma unidade de entrada e o 'nó'. No caso das redes totalmente conectadas, todas as unidades possuem conexões para todas as unidades que não são unidades de entrada.

A abordagem relatada nesta seção consiste em uma adaptação do esqueleto descrito na Figura 2. O agente resultante foi denominado ADEANN (Artificial Development and Evolution of Artificial Neural Networks) [De Campos 2016]. Este agente encapsula um esquema especial de representação de soluções, e sua estratégia de busca local é baseada em algoritmos genéticos (GA) para resolver o problema de projeto automático. Em relação ao esquema de representação, a abordagem adota uma representação generativa, ou seja, em vez de uma topologia de RNA codificada, cada cromossomo armazena um conjunto de regras de produção de um sistema de Lindenmayer [Rozenberg and Salomaa 1980], que, por sua vez, gera topologias de RNA.

No início do processo de busca de uma solução para o problema, a função '*gerar_solução*' do agente ADEANN gera uma população inicial de RNA, onde cada rede é representada por um conjunto de regras de produção de um sistema Lindenmayer, codificado em cada cromossomo da população. Esta função considera a informação sobre parâmetros do GA como, por exemplo, o número desejado de RNA na população inicial e sobre o comprimento desejado para os cromossomos na população.

O Sistema_L utilizado pode ser descrito como uma gramática G = {□, □, à}, onde o alfabeto consiste nos elementos do conjunto □**= {. , f, F, n, [,], *, B}** e as regras de produção □ são descritas na Tabela 1. O axioma **à = .** é o ponto de partida do processo de desenvolvimento, onde **f** denota um neurônio e **F** é uma conexão entre neurônios, os símbolos '**[**' e '**]**' indicam armazenamento e recuperação, respectivamente, do estado atual do desenvolvimento ***** indica que anteriormente armazenada é recuperada, **B** representa a conexão de um neurônio com um bloco de neurônios. A segunda regra. **.→ (f ... f) n**, significa substituir o ponto de partida do desenvolvimento pelos neurônios da camada de entrada.

Identificador	Regra
1,2	S→. (axiom) (2) .→(f...f)n
3	(3.1) f→[f (3.2) f→fFf (3.3) f→fF (3.4) f→n
3,4,5	(3.5) f→f (3.6) f→fB (4) [→[Ff] (5) f→f*

Tabela 1 – Regras de Produção do Sistema-L utilizado

A Figura 4 apresenta um exemplo de aplicação das regras, partindo-se do axioma S=., até a obtenção do fenótipo **||FnFnB]FnFnB]n||**, o qual representa uma rede neural com três camadas, ou seja, um neurônio na entrada, dois na intermediária e dois na saída.

```
||  S  ||
||  .  ||
||  f  ||
||  [ f  ||
||  [ F f ] F f ] f  ||
||  [ F f F f ] F f F f ] f  ||
||  [ F f F f B ] F f F f B ] f  ||
||  [ F n F n B ] F n F n B ] n  ||


Numero de Neuronios Camada Intermediaria -> 2
Numero de Neuronios por Ramificacao -> 2
NENT -> 1
NSAI -> 2
```

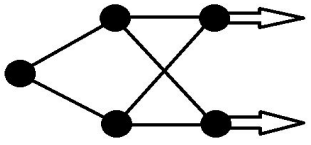


Figura 4. Derivação de uma RNA aplicando as regras de produção do Sistema-L

A função '*avaliar*' determina o valor de desempenho calculado de cada topologia de RNA em uma população corrente, com base no erro quadrático médio (EQM) calculado na camada de saída de cada RNA na população. O sistema ajusta os pesos das redes neurais candidatas à solução para o problema visando minimizar o EQM nos conjuntos de treinamento e teste. A abordagem busca minimizar o EMQ de

cada RNA na fase de treinamento, para que a melhor rede selecionada, possa também generalizar e predizer corretamente o conjunto de testes. Propôs-se uma abordagem de penalidade que impede o algoritmo de produzir desnecessariamente RNAs com muitos neurônios ocultos. Assim, a avaliação de soluções foi implementada considerando-se a seguinte função:

Função_Heurística = exp(-**EMQ**) x (exp(**NNCO**) + 1/(**EMQ x NNCO**).

A condição de parada na função '*teste*' é descrita por meio de uma proposição que relaciona as informações sobre o número de máximo de gerações a ser realizada pela estratégia de busca local (máximo de loops no esquema de repetição), e sobre um valor de *Função_Heurística* aceitável (valor de avaliação) para uma solução satisfatória para o problema de projeto automático de RNA. A função '*modificar*' é executada repetidamente visando transformar uma população corrente de RNAs, candidatas à solução para o problema, em uma nova população de RNAs. Na abordagem proposta, esta função encapsula os operadores de seleção de pares de soluções candidatas, de cruzamento em pares e de mutação individual, empregados nos algoritmos genéticos.

A seleção de pares foi concebida empregando-se: "torneio", "classificação" ou "ranking" e "truncamento". Ao recombinar os códigos genéticos de dois ancestrais, o cruzamento produz duas soluções em uma região ainda não visitada do espaço de busca. Para cada par de ancestrais, o cruzamento é aplicado com uma certa probabilidade de cruzamento Pc. A abordagem utiliza vários pontos cruzamento, que são selecionados aleatoriamente. O número de pontos de cruzamento também é proporcional ao comprimento da solução, escolhidos aleatoriamente com probabilidade Pm.

Os experimentos foram realizados utilizando-se uma população de RNAs treinada com um algoritmo de aprendizado hebbiano, significando que a abordagem seleciona, portanto, as melhores arquiteturas de RNAs, que simula a contento dois problemas: aprendizagem de comportamentos estereotipados tais como taxias e reflexo, e aprendizagem de comportamentos reativos e instintivos. Foram utilizados três "datasets" para os testes da abordagem, sendo que para cada um foi necessário ajustar e configurar o algoritmo com diversos valores para taxas de aprendizagem e taxas de esquecimento, para que o algoritmo conseguir uma maior precisão.

5. Considerações Finais

O desenvolvimento de novos modelos para representação e implementação de agentes inteligentes podem contribuir, de um ponto de vista educacional, para um melhor entendimento e ensino dessa tecnologia assim como, de um ponto de vista tecnológico, por serem facilmente adaptadas, facilitam durante o processo de concepção de agentes para resolver problemas em diversos ambientes de tarefas. Apesar de grande utilidade, o referencial teórico sobre agentes de resolução de problemas ainda está em desenvolvimento, faltando preencher algumas lacunas, o que os impede de ser usados em todas as situações possíveis nestes ambientes. A maioria dos trabalhos considera que os problemas ocorrem em ambientes observáveis e determinísticos. O referencial desenvolvido para ambientes difíceis como, por exemplo, em que o agente está diante de problemas de múltiplos estados, de contingências e de exploração, está sendo consolidado à medida que novas aplicações da ideia vão sendo implementadas.

A contribuição apresentada neste artigo consiste em uma proposta de extensão do referencial atual, ou seja, visando preencher uma outra lacuna, que é uma integração fluida do programa de agente de resolução de problemas, inicialmente desenvolvido para estratégias de busca sistemática, para o caso das estratégias de busca local. Apesar da extensão proposta ser adaptável a partir da maioria das estratégias de busca, as aplicações da ideia, relatadas, foram adaptadas a partir de uma única estratégia, ou seja, o algoritmo genético, que é baseado em uma população de soluções correntes. Independentemente de ter sido testado com outra estratégia, o programa proposto facilitou o projeto e implementação dos agentes mencionados. Todas as estruturas e funções foram devidamente implementadas de acordo com o domínio específico das aplicações. Os objetivos da pesquisa a seguir estão relacionados com a aplicação da ideia considerando outras estratégias e com as adaptações necessárias para o caso de problemas mais difíceis que os bem definidos.

References

Campos, G. A. L. e Vinhas, R. R. (2000) "Busca Heurística – Classificação e Um Quadro Formal", Anais do XIII Congresso Brasileiro de Automática, Florianópolis-BR, p. 1568-1573.

De Campos, L. M. L. (2016) "Uma Metodologia Biologicamente Inspirada para Projeto Automático de Redes Neurais Artificiais usando Sistemas-L Paramétricos com Memória", Tese de Doutorado PPGEE/UFPA.

Michiels, W., Aarts, E., Korst, J. (2007) "Theoretical Aspects of Local Search", Springer-Verlag Berlin Heidelberg, Vol III.

Nguyen, C. D., Miles, S., Perini, A., Tonella, P. Harman, A. Luck, M. (2012) "Evolutionary testing of autonomous software agents", Autonomous Agents and Multi-Agent Systems 25 (2), 260-283.

Rozenberg, G. and Salomaa, A. (1980) "The Mathematical Theory of L Systems, Volume 90", Academic Press, 1th edition.

Russell, S. J. and Norvig, P. (2010) "Artificial Intelligence A Modern Approach", Prentice Hall, 3th edition.

Silveira, S. R. V. (2013) "Uma Abordagem Fundamentada em Agentes Racionais para o Teste de Agentes Racionais", Dissertação de Mestrado MACC/UECE.

Simon, H. A. (1977) "Models of Discovery", D. Reidel Publishing Company, Netherlands.

Wooldridge, M. (2002) "Introduction to MultiAgent Systems", Wiley, 2th edition.

Yao, X. (1987) "Evolving artificial neural networks". Proc. IEEE 87, 1423-1447.