Aplicação de Redes Neurais Artificiais em Problemas de Satisfação de Restrições

Application of Artificial Neural Networks to Constraints Satisfaction Problems

Jesmmer da S. Alves
IF Goiano Campus Morrinhos
Núcleo de Computação
Morrinhos, Goiás, Brasil
jesmmer.alves@ifgoiano.edu.br

Cedric L. Carvalho
Univ. Federal de Goiás
Instituto de Informática
Goiânia, Goiás, Brasil
cedric@inf.ufg.br

Gabrielly de A. Sabino
IF Goiano Campus Morrinhos
Núcleo de Computação
Morrinhos, Goiás, Brasil
gabrielly.sabino2000@gmail.com

RESUMO

A aplicação de redes neurais a problemas de otimização, devido à dificuldade deste tipo de problema, tem se tornado uma opção na busca por soluções em tempos aceitáveis. Este artigo apresenta um modelo de rede neural aplicado a Problemas de Satisfação de Restrições. O modelo utiliza redes neurais específicas para identificar soluções candidatas que violam restrições ao problema. Uma implementação para o problema de Distribuição de Tarefas foi feita para analisar a aplicabilidade do modelo a este tipo de problema. Nesta implementação, foram desenvolvidas estratégias que permitissem treinamento mais rápido das redes e melhor identificação de características específicas nas soluções candidatas.

Palavras chave

redes neurais, aplicações em IA, problemas de satisfação de restrições, problemas de distribuição de tarefas

ABSTRACT

Application of neural networks to optimization problems, due to the complexity of such problem, has become an option in the search for getting solutions in acceptable times. This paper presents a neural network model applied to Constraints Satisfaction Problems. The model aims to utilize specific networks to identify candidate solutions that violate constraints to the problem. An implementation to the Timetabling Problem was made in order to analyze the applicability of the model to this type of problem. In this implementation, strategies were developed to allow faster network training and better identification of specific characteristics in the candidate solutions.

Keywords

neural networks; IA applications; constraints satisfaction problems; timetabling problem

1. INTRODUÇÃO

Uma Rede Neural Artificial assemelha-se à rede neural do cérebro humano e dos animais, principalmente por sua capacidade de aprendizagem, pela utilização de forças/pesos para ressaltar a importância de determinadas entradas e pela capacidade de modificar sua estrutura de acordo com os conhecimentos adquiridos. Desta forma, tais redes podem utilizar acontecimentos obtidos no passado para tomar decisões sobre acontecimentos no presente ou futuro.

As redes neurais utilizam métodos heurísticos para tomar decisões a partir de conhecimentos adquiridos. Este comportamento "decisório" pode se tornar uma boa opção na busca por soluções adequadas para problemas em que a utilização de métodos determinísticos não produziu bons resultados. Sobretudo, diferentes estratégias podem ser usadas como forma de evidenciar características específicas em soluções e, com isto, permitir o treinamento e aplicação de redes neurais a problemas de otimização, por exemplo, na solução do problema de distribuição de tarefas (*Timetabling Problem* [4]).

Este trabalho tem como objetivos apresentar conceitos importantes da implementação de redes neurais artificiais, descrever um modelo para aplicação de redes neurais a problemas de satisfação de restrições (Constraint Satisfaction Problem - CSP [11]) e descrever detalhes de uma aplicação destas redes ao problema de distribuição de tarefas em uma universidade.

Com esse propósito, a Seção 2 apresenta a definição do problema de satisfação de restrições. A Seção 3 descreve conceitos importantes sobre a implementação de redes neurais utilizando o *software* Matlab, principais arquiteturas e processo de aprendizagem. A Seção 4 descreve características relevantes com relação à modelagem e utilização destas redes e a Seção 5 descreve trabalhos relacionados. Um modelo para aplicação de redes neurais a problemas de satisfação de restrições é descrito na Seção 6 e uma aplicação deste modelo ao problema de distribuição de tarefas é apresentado na Seção 7. Finalmente, na Seção 8, são apresentadas as considerações finais dos autores.

2. SATISFAÇÃO DE RESTRIÇÕES

A satisfação de restrições é uma técnica para modelagem e solução de problemas computacionais. A satisfação das restrições depende de uma descrição declarativa do problema que consiste em um conjunto de variáveis e seus respectivos domínios. Cada domínio é composto por um conjunto de

valores possíveis e as restrições restringem o conjunto de combinações possíveis das variáveis.

Um problema de Satisfação de Restrições, do inglês *Constraint Satisfaction Problem - CSP*, é um par (S, ϕ), onde S é um espaço de busca livre e ϕ é uma fórmula (função booleana sobre S) [11]. A solução de um problema de satisfação de restrições é um $s \in S$ com $\phi(s) = true$. Usualmente, um *CSP* é iniciado como um problema de encontrar uma instanciação para as variáveis $v_1, \ldots, v_n$ com domínios finitos $D_1, \ldots, D_n$ tais que as restrições $c_1, \ldots, c_n$, definidas para as variáveis (ou algumas delas), sejam satisfeitas. O leitor pode encontrar mais informações sobre o *CSP* em [8, 14].

3. REDES NEURAIS

Uma Rede Neural [15] consiste de um conjunto de unidades de processamento básico (neurônios) que são organizados em camadas e se comunicam enviando sinais de uma para outra, com cada conexão associada a um peso. Um Neurônio (veja Figura 1) é a representação de uma unidade básica de processamento. Este recebe sinais de entrada, por meio da camada de entrada ou de outros neurônios, que são multiplicados pelos pesos das respectivas conexões e, juntamente com a *bias*, são somados por meio de uma função somatório. A *bias*[1] é usada como uma forma de ajustar todas as entradas. Uma função de ativação determina exatamente como deve ser a saída do neurônio.

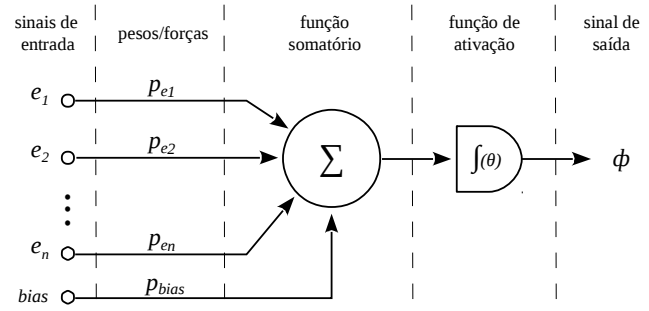

Figura 1: Unidade básica de processamento - Neurônio [15].

A arquitetura de redes neurais é baseada na construção de blocos ou camadas, com um ou mais neurônios, que executam o processamento. As três principais configurações para uma rede neural são: *feed forward* com uma camada; *feed forward* com múltiplas camadas; e redes recorrentes. O termo *feed forward* é usado para representar redes neurais em que os sinais são transportados em uma única direção, ou seja, sem recorrência (*feedback*).

A arquitetura mais simples de uma rede neural é definida com uma única camada e um ou mais neurônios representando a(s) saída(s) da rede (veja Figura 2). Neste caso, a camada contendo os dados de entrada não é contada porque nenhum tipo de processamento é efetuado nesta camada.

Múltiplas camadas intermediárias podem ser utilizadas para dar maior flexibilidade à rede na adequação da estrutura a problemas que exigem maior processamento de dados. É importante ressaltar que a rede neural é composta de unidades de processamento básico, ou seja, cada neurônio recebe um conjunto de sinais, faz um somatório e exibe uma saída. Desta forma, para a grande maioria dos problemas, uma única camada de saída com alguns neurônios não conseguirá bons resultados. A Figura 3 apresenta uma arquitetura em duas camadas, com quatro neurônios na camada intermediária e dois neurônios na camada de saída.

O projetista da rede neural pode fazer uso de recorrência na arquitetura da rede. Este tipo de projeto de rede se caracteriza por permitir que as saídas dos neurônios em uma camada sejam entradas para neurônios em camadas precedentes (veja Figura 4). Em alguns casos, o projetista deve fazer uso de unidades de atraso (*delay*) para evitar que um neurônio processe uma informação que ainda está dependente do processamento de outros neurônios.

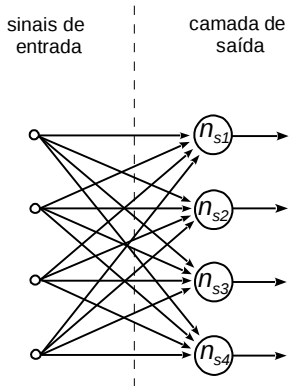

Figura 2: Uma camada (saída).

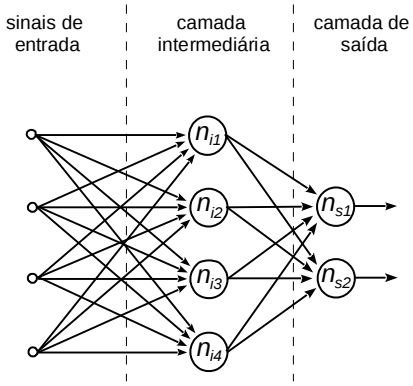

Figura 3: Múltiplas camadas.

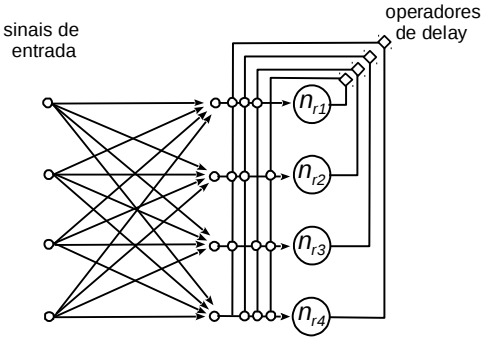

Figura 4: Redes Recorrentes.

[1]A *bias* pode ser usada como um conceito estatístico para avaliar modelos antes do treinamento ou como um tipo de neurônio específico, chamado de neurônio *bias*.

Dois aspectos importantes devem ser considerados no projeto de uma rede neural, o Conhecimento e a Aprendizagem. O conhecimento se refere às informações armazenadas ou modelos usados por uma pessoa ou máquina para interpretar, prever e apropriadamente responder ao mundo exterior [15]. Em redes neurais, o conhecimento fica armazenado na própria estrutura da rede, com a utilização de pesos sinápticos e *bias*, e representa modelos caracterizados por mapeamentos de entrada e saída de informações.

Aprendizagem é um processo em que os parâmetros livres de uma rede neural são adaptados por meio de um processo de estimulação pelo ambiente da rede [15]. Desta forma, no processo de aprendizagem, a rede é estimulada pelo seu ambiente (com informações passadas pela camada de entrada), recebe mudanças em seus parâmetros (ajustes para adequar a saída da rede) e pode responder de maneira diferente.

4. MODELAGEM DE REDES NEURAIS

Para definir a arquitetura de uma rede neural, em geral, o projetista começa definindo uma rede de tamanho mínimo e, caso não consiga treiná-la, inicia um processo que envolve aumentar sucessivamente o número de camadas intermediárias e neurônios nas camadas até conseguir uma taxa de erro aceitável. Este processo também pode ser visto como um problema de otimização em que o objetivo é encontrar uma configuração de rede com taxa de erro mínimo. Neste caso, primeiro diferentes arquiteturas de rede (redes candidatas) são treinadas, depois são estimados os erros de generalização para cada uma e, por fim, a rede com erro de generalização mínimo é selecionada.

A relação entre os erros que acontecem no treinamento e na generalização com o tamanho da rede neural (em camadas ou neurônios) revela um tamanho ideal/ótimo da rede neural [5]. Isto implica que o aumento do tamanho da rede na fase de treinamento diminui a quantidade de erros nesta fase. No entanto, a partir de certo ponto, começa a influenciar de forma negativa a quantidade de erros que acontecem na fase de generalização. Outros aspectos podem ser importantes na modelagem de redes neurais, tais como:

- Dividir e validar dados. Técnicas de divisão dos dados estão relacionadas com as formas de dividir o conjunto de dados disponíveis em dados de treinamento e dados de teste (generalização). A validação têm como objetivo fornecer parâmetros que possam ser úteis no processo de seleção de uma rede ótima.

- Utilizar informações a *priori*. Utilização de conhecimentos prévios têm como objetivo diminuir as possibilidades de ajustes de parâmetros livres na correção de erros em redes *backpropagation* [15], geralmente fazendo com que a saída de um neurônio em uma camada não seja entrada para todos os neurônios nas camadas seguintes.

- Gerar automaticamente a estrutura da rede. O *Cascade Correlation* [7] é um algoritmo de aprendizagem supervisionado que define uma arquitetura "ótima" para a rede neural. Ao invés de ajustar os pesos em uma rede com uma topologia fixa, o algoritmo começa com uma rede mínima, então automaticamente treina e adiciona novas unidades de camadas intermediárias, uma a uma, criando uma estrutura multicamadas.

5. TRABALHOS RELACIONADOS

Diferentes estratégias já foram utilizadas para resolver problemas de satisfação de restrições. A dificuldade de desenvolvimento de algoritmos determinísticos para o *CSP* serviu de motivação para Eiben e Ruttkay [6] utilizarem algoritmos genéticos na solução deste tipo de problema. Os autores fizeram transformações no problema original e aplicaram algoritmos evolucionários a estas transformações. Uma característica importante desta forma de solução do *CSP* é que as transformações podem acarretar em situações ainda mais difíceis de serem resolvidas que o próprio problema original.

A combinação de algoritmos (algoritmos híbridos) pode ser uma alternativa interessante na solução do *CSP*. Prosser, P. [12] utilizou a combinação de algoritmos clássicos para formar quatro algoritmos híbridos (*backmarking* com *backjumping*, *backmarking* com *conflict-directed backjumping*, *forward checking* com *backjumping* e *forward checking* com *conflict-directed backjumping*). Em uma abordagem similar, algoritmos híbridos foram utilizados para solucionar o *CSP* com base em tarefas de configuração de sistemas [17].

Redes neurais também têm sido uma boa opção na solução de *CSPs*. Adorf *et al.* [1], desenvolveram um algoritmo baseado em redes *Hopfield* para resolver vários tipos de *CSPs*, dentre eles o problema das N-Rainhas (*N-Queen Problem*). Outro exemplo da aplicação de redes neurais na solução do *CSP* foi apresentada por Wang e Tsang [16]. Os autores associaram as variáveis do problema à ativação de neurônios para fazer com que as restrições representem as conexões da rede, possivelmente com pesos diferentes. Assim, quando a rede converge, o conjunto de neurônios representando um conjunto de associações formam uma solução. Um dos problemas da utilização deste tipo de técnica é que a rede pode convergir para um local mínimo.

Mais recentemente, outros trabalhos têm abordado a utilização de redes neurais para resolver o problema de satisfação de restrições. Dentre as estratégias mais satisfatórias está a utilização das Redes Neurais de *Hopfield*. Este tipo de rede neural é inspirada em conceitos das áreas de física, dinâmica não-linear e tem como principais características: unidades computacionais não-lineares; simetria nas conexões sinápticas; e são totalmente realimentadas (exceto auto-realimentação). O leitor pode encontrar mais informações sobre a utilização das Redes Neurais de *Hopfield* em [2, 3, 9, 10].

6. MODELO APLICADO AO CSP

O modelo aqui proposto tem como objetivo definir e treinar redes neurais para identificar soluções (geradas aleatoriamente) que violem as restrições para o problema de satisfação de restrições (veja na Figura 5). Ou seja, deve ser criada uma rede neural para identificar soluções que violem cada restrição do problema. As soluções são passadas individualmente a cada rede e separadas de acordo com as restrições que foram violadas. Por fim, o modelo disponibiliza bancos de dados contendo soluções que não violem nenhuma restrição e soluções que violem *1, 2, ... , n* restrições (que podem ser consideradas relaxações para o problema).

Na Figura 5, as redes neurais (R_1, R_2, ..., R_n) são representadas por nuvens e os cilindros são bancos de dados contendo inicialmente o conjunto total de soluções candidatas. Após a execução de cada rede, são criados novos bancos de dados contendo o conjunto de soluções que violam alguma

restrição ("soluções violam Rx"), e o conjunto de soluções que não violam nenhuma restrição para problema ("soluções após Rx").

Na implementação do modelo, o sistema deve funcionar como um "testador de soluções", ou seja, as redes neurais devem ser criadas a partir das restrições para um problema e aplicadas individualmente às soluções candidatas (geradas aleatoriamente) em busca de soluções que não violem nenhuma restrição ou, soluções que violem um número/conjunto permitido de restrições (relaxações ao problema). A seguir, são descritos os passos necessários na implementação do modelo:

1. Gerar as soluções candidatas (criar o banco de dados inicial "soluções");
2. Gerar soluções de treinamento para cada uma das redes;
3. Definir e treinar as redes neurais;
4. Aplicar a primeira rede neural (R_1) ao conjunto de soluções candidatas (soluções);
5. Aplicar a segunda rede neural (R_2) aos resultados obtidos com a rede neural R_1 ("soluções após R_1");
6. Aplicar a rede neural R_n aos resultados obtidos com a rede neural R_{n-1} ("soluções após R_{n-1}").

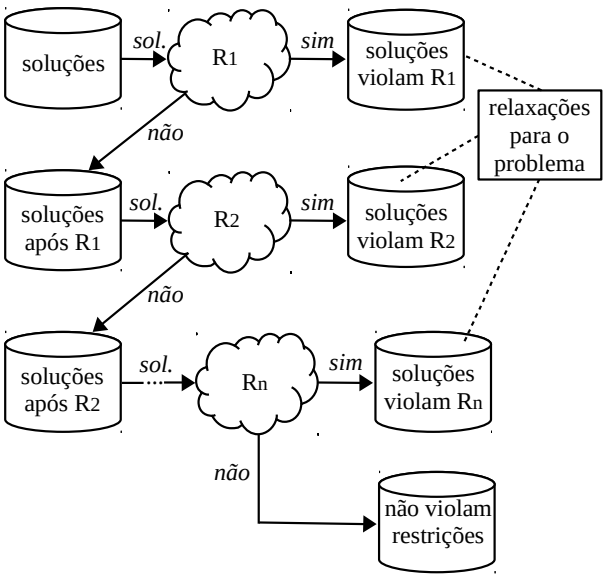

Figura 5: Modelo Aplicado ao Problema de Satisfação de Restrições.

O modelo proposto, além de permitir a aplicação de redes neurais a problemas de satisfação de restrições, também permite: o agrupamento de soluções candidatas de acordo com características específicas; a identificação de relaxações ao problema (soluções candidatas que violam alguma restrição mas ainda podem ser utilizadas como soluções parciais para o problema); e também pode disponibilizar diferentes soluções que não violam nenhuma restrição (desde que o sistema pode continuar em execução, mesmo após a identificação de alguma solução que não viola nenhuma restrição).

7. IMPLEMENTAÇÃO PARA O STP

O modelo proposto neste artigo foi testado em uma versão do Problema de Distribuição de Tarefas (*School Timetabling Problem - STP*) para fazer a distribuição de tarefas a professores do Instituto de Informática da Universidade Federal de Goiás. A implementação foi feita com a utilização do *software* Matlab [13] e todos os experimentos foram feitos em um *desktop* com processador *Intel Core 2 Duo 2.20GHz* e *2.00GB* de RAM. As Seções seguintes descrevem o problema, modelagem e implementação, estratégias utilizadas e resultados obtidos.

7.1 O Problema

O problema foi dividido em duas etapas: alocação de professores para ministrar disciplinas do curso; e alocação de outras atividades para os professores. Na primeira etapa, foram consideradas a distribuição de horários a *priori* (horário de aulas pré-definido) e a existência de um número mínimo de professores para ministrar todas as disciplinas. Além disso, a alocação de disciplinas deve respeitar as seguintes restrições:

- Um professor não pode ser associado a duas disciplinas que tenham aulas no mesmo horário;
- A distribuição de disciplinas deve ser feita tentando satisfazer as preferências dos professores (cada professor indica três disciplinas com prioridade alta, média e baixa);
- Um professor deve ficar, necessariamente, com uma a três disciplinas por semestre;
- Cada professor deve ser associado a pelo menos uma disciplina de prioridade 1 ou 2, segundo suas preferências.

Na segunda etapa, outras atividades (orientação de alunos, participação em comissões, etc.) devem ser alocadas para os professores. Neste caso, deve ser considerada a disponibilidade de horas do professor e não os horários. Esta distribuição de atividades deve ser gerada de forma dinâmica, ou seja, quando uma nova demanda surgir, deverá ser indicado um professor para atendê-la.

7.2 Modelagem e Implementação

As redes foram definidas por meio de experimentos na busca por estruturas que retornassem melhores resultados. A Figura 6 mostra como foram definidas as arquiteturas para as redes utilizadas na primeira etapa (*R1*, para identificar soluções candidatas com professores sem disciplinas associadas, e *R2*, para identificar soluções candidatas com professores com mais de 3 disciplinas associadas). A Figura 7 mostra a estrutura da rede *R3*, definida na segunda etapa para fazer a distribuição dinâmica das tarefas.

As redes *R1* e *R2* têm como entrada um vetor de 35 posições (solução candidata), três camadas intermediárias (a primeira com 40 e as outras duas com 20 neurônios) e uma camada de saída (com um neurônio). O Algoritmo 1 descreve a definição e treinamento da rede *R1*. A rede *R2* foi definida e treinada de maneira similar à rede *R1*.

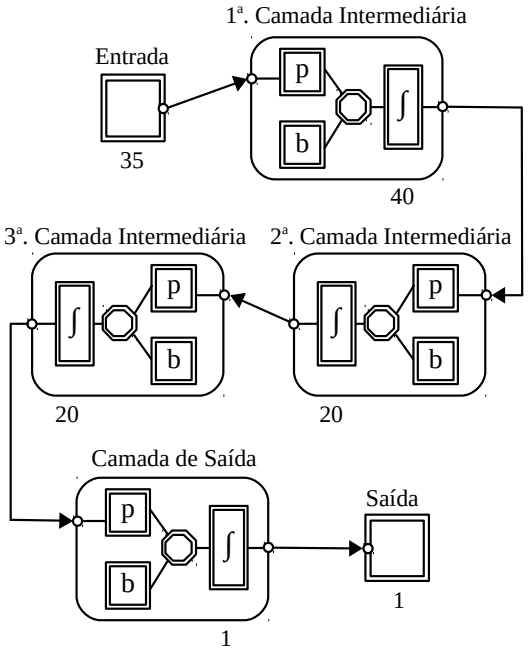

Figura 6: Rede R_1 - Identificar Soluções candidatas com professores sem disciplinas.

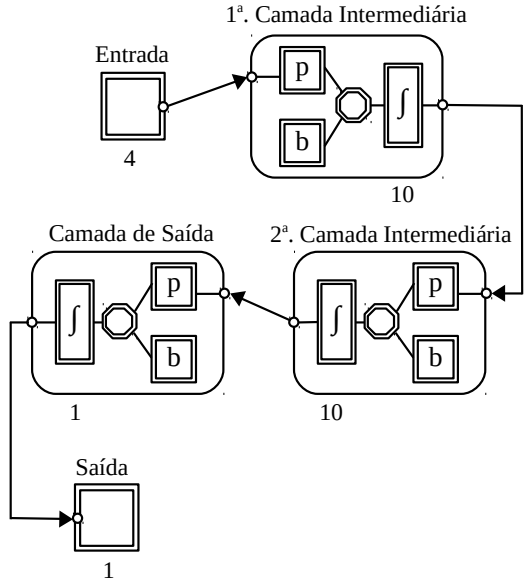

Figura 7: Rede R_3 - Distribuição dinâmica das tarefas.

Algoritmo 1: Arquitetura para a rede R1.

```
1  ...
2  redeR1=newff([1 21;1 21;1 21;1 21;1 21;
3  1 21;1 21;1 21;1 21;1 21;1 21;1 21;1 21;
4  1 21;1 21;1 21;1 21;1 21;1 21;1 21;1 21;
5  1 21;1 21;1 21;1 21;1 21;1 21;1 21;1 21;
6  1 21;1 21;1 21;1 21;1 21;],[0 1],
7  [40 20 20]);
8  rede_R1 = train(rede_R1, sort(trainR1(:,
9            horario(:,1))'),trainR1_res');
10 ...
```

A rede *R3* tem como entrada um vetor de quatro posições (quantidade de aulas de cada um dos professores e a tarefa a ser distribuída), duas camadas intermediárias (ambas com 10 neurônios) e uma camada de saída (com um neurônio). O Algoritmo 2 mostra como foi definida a rede *R3* no Matlab. O comando *newff* (linha 1) define os quatro parâmetros para os valores de entrada da rede (correspondente à faixa de valores para cada parâmetro), a faixa de valores correspondente aos códigos dos professores (de 1 a 21) e a quantidade de neurônios nas duas camadas. O comando *train* (linha 2) define qual é a rede a ser treinada (redeR3), os dados de entrada (colunas 1, 2, 3 e 4 da matriz trainR3) e os resultados desejados para cada uma das entradas (coluna 5 da matriz trainR3).

Algoritmo 2: Arquitetura para a rede R3.

```
1 redeR3 = newff([100 700; 1 10; 1 10; 1 10;],
2 [1 21],[10 10]);
3 redeR3 = train(redeR3,trainR3(:,[1,2,3,4])',
4 trainR3(:,[5])');
```

Na primeira etapa, uma solução candidata para o problema foi formada a partir do horário de aulas dos professores, ou seja, um vetor de 70 posições correspondente a 35 disciplinas (cada disciplina ocupando 2 posições no vetor) que devem ser distribuídas a 21 professores do curso de ciência da computação. Foi construído um banco de dados com 50.000 soluções candidatas (geradas a partir do horário de aulas definido a *priori*) para professores e disciplinas selecionados aleatoriamente.

As duas redes criadas para a primeira etapa (*R1* e *R2*), foram treinadas com 2.500 soluções "boas" (que não violam nenhuma restrição) e 2.500 soluções "ruins" (que violam alguma restrição), e trabalharam sobre as 50.000 soluções candidatas. A Figura 8 apresenta como foram definidas as soluções candidatas e as redes *R1* e *R2*.

Para segunda etapa, uma única rede neural (*R3*) foi treinada a partir da carga horária definida na distribuição de disciplinas da primeira etapa e a área da tarefa a ser distribuída (por exemplo: Álgebra, Algoritmos, Internet, etc.). Assim, uma nova tarefa a ser distribuída deve ser analisada, de acordo com a carga horária e o conjunto de disciplinas que cada professor está habilitado a ministrar.

O Algoritmo 3 descreve como foram geradas 500 soluções para treinamento da rede *R3*. Neste caso, a área é "area 100" (linha 2), cada professor nesta área recebe uma carga horária aleatória (linhas 3, 4 e 5) e a rede seleciona qual professor da "area 100" (linhas 6 a 12) que possui menor carga horária.

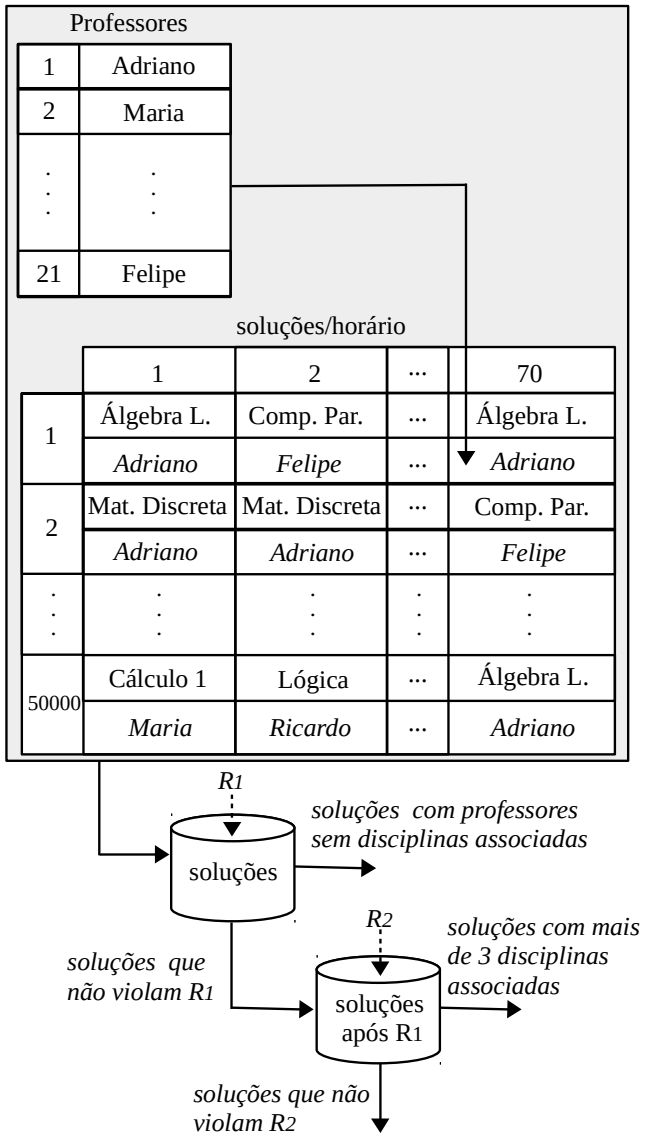

Figura 8: Soluções candidatas e configuração das redes *R1* e *R2*.

Algoritmo 3: Gerar 500 soluções de treinamento para a rede R3.

```
for l = 1 : 500
    trainR3(l,1) = 100;
        trainR3(l,2) =  randi(5);
        trainR3(l,3) =  randi(5);
        trainR3(l,4) =  randi(5);
        if (trainR3(l,2)<trainR3(l,3)) &
                (trainR3(l,2)<trainR3(l,4))
            trainR3(l,5) = 1;
        else if (trainR3(l,3)<train01(l,4))
                trainR3(l,5) = 2;
            else trainR3(l,5) = 3;
            end;
        end;
end;
```

7.3 Estratégias de Representação

Uma solução candidata, da maneira que foi definida inicialmente, corresponde a um vetor de 70 posições podendo ter em cada posição valores entre 1 e 21 (índice para identificar os professores). Tal definição expôs, durante os experimentos, uma dificuldade muito grande de treinamento das redes criadas. Além disto, possivelmente, foi responsável por um fraco desempenho da rede durante a fase de generalização. Isto se deve à grande quantidade de possibilidades que as redes precisavam lidar durante o treinamento.

Outro aspecto importante está na forma de trabalho das redes. O objetivo principal do modelo proposto é fazer com que a rede neural reconheça características particulares nas soluções e assim, identifique se as mesmas violam alguma restrição do problema. No entanto, redes neurais funcionam como uma "caixa preta" (a configuração da rede é feita pelo algoritmo de treinamento). Isto impede que o projetista da rede determine como a rede deve fazer os ajustes nos parâmetros livres para identificar características nas soluções.

Estes aspectos incentivaram o desenvolvimento de estratégias de representação (estratégias utilizadas para evidenciar características particulares nas soluções candidatas) que permitissem um treinamento mais rápido das redes e uma melhor identificação de características específicas nas soluções candidatas, são elas:

- **Reduzir o tamanho das soluções candidatas**. De acordo com o horário de aulas definido a *priori*, todas as disciplinas correspondem a duas posições neste horário (no mesmo dia ou não). Desta forma, foi criada uma matriz *horario* contendo as posições de cada disciplina e as soluções passaram a ser organizadas em um vetor de 35 posições (correspondente à primeira posição de cada disciplina na matriz *horario*). Para a segunda etapa, o vetor de 21 posições (quantidade de professores) foi reduzido para apresentar como entrada somente professores da área específica da tarefa a ser distribuída. Por exemplo, uma palestra sobre Segurança de Internet (tarefa) só pode ser atribuída a professores que estão na área "Sistemas para Internet".

- **Ordenar os valores nas soluções**. Esta estratégia foi motivada pelo fato que é mais fácil identificar elementos específicos em soluções ordenadas. Por exemplo, dados dois vetores: `A = {10 1 8 3 7 5 9 6 4 0}` e `B = {0 1 3 4 5 6 7 8 9 10}`, é mais natural ao ser humano identificar pelo vetor `B` ausência do número `2`.

- **Multiplicar os valores de cada posição em uma solução por um valor fixo**. Esta também foi uma forma de evidenciar elementos específicos nas soluções. Dado o vetor: `A = {10 1 8 3 7 5 9 6 4 0}`, a expressão: `if A[x] > 1, A[x] = A[x] * 500;` resulta em: `A = {5000 1 4000 1500 3500 2500 4500 3000 2000 0}`. Isto pode fazer com que a rede neural identifique de forma "mais fácil" as entradas 0 e 1.

- **Sair do mínimo gradiente.** O mínimo gradiente corresponde ao menor valor para a função de perda do algoritmo de treinamento. Em outras palavras, significa que o algoritmo de treinamento conseguiu ajustar da melhor maneira possível os parâmetros livres da rede de acordo com os dados de entrada. Em muitos casos, este ajuste pode ser bem demorado. No entanto,

o projetista da rede pode optar por treinamentos que se aproximam do mínimo gradiente e ainda oferecem bons resultados de generalização.

7.4 Resultados Obtidos

Devido à utilização de procedimentos heurísticos, os resultados (soluções e tempos estimados) tiveram variações em seus valores durante cada execução do sistema. Portanto, os resultados foram coletados e analisados a partir de 10 execuções em sequência. As redes *R1* e *R2* precisaram de 2.000 e 1.800 segundos respectivamente em seus treinamentos. A rede *R3* foi treinada com gradiente mínimo em aproximadamente 400 segundos.

A Tabela 1 apresenta de forma detalhada os dados coletados para a primeira etapa. A colunas *Qtde após R1* e *Qtde após R2* apresentam a quantidade de soluções que não violaram nenhuma restrição após a aplicação da rede *R1* e da rede *R2* respectivamente. Já as colunas *% Boas* e *% Ruins* indicam a percentagem de soluções que, após a execução das redes, não violam nenhuma restrição e que violam pelo menos uma restrição, respectivamente. A quantidade inicial em todas as execuções foi de 50.000 soluções.

	Qtde após R1	Qtde após R2	% Boas	% Ruins
1	120	8	80%	20%
2	80	12	50%	50%
3	87	6	73%	27%
4	70	15	60%	40%
5	55	13	70%	30%
6	63	5	60%	40%
7	45	10	46%	54%
8	90	8	72%	28%
9	49	12	41%	59%
10	72	16	85%	15%

Tabela 1: Resultados de 10 execuções da redes *R1* e *R2*.

Apesar de todas as redes terem sido treinadas até se conseguir o gradiente mínimo, ao final da execução do sistema, nem todas as soluções encontradas foram soluções "Boas". Isto é, apesar das redes não identificarem características que violassem alguma restrição, em uma análise pós-execução das redes, algumas soluções foram consideradas "Ruins" por ainda violarem restrições do problema. Isto aconteceu, possivelmente, por causa da utilização de funções para gerar de números aleatórios.

Para a segunda etapa, também foram realizadas 10 execuções, em todas elas a rede *R3* conseguiu responder de forma apropriada qual a melhor distribuição de tarefas de acordo com a quantidade de horas de cada professor e a área de aplicação da tarefa (ou área de atuação dos professores). Nesta etapa, a cada execução da rede, o banco de dados correspondente à carga horária dos professores era atualizado. Ou seja, se uma tarefa X com carga horária 8 for distribuída ao professor Y, a referida carga horária da tarefa é somada à carga horária atual do professor.

8. CONSIDERAÇÕES FINAIS

A opção por utilizar redes neurais para resolver problemas de satisfação de restrições surgiu devido à grande dificuldade de sistemas determinísticos em criar/encontrar soluções que não violassem nenhuma restrição ao problema. Esta dificuldade vem do fato de que, de forma geral, quando o sistema encontra uma característica que viole uma restrição do problema, todo ou quase todo o processo de busca/criação da solução deve ser reiniciado. Assim, uma rede neural que identifique características específicas em soluções sem a reinicialização de todo o processo apresentou-se como uma opção promissora.

Alguns aspectos chamaram mais a atenção durante o processo de modelagem e execução das redes. Por exemplo, desde que as soluções candidatas e as soluções de treinamento são geradas de forma aleatória, os resultados podem ser completamente diferentes em cada execução. Sobretudo, gerar soluções aleatoriamente pode exigir que o projetista da rede modifique a arquitetura da mesma (quantidade de camadas intermediárias e neurônios em cada camada) na busca por uma configuração que resulte em um gradiente mínimo. Além destes, outros aspectos importantes influenciaram muito nos resultados alcançados, são eles:

- Quantidade de camadas e neurônios. A quantidade de camadas intermediárias e neurônios foi influenciada pela quantidade de exemplos treinados. Ou seja, uma quantidade grande de exemplos de treinamento exigiu um número maior de camadas intermediárias e neurônios nestas camadas.
- Quantidade de soluções usadas para treinamento das redes. Tal quantidade de soluções pode melhorar a generalização. No entanto, também pode aumentar muito o tempo de treinamento das redes.
- Estratégias de representação. A utilização de estratégias que evidenciem características particulares nas soluções candidatas, tais como as apresentadas na Seção 7.3, são de real importância na redução do tempo de treinamento e em melhores resultados na fase de generalização.

Outro fator importante que deve ser ressaltado é o tempo de projeto e execução. O treinamento das redes, até que se consiga um mínimo gradiente, e a utilização de estratégias de representação pode aumentar muito o tempo de execução de todo o sistema. Esta situação pode ser amenizada com a aplicação de redes neurais por meio de sistemas paralelos/distribuídos. Isto porque a execução das redes neurais é assíncrona, ou seja, nenhuma execução depende de resultados de outras execuções. O tempo de execução também está relacionado com a modelagem e a capacidade do projetista de modificar a estrutura da rede a partir dos resultados alcançados.

Por fim, é importante destacar que o modelo apresentado pode ser utilizado com outras técnicas de Inteligência Artificial que tenham comportamento similar ao das redes neurais, por exemplo a utilização de árvores de decisão, algoritmos genéticos ou outras técnicas combinadas, ao invés de redes neurais. Sobretudo, a combinação das estratégias apresentadas neste artigo com utilização das Redes Neurais de *Hopfield* pode ser uma boa opção para trabalhos futuros.

Agradecimentos

Os autores agradecem à Universidade Federal de Goiás e ao Instituto Federal Goiano - Campus Morrinhos pelo apoio durante o desenvolvimento deste trabalho.

9. REFERÊNCIAS

[1] H.-M. Adorf and M. D. Johnston. A discrete stochastic neural network algorithm for constraint satisfaction problems. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 917–924. IEEE, 1990.

[2] A. Bouhouch, H. Bennis, L. Chakir, and A. El Qadi. Neural network and local search to solve binary csp. *Indonesian Journal of Electrical Engineering and Computer Science*, 10:1319–1330, 03 2018.

[3] A. Bouhouch, L. Chakir, and A. El Qadi. Scheduling meeting solved by neural network and min-conflict heuristic. In *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*, pages 773–778. IEEE, 2016.

[4] E. Burke, D. de Werra, et al. Applications to timetabling. In *Handbook of graph theory*, pages 530–562. Chapman and Hall/CRC, 2013.

[5] R. Caruana, S. Lawrence, and L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, pages 402–408, 2001.

[6] A. Eiben, P.-E. Raué, and Z. Ruttkay. Solving constraint satisfaction problems using genetic algorithms. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence.*, pages 542–547. IEEE, 1994.

[7] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

[8] B. Faltings. Distributed constraint programming. In *Foundations of Artificial Intelligence*, volume 2, pages 699–729. Elsevier, 2006.

[9] K. Haddouch, K. Elmoutaoukil, and M. Ettaouil. Solving the weighted constraint satisfaction problems via the neural network approach. *IJIMAI*, 4(1):56–60, 2016.

[10] K. Haddouch, M. Ettaouil, and L. Chakir. Continuous hopfield network and quadratic programming for solving the binary constraint satisfaction problems. *Journal of Theoretical and Applied Information Technology*, 56:362–372, 01 2013.

[11] A. K. Mackworth. Constraint satisfaction problems. *Encyclopedia of AI*, 285:293, 1992.

[12] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational intelligence*, 9(3):268–299, 1993.

[13] D. A. Rosenbaum. *MATLAB for behavioral scientists*. Psychology Press, 2012.

[14] T. Shultz. Constraint satisfaction models. 2001.

[15] H. Simon. *Neural networks: a Comprehensive Foundation*. Prentice Hall, 1999.

[16] C. Wang and E. Tsang. Solving constraint satisfaction problems using neural networks. In *Artificial Neural Networks, 1991., Second International Conference on*, pages 295–299. IET, 1991.

[17] L. Wang and W.-K. Ng. Hybrid solving algorithms for an extended dynamic constraint satisfaction problem based configuration system. *Concurrent Engineering*, 20(3):223–236, 2012.