Ajuste de Pesos de uma Rede Neural utilizando PSO para o Mario AI Competition

Fabio Yuki Kubagawa
Universidade Federal do Paraná
kyongxd@gmail.com

Gustavo Toshi Komura
Universidade Federal do Paraná
gustavo.toshi.k@gmail.com

11 de dezembro de 2013

Resumo

Este artigo apresenta uma forma de ajuste de pesos de uma rede neural utilizando o PSO para o Mario AI Competition. Neste novo algoritmo são utilizados dois tipos de topologia de vizinhança do PSO o totalmente conectado e o em anel, variando o tamanho da população entre 50, 100 ou 300, e também dois tamanhos de rede neural, a SmallMLP 3x3 e MediumMLP 5x5. Os resultados dos experimentos mostraram que uma rede neural com tamanho SmallMLP, população igual a 300 e com uma vizinhança totalmente conectada, possui a melhor fitness com relação às outras múltiplas configurações do novo algoritmo. No entanto esta configuração perde para o AG (Algoritmo Genético), algoritmo que já estava implementado no código fonte do Mario AI Competition. Além disso foi possível observar que o tipo de topologia não faz tanta diferença nos resultados e que com o aumento da população os resultados tendem a melhorar.

1 Introdução

Jogos digitais possuem ambientes complexos, dinâmicos, porém com um objetivo claro. A criação de um agente inteligente para solucionar um jogo não é trivial. A competição Mario AI Competition, organizada anualmente, foi criada para que a comunidade possa desenvolver, explorar e estudar soluções para um jogo de plataforma, nos quais o agente precisa realizar uma rota até seu objetivo passando por vários obstáculo. Esse tipo de problema pode ser resolvido utilizando Redes Neurais Artificiais, as quais dependem de um algoritmo que faça o ajuste dos seus pesos, que determina a sua saída. Assim, este trabalho tem como objetivo realizar um estudo sobre a utilização do algoritmo Particle Swarm Optimization para o ajuste dos pesos de uma Rede Neural Artificial na resolução do problema do Mario AI Competition.

Para isso, após a definição de conceitos essenciais dos algoritmos e do problema, serão analisados como o PSO se comporta com diversas configurações de parâmetros e estruturas diferentes. Também iremos analisar como o algoritmo se comporta neste problema utilizando diferentes topologias de vizinhança e diferentes estruturas da rede.

1.1 Redes Neurais Artificiais

Redes Neurais Artificiais (RNA) são modelos simplificados do sistema nervoso humano. Elas são compostas por diversos neurônios artificiais dispostos em camadas e interligados por conexões geralmente associadas a pesos. Essa técnica é utilizada para solucionar diversos problemas pela sua capacidade de aprender e se adaptar aos dados fornecidos. (KAYARVIZHY, KANMANI e UTHARIARAJ, 2013).

1.2 Particle Swarm Optimization

A otimização por enxame de partículas (PSO - Particle Swarm Optimization) é uma metaheurística proposta por (KENNEDY e EBERHART, 1995), inspirada no comportamento social de uma revoada de pássaros, insetos ou de um cardume de peixes. A idéia por trás dessa metaheurística é a de que os indivíduos dentro de um grupo ou de uma sociedade se baseiam em experiências bem sucedidas de si mesmo e dos outros integrantes quando vão tomar decisões (MAIA, 2009).

1.3 Ajuste de pesos numa Rede Neural Artificial utilizando PSO

Numa RNA, um dos critérios mais importantes é o ajuste de pesos. Uma das formas de calcular esses pesos, entre várias outras, é adaptando o PSO. Utilizamos todas as combinações de valores possíveis para cada peso como nosso espaço de busca, cada partícula voa sobre esse espaço de busca tentando encontrar o conjunto ótimo de pesos. Para avaliarmos a fitness duma partícula do PSO, utilizamos a precisão que os pesos da partícula têm ao serem aplicados à RNA. (KAYARVIZHY, KANMANI e UTHARIARAJ, 2013).

2 Problema e Implementação

2.1 Mario AI Competition

A Mario AI Competition é uma competição realizada anualmente desde 2009 para o desenvolvimento e aprendizado do melhor agente para uma versão de Super Mario Bros. `http://julian.togelius.com/mariocompetition2009/` A competição é baseada no Infinite Mario Bros por Markus Persson `https://mojang.com/notch/mario/`, uma cópia do Super Mario Bros open source desenvolvida em Java com geração de fases procedural. Os autores da competição criaram uma interface para controladores e possibilitaram rodar a aplicação sem a interface gráfica, para possibilitar testes e coleta de dados.

Nesta interface que os agentes possuem, a cada passo de tempo, que ocorre 24 vezes por segundo, o agente recebe uma representação do ambiente, tanto as informações de blocos ou inimigos que estão na tela quanto a posição exata do agente e onde ele está no seu pulo. Dado essa representação, o agente deve devolver uma ação, composta de 5 bits, que representam os 5 botões: esquerda, direita, baixo, pulo e correr. Um agente pode ser avaliado em relação ao seu desempenho por diversos fatores, mas o utilizado para a competição foi o quão longe os agentes chegavam em cada fase.

O framework disponibilizado para a competição possui toda a estrutura necessária para o desenvolvimento de novos agentes, além de incluir uma base inicial de alguns agentes simples, incluindo um agente que utiliza RNA. Utilizando esse framework também podemos criar instâncias do problema para os agentes resolverem, que é gerado a partir dos parâmetros fornecidos, como dificuldade da fase, tamanho, tempo limite, inimigos presentes, entre outros.

A competição apresenta um problema dinâmico, exigindo que os agentes consigam interpretar um espaço de estados muito grande e consigam reagir a situações bem diferentes.

2.2 Redes Neurais para resolver o problema do Mario AI

A implementação encontrada para resolver este problema que utiliza RNAs utiliza a idéia de utilizar matrizes para representar o ambiente próximo ao agente e algumas variáveis para melhor descrever o estado que o agente está como entrada e um vetor binário de 5 posições que representa as 5 ações possíveis que o agente pode tomar. Existem três implementações com tamanhos de rede diferentes. Na rede Small utilizamos duas matrizes 3x3, uma para representar as colisões, com 0 para espaço vazio e 1 para espaço sólido, e outro para representar a existência de inimigos, com 0 para espaço vazio e 1 para espaço ocupado por um inimigo, com o centro da matriz sendo a posição atual do agente. Na rede Medium utilizamos duas matrizes semelhantes 5x5, e na Large elas são 7x7. Além dessas duas matrizes, todas as implementações também utilizam uma entrada que indica se o agente está pisando no chão e outra que indica se ele pode ou não iniciar um pulo de onde ele está.

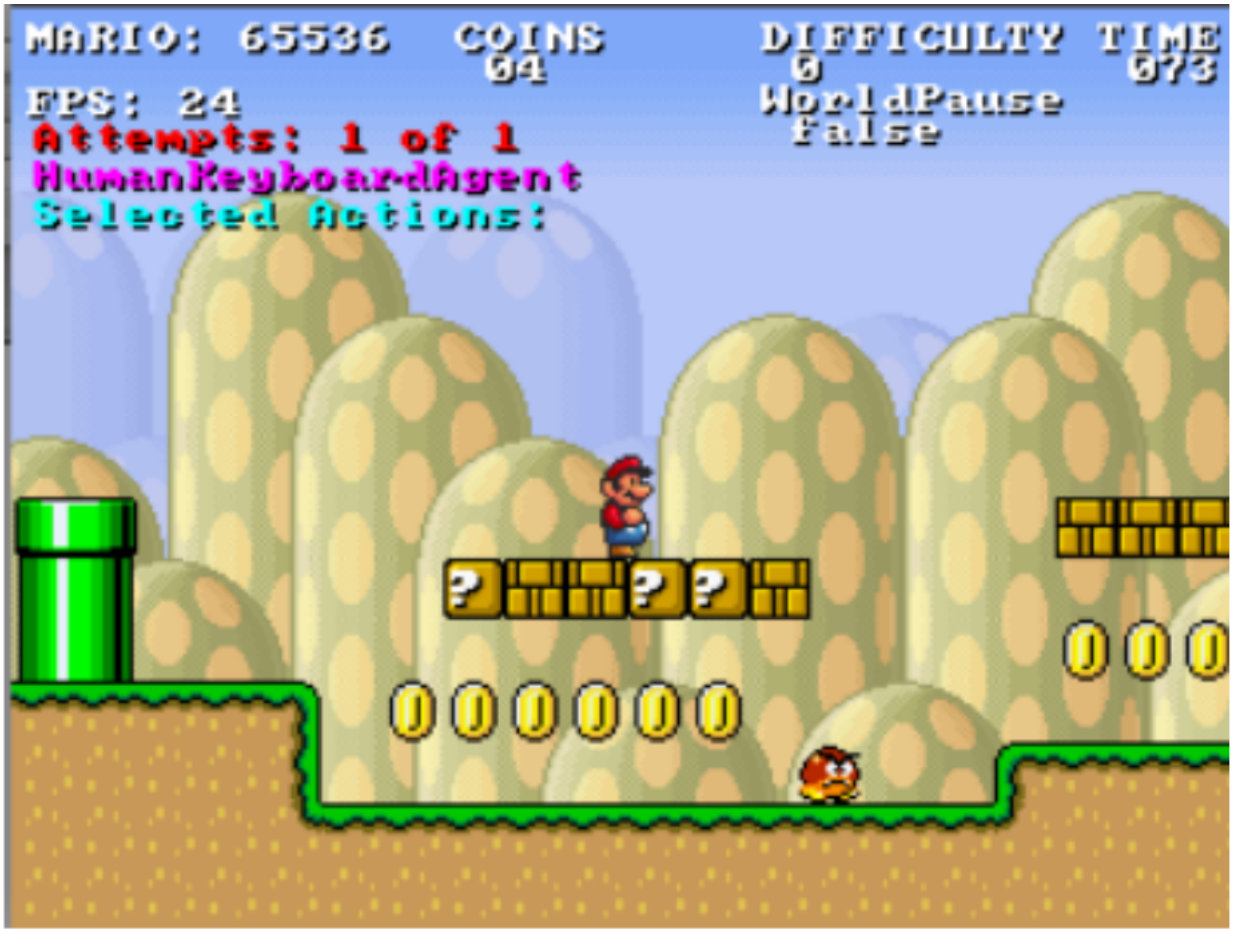

Figura 1: Visualização gráfica do Mario AI Competition

Essas RNAs possuem uma camada escondida com 10 neurônios entre a camada de entrada e de saída. Para cada conexão entre cada entrada e cada neurônio da camada escondida e entre cada neurônio da camada escondida para cada saída, temos um peso.

O algoritmo utilizado para ajuste desses pesos é, por padrão, um algoritmo genético. Esse algoritmo genético possui 100 indivíduos que armazenam todos os pesos da RNA e passam por 100 gerações. A cada geração é feito uma avaliação de todos os indivíduos da população, então é realizado um elitismo de 50% dessa população. Essa elite é então copiada para a outra metade e é feito uma mutação sobre cada uma delas. Essa mutação é feita da seguinte forma: cada um dos pesos do indivíduo é somado com um valor aleatório gerado por uma função que segue uma função de distribuição normal com média igual a zero e com 70% das saídas entre -1 e 1, esse valor aleatório é multiplicado por uma constante de mutação definida em 0,1.

Esta implementação consegue atingir resultados satisfatórios, mas ainda menor do que os agentes baseados em regras simples, pelo fato de terem um comportamento mais determinístico e não dependerem da fase de treinamento, só seguirem uma heurística.

3 Experimento

O foco dos experimentos é fazer a comparação do novo algoritmos com suas múltiplas configurações, e além de compará-los entre si, fazer uma comparação com o AG (Algoritmo Genético) que já estava implementado no código fonte do Mario IA Competition.

Para a estrutura do PSO, escolhemos como critério de parada do algoritmo uma evolução máxima de 100 gerações da população ou o término da fase, onde neste caso é uma fitness com valor igual a 4096. Para o tamanho da população, inciamos os testes com 50 partículas, sendo em seguida trocada por 100 partículas e por fim para 300 partículas. Por fim escolhemos dois tipos de topologias, a vizinhança totalmente conectada (TC), onde uma partícula está conectada com todas as outras partículas da

população, e a vizinhança em anel (Anl), onde pré-definimos dois vizinhos para cada partícula, que não necessariamente são vizinhos geográficos.

Para a rede neural utilizamos dois tipos de tamanhos distintos, que neste caso são a SmallMPL 3x3 e a MediumMPL 5x5, onde somente para recapitular, uma matriz 3x3 representa o ambiente envolta do personagem, com duas matrizes, uma de colisão e uma outra de inimigos. Acabamos por não inserir o tamanho LargeMPL, pois ao se fazer os testes com a SmallMPL., percebemos que os testes com ele acabariam sendo muito custosos e pelo tempo que nos restava isso não seria possível.

Para o algoritmo AG (Algoritmo Genético), utilizamos o algoritmo que já havia sido implementado e que estava contido no código fonte do Mario AI Competition. Esse algoritmo genético possui como condição de parada uma evolução máxima de 100 gerações. Além disso possui 100 indivíduos no tamanho da população com um elitismo de 50% e uma constante de mutação definida em 0,1.

Para os experimentos foi utilizado o ambiente de desenvolvimento intelliJ IDEA no ambiente Linux. O código foi modificado a partir da fonte disponível em `https://code.google.com/p/marioai/`. Rodamos três vezes o processo de treinamento de rede para cada combinação de dificuldade, vizinhança e estrutura de rede, realizando uma média aritmética do quão longe o agente chegou em cada treinamento, a qual foi a métrica utilizada para avaliar cada algoritmo. Cada fase tem 4096 unidades de comprimento e portanto é a pontuação máxima que um algoritmo pode obter.

As fases, apesar de geradas aleatoriamente, são criadas a partir de uma semente, que foi definida em 0 para que todo agente treinado na mesma dificuldade seja treinado na exata mesma fase, para uma comparação mais justa.

4 Análise dos Resultados

Os resultados gerais dos experimentos podem ser vistos na Figura 2, temos uma tabela com as três primeiras colunas em azul claro indicando qual rede foi utilizada, com tamanho da rede, Small 3x3 ou Medium 5x5, a topologia de vizinhança utilizada, e o tamanho da população. Em seguida temos as 11 colunas que representam cada uma das dificuldades diferentes de fases nas quais os algoritmos foram testados. Nas células destas colunas podemos notar os maiores valores de cada coluna em amarelo, os segundos melhores valores em cinza e os piores valores em vermelho.

Dentre as topologias de vizinhança, representamos a Totalmente Conectada como TC e a em Anel por Anl.

É aparente como a implementação padrão do algoritmo genético (AG) possuiu um desempenho melhor. Isto aparenta vir do fato do processo de mutação utilizado ser mais drástico, trazendo mais diversidade, e tendo uma população e gerações suficientes para alcançar um bom resultado.

Podemos notar também que, consistentemente, os resultados os algoritmos que utilizam PSO têm resultados melhores quando o tamanho da sua população é maior. Com base nesse fato iremos basear nossas análises sobre os resultados com população igual a 300.

Após o AG, o algoritmo com PSO que utiliza a estrutura Small, topologia Totalmente Conectado possuiu um desempenho no geral melhor que os outros, mesmo que o mesmo algoritmo com populações mais baixas acumulou o maior número de piores resultados. Assim percebemos que esse tipo de estrutura, mais reativa, pois tem uma visão mais "fechada" do ambiente, necessita duma população maior que os outros para obter os mesmos resultados.

Uma anomalia que pode-se notar na Figura 2 é como várias vezes os algoritmos possuem resultados melhores numa fase de dificuldade maior em relação à outra com dificuldade menor. Como as fases são geradas aleatoriamente, isso pode acabar acontecendo onde uma fase não fica na prática mais difícil apesar dos parâmetros utilizados para gerá-la tiverem um potencial de deixá-la mais difícil.

4.0.1 Impacto da Vizinhança

Pela Figura 3 podemos visualizar uma análise direcionada à topologia de vizinhança. As 2 subtabelas mais a esquerda indicam na primeira coluna a dificuldade e nas duas seguintes os resultados de um determinado algoritmo em cada uma dessas dificuldades, e os quadradinhos que estão em cinza possuem o melhor resultado da comparação entre estas duas colunas. A última linha ("Qtd") indica a

Tam	Viz	Pop	0	1	2	3	4	5	6	7	8	9	10
Small	TC	50	1597	459	371	758	510	626	425	538	613	597	470
Small	TC	100	2925	735	1024	756	507	550	676	574	553	501	484
Small	TC	300	3528	1975	1310	745	1187	1120	682	807	894	787	1267
Small	Anl	50	1692	417	380	719	564	695	607	519	584	573	595
Small	Anl	100	3345	848	779	862	702	590	729	720	623	675	828
Small	Anl	300	3811	1265	1091	1074	776	934	792	813	743	790	847
Medium	TC	50	723	1137	1123	633	675	600	682	667	457	768	627
Medium	TC	100	2719	315	879	854	795	620	851	649	771	524	508
Medium	TC	300	3366	1854	610	1137	1033	721	773	879	769	896	854
Medium	Anl	50	1798	451	464	811	702	714	609	483	590	612	669
Medium	Anl	100	1547	783	609	742	801	699	731	803	616	653	695
Medium	Anl	300	3433	1035	790	1020	951	585	792	982	838	686	798
AG		100	4096	1745	1449	1053	1264	1078	949	1084	894	919	1080

Figura 2: Resultados dos experimentos

contabilização da quantidade de melhores resultados de cada algoritmo. A tabela mais a direita indica o soma de ("Qtd") de TC e Anl da quantidade de melhores resultados de cada sub-tabela.

Pela Figura 3 podemos notar que a topologia não parece ter relação com a estrutura da rede, já que as duas primeiras sub-tabelas possuem subtotais iguais.

Também analisamos que a quantidade de vezes que a topologia em Anel foi melhor que a Totalmente Conectada é bem próxima da Totalmente Conectada, apesar de não superá-la. Por essa diferença ser tão pequena, e muitas vezes os resultados das duas ser bem próximo, concluímos que a topologia não faz uma diferença significativa o suficiente neste caso, onde usamos 100 gerações.

4.0.2 Impacto da estrutura RNA

Também podemos visualizar os resultados focando na estrutura da rede utilizada. Na Figura 4 temos essa comparação, com as diferentes estruturas de rede sendo comparadas, contando quantas vezes o desempenho da estrutura Small ou Medium supera a outra. Na primeira coluna da tabela na Figura 4 temos as dificuldades seguido por 4 colunas que determinam os resultados em cada algoritmo, e os quadradinhos que estão em cinza possuem o melhor resultado da comparação entre estas quatro colunas. A última linha ("Qtd") indica a contabilização da quantidade de melhores resultados de cada algoritmo.

Dif	Small TC 300	Small Anl 300
0	3528	3811
1	1975	1265
2	1310	1091
3	745	1074
4	1187	776
5	1120	934
6	682	792
7	807	813
8	894	743
9	787	790
10	1267	847
Qtd	6	5

Dif	Medium TC 300	Medium Anl 300
0	3366	3433
1	1854	1035
2	610	790
3	1137	1020
4	1033	951
5	721	585
6	773	792
7	879	982
8	769	838
9	896	686
10	854	798
Qtd	6	5

	TC	Anl
Total	12	10

Figura 3: Análise dos resultados com relação à topologia de vizinhança

Na Figura 4 podemos ver como a diferença entre a estrutura Small e Medium é significativa, com 8 vezes o resultado da Small sendo superior à Medium.

A principal diferença entre as duas estruturas é que a Small possui uma visibilidade mais baixa que a Medium. Isso faz com que os agentes que utilizam a estrutura Small sejam mais reativos a situação, só percebendo inimigos quando eles estão bem próximos dele, ou só notando buracos e obstáculos quando ele já está do lado dele. No Medium, como ele tem um campo de visão maior e não consegue diferenciar espaços em relação a posição geográfica deles, ou seja, diferenciar o que está atrás do que está na frente, muitas vezes eles acabam sendo influenciados por objetos que não deveriam modificar sua saída. Com 100 gerações talvez não há tempo suficiente para isso ser aprimorado, porque existem muitas combinações de entradas que são bem diferentes mas que deveriam ter saídas iguais.

4.1 Considerações Finais

Apesar dos algoritmos propostos não tiverem um desempenho melhor que o algoritmo padrão já implementado, muitas informações novas podem ser extraídas dos testes realizados. Notamos que uma das principais fraquezas que todos os algoritmos analisados apresentam é sua saída. A definição de como a saída e a entrada da rede é feita é tão importante quanto à estrutura de seus neurônios, e a que foi utilizada aparenta não ser a ideal.

Um dos maiores problemas encontrados durante a inspeção de certos testes no modo gráfico foi a presença, repetidamente, de situações onde o agente possui como saída uma combinação de ações indesejada, como andar para a esquerda e para a direita simultaneamente, resultando no agente ficando preso. Como o número de saídas possíveis que isso acontece não é baixa, isso acabava sendo comum, principalmente nas primeiras gerações que possuem um pouco da influência da sua geração aleatória.

Uma possível solução para esse problema é, ao invés de utilizarmos como saída o vetor de bits que será passado para o agente, codificarmos de alguma forma uma lista de ações possíveis que o agente pode tomar, e depois transformar essa ação num vetor de bits.

Na Figura 5 podemos ver como apesar do desempenho das outras topologias for, em geral, inferior

Dificuldade	Small TC 300	Small Anl 300	Med TC 300	Med Anl 300
0	3528	3811	3366	3433
1	1975	1265	1854	1035
2	1310	1091	610	790
3	745	1074	1137	1020
4	1187	776	1033	951
5	1120	934	721	585
6	682	792	773	792
7	807	813	879	982
8	894	743	769	838
9	787	790	896	686
10	1267	847	854	798
Subtotal	6	2	2	2
Total	8		4	

Figura 4: Análise dos resultados com relação à estrutura da rede

à AG, elas não estão tão longe dela, mas tendem a "flutuar" bastante de valor. A discrepância grande de valores se deve ao fato da fitness ser o quão longe o agente chegou na fase de 4096 de comprimento. Geralmente existem certos obstáculos mais significativos que exigem mais do agente em certos pontos, de forma que passando de um deles aumenta a fitness consideravelmente.

5 Conclusão

Este trabalho apresentou uma introdução sobre Redes Neurais e o algoritmo PSO que podem ser utilizados em conjunto para resolver diversos problemas. Então apresentamos como o problema do Mario AI Competition pode ser solucionado utilizando essas ferramentas. Com a definição do problema e a base já implementada, diversos parâmetros e modelos foram estudados, implementados e seus resultados analisados, afim de comparar o desempenho do algoritmo PSO com suas diversas configurações com o algoritmo genético já implementado.

Os resultados dos testes nos mostraram que a topologia é irrelevante com relação a fitness escolhida e que o aumento da quantidade de partículas influência diretamente e positivamente nela. Também foi possível observar que o AG (Algoritmo Genético) é melhor que novo algoritmo com ajuste de pesos com o PSO, no entanto numa comparação entre eles, percebeu-se que os resultados ficaram bem próximos, apesar dos resultados da configuração do PSO flutuarem bastante.

Para trabalhos futuros, modificar a estrutura de saída da rede, para evitarmos saídas redundantes, ou expandir a análise dos parâmetros podem prover resultados melhores para o problema. Testes podem ser realizados explorando um número maior de partículas, para analisar se há decaimento do

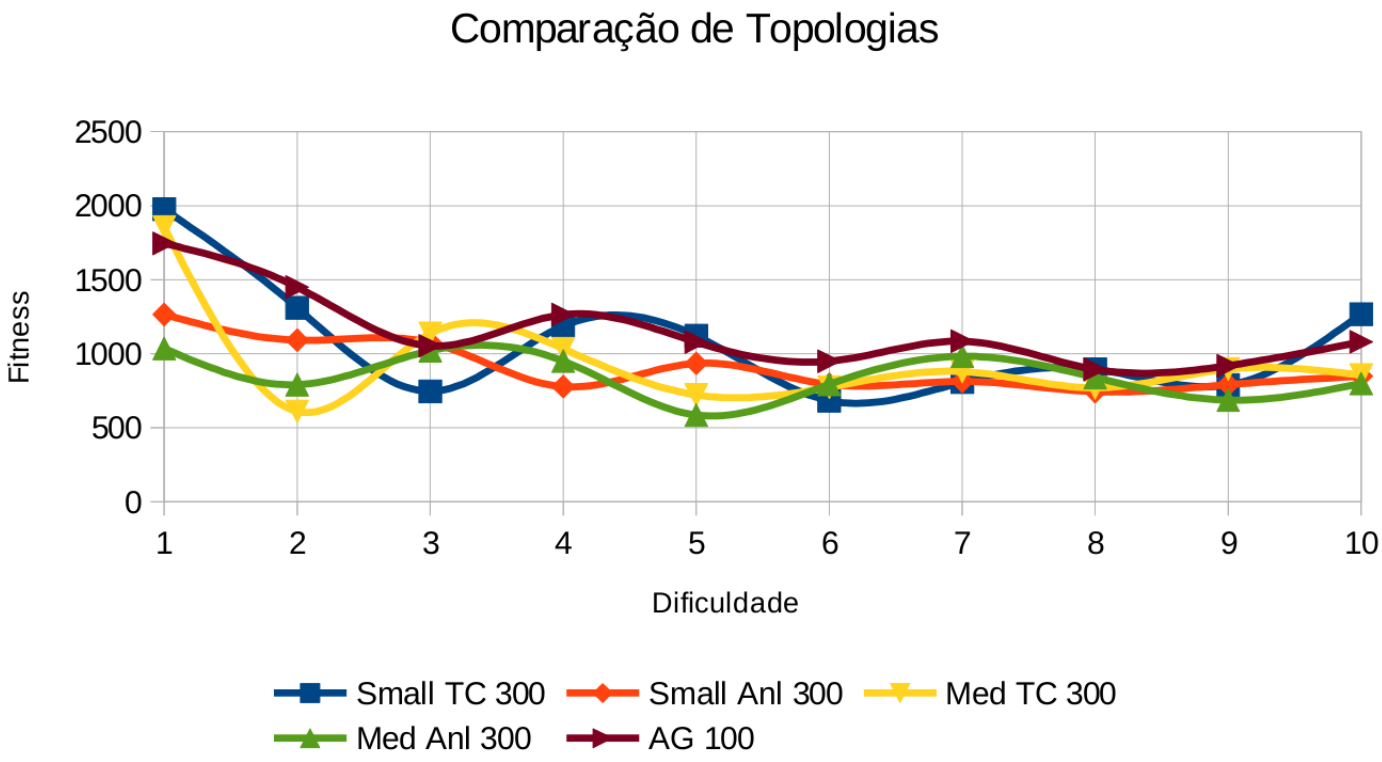

Figura 5: Gráfico de fitness das topologias em cada dificuldade

desempenho do algoritmo depois de um certo limiar, aumentando o número de gerações, notando se a topologia em anel possa ter um desempenho superior ao totalmente conectado, e realizando uma quantidade significativamente maior de casos de teste, para obtermos resultados mais estatisticamente corretos.

O problema do Mario AI Competition é complexo, exigindo que os algoritmos que o resolvam sejam genéricos o suficientes para conseguir resolver as diversas situações possíveis. É um problema dinâmico que testa o algoritmo em ângulos interessantes e pode ser uma boa forma de comparar algoritmos bioinspirados para esses tipos de situações.

Referências

[1] Kennedy J., Eberhart, R., Particle Swarm Optimization, (Proceedings of IEEE International Conference on Neural Networks, 1995).

[2] Maia, P. P. C., Uma Metaheurística Híbrida Paralela para o Problema do Caixeiro Viajante, Universidade Federal do Rio Grande do Norte, Natal/RN, 2009.

[3] Kayarvizhy, N., Kanmani, S., Uthariaraj, V. R., Improving Fault Prediction using ANN-PSO in Object Oriented Systems, (International Journal of Computer Applications, 2013).

[4] Ziyu Chen, Zhongshi He e Cheng Zhang, Chongqing University, Chongqing, China, Particle swarm optimizer with self-adjusting neighborhoods (GECCO '10 Proceedings of the 12th annual conference on Genetic and evolutionary computation, pg.9-14, ACM New York, NY, USA ©2010, 2010).

[5] Xin Yao, Sch. of Comput. Sci., Birmingham Univ., UK, Evolving artificial neural networks (Proceedings of the IEEE (Volume:87 , Issue: 9, pg.1423 - 1447), Sep 1999).

[6] Kenneth O. Stanley, David B. D'Ambrosio e Jason Gauci, A hypercube-based encoding for evolving large-scale neural networks (Journal Artificial Life archive, Volume 15 Issue 2, Spring 2009, pg.185-212, MIT Press Cambridge, MA, USA, 2009).

[7] Julian Togelius IT University of Copenhagen, Copenhagen S, Denmark Sergey Karakovskiy IDSIA, Manno-Lugano, Switzerland Jan Koutník IDSIA, Manno-Lugano, Switzerland Jürgen Schmidhuber IDSIA, Manno-Lugano, Switzerland, Super mario evolution (Proceeding CIG'09 Proceedings of the 5th international conference on Computational Intelligence and Games pg.156-161 IEEE Press Piscataway, NJ, USA ©2009 2009).

[8] Zdeněk Buk, Jan Koutník e Miroslav Šnorek, NEAT in HyperNEAT Substituted with Genetic Programming (Springer Berlin Heidelberg, pg.243-252, 2009).