Departamento de Ciência da Computação

Instituto de Matemática e Estatística

Universidade de São Paulo

Introdução ao Escalonamento e Aplicações

Monografia

Inteligência de enxame e o algoritmo das abelhas

(Swarm Intelligence and Bees Algorithm)

Professor: Alfredo Goldman vel Lejbman

Glaucus Augustus, 6219168

Sumário

Introdução

Algoritmos de otimização baseados em enxame (*swarm-based optimisation algorithms - SOAs*) imitam modelos da natureza para chegar a soluções próximas do ótimo. A principal diferença entre SOAs e outros algoritmos de busca direta como *Hill Climbing* e *Random Walk* é que SOAs utilizam uma população de possíveis soluções para cada iteração ao invés de apenas uma como os outros algoritmos. Esta característica enquadra tais algoritmos como sendo populacionais, ou baseados em população []. Se um problema de otimização tem apenas uma solução ótima, é esperado que a população dos SOAs convirja para tal ótimo. Se um problema tem várias soluções ótimas, SOAs obtem tais soluções nos valores de sua população final. Alguns algoritmos SOAs são: Otimização por colônia de formigas(*Ant Colony Optimisation - ACO*), Otimização por enxame de partículas (*Particle Swarm Optimisation - PSO*), Algoritmos genéticos(*Genetic Algorithms - GA*) e Busca por difusão estocástica (*Stochastic Diffusion Search - SDS*).

A estratégia de todos os algoritmos baseados em população é gerar variações da população (solução) que esta sendo procurada. Alguns métodos utilizam um critério guloso (*greedy*) para decidir qual solução gerada será adota, ou seja, escolhe-se apenas a solução que for aumentar o valor, encontrado até o momento, para a função objetivo. Um algoritmo muito conhecido que não utiliza um critério guloso para a geração de resultados é a otimização por colônia de formigas (ACO).

Formigas são capazes de encontrar o caminho mais curto entre a fonte de alimento e sua colônia, utilizando para tanto uma substancia química gerada por ela mesma, chamada feromônio, que tem o papel de auxiliar na locomoção indicando direções e distancias. O feromônio é deixado no solo por onde a formiga passa formando um rastro. Outras formigas podem seguir tal rastro, dependendo da quantidade de feromônio ali presente, o que é diretamente proporcional a quantidade de formigas que ali passaram e a quanto tempo.

ACO foi utilizada pela primeira vez no contexto de otimização funcional por Bilchev [1]. Outras tentativas são informadas em [1 e 2].

O algoritmo genético é baseado na seleção natural e recombinação genética. O algoritmo funciona escolhendo soluções dentre as existentes na população atual e então aplicando operadores genéticos, como MUTACAO e CROSSOVER, cria uma nova geração (população). O algoritmo explora de forma eficiente o histórico de informações para verificar a existência de áreas de pesquisa com melhor desempenho [3]. Quando aplicado a problemas de otimização, o algoritmo genético tem a vantagem de executar uma busca global. O algoritmo genético também pode ser adaptado, introduzindo outras heurísticas, especificas para o domínio do problema, para melhorar sua performance em relação aos resultados, por exemplo, Mathur el al[4] descreve um híbrido de ACO e GA para otimização de uma função continua.

Otimização por enxame de partículas (*Particle Swarm Optimisation - PSO*) é um procedimento de otimização baseado no comportamento de grupos de organizações, como por exemplo, uma revoada de pássaros ou um cardume de peixes [5]. Soluções individuais

(cada membro em uma população) são consideradas partículas que evoluem e mudam suas posições no espaço de busca conforme sua própria experiência e também na de suas vizinhas, armazenando sempre de sua melhor posição visitada e da melhor posição visitada por suas vizinhas, logo, combinando métodos de busca local e global [5].

Otimização por Colônia de Formigas (*Ant Colony Optimisation - ACO*)

Otimização por colônia de formigas é o nome que se da a uma família de algoritmos que seguem um mesmo padrão de funcionamento, semelhante ao de uma colônia de formigas reais quanto estão em busca de comida. Existem diversos algoritmos diferentes que podem ser considerados como otimização por colônia de formigas, mas o primeiro deles foi o criado por Doringo[6] chamado Ant System. Neste trabalho trataremos todos os algoritmos não por seu nome, mas sim pelo nome de sua base/abordagem.

O algoritmo da otimização por colônia de formigas é um algoritmo de propósito geral que pode ser usado para resolver diferentes problemas de otimização combinatória. É desejável que qualquer algoritmo de otimização baseado na colônia de formigas possua as seguintes características:

- Versatilidade: pode ser aplicado a versões similares do mesmo problema: Ex: O algoritmo pode ser estendido, e passar a resolver além do problema do caixeiro viajante (*Travelling Salesman Problem - TSP*) o problema do caixeiro viajante assimétrico (*Assymetric Travelling Salesman Problem* - ATSP)
- Robustez: Com pequenas modificações pode ser aplicado a outro problema de otimização combinatória como o problema da atribuição quadrática (*Quadratic Assingment Problem - QAP*) e para *Job-shop Scheduling* (*Job-shop scheduling Problem - JSP*)
- Segue a metodologia baseada em população: É interessante, pois permite a exploração do mecanismo de *feed-back* positivo como um mecanismo de busca e também facilita a implementação paralela do algoritmo.

Estas características são contrabalanceadas pelo fato de que o algoritmo da colônia de formigas pode ser superado em desempenho por algoritmos mais específicos. Este problema também é encontrado por outros algoritmos conhecidos como recozimento simulado (*Simulated Annealing - SA*), e busca tabu (*Tabu Search -TS*). Mesmo assim, acredita-se que assim como os algoritmos citados, o algoritmo da colônia de formigas pode ser empregado em variações de problemas bem conhecidos, mas que acabam tornando a implementação do algoritmo com melhor desempenho impossível, como acontece com o ATSP

As “formigas” no algoritmo são tratadas como agentes com capacidades muito simples, que como era de se esperar, tentam imitar o comportamento das formigas do mundo real. De fato, o estudo sobre o comportamento de formigas reais é a grande inspiração do algoritmo da colônia das formigas. Um dos problemas estudados pelos etólogos (aquele que estuda o comportamento animal) é entender como um animal quase cego como as formigas pode estabelecer os melhores caminhos (os mais curtos) de sua colônia até fontes de alimento. Foi descoberto que o método utilizado para estabelecer comunicação entre os

indivíduos e sinalizar caminhos consiste na utilização de feromônio. Uma formiga dispersa feromônio no solo em quantidade diversificada, formando uma trilha. Enquanto uma formiga sozinha se movimenta de forma aleatória, ao encontrar uma trilha de feromônio, tal formiga decide, com uma grande probabilidade, seguir a trilha de feromônio, reforçando-a com o próprio. Quanto mais formigas seguem a trilha, mais atraente a trilha será devido a quantidade de feromônio. Isto é considerado como um mecanismo de feed-back positivo, onde a probabilidade de uma formiga escolher um determinado caminho aumenta de acordo com a quantidade de formigas que escolheram seguir tal caminho anteriormente.

Como exemplo, retirado do artigo de Doringo [6], existe um caminho pelo qual as formigas estão andando, por exemplo, da fonte de alimento A para a colônia E. De repente um obstáculo é introduzido no caminho. Logo, as formigas têm que decidir se vão para a esquerda ou para a direita. A escolha é influenciada pela intensidade do rastro de feromônio deixado pelas formigas que ali passaram anteriormente. Um maior nível de feromônio no caminho da direita faz com que as formigas tenham um estimulo maior, consequentemente uma probabilidade maior de seguir à direita. A primeira formiga a chegar ao ponto B tem a mesma probabilidade de virar à direita ou à esquerda, porque não existe nenhum rastro de feromônio que provoque algum estimulo na mesma. Como o caminho BCD é mais curto que o caminho BHD, a primeira formiga que seguir esta caminho irá chegar ao ponto D antes que a formiga que esta seguindo o outro caminho BHD. O resultado é que a formiga que esta retornando do caminho E para D encontrará um rastro mais forte no caminho DCB, ali deixado pela metade de todas as formigas. Como consequência, o numero de formigas que seguem o caminho BCD por unidade de tempo será maior que o numero de formigas que segue o caminho BHD. Isso faz com que a quantidade de feromônio no caminho mais curto cresça mais rapidamente que no caminho mais longo. Como a tendência de seguir caminhos das formigas é rapidamente inclinada a seguir o menor, todas as formigas acabaram seguindo o menor caminho.

O algoritmo da otimização da colônia de formigas tenta se basear no comportamento das formigas do mundo real, mas com pequenas diferenças:

- As formigas artificiais tem memória;
- Elas não são completamente cegas
- Elas vivem em um ambiente onde o tempo é discreto

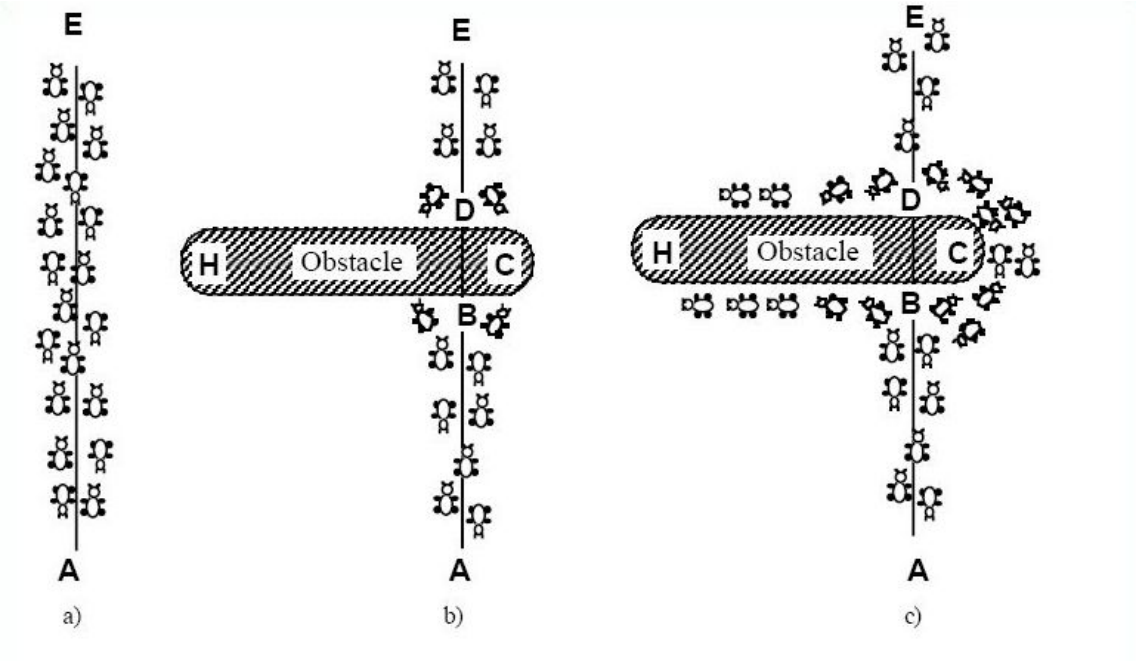

Figura 1 - Extraída de Dorigo[6]
a) Formigas seguem o caminho normalmente
b) Um obstáculo é inserido, e as formigas tem que desviar para a esquerda ou para a direita
c) O caminho mais curto possui mais feromônio

A discussão do funcionamento formal do algoritmo é feita no artigo de Dorigo[6]. Para não fugir do escopo da disciplina de introdução ao escalonamento e aplicações (quanto ao aplicações), optou-se por não demonstrar de maneira matemática que ocorre o funcionamento do algoritmo, confiando por sua vez na explicação através de exemplos e na correlação entre o algoritmo e o modelo natural de uma colônia de formigas.

Dorigo[6] implementou o algoritmo das formigas e o aplicou ao TSP, gerando a Figura 2. O algoritmo da colônia de formigas foi aplicado a uma instancia de 10 cidades do TSP. O comprimento das arestas é proporcional a distância entre as cidades; a espessura das linhas é proporcional ao fluxo da trilha. Inicialmente o fluxo é distribuído em todas as arestas, e a busca é direcionada apenas pela visibilidade do vértice (se o vértice é alcançável). Em seguida, no processo de busca, uma trilha é formada nas arestas que constituem uma boa rota, e o feromônio das arestas não utilizadas evapora completamente. Então as arestas que não fazem mais parte dos melhores caminhos são desconsideradas do grafo, reduzindo o espaço de busca do problema.

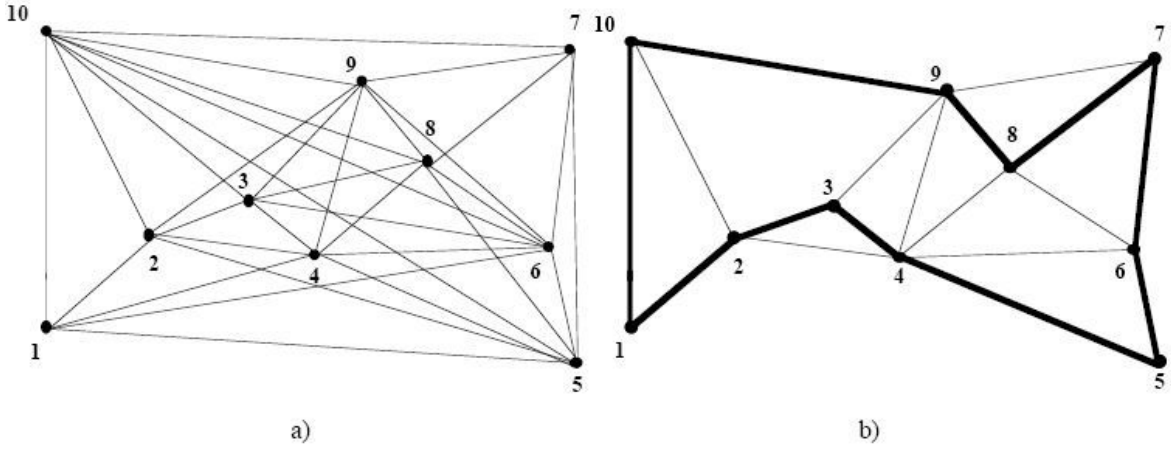

Figura 2 - Extraída de Dorigo[6]
a) Trilhas iniciais alcançáveis pelas formigas
b) Trilas utilizadas após 100 iterações

Escalonamento Job-shop

O escalonamento Job-shop (JSP) pode ser descrito da seguinte forma. Um conjunto de M máquinas e um conjunto de J tarefas são dadas. Para o j-ésima tarefa(j=1, ..., J) consistindo em uma seqüência ordenada de operações de um conjunto O={...ojm...}. Cada operação ojm pertencente a O, pertence a tarefa j e tem que ser processada na maquina m por djm vezes consecutivas. N=|O| é o numero total de operações. O problema é atribuir as operações aos intervalos de tempo sem que se processem duas tarefas ao mesmo tempo na mesma máquina, e o Cmax seja minimizado.

Para aplicar o algoritmo da colônia de formigas ao JSP, segue-se a seguinte representação. Um JSP com M máquina e J tarefas e um conjunto de operações O é representado como um grafo direcionado e com pesos Q=(O', A) onde O'= O união {o0}, e A é o conjunto de arcos que conectam o0 com a primeira operação de cada tarefa, e conecta completamente os vértices de O, exceto os que pertencem a mesma tarefa. Os vértices que pertencem a mesma tarefa são conectados em seqüência, ou seja, um vértice é conectado apenas ao seu sucessor. Note que o gráfico de Q não é o gráfico com cliques representando máquinas que normalmente é utilizado para representar o JSP. O nó O0 é necessário para especificar qual tarefa será escalonada primeiro, no caso de diversas tarefas terem a sua primeira operação na mesma máquina. Temos, portanto, N +1 nós e (N(N-1)/2)+J arcos, onde todos os nós são ligados dois a dois exceto O0, que é conectado somente com a primeira operação de cada tarefa. Cada arco tem o peso definido por um par de números (aij, bij), onde aij é a quantidade de feromônio na trilha e o segundo é a visibilidade e é calculado de acordo com uma medida conveniente derivada de uma heurística específica para problemas gulosos como Maior tempo de processamento ou Menor tempo de término. A ordem em que os nós são visitados por cada formiga especifica a solução proposta.

exemplo, considere um problema 3x2 (3 postos de trabalho, 2 máquinas): seria representado pelo gráfico apresentado na Figura 3.

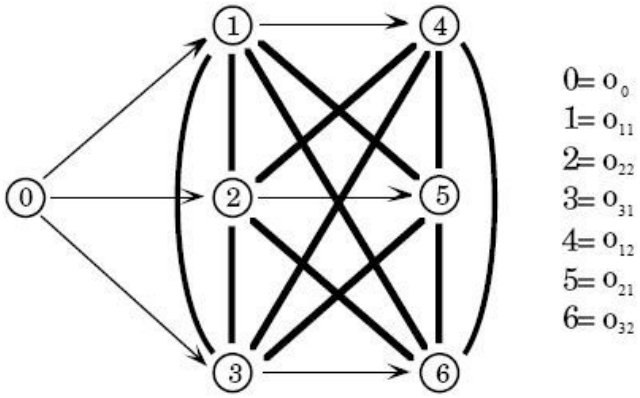

Figura 3 - 3 tarefas em 2 máquinas executando JSP

Todas as formigas estão inicialmente em o0, em seguida elas têm que identificar em cada etapa uma permutação possível dos nós restantes. Para lidar com este problema, as probabilidades de transição tem que ser modificadas. A fim de ter uma possível permutação é de fato necessário definir o conjunto de nós permitido em qualquer etapa, não só através da lista tabu, mas também de uma maneira que depende do problema. Para cada formiga k, consideramos Gk como sendo o conjunto de todos os nós que ainda serão visitados e SK o conjunto dos nós permitidos para a próxima etapa. Inicialmente Gk = (1, 2, 3, 4, 5, 6) e Sk = (1, 2, 3). Quando um nó é escolhido, é anexado à lista tabu e excluídos GK e da SK, se o nó escolhido não é o último de sua tarefa, então o seu sucessor imediato na cadeia de tarefas é adicionado ao Sk.

Este procedimento garante a possibilidade de produzir sempre uma solução viável, possivelmente a ótima. O processo é iterado até Gk = vazio. No final, a ordem dos nós na permutação dada pela lista tabu especifica a solução proposta pela formiga k. As trilhas podem, portanto, ser computadas da forma habitual.

Por exemplo, suponha que uma formiga resultou a solução P = (0, 1, 4, 2, 5, 3, 6); isso irá impor diretamente a ordem das operações a suas precedentes ((1,5), (1,3), (5,3)) e ((4,2), (4,6), (2,6)), respectivamente. Esta abordagem foi implementada e aplicada com sucesso em casos de dimensão JSP de 10x10 e 10x15 (10 postos de trabalho, 15 máquinas). Para cada um destes problemas que sempre obteve uma solução dentro de 10% do melhor, o que pode ser considerado um resultado promissor.

Otimização por Enxame de Partículas (*Particle Swarm Optimization - PSO*)

Otimização por enxame de partículas(*Particle swarm optimization - PSO*) é uma técnica estocástica baseada em população de soluções desenvolvida por Eberhart e Kennedy em 1995[5], inspirada no comportamento social da revoada de pássaros e de cardume de peixes.

PSO tem muitas características em comum com técnicas evolucionárias como algoritmos genéticos. O sistema é inicializado com uma população de soluções aleatórias e procura por um resultado ótimo melhorando gerações. Entretanto, diferentemente de algoritmos genéticos, PSO não tem operadores de evolução como *crossover* e *mutação*. Em PSO as soluções potenciais, chamadas de partículas, voam através do espaço do problema seguindo as então melhores partículas (que possuem os melhores valores no momento).

Comparado com algoritmos genéticos, PSO tem vantagens como ser fácil de implementar e existirem poucos parâmetros para serem ajustados. PSO vem sendo aplicada com sucesso em muitas áreas como otimização de funções, treinamento de redes neurais artificiais, controle de sistemas de lógica *Fuzzy*, e outras áreas onde se pode aplicar algoritmos genéticos.

Atualmente existem muitas técnicas computacionais baseadas em sistemas biológicos como redes neurais, que são uma simplificação do modelo do cérebro humano; algoritmos genéticos, que são inspirados na evolução humana.Existem também os modelos que são baseados em outros sistemas biológicos, os sistemas sociais, mais especificamente o comportamento coletivo de indivíduos simples interagindo com o ambiente e com outros indivíduos.

Na inteligência computacional são dois os mais conhecidos modelos baseados em enxames. Colônia de formigas e Enxame de partículas. Colônia de formigas é inspirado no comportamento de formigas do mundo real, e é melhor aplicado na otimização de problemas discretos[8].

O conceito de enxame de partículas tem origem na simulação de um sistema social simplificado. O objetivo inicial era simular graficamente a coreografia de um pássaro em uma revoada, entretanto, percebeu-se que o método poderia ser utilizado como otimizador[9].

No início, PSO simulava o comportamento de uma revoada de pássaros. Suponha o seguinte cenário: Um grupo de pássaros esta procurando aleatoriamente por comida por perto. Existe apenas uma fonte de comida na área onde esta sendo efetuada a busca. Nenhum pássaro sabe onde a comida esta, mas eles sabem o qual longe esta a comida a cada iteração de tempo (considerando um tempo discreto). Logo, a melhor estratégia para encontrar a comida é seguir o pássaro que esta mais perto dela.

PSO tenta imitar este cenário e utilizar isso para resolver problemas de otimização. Em PSO, casa solução separada corresponde a um pássaro no espaço de busca. Chama-se de partícula. Todas as partículas possuem resultados (*fitness*) que são verificados utilizando a função objetivo, e possuem velocidades que direcionam o vôo das partículas. As partículas voam através do espaço do problema seguindo as partículas que possuem a solução ótima atual.

PSO é inicializado com um grupo de partículas aleatórias (soluções) e então procuram pela solução ótima melhorando suas gerações. Em cada iteração, cada partícula é melhorada seguindo 2(dois) "melhores" valores. O primeiro é o melhor resultado que esta partícula encontrou anteriormente, chamado de pBest. O outro valor que é seguido pela partícula é o melhor valor obtido por qualquer outra partícula da população, e é chamado de gbest. Quando o problema requer uma abordagem subdividida em áreas, são adotadas vizinhanças, e o melhor valor obtido por todas as partículas de uma determinada vizinhança é lbest.

Depois de encontrar os dois melhores valores, a partícula atualiza sua velocidade e posição seguindo as duas equações:

1. v[] = v[] + c1 * rand() * (pbest[] - present[]) + c2 * rand() * (gbest[] - present[]) (a)
2. present[] = persent[] + v[] (b)

Onde v[] é a velocidade da partícula, persent[] é a solução atual da partícula. pbest[] e gbest[] são definidos antes. rand() é um numero aleatório entre 0 e 1. c1, c são fatores de aprendizado e usualmente são iguais a 2.

O pseudo código do algoritmo:

```
For each particle
   Initialize particle
END
Do
   For each particle
      Calculate fitness value
      If the fitness value is better than the best fitness value (pBest) in history
         set current value as the new pBest
   End

   Choose the particle with the best fitness value of all the particles as the gBest
   For each particle
      Calculate particle velocity according equation (a)
      Update particle position according equation (b)
   End
While maximum iterations or minimum error criteria is not attained
```

As velocidades das partículas em cada dimensão são elevadas ao valor máximo, vmax. Se a soma das acelerações forem fazer com que a velocidade naquela dimensão ultrapasse o valor vmax, que é um parâmetro especificado pelo usuário, então esta velocidade é limitada a vmax.

Comparações entre o algoritmo genético e PSO

A maioria das técnicas baseadas em evolução (evolucionárias) executam o seguinte procedimento:

1 - Geram aleatoriamente uma população inicial

2 - Calculam o melhor resultado (*finess*)

3 - Reproduzem a população baseados nos valores encontrados no passo anterior

4 - Se os critérios forem atendidos, então para, caso contrario, retorna ao passo 2

Com base nisto, pode-se concluir que PSO tem muito em comum com algoritmos genéticos. Ambos iniciam com uma população gerada de forma aleatória, ambos se baseiam em resultados da função objetivo para verificar a qualidade (acurácia)da população. Ambos atualizam a população com técnicas randômicas e nenhum dos dois algoritmos garante sucesso.Entretanto, PSO não tem operadores genéticos como *crossover* e mutação. Partículas atualizam a si mesmas com velocidade interna, elas também tem memória, o que é importante para o algoritmo.

Comparado com algoritmos genéticos, o mecanismo de compartilhamento de informação de PSO é significativamente diferente. Em algoritmos genéticos, os cromossomos compartilham informação uns com os outros. Então toda a população se move como um grupo em busca da área ótima. Em PSO, apenas o gBest, ou Lbest dependendo do escopo do problema, é compartilhado entre todos constituindo um mecanismo de comunicação de sentido único. A evolução leva em consideração apenas o melhor valor. Comparado com algoritmos genéticos, todas as partículas tendem a convergir para a melhor solução rapidamente, mesmo em versões que consideram buscas locais na maioria dos casos.

Redes neurais artificiais e PSO

Uma rede neural artificial é um paradigma de análise que simplifica o modelo do cérebro humano e o algoritmo de propagação reversa é um dos métodos mais populares para se treinar uma rede neural artificial. Recentemente houve pesquisas para aplicar técnicas de computação evolucionária com o propósito de evoluir um ou mais aspectos em uma rede neural artificial.

Métodos de computação evolucionária vem sendo aplicados a três principais atributos de redes neurais artificiais: peso das conexões da rede, arquitetura da rede (topologia, função de transferência) e algoritmos de aprendizado.

A maioria do trabalho envolvendo a evolução de redes neurais artificiais possui foco no peso das redes e na estrutura topológica. Usualmente os pesos e as estruturas topologias são representados por cromossomos em algoritmos genéticos. A seleção da função objetivo depende dos objetivos da pesquisa. Para um problema de classificação, a medida dos padrões não classificados pode ser considerada como um valor objetivo (*fitness value*), ou seja, um valor para o qual pode-se tentar otimizar a função.

Existem vários artigos que descrevem a utilização de PSO para substituir o algoritmo de aprendizado da propagação reversa. Isso mostra que PSO é um método promissor para treinar redes neurais. É mais rápido e alcança melhores resultados na maioria dos casos.

Um exemplo simples de evolução de uma rede neural artificial com PSO é uma função de referência (*benchmark*) do problema de classificação chamado conjunto de dados Iris.

Medições de quatro atributos de flores de íris são fornecidas para cada conjunto de dados de registro: comprimento da sépala (parecida com pétala, mas é menor e tem a função de proteger o botão de flor), largura da sépala, comprimento da pétala, largura da pétala. Cinqüenta conjuntos de medições são apresentados em cada uma das três variedades de flores de íris, para um total de 150 registros, ou padrões.

Uma rede neural de 3 camadas é usada para fazer a classificação. Há 4 entradas e 3 saídas, logo, a camada de entrada possui 4 neurônios e a camada de saída 3. Pode-se evoluir o número de neurônios escondidos. No entanto, vamos supor que a camada oculta possui 6 neurônios. Podemos evoluir outros parâmetros na rede de *feed-forward* (como feed-back, porém para frente). Aqui nós só evoluiremos os pesos da rede. Portanto, a partícula vai ser um grupo de pesos, há 4 * 6 +6 * 3 = 42 pesos, de modo que a partícula é constituída por 42 números reais. A gama de pesos pode ser definida como [-100, 100] (este é apenas um exemplo, em casos reais, pode-se tentar diferentes gamas). Após a codificação das partículas, é preciso determinar a função objetivo (*fitness*). Para o problema de classificação, nós alimentamos todos os padrões da rede, cujo peso é determinado pela partícula, pegamos os resultados e comparamos com as saídas padrão. Então, registramos o número de padrões erroneamente classificados como o valor objetivo das partículas. Agora podemos aplicar PSO para treinar a RNA para obter menor número possível de padrões erroneamente classificados.

Não existem muitos parâmetros de PSO que precisam ser ajustados.Só precisa-se ajustar o número de camadas escondidas e o intervalo dos pesos para obter melhores resultados em ensaios diferentes.

PSO parâmetro de controle

A partir do caso acima, podemos aprender que há duas etapas fundamentais na aplicação do PSO para problemas de otimização: 1 - a representação da solução e 2 - da função objetivo. Uma das vantagens da PSO é ter números reais como partículas. Não é como GA, que precisa mudar a codificação binária, ou operadores genéticos especiais têm de ser utilizados. Por exemplo, podemos tentar encontrar a solução para f (x) = x1 ^ 2 + x2 ^ 2 + x3 ^ 2, a partícula pode ser definida como (x1, x2, x3), e função de aptidão é f (x). Então nós podemos usar o procedimento padrão para encontrar o melhor. A busca é um processo de repetição, e

os critérios de parada são: que o número máximo de iterações seja atingido ou a condição de erro mínimo seja satisfeita.

Não existem muitos parâmetros que precisam ser ajustados em PSO. Aqui está uma lista dos parâmetros e seus valores típicos.

O número de partículas: o alcance típico é de 20 - 40. Na verdade, para a maioria dos problemas 10 partículas é grande o suficiente para obter bons resultados. Para alguns problemas difíceis ou especiais, pode tentar um 100 ou 200 partículas tão bem.

Dimensão das partículas: É determinado pelo problema a ser otimizado,

Faixa de partículas: É também determinado pelo problema a ser otimizado, você pode especificar intervalos diferentes para diferentes dimensões das partículas.

Vmax: ela determina a variação máxima de uma partícula pode tomar durante uma iteração. Geralmente defini-se o intervalo da partícula como Vmax por exemplo, a partícula (x1, x2, x3) X1 pertence [-10, 10], em seguida, Vmax = 20

Fatores de aprendizagem: C1 e C2 geralmente igual a 2. No entanto, outras configurações também podem ser utilizadas em papéis diferentes. Mas geralmente C1 e C2 são iguais variando entre [0, 4].

A condição de parada: número máximo de iterações que a PSO pode executar e atingir a exigência mínima de erro. Por exemplo, para a formação da rede neural artificial na seção anterior,definimos a exigência mínima de erro. O número máximo de iterações é definido para 2000. Esta condição de parada depende do problema a ser otimizado.

Versão global versus versão local: introduziu-se duas versões de PSO. A versão global e a versão local. A global é mais rápida, mas pode convergir para o melhor local para alguns problemas. Já a versão local é um pouco mais lenta, mas não dificilmente fica presa em um ótimo local. Pode-se utilizar a versão global para obter resultados rápidos e usar a versão local para refinar a busca.

Busca por Difusão Estocástica (*Stochastic Diffusion Search - SDS*)

Ultimamente vem crescendo o interesse por métodos computacionais que utilizam interação entre simples agentes (ex: algoritmos evolucionários, otimização por enxame de partículas, otimização por colônia de formigas, etc.). Alguns desses sistemas de inteligência de enxame são elaborados através da observação e implementação direta do comportamento de insetos sociais, como formigas e abelhas. Por exemplo, o algoritmo da otimização por colônia de formigas utiliza comunicação através de uma trilha ou rastro de feromônio. Esta forma de comunicação indireta, baseada na modificação física do estado do ambiente recebe o nome de comunicação *stigmergetic*. A habilidade de resolução de problemas destes algoritmos surge a partir de um mecanismo de feedback positivo e características espaciais e temporais do sistema de recrutamento em massa por feromônio que eles implementam. Entretanto, em contraposição aos mecanismos de comunicação do tipo *stigmergetic* esta o mecanismo de busca por difusão estocástica, que também é uma metaheurística baseada em inteligência de enxame [De Meyer et al, 2006], que usa comunicação direta entre agentes, parecida com a comunicação dois a dois estabelecida por um tipo especifico de formigas chamadas *Leptothorax Acervorum*.

O mecanismo de busca por difusão estocástica foi proposto pela primeira vez por Bishop (1989) como um algoritmo baseado em população para encontrar padrões com melhor ajuste. Este tipo de algoritmo pode ser reformulado e transformado em um algoritmo de otimização com a alteração da função objetivo, F(x) para uma hipótese (x) sobre a localidade que melhor se encaixa na solução, como a similaridade entre o padrão objetivo e a região correspondente a posição (x) no espaço de pesquisa, e procurando um x tal que F(x) é máximo.

Geralmente, SDS podem ser aplicados mais facilmente a problemas de otimização onde a função objetivo pode ser decomposta em componentes que poder ser verificados de forma independente. Para encontrar o ótimo de uma função objetivo dada, SDS emprega enxames de agentes, cada um possuindo uma hipótese (xi) sobre o ótimo. O SDS repete a iteração de testar e difundir até que um enxame de agentes convirja para uma hipótese ótima

O algoritmo

INITIALISE(agents)

REPEAT

    TEST(agents)

    DIFFUSE(agents)

UNTIL CONVERGENCE(agents)

INITIALISE: Tipicamente a hipótese de cada agente é uniformemente selecionada de forma aleatória dentro do espaço de pesquisa. Entretanto, qualquer informação sobre probabilidade de solução pode ser usada na primeira seleção de hipótese.

TEST: Para cada agente no enxame que possui a hipótese (xi), uma função de teste (booleana) retorna verdadeiro se uma seleção parcial aleatoriamente selecionada da função objetivo em (xi) é indicativo de uma boa hipótese. Por exemplo, em procura por cadeias de palavras, data uma posição hipotética (xi), a função de teste pode retornar verdadeiro se o fragmento j da cadeia procurada esta presente na posição xi;j. Se o teste retornar verdadeiro, o agente se torna um membro ativo do enxame, caso contrário, se torna um membro passivo.

A "nota-teste" de uma dada hipótese (xi) é a probabilidade que a função de teste tem de retornar verdadeiro, e portanto, um fragmento representativo da função objetivo F(xi).

DIFFUSION: As hipóteses são comunicadas através de um mecanismo entre agentes. No modelo tradicional de SDS, agentes passivos se comunicam com outro membro selecionado aleatoriamente dentro do enxame. Se o agente é ativo, ele transmite sua hipótese para agentes passivos. Em outro caso, o agente passivo reinicia seu valor aleatoriamente.

CONVERGENCE:O maior número de agentes com a mesma hipótese define a solução ótima. Para detectar a convergência existem duas maneiras: A primeira é checar se o maior conjunto de agentes formado excedeu um certo número definido, verificando que o tamanho do grupo esta estocasticamente estabilizado; O outro é checar o número de agentes que ainda estão ativos, ou seja, que ainda não encontraram uma solução. Quando esse número atinge um valor pequeno, pode-se considerar que o enxame se estabilizou estocasticamente.

O algoritmo SDS já é paralelizável a princípio, entretanto, na prática, executar o SDS em paralelo pode ser complicado devido a:

1 - Cada agente precisa se comunicar com todos os outros

2 - O volume de comunicação entre os agentes é grande.

Uma solução para este tipo de problema é agrupar os agentes em uma grade com comunicação direta apenas entre os k vizinhos mais próximos. De forma alternativa o enxame de agentes pode ser dividido em vários sub enxames de agentes, cada um rodando em um processador e totalmente conectados, com pouca freqüência de comunicação entre estes sub enxames(De Meyer et al, 2002).

Estratégias de recrutamento

Estratégias de recrutamento afetam apenas a fase de difusão(DIFFUSION) da SDS, o resto do algoritmo permanece inalterado. As três principais estratégias de recrutamento investigadas são: recrutamento passivo (o mecanismo padrão); recrutamento ativo e recrutamento duplo.

Tal como referido na descrição do algoritmo padrão acima, no recrutamento de agentes passivos seleciona-se agentes ativos para a comunicação das soluções possíveis, cada

agente passivo A seleciona aleatoriamente um outro agente B para se comunicar, e se B está ativo então a hipótese B é comunicada à A.

Recrutamento ativo baseia-se no comportamento das espécies de insetos que pululam ativamente e tentam recrutar outros indivíduos para uma fonte alimentar ou local do ninho. Em algumas espécies (por exemplo, *Leptothorax Acervorum*) formigas irão realizar chamadas na vizinhança duas a duas, onde uma formiga que tem encontrado alimento iterativamente tentar recrutar outras formigas para a fonte. Da mesma forma, as abelhas irão realizar uma dança *waggle* dentro da colméia, a fim de indicar para as outras abelhas a localização de fontes de alimento promissor. No recrutamento ativo durante a fase de difusão, agentes ativos encontram agentes passivos e comunicam a sua hipótese, cada agente ativo A contacta aleatoriamente outro agente B e se B é passivo, então B é recrutado pela hipótese A (A é comunicada ao B). É evidente que, em contraste com o recrutamento passivo, onde, teoricamente, o conjunto ideal pode aumentar para um valor em uma única iteração, para o recrutamento ativo o tamanho do cluster pode, no máximo, aumentar duas vezes por iteração (se a cada agente ativo seleciona um agente passivo).

Mecanismos de recrutamento duplo, ativa e passivo funcionam em simultâneo. Agentes, portanto, ativa e passiva fazem seleções de agentes e podem provocar a transferência de ativos para as hipóteses de agentes passivos. Esta combinação de mecanismos de recrutamento de um sistema é considerado o mais biologicamente plausível e, portanto, de especial interesse. Na prática, o recrutamento duplo pode causar um conflito na hipótese de cessão e, conseqüentemente, a prioridade deve ser dada a um ou outro agente ativo atribuindo uma hipótese para um agente passivo (prioridade ativa) ou passiva um agente copiar uma hipótese de um agente ativo (prioridade passiva) .

Manipulando o processo de alocamento de recursos:

(a): o reequilíbrio da pesquisa para a exploração local

O SDS padrão não tem nenhum mecanismo para explorar a auto-semelhança na função objetivo - uma regularidade exibida por muitos problemas do mundo real: a saber, o fato de que as soluções próximas no espaço de solução, muitas vezes têm valores similares de função objetivo. No entanto, um mecanismo de introdução de pequenas variações sobre a diversidade de hipóteses já presentes no enxame podem ser facilmente incorporado no algoritmo. Uma possibilidade é perturbar (modificar) a cópia dos parâmetros das hipóteses, adicionando um pequeno deslocamento aleatório durante a replicação de uma hipótese na fase de difusão, bem como a mutação em algoritmos evolutivos. O seu efeito é manchar grandes aglomerados de agentes locais e seus vizinhos no espaço de solução. Ele permite que o processo de SDSseja utilizado para implicitamente executar hill-climbing - resultando em melhoria dos tempos de convergência nos espaços de solução com a auto-semelhança , bem como no monitoramento de picos de movimento nos problemas *nonstationarly* ou dinâmico.

(b): o reequilíbrio da pesquisa para a exploração global

As demandas conflitantes de uma variedade de exploração contínua do espaço de soluções - especialmente em ambientes dinâmicos - versus a necessidade de um cluster estável explorando a melhor solução descoberta até agora, não são necessariamente satisfeitas da maneira mais ideal por SDS-padrão. Seu processo de atribuição é voraz: uma vez que uma boa solução é detectada, uma grande parte do enxame é alocada para a sua exploração.

SDS sensíveis ao contexto diferem da SDS padrão apenas em sua fase de difusão de agentes ativos. Na SDS-padrão, agentes ativos sempre mantém a sua hipótese atual. No contexto SDS sensível cada agente ativo escolhe outro agente de forma aleatória, se o agente selecionado está ativo e apóia a mesma hipótese, então a escolha torna-se agente passivo e pega uma nova hipótese aleatória do espaço de busca. Este mecanismo de auto-regulação contra a formação de aglomerados muito grandes como a probabilidade de que dois agentes ativos com a mesma hipótese comunicam-se durante a fase de difusão aumenta com o tamanho do cluster relativo. Isso introduz um mecanismo de seleção negativa ou feedback negativo para o algoritmo original e, para alguns resultados dos testes como também permite manter agregados SDS relativamente grandes em múltiplas soluções semelhantes, perto do ideal.

Algoritmo das Abelhas (*Bees Algorithm - BA*)

Uma colônia de abelhas (*honey bees* - abelhas européias) pode se espalhar em um raio de 10 km para que seja explorado um maior número de fontes de comida [10, 11]. Uma colônia prospéra colocando suas abelhas batedoras (forrageiras - que procuram por comida) em bons campos. No início, grandes campos de flores, que contem muito néctar e pólen dever ser visitados com mais freqüência e por mais abelhas, já aos que não tem tanta abundancia, é dedicado menos esforço [12, 13].

O processo de procura por comida começa na colônia com as abelhas batedoras sendo enviadas para locais promissores. Tais abelhas batedoras se movem randomicamente de um local a outro. Na época da colheita (captação de comida), uma colônia continua sua exploração mantendo uma porcentagem de sua população como abelhas batedoras [11]

Quando as abelhas batedoras retornam à colméia, aquelas que encontraram campos vantajosos (segundo um critério composto por vários fatores, entre eles a distância) depositam o pólen colhido e vão para um local dentro da colméia conhecido como salão de dança, onde executam uma dança conhecida como *waggle* dance [10].

Tal dança é essencial para a comunicação da colônia e contém três partes principais quanto ao campo encontrado: a direção, a distância e sua qualidade (*fitness*)[10, 13]. Essa informação ajuda a colônia a enviar suas abelhas para os campos de flores precisamente, sem utilizar guias ou mapas. Cada conhecimento individual é adquirido unicamente a partir da *waggle* dance. Tal dança possibilita que a colônia avalie o mérito relativo de diferentes campos de flores de acordo com ambos, qualidade da comida lá contida, e a quantidade de esforço necessário para obtela [13].

Após a dança, a abelha "dançarina" retorna ao campo que encontrou seguida por abelhas colhedoras que estavam apenas esperando dentro da colméia. Uma maior quantidade de abelhas colhedoras é enviada ao campo que é mais promissor, permitindo a colônia obter comida de forma rápida e eficiente. A monitoração do recurso contido no campo, bem como da produção do mesmo, é dada pela *waggle* dance, que é repetida sempre que a abelha batedora retorna, ou da busca ou do campo de colheita em que esta. Se o campo continua vantajoso, a colheita continua, caso contrário, é escolhido outro campo para o qual o esforço será concentrado.

O algoritmo

O algoritmo das abelhas é um algoritmo de otimização inspirado pelo comportamento natural de forrageamento (procura por comida) de abelhas para encontrar a melhor solução

[5]. Abaixo esta descrito o pseudo código elaborado por Pham [14] para o algoritmo em sua forma mais simples. O algoritmo requer um certo número de parâmetros, a saber: número de abelhas batedoras (n), o número de locais selecionados de n locais visitados (m), número de melhores locais de m locais selecionados (e), número de abelhas recrutadas para o melhor local(NEP), número de abelhas recrutadas para o outros (me) locais selecionados (NSP), o tamanho inicial dos campos (ngh) que inclui local e sua vizinhança e critério de parada. O algoritmo inicia com o n abelhas batedoras sendo colocadas aleatoriamente no espaço de busca. As aptidões dos locais visitados pelas abelhas são avaliadas na etapa 2.

1. Initialise population with random solutions.

2. Evaluate fitness of the population.

3. While (stopping criterion not met)

//Forming new population.

4. Select sites for neighbourhood search.

5. Recruit bees for selected sites (more bees

for best e sites) and evaluate fitnesses.

6. Select the fittest bee from each patch.

7. Assign remaining bees to search randomly

and evaluate their fitnesses.

8. End While.

No passo 4, as abelhas que possuem campos mais vantajosos são escolhidas como "abelhas selecionadas" e os campos que foram visitadas por elas são escolhidos para terem suas vizinhanças incluídas na busca (*neighbourhood search*). Nos passos 5 e 6, o algoritmo conduz buscas na vizinhança dos campos selecionados, alocando mais abelhas para procurar próximo aos melhores campos.As abelhas podem ser escolhidas diretamente de acordo com o resultado associado aos campos que elas estão visitando. De forma alternativa, os resultados dos campos obtidos podem ser utilizados para influir na probabilidade da abelha ser escolhida. Busca na vizinhança dos melhores e campos que representam melhores soluções são feitas de forma mais detalhada, recrutando mais abelhas para tanto. Juntamente com a operação exercida pelas abelhas batedoras, esta operação de recrutamento é um dos principais passos do algoritmo das abelhas.

No passo 6, para cada fragmento de campo, apenas uma abelha com o melhor resultado será selecionada para formar a próxima população. Na natureza não existe tanta restrição. Tal restrição é introduzida no algoritmo para não existirem muitos pontos a serem

explorados. No passo 7, as abelhas restantes na população são enviadas a espaços aleatoriamente escolhidos dentro do espaço de solução para encontrar novas soluções em potencial. Estes passos são repetidos até que um critério de parada seja atingido. No final de cada iteração a colônia possuirá duas partes na sua nova população, as abelhas que são representativas de acordo com os fragmentos selecionados e as abelhas batedoras que conduzem buscas aleatórias

Pham efetuou testes para comparar a performance do algoritmo das abelhas com outros algoritmos de otimização, tal teste esta representado na **Erro! Fonte de referência não encontrada.**.

O algoritmo das abelhas foi testado em 8 funções para que sua performance fosse avaliada e comparada com a performance de outros algoritmos de otimização. Mais resultados e informações a respeito dos testes são encontrados em [15]. A Tabela 2 representa os resultados obtidos pelos algoritmos: algoritmo das abelhas, Simplex determinístico (*SIMPSA*)[2]. recozimento simulado estocástico (*stochastic simulated annealing - NE SIMPSA*)[2], algoritmo genético (GA)[2] e Otimização por colônia de fórmicas(ANTS)[2].

No	Function Name	Interval	Function	Global Optimum
1	De Jong	[-2.048, 2.048]	$\max F = (3905.93) - 100(x_1^2 - x_2)^2 - (1 - x_1)^2$	X(1,1) F=3905.93
2	Goldstein & Price	[-2, 2]	$\min F = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $X[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	X(0,-1) F=3
3	Branin	[-5, 10]	$\min F = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1-f)\cos(x_1) + e$ $a = 1, b = \frac{5.1}{4}\left(\frac{7}{22}\right)^2, c = \frac{5}{22}X7, d = 6, e = 10, f = \frac{1}{8}X\frac{7}{22}$	X(-22/7,12.275) X(22/7,2.275) X(66/7,2.475) F=0.3977272
4	Martin & Gaddy	[0, 10]	$\min F = (x_1 - x_2)^2 + ((x_1 + x_2 - 10)/3)^2$	X(5,5) F=0
5	Rosenbrock	(a) [-1.2, 1.2] (b) [-10, 10]	$\min \ F = 100\ (x_1^2 - x_2)^2 + (1 - x_1)^2$	X(1,1) F=0
6	Rosenbrock	[-1.2, 1.2]	$\min F = \sum_{i=1}^{3} \{100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2\}$	X(1,1,1,1) F=0
7	Hyper sphere	[-5.12, 5.12]	$\min \ F = \sum_{i=1}^{6} x_i^2$	X(0,0,0,0,0,0) F=0
8	Griewangk	[-512, 512]	$\max \ F = \frac{1}{0.1 + \left(\sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)}$	X(0,0,0,0,0,0,0,0,0,0) F=10

Tabela 1 - Extraida de Pham[14] Funções de testes

func no	SIMPSA		NE SIMPSA		GA		ANTS		Bees Algorithm	
	succ %	mean no. of evaluations	succ %	mean no. of evaluations	succ %	mean no. of evaluations	succ %	mean no. of evaluations	Succ %	mean no. of evaluations
1	****	****	****	****	100	10160	100	6000	100	**868**
2	****	****	****	****	100	5662	100	5330	100	**999**
3	****	****	****	****	100	7325	100	1936	100	**1657**
4	****	****	****	****	100	2844	100	1688	100	**526**
5a	100	10780	100	4508	100	10212	100	6842	100	**631**
5b	100	12500	100	5007	****	****	100	7505	100	**2306**
6	99	21177	94	3053	****	****	100	8471	100	**28529**
7	****	****	****	****	100	15468	100	22050	100	**7113**
8	****	****	****	****	100	200000	100	50000	100	**1847**

**** Data not available

Tabela 2 - Exraida de Pham[14] Resultados da comparação

O critério de parada adotado foi de quando a diferença entre o resultado máximo obtido e o ótimo global for menor que 0.1% do valor ótimo, ou menor que 0.001. Se acaso o ótimo for 0, a solução é aceita se a diferença para com o valor ótimo for menor que 0.001.

A primeira função a ser testada foi De Jong, para a qual o algoritmo das abelhas pode encontrar o ótimo 120 vezes mais rápido que o algoritmo da colônia de formigas e 207 vezes mais rápido que o algoritmo genético, com uma media de sucesso de 100%.

A segunda função foi Goldstein and Price, para a qual o algoritmo das abelhas encontrou o ótimo quase 5 vezes mais rápido que o a colônia de formigas se o algoritmo genético, novamente com 100% de sucesso.

Com a função de Brainin, o algoritmo das abelhas foi 15% mais rápido quando comparado ao algoritmo da colônia de formigas e 77% quando comparado com o algoritmo genético, novamente com 100% de sucesso.

As funções 5 e 6, Rosenbrock, em 2 e quatro dimensões respectivamente. Na versão de duas dimensões, o algoritmo das abelhas obteve 100% de sucesso e uma boa melhora perante os outros métodos (no mínimo duas vezes mais verificações). Na versão com 4 dimensões, o algoritmo das abelhas precisou de mais verificações para chegar a um ótimo com 100% de sucesso. NE SIMPSA pôde encontrar o ótimo com 10 vezes mais avaliações da função objetivo, porém sua porcentagem de sucesso foi de 94%. O algoritmo da colônia das formigas encontrou o ótimo com 100% de sucesso e 3,5 vezes mais rápido que o algoritmo das abelhas.

A função 7 foi o modelo da hyperesfera de seis dimensões. O algoritmo das abelhas precisou de metade do número de verificações da função objetivo quando comparado com o algoritmo genético e um terço quando comparado com o algoritmo da colônia de formigas.

A oitava função de teste foi uma função com 10 dimensões. O algoritmo das abelhas pôde encontrar o ótimo 10 vezes mais rápido que o algoritmo genético e 25 vezes mais rápido que o algoritmo da colônia de formigas, com um sucesso de 100%.

O algoritmo teve 100% de sucesso em todos os casos, ou seja, convergiu para um máximo ou mínimo sem ser iludido por um mínimo local, ultrapassando os outros algoritmos a ele comparado em vários casos quanto ao desempenho de execução e acurácia dos resultados, um dos inconvenientes do algoritmo é o numero de parâmetros por ele utilizados, o que pode ser compensado pela execução de alguns testes. O algoritmo ainda pode ser melhorado em relação aos parâmetros de entrada e aos mecanismos de aprendizado.

Papova Nhina Nhicolaievna, Le Van Thanh propuseram uma implementação do algoritmo das abelhas para otimizar o problema da mochila(Knapsack) multidimensional, que é NP-completo, e compararam a performance do algoritmo implementado com implementações do algoritmo da colônia de formigas e do algoritmo genético.

Eles fizeram duas versões do algoritmo implementado, mudando apenas os seus parâmetros de entrada, o que resultou em melhora da performance do algoritmo.

Os parâmetros de entrada dos algoritmos são verificados na Tabela 3, já as performances podem ser verificadas na Tabela 4.

Parameter	**Bee 1**	**Bee 2**	**Ant**	**Tabu**
Iterations, N_{max}	200	1250	500	50
Population, l	No. of jobs	No. of jobs	No. of jobs	
Alpha, α	1.0	1.0	1.0	
Beta, β	1.0	1.0	1.0	
Rating, ρ_{ij}	0.99	0.99		
Waggle dance scaling factor, A	100	100		
Probability to perform waggle dance, p	0.001	0.001		
Evaporation coefficient, ρ			0.01	
Max. no.. of elite solution	20	20		20
Max. size of tabu list				8

Tabela 3 - Parâmetros de entrada dos algoritmos

Relative Improve-ment	Bee 1	Bee 2	Ant	Tabu	SBP
Mean (%)	12.32	7.74	11.85	8.06	7.42
Min. (%)	0	0	0	0	0
Max. (%)	40.08	29.35	38.24	38.86	37.14
Best solutions	13	19	14	21	23
Execution time	1.95x	1x	1.86x	1.25x	1.59x

Tabela 4 - Desempenho relativo

Conclusão

Dado o escopo da disciplina, procurou-se apresentar de maneira ilustrativa porém clara e ligeiramente aprofundada os principais algoritmos que podem ser classificados como baseados em Inteligência de enxame, como otimização por colônia de formigas, otimização por enxame de partículas, busca por difusão estocástica e o algoritmo das abelhas.

Tentou-se apresentar diversos dados referentes a performance e aplicação dos algoritmos em problemas corriqueiros ou bem conhecidos por todos, como TSP e JSP, e para isso varias partes de artigos e livros, bem como suas figuras e tabelas foram extraídas, para que a melhor forma de explicação se mantivesse preservada nesse trabalho.

Em relação aos algoritmos, a técnica de inteligência de enxame é nova quando comparada com outras técnicas como busca tabu, ou com outros algoritmos como os utilizados para gerar arvores geradoras mínimas em um grafo (mesmo problema resolvido pela colônia de formigas), o que leva a pensar que tais algoritmos tendem a evoluir muito, e ainda estão abertos a novas pesquisas e implementações, e possivelmente podem ser utilizados como sendo melhor solução para determinados casos.

O ultimo algoritmo, o algoritmo das abelhas, data de 2002, ou seja, é muito novo, portanto ainda existem poucos trabalhos, ou uma pequena cobertura cientifica em relação a tal algoritmo, que devido ao seu desempenho e capacidade iniciais tende a ser um algoritmo promissor.

É importante ressaltar que alem deste algoritmos apresentados, existem outros algoritmos sob o mesmo padrão (inteligência de enxame) que não foram abordados, bem como a abordagem realizada sobre estes algoritmos não foi profunda o suficiente para que este trabalho possa servir como orientação definitiva acerca de qualquer um destes algoritmos.

Referências e Bibliografia

1 - Bilchev G and Parmee IC. The Ant Colony Metaphor for Searching Continuous Design Spaces.in Selected Papers from AISB Workshop on Evolutionary Computing. (1995) 25-39.

2 - Mathur M, Karale SB, Priye S, Jayaraman VK and Kulkarni BD. Ant Colony Approach to Continuous Function Optimization. Ind. Eng. Chem. Res. 39(10) (2000) 3814-3822.

3 - Goldberg DE. Genetic Algorithms in Search,Optimization and Machine Learning. Reading:Addison-Wesley Longman, 1989.

4 - Mathur M, Karale SB, Priye S, Jayaraman VK and Kulkarni BD. Ant Colony Approach to Continuous Function Optimization. Ind. ng. Chem. Res. 39(10) (2000) 3814-3822.

5 - Eberhart, R., Y. Shi, and J. Kennedy, Swarm Intelligence. Morgan Kaufmann, San Francisco, 2001.

6 - M. Dorigo, 1992. Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy

7 – (site publicado pelo autor do algoritmo) http://www.swarmintelligence.org/tutorials.php

8 – (site publicado pelo autor do algoritmo) http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html

9 - (site publicado pelo autor doalgoritmo) http://www.engr.iupui.edu/~shi/Coference/psopap4.html

10 - Von Frisch K. Bees: Their Vision, Chemical Senses and Language. (Revised edn) Cornell University Press, N.Y., Ithaca, 1976.

11- Seeley TD. The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies. Massachusetts:Harvard University Press, Cambridge, 1996.

12 - Bonabeau E, Dorigo M, and Theraulaz G. Swarm Intelligence: from Natural to Artificial Systems. Oxford University Press, New York, 1999.

13 - Camazine S, Deneubourg J, Franks NR, Sneyd J, Theraula G and Bonabeau E. Self-Organization in Biological Systems. Princeton: Princeton University Press, 2003.

14 - The Bees Algorithm – A Novel Tool for Complex Optimisation Problems D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri , S. Rahim , M. Zaidi Manufacturing Engineering Centre, Cardiff University, Cardiff CF24 3AA, UK

15 - Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.