ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Bruna Thabata Ribeiro de Souza

UTILIZAÇÃO DO ALGORITMO *PARTICLE SWARM OPTIMIZATION* PARA RESOLVER O PROBLEMA DE *TIMETABLING* NA ELABORAÇÃO DA GRADE DE HORÁRIOS EM UM CURSO SUPERIOR

Palmas

2013

Bruna Thabata Ribeiro de Souza

UTILIZAÇÃO DO ALGORITMO *PARTICLE SWARM OPTIMIZATION* PARA RESOLVER O PROBLEMA DE *TIMETABLING* NA ELABORAÇÃO DA GRADE DE HORÁRIOS EM UM CURSO SUPERIOR

Trabalho apresentado como requisito parcial da disciplina Trabalho de Conclusão de Curso (TCC) do curso de Sistemas de Informação, orientado pelo Professor Mestre Fabiano Fagundes.

Palmas

2013

Bruna Thabata Ribeiro de Souza

UTILIZAÇÃO DO ALGORITMO *PARTICLE SWARM OPTIMIZATION* PARA RESOLVER O PROBLEMA DE *TIMETABLING* NA ELABORAÇÃO DA GRADE DE HORÁRIOS EM UM CURSO SUPERIOR

Trabalho apresentado como requisito parcial da disciplina Trabalho de Conclusão de Curso (TCC) do curso de Sistemas de Informação, orientado pelo Professor Mestre Fabiano Fagundes.

Aprovada em xxxxxxx de 2013.

BANCA EXAMINADORA

_____

Prof. M.Sc. Fabiano Fagundes

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Parcilene Fernandes de Brito

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas

Palmas

2013

AGRADECIMENTOS

Agradeço aos meus pais – Ari e Nina – por me darem todo o apoio nos melhores e piores momentos dessa trajetória, serem o meu alicerce, porto seguro e exemplos a serem seguidos. Agradeço por cada incentivo, toda a preocupação e também por toda a confiança depositada.

Aos meus irmãos, Tiago e Rafael e toda a minha família.

Aos os meus professores – Cristina, Edeilson, Fernando, Jackson, Mádia e Parcilene – pelos ensinamentos passados que vão além das salas de aula. E em especial ao meu professor/orientador/pai, Fabiano Fagundes, por todo apoio e força que me deu. Por, muitas vezes, acreditar em mim mais do que eu mesma, e agradeço também por todos os puxões de orelha, muito bem dados.

Aos meus amigos e colegas de curso, que estiveram comigo durante a caminhada, em especial Cristiane, George Lucas, Felipe Bunto, Rauricio, Wellington e Roneylson.

Agradeço à Fernanda e à tia Iza. À Firi por ser a irmã que eu nunca tive e a tia Iza por ser uma segunda mãe, por terem cuidado de mim nos momentos em que meus pais não puderam estar presentes, por me tratar como parte da família, dando a oportunidade que eu precisava para não desistir.

Agradeço ainda aos meus amigos, Marina, Luiz, Kaio, Lane, Eduardo, Khallil e Rebeca, por me apoiarem e incentivarem, além de compreender a minha ausência, principalmente nesta reta final.

Ao meu primo Eliel, por ser o meu irmão mais legal, por estar presente e pronto para me ajudar no que fosse possível. E aos meus quase tios Renato e Vanubia, por sempre me encorajarem e por serem fontes de inspiração para mim.

E à todas as pessoas que não foram citadas, mas que estiveram comigo, pessoalmente ou não, me ajudando e dando forças para que eu continuasse.

SUMÁRIO

RESUMO

Este trabalho tem por objetivo resolver o problema de *Timetabling* Universitário no Departamento de Computação do Centro Universitário Luterano De Palmas (CEULP/ULBRA), utilizando como técnica o algoritmo de inteligência artificial *Particle Swarm Optization* (PSO). O modelo Universitário é uma variação do *Timetabling* Escolar, e consiste em agendar aulas de determinados cursos utilizando recursos como professores e disciplinas, entre outras, num determinado espaço de tempo, levando em consideração restrições que variam de acordo com as necessidades da instituição. O PSO é um algoritmo baseado em Inteligência de Enxame (*Swarm Intelligence*) que simula o voo de um bando de pássaros em busca de alimento. O desenvolvimento do trabalho foi realizado com a linguagem de programação PHP.

PALAVRAS-CHAVE: PSO, *Timetabling* Universitário, PHP, CEULP/ULBRA

LISTA DE FIGURAS

1 INTRODUÇÃO

Todos os anos instituições de ensino de todo o tipo, nos mais diversos lugares, enfrentam, principalmente no início do ano letivo, uma grande dificuldade para gerar suas grades horárias. Para a criação de um calendário letivo de qualidade é necessário levar em conta diversos fatores, tais como: número de matérias, de alunos e professores; salas de aula disponíveis; tempo de duração de aula; número de aulas por semana; respeitando certo número de restrições.

Esta dificuldade em gerar grades horárias é conhecida na literatura como problema de *Timetabling,* que consiste em associar horários e recursos a eventos (aulas), tentando satisfazer, da melhor forma possível, diversas restrições (BURKE et.al., 2003 apud VIEIRA; MACEDO, 2011).

Gerenciar todas essas informações de forma manual demanda muito esforço e tempo e, dependendo da complexidade do problema, torna-se inviável. Devido a essa complexidade, desde a década de 1960 o problema de *Timetabling* vem sendo investigado e diversas técnicas para sua automatização foram propostas. Algumas destas propostas envolvem a utilização de técnicas de inteligência artificial.

O *Particle Swarm Optimization* (Otimização por Enxame de Partículas) foi desenvolvido por Kenned e Eberhart em 1995, e é um algoritmo de busca estocástica similar aos da computação evolucionária, mas a forma como a informação é utilizada é inspirada não em operadores genéticos, mas na dinâmica de grupos sociais na natureza.

Esta técnica foi desenvolvida após a observação de grupos de pássaros que percorrem o espaço, de forma aparentemente aleatória, em busca de alimento, enquanto seguem o pássaro mais próximo do objetivo. Essa busca coordenada pelo grupo é chamada de inteligência de enxame. Cada partícula representa uma solução no espaço de busca e sua posição é regida por uma equação que, a cada iteração, e de acordo com sua melhor posição encontrada pelo enxame, altera a velocidade da partícula em busca de melhores resultados.

Segundo Montero, Riff e Altamirano (2011), o algoritmo PSO tem sido utilizado para a resolução de problemas de *timetabling*, pois é capaz de gerar soluções de qualidade para o problema, além de incorporar poucos parâmetros.

1.1. Objetivos

1.1.1. Objetivo Geral

O objetivo deste trabalho é o desenvolvimento de um protótipo para um sistema que apresente opções de grades de horários em uma instituição de ensino superior, utilizando o algoritmo *Particle Swarm Optimization* e tendo como parâmetros de entrada critérios pré-determinados pela instituição.

1.1.2. Objetivos específicos

- Realizar um amplo estudo do problema de *Timetabling* na geração de grades de horários;
- Realizar um estudo do algoritmo *Particle Swarm Optimization;*
- Elaborar material de referência sobre os conceitos estudados;
- Analisar o problema de *Timetabling* em uma instituição de ensino superior;
- Desenvolver um protótipo que permita receber critérios de organização de grades de horários e apresente uma solução adequada ao usuário.

1.2. Justificativa

A criação de grades de horários para aulas é um problema enfrentado com frequência por pequenas e grandes instituições de ensino. Esse problema ocorre por diversos fatores e alguns podem variar de instituição para instituição, dificultando a criação de uma solução genérica.

Alguns fatores que podem afetar a elaboração da grade são: quantidade de turmas, quantidade de professores, restrições de alguns professores quanto a algum horário etc. Quanto maior o número de turmas e de restrições, mais complexo se torna o problema, ficando bastante complicado gerenciar todas as variáveis envolvidas, para gerar uma grade sem conflitos de horários, que satisfaça as necessidades de todas as partes envolvidas.

Gerenciar todas as necessidades e restrições manualmente é uma tarefa que demanda muito tempo e nem sempre garante resultados satisfatórios, o que torna

interessante automatizar o processo. Essa automatização pode ser feita através do algoritmo *Particle Swarm Optimization*, por se tratar de um algoritmo de otimização desenvolvido para resolver problemas combinatórios complexos.

1.3. Problema

Levando em consideração a grande quantidade de variáveis existentes para a elaboração da grade de horários de uma instituição de ensino superior, como automatizar esse processo de forma que o tempo seja minimizado e a eficiência seja maximizada?

1.4. Hipótese

Com os critérios determinados pela instituição de ensino superior, é possível desenvolver um sistema que determine uma solução para a grade de horários que atenda as necessidades de cada curso, levando em conta suas restrições e não permitindo conflitos de horários utilizando o *Particle Swarm Optimization.*

2 REFERENCIAL TEÓRICO

Nesta seção serão apresentados os principais conceitos relacionados ao trabalho: *Swarm Intellingece*; o algoritmo *Particle Swarm Optimization*; os problemas de *Timetabling*; e *Timetabling* Universitário.

2.1. *Swarm Intelligence*

De acordo com Zhu e Tang (2010), existe uma variedade de organismos na natureza que possuem a habilidade de buscar alimento de maneira cooperativa, enquanto tentam evitar predadores e outros riscos, o que é chamado de "comportamento de enxame". Esse comportamento é encontrado em pássaros, peixes, formigas, abelhas, cupins e outros tipos de insetos. A vida em sociedade oferece mais chances de sobrevivência a esses organismos do que se eles vivessem de forma isolada.

Segundo Zhu e Tang (2010) e Rosendo (2010) esse tipo de comportamento geralmente não segue comandos de um líder e não possui um sistema hierárquico, mas mesmo não havendo um controle centralizado ou um plano global, cada organismo do enxame segue regras locais de interação para comandar suas ações, o que pode gerar um padrão global de comportamento, dessa maneira o enxame acaba atingindo seus objetivos. Deve-se resaltar, contudo, que os agentes individuais não têm conhecimento explícito de resolução de problemas, sendo que o comportamento inteligente surge (ou emerge) por causa das ações sociais dos agentes (WHITE; PAGUREK, 1998).

Os indivíduos do enxame interagem entre si e com o ambiente para alcançar um determinado objetivo. Por exemplo, na busca por alimento, os indivíduos constantemente trocam informações, para saber que direção seguir, utilizando sua experiência individual e também a experiência de seus companheiros, com base na melhor posição que já ocuparam em relação ao alimento que buscam.

O termo "enxame" é utilizado de maneira genérica para se referir a qualquer coleção estruturada de agentes capazes de interagir, sendo o exemplo clássico um

enxame de abelhas, mas a metáfora pode ser estendida a outros sistemas com arquitetura similar, tais como colônia de formigas, revoada de pássaros, engarrafamento, multidão de pessoas, o sistema imunológico etc. (ZUBEN; ATTUX, 2008).

De acordo com Zuben e Attux (2008) esse tipo de comportamento social inspira pesquisadores a desenvolver diversas ferramentas computacionais para a resolução de problemas e estratégias de coordenação e controle de robôs. Assim surgiu o termo *Swarm Intelligence* (SI) no fim da década de 1980, proposto por Beni e Wang para se referir a sistemas robóticos formados por uma coleção de agentes simples que interagem em um ambiente seguindo regras locais (ROSENDO, 2010).

De acordo com White e Pagurek (1998) SI é uma propriedade de sistemas de agentes não inteligentes, com capacidades individuais limitadas, que exibem um comportamento coletivo inteligente. Para Bonabeau, Dorigo e Theraulaz (1999) *Swarm Intelligence* inclui qualquer tentativa de projetar algoritmos ou dispositivos distribuídos de solução de problemas inspirados no comportamento coletivo de insetos sociais e outras sociedades animais.

De acordo com Rosendo (2010), *Swarm Intelligence* é uma técnica de Inteligência Computacional que estuda o comportamento coletivo de agentes descentralizados. E com base nestes sistemas naturais de comportamento emergente, com o passar do tempo diversos sistemas artificiais de otimização têm sido desenvolvidos e aprimorados. Algumas características da SI, segundo Zuben e Attux (2008) são:

- proximidade: os agentes devem ser capazes de interagir uns com os outros;
- qualidade: os agentes devem ser capazes de avaliar suas ações;
- diversidade: o sistema deve reagir a situações inesperadas;
- estabilidade: nem todas as variações de ambiente devem afetar os agentes;
- adaptabilidade: capacidade de se adequar as variações ambientais.

A próxima seção trata do algoritmo *Particle Swarm Optimization* (PSO), descrevendo suas principais características e o seu funcionamento.

2.1.1. Particle Swarm Optimization

O *Particle Swarm Optimization* (Otimização por Enxame de Partículas) é um algoritmo de otimização estocástico, desenvolvido inicialmente pelo psicólogo social

James Kennedy e o engenheiro eletricista Russel Eberhart em 1995, e surgiu da observação do comportamento de pássaros na natureza (KENNEDY; EBERHART, 1995). Problemas de otimização estocásticos se apresentam quando o modelo não pode ser descrito completamente porque depende de variáveis desconhecidas no momento de sua formulação (NOCEDAL; WRIGHT, 1999).

Segundo Kennedy e Eberhart (1995), inicialmente o algoritmo era uma simulação de um meio social simplificado e as partículas foram pensadas para ser pássaros a prova de colisão; a intenção era simular graficamente a imprevisível coreografia dos pássaros. Depois de testado e corrigido, foram feitos testes onde o algoritmo foi utilizado para treinamento de Redes Neurais e ainda foi comparado com Algoritmos Genéticos.

De acordo com Reyes-Sierra e Coello (2006), existem dois aspectos que contribuem para a popularização do PSO: o primeiro aspecto refere-se ao fato de que desde sua versão original, o PSO adota apenas um operador, a partícula, para criar novas solução e consequentemente sua implementação é simples. Aliado a isso existe uma grande quantidade de código fonte disponível em domínio público. O segundo aspecto é que o PSO foi desenvolvido para ser eficaz em diversas aplicações diferentes, sendo capaz de produzir bons resultados com baixo custo computacional.

De acordo com Kennedy e Eberhart (1995) o PSO tem raízes em duas principais metodologias de componentes, sendo sua ligação mais óbvia com a Vida Artificial (*A-life*), mas também tem relação com a computação evolutiva.

“Por ter um comportamento emergente, em uma nuvem de partículas não existe um controle central. Cada partícula atua e toma decisões com base em informações locais e globais, como nas demais técnicas de Vida Artificial” (VESTERSTRØM; RIGET, 2002 apud ROSENDO, 2010, p. 17). Além disso, de acordo com Kennedy e Eberhart (1995), o PSO segue as características da SI, citadas na seção 2.1.

De acordo com Silva (2008) o PSO possui várias similaridades com as técnicas da computação evolutiva, tais como os algoritmos genéticos (AG). Ambos são inspirados pela natureza e baseiam-se no fato de que a experiência de animais em um grupo contribui para a experiência de todo o grupo. Assim como no AG, o PSO é uma forma de otimização com base em população, onde cada indivíduo é uma partícula, e esta é uma possível solução do problema em questão.

Entretanto, segundo Silva (2008) e Rosendo (2010), o comportamento do PSO e de outros métodos evolutivos diferem pelo fato de que o PSO simula a interação social mantendo todos os indivíduos até o final de suas iterações, enquanto na computação evolutiva utiliza-se o conceito de cruzamento, onde as partículas competem entre si para perpetuar suas características às próximas iterações, simulando a teoria da evolução de Charles Darwin.

O PSO é baseado no comportamento social de uma população (enxame) de indivíduos (partículas) e cada partícula age como se fosse um pássaro do bando em busca de alimento ou de seu ninho. De acordo com Augustus (2009), o PSO imita este cenário e utiliza-o para resolver problemas de otimização. Cada possível solução corresponde a um pássaro (partícula) e todas as partículas possuem resultado (*fitness*), que é verificado com a utilização de uma função objetivo, e possuem velocidade, que serve para direcionar seu voo. As partículas voam pelo espaço do problema seguindo aquelas que até o momento obtiveram as melhores soluções, como pode ser visto na Figura 1.

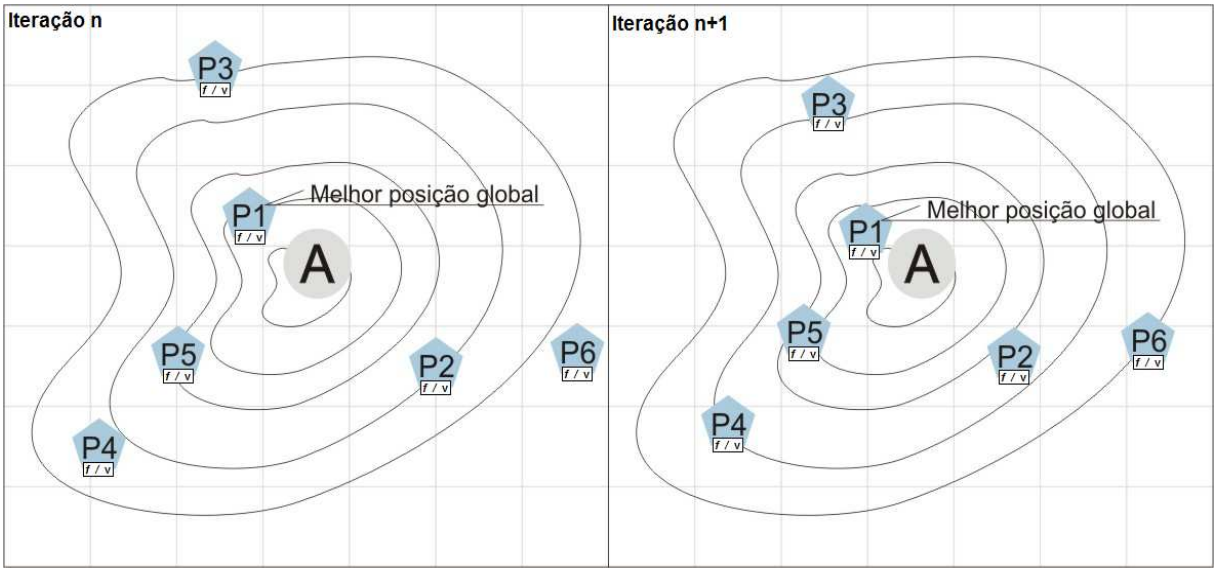

Figura 1 - Partículas no espaço do problema

Como mostrado na Figura 1, cada partícula é distribuída aleatoriamente no espaço de busca e cada uma possui seu valor de *fitness (f)* e velocidade (v). A cada iteração elas voam na direção das partículas que possuem no momento as melhores posições em relação ao alimento (A). De acordo com Silva (2008), cada partícula mantém-se informada sobre sua própria coordenada no espaço do problema e essa posição é associada a melhor solução encontrada.

De acordo com Rosendo (2010), as melhores posições que as partículas procuram seguir classificam-se em três grupos distintos: *Pbest* (*personal best*), que é a melhor posição encontrada pela própria partícula até o momento; *Lbest* (*local best*), que é a melhor posição encontrada entre as vizinhas de uma partícula até o momento; e *Gbest* (*global best*), que é a melhor posição encontrada em todo o enxame até o momento. Vizinhanças são adotadas quando o problema requer uma abordagem subdividida em áreas (AUGUSTUS, 2009).

Assim, cada partícula considera dois tipos de informação no processo de decisão: a primeira é a sua própria experiência e a segunda é baseada na experiência de outras partículas (SILVA, 2008). Ou seja, a partícula já tomou suas decisões no decorrer das iterações, sabendo em quais posições obteve um resultado mais promissor; ela sabe também as posições mais promissoras entre as outras partículas do grupo. O conceito de alteração da posição de uma partícula pode ser visto na Figura 2.

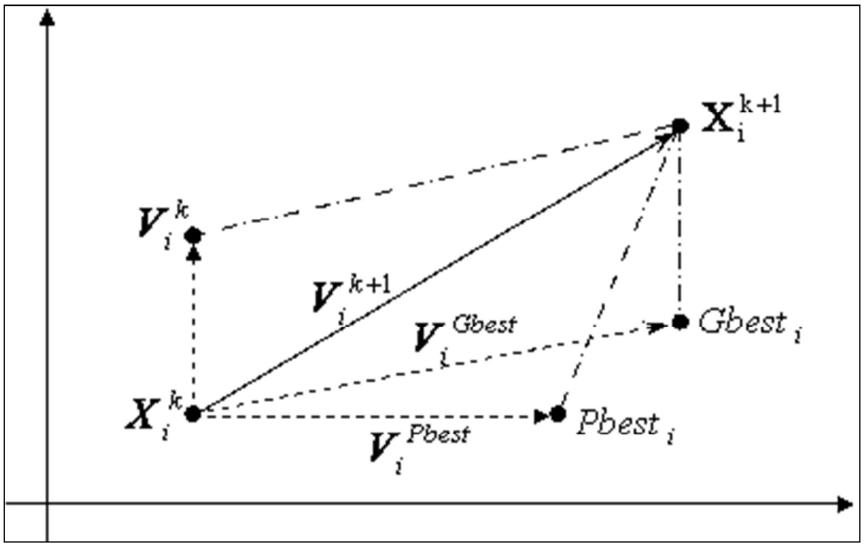

Figura 2 - Conceito de alteração do ponto de busca (ALLAOUA et.al., 2009. p. 10)

A Figura 2 mostra, conceitualmente, como é realizada a alteração do local em que a partícula está no espaço de busca: X^k é a posição atual da partícula e V^k é a sua velocidade atual, *Pbest* é a melhor posição em que essa partícula já esteve em algum momento e *Gbest* é a melhor posição entre todo o enxame, V^{Pbest} é a velocidade baseada em *Pbest* e V^{Gbest} é a velocidade baseada em *Gbest*. Com base nisso, é calculada uma nova velocidade que é atribuída à velocidade modificada V^{k+1} e a partícula passa de sua posição atual para a posição modificada X^{k+1}.

1. O Algoritmo

O procedimento de implementação do algoritmo *Particle Swarm Optimization* é dividido em algumas etapas, como pode ser visto na Figura 3. Esse procedimento é explicado com base em Silva (2008).

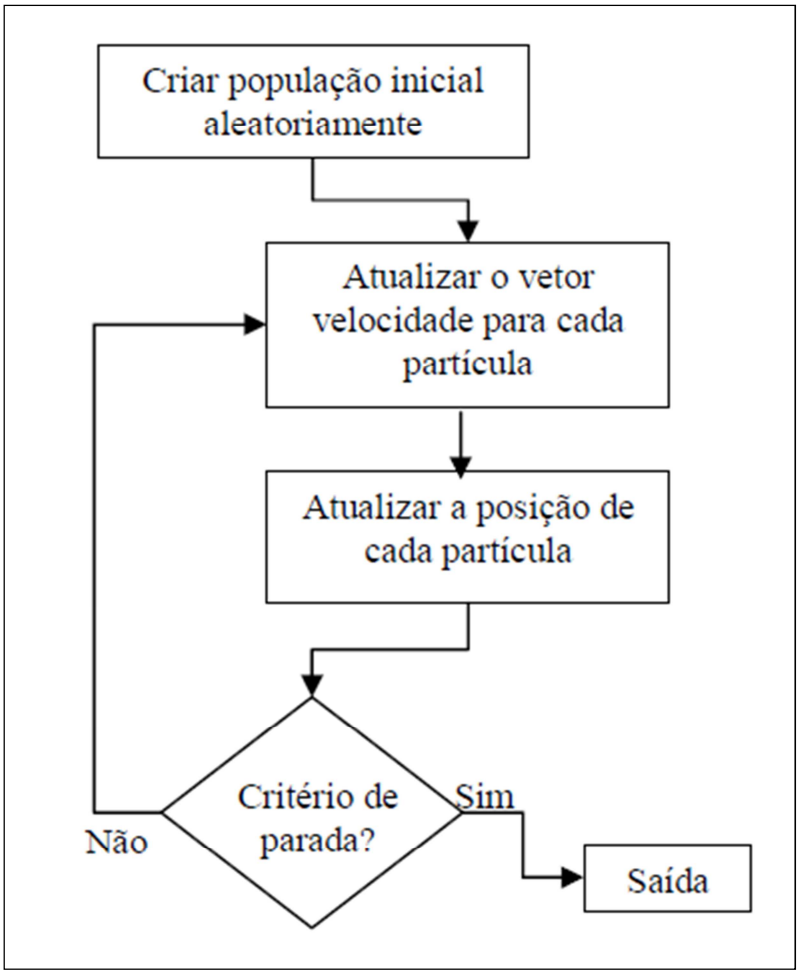

Figura 3 - Fluxograma do PSO (SILVA, 2008, p.47)

A Figura 3 apresenta um fluxograma resumido do processo de implementação do algoritmo PSO. Como pode ser visto a população de partículas é gerada aleatoriamente, depois é feita a atualização das velocidades e com isso as posições das partículas são atualizadas. Após a atualização do posicionamento das partículas o critério de parada é verificado: se a condição de parada for satisfeita é apresentada a saída, senão, o processo volta para a etapa de atualização das velocidades. A seguir são apresentadas as etapas que devem ser seguidas para a implementação do código:

Etapa 1 – Iniciar uma população de partículas com posições e velocidades geradas aleatoriamente no espaço n dimensional;
Etapa 2 – Avaliar o *fitness*;
Etapa 3 – Atualizar a velocidade de cada partícula de acordo com a Equação 1;
Etapa 4 – Atualizar a posição de cada partícula, de acordo com a Equação 2;
Etapa 5 – Mapear a posição de cada partícula dentro do espaço de solução e avaliar o valor da função *fitness*. E ao mesmo tempo, atualizar o *Pbest* e *Gbest* se necessário;
Etapa 6 – Voltar à Etapa 2 até que o critério de parada seja atingido. Geralmente esse critério é um *fitness* suficientemente bom ou atingiu um número máximo de iterações.

A Equação 1 e a Equação 2 são utilizadas para atualizar a velocidade e posição, respectivamente, de cada partícula *k* e dimensão *j.*

$$v_{k,j}(t+1) = w \cdot \mathrm{v}_{k,j}(t) + c_1 \cdot r_1 \cdot (pbest_{k,j} - x_{k,j}(t)) + c_2 \cdot r_2 \cdot (gbest_j - x_{k,j}(t)) \quad (1)$$

$$x_{k,j}(t+1) = x_{k,j}(t) + \mathrm{v}_{k,j}(t) \quad (2)$$

onde:

- t é a iteração atual;
- w é o fator de inércia da partícula;
- r_1 é variável aleatória para a parte cognitiva;
- r_2 é variável aleatória para a parte social;
- c_1 é o parâmetro de confiança para a parte cognitiva;
- c_2 é o parâmetro de confiança para a parte social;
- $v_{k,j}$ é a velocidade da partícula k na dimensão j;
- $pbest_{k,j}$ é o *Pbest* da partícula k na dimensão j;
- $gbest_{k,j}$ é o *Gbest* da partícula k na dimensão j.

O *Pbest*, melhor posição encontrada por uma partícula, pode ser entendido como a memória da partícula. Diversas restrições podem ser aplicadas para se definir qual a melhor posição, se adaptando a diferentes problemas. Isso não diminui a habilidade de busca e desempenho. Por exemplo, em problemas de otimização

linear as partículas lembram apenas as posições no espaço viável, desconsiderando as soluções inviáveis.

Em problemas de otimização de múltiplos objetivos a melhor posição é determinada pela eficiência de Pareto. Segundo Messac, Ismail-Yahaya e Mattson (2003, online), solução de Pareto é aquela para o qual qualquer melhoria em um objetivo só pode ocorrer se pelo menos um outro objetivo piorar.

E, em ambientes dinâmicos o Pbest é reiniciado para o valor atual caso o ambiente mude, seguindo uma técnica chamada de reinício de memória (*memory reset*). De acordo com Cui, Charles e Potok (2009), esta técnica consiste em reprogramar periodicamente toda a memória da partícula, substituindo o melhor valor de ajuste e vetor de localização com sua localização atual e valor de *fitness* para forçar a partícula a "esquecer" sua experiência anterior.

Os parâmetros de confiança c_1 e c_2, presentes na Equação 1, representam a ponderação da aceleração que empurra cada partícula em direção às posições *Pbest* ou *Gbest*, porque indicam o quanto uma partícula confia em si mesma (c_1 – parte cognitiva); e no enxame (c_2 – parte social). O termo “cognitivo” representa o quanto os indivíduos tendem a seguir seu próprio comportamento passado que obteve sucesso, enquanto o termo “social” representa a tendência de seguir o sucesso de outros indivíduos.

Em geral c_1 e c_2 possuem valores iguais, e ambos são essenciais para o sucesso da busca. Dessa forma os valores de confiança de uma partícula em si mesma e no grupo iniciam sem tender mais para um lado ou outro, a confiança da partícula será alterada a partir das variáveis aleatórias r_1 e r_2.

Na Equação 1 r_1 e r_2 são variáveis aleatórias que recebem um valor entre 0 e 1 e estão associadas aos parâmetros de confiança c_1 e c_2., onde r_1 é a variável aleatória para a parte cognitiva e r_2 é a variável aleatória para a parte social.

Segundo Siciliano (2007), além da velocidade, existem três fatores que influenciam a movimentação de uma partícula: inércia; memória; e cooperação. Na Equação 1 são representados respectivamente por w, c_1 e c_2. Segundo Souza (2006) esses coeficientes desempenham papéis social/cognitivos, que são respectivamente:

- o quanto a partícula confia em si mesma;
- o quanto a partícula confia em sua memória;
- o quanto a partícula confia em seus vizinhos;

O fator de inércia é usado para controlar a exploração e utilização. Um alto valor faz com que a partícula siga um comportamento mais global, enquanto um valor mais baixo faz com a que partícula tenha um comportamento mais local. Segundo Rosendo (2010), um alto valor para *w* faz com que a partícula tenha confiança em si mesma, fazendo com que ela busque seguir o seu próprio caminho, ao invés de seguir as melhores posições já encontradas por ela e pelo enxame, o que poderia. Um maior valor para *w* pode prevenir que as partículas de fiquem “presas” em um ponto ótimo, e um valor menor *w* incita às partículas a utilizarem a mesma área de busca.

A Figura 4 mostra como funciona a influência do fator de inércia, de memória e de cooperação na movimentação da partícula.

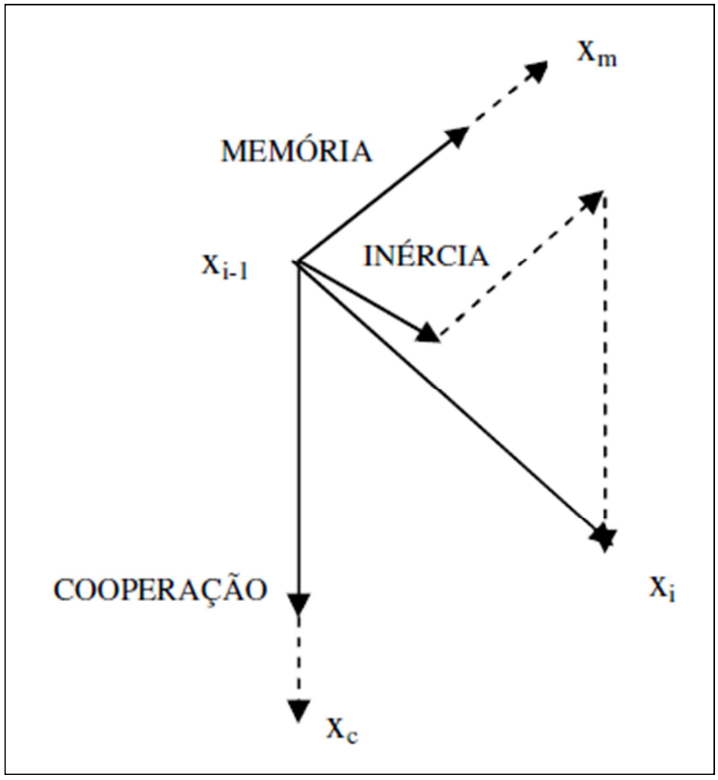

Figura 4 – Coeficientes de inércia, memória e cooperação (BORGES, 2006 apud SICILIANO, 2007, p. 2)

A Figura 4 é uma representação da influência dos coeficientes de inércia, memória e cooperação para a movimentação. O coeficiente de memória, representado na Equação 1 por c_1 em conjunto com a variável r_1 representa a força que leva uma partícula a seguir sua melhor posição (*lbest*). O coeficiente de cooperação, representado na Equação 1 por c_2 em conjunto com a variável r_2 representa a força que leva uma partícula a seguir a melhor posição do enxame

(*gbest*). E o coeficiente de inércia (*w*) é a força que leva uma partícula a seguir o caminho que já estava seguindo, que pode ser entendido como sendo sua velocidade V_{i-1}.

A partir dos cálculos realizados, levando em conta esses três coeficientes, é determinada a velocidade da atual da partícula, ou seja, sua direção, e assim, ela pode ir do ponto X_{i-1} para o ponto X_i.

2. Topologias de vizinhança

Duas classes de vizinhança podem ser identificadas: a vizinhança "física" e a "social" (SOUZA, 2006) ou topologia dinâmica e topologia estática (POLI; KENNEDY; BLACKWELL, 2007):

- vizinhança "física" (geográfica) ou dinâmica: leva em conta a distâncias entre as partículas, com base em um cálculo da distância euclidiana. Essas distâncias são computadas e a cada passo as arestas mais próximas são definidas como vizinhas.
- vizinhança "social" ou estática: leva em conta relacionamentos pré-definidos, não havendo a necessidade de cálculo de distância e atualização do grupo de vizinhos. A vizinhança de uma partícula é definida através de uma lista.

De acordo com Poli, Kennedy e Blackwell (2007), a topologia dinâmica, além de ser computacionalmente intensiva, é um tipo de estrutura de comunicação que contém propriedades de convergência indesejáveis. Por isso, este tipo de vizinhança euclidiana foi logo abandonado.

Com base em Reyes-Sierra e Coello (2006) serão apresentadas algumas das principais topologias de vizinhança social (estática):

- grafo vazio: Nesta topologia as partículas são isoladas. Cada partícula é conectada apenas com ela mesma, assim a comparação de sua posição atual é feita apenas com a melhor posição até o momento (*Pbest*);
- melhor local: Nesta topologia cada partícula é ligada a k partículas. As partículas são influenciadas por sua melhor posição (*Pbest*) e também pela melhor posição da vizinhança (*Lbest*). Quando k=2, esta estrutura equivale a uma topologia de anel, onde as partículas são influenciadas por seus dois vizinhos imediatos, esta estrutura pode ser vista na Figura 5. Esta topologia utiliza poucas operações para

calcular a movimentação das partículas e esse movimento é influenciado por um número pequeno de partículas;

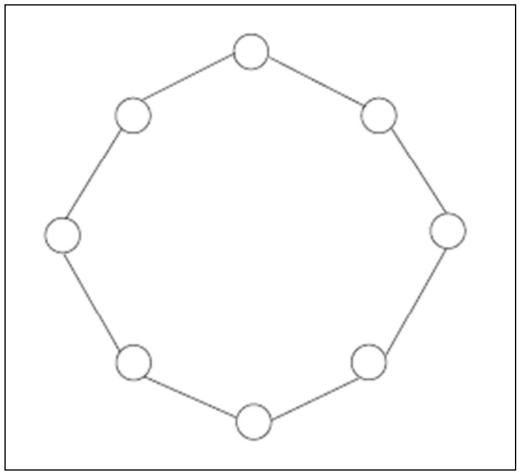

Figura 5 - Topologia em forma de anel (REYES-SIERRA; COELLO, 2006, p.290)

- estrela: Nesta topologia todas as partículas são conectadas a apenas uma partícula que é chamada de partícula focal. A partícula focal compara o desempenho das demais partículas e com base nisso faz seu movimento. As outras partículas seguem o movimento da focal, como pode ser visto na Figura 6. Essa topologia necessita de poucas operações para fazer o movimento influenciado pela melhor posição, mas a propagação dessa posição depende da partícula focal;

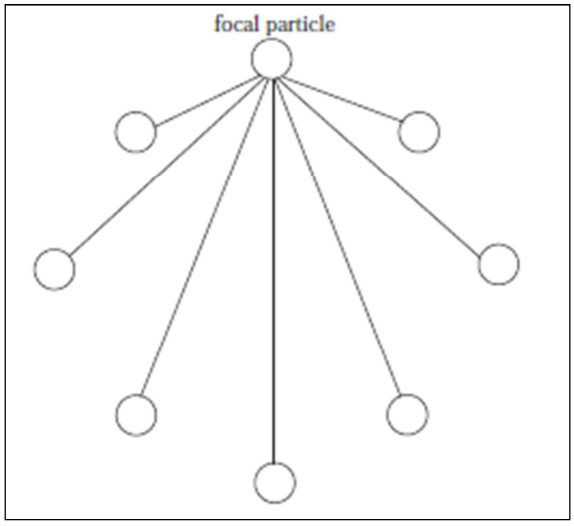

Figura 6 - Topologia em forma de estrela (REYES-SIERRA; COELLO, 2006, p.290)

- árvore: Nesta topologia as partículas são dispostas em formato de árvore e cada nó da árvore possui apenas uma partícula, como pode ser visto na Figura 7. Cada partícula é influenciada pela sua melhor posição (*Pbest*) e pela melhor

posição do nó imediatamente acima (nó pai - *parent*). Quando uma partícula encontra uma solução melhor que a de seu nó pai ocorre uma troca de posição entre as partículas. Dessa maneira, esta topologia acaba oferecendo uma vizinhança dinâmica. Essa estrutura é chamada de topologia hierárquica.

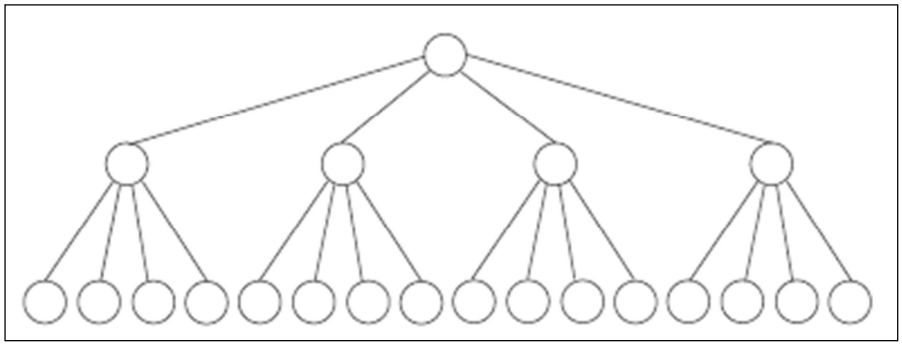

Figura 7 - Topologia em forma de árvore (REYES-SIERRA; COELLO, 2006, p.290)

- grafo totalmente conectado: Esta topologia é o oposto do grafo vazio, nela todas as partículas são ligadas umas as outras, como pode ser visto na Figura 8. Cada partícula é influenciada pela sua melhor posição (*Pbest*) e também pela melhor posição de uma partícula considerando todo o enxame (*Gbest*).

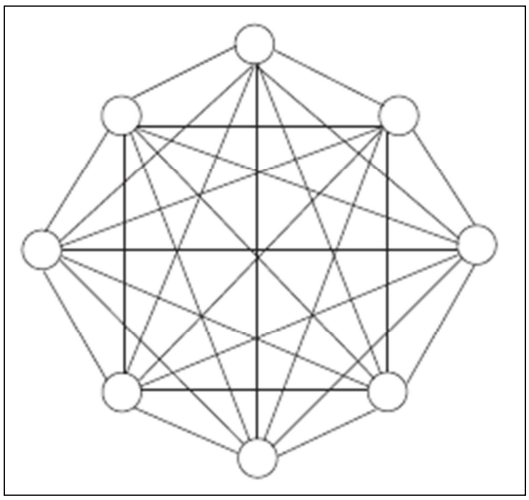

Figura 8 – Topologia em forma de Grafo totalmente conectado (REYES-SIERRA; COELLO, 2006, p.290)

As Figuras 5, 6, 7 e 8 foram mostradas para exemplificar os diferentes tipos de topologia de vizinhança e em todas as figuras cada nó representa uma partícula e cada aresta representa um relacionamento de vizinhança entre duas partículas.

De acordo com Poli, Kennedy e Blackwell (2007) a topologia *Gbest* (Melhor Global) pode ser conceituada como um grafo totalmente conectado, mas, na prática,

o programa necessita apenas manter o controle da melhor função resultado encontrada e o índice da partícula que a encontrou.

3. PSO: Variantes e especializações

Muitos ajustes e adaptações foram feitos para o algoritmo básico na última década. Alguns têm resultado em desempenho geral melhorado, e alguns têm um melhor desempenho em relação a determinados tipos de problemas (POLI; KENNEDY; BLACKWELL, 2007). A seguir serão brevemente descritas algumas das correntes do *Particle Swarm Optimization*:

- **PSO Binário**

Neste modelo cada individuo da população só tem conhecimento da decisão binária que deve tomar: sim/não, verdadeiro/falso etc. (Luna, 2004). Kennedy e Eberhart (1997), descreveram uma reformulação do algoritmo PSO original para operar com variáveis binárias discretas. Neste caso os operadores usam *bit-strings* em vez de números reais (POLI; KENNEDY; BLACKWELL, 2007).

De acordo com Kennedy e Eberhart (1997), na versão binária as particualas não são encaradas como possíveis soluções, mas sim probabilidades e as trajetórias são mudanças na probabilidade de que uma coordenada assumirá o valor zero ou um.

Trabalhos relacionados ao PSO Binário podem ser vistos em: Kennedy e Spears (1998), Tasgetiren, Suganthan e Pan (2007), Prata (2009) e Sudholt e Witt (2010).

- **problemas dinâmicos**

De acordo com Cui, Charles e Potok (2009), o conhecimento da partícula em um ambiente dinâmico fica desatualizado, então para parar a partícula com conhecimento desatualizado é necessário reprogramar periodicamente a memória da partícula (*Memory Reset*).

Problemas dinâmicos são desafiadores para PSO. Estes são tipicamente modelados pelas funções *fitness* que mudam ao longo do tempo, tornando obsoleta a memória das partículas (HU; EBERHART, 2001 apud POLI; KENNEDY; BLACKWELL, 2007).

Segundo Cui, Charles e Potok (2009), a grande desvantagem deste mecanismo é a dificuldade de determinar a frequência do *reset,* pois não se tem conhecimento prévio do ambiente. A essência do algoritmo PSO está na aprendizagem de cada partícula, na sua experiência de busca em seu passado e dos vizinhos, por isso essa técnica reduz a eficiência de busca do enxame. Redefinir frequentemente a memória das partículas pode causar incapacidade de convergência, então é importante que após cada *reset*, o algoritmo tenha um tempo extra para reavaliar cada partícula e seu calor atual.

- **PSOs Híbridos e adaptativos**

Segundo Poli, Kennedy e Blackwell (2007), vários investigadores tem tentado adaptar parâmetros do PSO, em resposta a informação a partir do ambiente. Técnicas da computação evolucionária e outros métodos, como Algoritmos Genéticos e algoritmo *Hill-Climb* foram adaptados pelos pesquisadores do PSO.

O primeiro PSO intencionalmente híbrido foi proposto por Angeline (1998), onde ela aplicou o processo de seleção ás partículas. Partículas “boas” eram reproduzidas e partículas “ruins” eram eliminadas.

Outros trabalho híbridos ou adaptados de algoritmos evolucionários e outras técnicas com o PSO são encontrados em: Loøvbjerg, Rasmussen e Krink (2001), Miranda e Fonseca (2002), Krink and Loøvbjerg (2002) e Poli e Stephens (2004).

- **PSOs com controle de diversidade**

Alguns pesquisadores tem notado a tendência do enxame de convergir prematuramente na ótima local. Várias abordagem têm sido implementadas a fim de corrigir o declínio da diversidade conforme o enxame se concentra em um único ótimo (POLI; KENNEDY; BLACKWELL, 2007).

Loøvbjerg (2002), utiliza a técnica *self-organized criticality* (SOC) para ajudar o PSO a obter mais diversidade, que faz com que o algoritmo se torne menos vulnerável a ótima local. Outros pesquisadores como Blackwell e Bentley (2002), Krink, Vesterstrøm e Riget (2002) têm tentado diversificar o enxame de partículas, evitando aglomeração das partículas em uma região do espaço de busca. Xie, Zhang e Yang (2002) descreve a entropia negativa, que serve para desencorajar a convergência prematura.

- **PSO *Bare Bones***

De acordo com Mello (2010) o *Bare Bones Particle Swarm*, que foi proposto por Kennedy em 2003 é um tipo de trabalho que propõe supressão do uso do termo velocidade, nesse caso a nova posição da partícula é definida através de distribuição gaussiana N centrada na média entre as influências pessoal e social, sendo o desvio definido pela distância entre os pontos.

Distribuição Gaussiana é uma das mais importantes distribuições da estatística e é inteiramente descrita por seus parâmetros de média e desvio padrão, ou seja, onde através destes consegue-se determinar qualquer probabilidade em uma distribuição (LEITÃO, 2010).

Segundo Hsieh e Lee (2010) simplifica o enxame de partículas, removendo a regra de velocidade, mas o desempenho não parece tão bom como o algoritmo canônico, eles propõem algumas melhorias para o *Bare bones.*

Trabalhos relacionados ao PSO *Bare Bones* podem ser vistos em: Omran e Al-Sharhan (2007), Omran, Engelbrecht e Salman (2008) e al-Rifaie e Blackwell (2012).

2.2. Problema de *Timetabling*

Segundo Wren (1996 apud Burke e Petrovic, 2002), *Timetabling* é um tipo especial de *Scheduling* (Agendamento) e pode ser definido como a alocação, sujeita a restrições, de determinados recursos para objetos a serem colocados num espaço de tempo, de tal modo a satisfazer o máximo possível um conjunto de objetivos desejáveis.

O problema de *Timetabling* pode ser definido através de quatro parâmetros: T (*times*), conjunto finito de horários; R (*resources*), conjunto finito de recursos; M (*meetings*), conjunto de eventos; e C (*constraints*), conjunto de restrições. O problema consiste em associar os horários e os recursos aos eventos, tentando satisfazer, da melhor forma possível, as restrições (BURKE et.al., 2003 apud VIEIRA; MACEDO, 2011).

Segundo Burke et.al. (2006), as restrições de *Timetabling* que devem ser atendidas muitas vezes tornam o problema difícil de resolver em circunstâncias do mundo real. Estas restrições geralmente são divididas em dois tipos:

- restrições *hard*, que devem ser satisfeitas sob qualquer circunstância;
- restrições *soft*, que necessitam ser satisfeitas tanto quanto possível.

De acordo com Schaerf (1995), em alguns casos o problema de *Timetabling* consiste em encontrar qualquer calendário que satisfaça todas as restrições. Nestes casos, o problema é formulado com um problema de busca. Em outros casos, o problema é formulado como um problema de otimização, ou seja, é necessário um calendário que satisfaça todas as restrições *hard* e minimize (ou maximize) uma dada função objetivo que incorpora as restrições *soft*.

Ainda de acordo com Schaerf (1995), tanto no caso de busca como no caso de otimização, é feita a definição do problema base que, no caso da busca, é decidir se existe uma solução; e no caso da otimização é decidir se existe uma solução com um dado valor de função objetivo. Nos casos de busca são utilizados algoritmos de busca e nos casos de otimização são utilizadas heurísticas e algoritmos de otimização.

Segundo Chan (1994), problemas de *Timetabling* geralmente têm espaços de solução muito grandes e altamente restritos. Eles são típicos problemas de otimização combinatória que computacionalmente são NP-completos, isto é, não há como saber o tempo polinominal que o algoritmo pode levar para encontrar uma solução.

De acordo com Burke et.al. (2006), há mais de 40 anos o *Timetabling* tem atraído a atenção da Pesquisa Operacional e de comunidades de Inteligência Artificial. Sendo que os estudos iniciais sobre estratégias baseadas em computador para *Examination Timetabling* datam da década de 1960 (BURKE et.al. 2010).

O grande interesse científico em problemas de *Timetabling* resultou na criação da série de conferências PATAT (*Practice and Theory of Automated Timetabling*), a partir de 1995, com novas edições a cada dois anos. Esse interesse resultou ainda no surgimento da EURO (*Association of European Operational Research Societies*) e da WATT (*Working Group on Automated Timetabling*). Em 2002, com o apoia da PATAT foi criada a Competição internacional de *Timetabling* (ITC) (AROGUNDADE; AKINWALE; AWEDA, 2010).

Existe um grande número de variantes do problema de *Timetabling*. Essas variantes diferem entre si de acordo com as necessidades específicas de cada área de aplicação do problema. Estas áreas podem ser: esportiva (por exemplo, em Easton, Nemhauser e Trick, 2001); hospitalar (por exemplo, em Burke, Kendall e

Soubeiga, 2003, e Burke et.al., 2004); transportes (por exemplo, em Caprara et.al., 2001); horários de empregados (por exemplo, em Meisels; Schaerf, 2002, e Detienne et.al., 2007); educacional, que de acordo com Gunawan, Ng e Poh (2006), é uma das principais áreas de aplicação de *TimetablingI*; entre outras.

Segundo Schaerf (1995), o *Timetabling* Educacional pode ser dividido em três classes principais, que diferem de acordo com as restrições de cada uma e também com base no tipo de instituição envolvida. As três classes são:

- *Timetabling* Escolar (*School Timetabling*): agendamento semanal de todas as classes de uma escola, evitando que professores e alunos sejam reunidos em duas classes ao mesmo tempo e vice-versa;
- *Timetabling* Universitário (*Course* [*University*] *Timetabling*): agendamento seminal de todas as aulas de um curso universitário, minimizando sobreposições de aulas com alunos em comum e outros recursos.
- *Timetabling* de Exames (*Examination Timetabling*): Agendamento de um conjunto de exames para cursos universitários, distribuindo os exames tanto quanto possível, evitando sua sobreposição para os mesmos alunos.

De acordo com Vieira e Macedo (2011), a diferença entre essas três formas de classificação está na quantidade do conjunto finito de alguns parâmetros. O *Timetabling* Escolar possui quantidade menor de recursos porque uma turma já possui sala especificada e alunos normalmente predeterminados. O que torna sua complexidade reduzida em relação ao *Timetabling* Universitário, que possui um conjunto grande de restrições, pois disciplinas e turmas não possuem sala fixa e alunos de uma turma não são predeterminados, entre outras restrições. Já no *Timetabling* de Exames as restrições são mínimas e dizem respeito as avaliações de cada disciplina, em que dias e horários devem ocorrer ou não ocorrer, entre outras.

No entanto Schaerf (1995), diz que essa classificação não é rígida, no sentido de que existem alguns problemas específicos que podem cair entre duas classes, e não pode ser facilmente colocado dentro da classificação acima. Por exemplo, os problemas de *Timetabling* de uma escola específica que dá grande liberdade aos estudantes sobre um conjunto de cursos pode ser semelhante ao *Timetabling* Universítário. Na Figura 9 é apresentado um esquema de classificação do problema de *Timetabling* Educacional.

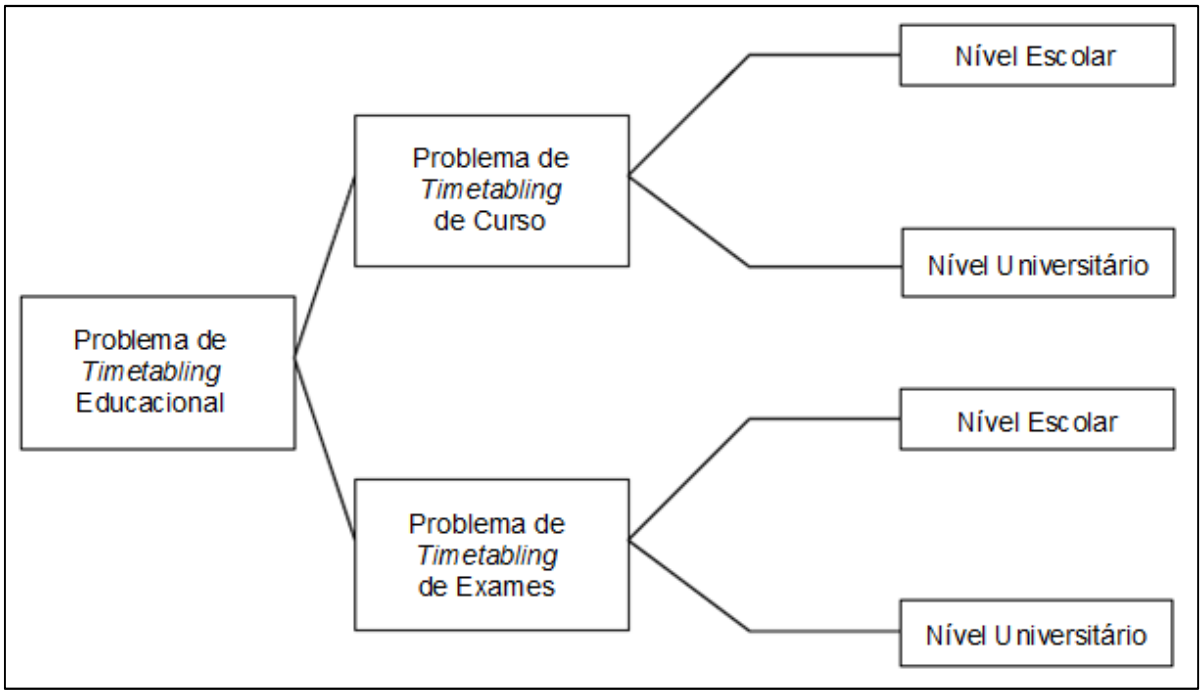

Figura 9 - Classificação do problema de *Timetabling* Educacional (Adaptada de Gunawan, Ng e Poh, 2006)

A Figura 9 mostra que o problema de *Timetabling* Educacional pode ser subdividido em *Timetabling* de Curso e de Exame e ambos são divididos nos níveis Escolar e Universitário.

De acordo com Gunawan, Ng e Poh (2006), em 1998 Carter e Laporte classificaram o problema de Timetabling de Curso em cinco subproblemas: atribuição de professor; horários sala/professor; agendamento de cursos; agendamento de alunos; e atribuição de salas.

Segundo Burke et.al. (2006), no inicio das pesquisas sobre *Timetabling* educacional foram investigadas abordagens de heurísticas de grafo e programação linear de inteiros. Porém essas técnicas são impraticáveis ou muito simples para resolver problemas complexos de *Timetabling* (BURKE et.al., 2006). Técnicas baseadas em restrições têm sido empregadas ao longo dos anos e, recentemente, foram investigadas técnicas de meta-heurística de busca, que foram bem sucedidas na resolução de uma variedade de problemas de *Timetabling*. Isso inclui técnicas de Busca Tabu, Recozimento Simulado e Algoritmos evolucionários.

Outras novas abordagens e metodologias têm sido estudadas, isso inclui raciocínio baseado em casos (*Case-Based Reasoning*), metodologia *fuzzy* e hiper-heurísticas (BURKE et.al., 2006).

A próxima seção trata sobre o problema de *Timetabling* Universitário, descrevendo a características dessa classe do problema de *Timetabling* e seus tipos de restrições.

2.2.1. *Timetabling* Universitário

Timetabling é um problema prático importante porque é frequentemente encontrado em instituições de ensino (HOSNY; FATIMA, 2011), e de acordo com Petrovic e Burke (2004) com certeza é uma das principais atividades administrativas na maioria das universidades (PETROVIC; BURKE, 2004).

Segundo Vieira e Macedo (2011), existem ferramentas comerciais que prometem fazer a geração automatizada de grades de horários, mas são pouco utilizadas, pois esse tipo de problema incorre em necessidades específicas de cada instituição de ensino ou curso, em detrimento de soluções genéricas. Devido a grande diversidade de regimes educacionais, as diferenças entre regiões e países e as características distintas de cada instituição de ensino, há um consenso entre a comunidade científica de que é difícil generalizar o problema de *Timetabling* (PAIM; GREIS, 2008).

De acordo com Burke et.al. (2003) o problema de *Timetabling* Universitário envolve o agendamento de aulas com um dado número de intervalos de tempo (período) e sua alocação em salas de aula disponíveis, geralmente em uma base semanal, enquanto satisfaz certas restrições.

Segundo Hosny e Fatima (2011), para este tipo de problema é necessário atribuir um número de eventos, como exames; ou cursos; ou alunos, para certo número de salas de aulas e um número de intervalos de tempo, enquanto adere a um conjunto pré-especificado de restrições.

1. Restrições

Para a construção de uma grade de horários, diversas regras devem ser seguidas, entre as quais algumas são tão importantes que nunca podem ser violadas (restrições *hard*) e outras que não são tão importantes e, geralmente, são obedecidas quando todas as restrições *hard* foram satisfeitas e ainda há espaço

para melhorar a qualidade da solução (restrições *soft*) (DASKALAKI; BIRBAS; HOUSOS, 2004).

Com base em Daskalaki, Birbas e Housos (2004), as restrições *hard* para *Timetabling* Universitário são regulados pelas seguintes regras:

- colisões não podem ser permitidas: em *Timetabling* uma colisão ocorre quando duas ou mais aulas estão agendadas no mesmo período de tempo, para um mesmo professor, para um mesmo grupo de alunos ou para a mesma sala de aula. Ocorre também quando dois ou mais professores são atribuídos para lecionar aulas diferentes para um mesmo grupo de alunos, ou quando duas ou mais salas são atribuídas a mesma aula, para o mesmo grupo de alunos.
- o Timetabling deve ser completo, e ele é completo quando todos as aulas previstas para cada grupo de alunos aparecem no calendário, com a quantidade certa de períodos de tempo para cada aula;
- o *Timetabling* deve acomodar pedidos de sessões de tempos letivos consecutivos, ou seja, dependendo da matéria e o número de períodos de aula designados por semana, um professor pode escolher dar a aula em sessões de período único ou multi-períodos. O *Timetabling* deve ser capaz de programar uma dada matéria em qualquer esquema que o professor responsável puder optar.

Ainda de acordo com Daskalaki, Birbas e Housos (2004), as restrições *soft* para *Timetabling* Universitário seguem as seguintes regras:

- preferências de intervalos de tempo para aula: cada professor pode expressar a sua opinião sobre seu período de aula preferido. Por exemplo, um professor pode preferir dar aula pela manhã e outro durante o período noturno, ou em um determinado dia da semana;
- os horários dos estudantes devem ser o mais compacto possível, no entanto permitir pausas;
- minimizar as trocas de sala de aula.

A próxima seção apresenta os materiais e a metodologia utilizados no desenvolvimento do trabalho.

3 MATERIAIS E MÉTODOS

Nesta seção são apresentados os materiais utilizados durante a realização do trabalho e a metodologia empregada para o desenvolvimento do sistema.

3.1. Materiais

Na etapa de modelagem do sistema, foi utilizado o software CorelDRAW X6, para desenhar o fluxograma de funcionamento do sistema. Na etapa de desenvolvimento, a linguagem utilizada foi o PHP; para a escrita do código foi utilizado o editor de texto e código fonte Notepad++ 6.5.1. O WAMPSERVER 2.4 foi utilizado para a execução do programa, pois a linguagem só roda em servidores. O banco de dados utilizado foi o Microsoft SQL Server 2012

O processo foi realizado no sistema operacional Windows 7 *Ultimate* – 64 bits, em um computador com 4 GB de memória RAM e processador Intel Core i5, 60 Hz.

3.2. Metodologia

A primeira parte da realização do trabalho consistiu em coletar e estudar material bibliográfico para criar o material de referência, essa coleta ocorreu em sua maioria em artigos científicos, monografias e dissertações. Além disso, como fonte de pesquisa secundária, foram considerados sites com conteúdo técnico sobre os conceitos estudados, assim foi alcançada uma maior compreensão do algoritmo PSO e do problema de *Timetabling*.

A etapa seguinte, de observação do problema, ocorreu no Centro Universitário Luterano de Palmas – CEULP/ULBRA, onde foi realizada uma reunião com a coordenadora dos cursos de Sistemas de Informação (SI), Ciência da Computação (CC) e Redes de Computadores (RC), visando conhecer melhor o processo criação da grade de horários do Departamento de Computação (DC) do CEULP/ULBRA. Através da reunião, pode-se saber o processo que a coordenadora realiza para gerar as grades horárias dos cursos de SI e CC, e conhecer um pouco mais sobre o funcionamento da Universidade quanto aos horários. As dúvidas que restaram foram sanadas com trocas de e-mails.

Após coletar informações na instituição foi iniciada a etapa de desenvolvimento do sistema, que ocorreu no segundo semestre do ano de 2013 e foi dividido em: planejamento e execução. No planejamento a primeira definição foi o banco de dados. Em seguida, foi realizada modelagem do sistema, onde ocorreu a definição do modelo de dados, a elaboração do fluxograma do sistema e a escolha da linguagem.

A fase de execução teve início com a criação do banco de dados do sistema. Depois teve início a programação. O código gerado foi dividido em dois arquivos: *class*.php, onde foi criada a classe slqsrv que contêm todas as funções executadas no sistemas e; *index*.php, que contêm a parte do código que gera a interface.

Para rodar a aplicação, como em todo programa em PHP, é necessário um servidor, sendo assim houve a necessidade de criar um servidor local; para isso foi utilizado o programa WAMPSERVER.

Na próxima seção são apresentados os resultados obtidos com o desenvolvimento do trabalho.

4 RESULTADOS E DISCUSSÃO

Nesta seção são apresentados os resultados obtidos no decorrer do desenvolvimento deste trabalho, começando pela descrição do problema, onde são relacionadas às necessidades da instituição e as regras utilizadas no desenvolvimento do protótipo. Após a descrição é apresentada uma visão geral de como ocorre o funcionamento do programa. Em seguida é apresentado o modelo relacional do banco de dados e, por fim, o detalhamento do programa, contendo a apresentação das telas geradas e a descrição de trechos mais importantes do código gerado.

4.1. Descrição do problema

O problema considerado para a elaboração do sistema é um *Timetabling* Universitário baseado na geração de grades horárias no Departamento de Computação do CEULP/ULBRA. O problema consiste em agendar eventos (aulas) através de recursos (Professores, disciplinas e período) em determinados horários (dias da semana e horário), respeitando uma série de restrições.

No problema considerado, atualmente, para gerar a grade a coordenadora dos cursos do DC primeiro define quais disciplinas serão ofertadas no semestre e que professor será o responsável por cada uma. Após isso, é feito um quadro onde são colocadas as matérias de cada período nos dias da semana e caso exista algum conflito, as disciplinas são mudadas até que se chegue a um resultado sem conflitos. Todo o processo é feito manualmente.

Devido ao fato de que a universidade disponibiliza grade aberta para que os alunos escolham quais matérias desejam fazer no semestre, não sendo obrigatória a escolha pela grade completa, a grade é gerada ao final de cada semestre, assim, quando os alunos forem fazer sua matrícula/rematrícula, as disciplinas que serão ofertadas já se encontram disponíveis e com os dias determinados. Por esse motivo, o aluno, que poderia ser um recurso, é irrelevante na construção do sistema.

Um fator que deve ser levado em consideração é o fato de que não há disponibilidade para que todas as disciplinas do curso sejam ofertadas todos os semestres, isso faz com que em algumas disciplinas, alunos de diferentes períodos façam parte da mesma turma. Ou seja, uma disciplina pode estar em mais de um

período no semestre. Isto foi considerado como uma das regras para o sistema. Ressalte-se que a mesma disciplina pode ser ofertada em turmas diferentes, podendo ser cada turma relacionada ao mesmo professor ou a diferentes professores.

A seguir são descritas as restrições *hard* e *soft* que foram consideradas na elaboração do sistema.

4.1.1. Restrições *Hard*

As restrições *hard* usadas como regras na construção do sistema são:

- Um professor não pode lecionar duas disciplinas no mesmo horário;
- Uma turma não pode estar associada a mais de uma aula por semana;
- Quando uma disciplina for oferecida para mais de um período, a disciplina deve estar na mesma turma, sendo a aula no mesmo dia e horário;
- Em um período duas disciplinas ministradas no mesmo dia não poderão estar no mesmo horário.

4.1.2. Restrições *Soft*

As restrições *soft* usadas como regras na construção do sistema são:

- Quando uma disciplina for ofertada duas vezes por professores diferentes, devem preferencialmente ser no mesmo dia;
- Aulas poderão acontecer no horário da tarde, se não houver possibilidade de encaixa-las nos demais horários.
- Determinados professores não podem dar aula em certos dias;
- Determinados professores gostariam de dar aula em certos dias;
- Determinadas disciplinas não podem ser ministradas em certos dias.

4.2. Visão geral do sistema

Como parte do planejamento foi elaborado o fluxograma de funcionamento do sistema, visando um melhor entendimento do todo, para uma melhor execução das partes. A próxima seção detalhará o fluxograma.

4.2.1. Fluxograma

A figura a seguir apresenta o fluxograma do sistema.

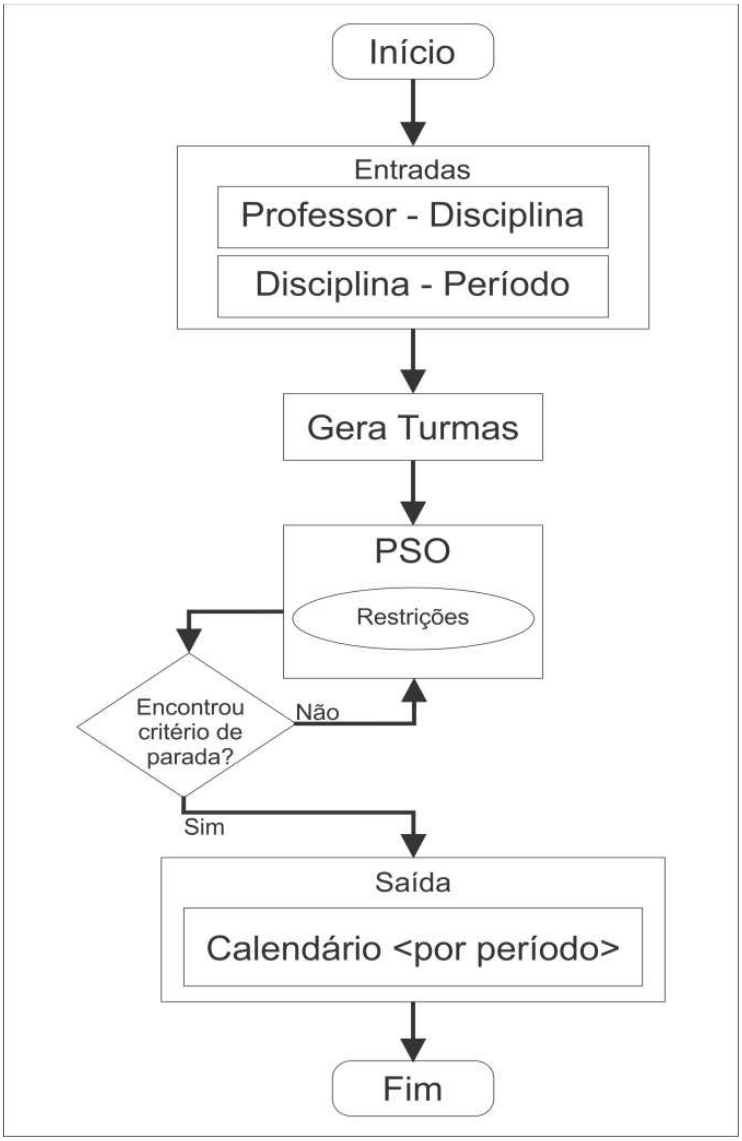

Figura 10 - Fluxograma de funcionamento do sistema

Como mostrado na Figura 10, como entrada o sistema recebe informações do usuário que deve relacionar primeiro professores e disciplinas, e depois, o relacionamento dessas disciplinas com períodos do curso. Todos os dados que o usuário relaciona são carregados da base de dados do programa. Após isso o programa gera as turmas. Após as turmas serem geradas, o usuário irá solicitar ao programa que a grade seja gerada. Neste momento o algoritmo PSO é acionado. A próxima figura mostra o fluxograma do algoritmo.

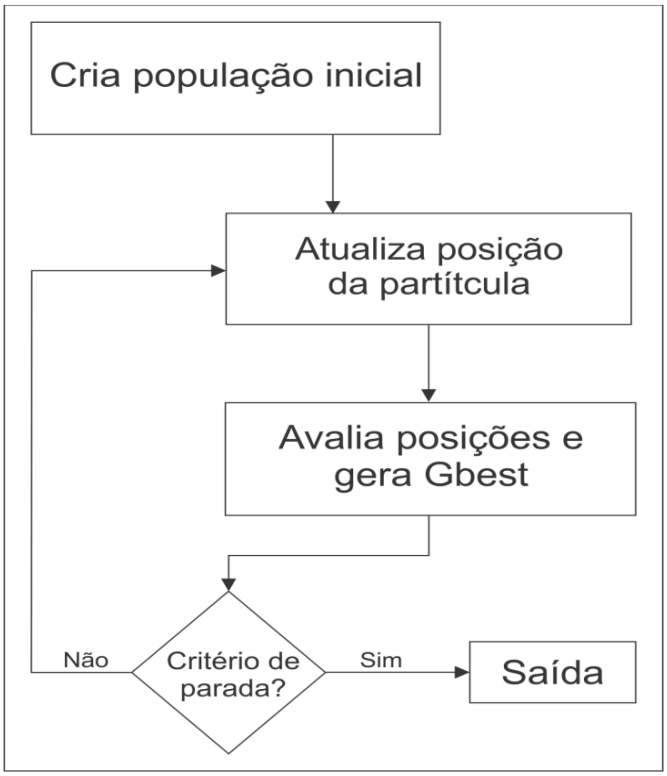

Figura 11 - Fluxograma do PSO

A Figura 11 apresenta o fluxograma de funcionamento do algoritmo PSO como é utilizado na geração da grade de horários. Primeiro as partículas – que figurativamente representam pássaros, sendo cada uma, uma possível solução para o problema – são geradas. As partículas possuem conhecimento da sua posição atual, sua melhor posição (*Pbest*) e da melhor posição de uma partícula entre todo o bando (*Gbest*).

Após a criação de cada partícula, sua posição é atualizada. Nesta atualização cada turma de cada período recebe um dia e horário para ser ministrada. Depois disso, as posições das partículas são avaliadas de acordo com a função *fitness*, que irá aplicar as regras *hard* e *soft*. A posição recebe uma pontuação (*Lbest*), que para cada regra infringida é feito um cálculo que tirará ponto do *Lbest.* Estando as posições avaliadas, e se for necessário, *Pbest* e *Gbest* são atualizados.

Após a avaliação o critério de parada é verificado, e se foi alcançado a saída é apresentada. Senão, volta para a atualização das posições e o ciclo se repete até que o critério seja alcançado. A seguir será apresentado o detalhamento do código e funcionamento do sistema.

4.3. Detalhamento do programa

O programa segue a mesma estrutura descrita na seção 4.2, e cada uma de suas partes será descrita detalhadamente através da apresentação da interface e explicação de trechos importantes do código.

As funções utilizadas para a execução do programa se encontram dentro da classe sqlsrv, que se encontra no arquivo *class*.php. O arquivo *index*.php contém o código que gera a interface do protótipo e nele são chamadas algumas funções da classe sqlsrv.

Para que o algoritmo possa gerar a grade, o usuário deverá fornecer as entradas ao sistema, primeiro relacionando as disciplinas aos professores e depois relacionando as disciplinas aos períodos. A figura a seguir mostra a parte da interface em que o usuário realiza a ligação de disciplinas e professores.

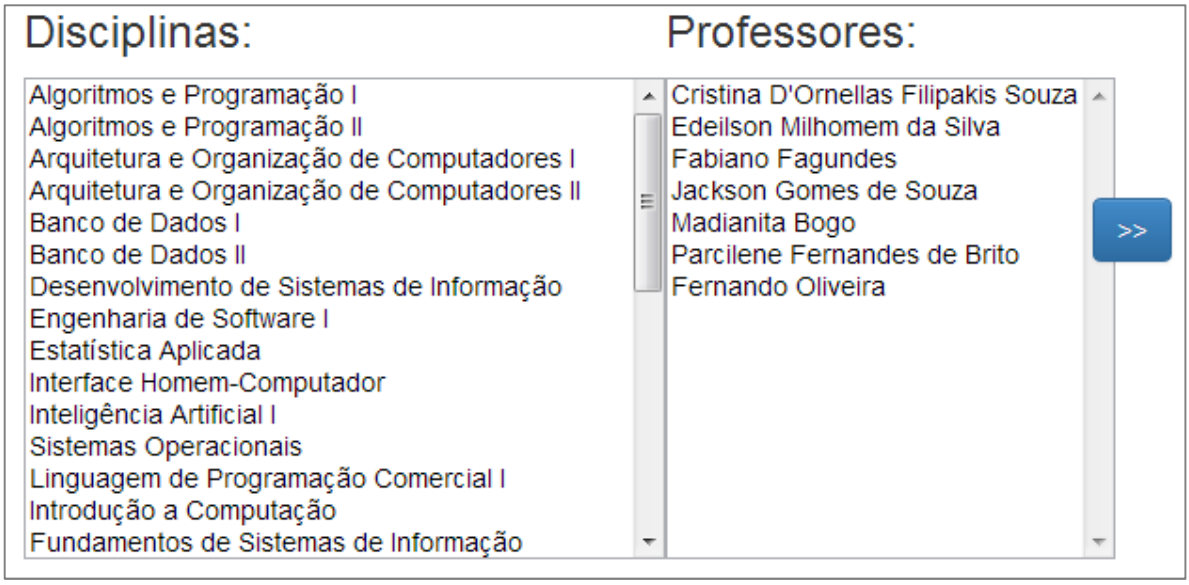

Figura 12 - Entrada 1 (Disciplinas - Professores)

A Figura 12 apresenta a área da primeira entrada do sistema, onde o usuário deve selecionar uma disciplina na lista de disciplinas à esquerda e depois selecionar o professor que irá lecionar tal disciplina no semestre, na lista de professores à direita, em seguida clicar no botão “>>” para criar uma turma. Os dados da turma serão a base para a criação da grade.

Após o usuário clicar no botão “>>” a turma é gerada. As figuras a seguir mostram o código que irá inserir os dados na tabela “turma” e a imagem da área de turmas na interface.

```php
if (!empty($_POST['add'])) {
    $professdor=$_POST['professor'];
    $disciplina=$_POST['disciplina'];
    if ($disciplina != "" && $professdor!=""){
        $sql="select * from turma where idprofessor='$professdor' and
        iddisciplina='$disciplina'";
        $db->query($sql);
        if ($db->linhas() == 0){
            $insert = "insert into turma values ('', '".$professdor."', '".
            $disciplina."')";
            $db->query($insert);
        }else{
            echo '<script>alert("Registro duplicado.");</script>';
        }
    }else{
        echo '<script>alert("Selecione um professor e uma
        disciplina.");</script>';
    }
}
```

Figura 13 - Inserindo dados na tabela “turma” (*index*)

A Figura 13 mostra o trecho do código do arquivo *index*.php que irá gerar as turmas no banco de dados. Para inserir os dados, um professor e uma disciplina devem estar selecionados, senão é retornada uma mensagem de erro. Estando a disciplina e o professor selecionados, estes têm suas chaves primárias buscadas em suas respectivas tabelas no banco de dados e inseridos na tabela “turma”. Esse trecho do código também faz uma verificação de dados duplicados. A figura a seguir mostra a interface que contém a lista de turmas geradas.

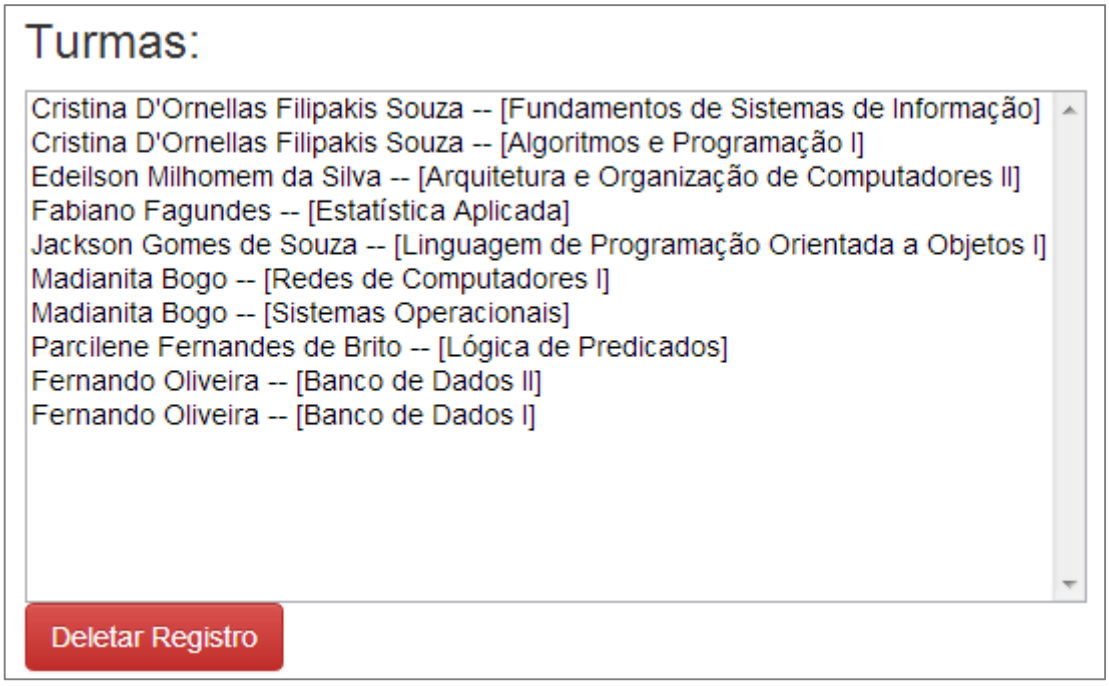

Figura 14 - Lista de turmas

A Figura 14 mostra a parte da interface que exibe as turmas geradas, mostrando o nome do professor e a disciplina que será ministrada por ele. Após a geração das turmas o usuário deve prosseguir fornecendo os dados de entrada, relacionando as disciplinas aos períodos dos quais farão parte. A próxima figura mostra a parte da interface em que o relacionamento de disciplinas e períodos é feito.

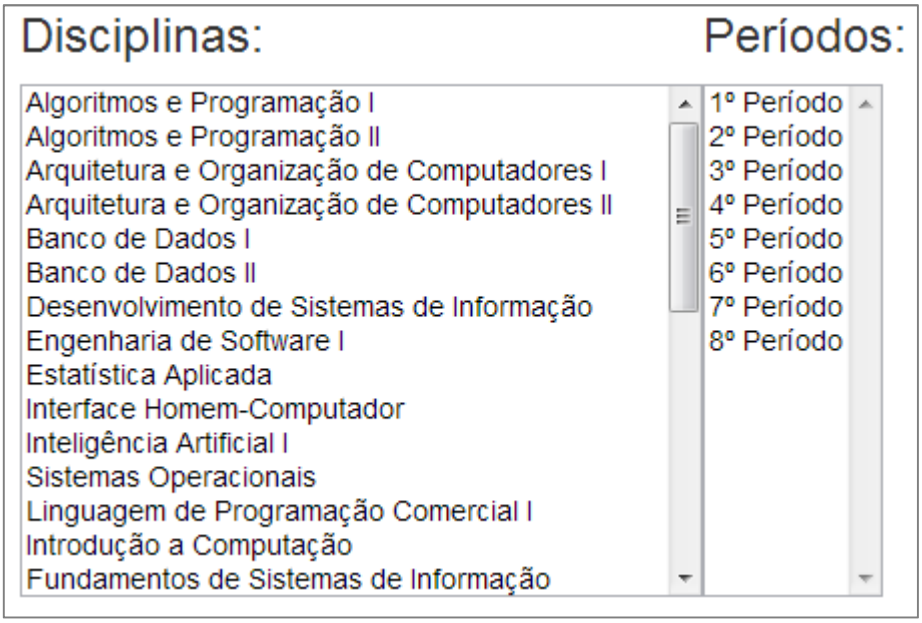

Figura 15 - Entrada 2 (Disciplinas - Períodos)

A Figura 15 apresenta a área da segunda entrada do sistema, onde o usuário deve selecionar uma disciplina na lista de disciplinas à esquerda e depois selecionar o período em que será ofertada a disciplina, na lista de períodos à direita, em seguida clicar no botão “>>” para relacionar as disciplinas aos períodos e preencher a tabela “disciplinaperiodo”.

O preenchimento da tabela “disciplina” período é importante para que a restrição *hard* que diz que “quando uma disciplina for oferecida para mais de um período, a disciplina deve estar na mesma turma, sendo a aula no mesmo dia e horário” seja respeitada.

Após o usuário clicar no botão “>>” a lista de disciplinas por período é gerada. As figuras a seguir mostram o código que irá inserir os dados na tabela “disciplinaperiodo” e a imagem da área de disciplinas por período na interface.

```php
if (!empty($_POST['add2'])) {
    $periodo=$_POST['periodo'];
    $disciplina=$_POST['disciplina2'];
    if ($disciplina != "" && $periodo!=""){
        $sql="select * from disciplinaperiodo where idperiodo='$periodo' and
        iddisciplina='$disciplina'";
        $db->query($sql);
        if ($db->linhas() == 0){
            $insert = "insert into disciplinaperiodo values ('', '".$periodo.
            "', '".$disciplina."')";
            $db->query($insert);
        }else{
            echo '<script>alert("Registro duplicado.");</script>';
        }
    }else{
        echo '<script>alert("Selecione um periodo e uma
        disciplina.");</script>';
    }
```

Figura 16 - Inserindo dados na tabela "disciplinaperiodo" (*index*)

A Figura 16 mostra o trecho do código do arquivo *index*.php que irá gerar as disciplinas por período no banco de dados. Para inserir os dados, uma disciplina e um período devem estar selecionados, senão é retornada uma mensagem de erro. Estando a disciplina e o período selecionados, estes têm suas chaves primárias buscadas em suas respectivas tabelas no banco de dados e inseridas na tabela "disciplinaperiodo". Esse trecho do código também faz uma verificação de dados duplicados. A figura a seguir mostra a interface que contem a lista de disciplinas por período geradas.

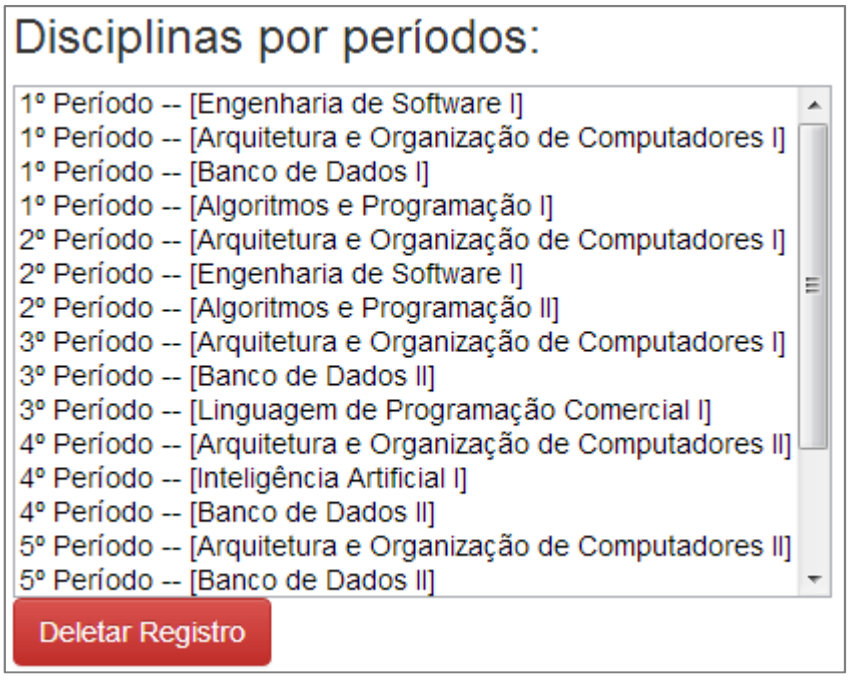

Figura 17 - Lista de disciplinas por período

A Figura 17 mostra a parte da interface que exibe as disciplinas por período, mostrando o período e a disciplina que será ofertada nele.

Depois que todos os parâmetros de entrada forem passados, o usuário deve clicar no botão “Gerar” da área da grade para invocar o algoritmo PSO. A próxima seção mostrará o funcionamento do algoritmo PSO e as funções que auxiliam seu funcionamento.

4.3.1. PSO

Nesta seção será apresentado o funcionamento do algoritmo PSO para gerar a grade de horários, bem como as funções que são invocadas durante a sua execução. A figura a seguir mostra o trecho inicial do algoritmo.

```
180     function PSO(){
181         $this->time_start = $this->microtime_float();
182         while(true){
183             while ($this->buscaposicao() > 9 ){
184                 $this->posicao = $this->geradiaperiodo();
```

Figura 18 - Algoritmo PSO (início)

A Figura 18 apresenta o código do algoritmo PSO utilizado para gerar a grade horária. A primeira ação executada é a inicialização do tempo, que servirá para calcular o tempo que o algoritmo levou para gerar a grade, ao final de sua execução, e também será utilizado para gerar o id de cada posição.

Em seguida tem início o laço de repetição que irá gerar as partículas e suas posições, esse laço irá se repetir dez vezes, gerando assim 10 partículas. As posições são geradas através da função geradiaperiodo, que será detalhada na figura a seguir.

```
91      function geradiaperiodo(){
92          //Uma turma não pode estar associada a mais de uma aula por semana;
93          $this->consulta = sqlsrv_query("select * from turma", $this->dbi);
94          $this->arrysstr = array(" ", ".");
95          //Gera uma posição baseado na hora que o registro foi inserido
96          $this->idposicao = md5(microtime() . rand(5, 15));
97          while($this->linha = sqlsrv_fetch_array($this->consulta)){
98              $this->idturma = $this->linha[0];
99              $this->idprofessor = $this->linha[1];
100             $this->consulta2 = sqlsrv_query("select * from disciplinaperiodo
                where iddisciplina=" . $this->linha[2], $this->dbi);
101             $this->iddia = $this->sorteiaiddia();
102             //Quando uma disciplina for oferecida para mais de um período, a
                disciplina deve estar na mesma turma, sendo a aula no mesmo dia e
                horário;
103             while($this->linha2 = sqlsrv_fetch_array($this->consulta2)){
104                 $this->idperiodo = $this->linha2[1];
105                 $this->consulta3 = sqlsrv_query("insert into diaperiodo
                    values('', '".$this->idposicao."', '".$this->idperiodo."', '".
                    $this->iddia."', '".$this->idturma."')", $this->dbi);
106             }
107         }
108         $this->consulta4 = sqlsrv_query("insert into posicao values('', '".
            $this->idposicao."', '')", $this->dbi);
109         return $this->idposicao;
110     }
```

Figura 19 - Função geradiaperiodo

A Figura 19 apresenta o código da função geradiaperiodo, que irá gerar as posições que serão percorridas pelas partículas. Para que as posições sejam criadas, o primeiro passo será inserir valores para a tabela “diaperiodo” do banco de dados, só então a tabela “posicao” terá dados inseridos. Para gerar os dados que serão inseridos na tabela “diaperiodo” será criado um *array* baseado na quantidade de turmas cadastradas, de forma que essas turmas serão colocadas uma em cada posição do *array*, evitando que as turmas se repitam, validando assim a restrição hard que diz que “uma turma não pode estar associada a mais de uma aula por semana”.

Em seguida será gerado o identificador da posição, baseado na hora em que o registro foi criado. Após a posição ser criada, será iniciado um laço de repetição que percorrerá o *array*, para inserir os dados na tabela “diaperiodo”. Nesta tabela serão inseridos o id de uma turma, o id do professor e o id da disciplina que estão relacionados a esta turma, e então será sorteado um dia, através da função sorteiaiddia, que fará o sorteio de forma aleatória, o id do dia sorteado também será inserido na tabela “diaperiodo”.

Após isso, é aplicado mais um laço de repetição, para validar a restrição *hard* que diz que “quando uma disciplina for oferecida para mais de um período, a disciplina deve estar na mesma turma, sendo a aula no mesmo dia e horário”. Assim será verificado se a disciplina está em mais de um período, e aplicará a regra. Por fim o dado de identificação da posição é inserido na tabela “posicao”. A próxima figura mostra a continuação do algoritmo PSO.

```
185                 $this->sqlinsertparticula = "insert into Particula values('',
                    '".$this->linha[0]."', '".$this->linha[2]."')";
186                 $this->insertparticula = mysql_query($this->sqlinsertparticula
                    , $this->dbi);
187             }
188             $this->fitness($this->posicao);
```

Figura 20 - PSO (continuação)

A Figura 20 apresenta a continuação do algoritmo após a posição ser gerada. Ainda dentro do laço de repetição a partícula é de fato criada, nessa fase será criado seu id e sua posição atual. Após a criação de todas as partículas o laço de repetição é fechado e a função *fitness* é invocada, para avaliar as posições. A figura a seguir mostra o código da função *fitness* que avalia as restrições *hard*.

```
113     function fitness($idposicao){
114         $this->lbest = 1000;
115
116
            //---------------------------------HARD---------------------------------
            -
117         //   Em um período duas disciplinas ministradas no mesmo dia não
            poderão estar no mesmo horário.
118         $this->consulta = sqlsrv_query("SELECT COUNT( b.nome ) , COUNT( c.nome
            ) , COUNT( c.hora )  FROM diaperiodo a, periodo b, dia c, turma d,
            professores e, disciplina f WHERE a.idperiodo = b.id AND a.iddia =
            c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina
            = f.id AND a.posicao =  '" . $idposicao . "' GROUP BY b.nome, c.nome,
            c.hora" , $this->dbi);
119         while($this->linha = sqlsrv_fetch_array($this->consulta)){
120             if($this->linha[0] > 1){ $this->lbest = 0; }
121         }
122         //  Um professor não pode lecionar duas disciplinas no mesmo horário;
123         $this->consulta2 = sqlsrv_query("SELECT c.nome, COUNT( c.nome ) ,
            c.hora, COUNT( c.hora ),  e.nome, count(e.nome) FROM diaperiodo a,
            periodo b, dia c, turma d, professores e, disciplina f WHERE
            a.idperiodo = b.id AND a.iddia = c.id AND a.idturma = d.id AND
            d.idprofessor = e.id AND d.iddisciplina = f.id and  a.posicao =  '" .
            $idposicao . "' GROUP BY b.nome, c.nome, c.hora, e.nome" , $this->dbi);
124         while($this->linha2 = sqlsrv_fetch_array($this->consulta2)){
125             if($this->linha2[0] > 1){ $this->lbest = 0; }
126         }
```

Figura 21 - Função *fitness* (restrições *hard*)

A Figura 21 mostra a parte do código da função *fitness* que avalia as restrições *hard*. Apenas duas das quatro restrições são avaliadas, pois as outras duas restrições foram respeitadas no momento em que as posições foram geradas.

No início da função a variável *lbest* recebe mil pontos. Esta variável representa a pontuação da posição, e sofrerá decréscimo para cada restrição que a posição não atender.

Para cada uma das restrições hard – “em um período duas disciplinas ministradas no mesmo dia não poderão estar no mesmo horário” e “um professor não pode lecionar duas disciplinas no mesmo horário” – a função fará uma consulta ao BD que irá selecionar os dados da posição atual, cada consulta realizada de modo que a verificação de cada regra possa ser aplicada. Após isso utilizará um *array* que irá percorrer o resultado da consulta e verificará se a restrição foi infringida. Caso tenha sido, a variável *lbest* receberá o valo zero. As próximas figuras mostram a aplicação das restrições *soft*.

```
130         $this->consulta3 = sqlsrv_query("SELECT f.nome as Disciplina, count(
            f.nome) FROM diaperiodo a, periodo b, dia c, turma d, professores e,
            disciplina f WHERE a.idperiodo = b.id AND a.iddia = c.id AND a.idturma
            = d.id AND d.idprofessor = e.id AND d.iddisciplina = f.id and
            a.posicao = '" . $idposicao . "'  group by Disciplina" , $this->dbi);
131         while($this->linha3 = sqlsrv_fetch_array($this->consulta3)){
132             if($this->linha3[1] > 1){
133                 $this->consulta4 = sqlsrv_query("SELECT b.nome, c.nome,
                    c.hora,  e.nome, f.nome FROM diaperiodo a, periodo b, dia c,
                    turma d, professores e, disciplina f WHERE a.idperiodo = b.id
                    AND a.iddia = c.id AND a.idturma = d.id AND d.idprofessor =
                    e.id AND d.iddisciplina = f.id and  a.posicao = '" . $idposicao
                     . "' and f.nome='".$this->linha3[0]."' group by f.nome,
                    c.nome" , $this->dbi);
134                 if (sqlsrv_num_rows($this->consulta4)>1){
135                     $this->lbest = $this->lbest - 100;
136                 }
137             }
138         }
```

Figura 22 - Função *fitness* (restrição *soft* 1)

A Figura 22 mostra a parte do código da função *fitness* que avalia a restrição *soft* 1 – “quando uma disciplina for ofertada duas vezes por professores diferentes, devem preferencialmente ser no mesmo dia”.

Para tratar dessa regra, a função fará uma consulta ao BD que irá selecionar os dados da posição atual e depois irá ordenar o resultado da consulta e após isso utilizará um *array* que irá percorrer o resultado da consulta e verificará se existe uma disciplina que é ofertada mais de uma vez no semestre. Se a condição for

verdadeira, será feita uma nova consulta ao BD, para buscar as disciplinas que são ofertadas duas vezes e após isso é feita a verificação para ver se estão sendo ofertadas no mesmo dia. Se a restrição for infringida a variável *lbest* sofrerá um decréscimo de 100 pontos. A próxima figura mostra o trecho do código da função *fitness*, que trata restrição da segunda restrição *soft*.

```
140         $this->consulta5 = sqlsrv_query("SELECT  b.nome ,  c.nome,
            count(c.nome)  , c.hora FROM diaperiodo a, periodo b, dia c, turma d,
            professores e, disciplina f WHERE a.idperiodo = b.id AND a.iddia =
            c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina
            = f.id AND a.posicao =  '" . $idposicao . "' group by  b.nome, c.nome"
            , $this->dbi);
141         while($this->linha5 = sqlsrv_fetch_array($this->consulta5)){
142             if($this->linha5[2] == 1 && $this->linha5[3]=="Tarde"){ $this->
                lbest = $this->lbest - 100; }
143         }
```

Figura 23 - Função *fitness* (restrição soft 2)

A Figura 23 mostra a parte do código da função *fitness* que avalia a restrição *soft* 2 – “aulas poderão acontecer no horário da tarde, se não houver possibilidade de encaixa-las nos demais horários”.

Para tratar dessa regra a função fará uma consulta ao BD que irá selecionar os dados da posição atual e depois irá ordenar o resultado da consulta. Após isso utilizará um *array* que irá percorrer o resultado da consulta e verificará se a restrição foi infringida. Caso tenha sido, a variável *lbest* sofrerá um decréscimo de 100 pontos. A próxima figura mostra o último trecho do código da função *fitness*, que trata das últimas regras.

```
146         $this->consulta6 = sqlsrv_query("SELECT  d.id, c.nome, g.daraula,
            g.ndaraula FROM diaperiodo a, periodo b, dia c, turma d, professores
            e, disciplina f, preferencias g WHERE a.idperiodo = b.id AND a.iddia =
            c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina
            = f.id AND g.idprofdisc=d.id and  a.posicao ='" . $idposicao . "'" ,
            $this->dbi);
147         while($this->linha6 = sqlsrv_fetch_array($this->consulta6)){
148             if($this->linha6[1] !=  $this->linha6[2]){ $this->lbest = $this
                ->lbest - 100; }
149             if($this->linha6[1] ==  $this->linha6[3]){ $this->lbest = $this
                ->lbest - 100; }
150         }
151
152
153         $this->consultafinal = sqlsrv_query("update posicao set lbest=" . $this
            ->lbest . " where posicao='" . $idposicao . "'", $this->dbi);
154
155     }
```

Figura 24 - Função *fitness* (restrições *soft* preferências)

A Figura 24 mostra a parte do código da função *fitness* que avalia as restrições *soft* que tratam de preferência – “determinados professores não podem dar aula em certos dias”, “determinados professores gostariam de dar aula em certos dias” e “determinadas disciplinas não podem ser ministradas em certos dias”.

Para tratar dessas três regras a função fará uma única consulta ao BD que irá selecionar os dados da posição atual, incluindo dados da tabela “preferencias”, e depois irá ordenar o resultado da consulta. Após isso utilizará um *array* que irá percorrer o resultado da consulta e fará duas verificações: a primeira verificará a preferência para que a aula seja em um determinado dia; a segunda verificará a preferência para que a aula não seja em um determinado dia. Para cada restrição que tenha sido restringida, a variável *lbest* sofrerá um decréscimo de 100 pontos.

A última parte da função *fitness* é atualizar a pontuação da posição no banco de dados. A Figura 25 mostra a continuidade do algoritmo PSO após a avaliação ser realizada.

```
188                 $this->geragbest();
189                 if ($this->consultaposicao = mysql_query("SELECT * FROM posicao
                    where lbest >= 700")){
190                     break;
191                 }
192             }
193             $this->time_end = $this->microtime_float();
194             $this->timem = $this->time_end - $this->time_start;
195             return $this->timem;
196         }
```

Figura 25 - PSO (última parte)

Após a avaliação das posições ocorre a atualização do *gbest* (melhor posição do bando), através da função geragbest. Essa função compara a melhor posição de cada partícula (*pbest*) e aquela que tiver a maior pontuação será o novo *gbest,* e então a posição e o id partícula são inseridos na tabela “gbest”. Em seguida é feita uma verificação para ver se o valor de *lbest* (pontuação) da posição é maior ou igual a 700. Caso seja, a condição de parada – obedecer todas as restrições *hard* e pelo menos uma *soft* – é alcançado e o laço de repetição é parado. Caso não seja, o processo recomeça a partir da atualização das posições de cada partícula.

Quando a condição de parada é alcançada a função retorna o tempo que durou sua execução, ou seja, quanto tempo o programa levou para construir a grade de horários, como pode ser visto na Figura 26.

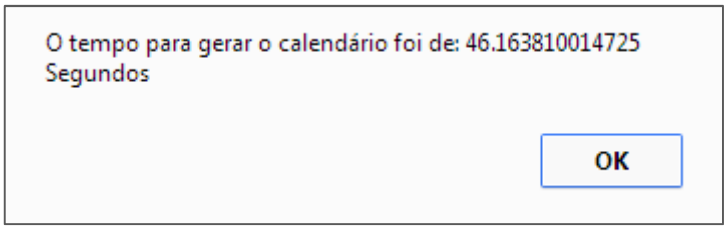

Figura 26 - Mensagem de tempo

Esta figura mostra a mensagem que é exibida ao usuário após a execução do algoritmo PSO. Após o usuário clicar em “OK” o sistema exibe a grade de gerada na interface, como pode ser visto na Figura 28.

	Segunda-feira	Terça-feira	Quarta-feira	Quinta-feira	Sexta-feira	Sábado
1º Período	**Noite - Turma:92** - Algoritmos e Programação I - Prof:Madianita Bogo **Tarde - Turma:106** - Comunicação e Expressão - Prof:Cristina D'Ornellas Filipakis Souza	**Noite - Turma:90** - Algoritmos e Programação I - Prof:Cristina D'Ornellas Filipakis Souza		**Noite - Turma:104** - Introdução a Computação - Prof:Fabiano Fagundes	**Noite - Turma:95** - Banco de Dados I - Prof:Fernando Oliveira	
2º Período	**Noite - Turma:93** - Algoritmos e Programação II - Prof:Edeilson Milhomem da Silva	**Noite - Turma:94** - Arquitetura e Organização de Computadores I - Prof:Jackson Gomes de Souza	**Noite - Turma:105** - Lógica de Predicados - Prof:Parcilene Fernandes de Brito	**Noite - Turma:96** - Banco de Dados II - Prof:Fernando Oliveira		

Figura 27 - Grade gerada

A Figura 27 mostra como é exibida a grade de horários gerada pelo protótipo. Essa grade é mostrada no formato de tabela, onde cada linha representa um período do curso e cada coluna um dia da semana. O conteúdo de cada dia é ordenado pelo horário da aula, caso haja mais de uma aula no dia, e os dados exibidos são: horário, número da turma, disciplina e professor. A Figura 28 mostra o código que seleciona o resultado para ser apresentado na interface.

```
213         $sql = "select * from periodo order by nome";
214         $db->query($sql);
215         for ($i = 0; $i < $db->linhas(); $i++) {
216           $idperiodo = $db->result($i, 0);
217           $nomeperiodo = $db->result($i, 1);
218                 $sql2 = "select b.nome as Periodo, c.nome as Dia, c.hora as Hora, a.idturma
                    as Turma, e.nome as Professor, f.nome as Disciplina  from diaperiodo a,
                    periodo b, dia c, turma d, professores e, disciplina f, posicao g, gbest h
                    where a.idperiodo=b.id and a.iddia=c.id and a.idturma=d.id and
                    d.idprofessor=e.id and d.iddisciplina=f.id and a.posicao=g.posicao and
                    h.idposicao=g.id and a.idperiodo=" . $idperiodo . " order by c.hora";
219                 $dados=sqlsrv_query($sql2);
220                 $dados2=sqlsrv_query($sql2);
221                 $dados3=sqlsrv_query($sql2);
222                 $dados4=sqlsrv_query($sql2);
223                 $dados5=sqlsrv_query($sql2);
224                 $dados6=sqlsrv_query($sql2);
```

Figura 28 - Buscando o resultado

Para que os dados sejam exibidos na interface é criado um laço de repetição que fará uma consulta ao banco de dados para selecionar, para cada período, os dados da melhor posição, através da tabela "*gbest*". Com a busca realizada, será criado um *array* para cada dia da semana. Então os dados inseridos em *divs* que já haviam sido criadas. A figura a seguir mostra como os dados são inseridos nas *divs.*

```
225                 echo '<div class="row">';
226                 echo '  <div class="col-md-1 grid"><b>'.$nomeperiodo.'</b></div>
227                       <div class="col-md-1 grid">'; while ($row = sqlsrv_fetch_array($dados
                          )){if($row[1] == "segunda-feira"){echo "<b>". $row[2] ." - Turma:" .
                          $row[3]. "</b> - " . $row[5] . " - Prof:". $row[4] . '<br><br>';}}
                          echo '</div>
228                       <div class="col-md-1 grid">'; while ($row2 = sqlsrv_fetch_array(
                          $dados2)){if($row2[1] == "terça-feira"){echo "<b>". $row2[2] ." -
                          Turma:" . $row2[3]. "</b> - " . $row2[5] . " - Prof:". $row2[4] .
                          '<br><br>';}} echo '</div>
229                       <div class="col-md-1 grid">'; while ($row3 = sqlsrv_fetch_array(
                          $dados3)){if($row3[1] == "quarta-feira"){echo "<b>". $row3[2] ." -
                          Turma:" . $row3[3]. "</b> - " . $row3[5] . " - Prof:". $row3[4] .
                          '<br><br>';}} echo '</div>
230                       <div class="col-md-1 grid">'; while ($row4 = sqlsrv_fetch_array(
                          $dados4)){if($row4[1] == "quinta-feira"){echo "<b>". $row4[2] ." -
                          Turma:" . $row4[3]. "</b> - " . $row4[5] . " - Prof:". $row4[4] .
                          '<br><br>';}} echo '</div>
231                       <div class="col-md-1 grid">'; while ($row5 = sqlsrv_fetch_array(
                          $dados5)){if($row5[1] == "sexta-feira"){echo "<b>". $row5[2] ." -
                          Turma:" . $row5[3]. "</b> - " . $row5[5] . " - Prof:". $row5[4] .
                          '<br><br>';}} echo '</div>
232                       <div class="col-md-1 grid">'; while ($row6 = sqlsrv_fetch_array(
                          $dados6)){if($row6[1] == "sabado"){echo "<b>". $row6[2] ." - Turma:" .
                           $row6[3]. "</b> - " . $row6[5] . " - Prof:". $row6[4] . '<br><br>';}}
                           echo '</div>
233                       ';
234                 echo "</div>";
```

Figura 29 - Inserindo os dados na tabela

Como mostra a Figura 29, para cada dia da semana é feito um laço de repetição que irá rodar pelos *arrays* que foram definidos na Figura 28, e os dados

serão inseridos, apresentando em cada linha uma informação referente à turma, conforme já apresentado na Figura 27.

A seção a seguir apresenta as conclusões finais acerca da elaboração deste trabalho.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi implementado um protótipo do sistema PSO-*Timetabling Generator,* que se propõe a gerar a grade de horários das aulas (Timetabling Universitário) do Departamento de Computação do Centro Universitário Luterano de Palmas (CEULP/ULBRA). Esse sistema tem como mecanismo principal o algoritmo *Particle Swarm Optimization* (PSO), um algoritmo de otimização estocástico que se baseia na interação de pássaros voando em busca de alimento.

Durante a criação da solução foi necessário fazer a modelagem do domínio. Para isso foi necessário abstrair e combinar os elementos do problema de Timetabling e as necessidades da instituição com elementos fundamentais do algoritmo, como o ambiente, as partículas e as posições. E, também, entender e definir como ocorrerão os processos fundamentais do algoritmo, que são a inicialização, construção e avaliação da solução e a atualização das posições.

Para a elaboração do PSO-*Timetabling Generator,* o ambiente foi representado por um conjunto de professores, disciplinas, períodos e dias letivos. Os atributos necessários de cada um desses elementos foram definidos com base na observação do problema real na instituição. A partir disso, definiu-se como seriam os relacionamentos no modelo de dados e posteriormente como seria a sua implementação.

Uma das principais dificuldades nesse processo foi a modelagem do domínio, pois inicialmente a abstração do problema e a sua aplicação ao algoritmo estava um pouco confusa e obscura, a solução foi ficando mais clara a medida que trabalhos relacionados – principalmente a aplicação de algoritmos evolucionários para solucionar problemas de *Timetabling* – eram lidos e comparados ao domínio específico deste trabalho.

Utilizou-se como base para a construção da solução, o conceito de semestre letivo, tendo como contexto uma semana de aula. Para isso, foram consideradas as disciplinas do curso e os professores que ministrariam tais disciplinas no semestre, formando a partir disso as turmas. Além disso, foi considerado o período do curso em que as disciplinas devem ser ofertadas.

Para a construção do algoritmo, a partícula foi fisicamente formulada como descrita na teoria, contendo sua posição atual, armazenando a melhor posição em que já esteve e tendo conhecimento da melhor posição entre todas as partículas geradas. Cada posição foi criada como sendo uma grade completa, ou seja, a possível solução para o problema; para que isso fosse feito, foi necessário que a posição fosse um agrupador de turmas ligadas aos dias da semana.

A parte do algoritmo que trata da criação de cada posição foi feita mesclando a geração aleatória de dias à aplicação de duas das regras do sistema. E a função de avaliação das posições foi criada com o restante das restrições. Essas restrições foram baseadas nas necessidades e regras da instituição.

Os resultados são considerados satisfatórios se considerado que o objetivo do trabalho foi alcançado, por outro lado, algumas lacunas foram deixadas: A saída ideal (criação da grade respeitando todas as restrições *hard* e *soft*) não pôde ser alcançada, pois isso demandaria muito tempo de execução, tornando o procedimento inviável, por esse motivo optou-se por considerar na criação da solução final a posição que atendesse todas as restrições *hard* e pelo menos uma restrição *soft*.

Mas por se tratar de um protótipo, propõe-se como trabalho futuro o melhoramento do programa. O que pode ser feito através das seguintes propostas: otimizar o algoritmo; melhorar as pesquisas e operações no banco de dados, incluindo a criação de *views* que facilitem as buscas; melhorar o código como um todo, deixando-o mais enxuto e; melhorar a interface, deixando-a mais amigável ao usuário.

6 REFERÊNCIAS BIBLIOGRÁFICAS

AL-RIFAIE; Mohammad Majid; BLACKWELL, Tim. Bare Bones Particle Swarms with Jumps. **Lecture Notes in Computer Science.** v. 7461, p. 49-60, 2012.

ALLAOUA, Boumediene; LAOUFI, Abdellah; GASBAOUI, Brahim; ABDERRAHMANI, Abdessalam. Neuro-Fuzzy DC Motor Speed Control Using Particle Swarm Optimization. **Leonardo Electronic Journal of Practices and Technologies,** v. 15, p. 1-18, jul. / dez. 2009.

ANGELINE, P. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In: **Proceedings of evolutionary programming VII.** Berlin, Alemanha: Springer, 1998, p. 601-610.

AROGUNDADE, Oluwasefunmi T.; AKINWALE, Adio T.; AWEDA, Omotoyosi M. A Genetic Algorithm Approach for a Real-World University Examination Timetabling Problem. **International Journal of Computer Applications,** v. 12, n. 5, p. 1-4. 2010.

AUGUSTUS, Glaucus. **Inteligência de enxame e o algoritmo das abelhas.** 27 p. Monografia. Universidade de São Paulo, 2009

BLACKWELL, T.; BENTLEY, P. J.. Don't push me! Collision-avoiding swarms. In: **Proceedings of the IEEE congress on evolutionary computation (CEC).** Honolulu, HI. Piscataway: IEEE, 2002, p. 1691–1696.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. **Swarm Intelligence**: From Natural to Artificial Systems. Oxford University Press, New York, NJ, 1999.

BURKE, Edmund K; BYKOV, Yuri; NEWALL, James; PETROVIC, Sanja. A Time-Predefined Approach To Course Timetabling. **Yugoslav Journal of Operations Research,** v. 13 p. 139-151, 2003.

BURKE, Edmund K; CAUSMAECKER, Patrick; BERGHE Greet V.; LANDEGHEM, Hendrik V. The State of the Art of Nurse Rostering. **Journal of Scheduling,** v. 7, n. 6, p. 441-499, nov. / dez. 2004.

BURKE, Edmund K.; MCCOLLUM, Barry; MEISELS, Amnon; PETROVIC, Sanja; QU, rong. A Graph-Based Hyper-Heuristic for Educational. **European Journal of Operational Research,** v. 176, n. 1, p. 177-192, 2006.

BURKE, Edmund K; KENDALL, Graham; MISIR, Mustafa, ÖZCAN, Ender. Monte Carlo hyper-heuristics for examination timetabling. In: **Annals of Operations Research**, Alemanha: Baltzer Science Publishers, 2012, v. 196, p. 72-90. 2010.

BURKE, Edmund K.; KENDALL, Graham; SOUBEIGA, E. A Tabu-Search Hyperheuristic for Timetabling and Rostering. **Journal of Heuristics,** v. 9, p. 451-470, 2003.

BURKE, Edmund K; PETROVIC, Sanja. Recent Research Directions in Automated Timetabling. **European Journal of Operational Research – EJOR,** v. 140, n. 2, 2002.

CAPRARA, A.; FISCHETTI, M.; GUIDA, P; MONACI, M.; SACCO, G; TOTH, P. Solution of Real-World Train Timetabling Problems. In: **ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS)**, 34, Washington, USA, 2001, v. 3, p. 3030.

CHAN, Yam Ling. **A Genetc Algorithm (GA) Shell for Iterative Timetabling.** 1994. 66 p. Dissertação (Mestrado) – Departamento de Ciência da Computação RMIT.

CUI, Xiaohui; CHARLES, Jesse St.; POTOK, Thomas E. A Simple Distributed Particle Swarm Optimization for Dynamic and Noisy Environments. In: KRASNOGOR, N.; MELIÁN-BATISTA, B.; MORENO-PÉREZ, J.A.; MORENO-VEGA,

J.M.; PELTA, D.A. **Nature Inspired Cooperative Strategies for Optimization (NICSO 2008):** Studies in Computacional Intelligence. 2009. Cap. 8. p. 89-102.

Daskalaki, S; BIRBAS, T.; HOUSOS, E. An integer programming formulation for a case study in university timetabling. **European Journal of Operational Research,** v. 153, p 117-135, 2004.

DETIENNE, Boris; P´ERIDY, Laurent; PINSON, Éric; RIVREAU, David. Cut Generation for an Employee Timetabling Problem. **European Journal of Operational Research,** v. 197, p. 1178-1184, 2007.

EASTON, Kelly; NEMHAUSER, George; TRICK, Micheal. The Traveling Tournament Problem Description and Benchmarks**.** In: **Proceedings of International Conference on Principles and Practice of Constraint Programming (CP '01)**, 7, Londres, UK, 2001, p. 580-584.

GUNAWAN, Aldy; NG, Kien Ming; POH, Kim Leng. **A Mathematical Programming Model For A Timetabling Problem.** p. 1-6, 2006.

HOSNY, Manar; FATIMA, Shameem. A Survey of Genetic Algorithms for the University Timetabling Problem. In: **International Conference on Future Information Technology IPCSIT,** Singapura, 2011.

HSIEH, Horng-I; LEE, Tian-Shyug. A Modified Algorithm of Bare Bones Particle Swarm Optimization. **IJCSI International Journal of Computer Science Issues.** v. 7, n, 6, p. 12-17, 2010.

KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: **Proceedings of IEEE International Conference on Neural Networks. Piscataway**, NJ, USA, 1995. p. 1942–1948.

KENNEDY, J.; EBERHART, R. C. A discrete binary version of the particle swarm algorithm. In: **Proceedings of the conference on systems, man, and cybernetics.** Piscataway: IEEE, 1997 p. 4104–4109.

KENNEDY, J.; SPEARS, W. M. Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In: **Proceedings international conference on evolutionary computation.** Piscataway: IEEE, 1998, p. 78-83.

KRINK, T.; VESTERSTROØM, J. S.; RIGET, J. Particle swarm optimization with spatial particle extension. In: **Proceedings of the IEEE congress on evolutionary computation (CEC-2002).** Piscataway: IEEE, 2002 p. 1474–1479.

LEITÃO, Gustavo. **Probabilidade e Estatística.** Instituto Federal de Educação, Ciência e Tecnologia, Rio Grande do Norte, 2010.

LOØVBJERG, M.;KRINK, T. Extending particle swarms with self-organized criticality. In **Proceedings of the IEEE congress on evolutionary computation (CEC-2002).** Piscataway: IEEE, 2002 p. 1588–1593.

LOØVBJERG, M.; RASMUSSEN, T. K.; KRINK, T. Hybrid particle swarm optimiser with breeding and subpopulations. In: **Proceedings of the third genetic and evolutionary computation conference (GECCO.** San Francisco: Kaufmann, 2001, p. 469–476.

LUNA, Erika Hernández. **Diseño de circuitos lógicos combinatórios usando optimización mediante cúmulos de partículas.** 126 p. Dissertação (Mestrado) - Instituto Politécnico Nacional do México, 2004. Disponível em: <http://pt.scribd.com/doc/72869380/49/Algoritmo-de-PSO-binario> Acesso em: 14 Dez. 2012.

MEISELS, Amnon; SCHAERF, Andrea. Modelling ans Solving Employee Timetabling Problems. In: **Annals of Mathematics and Artificial Intelligence,** Kluwer Academic Publishers, 2003, v. 39, n 1-2, p. 41-59, 2002.

MELLO, Alan Godoy Souza. **Aplicação de Redes Complexas para a Definição de Vizinhança na Otimização por Enxame de Partículas.** 2010. Dissertação (Mestrado) - Universidade Estadual de Campinas.

MESSAC, A; ISMAIL-YAHAYA, A; MATTSON, C.A. The normalized normal constraint method for generating the Pareto frontier. **Structural and Multidisciplinary Optimization,** v. 25, n. 2, p. 86-98, 2003. Disponível em: <http://link.springer.com/article/10.1007%2Fs00158-002-0276-1?LI=true#page-1> Acesso em: 24 Nov. 2012.

MIRANDA, V.; FONSECA, N. New evolutionary particle swarm algorithm (EPSO) applied to voltage/VAR control. In: **Proceedings of the 14th power systems computation conference (PSCC).** Seville, Spain, 2002, v. 21, n. 5, p. 1–6.

MONTERO, Elizabeth; RIFF, María-Cristina; ALTAMIRANO, Leopoldo. A PSO algorithm to solve a Real Course+Exam Timetabling Problem. In: **International conference on swarm intelligence (ICSI 2011),** Paris, França, 2011, 24p.

NOCEDAL, Jorge; WRIGHT, Stephen J. **Numerical Optimization**. Springer, New York, 1999. Disponível em: <http://books.google.com.br/books?id=w1kJYpOkPykC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false> Acesso em: 14 dez. 2012.

OMRAN, Mahamed G.H.; ANDRIES, Engelbrecht P.; SALMAN, Ayed. Bare ones differential evolution. **European Journal of Operational Research.** p. 1-12, 2008.

OMRAN, M; AL-SHARHAN S. Barebones Particle Swarm Methods for Unsupervised Image Classification. In: **IEEE Congress on Evolutionary Computation (CEC).** Singapura, 2007, p. 3247 – 3252.

PAIM, A. S.; Greis, I. G. Abordagens Para Elaboração Automatizada De Tabela De Horários Acadêmicos. In: **Seminário Intermunicipal De Pesquisa,** 6, 26 p. 2008.

PARSOPOULOS, K. E.; VRAHATIS, M. N. Particle swarm optimizer in noisy and continuously changing environments. In: **Artificial intelligence and soft computing**, M. H. Hamza (Ed.), Anaheim: IASTED/ACTA, 2001 p. 289–294.

PETROVIC, Sanja; BURKE, Edmund K. University Timetabling. In: LEUNG, J. (ed.) **Handbook of Scheduling**: Algorithms, Models, and Performance Analysis. CRC Press, 2004 Cap. 45.

POLI, Riccardo; KENNEDY, James; BLACKWELL, Tim. Particle swarm optimization: An Overview. **Swarm Intelligence,** v. 1, n. 1, p. 33-57, 2007.

POLI, R.; STEPHENS, C. R. Constrained molecular dynamics as a search and optimization tool. In: **Proceedings of the 7th European conference on genetic programming (EuroGP).** Lecture notes in computer science, M. Keijzer et al. (Eds.), Coimbra, Portugal, 2004, Berlin: Springer, 2002, v. 3002, p. 150–161.

PRATA, Bruno de Athayde. Um algoritmo enxame de partículas para uma variante do problema de máxima cobertura. **GEPROS. Gestão da Produção, Operações e Sistemas.** p. 139-148, 2012.

REYES-SIERRA, Margarita; COELLO, Carlos A. Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art. **International Journal of Computational Intelligence Research,** v. 2, n. 3, p. 287-308, 2006.

ROSENDO, Matheus. **Um Algoritmo De Otimização Por Nuvem De Partículas Para Resolução De Problemas Combinatórios.** 2010. Dissertação (Mestrado) – Universidade Federal do Paraná, Curitiba.

SCHAERF, A. A Survey of Automated Timetabling. **Artificial Intelligence Review,** v. 13, n. 2, p. 87-127, 1995.

SICILIANO, A. V. **Algoritmos Genéticos e Particle Swarm Optimization e suas aplicações problemas de Guerra Eletrônica.** 5 p. 2007.

SILVA, Luiz, A. W. **Otimização De Uma Cadeia De Suprimentos Usando A Metaheurística Enxame De Partículas.** 86 p. Dissertação (Mestrado) – Pontifícia Universidade Católica Do Paraná, 2008.

SOUZA, G. R. **Uma abordagem por nuvem de partículas para problemas de otimização combinatória.** 75 p. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2006.

SUDHOLT, Dirk; WITT, Carsten. Runtime Analysis of a Binary Particle Swarm Optimizer. **Theoretical Computer Science,** v. 411, n. 21, p. 2084−2100, 2010.

TASGETIREN, M. Fatih; SUGANTHAN, P. N.; PAN, Quan-Ke. A Discrete Particle Swarm Optimization Algorithm for the Generalized Traveling Salesman Problem. In: **Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO '07).** NY, USA, 2007, p. 158-165.

VIEIRA, F; MACEDO, H. Sistema de Alocação de Horários de Cursos Universitários: Um Estudo de Caso no Departamento de Computação da Universidade Federal de Sergipe. In: **Scientia Plena,** 7, n. 3, p. 1-12, 2011.

WHITE, Tony; PAGUREK, Bernard. Towards Multi-Swarm Problem Solving In: **In Proceedings of Third International Conference on Multi-Agent Systems (ICMAS' 98. IEEE Computer Society**, 1998. p. 333–340.

XIE, X.; ZHANG, W.; YANG, Z. Dissipative particle swarm optimization. In: **Proceedings of the IEEEcongress on evolutionary computation (CEC).** Honolulu, HI. Piscataway: IEEE, 2002 p. 1456–1461.

ZHU, Yan-fei; TANG, Xiong-min. Overview of Swarm Intelligence. In: **International Conference on Computer Application and System Modeling (ICCASM 2010),** Taiyuan, v. 9, p. 400-403, 2010.

ZUBEN, Fernando J. Von; ATTUX, Romis R. F. **Inteligência de Enxame.** DCA/FEEC/Unicamp e DECOM/FEEC/Unicamp, 2008.

7 APÊNDICES

Apêndice A – Arquivo *class*.php

```php
<?php
define("DB_HOSTI","localhost"); // host de conexão com o SQL
define("DB_USERNAMEI","root"); // nome do usuário para conexão
define("DB_PASSWORDI","cmbr*033"); // senha do usuário para conexão
define("DB_DATABASEI","PSO_TT_Generator"); // nome do bd


class sqlsrv
{
	var $dbi;
	var $query;

	// função que starta o Sql, sem ela é impossivel conectar ao banco
	function open()
	{
		// conecta o bd com as variáveis predefinidas
		$this->dbi = sqlsrv_connect(DB_HOSTI, DB_USERNAMEI,
		DB_PASSWORDI);
		if (!$this->dbi) {
			echo "Erro na conexão!";
		}
		if (!sqlsrv_select_db(DB_DATABASEI)) {
			echo "Erro na seleção do banco de dados!";
		}
	}

	// fecha a conexão com o bd
	function close()
	{
		Sqlsrv_close($this->dbi);
	}

	// executa uma string SQL
	function query($sql)
	{
		$this->query = sqlsrv_query($sql, $this->dbi);
		return $this->query;
	}

	// retorna quantas linhas aquela query resultou
	function linhas()
	{
		return sqlsrv_num_rows($this->query);
	}

	// retorna o conteúdo do campo e linha escolhidos
	function result($linha, $campo)
	{
		return sqlsrv_result( $this->query, $linha, $campo );
	}

	// mesma coisa que o result()
	function retorno($linha, $campo)
	{
		return sqlsrv_result($this->consulta, $linha, $campo);
	}

	// mesma coisa que o linhas()
	function resultado()
	{
		return sqlsrv_num_rows($this->consulta);
	}

	//Busca o nome do Professor
	function getprofessor($id)
```

```
{
	$this->consulta = sqlsrv_query("select * from professores where
id=" . $id, $this->dbi);
	return sqlsrv_result( $this->consulta, 0, 1 );
}

//Busca o Nome da Disciplina
function getdisciplina($id)
{
	$this->consulta = sqlsrv_query("select * from disciplina where
	id=" . $id, $this->dbi);
	return sqlsrv_result( $this->consulta, 0, 1 );
}

//Busca pelo período
function getperiodo($id)
{
	$this->consulta = sqlsrv_query("select * from periodo where
	id=" . $id, $this->dbi);
	return sqlsrv_result( $this->consulta, 0, 1 );
}

//Seleciona aleatóriamente um dia de aula
function sorteiaiddia(){
	$this->consultasorteiaiddia = sqlsrv_query("SELECT id FROM
	`dia` order by rand()limit 1", $this->dbi);
	return sqlsrv_result( $this->consultasorteiaiddia, 0, 0 );
}

//Busca posição valida
function buscaposicao(){
	$this->consultaposica = sqlsrv_query("SELECT * FROM posicao
	where lbest > 0", $this->dbi);
	return sqlsrv_num_rows($this->consultaposica);
}

//Popula a Tabela DiaPeriodo
function geradiaperiodo(){
	//Uma turma não pode estar associada a mais de uma aula por
	semana;
	$this->consulta = sqlsrv_query("select * from turma", $this-
	>dbi);
	$this->arrysstr = array(" ", ".");
	//Gera uma posição baseado na hora que o registro foi inserido
	$this->idposicao = md5(microtime() . rand(5, 15));
	while($this->linha = sqlsrv_fetch_array($this->consulta)){
		$this->idturma = $this->linha[0];
		$this->idprofessor = $this->linha[1];
		$this->consulta2 = sqlsrv_query("select * from
		disciplinaperiodo where iddisciplina=" . $this->linha[2],
		$this->dbi);
		$this->iddia = $this->sorteiaiddia();
		//Quando uma disciplina for oferecida para mais de um
		período, a disciplina deve estar na mesma turma, sendo a
		aula no mesmo dia e horário;
		while($this->linha2 = sqlsrv_fetch_array($this-
		>consulta2)){
			$this->idperiodo = $this->linha2[1];
			$this->consulta3 = sqlsrv_query("insert into
			diaperiodo values('', '".$this->idposicao.
			"','".$this->idperiodo."', '".$this->iddia."',
			'".$this->idturma."')", $this->dbi);
		}
	}
	$this->consulta4 = sqlsrv_query("insert into posicao values('',
	'".$this->idposicao."', '')", $this->dbi);
	return $this->idposicao;
}
```

```
//Faz a atualização do lbest nos registros de Posição baseando-se nas restrições aplicadas

function fitness($idposicao){
	$this->lbest = 1000;

	//----------------------------------HARD-----------------------------------
	// Em um período duas disciplinas ministradas no mesmo dia não poderão estar no mesmo horário.
	$this->consulta = sqlsrv_query("SELECT COUNT( b.nome ) , COUNT( c.nome ) , COUNT( c.hora ) FROM diaperiodo a, periodo b, dia c, turma d, professores e, disciplina f WHERE a.idperiodo = b.id AND a.iddia = c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina = f.id AND a.posicao = '" . $idposicao . "' GROUP BY b.nome, c.nome, c.hora" , $this->dbi);
	while($this->linha = sqlsrv_fetch_array($this->consulta)){
		if($this->linha[0] > 1){ $this->lbest = 0; }
	}
	// Um professor não pode lecionar duas disciplinas no mesmo horário;
	$this->consulta2 = sqlsrv_query("SELECT c.nome, COUNT( c.nome ) , c.hora, COUNT( c.hora ), e.nome, count(e.nome) FROM diaperiodo a, periodo b, dia c, turma d, professores e, disciplina f WHERE a.idperiodo = b.id AND a.iddia = c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina = f.id and a.posicao = '" . $idposicao . "' GROUP BY b.nome, c.nome, c.hora, e.nome" , $this->dbi);
	while($this->linha2 = sqlsrv_fetch_array($this->consulta2)){
		if($this->linha2[0] > 1){ $this->lbest = 0; }
	}

	//----------------------------------SOFT-----------------------------------
	//Quando uma disciplina for ofertada duas vezes por professores diferentes, devem preferencialmente ser no mesmo dia;
	$this->consulta3 = sqlsrv_query("SELECT f.nome as Disciplina, count( f.nome) FROM diaperiodo a, periodo b, dia c, turma d, professores e, disciplina f WHERE a.idperiodo = b.id AND a.iddia = c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina = f.id and a.posicao = '" . $idposicao . "' group by Disciplina" , $this->dbi);
	while($this->linha3 = sqlsrv_fetch_array($this->consulta3)){
		if($this->linha3[1] > 1){
			$this->consulta4 = sqlsrv_query("SELECT b.nome, c.nome, c.hora, e.nome, f.nome FROM diaperiodo a, periodo b, dia c, turma d, professores e, disciplina f WHERE a.idperiodo = b.id AND a.iddia = c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina = f.id and a.posicao = '" . $idposicao . "' and f.nome='".$this->linha3[0]."' Group by f.nome, c.nome" , $this->dbi);
			if (sqlsrv_num_rows($this->consulta4)>1){
				$this->lbest = $this->lbest - 100;
			}
		}
	}

	// Aulas poderão acontecer no horário da tarde, se não houver possibilidade de encaixa-las nos demais horários(Cada turma que não cumprir essa regra será subtraído -100 do lbest da posição).
	$this->consulta5 = sqlsrv_query("SELECT b.nome , c.nome, count(c.nome) , c.hora FROM diaperiodo a, periodo b, dia c, turma d, professores e, disciplina f WHERE a.idperiodo = b.id AND a.iddia = c.id AND a.idturma = d.id AND d.idprofessor = e.id AND d.iddisciplina = f.id AND a.posicao = '" . $idposicao . "' group by b.nome, c.nome" , $this->dbi);
while($this->linha5 = sqlsrv_fetch_array($this->consulta5)){
```

```
            if($this->linha5[2] == 1 && $this->linha5[3]=="Tarde"){ $this-
            >lbest = $this->lbest - 100; }
      }

      // Determinados professores gostariam de dar aula em certos dias;
      $this->consulta6 = sqlsrv_query("SELECT d.id, c.nome, g.daraula,
      g.ndaraula FROM diaperiodo a, periodo b, dia c, turma d, professores
      e, disciplina f, preferencias g WHERE a.idperiodo = b.id AND a.iddia
      = c.id AND a.idturma = d.id AND d.idprofessor = e.id AND
      d.iddisciplina = f.id AND g.idprofdisc=d.id and a.posicao ='" .
      $idposicao . "'" , $this->dbi);
      while($this->linha6 = sqlsrv_fetch_array($this->consulta6)){
            if($this->linha6[1] != $this->linha6[2]){ $this->lbest = $this-
            >lbest - 100;
            }
            if($this->linha6[1] == $this->linha6[3]){ $this->lbest = $this-
            >lbest - 100;
            }
      }
      $this->consultafinal = sqlsrv_query("update posicao set lbest=" .
      $this->lbest . "where posicao='" . $idposicao . "'", $this->dbi);
}

function geragbest(){
      $this->apagaposicao = sqlsrv_query("TRUNCATE TABLE gbest", $this-
      >dbi);
      $this->consultaparticula = sqlsrv_query("SELECT max(`Pbest`) FROM
      `Particula`", $this->dbi);
      $this->particulagbestid = sqlsrv_result( $this->consultaparticula, 0,
      0 );
      $this->consultaparticulaid = sqlsrv_query("SELECT * FROM `Particula`
      where Pbest=".$this->particulagbestid, $this->dbi);
      $this->particulagbest = sqlsrv_result( $this->consultaparticulaid, 0,
      0 );
      $this->posicaogbest = sqlsrv_result( $this->consultaparticulaid, 0, 1
      );
      $this->sqlinsertgbest = sqlsrv_query("insert into gbest values('',
      '".$this->particulagbest."', '".$this->posicaogbest."')");
      }

      function limpar(){
            $this->apagaposicao = sqlsrv_query("TRUNCATE TABLE gbest",
            $this->dbi);
            $this->apagadiaperiodo = sqlsrv_query("TRUNCATE TABLE
            diaperiodo", $this->dbi);
            $this->apagaposicao = sqlsrv_query("TRUNCATE TABLE posicao",
            $this->dbi);
            $this->apagaposicao = sqlsrv_query("TRUNCATE TABLE Particula",
            $this->dbi);
      }

      function microtime_float(){
            list($this->usec, $this->sec) = explode(" ", microtime());
            return ((float)$this->usec + (float)$this->sec);
      }

      function PSO(){
            $this->time_start = $this->microtime_float();
            while(true){
                  while ($this->buscaposicao() > 9 ){
                        $this->posicao = $this->geradiaperiodo();
                        $this->sqlinsertparticula = "insert into Particula
                        values('', '".$this->linha[0]."', '".$this-
                        >linha[2]."')";
                        $this->insertparticula = mysql_query($this-
                        >sqlinsertparticula , $this->dbi);
                  }
                  $this->fitness($this->posicao);
                  $this->geragbest();
```

```
                    if ($this->consultaposicao = mysql_query("SELECT * FROM
                    posicao where lbest >=700")){
                        break;
                    }
                }
                $this->time_end = $this->microtime_float();
                $this->timem = $this->time_end - $this->time_start;
                return $this->timem;
        }
}
?>
```

Apêndice B – Arquivo *index*.php

```
<?php
require "class.php";
$db = new sqlsrv;
$db->open();

if (!empty($_POST['add'])) {
      $professdor=$_POST['professor'];
      $disciplina=$_POST['disciplina'];
      if ($disciplina != "" && $professdor!=""){
            $sql="select * from turma where idprofessor='$professdor' and
            iddisciplina='$disciplina'";
            $db->query($sql);
            if ($db->linhas() == 0){
                  $insert = "insert into turma values ('',
            '".$professdor."', '".$disciplina."')";
            $db->query($insert);
      }else{
            echo '<script>alert("Registro duplicado.");</script>';
      }
      }else{
            echo '<script>alert("Selecione um professor e uma
            disciplina.");</script>';
      }
}

if (!empty($_POST['add2'])) {
      $periodo=$_POST['periodo'];
      $disciplina=$_POST['disciplina2'];
      if ($disciplina != "" && $periodo!=""){
            $sql="select * from disciplinaperiodo where
            idperiodo='$periodo' and iddisciplina='$disciplina'";
            $db->query($sql);
            if ($db->linhas() == 0){
                  $insert = "insert into disciplinaperiodo values ('',
                  '".$periodo."', '".$disciplina."')";
                  $db->query($insert);
            }else{
                  echo '<script>alert("Registro duplicado.");</script>';
            }
      }else{
            echo '<script>alert("Selecione um periodo e uma
            disciplina.");</script>';
      }
}

if (!empty($_POST['delete'])) {
      if($_POST['turma'] != ""){
            $sql = "delete from turma where id=" . $_POST['turma'];
            $db->query($sql);
      }else{
            echo '<script>alert("Selecione um registro para
            deletar.");</script>';
      }
}

if (!empty($_POST['delete2'])) {
      if($_POST['disciplinaperiodo'] != ""){
            $sql = "delete from disciplinaperiodo where id=" .
            $_POST['disciplinaperiodo'];
            $db->query($sql);
      }else{
            echo '<script>alert("Selecione um registro para
            deletar.");</script>';
      }
}

if (!empty($_POST['gerar'])) {
```

```
	$tempo = $db->PSO();
	echo '<script>alert("O tempo para gerar o calendário foi de:
	'.$tempo.' Segundos");</script>';
}

if (!empty($_POST['limpar'])) {
	$db->limpar();
}

?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<h2><center>PSO-Timetabling Generator</center></h2>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<!-- Latest compiled and minified CSS -->
<link rel="stylesheet"href=
"//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css">

<!-- Optional theme -->
<link rel="stylesheet" href=
"//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap-theme.min.css">

<!-- Latest compiled and minified JavaScript -->
<script src=
"//netdna.bootstrapcdn.com/bootstrap/3.0.2/js/bootstrap.min.js"></script>
<title>Untitled Document</title>
<style type="text/css">

.row {
	margin-right: 0x;
	margin-left: 0px;
}
.grid{
	background: rgb(195, 229, 245);
	border: 1px solid #cccccc;
}
</style>


</head>
<body>

<form action="" method="post" name="form1" id="form1">
<div class="row">
<div class="col-md-3">
<h3>Disciplinas:</h3>
	<select name="disciplina" size="15">
		<?php
			$sql = "select * from disciplina";
			$db->query($sql);
			for ($i = 0; $i < $db->linhas(); $i++) {
				echo "<option value=" . $db->result($i, 0) . ">".
				$db->result($i, 1) ."</option>" ;
			}
		?>
	</select>
</div>
<div class="col-md-2">
<h3>Professores:</h3>
	<select name="professor" size="15">
		<?php
			$sql = "select * from professores";
			$db->query($sql);
			for ($i = 0; $i < $db->linhas(); $i++) {
				echo "<option value=" . $db->result($i, 0) . ">".
				$db->result($i, 1) ."</option>" ;
```

```
                }
            ?>
        </select>
</div>
<div class="col-md-1"><br /><br /><br /><br /><br /> <input
name="add" type="submit" value="&gt;&gt;" id="add" class="btn btn-primary">
</div>
<div class="col-md-5">
<h3>Turmas:</h3>
        <select name="turma" size="15">
            <?php
                $sql = "select * from turma order by idprofessor";
                $db->query($sql);
                for ($i = 0; $i < $db->linhas(); $i++) {
                    $idturma = $db->result($i, 0);
                    $nomeprofessor = $db->getprofessor($db->result($i,
                    1));
                    $nomedisciplina = $db->getdisciplina($db-
                    >result($i, 2));
                    echo "<option value=" . $idturma . ">".
                    $nomeprofessor ." -- [".
                    $nomedisciplina ."]</option>" ;
                }
            ?>
        </select>
<input name="delete" type="submit" value="Deletar Registro" id="delete"
class="btn btn-danger">
</div>
</div>
</form>


<form action="" method="post" name="form2" id="form2">
<div class="row">
<div class="col-md-3">
<h3>Disciplinas:</h3>
        <select name="disciplina2" size="15">
            <?php
                $sql = "select * from disciplina";
                $db->query($sql);
                for ($i = 0; $i < $db->linhas(); $i++) {
                    echo "<option value=" . $db->result($i, 0) . ">".
                    $db->result($i, 1) ."</option>" ;
                }
            ?>

        </select>
</div>
<div class="col-md-2">
<h3>Períodos:</h3>
        <select name="periodo" size="15">
            <?php
                $sql = "select * from periodo";
                $db->query($sql);
                for ($i = 0; $i < $db->linhas(); $i++) {
                    echo "<option value=" . $db->result($i, 0) . ">".
                    $db->result($i, 1) ."</option>" ;
                }
            ?>
        </select>
</div>
<div class="col-md-1"><br /><br /><br /><br /><br /><input
name="add2" type="submit" value="&gt;&gt;" id="add2" class="btn btn-
primary"></div>
<div class="col-md-5">
<h3>Disciplinas por períodos:</h3>
        <select name="disciplinaperiodo" size="15">

            <?php
```

```
                  $sql = "select * from disciplinaperiodo order by
                  idperiodo";
                  $db->query($sql);
                  for ($i = 0; $i < $db->linhas(); $i++) {
                        $idturma = $db->result($i, 0);
                        $nomeperiodo = $db->getperiodo($db->result($i, 1));
                        $nomedisciplina = $db->getdisciplina($db-
                        >result($i, 2));
                        echo "<option value=" . $idturma . ">".
                        $nomeperiodo ." -- [". $nomedisciplina
                        ."]</option>" ;
                  }
            ?>
      </select>
<input name="delete2" type="submit" value="Deletar Registro" id="delete2"
class="btn btn-danger">
</div>
</form>
<div class="clearfix visible-xs"></div>
<div class="row" >
<p><br />
      <span class="col-md-5">
      <input name="gerar" type="submit" value="Gerar" id="gerar" class="btn
      btn-danger" />
</span><span class="col-md-5">
<input name="limpar" type="submit" value="Limpar" id="limpar" class="btn
btn-danger" />
</span></p>
<p><br />
</p></div>

</p>
      <div>
      <div class="row">
      <div class="col-md-1"></div>
      <div class="col-md-1 grid"><b>Segunda-feira</b></div>
      <div class="col-md-1 grid"><b>Terça-feira</b></div>
      <div class="col-md-1 grid"><b>Quarta-feira</b></div>
      <div class="col-md-1 grid"><b>Quinta-feira</b></div>
      <div class="col-md-1 grid"><b>Sexta-feira</b></div>
      <div class="col-md-1 grid"><b>Sábado</b></div>
</div>


<?php
      $sql = "select * from periodo order by nome";
      $db->query($sql);
      for ($i = 0; $i < $db->linhas(); $i++) {
            $idperiodo = $db->result($i, 0);
            $nomeperiodo = $db->result($i, 1);
                  $sql2 = "select b.nome as Periodo, c.nome as Dia, c.hora
                  as Hora, a.idturma as Turma, e.nome as Professor, f.nome
                  as Disciplina from diaperiodo a, periodo b, dia c, turma
                  d, professores e, disciplina f, posicao g, gbest h where
                  a.idperiodo=b.id and a.iddia=c.id and a.idturma=d.id and
                  d.idprofessor=e.id and d.iddisciplina=f.id and
                  a.posicao=g.posicao and h.idposicao=g.id and
                  a.idperiodo=" . $idperiodo . " order by c.hora";
                  $dados=sqlsrv_query($sql2);
                  $dados2=sqlsrv_query($sql2);
                  $dados3=sqlsrv_query($sql2);
                  $dados4=sqlsrv_query($sql2);
                  $dados5=sqlsrv_query($sql2);
                  $dados6=sqlsrv_query($sql2);

                  echo '<div class="row">';
                  echo ' <div class="col-md-1
                  grid"><b>'.$nomeperiodo.'</b></div>
```

```
                        <div class="col-md-1 grid">'; while ($row =
                        sqlsrv_fetch_array($dados)){if($row[1] == "segunda-
                        feira"){echo "<b>". $row[2] ." - Turma:" .$row[3].
                        "</b> - " . $row[5] . " - Prof:". $row[4]
                        .'<br><br>';}}echo '</div>
                        <div class="col-md-1 grid">'; while ($row2 =
                        sqlsrv_fetch_array($dados2)){if($row2[1] == "terça-
                        feira"){echo "<b>". $row2[2] ." -Turma:" .
                        $row2[3]. "</b> - " . $row2[5] . " - Prof:".
                        $row2[4] .'<br><br>';}} echo '</div><div
                        class="col-md-1 grid">'; while ($row3 =
                        sqlsrv_fetch_array($dados3)){if($row3[1] ==
                        "quarta-feira"){echo "<b>". $row3[2] ." -Turma:" .
                        $row3[3]. "</b> - " . $row3[5] . " - Prof:".
                        row3[4] .'<br><br>';}} echo '</div><div class="col-
                        md-1 grid">'; while ($row4 =
                        sqlsrv_fetch_array($dados4)){if($row4[1] == quinta-
                        feira"){echo "<b>". $row4[2] ." -Turma:" .
                        $row4[3]. "</b> - " . $row4[5] . " - Prof:".
                        $row4[4] .'<br><br>';}} echo '</div><div
                        class="col-md-1 grid">'; while ($row5 =
                        sqlsrv_fetch_array($dados5)){if($row5[1] == "sexta-
                        feira"){echo "<b>". $row5[2] ." -Turma:" .
                        $row5[3]. "</b> - " . $row5[5] . " - Prof:".
                        $row5[4] .'<br><br>';}} echo '</div><div
                        class="col-md-1 grid">'; while ($row6 =
                        sqlsrv_fetch_array($dados6)){if($row6[1] ==
                        "sabado"){echo "<b>". $row6[2] ." - Turma:".
                        $row6[3]. "</b> - " . $row6[5] . " - Prof:".
                        $row6[4] . '<br><br>';}} echo '</div>';
                    echo "</div>";
                }
            ?>
    </div>

    </body>
    </html>
```