# Tópicos Especiais em Otimização I
# 2023/2
# Atividade 2: Otimização por Enxame de Partículas aplicada ao problema de Tabela-Horário

**Filipe Gomes Arante de Souza**[1]

[1]Departamento de Informática – Universidade Federal do Espírito Santo (UFES)

filipe.ga.souza@edu.ufes.br

***Abstract.*** *This article studies the Time Table Problem in the ITC-2007 formulation, a challenge of great relevance to educational institutions. The paper proposes a solution using a new approach to the Particle Swarm Optimization metaheuristic, with the aim of improving the distribution of subjects, rooms and teachers over time. Unfortunately, the algorithm failed to improve the results obtained at the beginning and converged to values higher than those contained in the initial population.*
*Keywords: Time Table, Particle Swarm, PSO, Optimization, GRASP*

***Resumo.*** *Este artigo estuda o Problema de Tabela Horário na formulação do ITC-2007, um desafio de grande relevância para instituições educacionais. O trabalho propõe uma solução utilizando uma nova abordagem da metaheurística de Otimização por Enxame de Partículas, visando aperfeiçoar a distribuição de disciplinas, salas e professores ao longo do tempo. Infelizmente, o algoritmo não conseguiu melhorar os resultados obtidos no começo e convergiu para valores acima dos que a população inicial continha.*
*Palavras-chave: Tabela Horário, Enxame de Partículas, PSO, Otimização, GRASP*

## 1. Introdução

A Otimização por enxame de partículas (PSO) é um algoritmo evolutivo que surgiu dos estudos oriundos de simulações computacionais simplificadas de um bando de pássaros. Inicialmente, [Kennedy and Eberhart 1995] tinham interesse em estudar a movimentação de populações desses animais. À medida que sua pesquisa foi avançando, esse problema se tornou um problema de otimização e teve sua primeira proposta de algoritmo desenvolvida. Ao longo do tempo, o PSO adquiriu diversas adaptações e incrementos, que são utilizados até hoje em muita áreas, como telecomunicações, controle, mineração de dados, design, otimização combinatória, sistemas de energia, processamento de sinais, dentre outras.

Uma das classes de problemas esse algoritmo é capaz de atuar é a de escalonamento, podendo ser aplicado em elaboração de escalas de trabalho, agendamento de partidas para campeonatos e horários escolares. Assim sendo, este artigo tem a finalidade de apresentar resumidamente a versão original do PSO e mostrar sua aplicação na resolução do problema de Tabela Horário proposta pelo ITC-2007.

O documento é organizado da seguinte forma: A seção 2 aborda o conceito básico da meta-heurística em questão, enquanto as seções seguintes explicam o PTHU, apresentam a adaptação do algoritmo para este problema e sintetizam os resultados obtidos.

## 2. Versão original do PSO

O princípio que esta meta-heurística utiliza é buscar a solução ótima num determinado espaço de busca através da troca de informações entre indivíduos de uma população determinando qual trajetória cada membro dessa população deve tomar no espaço. Cada partícula possui memória de sua melhor posição percorrida e o bando inteiro conhece a melhor posição do espaço percorrida por todo o enxame. Assim sendo, o movimento de cada partícula é orquestrado de acordo com as equações abaixo: [Freitas et al. 2020]

$$v_{k+1} = w \cdot v_k + c_1 \cdot r_1 \cdot (p_{best_k} - x_k) + c_2 \cdot r_2 \cdot (g_{best} - x_k)$$

$$x_{k+1} = x_k + v_k$$

Onde $v_k$ é a velocidade, $x_k$ é a posição, $w$ é o coeficiente inercial, $c_1$ é o coeficiente individual, $c_2$ é o coeficiente social, $r_1$ e $r_2$ são valores aleatórios entre 0 e 1, $p_{best_k}$ é a melhor posição já encontrada da partícula $k$ e $g_{best}$ é a melhor posição já encontrada em todo o enxame.

## 3. O problema de Tabela Horário do ITC-2007

O PHTU pode ser definido como a tarefa de alocar aulas de disciplinas num conjunto finito de horários e salas, respeitando determinadas restrições. O Problema de Tabela Horário de Universidades da formulação CB-CTT proposta no ITC-2007 consiste na alocação de aulas em um horário semanal, num dado número restrito de salas e períodos, onde conflitos são definidos conforme o currículo.

Confira a definição do problema na seção 2 da Tese de Doutorado do professor [Kampke 2020] **clicando aqui**.

## 4. Representação das soluções

Numa instância do problema, sempre temos $D$ dias, $P$ períodos por dia e $R$ salas disponíveis e várias aulas para serem alocadas, cada uma com suas próprias restrições.

Portanto, de forma simplificada, cada uma das soluções presentes na população pode ser visto como um array de tuplas de quatro elementos que representam slots de aulas a serem alocadas. Elas contém o dia, periodo, sala e curso alocado. Formalmente falando, podemos representar uma partícula como uma matriz do seguinte tipo:

$$Particle = \begin{bmatrix} d_1 & p_1 & r_1 & c_1 \\ ... & ... & ... & ... \\ d_i & p_j & r_k & c_l \\ ... & ... & ... & ... \\ d_D & p_P & r_R & c_n \end{bmatrix}$$

Onde:

- $n = D \cdot P \cdot R$, quantificando o total de slots;
- $d_i$ representa o $i$-ésimo dia;
- $p_j$ representa o $j$-ésimo período;
- $r_k$ representa a $k$-ésima sala;
- $c_l$ representa o curso alocado no $l$-ésimo slot;

Entretanto, para tratar os conflitos entre as aulas, checar viabilidade da solução, calcular função objetivo, dentre outras tarefas, foi feita uma modelagem em classes. Confira-a na subseção a seguir.

### 4.1. Modelagem

Todas as entidades do problema foram separadas em várias classes, visando aproveitar o relacionamento entre elas para conseguir fazer as operações desejadas. Segue abaixo as classes definidas e seus respectivos papeis:

- **Instance**: Armazenar informações gerais da instância a ser executada, bem como gerar estruturas auxiliares para efetuar consultas de forma otimizada a respeito de conflito de cursos;
- **Constraint**: Armazenar um dia e período que um curso não pode ser alocado;
- **Room**: Armazenar o nome e capacidade de uma sala;

- **Course**: Armazenar informações referentes a um curso e prover métodos de verificação de conflito entre cursos, seja de professor ou de currículo.

- **Curricula**: Armazenar os cursos pertencentes a um mesmo currículo;

- **Slot**: Representa um terno de dia, período e sala. Possui métodos para alocar e desalocar cursos de um slot;

- **TimeTable**: Representa a tabela horário em si, possuindo métodos para alocar cursos em slots de forma a manter a viabilidade da solução. Além disso, também é responsável por checar as restrições fortes calcular as restrições fracas;

- **Particle**: É uma classe filha de TimeTable. Contém os métodos dos movimentos do PSO e geração de solução inicial;

- **BestParticle**: Armazena a cópia das informações mais importantes de uma partícula, como por exemplo o estado da tabela horário num dado momento e o valor de sua função objetivo. Serve para guardar os melhores locais e global do PSO;

- **Swarm**: Gerencia uma população de partículas, atualizando suas posições no espaço e mantendo os melhores locais e global atualizados;

- **PSO**: Determina os parâmetros do algoritmo e é encarregada de rodar o algoritmo, controlando o tempo de execução e gerando dados da execução em arquivos;

- **Report**: Faz o cálculo das métricas de avaliação de uma execução do PSO e gera relatórioscom estes dados;

Veja a ilustração deste modelo na Figura 1 logo abaixo:

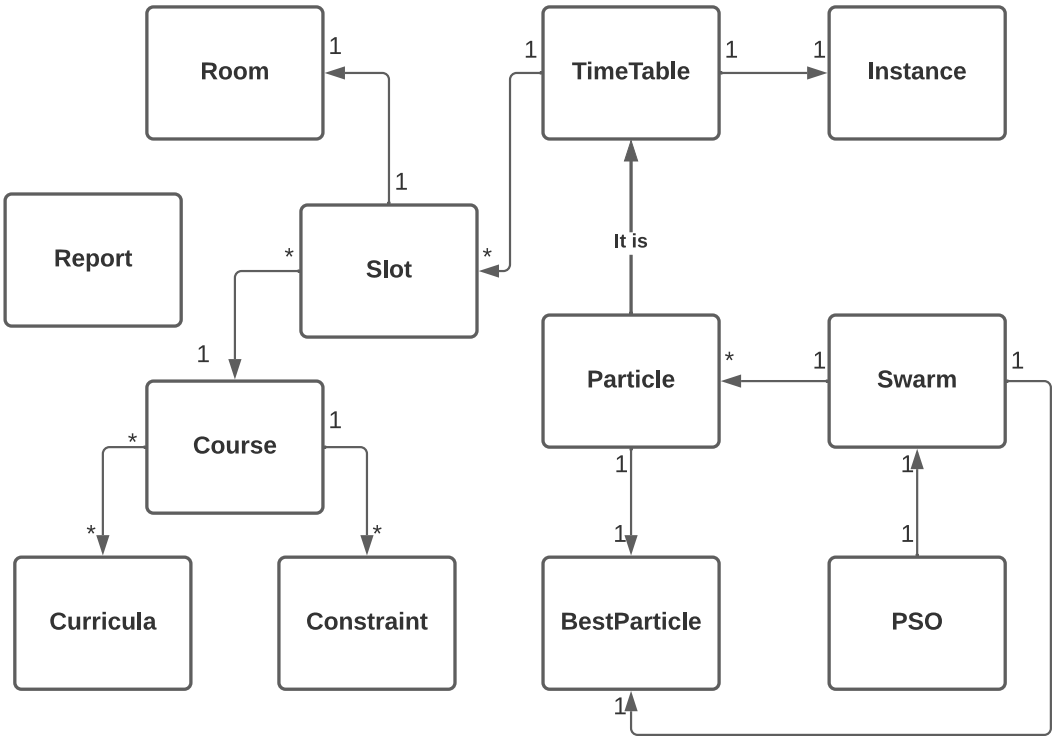



**Figure 1. Modelagem de classes para implementar solução do PHTU.**

## 5. PSO para o PHTU

Sabemos que a Otimização por Enxame de Partículas foi originalmente projetada para lidar com otimização contínua. Portanto, foi necessário adaptar a meta-heurística pra uma versão combinatória.

### 5.1. Geração da população inicial

Para gerar a população inicial, precisamos de alguma heurística com características gulosa e aleatória, pois queremos iniciar com partículas distintas entre si e com certa qualidade. Uma estratégia que deu muito certo foi construir cada solução utilizando o construtor do GRASP, utilizado por [de Almeida Segatto 2017] em sua dissertação.

Foi feita uma adaptação para o algoritmo gerar apenas soluções viáveis. Caso ele não consiga, é retornado um conjunto de slots vazio. Segue abaixo pseudocódigo no algoritmo 1:

**Algorithm 1** GRASP Constructor

**function** GRASP_CONSTRUCTOR($A$, $B$, $\alpha$)
    ▷ $A$ is the set of all classes to alloc
    ▷ $B$ is the set of all slots in timetable
    ▷ $\alpha$ is a real number between 0 and 1

    $C \leftarrow A$
    $S \leftarrow B$

    **while** $C \neq \emptyset$ **do**
        Get the most conflitant class $c' \in C$
        $\forall$ available $s \in S$ compute $g(c', s)$

        $c_{min} \leftarrow min\{g(c', s) : s \in S\}$
        $c_{max} \leftarrow max\{g(c', s) : s \in S\}$

        $LRC \leftarrow \{s \in S : g(c', s) \leq c_{min} + \alpha \cdot (c_{max} - c_{min})$ and $g(c', s)$ is finite $\}$

        **if** $LRC = \emptyset$ **then**
            $S \leftarrow \emptyset$
            **return** $S$ ▷ Can't find feasible solution
        **end if**

        **for** $\forall i \in [1, 10]$ **do** ▷ Try 10 times find a valid slot to alloc $c'$
            Select a random slot $s'$ from $LRC$

            **if** is possible alloc $c'$ in $s'$ **then**
                Alloc course $c'$ in slot $s'$
                **break**
            **end if**

            **if** $i = 10$ **then** ▷ Last iteration
                $S \leftarrow \emptyset$
                **return** $S$ ▷ Can't find feasible solution
            **end if**
        **end for**

        $C \leftarrow C - \{c'\}$
        $S \leftarrow S - \{s'\}$
    **end while**

    **return** $S$ ▷ A feasible solution was found
**end function**

A aula mais conflitante é a que possui menos horários disponíveis na tabela horário preenchida. Caso haja empate, a aula presente em mais currículos é escolhida. Caso ainda

tenha empate nesse quesito, a aula escolhida é sorteada.

Já a função $g$, chamada de função gulosa, foi definida como o valor da função objetivo considerando apenas as restrições fracas da tabela horário, que está preenchida parcialmente durante a execução do algoritmo.

Por fim, tendo o construtor do GRASP em mãos, é trivial inicializar a população inicial, como mostra o algoritmo 2:

**Algorithm 2** Population Constructor

```
1:  function POPULATION_CONSTRUCTOR(A, B, α, N)
2:      ▷ A is the set of all classes to alloc
3:      ▷ B is the set of all slots in timetable
4:      ▷ α is a real number between 0 and 1
5:      ▷ N is the size of population
6:
7:      P ← ∅
8:
9:      for ∀i ∈ [1, N] do
10:         p ← GRASP_CONSTRUCTOR(A, B, α)
11:
12:         if p ≠ ∅ then
13:             P ← P ∪ {p}
14:         end if
15:     end for
16:
17:     return P
18: end function
```

Como o construtor do GRASP não garante retornar uma solução, pode acontecer da população vir com tamanho menor que $N$. Há o caso mais extremo de $p$ ser vazio em todas as iterações, mas as $10$ tentativas feitas no construtor do GRASP foram suficientes para contornar este caso nesse trabalho.

### 5.2. Operadores de modificação de soluções

Os operadores do PSO foram inspirados no trabalho de [Chu et al. 2006]. Eles definem os movimentos Rand Mutate e Rand Change, que consistem muma forma de se aproximar do melhor local e do melhor global a cada iteração através de moves e swaps estratégicos na tabela horário. Para este artigo, foi desenvolvida uma adaptação deles, entituladas como $W$ Rand Mutate e $C$ Rand Change.

#### 5.2.1. W Rand Mutate

Consiste em fazer $W$ mutações aleatórias numa partícula. Assim sendo, dois slots são sorteados para terem seus cursos alocados trocados. Se um dos slots estiver vazio, ocorrerá um move. Se os dois estiverem vazios, não ocorrerá modificação na tabela horário.

Cada um dos swaps/moves são aceitos apenas caso a solução continue viável, sendo feitas um máximo de 10 tentativas. Se todas elas falharem, o $i$-ésimo movimento do Rand Mutate é cancelado, onde $i \in [1, W]$.

Note que o parâmetro W serve para promover a diversificação do algoritmo. Veja o pseudocódigo dessa operação no algoritmo 3:

**Algorithm 3** W Rand Mutate

```
1:  function RAND_MUTATE(X, W)
2:      ▷ X is a particle
3:      ▷ W is a positive integer
4:
5:      for ∀i ∈ [1, W] do
6:          for ∀j ∈ [1, 10] do ▷ Try 10 times a successful swap
7:              Select two random slots S1 and S2 from X
8:              Swap the allocated courses in S1 and S2
9:
10:             if X is feasible then
11:                 break
12:             else
13:                 Undo the swap of allocated courses in S1 and S2
14:             end if
15:         end for
16:     end for
17: end function
```

### 5.2.2. C Rand Change

Consiste em repetir $C$ vezes o processo descrito a seguir. Considere uma partícula $P_1$ de referência e uma partícula $P_2$ a ser modificada. Um slot $S_1$ de $P_1$ que possui uma aula $A_1$ é sorteado, e a ideia é colocar a matéria $A_1$ no respectivo slot de $P_2$. Portanto, é feito um swap em $P_2$ com algum de seus slots que contenham a aula $A_1$. Assim como no $W$ Rand Mutate, se o slot sorteado estiver vazio, ocorrerá um move.

Cada um dos swaps/moves são aceitos apenas caso a solução continue viável, sendo feitas um máximo de 10 tentativas. Se todas elas falharem, o $i$-ésimo movimento do Rand Change é cancelado, onde $i \in [1, C]$.

No contexto do PSO, o $C$ Rand Change é utilizado para aproximar uma partícula $X_k$ do enxame ao seu melhor local $P_k$ e ao melhor global do bando $G$. Veja o pseudocódigo dessa operação no algoritmo 4:

**Algorithm 4** C Rand Change

```
1: function RAND_CHANGE(X, R, C)
2:     ▷ X is a particle to be modified
3:     ▷ R is a reference particle
4:     ▷ C is a positive integer
5:
6:     for ∀i ∈ [1, C] do
7:         for ∀j ∈ [1, 10] do ▷ Try 10 times a successful swap
8:             Select a random slot S_R from R
9:             Get the respective slot S_X of S_R in X
10:            Find some slot S'_X from X that contains the same allocated course of S_R
11:            Swap the allocated courses in S_X and S'_X
12:
13:            if X is feasible then
14:                break
15:            else
16:                Undo the swap of allocated courses in S_x and S'_x
17:            end if
18:        end for
19:    end for
20: end function
```

### 5.2.3. Equação do movimento do PSO adaptada

Nessa adaptação do PSO, temos um novo conjunto de equações que gerenciam as partículas do PSO:

$$S_k^{i+1} = rand\_mutate(X_k^i, W)$$
$$W_k^{i+1} = rand\_change(S_k^{i+1}, P_k^i, C_1)$$
$$X_k^{i+1} = rand\_change(W_k^{i+1}, G^i, C_2)$$

Onde:

- $W$, $C_1$ e $C_2$ são os parâmetros do PSO que indicam quantos swaps/moves serão feitos de acordo com o movimento que são aplicados;
- $X_k^i$ é a partícula $k$ na iteração $i$;
- $P_k^i$ é o melhor local da partícula $k$ na iteração $i$;
- $G^i$ é o melhor global do enxame na iteração $i$;
- $S_k^{i+1}$ e $W_k^{i+1}$ são os estados intermediários da partícula após o rand_mutate e rand_change;
- $X_k^{i+1}$ é a partícula $k$ na iteração $i+1$;

## 6. Implementação

O algoritmo foi testado num conjunto de 21 instâncias do ITC-2007, nomeadas de $comp01$ a $comp21$. Cada uma delas foi executada 5 vezes para coletar algumas métricas que serão

mostradas na próxima subseção. Por questões de tempo, as instâncias foram executadas em três computadores:

- $comp01 - 05$ e $comp16 - 21$: Computador com benchmark de $204$ segundos;
- $comp06 - 10$: Computador com benchmark de $228$ segundos;
- $comp11 - 15$: Computador com benchmark de $216$ segundos;

Os parâmetros julgados como melhores pelo autor para o PSO foram os seguintes:

$$W = 1, C_1 = 1, C_2 = 1$$

Devido ao tempo muito grande de execução, definiu-se o tamanho da população com **5 partículas**.

### 6.1. Resultados computacionais

Na tabela 1 encontram-se as métricas da execução das instâncias.

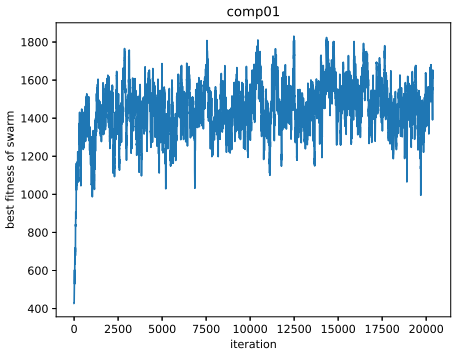
**Observação importante**: A restrição fraca de aulas isoladas foi calculada considerando apenas aulas únicas sem adjacentes do mesmo currículo, o que diminuiu um pouco os valores obtidos em geral. Portanto, considere que os resultados deveriam ser um pouco maiores, e que as linhas que superaram o ótimo, provavelmente não superaram, infelizmente.

| | Ótimo | $f_{PSO}$ | | Gaps | |
|---|---|---|---|---|---|
| Instância | $f_{otimo}$ | $Min_5$ | $Media_5$ | $gap_{Min}$ | $gap_{Media}$ |
| comp01 | 5 | 252 | 330.80 | 49.40 | 65.16 |
| comp02 | 24 | 257 | 291.80 | 9.71 | 11.16 |
| comp03 | 64 | 285 | 295.40 | 3.45 | 3.62 |
| comp04 | 35 | 351 | 369.40 | 9.03 | 9.55 |
| comp05 | 284 | 219 | 301.60 | -0.23 | 0.06 |
| comp06 | 27 | 668 | 729.00 | 23.74 | 26.00 |
| comp07 | 6 | 771 | 931.80 | 127.50 | 154.30 |
| comp08 | 37 | 294 | 326.00 | 6.95 | 7.81 |
| comp09 | 96 | 216 | 224.00 | 1.25 | 1.33 |
| comp10 | 4 | 354 | 394.40 | 87.50 | 97.60 |
| comp11 | 0 | 150 | 268.00 | $\infty$ | $\infty$ |
| comp12 | 294 | 145 | 195.00 | -0.51 | -0.37 |
| comp13 | 59 | 350 | 384.60 | 4.93 | 5.52 |
| comp14 | 51 | 254 | 257.60 | 3.98 | 4.05 |
| comp15 | 62 | 285 | 305.20 | 3.60 | 3.92 |
| comp16 | 18 | 373 | 435.80 | 19.72 | 23.21 |
| comp17 | 56 | 379 | 474.20 | 5.77 | 7.47 |
| comp18 | 61 | 42 | 52.20 | -0.31 | -0.14 |
| comp19 | 57 | 273 | 283.8 | 3.79 | 3.98 |
| comp20 | 4 | 758 | 846.40 | 188.50 | 210.60 |
| comp21 | 74 | 277 | 289.80 | 2.74 | 2.92 |

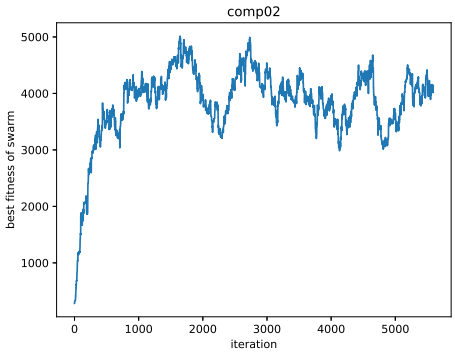
**Table 1. Resultados obtidos pelo PSO nas instâncias disponibilizadas.**

### 6.2. Melhor partícula do bando vs iteração do PSO

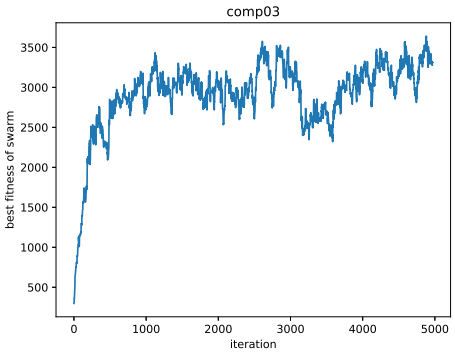
Para cada instância, veja o valor fitness da melhor partícula do bando ao longo das iterações na figura 2. Note que a meta-heurística não foi capaz de melhorar os resultados, sendo a melhor partícula da primeira geração a melhor de todas as execuções em **todos** os casos.



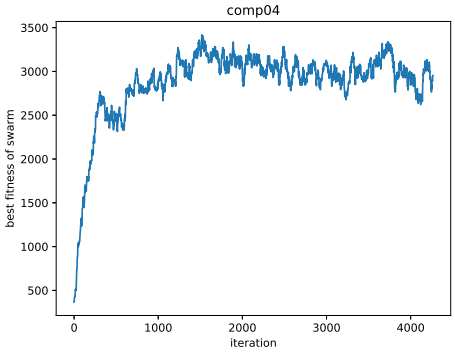

**(a)** Instância comp01



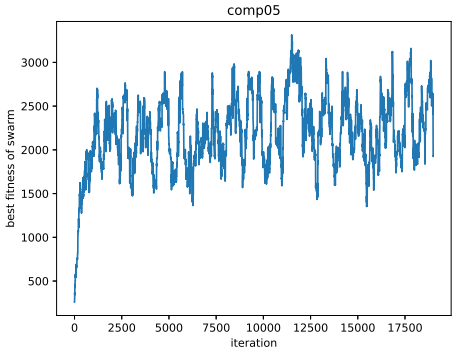

**(b)** Instância comp02



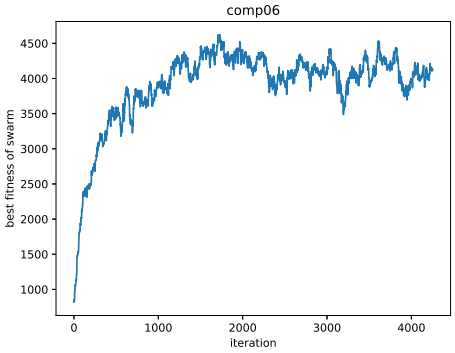

**(c)** Instância comp03



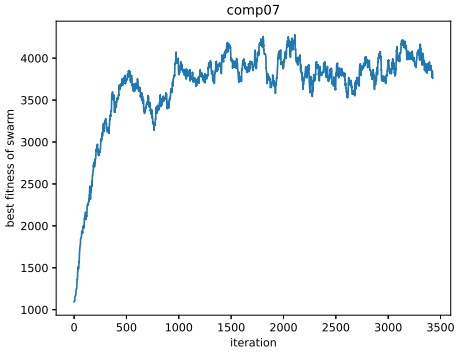

**(d)** Instância comp04



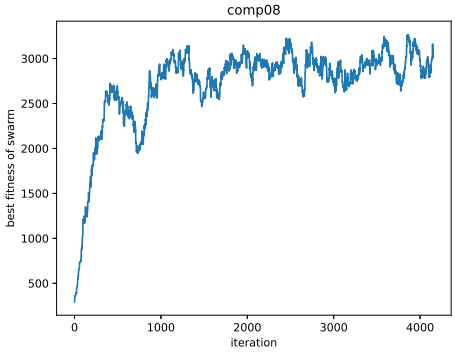

**(e)** Instância comp05



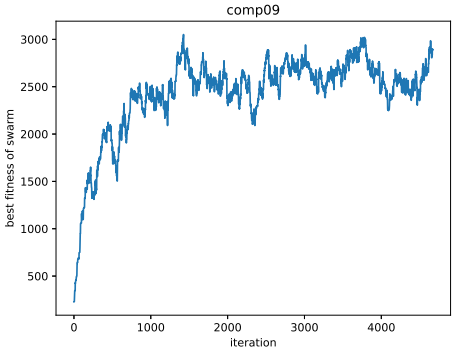

**(f)** Instância comp06



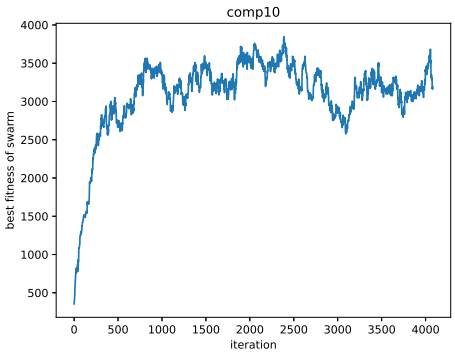

**(g)** Instância comp07



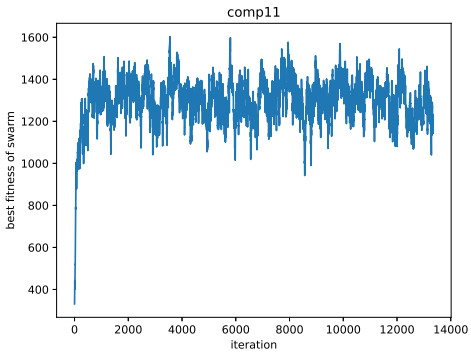

**(h)** Instância comp08



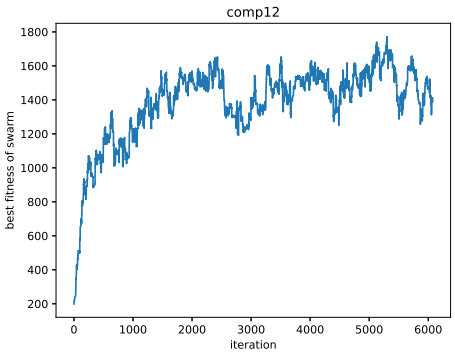

**(i)** Instância comp09



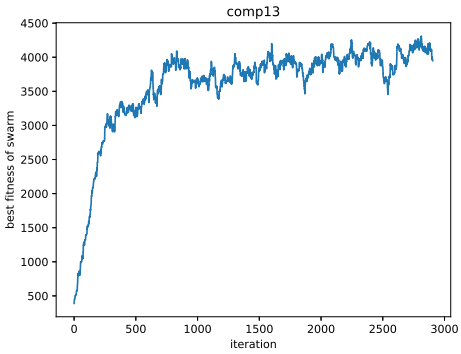

**(j)** Instância comp10



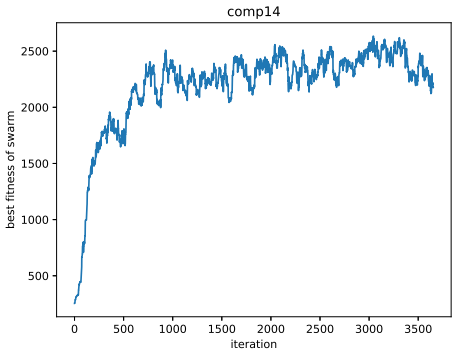

**(k)** Instância comp11



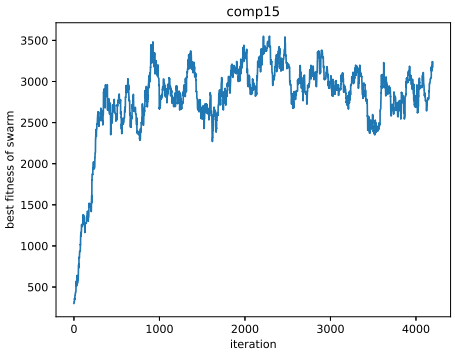

**(l)** Instância comp12



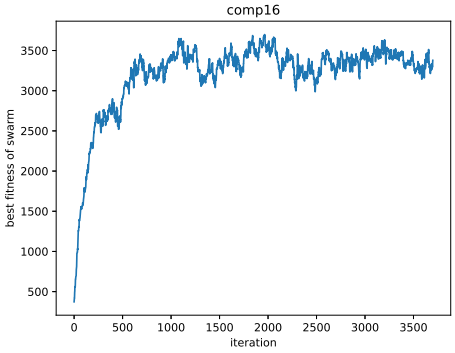

**(m)** Instância comp13



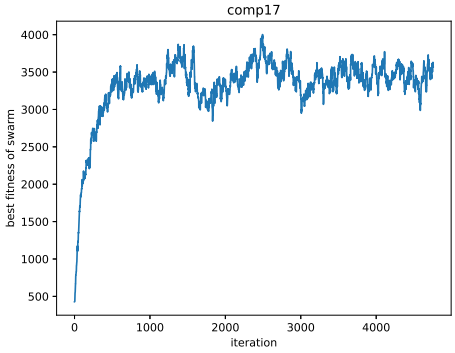

**(n)** Instância comp14



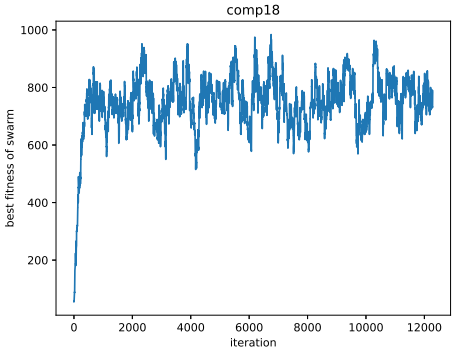

**(o)** Instância comp15



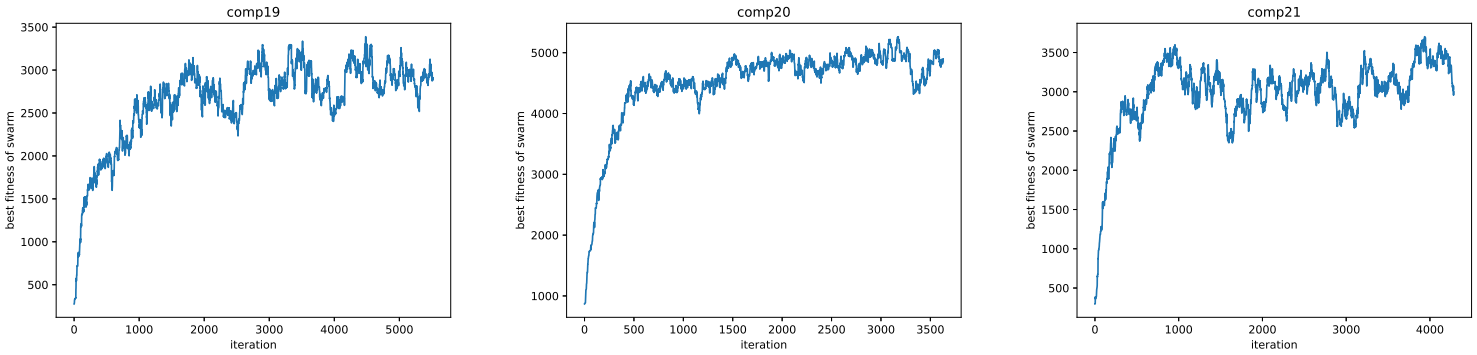

**(p)** Instância comp16




**(q)** Instância comp17




**(r)** Instância comp18




**(s)** Instância comp19 **(t)** Instância comp20 **(u)** Instância comp21

**Figure 2. Valor da função objetivo da melhor partícula do bando em função das iterações do PSO.**

## 7. Análise dos resultados

Os resultados finais foram muito bons, considerando o tempo de desenvolvimento deste trabalho. O algoritmo construtivo do GRASP funcionou muito bem para o Problema de Tabela Horário, encontrando soluções viáveis para todas as instâncias e já com valores relativamente baixos na maioria dos casos. As que foram mais difíceis de viabilizar foram a $comp05$ e $comp19$, que executaram com população de tamanho menor que $5$.

Por outro lado, o PSO não performou bem como se esperava. Para quaisquer valor de $W$, $C_1$ e $C_2$ testados os movimentos de W Rand Mutate e C Rand Change não conseguiam diminuir o valor da função objetivo. O valor do menor fitness do bando cresceu descontroladamente até convergir num determinado intervalo.

Um teste feito para tentar interromper esse fato foi somente aceitar soluções cujos valores após os movimentos fossem melhores que os anteriores. Porém, esta estratégia não deu certo pois não permite diversificação.

Talvez seja interessante hibridizar o algoritmo e criar algum critério de aceitação de soluções, como o Simulated Annealing faz. O PSO aceita qualquer posição que a partícula atinja, o que deve ter ocasionado o rápido incremento do fitness nas iterações.

Afinal, tendo em vista que os valores iniciaram baixos, é razoável acreditar que é mais provável um swap/move piorar a solução do que melhorar. Logo, esses tipos de movimentos não são os melhores para lidar com estes casos. Uma melhoria a ser estudada é como destruir e reconstruir a solução de modo que ela continue viável e se aproxime das partículas $P^k_{best}$ e $G_{best}$ ao longo do tempo.

Uma hipótese a ser levantada para o motivo do fracasso da meta-heurística é o fato de [Chu et al. 2006] trabalhar com uma formulação mais simples do Problema de Tabela Horário. A formulação mais rigorosa do ITC-2007 pode ter tirado o potencial da metodologia do PSO.

## 8. Conclusões

Vimos ao longo deste artigo uma adaptação do PSO para o PHTU, da área de otimização combinatória. Este problema é NP-Completo para a maioria de suas formulações, o que

prova ser um desafio resolvê-lo de forma eficaz e eficiente.

A modelagem em classes do problema facilitou muito o desenvolvimento, proporcionando o reúso de código, aumentando a velocidade de desenvolvimento do código fonte. Além disso, o uso de estruturas auxiliares para consultas de conflito entre cursos foi uma boa sacada para o algoritmo obter melhor performance.

Convém abordar que a geração de uma solução inicial necessariamente viável torna o processo de inicialização da população bastante demorado. Em vários casos esse tempo para uma população de tamanho $5$ foi maior que o tempo do benchmark. Por esse motivo, foi necessário "trapacear" na execução das instâncias, levando a considerar esse tempo apenas na execução da meta-heurística.

Além disso, o algoritmo construtivo do GRASP trouxe os melhores resultados gerais deste trabalho, o que é um indício desse algoritmo ser poderoso nessa área.

Todavia, a Otimização por Enxame de Partículas não foi capaz de diminuir os resultados obtidos pelo GRASP, elevando os resultados para todos os casos de teste. Portanto, são necessários ajustes para trabalhos futuros.

## References


Chu, S.-C., Chen, Y.-T., and Ho, J.-H. (2006). Timetable scheduling using particle swarm optimization. In *First International Conference on Innovative Computing, Information and Control - Volume I (ICICIC'06)*, volume 3, pages 324–327.

de Almeida Segatto, E. (2017). Um estudo de estruturas de vizinhanças no grasp aplicado ao problema de tabela-horário para universidadess. Mestrado em ciência da computação, Universidade Federal do Espírito Santo, Vitória - ES.

Freitas, D., Lopes, L. G., and Morgado-Dias, F. (2020). Particle swarm optimisation: A historical review up to the current developments. *Entropy*, 22(3).

Kampke, E. H. (2020). *Novas Estratégias para Obtenção de Limitantes Inferiores para o Problema de Tabela-Horário de Universidades*. Doutorado em informática, Universidade Federal do Espírito Santo, Vitória - ES.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.