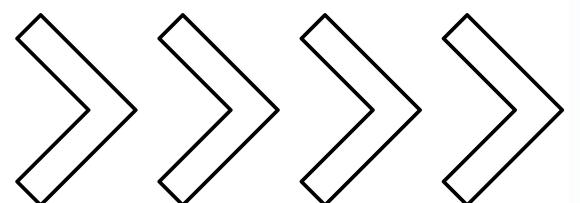


GRASP PARA SOLUÇÃO DO PROBLEMA DE RECOMENDAÇÃO DE MATRÍCULA

METAHEURÍSTICA



Aluno: Victor Kruze Fiori
Professor: Rafael Frinhani

Índice

01 Introdução
Grasp/Problema

02 Problema

03 Grasp

04 Experimentos e
Resultados obtidos

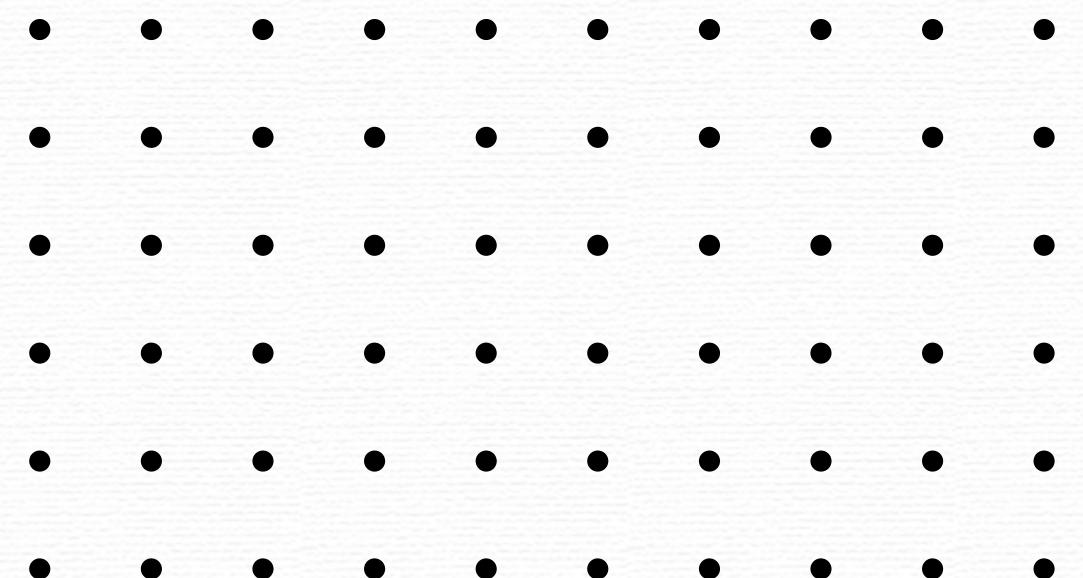
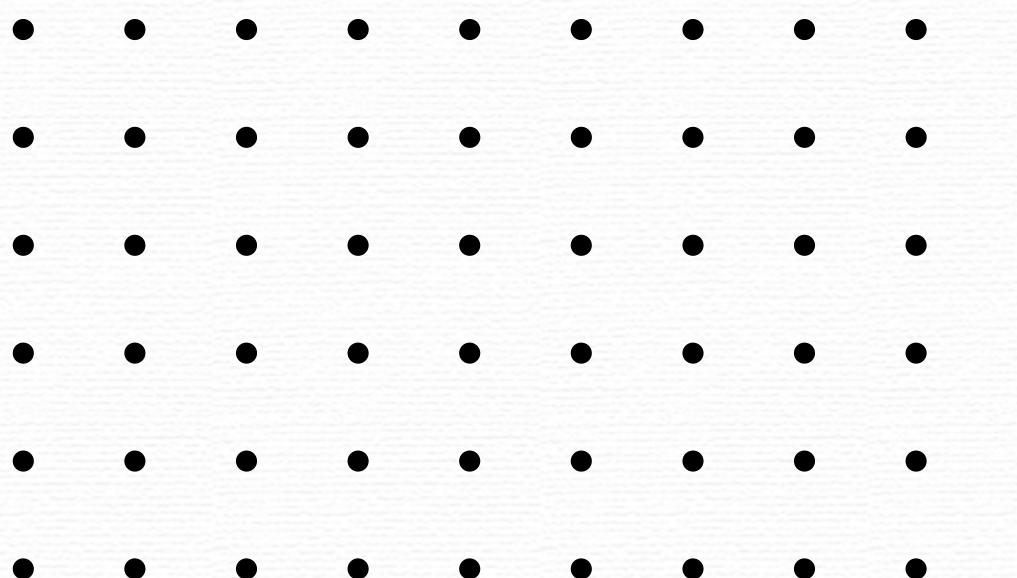
05 conclusões

06 Referências

Introdução

Em um curso superior, é de extrema importância otimizar o rendimento acadêmico do aluno. Para isso, é de suma importância uma matrícula semestral coerente e com o objetivo de acelerar o processo de formação sem prejudicar o discente.

Com esse problema de recomendação, metaheurísticas e algoritmos de resolução de problemas podem ser usados para achar a melhor solução para o problema de recomendação de matrículas.



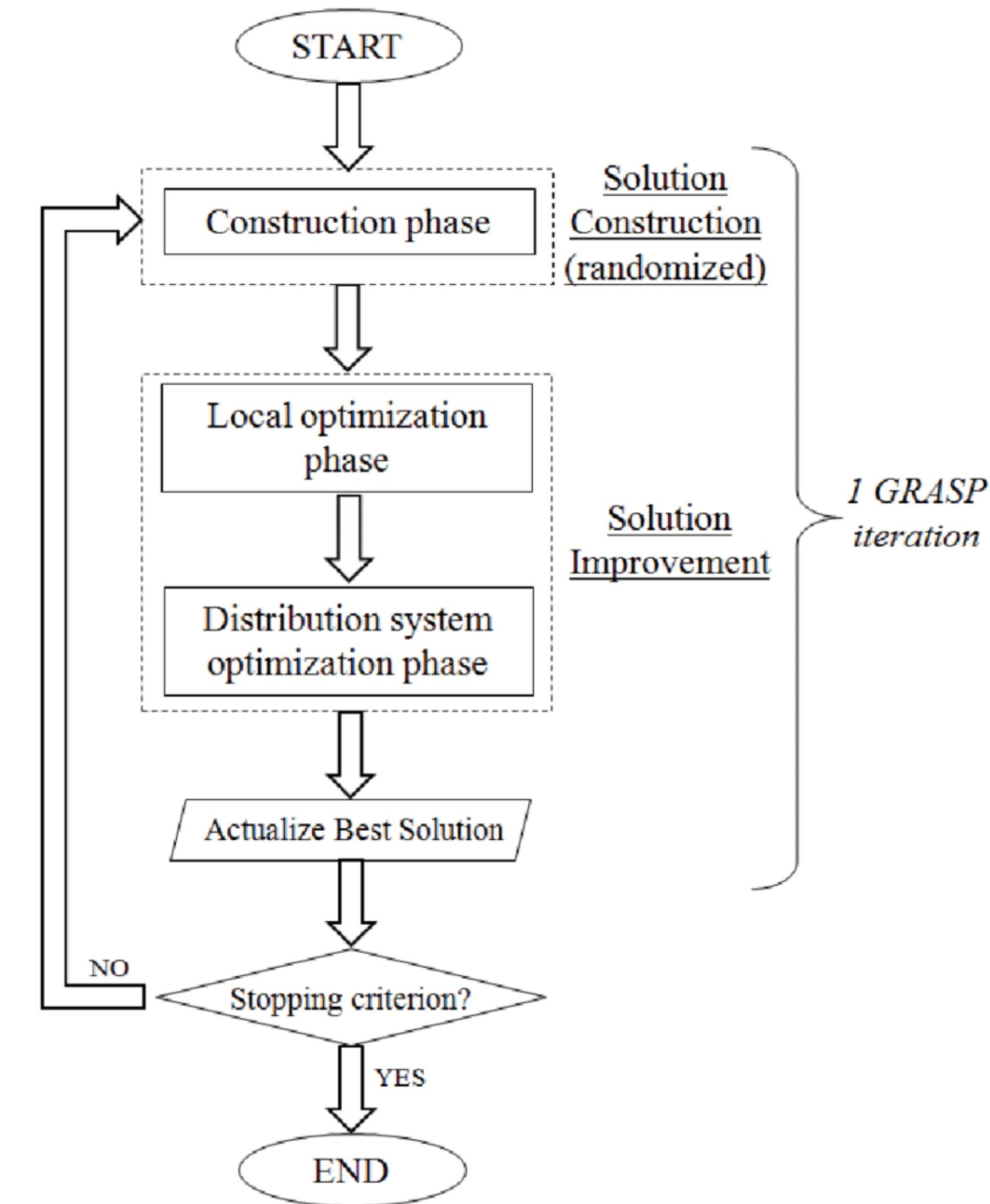
Introdução Grasp

Fases do Grasp:

- Construção ramdomica e miope
- Busca local
- *Distribution system*
- Atualiza melhor solução
- Critério de parada

Parâmetros do Grasp:

- K → Tamanho da RCL (Restricted Candidate List), Grau de aleatoriedade.
- Iterations → Número de iterações que o Grasp vai realizar



Fonte: https://www.researchgate.net/figure/Main-structure-of-the-GRASP-based-algorithm_fig33_280246925

Modelagem do problema

datasets utilizados no problema:

- PDF com histórico do aluno.
- Horários de disciplinas (24.1 a 25.1).
- pré-requisitos (Json criado com os dados)
- Equivalências

Objetivo da função:

$$f(D) = \sum_{d_i \in D} \text{score}(d_i)$$

Para definir a **melhor solução**, foi utilizada uma função de **máximo**, cujo objetivo é pegar a maior quantidade de matérias assim como as melhores **matérias possíveis** para aquele semestre.

Modelagem do problema

Criação da estrutura de dados:

- PDF com histórico do aluno.
 - Regex para ler o PDF
 - Pegar: "Componentes Curriculares Obrigatórios Pendentes"
 - Devolve uma estrutura no formato:
 - { "historico": List[Dict[str, Any]],
 - "pendentes": List[str]}
-
- Horários de disciplinas (24.1 a 25.1).
 - Excel extraído e formatado criando um CSV
 - CSV com cabeçalho de cada dia da semana e o horário de aula

Modelagem do problema

Criação da estrutura de dados:

```
    "periodo": "2024.2",
    "Disciplina": "XMC001",
    "Nota": "--",
    "Situação": "MATR"
  },
  {
    "periodo": "2024.2",
    "Disciplina": "XRSC01",
    "Nota": "--",
    "Situação": "MATR"
  }
],
"pendentes": [
  "MAT00B",
  "CTC004",
  "CMAC05",
  "XRSC01",
  "CRSC05",
  "XPAD01",
  "XMC001",
  "CTC005",
  "CTC006",
  "XAHC01",
  "XAHC03"
]
```

Horário,Segunda-feira,Terça-feira,Quarta-feira,Quinta-feira,
7:00,,,
7:55,,XMAC01,XMAC01,,
8:50,,XMAC01,XMAC01,,
10:10,,XDES01,XDES01,,
11:05,,XDES01,XDES01,,
13:30,,,MAT00A,CAHC04,
14:25,,,MAT00A,CAHC04,
15:45,MAT00A,CRSC03,,CRSC03,
16:40,MAT00A,CRSC03,,CRSC03,
17:35,,,
19:00,MAT00A,XDES01,IEPG22,SAHC05,SACH04
19:50,MAT00A,XDES01,IEPG22,SAHC05,SACH04
21:00,SAHC05,IEPG01,MAT00A,XDES01,

Modelagem do problema

Criação da estrutura de dados:

pré-requisitos :

- Json criado com os dados de todos as matérias e seus pré-requisitos

```
},
"IEPG21": {
    "name": "Ciências Humanas e Sociais",
    "prerequisites": [],
    "isMandatory": false
},
"ADM08H": {
    "name": "Psicologia: As relações Indivíduo-Grupo",
    "prerequisites": [],
    "isMandatory": false
},
"LET007": {
    "name": "LIBRAS - Linguagem Brasileira de Sinais",
    "prerequisites": [],
    "isMandatory": false
},
"ADM51H": {
    "name": "Ciências Tecnológicas e Organizacionais"
```

Equivalências:

- Excel extraido e formatado criando um CSV
- CSV com cabeçalho sendo o código da matéria e as suas equivalências nas colunas seguintes

Modelagem do problema

Restrições do problema

- pré requisitos
- conflitos de horário

Função de Score

Verifica o semestre anterior e procura matérias que só são ofertadas 1x no ano. atribui um peso maior para essas matérias

Greedy Randomized Adaptive Search Procedure

Chamada da função GRASP

```
best_solution, best_score = grasp.grasp(  
    missing_disciplines,  
    availableDisciplines,  
    prerequisites,  
    catalog_previous=neighborDisciplines,  
    equivalences=equivalences,  
    iterations=arg,  
    k=arg  
)
```

Greedy Randomized Adaptive Search Procedure

Início Construção Random e Míope

- Por ser randomica evita sempre cair na mesma solução e explora novas possibilidades
- Míope, informação imediata, vai buscar o melhor agora sem enxergar muito a frente
- A cada iteração verifica as possibilidades viáveis
- Das viáveis cria uma RCL (Lista de candidatos restrita)
- Um subconjunto com até K melhores disciplinas

Pseudocódigo Construtiva:

```
Função construct_solution(missing_disciplines, catalog, prerequisites, equivalences_map, schedule_map, k)
    candidates ← missing_disciplines
    Enquanto candidates não estiver vazio:
        feasible_candidates ← []
        Para cada disc em candidates:
            equivils ← equivalentes de disc (ou disc)
            Se existe candidate_disc em equivils que:
                - Está em schedule_map
                - Não está em solution
                - Pode ser adicionada (can_add_discipline)
            Então adicione candidate_disc a feasible_candidates
            Se feasible_candidates vazio:
                Pare
            rcl ← primeiros k de feasible_candidates (ou todos se < k)
            chosen ← escolha aleatória de rcl
            Adicione chosen a solution
            Remova de candidates todas as disciplinas equivalentes a chosen
    Retorne solution
```

1. Enquanto houver candidatos:
2. Selecione disciplinas equivalentes viáveis (existem, não estão na solução, respeitam pré-requisitos).
3. Se não houver viáveis, pare.
4. Escolha aleatoriamente uma entre as k melhores e adicione à solução.
5. Remova da lista de candidatos todas as equivalentes à escolhida.

Greedy Randomized Adaptive Search Procedure

Fase de busca local

- Parâmetros:
 - solution: a solução inicial (lista de disciplinas selecionadas).
 - missing_disciplines: disciplinas que o aluno ainda precisa cursar.
 - catalog: catálogo de disciplinas com horários.
 - prerequisites: mapa de pré-requisitos.
 - equivalences_map: mapa de equivalências entre disciplinas.
 - schedule_map: mapa de horários para cada disciplina.
 - rare_disciplines: conjunto de disciplinas que são ofertadas 1x no ano valem mais pontos.

Greedy **R**andomized **A**daptive **S**earch **P**rocedure

Fase de busca local

Inicia a melhor solução e o score dela e então realiza um loop para melhorar essa solução e continua no loop até não conseguir melhorar ela mais, dentro do loop identifica candidatos para adicionar e tenta adicionar ou substituir cada candidato, considerando também as candidatas. Após isso, verifica conflitos de horário e remove as disciplinas conflitantes.

```

Função local_search(solution, missing_disciplines, catalog, prerequisites, equivalences_map, schedule_map, rare_disciplines)
    best_solution ← cópia de solution
    best_score ← score(best_solution, rare_disciplines)
    improved ← True
    Enquanto improved for True:
        improved ← False
        candidates ← missing_disciplines - best_solution
        Para cada cand em candidates:
            equivs ← equivalentes de cand (ou cand)
            Para cada candidate_disc em equivs:
                Se candidate_disc não está em schedule_map → continue
                Se candidate_disc já está em best_solution → continue
                    conflict_disc ← None
                    candidate_schedule ← horários de candidate_disc
                    Para cada sol_disc em best_solution:
                        sol_schedule ← horários de sol_disc
                    Se candidate_schedule conflita com sol_schedule:
                        conflict_disc ← sol_disc
                        Pare
                        Se conflict_disc existe:
                            new_solution ← best_solution - conflict_disc + candidate_disc
                            Senão:
                                new_solution ← best_solution + candidate_disc
                                Se can_add_discipline(candidate_disc, new_solution, ...) e
                                Se todos os pré-requisitos de new_solution são satisfeitos:
                                    new_score ← score(new_solution, rare_disciplines)
                                    Se new_score > best_score:
                                        best_solution ← new_solution
                                        best_score ← new_score
                                        improved ← True
                                        Pare
                                        Se improved:
                                            Pare (para reiniciar busca com nova solução)
                                            Retorne best_solution

```

Pseudocódigo busca local

Resumo:

- Tenta adicionar disciplinas que ainda faltam. (Vizinhança)
- Se houver conflito de horário, troca a disciplina conflitante.
- Só aceita a troca se:
 - Não violar pré-requisitos
 - Melhorar a pontuação da solução.
- Repete até não conseguir mais melhorar.

Ao final:

Após rodar as N iterações de construtiva e busca local o código retorna a melhor solução e o seu score.

Experimentos e resultados obtidos

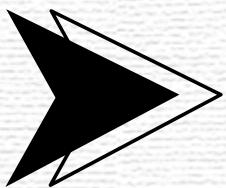
Execução do código

```
python src/main.py --dataset_name "Nome do histórico" --period "25.1" --mh "grasp"
```

Parâmetros opcionais:

- `--update_json` → Faz o parser completo do PDF e atualiza o Json dele para futuras leituras
- `--k` e `--i` → parâmetros para o Grasp, valor padrão deles é de 3 e 100 respectivamente

Saída esperada:



```
python src/main.py --dataset_name "historico_SIN-4" --period "25.1" --mh "grasp"  
Namespace(dataset_name='historico_SIN-4', update_json='n', period='25.1', mh='grasp',
```

k=3, i=100)

25.1

24.2

Executando o algoritmo Grasp...

Melhor solução encontrada:

```
['SADG01', 'SPAD02', 'ADM51E', 'XMAC01']
```

Pontuação: 4

Total execution time: 0.070254 seconds

```
● kruzeiori@fedora:~/Repositórios/matrícula_metaheuristica$ python src/main.py --dataset_name "historico_SIN-4" --period "25.1" --mh "grasp"  
Namespace(dataset_name='historico_SIN-4', update_json='n', period='25.1', mh='grasp', k=3, i=100)  
25.1  
24.2  
Executando o algoritmo Grasp...  
Melhor solução encontrada:  
['SADG01', 'SPAD02', 'ADM51E', 'XMAC01']  
Pontuação: 4  
Total execution time: 0.070254 seconds
```

Resultados obtidos:

Dataset	Resultado
SIN_1	Null
SIN_2	Null
SIN_3	'ADM51E', 'SPAD02'
SIN_4	'SPAD02', 'SPAD03', 'ADM51E', 'XMAC01'
SIN_5	'SADG01', 'SPAD02', 'ADM51E'

Conclusão

O algoritmo GRASP tem uma simples implementação e bem forte para encontrar a melhor solução de problemas. Nesse projeto a maior dificuldade foi da implementação de leitura dos diferentes arquivos e variáveis que envolvem a matrícula assim como alguns datasets estarem em situações complicadas quanto a respeito as disciplinas e estado atual do aluno.

Os resultados obtidos parecem condizentes com o cenário dos datasets e das disciplinas disponíveis para o dado período para alguns datasets. Outros tornam mais complicado a leitura devido a muitos trancamentos ou reprovações completas em mais de um semestre.

Referências

PESQUISAS BIBLIOGRÁFICAS

- Resende, M. G., & Ribeiro, C. C. (2016). Optimization by GRASP. Springer Science+ Business Media New York.
- González-Neira, E. M., & Montoya-Torres, J. R. (2017). A GRASP meta-heuristic for the hybrid flowshop scheduling problem. *Journal of Decision systems*, 26(3), 294-306.

VÍDEOS

- https://www.youtube.com/watch?v=dgKe06K_hlc&ab_channel=CarlosGarc%C3%ADa-Mart%C3%ADnez
- https://www.youtube.com/watch?v=9R-_BL_5Gno&t=790s&ab_channel=BrunoPrata