AIAD - MIEIC - FEUP

Alocação de alunos de 2ª fase em turmas conforme paridade e ocupação da sala

Ano letivo 2020/2021

Profª Ana Paula Rocha

Alunos:
Henrique Freitas, 201707046
José Gomes, 201707054
Liliana Almeida, 201706908

Descrição do problema

O problema consiste na atribuição de alunos que entram no ensino superior na 2ª fase de colocação em turmas já existentes, tendo em conta a paridade dos mesmos (considerando o problema que surgiu com a pandemia do Covid-19).

Para tal, recorreu-se a dois tipos de agentes:

- **Student -** representa um aluno e é caracterizado pela sua paridade.
- **CUClass -** representa uma turma e é caracterizado pela sua capacidade, ocupação atual e número atual de alunos pares.

Sempre que um aluno entra no sistema, subscreve a todas as turmas existentes, através do protocolo FIPA-Subscribe, para se manter atualizado do valor da *utility* de cada uma delas.

Após verificar qual a turma com melhor *utility*, tendo em conta a sua paridade, faz um pedido para ser alocado na mesma. Se o último valor da *utility* guardado pelo aluno estiver atualizado, a turma aceita-o e aloca-o. Caso contrário, é rejeitado e o aluno atualiza os seus valores e volta a fazer a repetir o processo.

Variáveis Independentes

As variáveis independentes, as quais são possíveis definir na interface do Repast e/ou através de ficheiros de parâmetros, são as seguintes:

NumerOfEvenStudents - Número de alunos pares a entrar

NumerOfOddStudents - Número de alunos ímpares a entrar

ClassesStats - Informações sobre as turmas, de onde se pode retirar:

- Número de turmas
- Capacidade de cada turma
- Ocupação inicial de cada turma
- Número de alunos pares inicialmente em cada turma

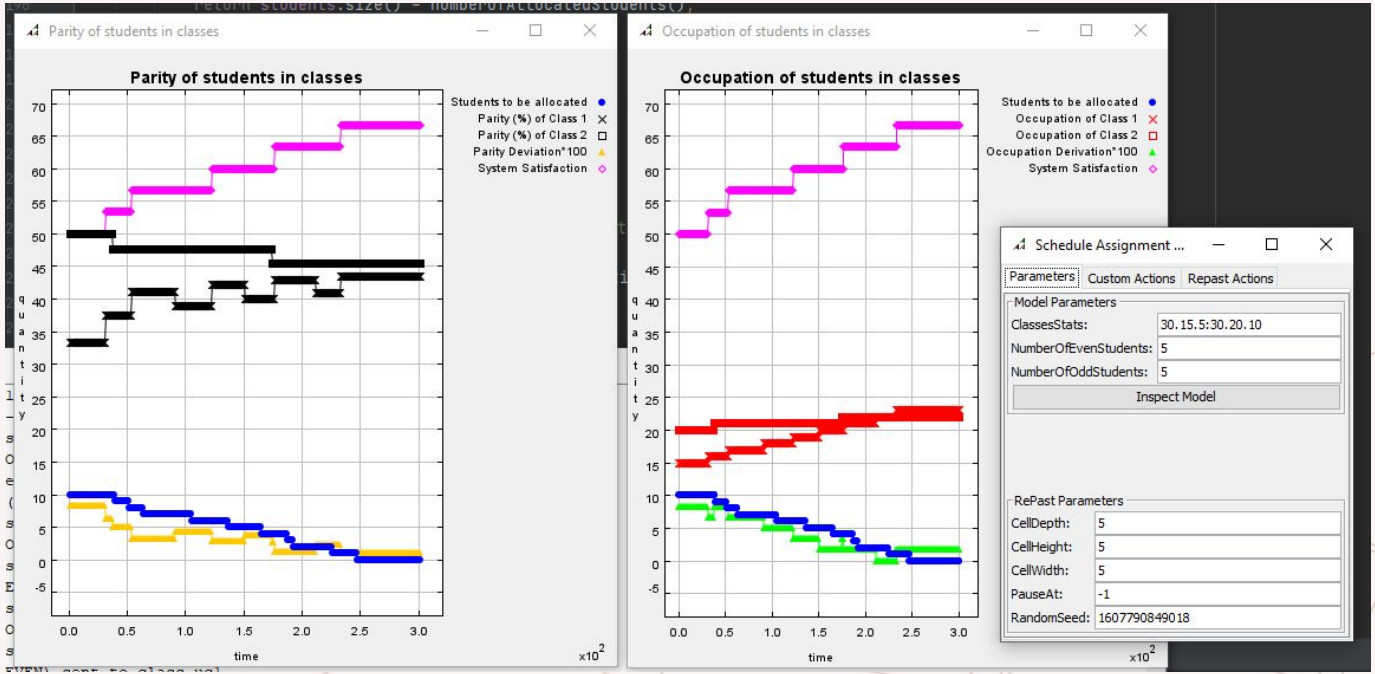
Variáveis Dependentes a Observar

As variáveis dependentes que são visualizadas em gráficos, em runtime, são as seguintes:

- Número de alunos que falta alocar
- Satisfação global do sistema
- Percentagem de alunos pares em cada turma
- Desvio padrão global da paridade das turmas
- Ocupação atual de cada turma
- Desvio padrão global da ocupação das turmas

Visualização da execução da simulação

A visualização da simulação foi dividida em dois gráficos que ilustram a evolução, ao longo do tempo, da paridade e da ocupação respetivamente. A linha azul de cada gráfico corresponde aos alunos que ainda não foram alocados, pelo que uma descida nessa linha simboliza um aluno a ser alocado. A linha rosa representa a satisfação total do sistema.

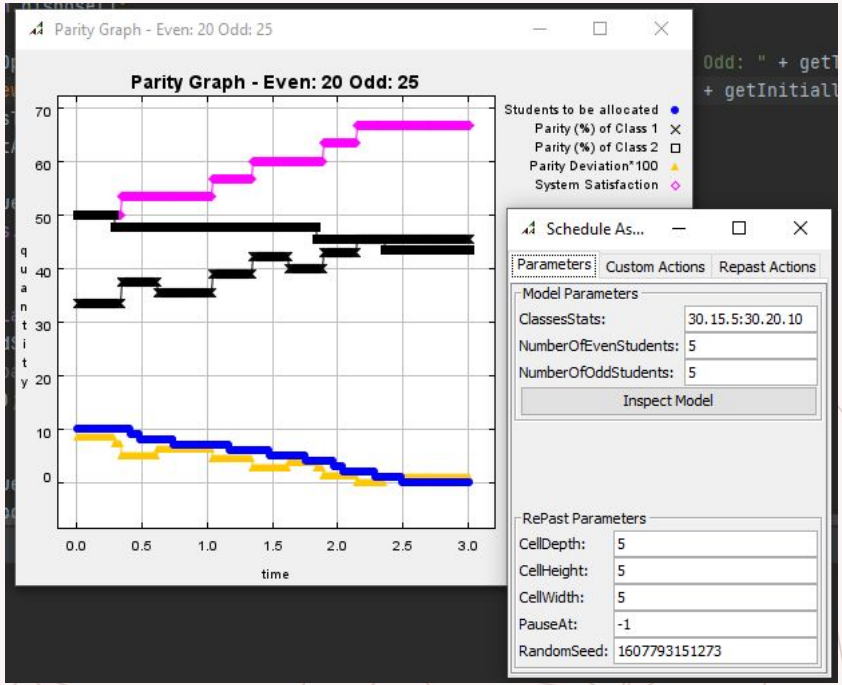


Visualização da execução da simulação

Gráfico da Paridade

Este gráfico traça a evolução da paridade do sistema ao longo do tempo. Cada linha preta representa o rácio de alunos pares sobre o número total de alunos alocados a uma turma. O número ideal para este valor é de **50%** que significa que existem **tantos alunos pares como ímpares na turma**.

Para medir o equilíbrio das turmas, a linha amarela representa o desvio padrão da paridade entre as turmas. Assim, quanto mais longe as linhas pretas estiverem umas das outras, maior será o valor da linha amarela. O valor ideal para esta linha é de **zero** e significa que todas as turmas têm o mesmo rácio de paridade.

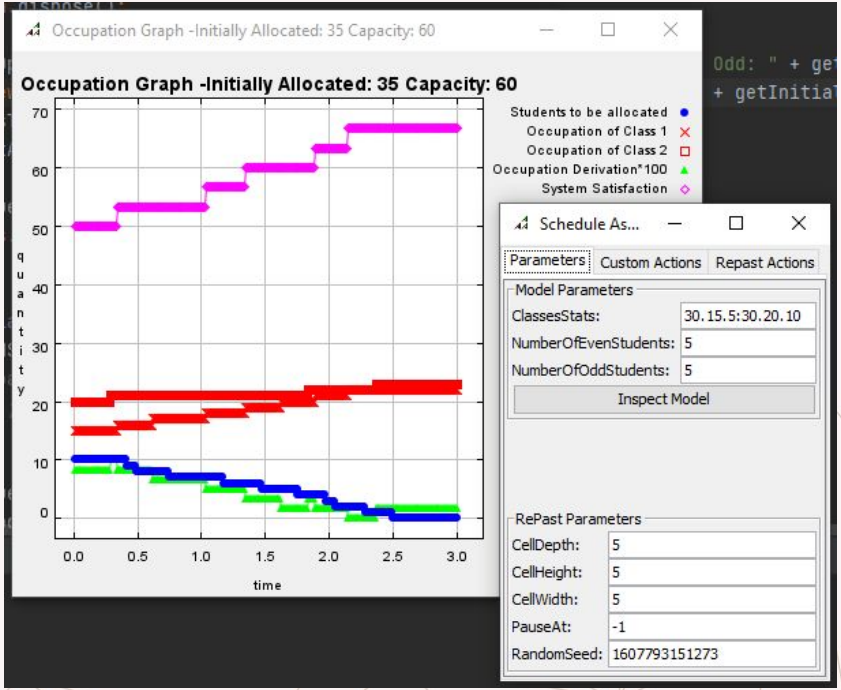


Visualização da execução da simulação

Gráfico da Ocupação

Este gráfico traça a evolução da ocupação do sistema ao longo do tempo. Cada linha vermelha representa o número de alunos alocados a uma turma. O valor ideal para esta linha é igual à **capacidade máxima de cada turma**.

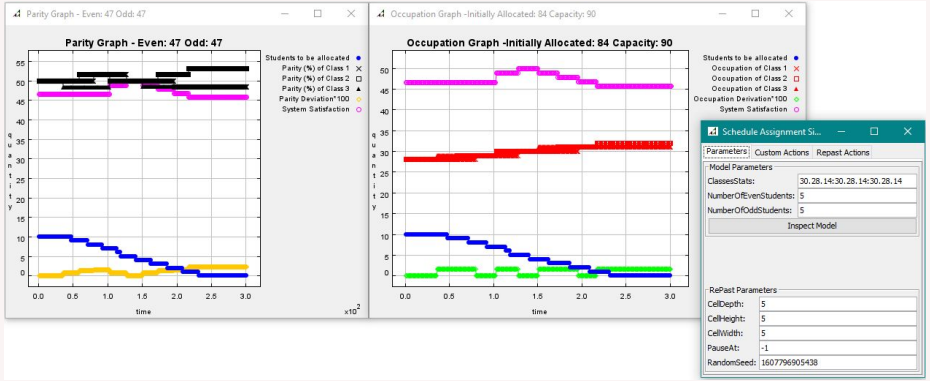
Para medir o equilíbrio da turma, a linha verde representa o desvio padrão da ocupação entre as turmas. Assim, quanto mais perto as linhas linha vermelhas estiverem umas das outras, menor será o valor da linha verde. O valor ideal para esta linha é de **zero** e significa que todas as turmas têm o mesmo número de alunos.



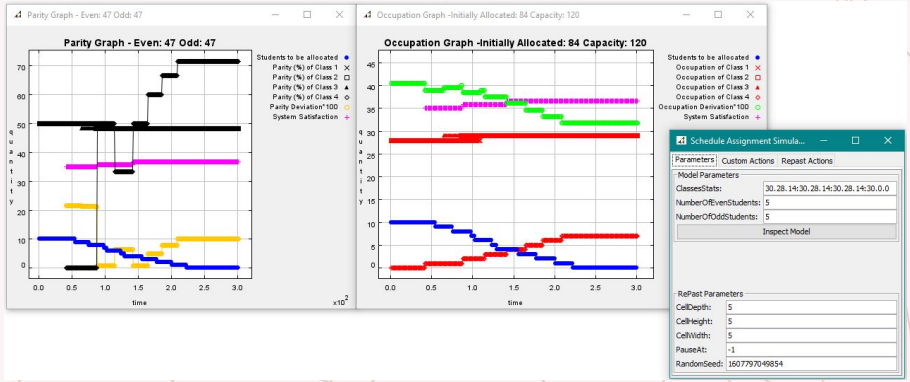
Experiências realizadas e análise de resultados

No caso de existirem **3 turmas equilibradas**, em termos de paridade, e praticamente cheias, faltando apenas **2 alunos para preencher cada uma** delas, ao tentarmos **adicionar 10 alunos** (5 pares e 5 ímpares) conseguimos observar que **a satisfação global aumenta até a um valor máximo** e depois diminui. Isto acontece porque a quantidade de alunos em cada turma excede a sua capacidade total.

Quando se adiciona uma turma vazia às já existentes, é possível concluir que a satisfação global do sistema aumenta sempre ao longo do tempo. No entanto, este valor é inferior ao obtido no caso anterior.



Caso 1: 3 turmas equilibradas praticamente cheias

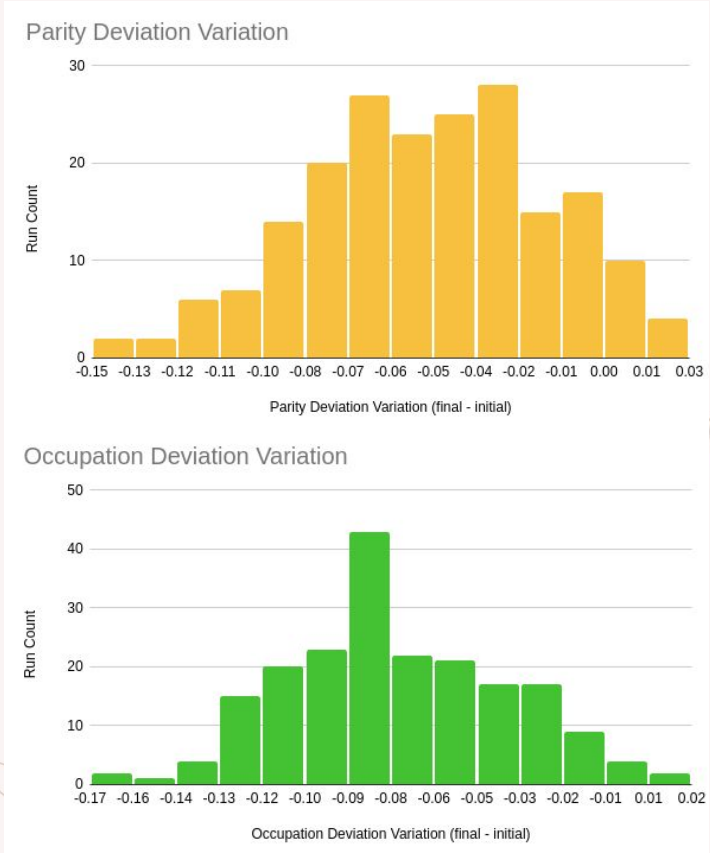


Caso 2: 3 turmas equilibradas praticamente cheias e uma turma vazia

Experiências realizadas e análise de resultados

Estes 2 gráficos apresentam, respectivamente, a distribuição da variação do desvio-padrão da paridade e da ocupação, numa simulação com **200 *runs***. Os dados utilizados neste conjunto são ligeiramente aleatórios*. Desta forma, inicialmente as turmas estão **desequilibradas**.

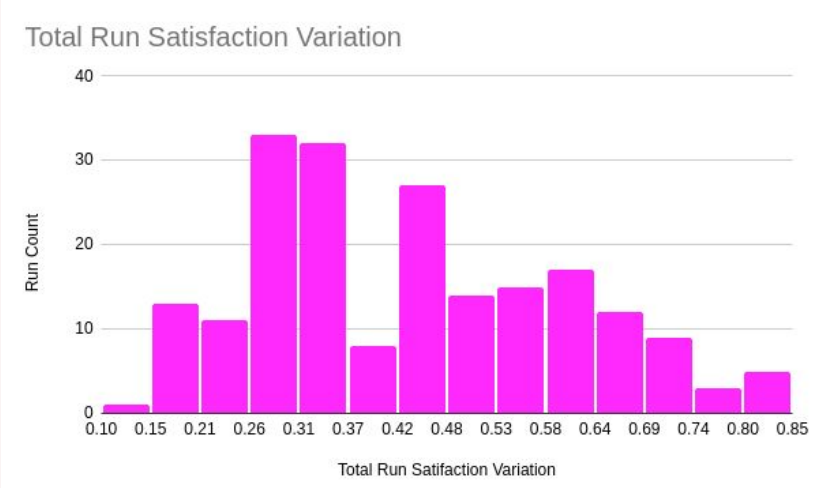
Como é possível observar, a variação é **maioritariamente negativa**, e significa que o valor de desvio-padrão, tanto na paridade como na ocupação, desce do início para o fim. Assim, pode-se provar que o sistema tem como prioridade **equilibrar**, dentro das suas possibilidades, as turmas. Deste modo, há a garantia de que, já estando as turmas equilibradas (como no mundo real estariam), **o algoritmo tenta manter o equilíbrio**.



* (capacidade das turmas = 30, ocupação = [10, 20], paridade = [35, 65]%, alunos pares=[5, 20], alunos ímpares=[5, 20], número de turmas=[2, 4])

Experiências realizadas e análise de resultados

Relativamente à satisfação geral do sistema, neste gráfico é possível verificar a distribuição dos valores de variação ao longo de cada *run* (na mesma simulação de 200 *runs*). Note-se que em todos os casos a variação é positiva. Isto deve-se em parte ao cálculo da satisfação favorecer paridade próxima de 50% e ocupação próxima de 100% (capacidade). Isto faz com que, ao adicionar um aluno que faça a paridade afastar-se de 50%, a satisfação não seja tão penalizada (ou até beneficiada), dado que a ocupação aproximou-se de 100%.



Conclusão

Esta segunda parte do trabalho permitiu realizar as experiências efetuadas na primeira parte do trabalho de uma forma mais **fácil e eficiente** e permitiu também melhor **analisar os dados** de cada experiência.

Conseguimos perceber que o problema que modelamos, embora seja uma versão simplificada da realidade, tem muitos pormenores e detalhes que podem ser estudados e tidos em conta num sistema real.

Achamos que os **objetivos** do projeto foram **alcançados com sucesso** na medida em que o modelo que estudamos foi trabalhado em modo de **simulação** utilizando as funcionalidades do ***Repast3*** e também as capacidades de **Comunicação** e dos **Agentes** do ***Jade***.

AIAD - MIEIC - FEUP

Alocação de alunos de 2ª fase em turmas conforme paridade e ocupação da sala

Informações Adicionais

Ano letivo 2020/2021

Profª Ana Paula Rocha

Alunos:
Henrique Freitas, 201707046
José Gomes, 201707054
Liliana Almeida, 201706908

Exemplos detalhados de execução

Tomando como exemplo o *Case1,* ao iniciarem, os alunos procuram as turmas e subscrevem à sua *utility* (linha 5). Quando obtém a subscrição de todas as turmas imprimem uma mensagem de confirmação (linha 7).

```
5: Agent uc2: Agrees Student3 subscription
6: Agent Student3 : uc2 agreed utility
7: Agent Student3 :  confirmed all subscriptions
```

De seguida, os alunos iniciam o processo de escolha das turmas, selecionando a que tem melhor *utility* (linha 10) e iniciando o processo de atribuição (linha 11). Caso tudo aconteça sem problemas, o output será parecido ao seguinte:

```
10: Agent: Student4 best Class: uc1 utility -> 40.0
11: Agent uc1: ASSIGN received from Student4
12: Agent Student4 : uc1 agreed to add
13: Agent uc1: Action successfully performed
14: Agent Student4 : uc1 added this student
```

Nota: As referências às linhas utilizadas nestes slides não correspondem exatamente à ordem da execução do código dada a sua natureza assíncrona

Exemplos detalhados de execução

No caso descrito em baixo, a *utility* que o aluno julgava a turma ter foi alterada a meio do processo de atribuição (linha 32). Assim, a turma recusa o pedido de atribuição (linha 33) e o aluno recomeça o processo com a nova *utility* (linha 34).

```
30: Agent: Student2 best Class: uc1 utility -> 40.0
31: Agent uc1: ASSIGN received from Student2
32: Agent Student2 : uc1 updated utility
33: Agent Student2 : uc1 said invalid utility
34: Agent: Student2 best Class: uc2 utility -> 40.0
```

Nota: As referências às linhas utilizadas nestes slides não correspondem exatamente à ordem da execução do código dada a sua natureza assíncrona

Exemplos detalhados de execução

Ao terminar a execução do programa de forma graciosa, cada aluno imprime a turma na qual entrou, a sua paridade e a *utility* que a mesma tinha.

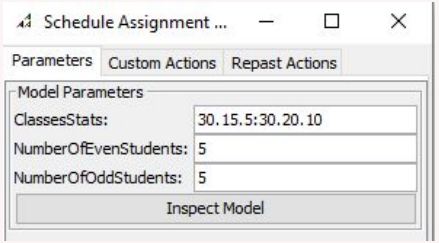
```
40: Agent: Student2 best Class: uc2 utility -> 40.0
41: Student Student2 (EVEN) sent to class uc2
42: Agent: Student1 best Class: uc2 utility -> 34.833332
43: Student Student1 (EVEN) sent to class uc2
44: Agent: Student3 best Class: uc1 utility -> 30.857143
45: Student Student3 (EVEN) sent to class uc1
46: Agent: Student4 best Class: uc1 utility -> 40.0
47: Student Student4 (EVEN) sent to class uc1
48: Agent: Student5 best Class: uc1 utility -> 34.833332
49: Student Student5 (EVEN) sent to class uc1
```

Nota: As referências às linhas utilizadas nestes slides não correspondem exatamente à ordem da execução do código dada a sua natureza assíncrona

Exemplos detalhados de execução

Repast - Single Run

Para correr apenas uma simulação os valores iniciais podem ser introduzidos na GUI do Repast de acordo com a imagem abaixo. O parâmetro ***"ClassesStats"*** é uma **string** no formato: *"Capacity1.OccupiedSeats1.EvenStudents1:Capacity2.OccupiedSeats2.EvenStudents2"* em que cada turma está separada por "**:**" e cada atributo dentro da turma está separado por "**.**".



Exemplos detalhados de execução

Repast - BatchMode

Para correr várias simulações pode ser utilizado um ficheiro de parâmetros que siga o formato abaixo dado como exemplo.

```
runs: 1
numberOfEvenStudents {
 set_list: 5 4 3 3

}
numberOfOddStudents {
 set_list: 5 1 2 2
}

classesStats {
 set_string_list: 30.15.5:30.20.10 30.10.1:30.10.9 30.6.3:30.20.10:30.30.15 30.10.1:30.10.9:30.10.5
}
```