Auto Scheduler

Generated by Doxygen 1.8.11

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Namespace Documentation

3.1 csp Namespace Reference

Functions

- def consequtive_exams ()
- def prof_preference ()
- def calc_score ()

Variables

- int total_capacity = 0
- int slots_per_day = 4
- int days = 6
- total_slots = days∗slots_per_day
- dictionary course_numstudents = {}
- list courses = [ ]
- dictionary common_students = {}
- dictionary available_slots = {}
- dictionary course_slot = {}
- dictionary slot_courses = {}
- dictionary slot_numstudents = {}
- dictionary prof_input = {}
- int n = 1
- list weight = [100∗n , 100∗n , 25∗n]
- int temperature = 30000

  Here we attempt to get a random feasible solution(ie a solution which satisfies all hard constraints).
- float alpha = 0.8
- t_limit = math.pow(10,-11)
- int iterator = 10
- c = course_list[course]

  Create a 2d array of the common students in the course1 and course2.
- courses_copy = list(courses)
- int val = 0
- minslots = total_slots
- list minlist = [ ]
- num_aval_slots = len(available_slots[course])

- s = secrets.choice(available_slots[c])
- int count_t = 0
- slot_list = list(range(1,total_slots+1))
- o_x = calc_score()
- o_q = calc_score()
- var = o_x-o_q
- prob = math.exp(var/temperature)
- rand = numpy.random.choice([0,1], p=[prob, 1-prob])
- int iterate = 0

  Hill Climbing.

3.1.1 Function Documentation

3.1.1.1 def csp.calc_score ( )

3.1.1.2 def csp.consequtive_exams ( )

3.1.1.3 def csp.prof_preference ( )

3.1.2 Variable Documentation

3.1.2.1 float csp.alpha = 0.8

3.1.2.2 dictionary csp.available_slots = {}

3.1.2.3 csp.c = course_list[course]

Create a 2d array of the common students in the course1 and course2.

STEP ANHEALING.

Here we generate neighbourhood solutions. And try to get to to the most optimized solution by iterating over solutions.

3.1.2.4 dictionary csp.common_students = {}

3.1.2.5 int csp.count_t = 0

3.1.2.6 dictionary csp.course_numstudents = {}

3.1.2.7 dictionary csp.course_slot = {}

3.1.2.8 list csp.courses = [ ]

3.1.2.9 csp.courses_copy = list(courses)

3.1.2.10 int csp.days = 6

3.1.2.11 int csp.iterate = 0

Hill Climbing.

In approximately 5-10% of cases, the simulated annealing algorithm had not sufficiently explored the neighbourhood of its best solution.

3.1.2.12 int csp.iterator = 10

3.1.2.13 list csp.minlist = [ ]

3.1.2.14 csp.minslots = total_slots

3.1.2.15 int csp.n = 1

3.1.2.16 csp.num_aval_slots = len(available_slots[course])

3.1.2.17 csp.o_q = calc_score()

3.1.2.18 csp.o_x = calc_score()

3.1.2.19 csp.prob = math.exp(var/temperature)

3.1.2.20 dictionary csp.prof_input = {}

3.1.2.21 csp.rand = numpy.random.choice([0,1], p=[prob, 1-prob])

3.1.2.22 csp.s = secrets.choice(available_slots[c])

3.1.2.23 dictionary csp.slot_courses = {}

3.1.2.24 csp.slot_list = list(range(1,total_slots+1))

3.1.2.25 dictionary csp.slot_numstudents = {}

3.1.2.26 int csp.slots_per_day = 4

3.1.2.27 csp.t_limit = math.pow(10,-11)

3.1.2.28 int csp.temperature = 30000

Here we attempt to get a random feasible solution(ie a solution which satisfies all hard constraints).

- `That` is we first randomly allot a course to to a slot.
- `Then` we allot slots to courses in similar fashion.
- `If` we fail in this attempt we restart the loop and allot the course some other random slot. Note- Here we are considering that there exists finitely many solutions for the given number of slots.

3.1.2.29 int csp.total_capacity = 0

3.1.2.30 csp.total_slots = days∗slots_per_day

3.1.2.31 int csp.val = 0

3.1.2.32 csp.var = o_x-o_q

3.1.2.33 list csp.weight = [100∗n , 100∗n , 25∗n]

3.2 generate_course_list Namespace Reference

Variables

- f = open("course_students.txt", "r")
- dictionary course_list = {}
- line1 = line.rstrip(",")
- cols = line1.split(",")

3.2.1 Variable Documentation

3.2.1.1 generate_course_list.cols = line1.split(",")

3.2.1.2 dictionary generate_course_list.course_list = {}

3.2.1.3 generate_course_list.f = open("course_students.txt", "r")

3.2.1.4 generate_course_list.line1 = line.rstrip(",")

3.3 generate_room_list Namespace Reference

Variables

- f = open("room_capacity.txt", "r")
- dictionary room_list = {}
- line_without_newline = line.rstrip()
- cols = line_without_newline.split(",")

3.3.1 Variable Documentation

3.3.1.1 generate_room_list.cols = line_without_newline.split(",")

3.3.1.2 generate_room_list.f = open("room_capacity.txt", "r")

3.3.1.3 generate_room_list.line_without_newline = line.rstrip()

3.3.1.4 dictionary generate_room_list.room_list = {}

3.4 generate_student_list Namespace Reference

Variables

- f = open("student_courses.txt", "r")
- dictionary student_list = {}
- line_without_newline = line.rstrip()
- cols = line_without_newline.split(",")

3.4.1 Variable Documentation

3.4.1.1 generate_student_list.cols = line_without_newline.split(",")

3.4.1.2 generate_student_list.f = open("student_courses.txt", "r")

3.4.1.3 generate_student_list.line_without_newline = line.rstrip()

3.4.1.4 dictionary generate_student_list.student_list = {}

3.5 room_allocation Namespace Reference

Variables

- dictionary room_status = {}
- dictionary course_rooms = {}
- dictionary aval_rooms = {}
- list available_rooms = [ ]
- dictionary room_capacity = {}
- course_num = dict(course_numstudents)
- sorted_room_list = sorted(room_list, key=lambda x: int(room_list[x]))
- r = aval_rooms[course][len(aval_rooms[course])-1]

3.5.1 Variable Documentation

3.5.1.1 list room_allocation.available_rooms = [ ]

3.5.1.2 dictionary room_allocation.aval_rooms = {}

3.5.1.3 room_allocation.course_num = dict(course_numstudents)

3.5.1.4 dictionary room_allocation.course_rooms = {}

3.5.1.5 room_allocation.r = aval_rooms[course][len(aval_rooms[course])-1]

3.5.1.6 dictionary room_allocation.room_capacity = {}

3.5.1.7 dictionary room_allocation.room_status = {}

3.5.1.8 room_allocation.sorted_room_list = sorted(room_list, key=lambda x: int(room_list[x]))

Chapter 4

File Documentation

4.1 csp.py File Reference

Namespaces

- csp

Functions

- def csp.consequtive_exams ()
- def csp.prof_preference ()
- def csp.calc_score ()

Variables

- int csp.total_capacity = 0
- int csp.slots_per_day = 4
- int csp.days = 6
- csp.total_slots = days∗slots_per_day
- dictionary csp.course_numstudents = {}
- list csp.courses = [ ]
- dictionary csp.common_students = {}
- dictionary csp.available_slots = {}
- dictionary csp.course_slot = {}
- dictionary csp.slot_courses = {}
- dictionary csp.slot_numstudents = {}
- dictionary csp.prof_input = {}
- int csp.n = 1
- list csp.weight = [100∗n , 100∗n , 25∗n]
- int csp.temperature = 30000

  Here we attempt to get a random feasible solution(ie a solution which satisfies all hard constraints).
- float csp.alpha = 0.8
- csp.t_limit = math.pow(10,-11)
- int csp.iterator = 10
- csp.c = course_list[course]

  Create a 2d array of the common students in the course1 and course2.

- csp.courses_copy = list(courses)
- int csp.val = 0
- csp.minslots = total_slots
- list csp.minlist = [ ]
- csp.num_aval_slots = len(available_slots[course])
- csp.s = secrets.choice(available_slots[c])
- int csp.count_t = 0
- csp.slot_list = list(range(1,total_slots+1))
- csp.o_x = calc_score()
- csp.o_q = calc_score()
- csp.var = o_x-o_q
- csp.prob = math.exp(var/temperature)
- csp.rand = numpy.random.choice([0,1], p=[prob, 1-prob])
- int csp.iterate = 0

  Hill Climbing.

4.2 generate_course_list.py File Reference

Namespaces

- generate_course_list

Variables

- generate_course_list.f = open("course_students.txt", "r")
- dictionary generate_course_list.course_list = {}
- generate_course_list.line1 = line.rstrip(",")
- generate_course_list.cols = line1.split(",")

4.3 generate_room_list.py File Reference

Namespaces

- generate_room_list

Variables

- generate_room_list.f = open("room_capacity.txt", "r")
- dictionary generate_room_list.room_list = {}
- generate_room_list.line_without_newline = line.rstrip()
- generate_room_list.cols = line_without_newline.split(",")

4.4 generate_student_list.py File Reference

Namespaces

- generate_student_list

Variables

- generate_student_list.f = open("student_courses.txt", "r")
- dictionary generate_student_list.student_list = {}
- generate_student_list.line_without_newline = line.rstrip()
- generate_student_list.cols = line_without_newline.split(",")

4.5 room_allocation.py File Reference

Namespaces

- room_allocation

Variables

- dictionary room_allocation.room_status = {}
- dictionary room_allocation.course_rooms = {}
- dictionary room_allocation.aval_rooms = {}
- list room_allocation.available_rooms = [ ]
- dictionary room_allocation.room_capacity = {}
- room_allocation.course_num = dict(course_numstudents)
- room_allocation.sorted_room_list = sorted(room_list, key=lambda x: int(room_list[x]))
- room_allocation.r = aval_rooms[course][len(aval_rooms[course])-1]

Index