COMP9414 Artificial Intelligence

Assignment 1: Constraint Satisfaction Search

@Authors: **Wayne Wobcke, Alfred Krzywicki, Stefano Mezza**

Due Date: Week 5, Friday, October 17, 5.00pm

Objective

This assignment concerns developing optimal solutions to a scheduling problem inspired by the scenario of a manufacturing plant that has to fulfil multiple customer orders with varying deadlines, but where there may be constraints on tasks and on relationships between tasks. Any number of tasks can be scheduled at the same time, but it is possible that some tasks cannot be finished before their deadline. A task finishing late is acceptable, however incurs a cost, which for this assignment is a simple (dollar) amount per hour that the task is late.

A *fuzzy scheduling* problem in this scenario is simplified by ignoring customer orders and having just one machine and a number of *tasks*, each with a fixed duration in hours. Each task must start and finish on the same day, within working hours (9am to 5pm). In addition, there can be *constraints*, both on single tasks and between two tasks. One type of constraint is that a task can have a deadline, which can be "hard" (the deadline must be met in any valid schedule) or "soft" (the task may be finished late – though still at or before 5pm – but with a "cost" per hour for missing the deadline). The aim is to develop an overall schedule for all the tasks (in a single week) that minimizes the total cost of all the tasks that finish late, provided that all the hard constraints on tasks are satisfied.

More technically, this assignment is an example of a *constraint optimization problem* (or *constrained optimization problem*), a problem that has constraints like a standard Constraint Satisfaction Problem (CSP), but also a *cost* associated with each solution. For this assignment, we will use a *greedy* algorithm to find optimal solutions to fuzzy scheduling problems that are specified as text strings. However, unlike the greedy search algorithm described in the lectures on search, this greedy algorithm has the property that it is guaranteed to find an optimal solution for any problem (if a solution exists).

The assignment will use the AIPython code of Poole & Mackworth. You are given code to translate fuzzy scheduling problems specified as text strings into CSPs with a cost, and you are given code for several constraint solving algorithms – based on domain splitting and arc consistency, and based on depth-first search. The assignment will be to implement some missing procedures and to analyse the performance of the constraint solving methods, both analytically and experimentally.

Submission Instructions

- This is an individual assignment.
- Write your answers in **this** notebook and submit **this** notebook on Moodle under **Assignment 1**.
- Name your submission `<zid>-<firstname>-<lastname>.ipynb` where `<firstname>-<lastname>` is your **real** (not Moodle) name.
- Make sure you set up AIPython (as done below) so the code can be run on either CSE machines or a marker's own machine.
- Do not submit any AIPython code. Hence do not change any AIPython code to make your code run.
- Make sure your notebook runs cleanly (restart the kernel, clear all outputs and run each cell to check).
- After checking that your notebook runs cleanly, run all cells and submit the notebook **with** the outputs included (do not submit the empty version).
- Make sure images (for plots/graphs) are **included** in the notebook you submit (sometimes images are saved on your machine but are not in the notebook).
- Do not modify the existing code in this notebook except to answer the questions. Marks will be given as and where indicated.
- If you want to submit additional code (e.g. for generating plots), add that at the end of the notebook.
- **Important: Do not distribute any of this code on the Internet. This includes ChatGPT. Do not put this assignment into any LLM.**

Late Penalties

Standard UNSW late penalties apply (5% of the value of the assignment per day or part day late).

Note: Unlike the CSE systems, there is no grace period on Moodle. The due date and time is 5pm **precisely** on Friday October 17.

Important: You can submit as many times as you want before the due date, but if you do submit before the due date, you cannot submit on Moodle after the due date. If you do not submit before the due date, you can submit on Moodle after the due date.

Plagiarism

Remember that ALL work submitted for this assignment must be your own work and no sharing or copying of code or answers is allowed. You may discuss the assignment with other students but must not collaborate on developing answers to the questions. You may use code from the Internet only with suitable attribution of the source. You may not use ChatGPT or any similar software to generate any part of your explanations, evaluations or code. Do not use public code repositories on sites such as github or file sharing sites such as Google Drive to save any part of your work – make sure your code repository or cloud storage is private and do not share any links. This also applies after you have finished the course, as we do not want next year's students accessing your solution, and plagiarism penalties can still apply after the course has finished.

All submitted assignments will be run through plagiarism detection software to detect similarities to other submissions, including from past years. You should **carefully** read the UNSW policy on academic integrity and plagiarism (linked from the course web page), noting, in particular, that collusion (working together on an assignment, or sharing parts of assignment solutions) is a form of plagiarism.

Finally, do not use any contract cheating "academies" or online "tutoring" services. This counts as serious misconduct with heavy penalties up to automatic failure of the course with 0 marks, and expulsion from the university for repeat offenders.

Fuzzy Scheduling

A CSP for this assignment is a set of variables representing tasks, binary constraints on pairs of tasks, and unary constraints (hard or soft) on tasks. The domains are all the working hours in one week, and a task duration is in hours. Days are represented (in the input and output) as strings 'mon', 'tue', 'wed', 'thu' and 'fri', and times are represented as strings '9am', '10am', '11am', '12pm', '1pm', '2pm', '3pm', '4pm' and '5pm'. The only possible values for the start and end times of a task are combinations of a day and times, e.g. 'mon 9am'. Each task name is a string (with no spaces), and the only soft constraints are the soft deadline constraints.

There are three types of constraint:

- **Binary Constraints:** These specify a hard requirement for the relationship between two tasks.
- **Hard Domain Constraints:** These specify hard requirements for the tasks themselves.
- **Soft Deadline Constraints:** These constraints specify that a task may finish late, but with a given cost.

Each soft constraint has a function defining the *cost* associated with violating the preference, that the constraint solver must minimize, while respecting all the hard constraints. The *cost* of a solution is simply the sum of the costs for the soft constraints that the solution violates (and is always a non-negative integer).

This is the list of possible constraints for a fuzzy scheduling problem (comments below are for explanation and do **not** appear in the input specification; however, the code we supply *should* work with comments that take up a full line):

```
# binary constraints
constraint, ⟨t1⟩ before ⟨t2⟩           # t1 ends when or before
t2 starts
constraint, ⟨t1⟩ after ⟨t2⟩            # t1 starts after or when
t2 ends
constraint, ⟨t1⟩ same-day ⟨t2⟩         # t1 and t2 are scheduled
on the same day
constraint, ⟨t1⟩ starts-at ⟨t2⟩        # t1 starts exactly when
t2 ends

# hard domain constraints
domain, ⟨t⟩, ⟨day⟩, hard                                    # t
starts on given day at any time
domain, ⟨t⟩, ⟨time⟩, hard                                   # t
starts at given time on any day
domain, ⟨t⟩, starts-before ⟨day⟩ ⟨time⟩, hard              # t
starts at or before day, time
domain, ⟨t⟩, starts-after ⟨day⟩ ⟨time⟩, hard               # t
starts at or after day, time
domain, ⟨t⟩, ends-before ⟨day⟩ ⟨time⟩, hard                # t
ends at or before day, time
domain, ⟨t⟩, ends-after ⟨day⟩ ⟨time⟩, hard                 # t
starts at or after day, time
domain, ⟨t⟩, starts-in ⟨day1⟩ ⟨time1⟩-⟨day2⟩ ⟨time2⟩, hard  # day-
time range for start time; includes day1, time1 and day2, time2
domain, ⟨t⟩, ends-in ⟨day1⟩ ⟨time1⟩-⟨day2⟩ ⟨time2⟩, hard    # day-
time range for end time; includes day1, time1 and day2, time2
domain, ⟨t⟩, starts-before ⟨time⟩, hard                     # t
starts at or before time on any day
domain, ⟨t⟩, ends-before ⟨time⟩, hard                       # t
ends at or before time on any day
domain, ⟨t⟩, starts-after ⟨time⟩, hard                      # t
starts at or after time on any day
domain, ⟨t⟩, ends-after ⟨time⟩, hard                        # t
ends at or after time on any day

# soft deadline constraint
domain, ⟨t⟩, ends-by ⟨day⟩ ⟨time⟩ ⟨cost⟩, soft          # cost per
hour of missing deadline
```

The input specification will consist of several “blocks”, listing the tasks, binary constraints, hard unary constraints and soft deadline constraints for the given problem. A ”declaration” of each task will be included before it is used in a constraint. A sample input specification is as follows. Comments are for explanation and do **not** have to be included in the input.

```
# two tasks with two binary constraints and soft deadlines
task, t1 3
task, t2 4
# two binary constraints
constraint, t1 before t2
constraint, t1 same-day t2
# domain constraint
domain, t2 mon
```

```
# soft deadline constraints
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
```

Preparation

1. Set up AIPython

You will need AIPython for this assignment. To find the aipython files, the aipython directory has to be added to the Python path.

Do this temporarily, as done here, so we can find AIPython and run your code (you will not submit any AIPython code).

You can add either the full path (using `os.path.abspath`), or as in the code below, the relative path.

In [12]:
```
import sys
sys.path.append('aipython') # change to your directory
sys.path # check that aipython is now on the path
```

Out[12]:
```
['/Users/subhasish/Desktop/COMP 9414',
 '/Library/Frameworks/Python.framework/Versions/3.11/lib/python311.zip',
 '/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11',
 '/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/lib-dynloa
d',
 '',
 '/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packag
es',
 'aipython',
 'aipython',
 'aipython']
```

2. Representation of Day Times

Input and output are day time strings such as 'mon 10am' or a range of day time strings such as 'mon 10am-mon 4pm'.

The CSP will represent these as integer hour numbers in the week, ranging from 0 to 39.

The following code handles the conversion between day time strings and hour numbers.

In [13]:
```
# -*- coding: utf-8 -*-

""" day_time string format is a day plus time, e.g. Mon 10am, Tue 4pm, or just T
    if only day or time, returns day number or hour number only
    day_time strings are converted to and from integer hours in the week from 0
"""
class Day_Time():
    num_hours_in_day = 8
    num_days_in_week = 5

    def __init__(self):
        self.day_names = ['mon','tue','wed','thu','fri']
        self.time_names = ['9am','10am','11am','12pm','1pm','2pm','3pm','4pm']

    def string_to_week_hour_number(self, day_time_str):
        """ convert a single day_time into an integer hour in the week """
        value = None
        value_type = None
        day_time_list = day_time_str.split()
        if len(day_time_list) == 1:
            str1 = day_time_list[0].strip()
            if str1 in self.time_names: # this is a time
                value = self.time_names.index(str1)
                value_type = 'hour_number'
            else:
                value = self.day_names.index(str1) # this is a day
                value_type = 'day_number'
            # if not day or time, throw an exception
        else:
            value = self.day_names.index(day_time_list[0].strip())*self.num_hour
                + self.time_names.index(day_time_list[1].strip())
            value_type = 'week_hour_number'
        return (value_type, value)

    def string_to_number_set(self, day_time_list_str):
        """ convert a list of day-times or ranges 'Mon 9am, Tue 9am-Tue 4pm' int
            e.g. 'mon 9am-1pm, mon 4pm' -> [0,1,2,3,4,7]
        """
        number_set = set()
        type1 = None
        for str1 in day_time_list_str.lower().split(','):
            if str1.find('-') > 0:
                # day time range
                type1, v1 = self.string_to_week_hour_number(str1.split('-')[0].s
                type2, v2 = self.string_to_week_hour_number(str1.split('-')[1].s
                if type1 != type2: return None # error, types in range spec are
                number_set.update({n for n in range(v1, v2+1)})
            else:
                # single day time
                type2, value2 = self.string_to_week_hour_number(str1)
                if type1 != None and type1 != type2: return None # error: type i
                type1 = type2
                number_set.update({value2})
        return (type1, number_set)

    # convert integer hour in week to day time string
    def week_hour_number_to_day_time(self, week_hour_number):
        hour = self.day_hour_number(week_hour_number)
        day = self.day_number(week_hour_number)
        return self.day_names[day]+' '+self.time_names[hour]

    # convert integer hour in week to integer day and integer time in day
    def hour_day_split(self, week_hour_number):
        return (self.day_hour_number(week_hour_number), self.day_number(week_hou

    # convert integer hour in week to integer day in week
    def day_number(self, week_hour_number):
        return int(week_hour_number / self.num_hours_in_day)

    # convert integer hour in week to integer time in day
    def day_hour_number(self, week_hour_number):
        return week_hour_number % self.num_hours_in_day
```

```python
    def __repr__(self):
        day_hour_number = self.week_hour_number % self.num_hours_in_day
        day_number = int(self.week_hour_number / self.num_hours_in_day)
        return self.day_names[day_number]+' '+self.time_names[day_hour_number]

    #dt = Day_Time()
    #print(dt.string_to_week_hour_number('fri 9am'))
```

3. Constraint Satisfaction Problems with Costs over Tasks with Durations

Since AI Python does not provide the CSP class with an explicit cost, we implement our own class that extends `CSP`.

We also store the cost functions and the durations of all tasks explicitly in the CSP.

The durations of the tasks are used in the `hold` function to evaluate constraints.

In [147…

```python
#Domain Splitting with Arc Consistency cost defined
from cspProblem import CSP, Constraint

# We need to override Constraint, because tasks have durations
class Task_Constraint(Constraint):
    """A Task_Constraint consists of
    * scope: a tuple of variables
    * spec: text description of the constraint used in debugging
    * condition: a function that can applied to a tuple of values for the variab
    * durations: durations of all tasks
    * func_key: index to the function used to evaluate the constraint
    """
    def __init__(self, scope, spec, condition, durations, func_key):
        super().__init__(scope, condition, spec)
        self.scope = scope
        self.condition = condition
        self.durations = durations
        self.func_key = func_key

    def holds(self, assignment):
        """returns the value of Constraint con evaluated in assignment.

        precondition: all variables are assigned in assignment

        CSP has only binary constraints
        condition is in the form week_hour_number1, week_hour_number2
        add task durations as appropriate to evaluate condition
        """
        if self.func_key == 'before':
            # t1 ends before t2 starts, so we need add duration to t1 assignment
            ass0 = assignment[self.scope[0]] + self.durations[self.scope[0]]
            ass1 = assignment[self.scope[1]]
        elif self.func_key == 'after':
            # t2 ends before t1 starts so we need add duration to t2 assignment
            ass0 = assignment[self.scope[0]]
            ass1 = assignment[self.scope[1]] + self.durations[self.scope[1]]
        elif self.func_key == 'starts-at':
            # t1 starts exactly when t2 ends, so we need add duration to t2 assi
            ass0 = assignment[self.scope[0]]
            ass1 = assignment[self.scope[1]] + self.durations[self.scope[1]]
        else:
            return self.condition(*tuple(assignment[v] for v in self.scope))
        # condition here comes from get_binary_constraint
        return self.condition(*tuple([ass0, ass1]))

# implement nodes as CSP problems with cost functions
class CSP_with_Cost(CSP):
    """ cost_functions maps a CSP var, here a task name, to a list of functions
    def __init__(self, domains, durations, constraints, cost_functions, soft_day
        self.domains = domains
        self.variables = self.domains.keys()
        super().__init__("title of csp", self.variables, constraints)
        self.durations = durations
        self.cost_functions = cost_functions
        self.soft_day_time = soft_day_time
        self.soft_costs = soft_costs
        self.cost = self.calculate_cost()

    # specific to fuzzy scheduling CSP problems
    def calculate_cost(self):
        """ this is really a function f = path cost + heuristic to be used by th
        t_cost = 0
        # TODO: write cost function
        """This is the path cost + heuristic for fuzzy scheduling CSP."""
        # soft deadlines
        for var in self.domains:
            domain = self.domains[var]

            #soft deadline penalties
            if var in self.soft_day_time:
                deadline = Day_Time().string_to_week_hour_number(self.soft_day_t
                penalty_weight = int(self.soft_costs[var])

                penalties = []
                for start in domain:
                    finish = start + self.durations[var]
                    delay = max(0, finish - deadline[1])
                    penalties.append(delay * penalty_weight)

                if penalties:
                    #print(penalties)
                    t_cost += min(penalties)

        return t_cost

    def __repr__(self):
        """ string representation of an arc"""
        return "CSP_with_Cost("+str(list(self.domains.keys()))+':'+str(self.cost
```

In [151…

```
#Domain Splitting with Arc Consistency cost = 0
from cspProblem import CSP, Constraint

# We need to override Constraint, because tasks have durations
class Task_Constraint(Constraint):
    """A Task_Constraint consists of
    * scope: a tuple of variables
    * spec: text description of the constraint used in debugging
    * condition: a function that can applied to a tuple of values for the variab
    * durations: durations of all tasks
```

```python
    * func_key: index to the function used to evaluate the constraint
    """
    def __init__(self, scope, spec, condition, durations, func_key):
        super().__init__(scope, condition, spec)
        self.scope = scope
        self.condition = condition
        self.durations = durations
        self.func_key = func_key

    def holds(self, assignment):
        """returns the value of Constraint con evaluated in assignment.

        precondition: all variables are assigned in assignment

        CSP has only binary constraints
        condition is in the form week_hour_number1, week_hour_number2
        add task durations as appropriate to evaluate condition
        """
        if self.func_key == 'before':
            # t1 ends before t2 starts, so we need add duration to t1 assignment
            ass0 = assignment[self.scope[0]] + self.durations[self.scope[0]]
            ass1 = assignment[self.scope[1]]
        elif self.func_key == 'after':
            # t2 ends before t1 starts so we need add duration to t2 assignment
            ass0 = assignment[self.scope[0]]
            ass1 = assignment[self.scope[1]] + self.durations[self.scope[1]]
        elif self.func_key == 'starts-at':
            # t1 starts exactly when t2 ends, so we need add duration to t2 assi
            ass0 = assignment[self.scope[0]]
            ass1 = assignment[self.scope[1]] + self.durations[self.scope[1]]
        else:
            return self.condition(*tuple(assignment[v] for v in self.scope))
        # condition here comes from get_binary_constraint
        return self.condition(*tuple([ass0, ass1]))

# implement nodes as CSP problems with cost functions
class CSP_with_Cost(CSP):
    """ cost_functions maps a CSP var, here a task name, to a list of functions
    def __init__(self, domains, durations, constraints, cost_functions, soft_day
        self.domains = domains
        self.variables = self.domains.keys()
        super().__init__("title of csp", self.variables, constraints)
        self.durations = durations
        self.cost_functions = cost_functions
        self.soft_day_time = soft_day_time
        self.soft_costs = soft_costs
        self.cost = self.calculate_cost()

    # specific to fuzzy scheduling CSP problems
    def calculate_cost(self):
        """ this is really a function f = path cost + heuristic to be used by th
        t_cost = 0
        # TODO: write cost function
        """This is the path cost + heuristic for fuzzy scheduling CSP."""

        return t_cost

    def __repr__(self):
        """ string representation of an arc"""
        return "CSP_with_Cost("+str(list(self.domains.keys()))+':'+str(self.cost
```

This formulates a solver for a CSP with cost as a search problem, using domain splitting with arc consistency to define the successors of a node.

In [95]:
```
from cspConsistency import Con_solver, select, partition_domain
from searchProblem import Arc, Search_problem
from operator import eq, le, ge

# rewrites rather than extends Search_with_AC_from_CSP
class Search_with_AC_from_Cost_CSP(Search_problem):
    """ A search problem with domain splitting and arc consistency """
    def __init__(self, csp):
        self.cons = Con_solver(csp) # copy of the CSP with access to arc consist
        self.domains = self.cons.make_arc_consistent(csp.domains)
        self.constraints = csp.constraints
        self.cost_functions = csp.cost_functions
        self.durations = csp.durations
        self.soft_day_time = csp.soft_day_time
        self.soft_costs = csp.soft_costs
        csp.domains = self.domains # after arc consistency
        self.csp = csp

    def is_goal(self, node):
        """ node is a goal if all domains have exactly 1 element """
        return all(len(node.domains[var]) == 1 for var in node.domains)

    def start_node(self):
        return CSP_with_Cost(self.domains, self.durations, self.constraints,
                             self.cost_functions, self.soft_day_time, self.soft_

    def neighbors(self, node):
        """returns the neighboring nodes of node.
        """
        neighs = []
        var = select(x for x in node.domains if len(node.domains[x]) > 1) # chos
        if var:
            dom1, dom2 = partition_domain(node.domains[var])
            self.display(2, "Splitting", var, "into", dom1, "and", dom2)
            to_do = self.cons.new_to_do(var, None)
            for dom in [dom1,dom2]:
                newdoms = node.domains | {var: dom} # overwrite domain of var wi
                cons_doms = self.cons.make_arc_consistent(newdoms, to_do)
                if all(len(cons_doms[v]) > 0 for v in cons_doms):
                    # all domains are non-empty
                    # make new CSP_with_Cost node to continue the search
                    csp_node = CSP_with_Cost(cons_doms, self.durations, self.con
                             self.cost_functions, self.soft_day_time, self.soft_
                    neighs.append(Arc(node, csp_node))
                else:
                    self.display(2,"...",var,"in",dom,"has no solution")
        return neighs

    def heuristic(self, n):
        return n.cost
```

4. Fuzzy Scheduling Constraint Satisfaction Problems

The following code sets up a CSP problem from a given specification.

Hard (unary) domain constraints are applied to reduce the domains of the variables before the constraint solver runs.

In [96]:
```python
# domain specific CSP builder for week schedule
class CSP_builder():
    # list of text lines without comments and empty lines
    _, default_domain = Day_Time().string_to_number_set('mon 9am-fri 4pm') # sho

    # hard unary constraints: domain is a list of values, params is a single val
    # starts-before, ends-before (for starts-before duration should be 0)
    # vals in domain are actual task start/end date/time, so must be val <= what
    def apply_before(self, param_type, params, duration, domain):
        domain_orig = domain.copy()
        param_val = params.pop()
        for val in domain_orig: # val is week_hour_number
            val1 = val + duration
            h, d = Day_Time().hour_day_split(val1)
            if param_type == 'hour_number' and h > param_val:
                if val in domain: domain.remove(val)
            if param_type == 'day_number' and d > param_val:
                if val in domain: domain.remove(val)
            if param_type == 'week_hour_number' and val1 > param_val:
                if val in domain: domain.remove(val)
        return domain

    def apply_after(self, param_type, params, duration, domain):
        domain_orig = domain.copy()
        param_val = params.pop()
        for val in domain_orig: # val is week_hour_number
            val1 = val + duration
            h, d = Day_Time().hour_day_split(val1)
            if param_type == 'hour_number' and h < param_val:
                if val in domain: domain.remove(val)
            if param_type == 'day_number' and d < param_val:
                if val in domain: domain.remove(val)
            if param_type == 'week_hour_number' and val1 < param_val:
                if val in domain: domain.remove(val)
        return domain

    # day time range only
    # includes starts-in, ends-in
    # duration is 0 for starts-in, task duration for ends-in
    def apply_in(self, params, duration, domain):
        domain_orig = domain.copy()
        for val in domain_orig: # val is week_hour_number
            # task must be within range
            if val in domain and val+duration not in params:
                domain.remove(val)
        return domain

    # task must start at day/time
    def apply_at(self, param_type, param,domain):
        domain_orig = domain.copy()
        for val in domain_orig:
            h, d = Day_Time().hour_day_split(val)
            if param_type == 'hour_number' and param != h:
                if val in domain: domain.remove(val)
            if param_type == 'day_number' and param != d:
                if val in domain: domain.remove(val)
            if param_type == 'week_hour_number' and param != val:
                if val in domain: domain.remove(val)
        return domain

    # soft deadline constraints: return cost to break constraint
    # ends-by implementation: domain_dt is the day, hour from the domain
    # constr_dt is the soft const spec, dur is the duration of task
    # soft_cost is the unit cost of completion delay
    # so if the tasks starts on domain_dt, it ends on domain_dt+dur
    """
    <t> ends-by <day> <time>, both must be specified
    delay = day_hour(T2) - day_hour(T1) + 24*(D2 - D1),
    where day_hour(9am) = 0, day_hour(4pm) = 7
    """
    def ends_by(self, domain_dt, constr_dt_str, dur, soft_cost):
        param_type,params = Day_Time().string_to_number_set(constr_dt_str)
        param_val = params.pop()
        dom_h, dom_d = Day_Time().hour_day_split(domain_dt+dur)
        if param_type == 'week_hour_number':
            con_h, con_d = Day_Time().hour_day_split(param_val)
            return 0 if domain_dt + dur <= param_val else soft_cost*(dom_h - con
        else:
           return None # not good, must be day and time

    def no_cost(self, day ,hour):
        return 0

    # hard binary constraint, the rest are implemented as gt, lt, eq
    def same_day(self, week_hour1, week_hour2):
        h1, d1 = Day_Time().hour_day_split(week_hour1)
        h2, d2 = Day_Time().hour_day_split(week_hour2)
        return d1 == d2

    # domain is a list of values
    def apply_hard_constraint(self, domain, duration, spec):
        tokens = func_key = spec.split(' ')
        if len(tokens) > 1:
           func_key = spec.split(' ')[0].strip()
           param_type, params = Day_Time().string_to_number_set(spec[len(func_ke
           if func_key == 'starts-before':
              # duration is 0 for starts before, since we do not modify the time
              return self.apply_before(param_type, params, 0, domain)
           if func_key == 'ends-before':
               return self.apply_before(param_type, params, duration, domain)
           if func_key == 'starts-after':
              return self.apply_after(param_type,params,0,domain)
           if func_key == 'ends-after':
              return self.apply_after(param_type, params, duration, domain)
           if func_key == 'starts-in':
               return self.apply_in(params, 0, domain)
           if func_key == 'ends-in':
               return self.apply_in(params, duration, domain)
        else:
           # here we have task day or time, it has no func key so we need to par
           param_type,params = Day_Time().string_to_week_hour_number(spec)
           return self.apply_at(param_type, params, domain)

    def get_cost_function(self, spec):
        func_dict = {'ends-by':self.ends_by, 'no-cost':self.no_cost}
        return [func_dict[spec]]
```

```python
    # spec is the text of a constraint, e.g. 't1 before t2'
    # durations are durations of all tasks
    def get_binary_constraint(self, spec, durations):
        tokens = spec.strip().split(' ')
        if len(tokens) != 3: return None # error in spec
        # task1 relation task2
        fun_dict = {'before':le, 'after':ge, 'starts-at':eq, 'same-day':self.sam
        return Task_Constraint((tokens[0].strip(), tokens[2].strip()), spec, fun

    def get_CSP_with_Cost(self, input_lines):
        # Note: It would be more elegant to make task a class but AIpython is no
        # CSP_with_Cost inherits from CSP, which takes domains and constraints f
        domains = dict()
        constraints = []
        cost_functions = dict()
        durations = dict() # durations of tasks
        soft_day_time = dict() # day time specs of soft constraints
        soft_costs = dict() # costs of soft constraints

        for input_line in input_lines:
            func_spec = None
            input_line_tokens = input_line.strip().split(',')
            if len(input_line_tokens) != 2:
                return None # must have number of tokens = 2
            line_token1 = input_line_tokens[0].strip()
            line_token2 = input_line_tokens[1].strip()
            if line_token1 == 'task':
                tokens = line_token2.split(' ')
                if len(tokens) != 2:
                    return None # must have number of tokens = 3
                key = tokens[0].strip()
                # check the duration and save it
                duration = int(tokens[1].strip())
                if duration > Day_Time().num_hours_in_day:
                    return None
                durations[key] = duration
                # set zero cost function for this task as default, may add real
                cost_functions[key] = self.get_cost_function('no-cost')
                soft_costs[key] = '0'
                soft_day_time[key] = 'fri 4pm'
                # restrict domain to times that are within allowed range
                # that is start 9-5, start+duration in 9-5
                domains[key] = {x for x in self.default_domain \
                                if Day_Time().day_number(x+duration) \
                                == Day_Time().day_number(x)}
            elif line_token1 == 'domain':
                tokens = line_token2.split(' ')
                if len(tokens) < 2:
                    return None # must have number of tokens >= 2
                key = tokens[0].strip()
                # if soft constraint, it is handled differently from hard constr
                if tokens[1].strip() == 'ends-by':
                    # need to retain day time and cost from the line
                    # must have task, 'end-by', day, time, cost
                    # or task, 'end-by', day, cost
                    # or task, 'end-by', time, cost
                    if len(tokens) != 5:
                        return None
                    # get the rest of the line after 'ends-by'
                    soft_costs[key] = int(tokens[len(tokens)-1].strip()) # last
                    # pass the day time string to avoid passing param_type
                    day_time_str = tokens[2] + ' ' + tokens[3]
                    soft_day_time[key] = day_time_str
                    cost_functions[key] = self.get_cost_function(tokens[1].strip
                else:
                    # the rest of domain spec, after key, are hard unary domain
                    # func spec has day time, we also need duration
                    dur = durations[key]
                    func_spec = line_token2[len(key):].strip()
                    domains[key] = self.apply_hard_constraint(domains[key], dur,
            elif line_token1 == 'constraint': # all binary constraints
                constraints.append(self.get_binary_constraint(line_token2, durat
            else:
                return None

        return CSP_with_Cost(domains, durations, constraints, cost_functions, so

def create_CSP_from_spec(spec: str):
    input_lines = list()
    spec = spec.split('\n')
    # strip comments
    for input_line in spec:
        input_line = input_line.split('#')
        if len(input_line[0]) > 0:
            input_lines.append(input_line[0])
            print(input_line[0])
    # construct initial CSP problem
    csp = CSP_builder()
    csp_problem = csp.get_CSP_with_Cost(input_lines)
    return csp_problem
```

5. Greedy Search Constraint Solver using Domain Splitting and Arc Consistency

Create a GreedySearcher to search over the CSP.

The *cost* function for CSP nodes is used as the heuristic, but is actually a direct estimate of the total path cost function *f* used in A* Search.

In [97]:
```
from searchGeneric import AStarSearcher

class GreedySearcher(AStarSearcher):
    """ returns a searcher for a problem.
    Paths can be found by repeatedly calling search().
    """
    def add_to_frontier(self, path):
        """ add path to the frontier with the appropriate cost """
        # value = path.cost + self.problem.heuristic(path.end()) -- A* definitio
        value = path.end().cost
        self.frontier.add(path, value)
```

Run the GreedySearcher on the CSP derived from the sample input.

Note: The solution cost will always be 0 (which is wrong for the sample input) until you write the cost function in the cell above.

In [139…

```
# Sample problem specification

"""
sample_spec =
# two tasks with two binary constraints and soft deadlines
task, t1 3
task, t2 4
# two binary constraints
constraint, t1 before t2
constraint, t1 same-day t2
# domain constraint
domain, t2 mon
# soft deadlines
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
"""

"""
sample_spec =
#Tasks
task, t1 6
task, t2 8
task, t3 2
#Binaryconstraints
constraint, t3 after t1
constraint, t1 starts-at t3
constraint, t3 after t2
#Hard Domain constraints
domain, t1 mon
domain, t3 starts-before 1pm
#soft deadline constraints
domain, t1 ends-by fri 3pm 50
"""

"""
sample_spec =
#Tasks
task, t1 2
task, t2 1
task, t3 2
#Binaryconstraints
constraint, t3 starts-at t1
#Hard Domain constraints
domain, t3 starts-before wed 9am
#soft deadline constraints
domain, t2 ends-by thu 10am 20
domain, t3 ends-by thu 9am 20
domain, t3 ends-by wed 2pm 20
"""

"""
sample_spec =
#Tasks
task, t1 8
task, t2 6
task, t3 3
task, t4 7
#Binaryconstraints
constraint, t2 same-day t4
```

```
#Hard Domain constraints
domain, t4 starts-before 11am
#soft deadline constraints
domain, t2 ends-by thu 2pm 70
domain, t3 ends-by tue 12pm 70
domain, t3 ends-by tue 3pm 70
domain, t4 ends-by mon 11am 70
"""
#Above sample spec No solution

"""
sample_spec =
#Tasks
task, t1 1
task, t2 3
task, t3 3
task, t4 5
#Binaryconstraints
constraint, t2 after t3
constraint, t2 after t4
constraint, t2 same-day t3
#Hard Domain constraints
domain, t3 ends-after mon 12pm
domain, t1 thu
#soft deadline constraints
domain, t3 ends-by wed 9am 60
domain, t3 ends-by mon 4pm 60
domain, t2 ends-by tue 9am 60
domain, t1 ends-by wed 3pm 60
"""

"""
sample_spec =
#Tasks
task, t1 6
task, t2 6
task, t3 6
task, t4 3
task, t5 1
#Binaryconstraints
constraint, t5 same-day t1
constraint, t5 starts-at t1
constraint, t3 before t4
constraint, t4 starts-at t1
constraint, t2 starts-at t1
#Hard Domain constraints
domain, t1 ends-after wed 11am
domain, t2 ends-after fri 4pm
#soft deadline constraints
domain, t5 ends-by thu 11am 40
domain, t1 ends-by tue 2pm 40
"""
```

In [149…

```
#Sample tasks for different tasks
#specification: 2

"""
sample_spec =
#Tasks
task, t1 1
```

```
task, t2 3
#Binaryconstraints
constraint, t1 starts-at t2
constraint, t1 starts-at t2
#Hard Domain constraints
domain, t2 tue
#soft deadline constraints
domain, t1 ends-by fri 12pm 70
"""

#specification: 3

"""
sample_spec =
#Tasks
task, t1 4
task, t2 2
task, t3 3
#Binaryconstraints
constraint, t2 starts-at t1
constraint, t3 after t2
#Hard Domain constraints
domain, t1 ends-after mon 9am
#soft deadline constraints
domain, t1 ends-by fri 11am 30
domain, t3 ends-by fri 2pm 30
"""

#specification: 4

"""
sample_spec =
#Tasks
task, t1 2
task, t2 1
task, t3 3
task, t4 2
#Binaryconstraints
constraint, t1 starts-at t3
#Hard Domain constraints
domain, t2 ends-after 11am
#soft deadline constraints
domain, t4 ends-by thu 11am 60
"""


#specification: 5
"""
sample_spec =
#Tasks
task, t1 4
task, t2 1
task, t3 2
task, t4 3
task, t5 1
#Binaryconstraints
constraint, t5 same-day t1
constraint, t5 starts-at t1
constraint, t3 before t4
constraint, t4 starts-at t1
```

```
constraint, t2 starts-at t1
#Hard Domain constraints
domain, t1 ends-after wed 11am
#soft deadline constraints
domain, t5 ends-by thu 11am 40
domain, t1 ends-by tue 2pm 40
"""

#specification: 6

"""
sample_spec =
#Tasks
task, t1 2
task, t2 3
task, t3 1
task, t4 3
task, t5 2
task, t6 2
#Binaryconstraints
constraint, t6 after t5
constraint, t1 before t4
constraint, t6 before t2
constraint, t3 same-day t5
constraint, t1 before t2
#Hard Domain constraints
domain, t4 starts-before tue 4pm
domain, t2 ends-after tue 1pm
domain, t1 10am
#soft deadline constraints
domain, t4 ends-by tue 10am 80
"""

#specification for: 7  tasks


sample_spec = """
#Tasks
task, t1 4
task, t2 2
task, t3 2
task, t4 2
task, t5 3
task, t6 1
task, t7 2
#Binaryconstraints
constraint, t3 starts-at t4
constraint, t2 before t4
constraint, t3 starts-at t4
#Hard Domain constraints
domain, t6 starts-before fri 1pm
#soft deadline constraints
domain, t3 ends-by wed 4pm 90
domain, t7 ends-by thu 10am 90
domain, t1 ends-by mon 1pm 90
domain, t6 ends-by tue 12pm 90
domain, t5 ends-by thu 3pm 90
"""
```

In [154…

```
# display details (max_display_level = 0 turns off, 1 = basic information, 2 = d
Con_solver.max_display_level = 0
Search_with_AC_from_Cost_CSP.max_display_level = 2
GreedySearcher.max_display_level = 2

def test_csp_solver(searcher):
    final_path = searcher.search()
    if final_path == None:
        print('No solution')
    else:
        domains = final_path.end().domains
        result_str = ''
        for name, domain in domains.items():
            for n in domain:
                result_str += '\n'+str(name)+': '+Day_Time().week_hour_number_to
        print(result_str[1:]+'\ncost: '+str(final_path.end().cost))

csp_problem = create_CSP_from_spec(sample_spec)
solver = GreedySearcher(Search_with_AC_from_Cost_CSP(csp_problem))
test_csp_solver(solver)
```

```
task, t1 4
task, t2 2
task, t3 2
task, t4 2
task, t5 3
task, t6 1
task, t7 2
constraint, t3 starts-at t4
constraint, t2 before t4
constraint, t3 starts-at t4
domain, t6 starts-before fri 1pm
domain, t3 ends-by wed 4pm 90
domain, t7 ends-by thu 10am 90
domain, t1 ends-by mon 1pm 90
domain, t6 ends-by tue 12pm 90
domain, t5 ends-by thu 3pm 90
Splitting t1 into {0, 1, 2, 3, 8, 9, 10, 11, 16, 17} and {32, 33, 34, 35, 18, 19,
24, 25, 26, 27}
Expanding: title of csp with neighbors [title of csp --> title of csp, title of c
sp --> title of csp]
Splitting t1 into {0, 1, 2, 3, 8} and {9, 10, 11, 16, 17}
Expanding: title of csp --> title of csp with neighbors [title of csp --> title o
f csp, title of csp --> title of csp]
Splitting t1 into {0, 1} and {8, 2, 3}
Expanding: title of csp --> title of csp --> title of csp with neighbors [title o
f csp --> title of csp, title of csp --> title of csp]
Splitting t1 into {0} and {1}
Expanding: title of csp --> title of csp --> title of csp --> title of csp with n
eighbors [title of csp --> title of csp, title of csp --> title of csp]
Splitting t2 into {0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16} and {32, 33, 17, 1
8, 19, 20, 21, 24, 25, 26, 27, 28, 29}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp with neighbors [title of csp --> title of csp, title of csp --> title
of csp]
Splitting t2 into {0, 1, 2, 3, 4, 5} and {8, 9, 10, 11, 12, 13, 16}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp with neighbors [title of csp --> title of csp, title
of csp --> title of csp]
Splitting t2 into {0, 1, 2} and {3, 4, 5}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp with neighbors [title of csp --> tit
le of csp, title of csp --> title of csp]
Splitting t2 into {0} and {1, 2}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp with neighbors [tit
le of csp --> title of csp, title of csp --> title of csp]
Splitting t3 into {34, 35, 4, 5, 36, 37, 10, 11, 12} and {13, 18, 19, 20, 21, 26,
27, 28, 29}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp wi
th neighbors [title of csp --> title of csp, title of csp --> title of csp]
Splitting t3 into {34, 35, 4, 5} and {36, 37, 10, 11, 12}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp with neighbors [title of csp --> title of csp, title of csp --> ti
tle of csp]
Splitting t3 into {34, 35} and {4, 5}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp with neighbors [title of csp --> title of csp, ti
tle of csp --> title of csp]
Splitting t3 into {34} and {35}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp with neighbors [title of csp -->
title of csp, title of csp --> title of csp]
Splitting t5 into {0, 1, 2, 3, 4, 8, 9, 10, 11, 12, 16, 17} and {32, 33, 34, 35,
36, 18, 19, 20, 24, 25, 26, 27, 28}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp with neighbors
[title of csp --> title of csp, title of csp --> title of csp]
Splitting t5 into {0, 1, 2, 3, 4, 8} and {9, 10, 11, 12, 16, 17}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p with neighbors [title of csp --> title of csp, title of csp --> title of csp]
Splitting t5 into {0, 1, 2} and {8, 3, 4}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp with neighbors [title of csp --> title of csp, title of csp --
> title of csp]
Splitting t5 into {0} and {1, 2}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp with neighbors [title of csp --> title of cs
p, title of csp --> title of csp]
Splitting t6 into {0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17} and {3
2, 33, 34, 35, 36, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp with neighbors [title of csp
--> title of csp, title of csp --> title of csp]
Splitting t6 into {0, 1, 2, 3, 4, 5, 6, 8} and {9, 10, 11, 12, 13, 14, 16, 17}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp --> title of csp with neighb
ors [title of csp --> title of csp, title of csp --> title of csp]
Splitting t6 into {0, 1, 2, 3} and {8, 4, 5, 6}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp --> title of csp --> title o
f csp with neighbors [title of csp --> title of csp, title of csp --> title of cs
p]
Splitting t6 into {0, 1} and {2, 3}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp --> title of csp --> title o
f csp --> title of csp with neighbors [title of csp --> title of csp, title of cs
p --> title of csp]
Splitting t6 into {0} and {1}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
```

```
p --> title of csp --> title of csp --> title of csp --> title of csp --> title o
f csp --> title of csp --> title of csp with neighbors [title of csp --> title of
csp, title of csp --> title of csp]
Splitting t7 into {0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18} and {32, 3
3, 34, 35, 36, 37, 19, 20, 21, 24, 25, 26, 27, 28, 29}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp --> title of csp --> title o
f csp --> title of csp --> title of csp --> title of csp with neighbors [title of
csp --> title of csp, title of csp --> title of csp]
Splitting t7 into {0, 1, 2, 3, 4, 5, 8} and {9, 10, 11, 12, 13, 16, 17, 18}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp --> title of csp --> title o
f csp --> title of csp --> title of csp --> title of csp --> title of csp with ne
ighbors [title of csp --> title of csp, title of csp --> title of csp]
Splitting t7 into {0, 1, 2} and {8, 3, 4, 5}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp --> title of csp --> title o
f csp --> title of csp --> title of csp --> title of csp --> title of csp --> tit
le of csp with neighbors [title of csp --> title of csp, title of csp --> title o
f csp]
Splitting t7 into {0} and {1, 2}
Expanding: title of csp --> title of csp --> title of csp --> title of csp --> ti
tle of csp --> title of csp --> title of csp --> title of csp --> title of csp --
> title of csp --> title of csp --> title of csp --> title of csp --> title of cs
p --> title of csp --> title of csp --> title of csp --> title of csp --> title o
f csp --> title of csp --> title of csp --> title of csp --> title of csp --> tit
le of csp --> title of csp with neighbors [title of csp --> title of csp, title o
f csp --> title of csp]
Solution: title of csp --> title of csp --> title of csp --> title of csp --> tit
le of csp --> title of csp --> title of csp --> title of csp --> title of csp -->
title of csp --> title of csp --> title of csp --> title of csp --> title of csp
--> title of csp --> title of csp --> title of csp --> title of csp --> title of
csp --> title of csp --> title of csp --> title of csp --> title of csp --> title
of csp --> title of csp --> title of csp (cost: 25)
 26 paths have been expanded and 25 paths remain in the frontier
t1: mon 9am
t2: mon 9am
t3: fri 11am
t4: fri 9am
t5: mon 9am
t6: mon 9am
t7: mon 9am
cost: 0
```

6. Depth-First Search Constraint Solver

The Depth-First Constraint Solver in AIPython by default uses a random ordering of the variables in the CSP.

We need to modify this code to make it compatible with the arc consistency solver.

Run the solver by calling `dfs_solve1` (first solution) or `dfs_solve_all` (all solutions).

In [152…

```
num_expanded = 0
display = True

def dfs_solver(constraints, domains, context, var_order):
    """ generator for all solutions to csp
        context is an assignment of values to some of the variables
        var_order is a list of the variables in csp that are not in context
    """
    global num_expanded, display
    to_eval = {c for c in constraints if c.can_evaluate(context)}
    if all(c.holds(context) for c in to_eval):
        if var_order == []:
            print("Nodes expanded to reach solution:", num_expanded)
            yield context
        else:
            rem_cons = [c for c in constraints if c not in to_eval]
            var = var_order[0]
            for val in domains[var]:
                if display:
                    print("Setting", var, "to", val)
                num_expanded += 1
                yield from dfs_solver(rem_cons, domains, context|{var:val}, var_

def dfs_solve_all(csp, var_order=None):
    """ depth-first CSP solver to return a list of all solutions to csp """
    global num_expanded
    num_expanded = 0
    if var_order == None:    # use an arbitrary variable order
        var_order = list(csp.domains)
    return list(dfs_solver(csp.constraints, csp.domains, {}, var_order))

def dfs_solve1(csp, var_order=None):
    """ depth-first CSP solver """
    global num_expanded
    num_expanded = 0
    if var_order == None:    # use an arbitrary variable order
        var_order = list(csp.domains)
    for sol in dfs_solver(csp.constraints, csp.domains, {}, var_order):
        return sol  # return first one
```

Run the Depth-First Solver on the sample problem.

Note: Again there are no costs calculated.

In [153…

```
def test_dfs_solver(csp_problem):
    solution = dfs_solve1(csp_problem)
    if solution == None:
        print('No solution')
    else:
        result_str = ''
        for name in solution.keys():
            result_str += '\n'+str(name)+': '+Day_Time().week_hour_number_to_day
        print(result_str[1:])

# call the Depth-First Search solver
csp_problem = create_CSP_from_spec(sample_spec)
test_dfs_solver(csp_problem) # set display to True to see nodes expanded
```

```
task, t1 4
task, t2 2
task, t3 2
task, t4 2
task, t5 3
task, t6 1
task, t7 2
constraint, t3 starts-at t4
constraint, t2 before t4
constraint, t3 starts-at t4
domain, t6 starts-before fri 1pm
domain, t3 ends-by wed 4pm 90
domain, t7 ends-by thu 10am 90
domain, t1 ends-by mon 1pm 90
domain, t6 ends-by tue 12pm 90
domain, t5 ends-by thu 3pm 90
Setting t1 to 0
Setting t2 to 0
Setting t3 to 0
Setting t4 to 0
Setting t4 to 1
Setting t4 to 2
Setting t4 to 3
Setting t4 to 4
Setting t4 to 5
Setting t4 to 8
Setting t4 to 9
Setting t4 to 10
Setting t4 to 11
Setting t4 to 12
Setting t4 to 13
Setting t4 to 16
Setting t4 to 17
Setting t4 to 18
Setting t4 to 19
Setting t4 to 20
Setting t4 to 21
Setting t4 to 24
Setting t4 to 25
Setting t4 to 26
Setting t4 to 27
Setting t4 to 28
Setting t4 to 29
Setting t4 to 32
Setting t4 to 33
Setting t4 to 34
Setting t4 to 35
Setting t4 to 36
Setting t4 to 37
Setting t3 to 1
Setting t4 to 0
Setting t4 to 1
Setting t4 to 2
Setting t4 to 3
Setting t4 to 4
Setting t4 to 5
Setting t4 to 8
Setting t4 to 9
Setting t4 to 10
Setting t4 to 11
Setting t4 to 12
Setting t4 to 13
Setting t4 to 16
Setting t4 to 17
Setting t4 to 18
Setting t4 to 19
Setting t4 to 20
Setting t4 to 21
Setting t4 to 24
Setting t4 to 25
Setting t4 to 26
Setting t4 to 27
Setting t4 to 28
Setting t4 to 29
Setting t4 to 32
Setting t4 to 33
Setting t4 to 34
Setting t4 to 35
Setting t4 to 36
Setting t4 to 37
Setting t3 to 2
Setting t4 to 0
Setting t4 to 1
Setting t4 to 2
Setting t4 to 3
Setting t4 to 4
Setting t4 to 5
Setting t4 to 8
Setting t4 to 9
Setting t4 to 10
Setting t4 to 11
Setting t4 to 12
Setting t4 to 13
Setting t4 to 16
Setting t4 to 17
Setting t4 to 18
Setting t4 to 19
Setting t4 to 20
Setting t4 to 21
Setting t4 to 24
Setting t4 to 25
Setting t4 to 26
Setting t4 to 27
Setting t4 to 28
Setting t4 to 29
Setting t4 to 32
Setting t4 to 33
Setting t4 to 34
Setting t4 to 35
Setting t4 to 36
Setting t4 to 37
Setting t3 to 3
Setting t4 to 0
Setting t4 to 1
Setting t4 to 2
Setting t4 to 3
Setting t4 to 4
Setting t4 to 5
Setting t4 to 8
Setting t4 to 9
Setting t4 to 10
Setting t4 to 11
Setting t4 to 12
Setting t4 to 13
Setting t4 to 16
Setting t4 to 17
Setting t4 to 18
Setting t4 to 19
Setting t4 to 20
Setting t4 to 21
Setting t4 to 24
Setting t4 to 25
Setting t4 to 26
Setting t4 to 27
Setting t4 to 28
Setting t4 to 29
Setting t4 to 32
Setting t4 to 33
Setting t4 to 34
Setting t4 to 35
Setting t4 to 36
Setting t4 to 37
Setting t3 to 4
Setting t4 to 0
Setting t4 to 1
Setting t4 to 2
Setting t5 to 0
Setting t6 to 0
Setting t7 to 0
Nodes expanded to reach solution: 133
t1: mon 9am
t2: mon 9am
t3: mon 1pm
t4: mon 11am
t5: mon 9am
t6: mon 9am
t7: mon 9am
```

7. Depth-First Search Constraint Solver using Forward Checking with MRV Heuristic

The Depth-First Constraint Solver in AIPython by default uses a random ordering of the variables in the CSP.

We redefine the `dfs_solver` methods to implement the MRV (Minimum Remaining Values) heuristic using forward checking.

Because the AIPython code is designed to manipulate domain sets, we also need to redefine `can_evaluate` to handle partial assignments.

In [90]:
```python
num_expanded = 0
display = True

def can_evaluate(c, assignment):
    """ assignment is a variable:value dictionary
        returns True if the constraint can be evaluated given assignment
    """
    return assignment != {} and all(v in assignment.keys() and type(assignment[v

def mrv_dfs_solver(constraints, domains, context, var_order):
    """ generator for all solutions to csp.
        context is an assignment of values to some of the variables.
        var_order is a list of the variables in csp that are not in context.
    """
    global num_expanded, display
    if display:
        print("Context", context)
    to_eval = {c for c in constraints if can_evaluate(c, context)}
    if all(c.holds(context) for c in to_eval):
        if var_order == []:
            print("Nodes expanded to reach solution:", num_expanded)
            yield context
        else:
            rem_cons = [c for c in constraints if c not in to_eval] # constraint
            var = var_order[0]
            rem_vars = var_order[1:]
            for val in domains[var]:
                if display:
                    print("Setting", var, "to", val)
                num_expanded += 1
                rem_context = context|{var:val}
                # apply forward checking on remaining variables
                if len(var_order) > 1:
                    rem_vars_original = list((v, list(domains[v].copy())) for v
                    if display:
                        print("Original domains:", rem_vars_original)
                    # constraints that can't already be evaluated in rem_cons
                    rem_cons_ff = [c for c in constraints if c in rem_cons and n
                    for rem_var in rem_vars:
                        # constraints that can be evaluated by adding a value of
                        any_value = list(domains[rem_var])[0]
                        rem_to_eval = {c for c in rem_cons_ff if can_evaluate(c,
                        # new domain for rem_var are the values for which all ne
                        rem_vals = domains[rem_var].copy()
                        for rem_val in domains[rem_var]:
                            # no constraint with rem_var in the existing context
                            for c in rem_to_eval:
                                if not c.holds(rem_context|{rem_var: rem_val}):
                                    if rem_val in rem_vals:
                                        rem_vals.remove(rem_val)
                        domains[rem_var] = rem_vals
                        # order remaining variables by MRV
                        rem_vars.sort(key=lambda v: len(domains[v]))
                    if display:
                        print("After forward checking:", list((v, domains[v]) fo
                if rem_vars == [] or all(len(domains[rem_var]) > 0 for rem_var i
                    yield from mrv_dfs_solver(rem_cons, domains, context|{var:va
                # restore original domains if changed through forward checking
                if len(var_order) > 1:
                    if display:
                        print("Restoring original domain", rem_vars_original)
                    for (v, domain) in rem_vars_original:
                        domains[v] = domain
            if display:
                print("Nodes expanded so far:", num_expanded)

def mrv_dfs_solve_all(csp, var_order=None):
    """ depth-first CSP solver to return a list of all solutions to csp """
    global num_expanded
    num_expanded = 0
    if var_order == None:    # order variables by MRV
        var_order = list(csp.domains)
        var_order.sort(key=lambda var: len(csp.domains[var]))
    return list(mrv_dfs_solver(csp.constraints, csp.domains, {}, var_order))

def mrv_dfs_solve1(csp, var_order=None):
    """ depth-first CSP solver """
    global num_expanded
    num_expanded = 0
    if var_order == None:    # order variables by MRV
        var_order = list(csp.domains)
        var_order.sort(key=lambda var: len(csp.domains[var]))
    for sol in mrv_dfs_solver(csp.constraints, csp.domains, {}, var_order):
        return sol  # return first one
```

Run this solver on the sample problem.

Note: Again there are no costs calculated.

In [110…

```
def test_mrv_dfs_solver(csp_problem):
    solution = mrv_dfs_solve1(csp_problem)
    if solution == None:
        print('No solution')
    else:
        result_str = ''
        for name in solution.keys():
            result_str += '\n'+str(name)+': '+Day_Time().week_hour_number_to_day
        print(result_str[1:])

# call the Depth-First MRV Search solver
csp_problem = create_CSP_from_spec(sample_spec)
test_mrv_dfs_solver(csp_problem) # set display to True to see nodes expanded
```

```
task, t1 2
task, t2 3
task, t3 1
task, t4 3
task, t5 2
task, t6 2
constraint, t6 after t5
constraint, t1 before t4
constraint, t6 before t2
constraint, t3 same-day t5
constraint, t1 before t2
domain, t4 starts-before tue 4pm
domain, t2 ends-after tue 1pm
domain, t1 10am
domain, t4 ends-by tue 10am 80
Context {}
Setting t1 to 1
Original domains: [('t4', [0, 1, 2, 3, 4, 8, 9, 10, 11, 12]), ('t2', [9, 10, 11,
12, 16, 17, 18, 19, 20, 24, 25, 26, 27, 28, 32, 33, 34, 35, 36]), ('t5', [0, 1,
2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29,
32, 33, 34, 35, 36, 37]), ('t6', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17,
18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t3', [0, 1,
2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26,
27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
After forward checking: [('t4', {3, 4, 8, 9, 10, 11, 12}), ('t2', {9, 10, 11, 12,
16, 17, 18, 19, 20, 24, 25, 26, 27, 28, 32, 33, 34, 35, 36}), ('t5', {0, 1, 2, 3,
4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 3
3, 34, 35, 36, 37}), ('t6', {0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18,
19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37}), ('t3', {0, 1, 2, 3,
4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 2
8, 29, 30, 32, 33, 34, 35, 36, 37, 38})]
Context {'t1': 1}
Setting t4 to 3
Original domains: [('t2', [9, 10, 11, 12, 16, 17, 18, 19, 20, 24, 25, 26, 27, 28,
32, 33, 34, 35, 36]), ('t5', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18,
19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t6', [0, 1, 2, 3,
4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 3
3, 34, 35, 36, 37]), ('t3', [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 1
7, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
After forward checking: [('t2', {9, 10, 11, 12, 16, 17, 18, 19, 20, 24, 25, 26, 2
7, 28, 32, 33, 34, 35, 36}), ('t5', {0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16,
17, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37}), ('t6', {0,
1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28,
29, 32, 33, 34, 35, 36, 37}), ('t3', {0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13,
14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 3
7, 38})]
Context {'t1': 1, 't4': 3}
Setting t2 to 9
Original domains: [('t5', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 1
9, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t6', [0, 1, 2, 3,
4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 3
3, 34, 35, 36, 37]), ('t3', [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 1
7, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
After forward checking: [('t6', {0, 1, 2, 3, 4, 5}), ('t5', {0, 1, 2, 3, 4, 5, 8,
9, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 3
5, 36, 37}), ('t3', {0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 1
9, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38})]
Context {'t1': 1, 't4': 3, 't2': 9}
Setting t6 to 0
Original domains: [('t5', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 1
9, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t3', [0, 1, 2, 3,
4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 2
8, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
After forward checking: [('t5', set()), ('t3', {0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 1
1, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 3
4, 35, 36, 37, 38})]
Restoring original domain [('t5', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 1
7, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t3', [0,
1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 2
6, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
Setting t6 to 1
Original domains: [('t5', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 1
9, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t3', [0, 1, 2, 3,
4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 2
8, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
After forward checking: [('t5', []), ('t3', [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 1
2, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 3
5, 36, 37, 38])]
Restoring original domain [('t5', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 1
7, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t3', [0,
1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 2
6, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
Setting t6 to 2
Original domains: [('t5', [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 16, 17, 18, 1
9, 20, 21, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37]), ('t3', [0, 1, 2, 3,
4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 2
8, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
After forward checking: [('t5', [0]), ('t3', [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11,
12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 3
5, 36, 37, 38])]
Context {'t1': 1, 't4': 3, 't2': 9, 't6': 2}
Setting t5 to 0
Original domains: [('t3', [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17,
18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38])]
After forward checking: [('t3', [0, 1, 2, 3, 4, 5, 6])]
Context {'t1': 1, 't4': 3, 't2': 9, 't6': 2, 't5': 0}
Setting t3 to 0
Context {'t1': 1, 't4': 3, 't2': 9, 't6': 2, 't5': 0, 't3': 0}
Nodes expanded to reach solution: 8
t1: mon 10am
t4: mon 12pm
t2: tue 10am
t6: mon 11am
t5: mon 9am
t3: mon 9am
```

Assignment

Name: Subhasish Bhandaro

zID: z5528648

Question 1 (4 marks)

Consider the search spaces for the fuzzy scheduling CSP solvers – domain splitting with arc consistency and the DFS solver (without forward checking).

- Describe the search spaces in terms of start state, successor functions and goal state(s) (1 mark)
- What is the branching factor and maximum depth to find any solution for the two algorithms (ignoring costs)? (1 mark)
- What is the worst case time and space complexity of the two search algorithms? (1 mark)
- Give one example of a fuzzy scheduling problem that is *easier* for the domain splitting with arc consistency solver than it is for the DFS solver, and explain why (1 mark)

For the second and third part-questions, give the answer in a general form in terms of fuzzy scheduling CSP size parameters.

Answers for Question 1

1.

i) Domain Splitting with Arc consistency:

Start state: Every variable has an intial domain of time slot for all weekdays from 9AM to 4PM (Mon-Fri)

Successor functions: An unassigned variable is picked and its domain is split into two equal subsets, thereby creating child CSPs. To enforce Arc consistency, prune inconsistent domain values (ones that can't satisfy the binary constraints)

Goal State: Every variable has a single value and all the binary and hard constraints are satisfied

ii) DFS Solver (without forward checking):

Start state: All the variables are unassigned and the domain intact

Successor functions: select one unassigned varible randomly and assign it a value from its domain. Here, every possible assignment sequence is generated.

Goal State: State in which every variable has a single value and all the binary and hard constraints are satisfied

2.

If d = max domain size n = no. of variables

i) Domain Splitting with Arc consistency:

Branching factor = 2 for binary domain splits (and k, where k is no. of domain partitions in a split) maximum depth to find any solution = n logk d per variable(k = 2 for binary domain splits)

ii) DFS Solver (without forward checking):

Branching factor = d (domain size per variable) maximum depth to find any solution = n (depth is the no. of variables as each variable gets assigned a value at each step)

3.

i) Domain Splitting with Arc consistency: e : no of arcs / binary constrains d : max domain size n : no. of variables Worst case time complexity : O(e*d^(n+3))

Arc consistency at each node : O(e*d^3) and no. of nodes : O(d^n)

Worst case space complexity :O(nd)

ii) DFS Solver (without forward checking):

Worst case time complexity : O(d^n), explores all assignments in worst case

Worst case space complexity : O(n)

4.

Lets say we have to schedule 3 classes (X, Y, Z) into 5 time slots (All weekdays at 10 AM is a slot) with the following constraints: a) X & Y must be on different days a) X should be on either Mon or Tue Each variable has the domain of {Mon 10am, Tue 10 am, Wed 10 am, Thursday 10am, Fri 10}

i) Arc consistency prunes incompatible combinations immediately When the first constraint is applied all the pairs (Mon, Mon), (Tue, Tue), (Wed, Wed)..... are pruned It also narrows down X's domain to Mon or Tue before any search

ii) DFS takes longer to reach the goal state:

DFS explores every possible combination of days for X, Y, Z : 5^3 = 125 combinations This finds out the conflicts like Both X and Y on Mon only after going deep No pruning occurs and a lot of options are explored unncecessarily

Thus Domain splitting with Arc Consitency narrows to a small number of feasible time-slot ranges very quickly where DFS explores all 125 comibinations.

Question 2 (5 marks)

Define the *cost* function for a fuzzy scheduling CSP (i.e. a node in the search space for domain splitting and arc consistency) as the total cost of the soft deadline constraints violated for all of the variables, assuming that each variable is assigned one of the best possible values from its domain, where a "best" value for a variable *v* is one that has the lowest cost to violate the soft deadline constraint (if any) for that variable *v*.

- Implement the cost function in the indicated cell and place a copy of the code below (3 marks)
- What is its computational complexity (give a general form in terms of fuzzy scheduling CSP size parameters)? (1 mark)

- Show that the cost function *f* never decreases along a path, and explain why this means the search algorithm is optimal (1 mark)

```
In [16]: # Code for Question 2
         # Place a copy of your code here and run it in the relevant cell
         def calculate_cost(self):
             """ this is really a function f = path cost + heuristic to be used by the co
             t_cost = 0
             # TODO: write cost function
             """This is the path cost + heuristic for fuzzy scheduling CSP."""
             # soft deadlines
             for var in self.domains:
                 domain = self.domains[var]

                 #soft deadline penalties
                 if var in self.soft_day_time:
                     deadline = Day_Time().string_to_week_hour_number(self.soft_day_time[
                     penalty_weight = int(self.soft_costs[var])

                     penalties = []
                     for start in domain:
                         finish = start + self.durations[var]
                         delay = max(0, finish - deadline[1])
                         penalties.append(delay * penalty_weight)

                     if penalties:
                         #print(penalties)
                         t_cost += min(penalties)

             return t_cost
```

Answers for Question 2

Write the other answers here.

ii) Time complexity of this cost function is O(n*d) where, n = no. of variables and d = max domain size per variable Because for each variable we check each value in its domain to find the minimum violation cost.

Space complexity is O(d) in worst case as penalties temporarily store upto d elements per variable.

iii)

In Domain splitting, as search progresses the domain of each variable reduces.

f (n) = g(n) + h(n), where g(n) : cost of path from start to n h(n) : estimate from n to goal As we go along the path (down the tree), the search algo expands nodes in order of increasing cost and as cost never decreases while expanding, once a solution is found, no other unexpanded node would have a lower cost. As f is non-decrasing along any root to leaf path the solution obtained is optimal.

Question 3 (4 marks)

Conduct an empirical evaluation of the domain splitting CSP solver using the cost function defined as above compared to using no cost function (i.e. the zero cost function, as originally defined in the above cell). Use the *average number of nodes expanded* as a metric to compare the two algorithms.

- Write a function `generate_problem(n)` that takes an integer `n` and generates a problem specification with `n` tasks and a random set of hard constraints and soft deadline constraints in the correct format for the constraint solvers (2 marks)

Run the CSP solver (with and without the cost function) over a number of problems of size `n` for a range of values of `n`.

- Plot the performance of the two constraint solving algorithms on the above metric against `n` (1 mark)
- Quantify the performance gain (if any) achieved by the use of this cost function (1 mark)

In [60]:
```
# Code for Question 3
# Place your code here

###### two tasks with two binary constraints and soft deadlines
#task, t1 3
#task, t2 4
###### two binary constraints
#constraint, t1 before t2
#constraint, t1 same-day t2
###### hard domain constraint
#domain, t2 mon
###### soft deadline constraints
#domain, t1 ends-by mon 3pm 10
#domain, t2 ends-by mon 3pm 10

import random

def generate_problem(n, seed=None):
    """
      Generate a random fuzzy scheduling CSP specification with n tasks.
    - Adds durations between 1 and 8 for each task
    - Returns the specification as a list of lines
    """
    if seed is not None:
        random.seed(seed)

    tasks = [f"t{i}" for i in range(1, n+1)]
    days = ["mon", "tue", "wed", "thu", "fri"]
    times = ["9am", "10am", "11am", "12pm", "1pm", "2pm", "3pm", "4pm"]

    binary_types = ["before", "after", "same-day", "starts-at"]
    hard_domain_types = ["day", "time", "starts-before", "starts-after", "ends-b

    spec_lines = []

    # **********Task durations**********
    spec_lines.append("#Tasks")
    for t in tasks:
        dur = random.randint(1, 4)
        spec_lines.append(f"task, {t} {dur}")

    # **********Binary Constraints**********
    num_bin = random.randint(1, n)
    spec_lines.append("#Binaryconstraints")
    for _ in range(num_bin):
        t1, t2 = random.sample(tasks, 2)
        btype = random.choice(binary_types)
        spec_lines.append(f"constraint, {t1} {btype} {t2}")

    # **********Hard Domain Constraints**********
    num_hard = random.randint(1, n)
    spec_lines.append("#Hard Domain constraints")
    for _ in range(num_hard):
        tsk = random.choice(tasks)
        htype = random.choice(hard_domain_types)

        if htype == "day":
            d = random.choice(days)
            spec_lines.append(f"domain, {tsk} {d}")

        elif htype == "time":
            tym = random.choice(times)
            spec_lines.append(f"domain, {tsk} {tym}")

        elif htype in ["starts-before", "starts-after", "ends-before", "ends-aft
            d = random.choice(days)
            tym = random.choice(times)
            spec_lines.append(f"domain, {tsk} {htype} {d} {tym}")

        elif htype in ["starts-in", "ends-in"]:
            d1, d2 = random.sample(days, 2)
            t1, t2 = random.sample(times, 2)
            spec_lines.append(f"domain, {tsk} {htype} {d1} {t1}-{d2} {t2}")

        elif htype == "starts-before<time>":
            tym = random.choice(times)
            spec_lines.append(f"domain, {tsk} starts-before {tym}")

        elif htype == "ends-before<time>":
            tym = random.choice(times)
            spec_lines.append(f"domain, {tsk} ends-before {tym}")

        elif htype == "starts-after<time>":
            tym = random.choice(times)
            spec_lines.append(f"domain, {tsk} starts-after {tym}")

        elif htype == "ends-after<time>":
            tym = random.choice(times)
            spec_lines.append(f"domain, {tsk} ends-after {tym}")

    # **********Soft Deadline Constraints**********
    num_soft = random.randint(1, n)
    spec_lines.append("#soft deadline constraints")
    #cost = random.randint(1, 10)
    cost = random.randrange(10, 100, 10)
    for _ in range(num_soft):
        tsk = random.choice(tasks)
        d = random.choice(days)
        tm = random.choice(times)
        spec_lines.append(f"domain, {tsk} ends-by {d} {tm} {cost}")

    return spec_lines

#***********Generation of random problem Specs**************
for i in range(2,9):
    spec = generate_problem(i, seed=None) # Run this to generate the task and no
    print("\n#specification for:",i," tasks\n")
    for line in spec:
        print(line)
```

```
#specification for: 2  tasks

#Tasks
task, t1 2
task, t2 3
#Binaryconstraints
constraint, t2 same-day t1
#Hard Domain constraints
domain, t2 starts-before tue 10am
domain, t2 ends-in tue 10am-wed 9am
#soft deadline constraints
domain, t1 ends-by wed 9am 90

#specification for: 3  tasks

#Tasks
task, t1 4
task, t2 1
task, t3 3
#Binaryconstraints
constraint, t2 after t3
constraint, t1 after t2
#Hard Domain constraints
domain, t1 ends-before 2pm
domain, t3 ends-after 10am
#soft deadline constraints
domain, t3 ends-by thu 1pm 40
domain, t1 ends-by thu 9am 40

#specification for: 4  tasks

#Tasks
task, t1 2
task, t2 2
task, t3 3
task, t4 4
#Binaryconstraints
constraint, t4 after t1
constraint, t3 after t2
#Hard Domain constraints
domain, t2 ends-after 2pm
domain, t3 thu
domain, t1 starts-after thu 12pm
domain, t1 starts-after wed 1pm
#soft deadline constraints
domain, t1 ends-by wed 11am 70
domain, t3 ends-by tue 3pm 70

#specification for: 5  tasks

#Tasks
task, t1 2
task, t2 4
task, t3 4
task, t4 3
task, t5 4
#Binaryconstraints
constraint, t2 starts-at t1
#Hard Domain constraints
domain, t3 ends-in fri 1pm-tue 9am
```

```
domain, t3 starts-after mon 12pm
domain, t5 thu
domain, t2 ends-after fri 2pm
domain, t2 starts-before fri 10am
#soft deadline constraints
domain, t4 ends-by mon 9am 60
domain, t3 ends-by thu 12pm 60

#specification for: 6  tasks

#Tasks
task, t1 2
task, t2 3
task, t3 1
task, t4 3
task, t5 2
task, t6 2
#Binaryconstraints
constraint, t6 after t5
constraint, t1 before t4
constraint, t6 before t2
constraint, t3 same-day t5
constraint, t1 before t2
#Hard Domain constraints
domain, t4 starts-before tue 4pm
domain, t2 ends-after tue 1pm
domain, t1 10am
#soft deadline constraints
domain, t4 ends-by tue 10am 80

#specification for: 7  tasks

#Tasks
task, t1 4
task, t2 2
task, t3 2
task, t4 2
task, t5 3
task, t6 1
task, t7 2
#Binaryconstraints
constraint, t3 starts-at t4
constraint, t2 before t4
constraint, t3 starts-at t4
#Hard Domain constraints
domain, t6 starts-before fri 1pm
#soft deadline constraints
domain, t3 ends-by wed 4pm 90
domain, t7 ends-by thu 10am 90
domain, t1 ends-by mon 1pm 90
domain, t6 ends-by tue 12pm 90
domain, t5 ends-by thu 3pm 90

#specification for: 8  tasks

#Tasks
task, t1 2
task, t2 4
task, t3 4
task, t4 1
```

```
task, t5 2
task, t6 3
task, t7 2
task, t8 3
#Binaryconstraints
constraint, t2 same-day t6
constraint, t3 starts-at t6
constraint, t4 same-day t8
constraint, t4 starts-at t7
constraint, t6 before t5
constraint, t4 same-day t1
constraint, t1 starts-at t3
#Hard Domain constraints
domain, t6 starts-before fri 9am
domain, t3 starts-after 12pm
domain, t5 ends-in mon 9am-fri 10am
domain, t1 starts-before 1pm
domain, t6 starts-after 3pm
domain, t1 ends-after 1pm
#soft deadline constraints
domain, t2 ends-by tue 2pm 30
domain, t1 ends-by thu 1pm 30
domain, t3 ends-by tue 10am 30
domain, t5 ends-by mon 9am 30
```

Answers for Question 3

Write the other answers here.

2.

specification: 4 tasks

Tasks

task, t1 1 task, t2 3 task, t3 3 task, t4 5

Binaryconstraints

constraint, t2 after t3 constraint, t2 after t4 constraint, t2 same-day t3

Hard Domain constraints

domain, t3 ends-after mon 12pm domain, t1 thu

soft deadline constraints

domain, t3 ends-by wed 9am 60 domain, t3 ends-by mon 4pm 60 domain, t2 ends-by tue 9am 60 domain, t1 ends-by wed 3pm 60

Without cost 8 paths have been expanded and 7 paths remain in the frontier t1: thu 9am t2: fri 12pm t3: fri 9am t4: mon 9am cost: 0

With cost 7 paths have been expanded and 6 paths remain in the frontier t1: thu 9am t2: tue 12pm t3am t4: mon 9am costaskst: 780

specification: 5

sample_spec = """

Tasks

task, t1 4 task, t2 1 task, t3 2 task, t4 3 task, t5 1

Binaryconstraints

constraint, t5 same-day t1 constraint, t5 starts-at t1 constraint, t3 before t4 constraint, t4 starts-at t1 constraint, t2 starts-at t1

Hard Domain constraints

domain, t1 ends-after wed 11am

soft deadline constraints

domai without costn, t5 ends-by thu 11am 40 domain, t1 ends-by tue 2pm 40 """

5 paths have been expanded and 4 paths remain in the frontier t1: With Costwed 9am t2: wed 1pm t3: mon 9am t4: wed 1pm t5: wed 1pm cost: 0

5 paths have been expanded and 4 paths remain in the front

With the above data it seems like the DFS domain splitting with cost performs better than the DFS Domain splitting without cost.

Efficiency = (8-7)/7 * 100 =14.286 ier t1: wed 9am t2: wed 1pm t3: mon 9am t4: wed 1pm t5: wed 1pm cost: 280

Question 4 (5 marks)

Compare the Depth-First Search (DFS) solver to the Depth-First Search solver using forward checking with Minimum Remaining Values heuristic (DFS-MRV). For this question, ignore the costs associated with the CSP problems.

- What is the worst case time and space complexity of each algorithm (give a general form in terms of fuzzy scheduling problem sizes)? (1 mark)
- What are the properties of the search algorithms (completeness, optimality)? (1 mark)
- Give an example of a problem that is *easier* for the DFS-MRV solver than it is for the DFS solver, and explain why (1 mark)
- Empirically compare the quality of the first solution found by DFS and DFS-MRV compared to the optimal solution (1 mark)
- Empirically compare DFS-MRV with DFS in terms of the number of nodes expanded (1 mark)

For the empirical evaluations, run the two algorithms on a variety of problems of size `n` for varying `n`. Note that the domain splitting CSP solver with costs should always find an optimal solution.

In [15]:
```
# Code for Question 4
# Place your code here
```

Answers for Question 4

If you want to submit additional code, put this at the end of the notebook. Here just give the answers (including plots or tables).

1.

i) Depth-First Search (DFS) solver (without forward checking) d : Domain of the tasks n : no. of variables / tasks Worst Case Time Complexity: O(d^n)

Worst Case Space Complexity: O(n)

ii) Depth-First Search solver using forward checking with Minimum Remaining Values heuristic (DFS-MRV): Worst Case Time Complexity: O(d^n)

Worst Case Space Complexity: O(nd)

2.

i) Completeness : Both the solvers explore all possible solutions if necessary and would provide a solution if there is one. ii) Optimality : These two solvers both stop at the first solution found, minimal cost is not considered for these two algorithms.

3.

Tasks : T1, T2, T3 Domain : T1, T2, T3 : (1pm, 2pm, 3pm, 4pm, 5pm)

Constraints : T1 != T2 != T3 T1 must finish before T2 starts T1 must finish before T3 starts T1 can start either at 1 or 2

DFS (without forward checking) might pick variables in any orders , lets assume the order as (T2, T3, T1) It assigns T2 =1, T3=1, T1 = 2 and then T2 = 2. DFS backtracks repeatedly, exploring many inconsistent T2 and T3 combinations before fixing the root cause that T1 has to be the first to start.

Whereas with the MRV heuristics, T1's domain is smallest hence picks T1 first. With forward checking T2 and T3 should be greater than 1 due to the finish before constraints. Thus DFS with MRV and FC solves very quickly if the variables are highly constrained as the most constrained variable is chosen.

4.

Based on the below sample spec:

Tasks

task, t1 4 task, t2 2 task, t3 2 task, t4 2 task, t5 3 task, t6 1 task, t7 2

Binaryconstraints

constraint, t3 starts-at t4 constraint, t2 before t4 constraint, t3 starts-at t4

Hard Domain constraints

domain, t6 starts-before fri 1pm

soft deadline constraints

domain, t3 ends-by wed 4pm 90 domain, t7 ends-by thu 10am 90 domain, t1 ends-by mon 1pm 90 domain, t6 ends-by tue 12pm 90 domain, t5 ends-by thu 3pm 90

Output of Domain Splitting without cost :

26 paths have been expanded and 25 paths remain in the frontier t1: mon 9am t2: mon 9am t3: fri 11am t4: fri 9am t5: mon 9am t6: mon 9am t7: mon 9am cost: 0

Output of Domain Splitting with cost : 26 paths have been expanded and 25 paths remain in the frontier t1: mon 9am t2: mon 9am t3: mon 1pm t4: mon 11am t5: mon 9am t6: mon 9am t7: mon 9am cost: 0

Output of DFS Solver: Nodes expanded to reach solution: 133 t1: mon 9am t2: mon 9am t3: mon 1pm t4: mon 11am t5: mon 9am t6: mon 9am t7: m Output of DFS Solver with MRV and FC Nodes expanded to reach solution: 7 t1: mon 9am t5: mon 9am t2: mon

9am t4: mon 11am t3: mon 1pm t7: mon 9am t6: m The above shows that optimal solution with the cost is best solution possible for the problem.on 9am on 9am

6.

No. of Assignments tasks = 2 tasks No. of nodes explored in DFS = 62 No. of nodes explored in DFS with MRV and forward checking = 2

No. of Assignments tasks = 3 tasks No. of nodes explored in DFS = 12 No. of nodes explored in DFS with MRV and forward checking = 3 1No. of Assignments tasks = 4 tasks No. of nodes explored in DFS = 2347 No. of nodes explored in DFS with MRV and forward checking = 4

No. of Assignments tasks = 5 tasks No. of nodes explored in DFS = 55 No. of nodes explored in DFS with MRV and forward checking = 5

No. of Assignments tasks = 6 tasks No. of nodes explored in DFS = 11 No. of nodes explored in DFS with MRV and forward checking = 8

No. of Assignments tasks = 7 tasks No. of nodes explored in DFS = 133 No. of nodes explored in DFS with MRV and forward checking = 7

Based on the above figure obtained by running the DFS Solver and DFS Solver with MRC & FC over a set of sample tasks mentioned in the tab : "#Sample tasks for different tasks". The no. of nodes explored with the plain DFS is much more than the nodes explored using DFS with MRV & FC. Hence, DFS with MRC and FC explored less nodes eliminates lot of options before hand. 33 7 d

Question 5 (4 marks)

The DFS solver chooses variables in random order, and systematically explores all values for those variables in no particular order.

Incorporate costs into the DFS constraint solver as heuristics to guide the search. Similar to the cost function for the domain splitting solver, for a given variable *v*, the cost of assigning the value *val* to *v* is the cost of violating the soft deadline constraint (if any) associated with *v* for the value *val*. The *minimum cost* for *v* is the lowest cost from amongst the values in the domain of *v*. The DFS solver should choose a variable *v* with lowest minimum cost, and explore its values in order of cost from lowest to highest.

- Implement this behaviour by modifying the code in `dfs_solver` and place a copy of the code below (2 marks)
- Empirically compare the performance of DFS with and without these heuristics (2 marks)

For the empirical evaluations, again run the two algorithms on a variety of problems of size `n` for varying `n`.

In [ ]:
```
# Code for Question 5
# Place a copy of your code here and run it in the relevant cell
```

In [158…

```
num_expanded = 0
display = True

def compute_value_cost(var, val, durations, soft_day_time, soft_costs):
    """
    Compute the cost for assigning value `val` to variable `var`
    based on soft deadlines (if any).
    """
    if var not in soft_day_time:
        return 0  # no soft constraint on this variable

    # Get deadline and penalty
    deadline = Day_Time().string_to_week_hour_number(soft_day_time[var])
    penalty_weight = int(soft_costs[var])
    duration = durations[var]

    # Compute lateness (if finish time exceeds deadline)
    finish = val + duration
    delay = max(0, finish - deadline[1])
    cost = delay * penalty_weight
    return cost


def dfs_solver(constraints, domains, context, var_order,
               durations=None, soft_day_time=None, soft_costs=None):
    """
    DFS generator for all CSP solutions, guided by soft-constraint cost heuristi
    Chooses variable with lowest min cost and explores domain values in cost ord
    """
    global num_expanded, display

    to_eval = {c for c in constraints if c.can_evaluate(context)}
    if all(c.holds(context) for c in to_eval):
        if var_order == []:
            print("Nodes expanded to reach solution:", num_expanded)
            yield context
        else:
            rem_cons = [c for c in constraints if c not in to_eval]

            #Compute heuristic: min cost for each remaining variable
            var_min_cost = {}
            for v in var_order:
                domain_costs = [compute_value_cost(v, val, durations, soft_day_t
                var_min_cost[v] = min(domain_costs) if domain_costs else float('

            #Select variable with lowest minimum cost
            var = min(var_min_cost, key=var_min_cost.get)

            #Sort domain values by ascending cost
            domain_sorted = sorted(
                domains[var],
                key=lambda val: compute_value_cost(var, val, durations, soft_day
            )

            remaining_vars = [v for v in var_order if v != var]

            #Explore values in increasing cost order
            for val in domain_sorted:
                if display:
                    print(f"Setting {var} = {val} (cost {compute_value_cost(var,
                num_expanded += 1
                yield from dfs_solver(rem_cons, domains, context | {var: val}, r


def dfs_solve_all(csp, var_order=None):
    """Depth-first CSP solver to return all solutions with cost-guided heuristic
    global num_expanded
    num_expanded = 0

    if var_order is None:
        var_order = list(csp.domains)

    return list(dfs_solver(csp.constraints, csp.domains, {}, var_order, duration


def dfs_solve1(csp, var_order=None):
    """Depth-first CSP solver returning the first solution (cost-guided)."""
    global num_expanded
    num_expanded = 0

    if var_order is None:
        var_order = list(csp.domains)

    for sol in dfs_solver(csp.constraints, csp.domains, {}, var_order, durations
        return sol  # Return first found solution
```

Answers for Question 5

Write the other answers here. Ther performance of the DFS constraint Solver with or without the heuristic of cost would be the same as it doesn't consider cost while performing the search for a solution.

Question 6 (3 marks)

The CSP solver with domain splitting splits a CSP variable domain into *exactly two* partitions. Poole & Mackworth claim that in practice, this is as good as splitting into a larger number of partitions. In this question, empirically evaluate this claim for fuzzy scheduling CSPs.

- Write a new `partition_domain` function that partitions a domain into a list of `k` partitions, where `k` is a parameter to the function (1 mark)
- Modify the CSP solver to use the list of `k` partitions and evaluate the performance of the solver using the above metric for a range of values of `k` (2 marks)

In [141…

```
# Code for Question 6
# Place a copy of your code here and run it in the relevant cell
def partition_domain(domain, k):
    """
    Partition a domain (list or set) into k roughly equal-sized subdomains.
    Returns a list of k lists (partitions).

    Parameters:
        domain (list or set): the domain values to partition
        k (int): number of partitions (k >= 1)

    """
    # Convert to list in case domain is a set
    domain = list(domain)
    n = len(domain)

    # Guard against invalid k
    if k <= 0:
        raise ValueError("k must be a positive integer.")
    if k > n:
        k = n  # can't make more partitions than elements

    # Determine partition size
    base_size = n // k
    remainder = n % k  # distribute leftover items

    partitions = []
    start = 0
    for i in range(k):
        end = start + base_size + (1 if i < remainder else 0)
        partitions.append(domain[start:end])
        start = end

    return partitions
```

In [142…

```
domain = list(range(1, 11))  # [1, 2, 3, ..., 10]

print(partition_domain(domain, 2))
# [[1,2,3,4,5], [6,7,8,9,10]]

print(partition_domain(domain, 3))
# [[1,2,3,4], [5,6,7], [8,9,10]]
```

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]]
[[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]
```

Answers for Question 6

Write the other answers here.