

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 AI Python 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 AI Python

You will need AI Python 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 AI Python and run your code (you will not submit any AI Python code).

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

```
In [1]: import sys
sys.path.append('aipython') # change to your directory
sys.path # check that aipython is now on the path
```

```
Out[1]: ['/Users/wangjiawei/PycharmProjects/PythonProject15',
'/Users/wangjiawei/PycharmProjects/PythonProject15/aipython',
'/Users/wangjiawei/Applications/PyCharm Professional Edition.app/Contents/plugins/python-ce/helpers/pydev',
'/Users/wangjiawei/Applications/PyCharm Professional Edition.app/Contents/plugins/python/helpers-pro/jupyter_debug',
'/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12.zip',
'/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12',
'/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/lib-dynload',
'',
'/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages',
'/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/aeosa',
'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 [2]: # -*- 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' into
        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].strip())
            type2, v2 = self.string_to_week_hour_number(str1.split('-')[1].strip())
            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

    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]

```

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 [3]: 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 variables
    * 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 assignment

```

```

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

        dt_converter = Day_Time()
        _, soft_day_time_val = dt_converter.string_to_week_hour_number(self.s

        soft_costs_val = self.soft_costs[var]
        if soft_costs_val == 0:
            continue
        deadline_h, deadline_d = dt_converter.hour_day_split(soft_day_time_v

        # No change to this part
        if len(domain) == 1:
            start_time = list(domain)[0]
            end_time = start_time + duration
            if end_time > soft_day_time_val:
                end_h, end_d = dt_converter.hour_day_split(end_time)
                delay_hours = (end_h - deadline_h) + 24 * (end_d - deadline_
                cost += delay_hours * soft_costs_val

        # The fix is in this 'else' block
        else:
            min_cost_for_var = float('inf')
            for start in domain:
                end_time = start + duration

                current_local_cost = 0
                if end_time > soft_day_time_val:
                    end_h, end_d = dt_converter.hour_day_split(end_time)
                    delay_hours = (end_h - deadline_h) + 24 * (end_d - deadl
                    current_local_cost = delay_hours * soft_costs_val

            if current_local_cost < min_cost_for_var:

```

```

        min_cost_for_var = current_local_cost

        # **THE FIX IS HERE:** If we find a way to get 0 cost, we ca
        # Stop searching for this variable and move to the next.
        if min_cost_for_var == 0:
            break

        if min_cost_for_var != float('inf'):
            cost += min_cost_for_var

    print(f"DEBUG: Heuristic cost for this node is: {cost}")
    return 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 [4]: 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 [5]: # 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(5pm) = 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]]

# 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 5pm'
            # 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 [6]: 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* definition
        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 [7]: # 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
"""
```

```
In [8]: # display details (0 turns off)
Con_solver.max_display_level = 0
Search_with_AC_from_Cost_CSP.max_display_level = 2
GreedySearcher.max_display_level = 0

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:]+\nncost: '+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 3
task, t2 4
constraint, t1 before t2
constraint, t1 same-day t2
domain, t2 mon
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 10
t1: mon 9am
t2: mon 12pm
cost: 10

```

6. Depth-First Search Constraint Solver

The Depth-First Constraint Solver in AI Python 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 [9]: num_expanded = 0
        display = False

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_order)

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 [10]: 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 3
task, t2 4
constraint, t1 before t2
constraint, t1 same-day t2
domain, t2 mon
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 5
t1: mon 9am
t2: mon 12pm

```

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

The Depth-First Constraint Solver in AI Python 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 AI Python code is designed to manipulate domain sets, we also need to redefine `can_evaluate` to handle partial assignments.

```

In [11]: num_expanded = 0
    display = False

    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 [12]: 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 3
task, t2 4
constraint, t1 before t2
constraint, t1 same-day t2
domain, t2 mon
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 5
t2: mon 12pm
t1: mon 9am

```

Assignment

Name: Jiawei Wang

zID: z5649403

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

Write the answers here.

The search space of the domain splitting with arc consistency is starting from the result of the first round use of the arc consistency, and its successor function is exploring the domain of the variable with divided into two part and using the arc consistency on the divided domain, the goal is when every variable only have single value. The search space of the DFS solver is an empty assignment for variables, and the successor function is finding the value in domain to the unassigned variable one by one if the new value violate the path will be backtracking, the goal for dfs is finding the result that all variable are assigned a value and there is no conflict.

The branch factor of the domain splitting with arc consistency is always 2, as it is domain divided into 2 part everytime. And the max depth is the searching the total domain D of all variable, and do the divide. it $\log_2(D)$. The branch factor of dfs is choosing from the domain of one variable, it has d choices each time, the branch factor is d . and the maximum depth is finding all value for n variable, it is n .

The worst case time of domain splitting with arc consistency is $\log(n)$ as it always searching in half. The worst case of space is use depth and cost is $O(n\log(n))$. The worst case time of dfs is testing with all the domain possible d of all the value n . it is $O(d^n)$, and the space complexity is try all the queue of n variable, $O(d*n)$.

The example is the situation when there is conflict for different value, for example a start before b , b start before c , c start before $mon\ 10am$, so it is impossible, for the domain splitting with arc consistency it can find there is no solution easily, but dfs need search the whole possible before get the result.

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 [13]: # Code for Question 2
# Place a copy of your code here and run it in the relevant cell
def calculate_cost(self):
    cost = 0
    # # TODO: write cost function
    for var in self.variables:
        domain = self.domains[var]
        duration = self.durations[var]

        dt_converter = Day_Time()
        _, soft_day_time_val = dt_converter.string_to_week_hour_number(self.s
        soft_costs_val = self.soft_costs[var]
        if soft_costs_val == 0:
            continue
        deadline_h, deadline_d = dt_converter.hour_day_split(soft_day_time_v

        if len(domain) == 1:
            start_time = list(domain)[0]
            end_time = start_time + duration
            if end_time > soft_day_time_val:
                end_h, end_d = dt_converter.hour_day_split(end_time)
                delay_hours = (end_h - deadline_h) + 24 * (end_d - deadline
                cost += delay_hours * soft_costs_val
            else:
                min_cost_for_var = float('inf')
                for start in domain:
                    end_time = start + duration
                    current_local_cost = 0
                    if end_time > soft_day_time_val:
                        end_h, end_d = dt_converter.hour_day_split(end_time)
                        delay_hours = (end_h - deadline_h) + 24 * (end_d - deadl
                        current_local_cost = delay_hours * soft_costs_val
                    if current_local_cost < min_cost_for_var:
                        min_cost_for_var = current_local_cost
                    if min_cost_for_var == 0:
                        break

                if min_cost_for_var != float('inf'):
                    cost += min_cost_for_var
    return cost
```

Answers for Question 2

Write the other answers here. The computational complexity for this is $O(n*d)$, as there are two loop, one is loop the whole variable, and the other is the inner loop of the value of the domain.

The cost function never decrease because every time, we choose the value with the lowest cost, and when it is divided, we choose either contain the lowest value or the bigger one, so it will never decrease along the path. it makes it optimal for only extend the value with the lowest cost.

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 [14]: import random

def generate_problem_solvable(n):
    DAYS = ['mon', 'tue', 'wed', 'thu', 'fri']
    TIMES = ['9am', '10am', '11am', '12pm', '1pm', '2pm', '3pm', '4pm']
    DEADLINE_TIMES = ['11am', '1pm', '3pm']
    spec_lines = []
    task_names = [f't{i}' for i in range(1, n + 1)]

    spec_lines.append(f'# {n} tasks with constraints and soft deadlines')
    task_durations = {name: random.randint(1, 4) for name in task_names}
    for name, duration in task_durations.items():
        spec_lines.append(f'task, {name} {duration}')
    spec_lines.append('')

    spec_lines.append('# Binary constraints')
    num_binary_constraints = max(1, n // 2)
    for _ in range(num_binary_constraints):
        t1, t2 = random.sample(task_names, 2)
        relation = random.choice(['before', 'same-day'])
        spec_lines.append(f'constraint, {t1} {relation} {t2}')
    spec_lines.append('')

    spec_lines.append('# Hard domain constraints')
    daily_hours = {day: 0 for day in DAYS}
    tasks_to_constrain = random.sample(task_names, k=random.randint(1, n // 2))
    for task in tasks_to_constrain:
        duration = task_durations[task]
        available_days = [day for day in DAYS if daily_hours[day] + duration <=
if available_days:
```

```

        day = random.choice(available_days)
        spec_lines.append(f'domain, {task} {day}')
        daily_hours[day] += duration
    spec_lines.append('')

    spec_lines.append('# Soft deadlines')
    for task in task_names:
        day = random.choice(DAYS[:4])
        time = random.choice(DEADLINE_TIMES)
        cost = random.randint(10, 50)
        spec_lines.append(f'domain, {task} ends-by {day} {time} {cost}')
    spec_lines.append('')

    return "\n".join(spec_lines)
# problem_for_4_tasks = generate_problem_solvable(4)
# print(problem_for_4_tasks)
#
# problem_for_10_tasks = generate_problem_solvable(10)

```

```

In [15]: import matplotlib.pyplot as plt
import time
import numpy as np

def run_single_experiment(spec, use_cost_heuristic=True):
    original_cost_func = CSP_with_Cost.calculate_cost
    if not use_cost_heuristic:
        def zero_cost_func(self):
            return 0
        CSP_with_Cost.calculate_cost = zero_cost_func

    try:
        import sys
        from io import StringIO
        old_stdout = sys.stdout
        sys.stdout = StringIO()
        csp_problem = create_CSP_from_spec(spec)
        solver = GreedySearcher(Search_with_AC_from_Cost_CSP(csp_problem))
        solver.search()
        sys.stdout = old_stdout
        return solver.num_expanded

    finally:
        CSP_with_Cost.calculate_cost = original_cost_func

n_values = range(2, 100, 5)
num_runs_per_n = 5
avg_nodes_with_cost=[]
avg_nodes_without_cost=[]

for n in n_values:
    start_time = time.time()
    nodes_with_cost = []
    nodes_without_cost = []

    for i in range(num_runs_per_n):
        problem_spec = generate_problem_solvable(n)
        expanded_with_cost = run_single_experiment(problem_spec, use_cost_heuris
        nodes_with_cost.append(expanded_with_cost)
        expanded_without_cost = run_single_experiment(problem_spec, use_cost_heu
        nodes_without_cost.append(expanded_without_cost)

```

```

avg_nodes_with_cost.append(np.mean(nodes_with_cost))
avg_nodes_without_cost.append(np.mean(nodes_without_cost))

end_time = time.time()
print(f"Finished n={n} in {end_time - start_time:.2f} seconds. "
      f"Avg Nodes (With/Without Cost): {avg_nodes_with_cost[-1]:.1f} / {avg_

plt.figure(figsize=(12, 7))
plt.plot(n_values, avg_nodes_with_cost, marker='o', linestyle='-', label='Solver
plt.plot(n_values, avg_nodes_without_cost, marker='x', linestyle='--', label='So

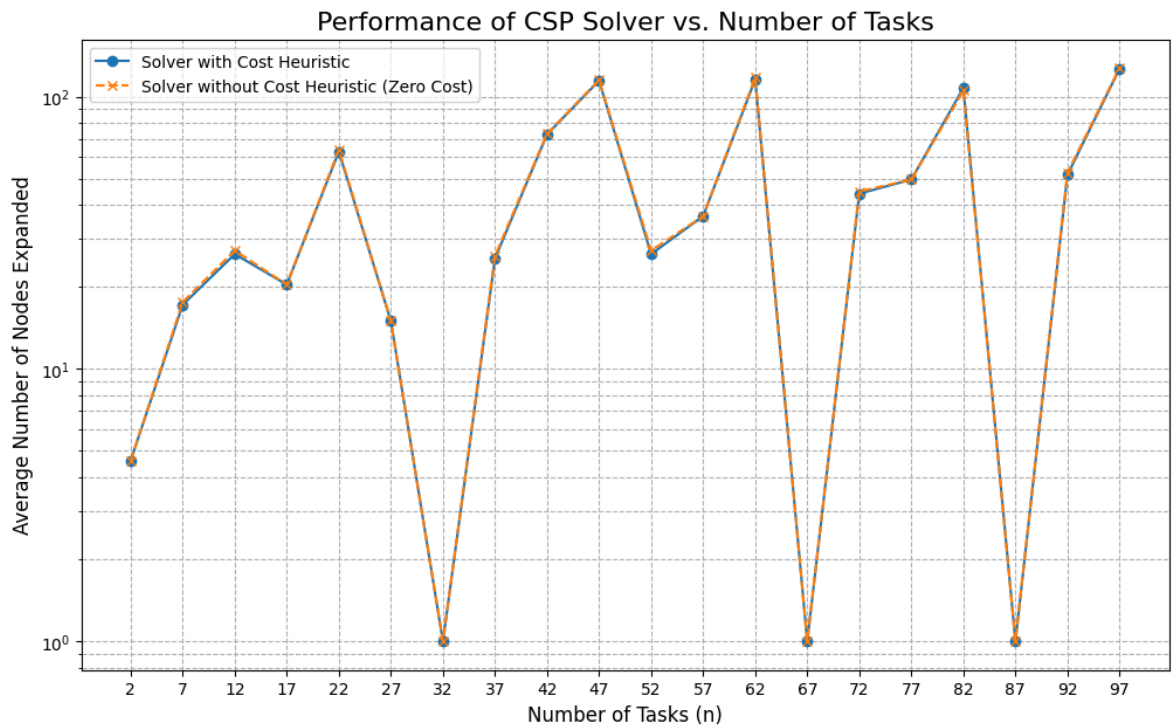
plt.title('Performance of CSP Solver vs. Number of Tasks', fontsize=16)
plt.xlabel('Number of Tasks (n)', fontsize=12)
plt.ylabel('Average Number of Nodes Expanded', fontsize=12)
plt.xticks(n_values)
plt.yscale('log')
plt.legend(fontsize=10)
plt.grid(True, which="both", ls="--")
plt.show()

```

```

Finished n=2 in 0.01 seconds. Avg Nodes (With/Without Cost): 4.6 / 4.6
Finished n=7 in 0.03 seconds. Avg Nodes (With/Without Cost): 17.2 / 17.6
Finished n=12 in 0.12 seconds. Avg Nodes (With/Without Cost): 26.4 / 27.2
Finished n=17 in 0.12 seconds. Avg Nodes (With/Without Cost): 20.4 / 20.4
Finished n=22 in 0.24 seconds. Avg Nodes (With/Without Cost): 62.8 / 63.8
Finished n=27 in 0.13 seconds. Avg Nodes (With/Without Cost): 15.0 / 15.0
Finished n=32 in 0.09 seconds. Avg Nodes (With/Without Cost): 1.0 / 1.0
Finished n=37 in 0.24 seconds. Avg Nodes (With/Without Cost): 25.4 / 26.0
Finished n=42 in 0.52 seconds. Avg Nodes (With/Without Cost): 72.6 / 73.0
Finished n=47 in 0.90 seconds. Avg Nodes (With/Without Cost): 114.4 / 115.0
Finished n=52 in 0.30 seconds. Avg Nodes (With/Without Cost): 26.4 / 27.2
Finished n=57 in 0.53 seconds. Avg Nodes (With/Without Cost): 36.4 / 36.4
Finished n=62 in 1.32 seconds. Avg Nodes (With/Without Cost): 115.6 / 117.4
Finished n=67 in 0.20 seconds. Avg Nodes (With/Without Cost): 1.0 / 1.0
Finished n=72 in 0.78 seconds. Avg Nodes (With/Without Cost): 44.0 / 44.8
Finished n=77 in 0.94 seconds. Avg Nodes (With/Without Cost): 49.6 / 49.6
Finished n=82 in 2.01 seconds. Avg Nodes (With/Without Cost): 107.6 / 105.2
Finished n=87 in 0.26 seconds. Avg Nodes (With/Without Cost): 1.0 / 1.0
Finished n=92 in 1.14 seconds. Avg Nodes (With/Without Cost): 51.6 / 52.6
Finished n=97 in 2.87 seconds. Avg Nodes (With/Without Cost): 126.4 / 127.2

```



```
In [16]: import pandas as pd
df = pd.DataFrame({
    'n_value': list(n_values),
    'With Cost': avg_nodes_with_cost,
    'Without Cost': avg_nodes_without_cost
})
df['Percentage Difference (%)'] = ((df['Without Cost'] - df['With Cost']) / df['

# 从 DataFrame 的列中计算总平均值
avg_with_cost_total = df['With Cost'].mean()
avg_without_cost_total = df['Without Cost'].mean()

# 计算总的百分比差异
percentage_diff_avg = ((avg_without_cost_total - avg_with_cost_total) / avg_with

print(df.to_string(index=False)) # 打印整个 DataFrame
print(f"\nOverall Average Difference: {percentage_diff_avg:.2f}%")
```

n_value	With Cost	Without Cost	Percentage Difference (%)
2	4.6	4.6	0.000000
7	17.2	17.6	2.325581
12	26.4	27.2	3.030303
17	20.4	20.4	0.000000
22	62.8	63.8	1.592357
27	15.0	15.0	0.000000
32	1.0	1.0	0.000000
37	25.4	26.0	2.362205
42	72.6	73.0	0.550964
47	114.4	115.0	0.524476
52	26.4	27.2	3.030303
57	36.4	36.4	0.000000
62	115.6	117.4	1.557093
67	1.0	1.0	0.000000
72	44.0	44.8	1.818182
77	49.6	49.6	0.000000
82	107.6	105.2	-2.230483
87	1.0	1.0	0.000000
92	51.6	52.6	1.937984
97	126.4	127.2	0.632911

Overall Average Difference: 0.72%

Answers for Question 3

Write the other answers here. There is a slight performance difference between the result, the reason is that the CSP without cost only return the result undirectly to get the result, but the CSP with cost would have the cost function to guide the search, so it would be more efficient.

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 [17]: # Code for Question 4
# Place your code here
probelm_string = generate_problem_solvable(6)
csp_problem_cost = create_CSP_from_spec(probelm_string)
test_dfs_solver(csp_problem_cost)
num_expanded
```

```
task, t1 4
task, t2 4
task, t3 4
task, t4 3
task, t5 4
task, t6 2
constraint, t6 same-day t3
constraint, t5 before t4
constraint, t5 before t3
domain, t2 mon
domain, t1 ends-by tue 3pm 29
domain, t2 ends-by tue 1pm 30
domain, t3 ends-by thu 11am 16
domain, t4 ends-by thu 11am 48
domain, t5 ends-by wed 3pm 41
domain, t6 ends-by wed 3pm 49
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 2200
t1: mon 9am
t2: mon 9am
t3: tue 9am
t4: mon 1pm
t5: mon 9am
t6: tue 9am
```

Out[17]: 2200

```
In [18]: csp_problem_cost = create_CSP_from_spec(probelm_string)
test_mrv_dfs_solver(csp_problem_cost)
num_expanded
```

```
task, t1 4
task, t2 4
task, t3 4
task, t4 3
task, t5 4
task, t6 2
constraint, t6 same-day t3
constraint, t5 before t4
constraint, t5 before t3
domain, t2 mon
domain, t1 ends-by tue 3pm 29
domain, t2 ends-by tue 1pm 30
domain, t3 ends-by thu 11am 16
domain, t4 ends-by thu 11am 48
domain, t5 ends-by wed 3pm 41
domain, t6 ends-by wed 3pm 49
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 10
t2: mon 9am
t1: mon 9am
t3: tue 9am
t5: mon 9am
t6: tue 9am
t4: mon 1pm
```

Out[18]: 10

```
In [19]: csp_problem = create_CSP_from_spec(probelm_string)
solver = GreedySearcher(Search_with_AC_from_Cost_CSP(csp_problem))
test_csp_solver(solver)
solver.num_expanded
```

```

task, t1 4
task, t2 4
task, t3 4
task, t4 3
task, t5 4
task, t6 2
constraint, t6 same-day t3
constraint, t5 before t4
constraint, t5 before t3
domain, t2 mon
domain, t1 ends-by tue 3pm 29
domain, t2 ends-by tue 1pm 30
domain, t3 ends-by thu 11am 16
domain, t4 ends-by thu 11am 48
domain, t5 ends-by wed 3pm 41
domain, t6 ends-by wed 3pm 49
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t1 into {0, 1, 2, 3, 8, 9, 10, 11, 16, 17} and {32, 33, 34, 35, 18, 19,
24, 25, 26, 27}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 696
Splitting t1 into {0, 1, 2, 3, 8} and {9, 10, 11, 16, 17}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t1 into {0, 1} and {8, 2, 3}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t1 into {0} and {1}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t2 into {0, 1} and {2, 3}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t2 into {0} and {1}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t3 into {32, 33, 34, 35, 8, 9, 10, 11} and {16, 17, 18, 19, 24, 25, 26,
27}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t3 into {32, 33, 34, 35} and {8, 9, 10, 11}
DEBUG: Heuristic cost for this node is: 2572
DEBUG: Heuristic cost for this node is: 0
Splitting t3 into {8, 9} and {10, 11}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t3 into {8} and {9}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t4 into {4, 8, 9, 10, 11, 12, 16, 17, 18, 19} and {32, 33, 34, 35, 36,
20, 24, 25, 26, 27, 28}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t4 into {4, 8, 9, 10, 11} and {12, 16, 17, 18, 19}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t4 into {8, 4} and {9, 10, 11}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0

```

```

Splitting t4 into {8} and {4}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t5 into {0, 1} and {2, 3}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t5 into {0} and {1}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t6 into {8, 9, 10} and {11, 12, 13}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
Splitting t6 into {8} and {9, 10}
DEBUG: Heuristic cost for this node is: 0
DEBUG: Heuristic cost for this node is: 0
t1: mon 9am
t2: mon 9am
t3: tue 9am
t4: tue 9am
t5: mon 9am
t6: tue 9am
cost: 0

```

Out[19]: 19

In [20]: `from matplotlib import pyplot as plt`

```

n_values = range(2,10)
results_dfs_first = []
results_dfsmrvt_first = []
for n_value in n_values:
    problem_string = generate_problem_solvable(n_value)
    csp_problem_cost = create_CSP_from_spec(problem_string)
    dfs_solve1(csp_problem_cost)
    results_dfs_first.append(num_expanded)
    csp_problem_cost = create_CSP_from_spec(problem_string)
    mrv_dfs_solve1(csp_problem_cost)
    results_dfsmrvt_first.append(num_expanded)

plt.plot(n_values, results_dfs_first, marker='o', linestyle='-', label='Standard')
plt.plot(n_values, results_dfsmrvt_first, marker='x', linestyle='--', label='DFS')
plt.title('Comparison of Expanded Nodes vs. Problem Size ($N$)')
plt.xlabel('Problem Size ($N$)')
plt.ylabel('Number of Expanded Nodes')
plt.legend()
plt.yscale('log')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()

```

```
task, t1 1
task, t2 4
constraint, t1 before t2
domain, t1 tue
domain, t1 ends-by tue 3pm 19
domain, t2 ends-by tue 3pm 41
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 7
task, t1 1
task, t2 4
constraint, t1 before t2
domain, t1 tue
domain, t1 ends-by tue 3pm 19
domain, t2 ends-by tue 3pm 41
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 2
task, t1 1
task, t2 4
task, t3 2
constraint, t1 same-day t3
domain, t3 wed
domain, t1 ends-by tue 1pm 31
domain, t2 ends-by wed 3pm 44
domain, t3 ends-by thu 11am 30
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 1977
task, t1 1
task, t2 4
task, t3 2
constraint, t1 same-day t3
domain, t3 wed
domain, t1 ends-by tue 1pm 31
domain, t2 ends-by wed 3pm 44
domain, t3 ends-by thu 11am 30
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 3
task, t1 2
task, t2 4
task, t3 2
task, t4 3
constraint, t4 before t3
constraint, t2 before t4
domain, t2 wed
domain, t4 mon
domain, t1 ends-by thu 3pm 36
domain, t2 ends-by wed 1pm 19
domain, t3 ends-by thu 1pm 17
domain, t4 ends-by thu 1pm 11
DEBUG: Heuristic cost for this node is: 0
task, t1 2
task, t2 4
task, t3 2
task, t4 3
constraint, t4 before t3
constraint, t2 before t4
domain, t2 wed
domain, t4 mon
domain, t1 ends-by thu 3pm 36
domain, t2 ends-by wed 1pm 19
domain, t3 ends-by thu 1pm 17
```

```

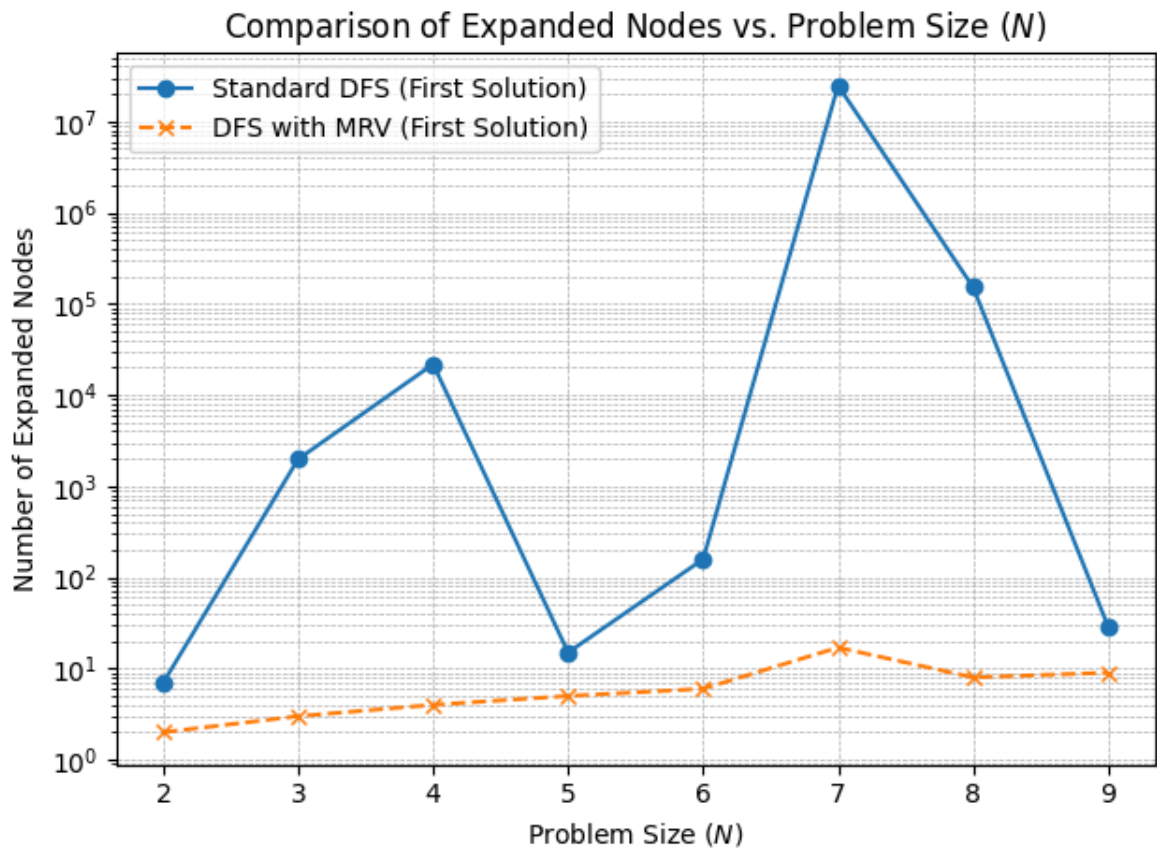
domain, t4 ends-by thu 1pm 11
DEBUG: Heuristic cost for this node is: 0
task, t1 2
task, t2 2
task, t3 3
task, t4 4
task, t5 4
constraint, t1 before t4
constraint, t2 same-day t3
domain, t1 wed
domain, t1 ends-by mon 11am 23
domain, t2 ends-by wed 3pm 25
domain, t3 ends-by mon 3pm 19
domain, t4 ends-by wed 1pm 39
domain, t5 ends-by tue 3pm 28
DEBUG: Heuristic cost for this node is: 1104
Nodes expanded to reach solution: 15
task, t1 2
task, t2 2
task, t3 3
task, t4 4
task, t5 4
constraint, t1 before t4
constraint, t2 same-day t3
domain, t1 wed
domain, t1 ends-by mon 11am 23
domain, t2 ends-by wed 3pm 25
domain, t3 ends-by mon 3pm 19
domain, t4 ends-by wed 1pm 39
domain, t5 ends-by tue 3pm 28
DEBUG: Heuristic cost for this node is: 1104
Nodes expanded to reach solution: 5
task, t1 4
task, t2 1
task, t3 4
task, t4 1
task, t5 4
task, t6 1
constraint, t3 before t5
constraint, t6 before t2
constraint, t2 before t1
domain, t2 fri
domain, t3 thu
domain, t4 thu
domain, t1 ends-by tue 11am 36
domain, t2 ends-by wed 11am 30
domain, t3 ends-by wed 1pm 34
domain, t4 ends-by wed 3pm 10
domain, t5 ends-by thu 11am 12
domain, t6 ends-by mon 11am 43
DEBUG: Heuristic cost for this node is: 2416
Nodes expanded to reach solution: 158
task, t1 4
task, t2 1
task, t3 4
task, t4 1
task, t5 4
task, t6 1
constraint, t3 before t5
constraint, t6 before t2

```

constraint, t2 before t1
domain, t2 fri
domain, t3 thu
domain, t4 thu
domain, t1 ends-by tue 11am 36
domain, t2 ends-by wed 11am 30
domain, t3 ends-by wed 1pm 34
domain, t4 ends-by wed 3pm 10
domain, t5 ends-by thu 11am 12
domain, t6 ends-by mon 11am 43
DEBUG: Heuristic cost for this node is: 2416
Nodes expanded to reach solution: 6
task, t1 3
task, t2 2
task, t3 2
task, t4 2
task, t5 1
task, t6 3
task, t7 4
constraint, t6 before t1
constraint, t4 before t6
constraint, t6 same-day t2
domain, t7 fri
domain, t5 fri
domain, t1 ends-by wed 11am 49
domain, t2 ends-by wed 1pm 33
domain, t3 ends-by wed 3pm 30
domain, t4 ends-by tue 1pm 22
domain, t5 ends-by tue 3pm 12
domain, t6 ends-by tue 11am 18
domain, t7 ends-by wed 1pm 10
DEBUG: Heuristic cost for this node is: 1284
Nodes expanded to reach solution: 24709664
task, t1 3
task, t2 2
task, t3 2
task, t4 2
task, t5 1
task, t6 3
task, t7 4
constraint, t6 before t1
constraint, t4 before t6
constraint, t6 same-day t2
domain, t7 fri
domain, t5 fri
domain, t1 ends-by wed 11am 49
domain, t2 ends-by wed 1pm 33
domain, t3 ends-by wed 3pm 30
domain, t4 ends-by tue 1pm 22
domain, t5 ends-by tue 3pm 12
domain, t6 ends-by tue 11am 18
domain, t7 ends-by wed 1pm 10
DEBUG: Heuristic cost for this node is: 1284
Nodes expanded to reach solution: 17
task, t1 1
task, t2 1
task, t3 3
task, t4 4
task, t5 1
task, t6 4

task, t7 4
task, t8 4
constraint, t1 same-day t7
constraint, t6 before t8
constraint, t6 same-day t7
constraint, t4 before t3
domain, t4 fri
domain, t2 tue
domain, t6 tue
domain, t1 ends-by wed 1pm 34
domain, t2 ends-by tue 11am 39
domain, t3 ends-by wed 3pm 11
domain, t4 ends-by thu 3pm 18
domain, t5 ends-by mon 11am 46
domain, t6 ends-by thu 3pm 20
domain, t7 ends-by tue 11am 17
domain, t8 ends-by thu 11am 50
DEBUG: Heuristic cost for this node is: 396
Nodes expanded to reach solution: 152096
task, t1 1
task, t2 1
task, t3 3
task, t4 4
task, t5 1
task, t6 4
task, t7 4
task, t8 4
constraint, t1 same-day t7
constraint, t6 before t8
constraint, t6 same-day t7
constraint, t4 before t3
domain, t4 fri
domain, t2 tue
domain, t6 tue
domain, t1 ends-by wed 1pm 34
domain, t2 ends-by tue 11am 39
domain, t3 ends-by wed 3pm 11
domain, t4 ends-by thu 3pm 18
domain, t5 ends-by mon 11am 46
domain, t6 ends-by thu 3pm 20
domain, t7 ends-by tue 11am 17
domain, t8 ends-by thu 11am 50
DEBUG: Heuristic cost for this node is: 396
Nodes expanded to reach solution: 8
task, t1 3
task, t2 2
task, t3 4
task, t4 2
task, t5 4
task, t6 1
task, t7 2
task, t8 1
task, t9 4
constraint, t8 before t6
constraint, t4 before t6
constraint, t4 same-day t3
constraint, t4 same-day t3
domain, t6 thu
domain, t3 thu
domain, t5 mon

```
domain, t9 fri
domain, t1 ends-by tue 11am 28
domain, t2 ends-by wed 11am 50
domain, t3 ends-by mon 1pm 10
domain, t4 ends-by tue 11am 41
domain, t5 ends-by thu 3pm 48
domain, t6 ends-by thu 1pm 17
domain, t7 ends-by wed 11am 34
domain, t8 ends-by tue 3pm 32
domain, t9 ends-by thu 3pm 40
DEBUG: Heuristic cost for this node is: 1600
Nodes expanded to reach solution: 29
task, t1 3
task, t2 2
task, t3 4
task, t4 2
task, t5 4
task, t6 1
task, t7 2
task, t8 1
task, t9 4
constraint, t8 before t6
constraint, t4 before t6
constraint, t4 same-day t3
constraint, t4 same-day t3
domain, t6 thu
domain, t3 thu
domain, t5 mon
domain, t9 fri
domain, t1 ends-by tue 11am 28
domain, t2 ends-by wed 11am 50
domain, t3 ends-by mon 1pm 10
domain, t4 ends-by tue 11am 41
domain, t5 ends-by thu 3pm 48
domain, t6 ends-by thu 1pm 17
domain, t7 ends-by wed 11am 34
domain, t8 ends-by tue 3pm 32
domain, t9 ends-by thu 3pm 40
DEBUG: Heuristic cost for this node is: 1600
Nodes expanded to reach solution: 9
```



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). DFS solver worst time is $O(d^n)$, worst space is $O(dn)$. DFS-MRV the worst time is the same as DFS $O(d^n)$, and for the space $O(nd^2)$ for it need to store the domain for each variable.

Both the algorithm is complete, as if there is a solution the two algorithm will find it. The DFS is not optimal, as it will return the first solution it found, and the DFS-MRV is not optimal as it will the variable with the least domain first, but it will not consider the cost.

The example is the situation when there is a value that make the searching faster. for example a start agter b, b start after c, c ends before mon 4pm for the DFS-MRV it can find c as it has the most restrictions, but DFS need search with a then b,c and if it find the c and violate the constraint it would researching, which makes DFS-MRV perform better.

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 [21]: def dfs_solver(constraints, domains, context, var_order, csp):
    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 not var_order:
            print("Nodes expanded to reach solution:", num_expanded)
            yield context
        else:
            var_min_costs = []
            for var in var_order:
                costs_for_this_var = []
                for val in domains[var]:
                    cost = 0
                    duration = csp.durations[var]
                    soft_day_time_str = csp.soft_day_time.get(var)
                    soft_costs_val = csp.soft_costs.get(var, 0)
                    if soft_day_time_str and soft_costs_val > 0:
                        dt_converter = Day_Time()
                        _, soft_day_time_val = dt_converter.string_to_week_hour_
                        end_time = val + duration
                        if end_time > soft_day_time_val:
                            end_h, end_d = dt_converter.hour_day_split(end_time)
                            deadline_h, deadline_d = dt_converter.hour_day_split(
                                soft_day_time_val)
                            delay_hours = (end_h - deadline_h) + 24 * (end_d -
                                deadline_d)
                            cost = delay_hours * soft_costs_val
                    costs_for_this_var.append(cost)

                if costs_for_this_var:
                    min_cost = min(costs_for_this_var)
                    var_min_costs.append((min_cost, var))

            if not var_min_costs:
                return

    var_min_costs.sort(key=lambda item: item[0])
    chosen_var = var_min_costs[0][1]
    value_costs = []
    for val in domains[chosen_var]:
        cost = 0
        dt_converter = Day_Time()
        duration = csp.durations[chosen_var]
        soft_day_time_str = csp.soft_day_time.get(chosen_var)
        soft_costs_val = csp.soft_costs.get(chosen_var, 0)
        if soft_day_time_str and soft_costs_val > 0:
            _, soft_day_time_val = dt_converter.string_to_week_hour_num
            end_time = val + duration
            if end_time > soft_day_time_val:
                deadline_h, deadline_d = dt_converter.hour_day_split(soft
                    day_time_val)
                end_h, end_d = dt_converter.hour_day_split(end_time)
```

```

        delay_hours = (end_h - deadline_h) + 24 * (end_d - deadl
        cost = delay_hours * soft_costs_val
        value_costs.append((cost, val))

    value_costs.sort(key=lambda item: item[0])

    rem_cons = [c for c in constraints if c not in to_eval]
    new_var_order = [v for v in var_order if v != chosen_var]

    for cost, val in value_costs:
        if display:
            print(f"Choosing var '{chosen_var}', trying value {val} (Cos
            num_expanded += 1
            yield from dfs_solver(rem_cons, domains, context | {chosen_var:

```

```

In [22]: def dfs_solve2(csp, var_order=None):
    global num_expanded
    num_expanded = 0
    if var_order == None:
        var_order = list(csp.domains)
    for sol in dfs_solver(csp.constraints, csp.domains, {}, var_order, csp):
        return sol
def test_dfs_solver2(csp_problem):
    solution = dfs_solve2(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:])

```

```

In [23]: def dfs_solver_before(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_before(rem_cons, domains, context | {var:val}
def dfs_solve3(csp, var_order=None):
    global num_expanded
    num_expanded = 0
    if var_order == None:
        var_order = list(csp.domains)
    for sol in dfs_solver_before(csp.constraints, csp.domains, {}, var_order):
        return sol
def test_dfs_solver3(csp_problem):

```

```

solution = dfs_solve3(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:])

```

```

In [33]: probelm_string = generate_problem_solvable(4)
csp_problem_cost = create_CSP_from_spec(probelm_string)
test_dfs_solver2(csp_problem_cost)
num_expanded

```

```

task, t1 3
task, t2 3
task, t3 3
task, t4 1
constraint, t2 same-day t3
constraint, t4 same-day t3
domain, t4 tue
domain, t3 wed
domain, t1 ends-by wed 11am 29
domain, t2 ends-by thu 1pm 44
domain, t3 ends-by mon 3pm 29
domain, t4 ends-by wed 3pm 15
DEBUG: Heuristic cost for this node is: 1305
No solution

```

Out[33]: 26900

```

In [34]: csp_problem_cost = create_CSP_from_spec(probelm_string)
test_dfs_solver3(csp_problem_cost)
num_expanded

```

```

task, t1 3
task, t2 3
task, t3 3
task, t4 1
constraint, t2 same-day t3
constraint, t4 same-day t3
domain, t4 tue
domain, t3 wed
domain, t1 ends-by wed 11am 29
domain, t2 ends-by thu 1pm 44
domain, t3 ends-by mon 3pm 29
domain, t4 ends-by wed 3pm 15
DEBUG: Heuristic cost for this node is: 1305
No solution

```

Out[34]: 8150

```

In [35]: import matplotlib.pyplot as plt

```

```

n_values = range(2, 9)

results_solver2 = []
results_solver3 = []

for n in n_values:
    print(f"Testing for problem size N={n}...")

```

```

problem_string = generate_problem_solvable(n)

csp_problem = create_CSP_from_spec(problem_string)
test_dfs_solver2(csp_problem)
results_solver2.append(num_expanded)

test_dfs_solver3(csp_problem)
results_solver3.append(num_expanded)

print("Comparison finished. Generating plot...")

plt.figure(figsize=(10, 6))
plt.plot(n_values, results_solver2, marker='o', linestyle='-', label='DFS with h')
plt.plot(n_values, results_solver3, marker='x', linestyle='--', label='DFS without h')

plt.title('Comparison of Expanded Nodes vs. Problem Size ($N$)')
plt.xlabel('Problem Size ($N$)')
plt.ylabel('Number of Expanded Nodes (log scale)')
plt.yscale('log')
plt.legend()
plt.grid(True, which="both", linestyle="--", linewidth=0.5)
plt.tight_layout()

```

```

Testing for problem size N=2...
task, t1 4
task, t2 4
constraint, t1 before t2
domain, t2 tue
domain, t1 ends-by thu 1pm 12
domain, t2 ends-by wed 1pm 28
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 2
t1: mon 9am
t2: tue 9am
Nodes expanded to reach solution: 2
t1: mon 9am
t2: tue 9am
Testing for problem size N=3...
task, t1 4
task, t2 2
task, t3 1
constraint, t3 before t2
domain, t1 mon
domain, t1 ends-by tue 11am 39
domain, t2 ends-by thu 3pm 12
domain, t3 ends-by thu 1pm 42
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 39
t1: mon 9am
t2: mon 10am
t3: mon 9am
Nodes expanded to reach solution: 39
t1: mon 9am
t2: mon 10am
t3: mon 9am
Testing for problem size N=4...
task, t1 1
task, t2 2
task, t3 4
task, t4 1
constraint, t2 same-day t4
constraint, t1 same-day t2
domain, t3 fri
domain, t1 fri
domain, t1 ends-by mon 3pm 11
domain, t2 ends-by wed 11am 50
domain, t3 ends-by thu 1pm 32
domain, t4 ends-by thu 1pm 33
DEBUG: Heuristic cost for this node is: 1769
Nodes expanded to reach solution: 6272
t2: fri 9am
t4: fri 9am
t3: fri 9am
t1: fri 9am
Nodes expanded to reach solution: 56
t1: fri 9am
t2: fri 9am
t3: fri 9am
t4: fri 9am
Testing for problem size N=5...
task, t1 3
task, t2 3
task, t3 2

```

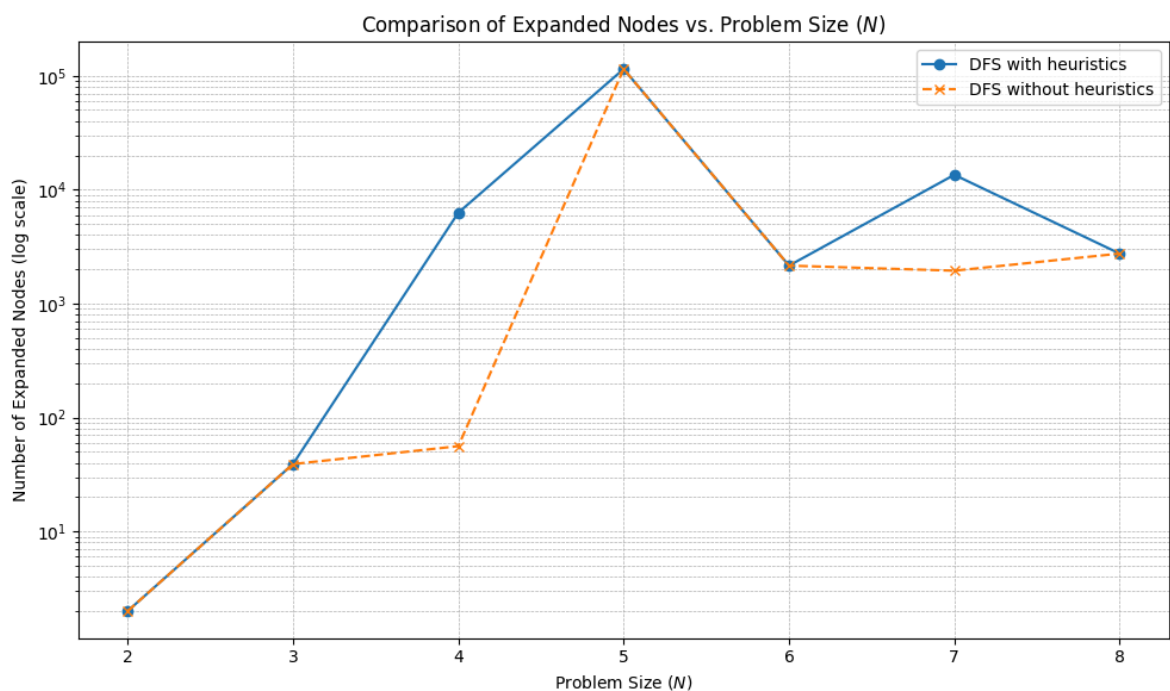
```
task, t4 1
task, t5 4
constraint, t2 same-day t3
constraint, t5 before t1
domain, t4 thu
domain, t5 thu
domain, t1 ends-by tue 11am 34
domain, t2 ends-by thu 3pm 35
domain, t3 ends-by wed 1pm 50
domain, t4 ends-by thu 3pm 43
domain, t5 ends-by thu 1pm 48
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 114499
t1: thu 1pm
t2: mon 9am
t3: mon 9am
t4: thu 9am
t5: thu 9am
Nodes expanded to reach solution: 114499
t1: thu 1pm
t2: mon 9am
t3: mon 9am
t4: thu 9am
t5: thu 9am
Testing for problem size N=6...
task, t1 4
task, t2 3
task, t3 2
task, t4 4
task, t5 2
task, t6 2
constraint, t4 before t2
constraint, t3 before t1
constraint, t1 same-day t4
domain, t3 wed
domain, t1 ends-by thu 11am 47
domain, t2 ends-by wed 11am 22
domain, t3 ends-by thu 3pm 48
domain, t4 ends-by mon 1pm 37
domain, t5 ends-by wed 3pm 35
domain, t6 ends-by mon 1pm 37
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 2152
t1: wed 11am
t2: wed 1pm
t3: wed 9am
t4: wed 9am
t5: mon 9am
t6: mon 9am
Nodes expanded to reach solution: 2152
t1: wed 11am
t2: wed 1pm
t3: wed 9am
t4: wed 9am
t5: mon 9am
t6: mon 9am
Testing for problem size N=7...
task, t1 1
task, t2 3
task, t3 4
```

```

task, t4 4
task, t5 2
task, t6 2
task, t7 3
constraint, t4 before t6
constraint, t7 same-day t5
constraint, t1 before t7
domain, t7 wed
domain, t1 tue
domain, t1 ends-by mon 11am 12
domain, t2 ends-by wed 11am 21
domain, t3 ends-by thu 11am 28
domain, t4 ends-by thu 3pm 16
domain, t5 ends-by wed 1pm 43
domain, t6 ends-by mon 3pm 39
domain, t7 ends-by tue 3pm 48
DEBUG: Heuristic cost for this node is: 1284
Nodes expanded to reach solution: 13487
t2: mon 9am
t3: mon 9am
t4: mon 9am
t5: wed 9am
t6: mon 1pm
t1: tue 9am
t7: wed 9am
Nodes expanded to reach solution: 1943
t1: tue 9am
t2: mon 9am
t3: mon 9am
t4: mon 9am
t5: wed 9am
t6: mon 1pm
t7: wed 9am
Testing for problem size N=8...
task, t1 1
task, t2 3
task, t3 2
task, t4 1
task, t5 3
task, t6 2
task, t7 3
task, t8 3
constraint, t1 same-day t3
constraint, t3 before t6
constraint, t5 same-day t2
constraint, t5 before t3
domain, t7 tue
domain, t1 ends-by tue 3pm 39
domain, t2 ends-by thu 11am 15
domain, t3 ends-by wed 3pm 25
domain, t4 ends-by tue 11am 14
domain, t5 ends-by thu 3pm 48
domain, t6 ends-by tue 3pm 17
domain, t7 ends-by thu 11am 44
domain, t8 ends-by thu 1pm 44
DEBUG: Heuristic cost for this node is: 0
Nodes expanded to reach solution: 2746
t1: mon 9am
t2: mon 9am
t3: mon 12pm

```

t4: mon 9am
 t5: mon 9am
 t6: mon 2pm
 t7: tue 9am
 t8: mon 9am
 Nodes expanded to reach solution: 2746
 t1: mon 9am
 t2: mon 9am
 t3: mon 12pm
 t4: mon 9am
 t5: mon 9am
 t6: mon 2pm
 t7: tue 9am
 t8: mon 9am
 Comparison finished. Generating plot...



Answers for Question 5

Write the other answers here.

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 [36]: # Code for Question 6
# Place a copy of your code here and run it in the relevant cell
def partition_domain(dom, k):

```

```

dom_list = list(dom)
n = len(dom_list)
if k > n:
    return [{elem} for elem in dom_list]

partitions = []
start_index = 0
base_size = n // k
remainder = n % k
for i in range(k):
    size = base_size + 1 if i < remainder else base_size
    end_index = start_index + size
    partition = set(dom_list[start_index:end_index])
    partitions.append(partition)
    start_index = end_index

return partitions

```

```

In [37]: class Search_with_AC_from_Cost_CSP(Search_problem):
    def __init__(self, csp, k):
        self.k = k
        self.cons = Con_solver(csp)
        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
        self.csp = csp

    def is_goal(self, node):
        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):
        neighs = []
        var = select(x for x in node.domains if len(node.domains[x]) > 1)
        if var:
            partitions = partition_domain(node.domains[var], self.k)
            self.display(2, f"Splitting {var} into {self.k} partitions...")
            to_do = self.cons.new_to_do(var, None)
            for dom in partitions:
                newdoms = node.domains | {var: dom}
                cons_doms = self.cons.make_arc_consistent(newdoms, to_do)

                if all(len(cons_doms[v]) > 0 for v in cons_doms):
                    csp_node = CSP_with_Cost(cons_doms, self.durations, self.con
                                                self.cost_functions, self.soft_day_
                    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

```

```
In [38]: problem=generate_problem_solvable(10)
csp_problem = create_CSP_from_spec(problem)
k=2
solver = GreedySearcher(Search_with_AC_from_Cost_CSP(csp_problem,k))
test_csp_solver(solver)
solver.num_expanded
```

[illegible]

```

DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1633
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1605
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1739
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1665
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1628
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 4091
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 2841
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1741
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1641
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
DEBUG: Heuristic cost for this node is: 1591
t1: wed 9am
t2: mon 9am
t3: mon 9am
t4: mon 9am
t5: mon 9am
t6: thu 9am
t7: thu 9am
t8: mon 9am
t9: mon 1pm
t10: mon 9am
cost: 1591

```

Out[38]: 33

```

In [41]: import matplotlib.pyplot as plt
k_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]
problem = generate_problem_solvable(20)
csp_problem=create_CSP_from_spec(problem)
results = {}
for k in k_values:
    search_problem = Search_with_AC_from_Cost_CSP(csp_problem, k)
    solver = GreedySearcher(search_problem)
    test_csp_solver(solver)
    nodes_expanded = solver.num_expanded
    results[k] = nodes_expanded

```

```
print("="*40)
print("Nodes Expanded vs. k Value")
print("="*40)
for k, nodes in results.items():
    print(f"k = {k:2d} | Nodes Expanded: {nodes}")
print("="*40)

plt.figure(figsize=(10, 6))
plt.plot(list(results.keys()), list(results.values()), marker='o', linestyle='-')
plt.title('Impact of k (Domain Partitions) on Search Effort')
plt.xlabel('k Value (Number of Partitions)')
plt.ylabel('Number of Nodes Expanded')
plt.xticks(k_values) # Ensure x-axis ticks match your tested k values
plt.grid(True)
plt.show()
```

task, t1 4
task, t2 4
task, t3 2
task, t4 1
task, t5 4
task, t6 4
task, t7 3
task, t8 2
task, t9 1
task, t10 4
task, t11 3
task, t12 2
task, t13 1
task, t14 3
task, t15 1
task, t16 2
task, t17 3
task, t18 4
task, t19 3
task, t20 1
constraint, t2 same-day t8
constraint, t15 before t7
constraint, t9 same-day t4
constraint, t8 same-day t10
constraint, t9 same-day t17
constraint, t6 same-day t16
constraint, t19 before t20
constraint, t11 same-day t7
constraint, t1 same-day t16
constraint, t17 same-day t7
domain, t19 thu
domain, t17 tue
domain, t2 thu
domain, t1 ends-by tue 11am 13
domain, t2 ends-by wed 3pm 30
domain, t3 ends-by mon 11am 12
domain, t4 ends-by thu 1pm 43
domain, t5 ends-by wed 3pm 47
domain, t6 ends-by wed 11am 29
domain, t7 ends-by tue 1pm 15
domain, t8 ends-by mon 1pm 44
domain, t9 ends-by thu 11am 26
domain, t10 ends-by wed 1pm 39
domain, t11 ends-by mon 1pm 15
domain, t12 ends-by thu 1pm 38
domain, t13 ends-by tue 11am 41
domain, t14 ends-by wed 1pm 32
domain, t15 ends-by thu 1pm 33
domain, t16 ends-by tue 3pm 29
domain, t17 ends-by mon 1pm 30
domain, t18 ends-by mon 11am 48
domain, t19 ends-by wed 3pm 43
domain, t20 ends-by tue 3pm 22
DEBUG: Heuristic cost for this node is: 2349
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8724
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7761
DEBUG: Heuristic cost for this node is: 7722

[illegible]

[illegible]

DEBUG: Heuristic cost for this node is: 8922
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7866
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7766
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8316
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 10428
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7761
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8058
DEBUG: Heuristic cost for this node is: 8610

[illegible]

[illegible]

DEBUG: Heuristic cost for this node is: 8316
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8272
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7761
DEBUG: Heuristic cost for this node is: 8724
DEBUG: Heuristic cost for this node is: 10441
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7748
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8034
DEBUG: Heuristic cost for this node is: 8346
DEBUG: Heuristic cost for this node is: 8646
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7746
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 8010
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7734
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8897
DEBUG: Heuristic cost for this node is: 7722

[illegible]

DEBUG: Heuristic cost for this node is: 7786
DEBUG: Heuristic cost for this node is: 8586
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7842
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8922
DEBUG: Heuristic cost for this node is: 10122
DEBUG: Heuristic cost for this node is: 11322
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 7866
DEBUG: Heuristic cost for this node is: 8874
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 8356
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 8250
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7766
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
DEBUG: Heuristic cost for this node is: 8316
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am

t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7748
DEBUG: Heuristic cost for this node is: 8698
DEBUG: Heuristic cost for this node is: 10402
DEBUG: Heuristic cost for this node is: 12106
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8010
DEBUG: Heuristic cost for this node is: 8298
DEBUG: Heuristic cost for this node is: 8586
DEBUG: Heuristic cost for this node is: 8874
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7746
DEBUG: Heuristic cost for this node is: 7758
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7734
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8756
DEBUG: Heuristic cost for this node is: 9884
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7737
DEBUG: Heuristic cost for this node is: 7752

[illegible]

DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7842
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8874
DEBUG: Heuristic cost for this node is: 10026
DEBUG: Heuristic cost for this node is: 11178
DEBUG: Heuristic cost for this node is: 12330
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 7866
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 8356
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 8250
DEBUG: Heuristic cost for this node is: 8294
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7766
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722

[illegible]

[illegible]

DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7842
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8874
DEBUG: Heuristic cost for this node is: 10026
DEBUG: Heuristic cost for this node is: 10170
DEBUG: Heuristic cost for this node is: 11274
DEBUG: Heuristic cost for this node is: 12378
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 7866
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 8356
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 8228
DEBUG: Heuristic cost for this node is: 8272
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
DEBUG: Heuristic cost for this node is: 7788
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8316
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7774
DEBUG: Heuristic cost for this node is: 8711
DEBUG: Heuristic cost for this node is: 10402
DEBUG: Heuristic cost for this node is: 10441
DEBUG: Heuristic cost for this node is: 12132

[illegible]

[illegible]

DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7866
DEBUG: Heuristic cost for this node is: 8970
DEBUG: Heuristic cost for this node is: 10074
DEBUG: Heuristic cost for this node is: 11178
DEBUG: Heuristic cost for this node is: 11322
DEBUG: Heuristic cost for this node is: 12426
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 8356
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 8228
DEBUG: Heuristic cost for this node is: 8272
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
DEBUG: Heuristic cost for this node is: 7766
DEBUG: Heuristic cost for this node is: 7788
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 8316
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7774
DEBUG: Heuristic cost for this node is: 8711
DEBUG: Heuristic cost for this node is: 10402
DEBUG: Heuristic cost for this node is: 10428
DEBUG: Heuristic cost for this node is: 12106
DEBUG: Heuristic cost for this node is: 12132
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7782

[illegible]

[illegible]

DEBUG: Heuristic cost for this node is: 10074
DEBUG: Heuristic cost for this node is: 11178
DEBUG: Heuristic cost for this node is: 11274
DEBUG: Heuristic cost for this node is: 12330
DEBUG: Heuristic cost for this node is: 12426
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 8356
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 8228
DEBUG: Heuristic cost for this node is: 8272
DEBUG: Heuristic cost for this node is: 8316
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
DEBUG: Heuristic cost for this node is: 7766
DEBUG: Heuristic cost for this node is: 7788
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7774
DEBUG: Heuristic cost for this node is: 8698
DEBUG: Heuristic cost for this node is: 8724
DEBUG: Heuristic cost for this node is: 10402
DEBUG: Heuristic cost for this node is: 10428
DEBUG: Heuristic cost for this node is: 12106
DEBUG: Heuristic cost for this node is: 12132
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752
DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7722

[illegible]

[illegible]

DEBUG: Heuristic cost for this node is: 8970
DEBUG: Heuristic cost for this node is: 10026
DEBUG: Heuristic cost for this node is: 10122
DEBUG: Heuristic cost for this node is: 11178
DEBUG: Heuristic cost for this node is: 11274
DEBUG: Heuristic cost for this node is: 12330
DEBUG: Heuristic cost for this node is: 12426
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 8356
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 8228
DEBUG: Heuristic cost for this node is: 8272
DEBUG: Heuristic cost for this node is: 8294
DEBUG: Heuristic cost for this node is: 8316
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
DEBUG: Heuristic cost for this node is: 7766
DEBUG: Heuristic cost for this node is: 7788
t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7748
DEBUG: Heuristic cost for this node is: 7774
DEBUG: Heuristic cost for this node is: 8698
DEBUG: Heuristic cost for this node is: 8724
DEBUG: Heuristic cost for this node is: 10402
DEBUG: Heuristic cost for this node is: 10428
DEBUG: Heuristic cost for this node is: 12106
DEBUG: Heuristic cost for this node is: 12132
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7752

[illegible]

[illegible]

DEBUG: Heuristic cost for this node is: 7782
DEBUG: Heuristic cost for this node is: 7812
DEBUG: Heuristic cost for this node is: 7842
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7818
DEBUG: Heuristic cost for this node is: 8874
DEBUG: Heuristic cost for this node is: 8970
DEBUG: Heuristic cost for this node is: 10026
DEBUG: Heuristic cost for this node is: 10122
DEBUG: Heuristic cost for this node is: 11178
DEBUG: Heuristic cost for this node is: 11274
DEBUG: Heuristic cost for this node is: 12330
DEBUG: Heuristic cost for this node is: 12426
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7770
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7787
DEBUG: Heuristic cost for this node is: 7852
DEBUG: Heuristic cost for this node is: 7917
DEBUG: Heuristic cost for this node is: 8356
DEBUG: Heuristic cost for this node is: 8184
DEBUG: Heuristic cost for this node is: 8228
DEBUG: Heuristic cost for this node is: 8250
DEBUG: Heuristic cost for this node is: 8272
DEBUG: Heuristic cost for this node is: 8294
DEBUG: Heuristic cost for this node is: 8316
DEBUG: Heuristic cost for this node is: 7722
DEBUG: Heuristic cost for this node is: 7744
DEBUG: Heuristic cost for this node is: 7766
DEBUG: Heuristic cost for this node is: 7788

t1: mon 9am
t2: thu 9am
t3: mon 9am
t4: tue 9am
t5: mon 9am
t6: mon 9am
t7: tue 9am
t8: thu 9am
t9: tue 9am
t10: thu 9am
t11: tue 9am
t12: mon 9am
t13: mon 9am
t14: mon 9am
t15: mon 9am
t16: mon 9am
t17: tue 9am
t18: mon 9am
t19: thu 9am
t20: thu 12pm
cost: 7722

=====
Nodes Expanded vs. k Value

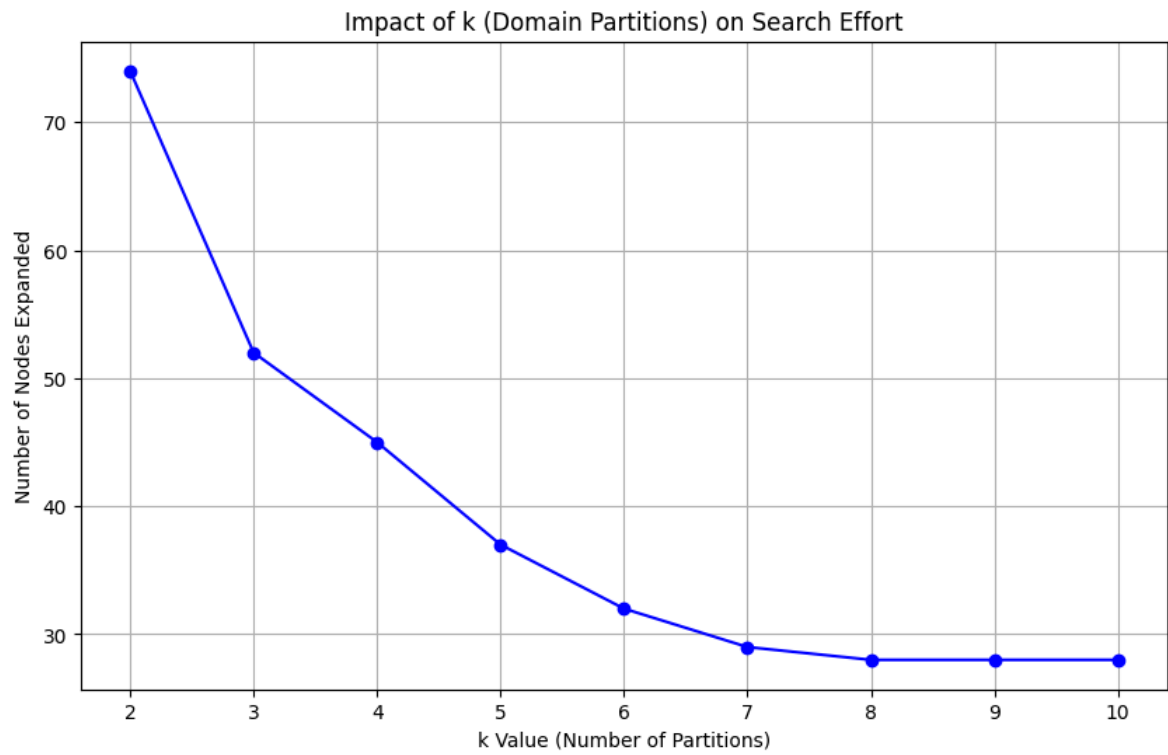
=====
k = 2 | Nodes Expanded: 74
k = 3 | Nodes Expanded: 52
k = 4 | Nodes Expanded: 45
k = 5 | Nodes Expanded: 37
k = 6 | Nodes Expanded: 32
k = 7 | Nodes Expanded: 29

k = 8 | Nodes Expanded: 28

k = 9 | Nodes Expanded: 28

k = 10 | Nodes Expanded: 28

=====



Answers for Question 6

Write the other answers here.