CPSC433 Assignment

Chan, Chung Loong, 30233802
Chien, Jing Wei, 30233818
Vo, Kenny, 30065390
Haji, Imran, 30141571
Chang, Tiffany, 30069445
Khan, Mohammad, 30103764
JC, Sicat, 30024897

October 27 2023

1 Introduction

This paper will describe two approaches in solving a scheduling problem as described in the given assignment. The first model approach will be based on a set based search model, and the second will be based on an and-tree-based search model.

2 Problem

In the assigned problem, there are courses and labs to be assigned to weekly time slots. Courses, Labs and slots are defined as:

$$Courses = \{c_1, \ldots, c_m\}$$

$$Labs = \{l_{11}, \ldots, l_{1k_1}, \ldots, l_{m1}, \ldots, l_{mk_m}\}$$

$$Slots = \{s_1, \ldots, s_n\}$$

where $l_{i1}, \ldots, l_{ik_i}$ denote the labs for course c_i. This also means that k_i can be 0, denoting that the course has no labs.

For each slot s_j, there is a limit to the number of courses and labs that can be assigned to a slot. This is denoted by $coursemax(s_j) \in \mathbb{N}$, and $labmax(s_j) \in \mathbb{N}$, where $\mathbb{N}$ represents the set of all natural numbers. We define \$ to be the symbol of an unassigned slot.

In addition to these sets, there is the function: $assign : Courses \cup Labs \rightarrow Slots$ which fulfills the two conditions:

- $Constr(assign) = true$ where $Constr$ tests the fulfillment of all hard constraints, which evaluates true if and only if all hard constraints are fulfilled.
- $Eval(assign)$ is minimal in the set of all possible assignments that fulfill $Constr$, where $Eval$ is the evaluation function that measures how well the assignment fulfills the soft constraints.

3 Set-Based Search

3.1 Overview

The problem seeks to assign courses and labs to appropriate time slots. This methodological approach is based on a set-based search paradigm.

Initially, the strategy focuses on satisfying all of the hard constraints provided by the problem. To this end, we deploy an or-tree-based search model. This model facilitates the generation of a population of solutions that adhere to those hard constraints. It is crucial to note that considerations regarding soft constraints are deferred to a later phase in the search process.

The or-tree-based search model is run multiple times and yields an initial population, size-bound by $initial\_pop$. This population serves as the foundational facts for the set-based search. To speed up the search process and maintain computational efficiency, there is a constant max_pop, which serves as an upper bound for the population size within the set-based search. The generation process uses the crossover operation on the slot assignments. The search model uses two extension rules: $Crossover$ and $Reduce$, described later on, with the choice of rule being arbitrated by the function f_{wert}.

Upon the population size exceeding the max_pop value, a reduction is triggered by f_{wert}. Otherwise, the function f_{select} is deployed to select the best transition resulting from applying $Crossover$. Upon a predefined number of generations being executed, or a specified time duration passing, the generation terminates.

The solution framework incorporates the $Eval$ function to evaluate the soft constraints provided in the assignment. We use the functions $Constr$ and its variant, $Constr^*$, to verify the fulfillment of the hard constraints. The latter, $Constr^*$, is used to assess partial solutions and is derived from the $Constr$ function.

3.2 Or-Tree-Based Search Model

$$Classes = \{Courses \cup Labs\}, t = |Classes|$$

$$Prob = \{\langle X_1, \ldots, X_t\rangle \mid X_i \in Slots \cup \{\$\}\}$$

i represents the index in $pr \in Prob$, corresponding to the index in $Classes$ which represents a specific course or lab. $Prob$ is a set of problem descriptions, where each one is represented by pr. Each pr within $Prob$ is a vector that contains a mapping for every course or lab to a specific slot. The courses and labs within pr are indexed based on an arbitrary order. It is important that the specific order is inconsequential, provided that each course and lab retains a unique index within pr, which has a length of t.

The alternative relation $Altern(pr, pr_1, \ldots, pr_n)$

$$Altern = \{\langle X_1, \ldots, X_i, \ldots, X_t\rangle, \langle X_1, \ldots, s_1, \ldots, X_t\rangle, \ldots, \langle X_1, \ldots, s_n, \ldots, X_t\rangle \mid X_i = \$, \forall s_j \in Slots, 1 \leq j \leq n\}$$

generates a set of alternative solutions by replacing the unassigned slot X_i in pr with every possible slot from $Slots$ and holds true if and only if, for a given solution vector pr of the form:

$$pr = \langle X_1, \ldots, X_i, \ldots X_t\rangle$$

the element X_i is an occurrence of \$ in pr. $\forall\ s_j \in Slots$, we can create a new solution pr_j by substituting X_i with s_j, yielding:

$$pr_j = \langle X_1, \ldots, s_j, \ldots, X_t\rangle$$

The criteria for solvability will be defined in later sections. Within our proposed solution, two primary functions, $Constr$ and $Constr^*$ are used to ensure compliance with the hard constraints. While $Constr$ evaluates the hard constraints in a complete solution, $Constr^*$ is an adapted variant, made to evaluate the hard constraints in a partial solution. It is important to note that $Constr^*$ is derived from $Constr$.

3.3 Or-Tree-Based Search Process

We define some functions to help with our search process:

$$f_{help} = \begin{cases} 1 & \text{if } X_i = \$ \\ 0 & \text{else} \end{cases}$$

$$f_{sum} = \sum_{i=1}^{t} f_{help}(X_i), \forall X_i \in pr_i$$

$$f_{score} = \begin{cases} 0 & \text{if } pr \text{ is solvable} \\ \infty & \text{if } pr \text{ is unsolvable} \\ f_{sum}(pr_i) & \text{otherwise} \end{cases}$$

- f_{help} acts as an indicator for unassigned slots.
- f_{sum} returns the count of unassigned slots within a given solution pr_i.
- $f_{score} : pr_i \rightarrow \mathbb{N} \cup \{0, \infty\}$, where $\mathbb{N}$ describes the set of all natural numbers, and ∞ is infinity, larger than any number in $\mathbb{N}$. f_{score} decides the next action at each tree depth, directing the control so that f_{leaf} can select one leaf to either continue expanding O_{tree}, or return a solution based on the f_{score} value.
- f_{leaf} selects the leaf with the lowest f_{score} value.

To clarify the process, the search operation proceeds in the following sequential steps:

1. Apply the function f_{score} to all leaf nodes of the tree.
2. Assess each leaf node against the solvability criteria. If it is a leaf node, represented as (pr_i, sol), meets the criteria, it is marked as solved by updating itself to (pr_i, yes) with $sol = yes$. The search operation then concludes.
3. If there are no solvable leaf nodes, the search control evaluates leaf nodes against the unsolvability criteria. Nodes identified as 'unsolvable' are processed sequentially from left to right. Their sol values are then updated to no.
4. If there are neither any solvable leaf nodes nor unsolvable leaf nodes, the search control identifies the leaf node with the lowest value of f_{score}. In cases where multiple leaf nodes share the same lowest value, a random valid node is selected. The function $Altern$ is then applied to this chosen leaf for further expansion.

3.4 Or-Tree-Based Search Instance

The initial state of the search is represented as:

$$s_0 = (pr, ?)$$

$$pr = \langle X_1, \ldots, X_t \rangle \, , \forall X_i \in pr, \, X_i = \$$$

Alternatively, pr can also be initialized based on a partial assignment provided as input.

The 'sol-entry' of a node is determined by the following criteria:

$$sol = \begin{cases} yes & \text{if } pr \text{ is solvable} \\ no & \text{if } pr \text{ is unsolvable} \\ ? & \text{otherwise} \end{cases}$$

The goal state $G_\vee$ is considered to be reached under two conditions: either when a leaf node is explicitly marked as (pr, yes) or upon the expiration of a predefined timer.

A solution pr is considered 'solved' if:

$$pr = \langle X_1, \ldots, X_t \rangle, \forall X_i \in pr, X_i \neq \$ \text{ and } Constr(pr) = true.$$

A solution pr is considered 'unsolvable' if:

$$pr = \langle X_1, \ldots, X_t \rangle, Constr(pr) = false$$

For partial assignments, we use the derivation function $Constr^*$ instead.

3.5 Set-Based Search Model

Upon the completion of the Or-Tree-Based search process, we now have a collection of potential solutions that fulfill all hard constraints. If there are duplicate solutions within the collection, we simply remove the duplicates and rerun the Or-Tree process again until we have a collection of solutions bounded by $initial\_pop$. The next phase involves employing a Set-Based Search to identify the most optimal solution from this collection.

$$f = \langle X_1, \ldots, X_t \rangle, \forall X_i \in f, X_i \neq \$, Constr(f) = true$$

$$F = \{f_1, \ldots, f_{initial\_pop}\}$$

$$Ext = \{A \rightarrow B | A, B \subseteq F, Crossover(A, B) \vee Reduce(A, B))\}$$

- The set of facts, F, is constructed from the potential solutions generated by the preceding Or-Tree-Based search, size bounded by $initial\_pop$ and without duplicates. Each vector f within F represents a potential solution, that fulfils all the hard constraints. The courses and labs within f are indexed based on an arbitrary order. It is important that the specific order is inconsequential, provided that each course and lab retains a unique index within f, which has a length of t.
- We introduce a collection of extension operations that modify potential solutions within the search space.
- The set Ext comprises extension operations which, we either apply $Crossover$ or $Reduce$.
- Each state s should have a length $initial\_pop \leq |s| \leq max\_pop$.

Detailed definitions of $Crossover$ and $Reduce$ will be elaborated upon in subsequent sections.

3.6 Set-Based Search Process

Given the population size $|s|$ where s is the current state, the function is defined as:

$$f_{wert} = \begin{cases} 1 & \text{if } |s| > max\_pop \\ 0 & \text{if } |s| \leq max\_pop \end{cases}$$

The function f_{wert} assesses the current size of the population relative to the predefined threshold max_pop. Depending on the current population size, the function will cause the search control to use the $Reduce$ operation, or continue with the $Crossover$ operation.

Based on the output of f_{wert}:

- If $f_{wert} = 1$, the population size exceeds max_pop, prompting the $Reduce$ operation.
- If $f_{wert} = 0$, the population size matches or is smaller than max_pop. The process continues with $Crossover$.

To decide on what the inputs for the $Crossover$ and $Reduce$ functions are, we have the following rubrics:

1. $Crossover : A = \{f_1, f_2\}$, where:
   (a) f_1 is a randomly selected fact of state s using the roulette wheel selection algorithm where the probability $p_1 = \frac{Eval(f_1)}{\sum_{k=1}^{len} Eval(f_k)}$, where $len = |s|$
   (b) f_2 is a randomly selected fact of state s using the roulette wheel selection algorithm where the probability $p_2 = \frac{Eval(f_2)}{\sum_{k=1}^{lenm} Eval(f_k)}$, where $lenm = |s \backslash \{f_1\}|$
2. $Reduce : A = \{f_1\}$, where:
   (a) $f_1 =$ the fact with the highest $Eval$ value.

We then define $Crossover$ and $Reduce$ functions as such.

3.6.1 Crossover

Crossover aims to combine elements from two distinct facts to produce a new solution. This process involves using a modified or-tree-based search to effectively merge traits from two parent solutions.

Given any $A, B \subseteq F$, with A containing two distinct individuals $\{f_1, f_2\}$, with $f_1 \neq f_2$ and B represents the parents and their offspring, denoted as $B = \{f_1, f_2, f_{child}\}$.

1. Consider individuals f_1 and f_2, both represented as $\langle s_{11}, s_{12}, \ldots, s_{1t}\rangle$ and $\langle s_{21}, s_{22}, \ldots, s_{2t}\rangle$ such that $\forall s_{1i}, s_{2i} \in Slots$, $1 \leq i \leq t$, and $f_1 \neq f_2$.
2. Create f_{child} by combining traits from f_1 and f_2 using an or-tree-based search with a modified f_{score} defined below.

Modified Or-Tree-Based Search:

The modified f_{score} function is then defined by:

$$f_{score} = \begin{cases} 0 & \text{if the solution is solvable} \\ 1 & \text{if the assigned slot is:} \\ & \quad \text{- Present in both } f_1 \text{ and } f_2 \text{ at the same index,} \\ & \quad \text{- and } Constr^* \text{ holds.} \\ 2 & \text{if the assigned slot is:} \\ & \quad \text{- Present in either } f_1 \text{ or } f_2 \text{ at the same index,} \\ & \quad \text{- and } Constr^* \text{ holds.} \\ 3 & \text{if only } Constr^* \text{ is true,} \\ \infty & \text{otherwise.} \end{cases}$$

f_{leaf} selects the leaf with the lowest f_{score} value. If there is a tie between multiple nodes, we randomly pick a valid node within the nodes with the lowest f_{score} values. The remaining processes are identical to the Or-Tree-based search as defined in earlier sections.

3.6.2 Reduce

Reduce is used to systematically prune the current set of potential solutions, ensuring efficient search and optimization processes.

For any subset A, where A comprises of the individual fact f where $f = \langle s_1, s_2, \ldots, s_t\rangle$ such that $\forall s_i \in Slots, 1 \leq i \leq t$,

1. Remove f from A.

3.7 Set-Based Search Instance

We denote the initial search state as s_0

$$s_0 = \{f_1, \ldots, f_{initial\_pop}\}$$

where $f_i, 1 \leq i \leq initial\_pop$, is a fact. This starting state represents the outcome from the preceding Or-Tree-Based search, initialized with a population of size $initial\_pop$.

The goal state G_{set} signifies the termination condition of our search. Formally, it is defined as:

$$G_{set} = \begin{cases} true & \text{if the search has completed the predetermined number of generations} \\ true & \text{if the search has exceeded a pre-defined timer} \\ false & \text{otherwise} \end{cases}$$

The search process will run until either the specified number of generations is reached or another stopping criterion is met, such as when a pre-defined timer runs out. The predetermined number of generations is set such that the $Eval$ value of the output solution is consistently tracked and shown to stabilise after a certain period of time. After which, the search is terminated.

At the conclusion of the search, the solution with the minimal evaluation value, as determined by the $Eval$ function, will be recognized as the optimal solution.

3.8 Example

Let us construct a simple example where

$Slots = \{s_1, s_2, s_3\}$, $Courses = \{c_1, c_2\}$, $Labs = \{l_{11}, l_{21}\}$
and $partassign = \langle \$, \$, \$, \$ \rangle$.

We index the courses and labs in the following order: $\langle c_1, c_2, l_{11}, l_{21} \rangle$.
The value below the tree node represents the f_{score} value for that node.

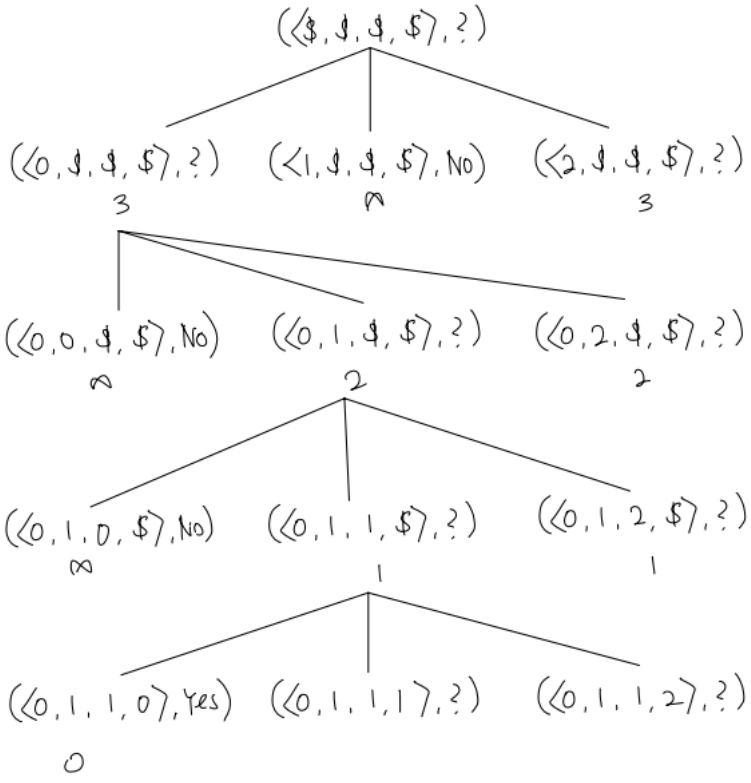
Hard Constraints:

1. $Constr$ and $Constr^*$ return false if c_1 is assigned to slot s_2
2. $Constr$ and $Constr^*$ return false if c_2 is assigned to slot s_1

Soft Constraints:

1. $Eval$ and $Eval^*$ returns 3 if c_1 is assigned to s_1.
2. $Eval$ and $Eval^*$ returns 1 if l_{21} is assigned to s_2.

Running the first or-tree-based search instance:



Assume that another valid potential solution is generated, i.e $\langle 2, 2, 1, 1 \rangle$.

We apply the crossover extension rule such that a child $\langle 0, 2, 2, 1 \rangle$ is generated.

We then call the $Eval$ function to evaluate against the soft constraints:

- $Eval(\langle 0, 1, 1, 0 \rangle)$ = 3
- $Eval(\langle 2, 2, 1, 1 \rangle)$ = 1
- $Eval(\langle 0, 2, 2, 1 \rangle)$ = 3

Hence, the optimal solution, $\langle 2, 2, 1, 1 \rangle$ or $\langle c_1$ in s_3, c_2 in s_3, l_{11} in s_2, l_{21} in $s_2 \rangle$ will be returned.

4 And-Tree-Based Search

4.1 Search Model

$$Classes = \{Courses \cup Labs\}, t = |Classes|$$

$$Prob = \{\langle X_1, \ldots, X_t\rangle \mid X_i \in Slots \cup \{\$\}\}$$

i represents the index in $pr \in Prob$, corresponding to the index in $Classes$ which represents a specific course or lab. $Prob$ is a set of problem descriptions, where each one is represented by pr. Each pr within $Prob$ is a vector that contains a mapping for every course or lab to a specific slot. The courses and labs within pr are indexed based on an arbitrary order. It is important that the specific order is inconsequential, provided that each course and lab retains a unique index within pr, which has a length of t.

The Div relation $Div(pr, pr_1, \ldots, pr_n)$

$$Div = \{\langle X_1, \ldots, X_i, \ldots, X_t\rangle, \langle X_1, \ldots, s_1, \ldots, X_t\rangle, \ldots, \langle X_1, \ldots, s_n, \ldots, X_t\rangle \mid X_i = \$, \forall s_j \in Slots, 1 \leq j \leq n\}$$

holds true if and only if, for a given solution vector pr of the form:

$$pr = \langle X_1, \ldots, X_i, \ldots X_t\rangle$$

the element X_i is an occurrence of \$ in pr. With $\forall\ s_j \in Slots$, we can create a new solution pr' by substituting X_i with s_j, yielding:

$$pr' = \langle X_1, \ldots, s_j, \ldots, X_t\rangle$$

Essentially, Div expands the tree by replacing the unassigned slot X_i in pr with each possible slot from $Slots$.

Within our proposed solution, two primary functions, $Constr$ and $Constr^*$ are used to ensure compliance with the hard constraints. While $Constr$ evaluates the hard constraints in a complete solution, $Constr^*$ is an adapted variant, made to evaluate the hard constraints in a partial solution. It is important to note that $Constr^*$ is derived from $Constr$.

4.2 Search Process

$$f_{score} = \begin{cases} \infty & \text{if } Constr(pr_i) = false \\ Eval^*(pr_i) & \text{otherwise} \end{cases}$$

- The function f_{leaf} assesses the leaves of the tree and selects the leaf with the lowest f_{score} value. The search control gives precedence to leaves with the lowest f_{score} value. In the case of ties, the leftmost leaf is chosen, depth-first.
- The function f_{score} is defined as: $f_{score} : pr_i \rightarrow \mathbb{N} \cup \{\infty, 0\}$, where $\mathbb{N}$ describes the set of all natural numbers, and ∞ is infinity, larger than any number in $\mathbb{N}$.
- f_{score} evaluates a penalty score of an assignment based on the soft and hard constraints. For partial assignments, we use $Eval^*$ and $Constr^*$ instead of $Eval$ and $Constr$, which evaluate the complete assignment. $Eval^*$ only evaluates certain soft constraints such as preference values. Other soft constraints are not considered.

f_{trans} simply chooses the first instance of X_i that is unassigned, ie. $X_i = \$$.

To populate the slots, the strategy progresses sequentially from left to right, assigning slots to elements of pr from the smallest to the largest indices.

To account for the cases where partial assignments with all slots assigned violates constraints, backtracking will be implemented to solve such edge cases.

The search operation proceeds in the following sequential steps:

1. Prioritize the leaf node with the lowest f_{score} value and apply Div. If there is a tie between multiple leaves with the lowest f_{score}, the leftmost leaf is chosen, depth-first.
2. Mark nodes as solved by setting $sol = yes$ if the criteria for solvability is fulfilled.
3. Opt for the leaf with the minimal f_{score} value, marked as $(pr, ?)$ and reapply Div. Repeat the process.

4.3 Search Instance

The initial state of the search is represented as:

$$s_0 = (pr, ?)$$

$$pr = \langle X_1, \ldots, X_t \rangle \, , \forall X_i \in pr, \, X_i = \$$$

Alternatively, pr can also be initialized based on a partial assignment provided as input.

A solution pr is considered 'solved' if:

$$pr = \langle X_1, \ldots, X_t \rangle, \forall X_i \in pr, X_i \neq \$ \text{ and } Constr(pr) = true.$$

The sol value of a node is determined by the following criteria:

$$sol = \begin{cases} yes & \text{if } pr \text{ is solvable} \\ ? & \text{otherwise} \end{cases}$$

The goal state $G_\wedge$ is achieved when:

- All remaining leaves are marked (pr, yes), or
- A pre-set timer, defining the maximum search duration, expires.

The optimal solution corresponds to the leaf node that exhibits the smallest f_{score} value.

4.4 Example

Let us construct a simple example where

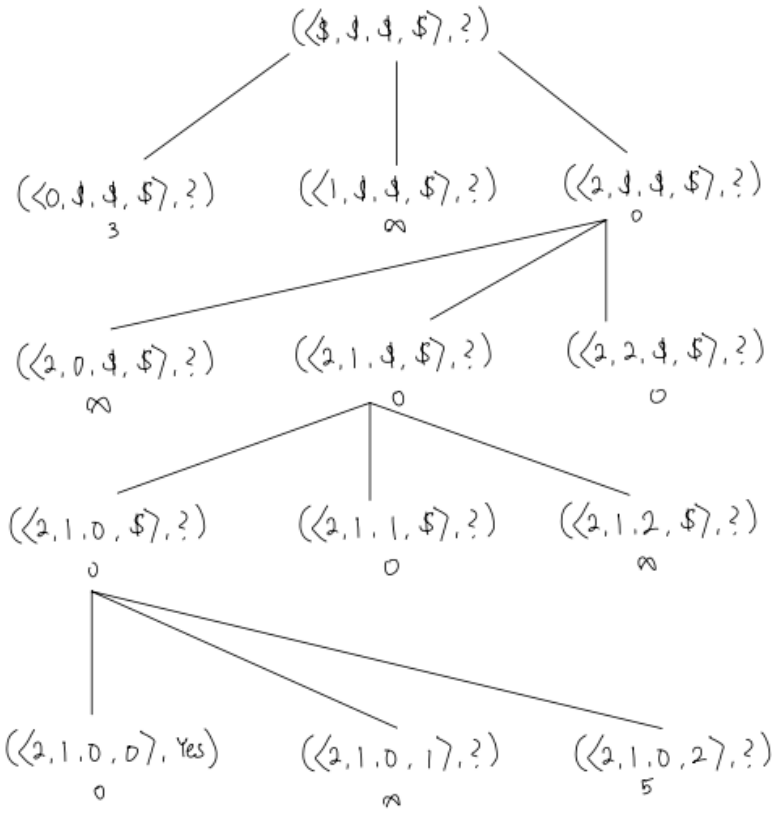
$Slots = \{s_1, s_2, s_3\}$, $Courses = \{c_1, c_2\}$, $Labs = \{l_{11}, l_{21}\}$ and $partassign = \langle \$, \$, \$, \$ \rangle$.

We index the courses and labs in the following order: $\langle c_1, c_2, l_{11}, l_{21} \rangle$.
As such, c_1 has index 0, c_2 has index 1, so on and so forth.
The value below the tree node represents the f_{score} value for that node.
Hard Constraints:

1. $Constr$ and $Constr^*$ returns false if c_1 is assigned to slot s_2
2. $Constr$ and $Constr^*$ returns false if c_2 is assigned to slot s_1

Soft Constraints:

1. $Eval$ and $Eval^*$ returns 3 if c_1 is assigned to s_1.
2. $Eval$ and $Eval^*$ returns 5 if l_{21} is assigned to s_3.
3. $Eval$ and $Eval^*$ returns 0 otherwise.



1_{st} Div: The model selects the right-most leaf to further divide as it has the lowest f_{score}.

2_{nd} Div: Since there is a tie breaker between leaf 2 and 3, the left-most leaf (leaf 2) is selected in the event of a tie breaker.

3_{rd} Div: Another tie breaker between leaf 1 and 2 results in the model selecting leaf 1.

4_{th} Div: Goal condition is reached for leaf 1 and is marked (pr, yes)

In this example shown, the final optimized solution is $\langle 2, 1, 0, 0\rangle$ or $\langle c_1$ in s_3, c_2 in s_2, l_{11} in s_1, l_{21} in $s_1\rangle$.