TWO SEARCH PARADIGMS FOR OPTIMAL ASSIGNMENT OF EVENTS TO TIME SLOTS

SHANE SIMS

1. Introduction

The task of this paper is to define a search system that when implemented, will find the optimal assignment of time slots to courses and labs. To that end, I will present a set based search model and control definition instantiations. Note: all variables not explicitly defined in the following pages before their first use are then as defined in the assignment description, as found on the course website.

2. Set Based Search Instantiation

The instantiation of our set based search is that of a genetic algorithm (GA). However, there is an inherent problem in using a GA in its standard form to approach an optimization problem like the one in this assignment, which contains both hard and soft constraints. The source of this problem lies in the fact that a standard GA allows all potential solutions to a problem, without regard to validity when determining the initial population (which is chosen at random) or from one genetic operation to the next; there being no assurance that a population that contains valid (i.e. hard constraint satisfying) individuals will even be reached in within certain time constraints- to say nothing of how this population approaches an optimized solution.

To counter this problem, our search system (following the suggestion of the course professor), while being based on a set based GA, will employ an or-tree based search in two situations. The first time the or-tree based search will be used will be in determining the population of the start state, $state_0$ (so notated to avoid confusion with time slots). To this end, the or-tree based search will operate by searching for a class configuration that simply satisfies all of the (seven) hard constraints. An element of randomness is incorporated into the control so as to ensure diversity in the individuals produced. The result of running this process several times will be a start state that contains only individuals that satisfy the hard constraints of the given problem. The GA will then take over in the standard way (described below) to evolve individuals towards optimizing soft constraint satisfaction.

The second time we will employ the or-tree search assistant is in the application of the genetic operation and this will be described more fully in the sections to come. In the next section we provide a description of the or-tree based search assistant, whose definitions will prove useful in the description of the set based search that will follow. It is the latter that forms the dominant part of the overall search system and it is for this reason that we classify our system as set based.

3. Or-tree Based Search Definitions

Or-tree Based Search Model: $\mathcal{A}_\vee = (\mathcal{S}_\vee, \mathcal{T}_\vee)$, where the set of problem descriptions and the alternatives relation are defined as follows:

- Let $(c_1, ..., l_{1_{k_1}}, ..., c_m, ..., l_{m_{k_m}})$ be the index vector for elements of $\mathcal{P}rob$
- Let $k = m + k_1, ..., k_m$ (latter variables as defined in the assignment description).
- $\mathcal{P}rob = \{(s'_1, ..., s'_k) \mid 1 \leq s'_i \leq n \text{ or } s'_i = \$ \text{ if } undecided, 1 \leq i \leq k\}$

  That is, $\mathcal{P}rob$ is the set of problem descriptions, which we describe as vectors representing possible course/lab time slot assignments in any stage of completeness. If no time slot has been assigned to a particular course/lab, this is represented with **$** in its position in the vector, according to the index vector given. Otherwise this position holds the integer subscript of the assigned time slot.

We define when a problem is solved and when a problem is unsolvable, as follows:

- Let $Constr^*$ be the extension of $Constr$ (as defined in the assignment description) to include partial solutions. Then:
- A given $pr \in \mathcal{P}rob$ is considered **solved** when pr contains no **$** elements and $Constr(pr) = True$.
- A given $pr \in \mathcal{P}rob$ is considered **unsolvable** when $Constr^*(pr) = False$, indicating a hard constrain violation.

Let s'_i be the left most element of a given $pr \in Prob$ such that $s'_i = \$$, where $1 \leq i \leq n$.

- $Altern((s'_1, ..., s'_i = \$, ..., s'_n)) = \{(s'_1, ..., s'_i = 1, ..., s'_n), ..., (s'_1, ..., s'_i = n, ..., s'_n)\}$

  That is, $Altern$ is a relation that when given a $pr \in \mathcal{P}rob$ containing a **$** in an arbitrary but specific position s'_i, we get as output all possible time slot assignments to this index element. That is, $Albern$ provides all possible solutions to this problem.

As mentioned previously, we require two search controls for our or-tree search. Again, the first will be used to generate population of valid (hard constraint satisfying) individuals for the start state of the set based search. The second will be used to implement the genetic operation used in the set based search. We define each control as in turn:

Or-tree Based Search Control One: $\mathcal{K}_{\vee -1}$

Let $(pr_1, ?), ..., (pr_0)$ be the open leaves in a given state.

Let $S\_unassgn(pr) = |\{s'_i | s'_i \in pr \wedge s'_i = \$, 1 \leq i \leq n\}|$

$\mathcal{K}_{\vee -1}$ selects the leaf to work on and transition to according to the following algorithm:

- **If** one of the problems pr is solved **then** perform the transition that changes the solution entry in this node from ? to *yes*. The search stops as the goal state has been reached.

- **Else-if** one of the problems is unsolvable **then** perform the transition that changes the solution entry in this node from ? to *no*. This branch will never produce a valid solution.

- **Else** select the unsolved leaf $(pr, ?)$ such that $S\_unassgn(pr) = \mathbf{min}(S\_unassgn(pr_1), ..., S\_unassgn(pr_0))$.
  If there is more than one leaf with this property
  then select the one that occurs deepest in the tree
  If there is more than one leaf at this level and the level is immediately below the root
  then select one randomly
  Else Selects the alternative from the set produced by *Altern*, by choosing the one that assigns the lowest slot number.

That is, after transitioning solved and unsolvable leaves to change their solution entry, the control will operate by choosing the leaf that has the most time slot assignments complete while still not being unsolvable. There will be cases with many leaves of the tree at equal distance from the solution and one from among the many will need to be chosen for processing. One example of this is the level immediately below the root, where all leaves will be unsolved and contain the same number of assignments. Because this search will be used to populate the start state of the GA, randomness is a requirement. To this end, when immediately below the root, we choose one at random to transition. The choice to select the alternative solution to the problem based on the one containing the new slot assignment that has the lowest number in all other cases was made for its simplicity, but could easily be changed to a random selection if required.

Or-tree Based Search Control Two: $\mathcal{K}_{\vee -2}(assign', assign'')$

Let $assign'$ and $assign''$ be valid assignments so that $Constr(assign') = Constr(assign'') = True$.

Let $(pr_1, ?), ..., (pr_0, ?)$ be the open leaves in a given state.

Let $S\_unassgn(pr) = |\{s'_i | s'_i \in pr \wedge s'_i = \$, 1 \leq i \leq n\}|$

$\mathcal{K}_{\vee-2}$ selects the leaf to work on and transition to according to the following algorithm:

- **If** one of the problems pr is solved **then** perform the transition that changes the solution entry in this node from ? to *yes*. The search stops as the goal state has been reached.

- **Else-if** one of the problems is unsolvable **then** perform the transition that changes the solution entry in this node from ? to *no*. This branch will never produce a valid solution.

- **Else** select the unsolved leaf $(pr, ?)$ such that $S\_unassgn(pr) = \mathbf{min}(S\_unassgn(pr_1), ..., S\_unassgn(pr_0))$.
  If there is more than one leaf with this property
  then select the one that occurs deepest in the tree
  If there is more than one leaf with this property
  then select one as follows:

  - **If** the solutions containing s_i' and s_i'' from $assign'$ and $assign''$ respectively, are both among the alternatives
    then choose one of these randomly
  - **Else-if** only one of the solutions containing s_i' and s_i'' are among the alternatives
    then select this solution
  - **Else** select one of the alternative solutions randomly

This search control operates much like the first one, except for how solutions from the set of alternatives are chosen. As can be seen, this is done to highly favour the parents by choosing the solution that matches one of the parents whenever possible. This is done to provide a crossover effect so that the child produced will share many of its characteristics with its parents. The choice to choose the solution to a given problem randomly in the event where a parent-like solution is not among the alternatives, is done to provide a built in mutation effect.

4. Example Or-tree Based Search Control One

We first demonstrate the operation of our or-tree based assistant using $\mathcal{K}_{or-1}$ with the following small example, based on the hard constraints of the general problem:

- $Courses = \{c_1, c_2, c_3\}$
- $Labs = \{l_{11}, l_{12}, l_{21}, l_{31}, l_{32}, l_{33}\}$
- $Slots = \{s_1, s_2\}$
  - $coursemax(s_1) = coursemax(s_2) = 4$
  - $labmax(s_1) = labmax(s_2) = 6$
- $not - compatible(a, b) = \emptyset$
- $partassign$: not provided
- $unwanted(a, b) = \emptyset$

[See Figure 1]
Since at the start all branches coming out of the root are of the same depth and all contain a single variable assignment, the next branch to explore is chosen randomly as the left most branch. This choice is marked ①. As the problem is not solved, the control again chooses from among the alternatives, the one with the lowest newly assigned slot number, marked ② in the example. This problem is solved because by inspecting the index vector, we see that this partial solution assigns the same time slot to a course and its corresponding lab; a violation of the first hard constraint. The control transitions the solution entry to *no*. The only other alternative at this level is then chosen and this is marked ③. The search continues in this way until the leaf marked ⑮ is reached. As the problem is solved by this course assignment, the solution entry is transitioned to *yes*, and the search terminates. By inspecting the index vector for this instance, $(c_1, l_{11}, l_{12}, c_2, l_{21}, c_3, l_{31}, l_{32}, l_{33})$, we see that this search produced an individual that satisfies the hard constraints of the problem: $(1, 2, 2, 1, 2, 1, 2, 2, 2)$.

5. Example Or-tree Based Search Control Two

Next we demonstrate the operation of the or-tree based search using $\mathcal{K}_{or-2}(assign', assign'')$ with the same problem parameters as in the last example, where:

Let $assign' = (1, 2, 2, 1, 2, 1, 2, 2, 2)$ and $assign' = (2, 1, 1, 1, 2, 2, 1, 1, 1)$.

[See Figure 2]. Note: the randomness in the last three steps was "guided" to produce a shorter example.

Beginning at the root of the tree, since there are no assignments made so far, and because both parent assignments are valid but different, one of the two is chosen at random. We see that this choice corresponds with ①, were slot 1 was assigned to course 1. Again faced with two valid parent choices, one is chosen at random. In this case slot 2 is chosen, as seen in node ②. Again we are faced with two valid but different parent assignment choices, and again slot 2 is the result of the random choice as marked by ③. For the next choice, both parents agree and slot 1 is chosen in ④. Node ⑤ is the result of both parents agreeing on the time slot. In node ⑥, both parent solutions differ, and 1 is chosen at random. The same situation occurs again, leading to ⑦. In the second last position, both parents disagree and 2 is chosen at random. This produces an invalid solution and as a result the solution position is transitioned to *no*, as seen in ⑧. Random choice between valid parents for the last slot produces a valid individual in node ⑩. Because there are only two time slots in this example and since there are two parents, no choice is ever made that doesn't correspond to one of the parents. As a result, there is no mutation aspect present in this particular example. The result of this search is a perfect crossover of the two parents: $(1, 2, 2, 1, 2, 2, 1, 1, 1)$.

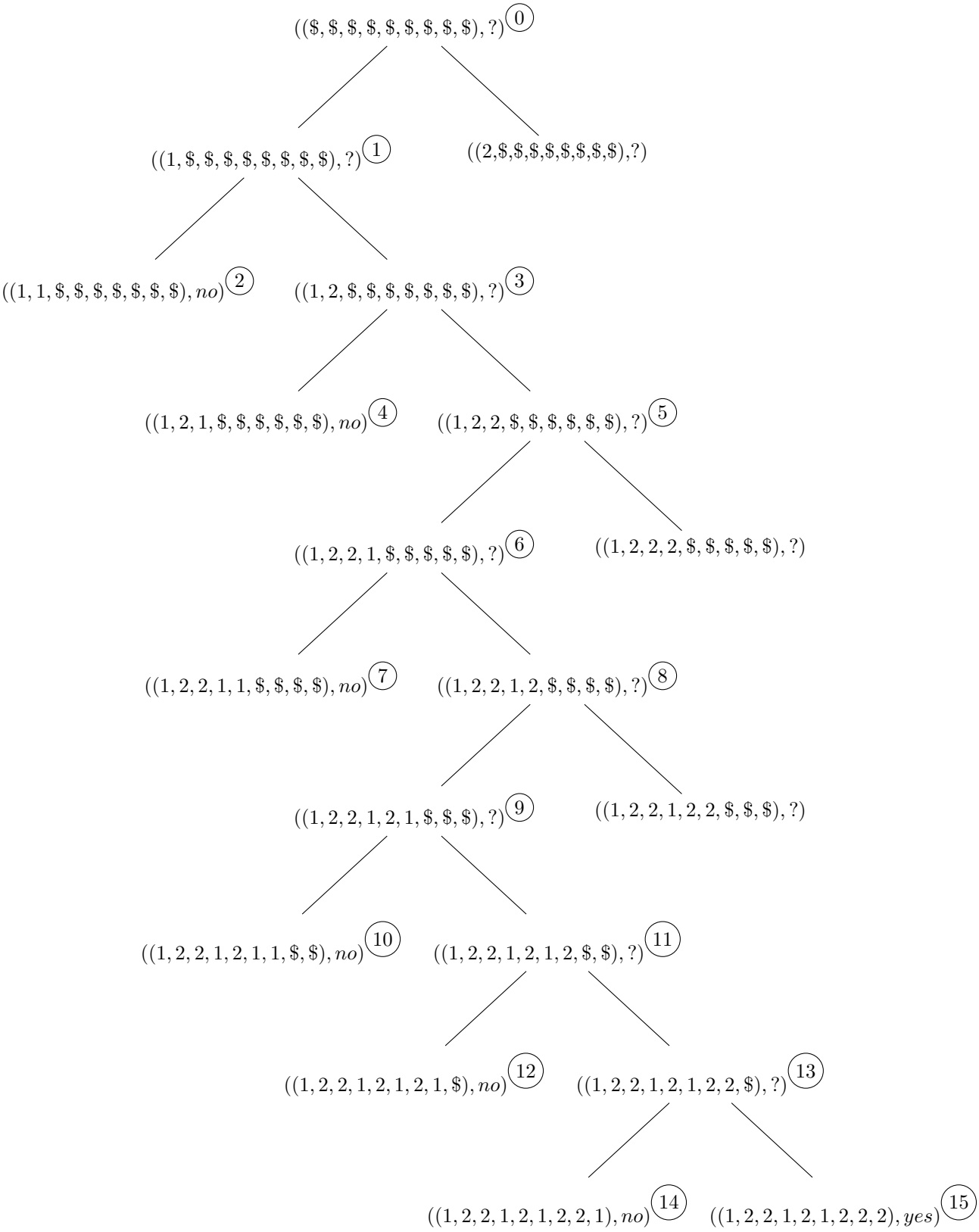

FIGURE 1. Or-tree operation with Control 1

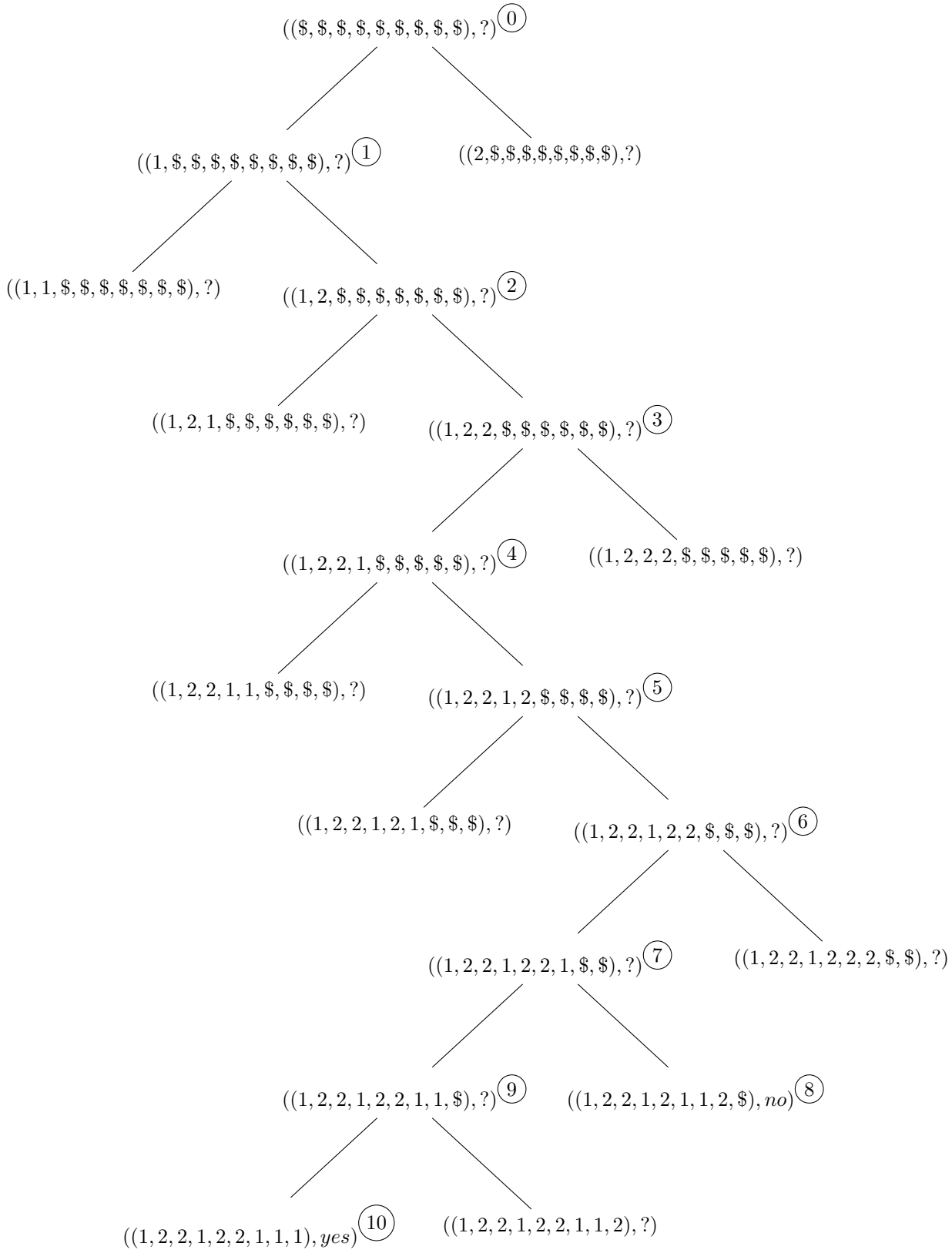

FIGURE 2. Or-tree operation with Control 2

6. Set Based Search Definitions

Set Based Search Model: $\mathcal{A}_{set} = (\mathcal{S}_{set}, \mathcal{T}_{set})$. Where the set of facts and extension rules are defined as follows:

- Let $(c_1, ..., l_{1_{k_1}}, ..., c_m, ..., l_{m_{k_m}})$ be the index vector for elements of $\mathcal{F}$
- Let $k = m + k_1, ..., k_m$ (latter variables as defined in the assignment description).
- $\mathcal{F} = \{(s'_1, ..., s'_k) \mid 1 \leq s'_i \leq n, 1 \leq i \leq k, Constr((s'_1, ..., s'_k)) = True\}$

  That is, $\mathcal{F}$ is a set of all possible assignment vectors that when passed into $Constr$, return $True$. Each assignment vector position is indexed with with the vector: $(c_1, ..., l_{1_{k_1}}, ..., c_m, ..., l_{m_{k_m}})$, so that each number s'_i in a vector in $\mathcal{F}$ represents the time slot number assigned to the course/lab in the i'th position in the index vector.
- $\mathcal{E}xt = \{\{(s'_1, ..., s'_k), (s''_1, ..., s''_k)\} \rightarrow \{(s'_1, ..., s'_k), (s''_1, ..., s''_k), (s'''_1, ..., s'''_k)\} | s'''_i \in \{s'_i, s''_i\},$ or $s'''_i \in [1, n] - \{s'_i, s''_i\}, 1 \leq i \leq k,$ $(determined\ by\ \mathcal{K}_{or-2})$, $Constr((s'''_1, ..., s'''_i, ..., s'''_k)) = True\}$

  That is, $\mathcal{E}xt$ is defined above to contain a single rule, which we call $CrossMut$, that operates as follows: when passed in two individuals serving as parents, an or-tree based search for a child begins, as described in $\mathcal{K}_{or-2}$. The result will be a child that shares as many time slot assignments with its parents as possible.

Set Based Search Control: $\mathcal{K}_{Set}$, is based on only one function:

- f_{select}: This function uses an element of randomness and $Eval$, to select the extension to apply from the set $\mathcal{E}xt$. Here $Eval$ is used to provide a measure of fitness of a valid solution; a lower value indicative of higher fitness.

The search control $\mathcal{K}_{Set}$ simply operates by using f_{select} to produce generation after generation until a predefined goal condition is met. We describe the operation of f_{select} in state $state_j$ (with $state$ used to avoid confusion with slots, s) as follows (note: inspiration drawn from Chapter 2 of the course textbook):

(1) Order all individuals of $state_j$ in descending order of fitness (i.e. those with lower $Eval$ values appear first) to get a vector, fit, of size $|state_j|$.
(2) Let $FIT = \Sigma_{i=1}^{|state_j|} Eval(assign_i)$ represent the total fitness of the population of state $state_j$.
(3) Associate with each individual a part of the interval $[1, FIT]$ as follows: $fit[i]$ gets $Eval(fit[|state_j| - i])$ out of the FIT number of positions in the interval.

(4) Select two random numbers with $RNG(1, FIT)$. The corresponding individuals on the $[1, FIT]$ interval, are those that have been selected.
(5) Pass selected individuals to the or-tree search assistant using $\mathcal{K}_{or-2}$ to find a valid (and unique) offspring to be introduced to the population.
(6) If goal state reached, stop searching.
(7) If operation results in a population of size greater that ℓ, (the maximum allowed at the beginning of a state), then delete the least fit individuals until the population is again equal to ℓ.

The search control and the functions contained therein, were chosen as just described for two primary reasons. First, by using the $CrossMute$ operation we allow for the opportunity of the fitness of the population to increase by ensuring that the fittest individuals have the greatest opportunity to produce offspring. Because a fit individual has qualities that we have defined as desirable (in the definition of $Eval$), it is likely that their offspring will also share some of these qualities in such a way that its fitness can possibly be higher than that of either parent. Second, the additional randomness quality of $CrossMute$ allows us to present a more simple search control, as opposed to having a pure mutation operation. Finally, we use step (6) to keep the population size limited, so as to limit the complexity of executing steps (1) - (4) above.

7. Example operation of Set Based Search

We demonstrate the operation of our set based search control with the following small example:

- $Courses = \{c_1, c_2, c_3\}$
- $Labs = \{l_{11}, l_{12}, l_{21}, l_{31}, l_{32}, l_{33}\}$
- $Slots = \{s_1, s_2, s_3\}$
  - $coursemax(s_1) = coursemax(s_2) = coursemax(s_3) = 4$
  - $labmax(s_1) = labmax(s_2) = labmax(s_3) = 6$
  - $coursemin(s_1) = coursemin(s_2) = coursemin(s_3) = 1$
  - $labmin(s_1) = labmin(s_2) = labmin(s_3) = 1$
  - $pen\_coursemin(s_1) = pen\_coursemin(s_2) = pen\_coursemin(s_3) = 2$
  - $pen\_labmin(s_1) = pen\_labmin(s_2) = pen\_labmin(s_3) = 1$
- $not - compatible(a, b) = \emptyset$
- $partassign$: not provided
- $unwanted(a, b) = \emptyset$

We begin the evolution towards an optimal solution to our problem with three randomly selected individuals generated by the or-tree search using control 1:

$state_0 = \{(1, 2, 2, 1, 2, 2, 1, 1, 1), (1, 2, 2, 1, 2, 1, 2, 2, 2), (3, 2, 1, 1, 2, 3, 1, 1, 2)\}$

(1) Calculate the $Eval$ value of each individual and place in increasing order in a vector, fit:

- $Eval((1, 2, 2, 1, 2, 2, 1, 1, 1)) = 2 + 1 = 3$ (s_3 - no course or lab)
- $Eval((1, 2, 2, 1, 2, 1, 2, 2, 2)) = 1 + 1 + 2 + 2 = 6$ (s_1 - no lab, s_3 - no course or lab, s_2 - no course)

- $Eval((3,2,1,1,2,3,1,1,2)) = 4+4+1 = 5$ (s_2 and s_1 - no course, s_3 - no lab)

Then $fit = ((1,2,2,1,2,2,1,1,1),(3,2,1,1,2,3,1,1,2),(1,2,2,1,2,1,2,2,2))$.

(2) $FIT = 3+6+5 = 14$

(3) $fit[1]$ gets from 1 - 6, $fit[2]$ gets from 7 - 11, and $fit[3]$ gets from 12 - 14 on the interval $[1,14]$

(4) Having obtained the random numbers 2 and 9, individuals $fit[1]$ and $fit[2]$ are selected for the $CrossMut$ operation.

(5) Search control $\mathcal{K}_{or-2}((1,2,2,1,2,2,1,1,1),(3,2,1,1,2,3,1,1,2))$ returns the individual $(3,2,1,1,2,2,3,3,1)$, which is actually *an* optimal solution to this very simplified example. If one of the goal states was defined so that the search is complete when an individual has an $Eval$ value of 0, then the search stops with the following state:

$$state_1 = \{(1,2,2,1,2,2,1,1,1),(1,2,2,1,2,1,2,2,2),\\ (3,2,1,1,2,3,1,1,2),(3,2,1,1,2,2,3,3,1)\}$$

SHANE.SIMS@UCALGARY.CA