CPSC 433 Group Assignment

Park, Sean (30061734),
Gantz, Eric (30031518),
Tadic, Adrian (30077647),
Pistner, Markus (30081575),
Belanger, Mitchel (30075310),
Elrafih, Sammy (30071355)

October 2020

1 And-Tree Based Search Model

We chose to use this search model because it is used by the branch and bound technique, which has properties that make it suitable for use with optimization problems – namely, the guarantee of finding an optimal solution given enough time and its ability to avoid unnecessary branches by bounding.
Since we are using the branch and bound paradigm, we do not need backtracking in this search model.

Prob
An arbitrary element of Prob, pr, is a vector of sets defined as follows:
$pr = (s_1, ..., s_n, StuffToBePlaced)$
such that
$StuffToBePlaced \subseteq Classes \cup Labs$
and
$s_i \subseteq Classes \cup Labs$ for all $1 \leq i \leq n$
and
$s_1 \cup ... \cup s_n \cup StuffToBePlaced = Classes \cup Labs$
and
$s_1 \cap ... \cap s_n \cap StuffToBePlaced = \emptyset$.
The sets $s_1, ...s_n$ each represent a given schedule slot. If a course or lab c_1 is

an element of a given slot set, s_i with $1 \leq i \leq n$,it means c_1 is scheduled for slot s_i.
On the other hand, if a course c_2 is an element of $stuffToBePlaced$, that means it has not been assigned any schedule slot and that pr represents a partial assignment.
-The conditions that $StuffToBePlaced \subseteq Classes \cup Labs$ and $s_i \subseteq Classes \cup Labs$ for all $1 \leq i \leq n$ ensure that the sets $s_1, ...s_n$ and $StuffToBePlaced$ are all sets of course and labs, so this represents an entire schedule to be filled.
-The condition that $s_1 \cup ... \cup s_n \cup StuffToBePlaced = Classes \cup Labs$ ensures that every course and lab is accounted for by pr − that is, if a $c \in Courses \cup Labs$ pr 'knows' what its 'scheduling status' is.
-The condition that $s_1 \cap ... \cap s_n \cap StuffToBePlaced = \emptyset$ ensures that a course or lab cannot be in more than one place with respect to pr. For example, if we have a course c_3, then that course should not be able to be an element of both s_i for $1 \leq i \leq n$ and $stuffToBePlaced$ because that would mean it is both scheduled and unscheduled.

When is a Node Solved?

Note that $0 \in \mathbb{N}$ here.

Before we define what it means for a node to be solved, we need to define the following:

We will use the phrase 'partial assignment' to refer to a problem where:
$pr = (s_1, ..., s_n, StuffToBePlaced) \in Prob$
such that $stuffToBePlaced \neq \emptyset$.
We will use the phrases 'complete assignment' and 'solution' to refer to a problem where:
$pr' = (s'_1, ..., s'_n, StuffToBePlaced') \in Prob$
such that $stuffToBePlaced' = \emptyset$.

We will define $Eval^* : Prob \to \mathbb{N}$ to be a function which measures how well a partial assignment fulfills soft constraints.

We will define $Constr^* : Prob \to \{true, false\}$ to be a function which determines if a partial assignment violates any hard constraints.

Since we are using the branch and bound technique, we need a function $f_{bound} : (pr, ?) \to \mathbb{N}$ that estimates how good a solution using all the scheduling decisions represented in a given node can get.
We will use $f_{bound}(pr, ?) = eval^*(pr)$. This provides a best-case estimate for the $Eval$ value of an assignment in a node descended from (pr, ?) because the assignment of other courses can only increase the value of $Eval * (pr)$.

Another requirement for branch and bound is a value which keeps track of the best know solution. We will use $Bestol \in \{-1\} \cup \mathbb{N}$ for this.
Note that $Bestol = -1$ if and only if no complete assignment has been found.

Now we are ready to define if a node is solved or not.
A node $pr = (s_1, ..., s_n, stuffToBePlaced)$ is solved if and only if either
$stuffToBePlaced = \emptyset$ and $Constr(pr) = true$
or
$Constr^*(pr) = false$
or
$f_{bound}(pr) > Bestsol$.

Div

Let $pr = (s_{10}, ..., s_{n0}, stuffToBePlaced_0) \in Prob$
and
let $b_j = (s_{1j}, ..., s_{nj}, stuffToBePlaced_j) \in Prob.$
Then

The division relation $Div(pr, b_1, ..., b_m)$ holds if and only if
$m = n$
excluded form div and
there exists some $c \in Classes \cup Labs$ such that $c \in stuffToBePlaced_0$
where for $1 \leq j \leq m$ there exists an s_{jj} such that $c \in s_{jj}$ and for all $l \neq j$ such that $1 \leq l \leq m$ $c \notin s_{jl}$
Such that for all $1 \leq p \leq m$, $stuffToBePlaced_p = stuffToBePlaced_0 \setminus \{c\}$.

-Suppose we have a node in our tree problem *pr* and $Div(pr, b_1, ..., b_m)$.
The idea of this Div is to make the problems $b_1, ..., b_m$ (henceforth to be referred to in this explanation as the b-problems) the partial assignments obtained from assigning a course c_u that is unassigned in the partial assignment described by *pr* to a single slot for every b-problem in such a way that each one has c_u in a different slot and all slots are 'covered' in this way.
-The condition that $m = n$ ensures that there are enough b-problems to accommodate all possible assignments of c_u and no more.
-The second condition ensures that our c_u is, in fact, unassigned.
-The third condition ensures that the the n^{th} b-problem (where $0<n \in \mathbb{N}$) has c_u scheduled the n^{th} slot.
-The last condition ensures that c_u is not registered as being unplaced anymore.
Note that the b-problems are by definition elements of *Prob*.

2 And-Tree Search Control

Leaf Selection
Our search control selects a leaf to be processed as follows:

If $Bestsol = -1$ (meaning no solution has been found yet), our control uses a depth-first search to look for a solution:
The control selects a leaf with ? as its sol value for processing.
If there are several of these, it selects the leaf that is deepest in the tree for processing.
If there are several of these, it selects the leaf containing the problem with the best $Eval^*$ value for processing.
If there are several of these, it selects the leftmost leaf among them for processing.

Otherwise, if $Bestsol \geq 0$ (meaning a solution has been found), our control can use a best-first search to select a leaf to be processed:
The control selects a leaf with ? as its sol value for processing.
If there are several of these, the control selects for processing the leaf with the lowest f_bound value.
If there are several of these, it selects the deepest node for processing.
If there are several of these it selects the leftmost leaf for processing.

Transition Selection
In order to define our transition selection function, we need to define the following:

Let $m = (pr, ?)$ with $pr = (s_1, ..., s_n, stuffToBePlaced) \in Prob$
and
$c \in stuffToBePlaced$.

The transition on m using c is the transition that results in each child of m, $(pr_j, ?)$ having a problem $pr_j = (s_{1j}, ..., s_{nj}, stuffToBePlaced_j) \in Prob$ such that $stuffToBePlaced_j = stuffToBePlaced \setminus \{c\}$.

Our search control decides what transition to apply to given a node $m = ((s_1, ..., s_n, stuffToBePlaced), ?)$ in the following manner:
It randomly selects a $c \in stuffToBePlaced$ and applies to the tree the transition on m using c.

3 And-Tree Search Process Example

Let $Classes \cup Labs = \{a, b, c\}$.
Let the be two slots.

Let our start state $s_0 = ((\{\}, \{\}, \{a, b, c\}), ?)$.
so we have

$(((\{\}, \{\}, \{a, b, c\}), ?)), Bestsol = -1$

Since there is only one node at the beginning with sol value ?, our control selects it.
Now, we say that our control randomly decides to apply The transition on the root using a, so we get

$(((\{\}, \{\}, \{a, b, c\}), ?), ((\{a\}, \{\}, \{b, c\}), ?), ((\{\}, \{a\}, \{b, c\}), ?), Bestsol = -1)$
Now, suppose $Eval*((\{a\}, \{\}, \{b, c\})) = 4)$ and $Eval*((\{\}, \{a\}, \{b, c\})) = 3)$.
Then the control selects $(\{\}, \{a\}, \{b, c\})$ as the leaf to be processed.
Suppose that the control randomly decides to apply The transition on this leaf using b, so we get

$(((\{\}, \{\}, \{a, b, c\}), ?), ((\{a\}, \{\}, \{b, c\}), ?), ((\{\}, \{a\}, \{b, c\}), ?, ((\{b\}, \{a\}, \{c\}), ?)((\{\}, \{a, b\}, \{c\}), ?)), Bestsol = -1)$

Now, since $(\{\}, \{a, b\}, \{c\})$ and $(\{b\}, \{a\}, \{c\})$ are the two deepest leafs, the search control must decide between them.
Let $Eval*((\{\}, \{a, b\}, \{c\})) = 5)$ and $Eval*((\{b\}, \{a\}, \{c\})) = 6)$.
Then the control will select the one with the best eval for processing, $(\{\}, \{a, b\}, \{c\})$.
Since there is only one possible transition to apply to this, we get

$(((\{\}, \{\}, \{a, b, c\}), ?), ((\{a\}, \{\}, \{b, c\}), ?),$
$((\{\}, \{a\}, \{b, c\}), ?, ((\{b\}, \{a\}, \{c\}), ?, ((\{cb\}, \{a\}, \{\}), ?,), ((\{b\}, \{ca\}, \{\}), ?,))((\{\}, \{a, b\}, \{c\}), ?)), Bestsol = -1)$

and from here it can begin the 'branch' phase as specified in the control.

4 Or-Tree Satisfying the Hard Constraints for Set Based:

Prob The Prob for this search model is the same as the *Prob* used by the and-tree.

When is a Node Solved?

- **Solvable:** A node $(pr, ?)$ is solvable if and only if the set $StuffToBePlaced$ is empty, and $Constr$ holds; in which case this $(pr, ?)$ would receive a minimal score from f_{leaf} as it would have greatest depth (and thereby a possible solution given $Const$ holds). We suppose that $Constr*$ is a function evaluating all hard constraints for a given partial assignment (And Constr is defined as a function evaluating full assignments). Thus $Erw_{\vee}((pr, ?), (pr, yes))$ iff the last set ($StuffToBePlaced$) of this pr is empty, and Const holds for this now full assignment.

- **Unsolvable:** A node $(pr, ?)$ is unsolvable if and only if for some transition $T_{\vee}$ that was just applied; the partial assignment reached invalidates $Constr*$. That is, the current instance (pr, sol) being checked has a partial assignment which infringes on some $Constr*$. $Constr*$ will be a series of predicates considering the truth or falseness of each hard constraint for a partial assignment so its able to check if partial assignments are "correct" in terms of these constraints. So $Erw_{\vee}((pr, ?), (pr, no))$ if $Constr*$ doesn't hold for the partial assignment described by pr.

- **Expandable / Altern:** A node $(pr, ?)$ is expandable to $(pr, ?, (pr_1, ?), ..., (pr_n, ?))$ if and only if $Altern$ recommends this expansion.

  The relation Altern used here is the same as the relation Div used for the and-tree.

The Search Controls:
f_{leaf}**:** f_{leaf} gives priority to solvable / unsolvable expansions (transitions). It gives 0 to solvable cases and 0.5 to unsolvable cases such that f_{trans} closes nodes to no or yes before applying expansions involving Altern.

- $f_{leaf} : (pr, ?) \rightarrow \mathbb{N}$ for which the outputted $\mathbb{N}$ corresponds to the length or cardinality of the set $StuffToBePlaced$ which is the last element of any given pr. The value of f_{leaf} is exactly this value in case a given leaf is otherwise not yet solvable or unsolvable.

f_{trans}**:** First we note that the solvable and unsolvable expansions receive the minimal scores 0, 0.5 respectively so that they have priority. Then f_{trans} randomly selects some class / lab a if a is still to be placed in $StuffToBePlaced$. f_{trans} will further favor the Altern of this class a based on the depth of a node for which f_{trans} is picking a transition.

5 Or-Tree for Genetic Operators:

General Set-Based Search Model:
We need to identify how to describe the problem genetically in terms of the outputs of the Or-Tree producing complete individuals for use. Thus F is a set of individuals describing complete and valid schedules whereas elements of F^* represents incomplete individuals (schedules). That is, we need a way to compare the genes of two parents, where each step we look at one component of their genes and pick one from either parent randomly. So a **subproblem** is an incomplete schedule produced from two valid parents, where the child schedule may be invalid. To define a partially assigned child fact, we define the set F^*.
Contrary to the or-tree for initialization we consider the "other way around" in which rather than considering for each slot as a set what classes / labs it contains or should contain; we consider for each class / lab in $Classes \cup Labs$ what element of $Slots$ it should be paired with. We suppose: $q = m + \Sigma_{i=1}^{m} k_i$ is the number of courses and labs. Further,
$F^* = \{\{(a_1, sl_{j_1}), ..., (a_q, sl_{j_q})\} | \forall r, 1 \leq r \leq q$ where $a_r \in Courses \cup Labs$ so that:

1. $\forall t, 1 \leq t \leq q$, where $a_t \in Courses \cup Labs$ and $r \neq t, a_r \neq a_t$ and,
2. $\exists j_r, 1 \leq j_r \leq n$ for $sl_{j_r} \in Slots$, where $assign(a_r) = sl_{j_r}\}$.

We then define a useful function for describing conditions with any partial assignment in F^*. $Assigned : F^* \rightarrow 2^{Courses \cup Labs}$.
For some $Z \in F^*$, there exists a partial assignment described by Z through $assign^* : Courses \cup Labs \rightarrow Slots \cup \{!\}$, then;
$Assigned(Z) = \{a | a \in Courses \cup Labs, assign^*(a) \neq !\}$.

Prob:
$Prob = \{(Unassigned, X, Y, Z) | Unassigned \subseteq Courses \cup Labs$,
X, Y, $\in F, Z \in F^*, Unassigned \cup Assigned(Z) = Courses \cup Labs\}$.

When is an Instance of Prob Solvable, Unsolvable, or can be Brought Nearer to Solution?

- **Solvable:** $(Unassigned, X, Y, Z) \in Prob$ is solved $\Leftrightarrow Unassigned = \emptyset$, and furthermore suppose $Z = \{(a_1, sl_{j_1}), ..., (a_q, sl_{j_q})\}$; then, by the definition of $Prob$,
$Assigned(Z) = (Courses \cup Labs) \backslash Unassigned = Courses \cup Labs$, such that
for all $1 \leq r \leq q$ where $a_r \in Courses \cup Labs$ with
$1 \leq j_r \leq n, assign(a_r) = sl_{j_r} \neq !$, and $sl_{j_r} \in Slots$; thus there exists a full assignment that can be described by Z through
$assign : Courses \cup Labs \rightarrow Slots$:

for all $1 \leq r \leq q$ where $a_r \in Courses \cup Labs$ with
$1 \leq j_r \leq n, assign(a_r) = assign^*(a_r) = sl_{j_r} \in Slots$, and
$Constr(assign)$.

- **Unsolvable:** $(Unassigned, X, Y, Z) \in Prob$ is unsolvable $\leftrightarrow$ for a partial assignment described by Z through
$assign^* : Courses \cup Labs \rightarrow Slots \cup !, \neg Constr^*(assign^*)$.

- **When a Problem Can be Brought Nearer to Solution:**
We can move closer to a solution by assigning a course or lab from $Unassigned$ a slot. This course or lab has the choice of being assigned any of the n slots.
$Altern((Unassigned, X, Y, Z), pr_1, ..., pr_n) \leftrightarrow$
Suppose $Z = \{(a_1, sl_{j_1}), ..., (a_q, sl_{j_q})\}$; for some
$1 \leq r \leq q, a_r \in Unassigned$; then, for all $1 \leq j \leq n$ with
$sl_j \in Slots, pr_j = (Unassigned \backslash \{a_r\}, X, Y, Z')$ with
$Z' = \{(a_1, sl'_{j_1}), ..., (a_q, sl'_{j_q}),$ such that:
$assign^*(a_r) = sl'_{j_r}$, and
for all $1 \leq t \leq q$ where $t \neq j$ with $1 \leq j_t \leq n, sl'_{j_t} = sl_{j_t}$.

6 Set-Based definitions:

$F =$
$\{\{(sl_1, \{a_{1,1}, \ldots, a_{1,q1}\}), (sl_2, \{a_{2,1}, \ldots, a_{2,q2}\}), \ldots, (sl_n, \{a_{n,1}, \ldots, a_{n,qn}\})\}|$
for all A$\in$ CoursesULabs, there exists a 1$\leq$ j$\leq$ n where sl$_j$$\in$ Slots and
assign(A)=sl$_j$, $such that for some 1\leq$ r$\leq$ q$_j$, $A = a_{j,r}; and$
$forall 1\leq$j$\leq$n, sl$_j$$\in$Slots such that for all 1$\leq$r$\leq$q$_j$, $a_{j,r}$$\in$CoursesULabs where
assign(a$_{j,r}$) $= sl_j$,
$forall 1\leq$i$\leq$n where sl$_i$$\in$Slots and j$\neq$i,sl$_i$$\neq$sl$_j$, and
$forall 1\leq$t$\leq$q$_i$ $where a_{i,}$$\in$CoursesULabs and r$\neq$t,a$_{j,r}$$\neq$a$_{i,t}$; and
$Constr(assign)\}$.
For each course or lab a, there exists a slot sj for which a is assigned to, which would be represented in the set of courses/labs of length qj paired with slj. Also, for each slot sj, every course or lab aj,r in sj's paired set is assigned to sj; sj must be unique compared to all other slots si, and for every course or lab in si's paired set, the courses and labs ai,t must be unique compared to aj,r. Lastly, Constr(assign) must hold.
In summary, a fact is a set of pairs, of a slot and the courses/labs assigned to it. To make the fact represent a valid schedule: every slot must be uniquely represented within one pair; all courses and labs must be assigned a slot, and every course or lab must be uniquely assigned to one slot; lastly, the full assignment function represented by this fact must have all hard constraints hold. Then, the set of facts F is the set of all valid schedules.

Genetic operators as or-tree-based search, which guarantees validity of result and allows specifying its control in detail.

Since the breeding or-tree-based search will also include the possibility of mutating an assigned course/lab if the parents' genes lead to broken hard constraints, the breeding extension rules are defined as generally as possible, knowing that the produced child Z is guaranteed to fulfill hard constraints by definition of F.

$$CrossAndMutateRules = \{\{X, Y\} \to \{X, Y, Z\} \mid X, Y, Z \in F\}$$
$$suppose\ X = \left\{\left(a_1^X, sl_{j_1}^X\right), \ldots, \left(a_q^X, sl_{j_q}^X\right)\right\}, Y = \left\{\left(a_1^Y, sl_{j_1}^Y\right), \ldots, \left(a_q^Y, sl_{j_q}^Y\right)\right\}, Z = \left\{\left(a_1^Z, sl_{j_1}^Z\right), \ldots, \left(a_q^Z, sl_{j_q}^Z\right)\right\}; then,$$
$$for\ all\ 1 \leq r \leq q\ where\ a_q^Z \in Courses \cup Labs,\ for\ 1 \leq j_q \leq n, sl_{j_q}^Z \in Slots\ and\ \left(unnecessary,\ only\ need\ these\ breeding\ extension\ rules\ to\ be\ general\right)$$

We define our population as having a maximum size, where once that limit is broken, some specified amount of individuals are picked as elites and the rest culled. Then, another specified amount of newly generated individuals are introduced into the population as replacements.

We define a maximum population size pop_max, the number of inidividuals picked at once $elite\_keep\_size$, and the number of new individuals generated in a cull as $new\_individuals\_size$; combined, $elite\_keep\_size + new\_individuals\_size < pop\_max$. Then,

$$CullAndGenRules = \{Population \to Elite \cup NewIndividuals \mid Population, NewIndividuals \subseteq F, Elite \subseteq Population,\ where$$
$$|Population| = pop\_max,\ |Elite| = elite\_keep\_size, |NewIndividuals| = new\_individuals\_size, so\ that$$
$$elite\_keep\_size + new\_individuals\_size < pop\_max,\ and$$
$$for\ all\ 1 \leq u \leq pop\_max\ where\ sch_u \in Elite\ defines\ an\ assignment\ assign_u : Courses \cup Labs \to Slots,$$
$$for\ all\ 1 \leq v \leq pop\_max\ where\ sch_v \in Population \setminus Elite\ defines\ an\ assignment\ assign_v : Courses \cup Labs \to Slots,\ Eval\left(assign_u\right) \leq Eval\left(assign_v\right)\}.$$

$$Ext = CrossAndMutateRules \cup CullAndGenRules$$

3. Set-based Search Model

$$A_{set} = (S_{set}, T_{set})\ where$$
$$F\ is\ as\ defined\ above,$$
$$Ext\ is\ as\ defined\ above,$$
$$S_{set} \subseteq 2^F,$$
$$T_{set} = \{(s, s') \mid there\ exists\ A \to B \in Ext\ where\ A \subseteq s\ and\ s' = (s - A) \cup B\}$$

General Set-based Search Process

1. Identify possible functions that measure a fact

Since this is a genetic algorithm, we use a fitness function.

$$f_{fit} : F \to \mathbb{N}\ where$$
$$f_{fit}(X) = Eval\left(assign_X\right).$$

2. Decide if we can rank extension rules based on children.

We could, though I don't think we should. We should trust that good parents will most likely give good children.

3. ...
4. ...
5. If you want to rely on random decisions (or include them), set f_{Wert} constant.

We will rank based on whether the rule is a cross and mutate rule or a cull and generate rule; deciding between rules within these groups includes randomness, thus it is not a factor in f_{Wert}.

$$f_{Wert} : 2^F \times 2^F \times Env \to \mathbb{N}\ where$$
$$f_{Wert}(A, B, e) = 2\ if\ A \to B \in CrossAndMutateRules,$$
$$f_{Wert}(A, B, e) = 1\ if\ A \to B \in CullAndGenRules.$$

6. Design f_{select}

$$f_{select} : 2^{2^F \times 2^F} \times Env \to 2^F \times 2^F.$$

When culling rules apply, which are prioritized over breeding rules, the selection of these culling rules depend only on the population and the elite within, where the selection is random based on the randomly generated new individuals. When culling does not apply, the breeding rule is chosen based on some fitness and some random factor.