TA Assignment Problem

University of Calgary - CPSC 433 - Fall 2013

Carson Tunna, Tyler Smith, Adam Thompson

October 22, 2013

Contents

1 Preamble

For this problem we decided on OR-Tree based search and Set based search. Our reasoning was as follows. We immediately had a solution using an OR-Tree based model that covered every combination of TA assignments to time slots. We figured using a greedy heuristic for our *altern* function we could likely come up with an optimal or slightly sub optimal solution with relative ease. Our second choice as set based search was very natural in that set based search allows for so much flexibility. On top of this, the F_{werth} function of set based search parallels with the soft(and hard) constraints detailed in the problem description. We were uninspired when we considered the option of using an AND-Tree based search model simply because it's difficult to apply a *div* function to the structure of this problem. Finally the main reason for selecting these search paradigms was professor Kremer confirming our instincts and telling us we made the right choice.

2 OR-Tree Based Search

Let L be the set of all tutorials or labs and T be set of all TA's. Then the problem is defined as follows:

1. $\forall x \in L$, $\exists y \in T$ such that instructs(y, l).
2. All hard constraints are satisfied[1]

2.1 Explanation of Model

We're looking for an assignment of TA's to labs such that no hard constraints are violated and as few soft constraints are violated.
The algorithm works as follows

1. Set the root node to the empty set
2. Pick a lab
3. From the root, draw a branch for every TA that can teach this TA without violating any hard constraint.
4. Sort the branches from the root node by goodness value in ascending order[2]
5. Set the left hand node to be your root node. Remove the last lab from your set of working labs. Recurse.

[1]See appendix

[2]Goodness value is calculated by the number of soft constraints are currently being violated. The largest goodness value possible is 0

6. When there are no more TA's available the left hand arm of the tree is the solution
7. While there is still time available for processing - Go back to the root of the tree and this time choose 2nd child of the $root = \emptyset$

2.2 Mathematical Description of the Model

Let L be the set of labs

Let L' be the set of labs without a TA assignment yet

Let T be the set of all TA's

Let $I : T \times L$ be a set of ordered pairs representing an assignment of a TA to a lab.

2.2.1 Defining the problem instance

We defined our problem instance as follows

$$pr = \langle (l_1, t_1), (l_2, t_2), ..., (l_n, t_n) \rangle$$

Where $(l_i, t_i) \in I$ and $pr : Prob = sequence\ I$

States can be described by

$$S = OTree(Prob, \{yes, no, ?\}, b_1 ... b_n)$$

2.2.2 Evaluating Solutions

We define the function $f_{leaf} : S \times Env \to \mathbb{N}$

$$f(s) = \sum_{t \in T} Cv(\langle (l_1, t_1) ... (l_n, t_n) \rangle)$$

Where $Cv : Prob \to \mathbb{N}$ is a function that takes in a sequence of assignments and returns the sum of constraints being violated.

2.2.3 When is a branch solvable or unsolvable

A pr is unsolvable when

$$Erw_{v,wt}((pr, ?), (pr, no)) \iff (l, t) \in I \wedge violates - hard - constraint((l, t))$$

I.e. for every assignment of a TA to a specific lab, a hard constraint is violated.

A pr is solved when

$$Erw_{v,wt}((pr,?),(pr,yes)) \iff \neg Erw_{v,wt}((pr,?),(pr,no)) \wedge |L'| = \emptyset$$

I.e. the problem is not unsolvable and all the labs have been assigned a TA.

2.2.4 Branching Definition

Define $Altern$ in the following way

$$Altern(s,s') \text{ such that}$$
$$(|s|+1=|s'|) \wedge getLab(last(s')) \in L' \wedge getTA(last(s')) \in T$$

Where $getLab() : I \times L \to L$ returns the lab from a tuple in I. Similarly for $getTA()$

I.e. the sequence s' is longer than s and we have assigned a TA to a lab which hasn't had any assignment prior.

$Erw((pr,?),(pr,?,(pr_1,?),(pr_2,?),...,(pr_n,?)))$ if pr_i is generated out of assigning a valid TA to pr_i such that this assignment doesn't violate any hard constants

Children are chosen using f_{trans} as follows

$$Erw((pr,?,b_1,...b_n)(pr',?,b'_1,...b'_n)) \text{ such that}$$
$$1 \le i \le n, pr' = min(f_{leaf}(append(pr,b_i)))$$

2.3 Demonstration of a small search instance

3 Set-Based Search

3.1 Explanation of Model

The Set based search algorithm works as follows.

1. Start with an initial state S_0 of approximately fifty facts. This can be generate via random walks of our OR-Tree based tree
2. Take the best fact in S_0 and set it to max
3. Modify each fact by swapping TA assignments such that it enhances or doesn't change the f_{wert} value of functions
4. Update the value of max
5. While there is still time left, go back to step 3

3.2 Mathematical Description of the Model

3.2.1 Model

Let L be the set of labs

Let T be the set of all TA's

Let $I : T \times L$ be a set of ordered pairs rerpresenting an assignment of a TA to a lab.

Let a fact $f \in I$

Let F be a set of facts

Let a state $S \subseteq 2^F$.

Let $A = (S, T)$.

Let $T : S \times S$ and $T = \{(s, s')|\exists A \rightarrow B \in Ext \wedge A \subseteq s \wedge s' = (s - A) \cup B\}$

A fact f does not violate hard-constraints given in Appendix **??**. That is,

1. No TA has more than *MAX_LABS* labs: for any fact f, $\forall t_i |(t_i, l) \in f| \leq MAX\_LABS$.
2. If a TA has a lab, that TA has at least *MIN_LABS* labs: for any fact f, $\forall t_i |(t_i, l) \in f| = 0 \vee |(t_i, l) \in f| \geq MIN\_LABS$
3. No lab has more than one TA and all labs have a TA: for any fact f, $\forall l_i |(t, l_i) \in f| = 1$
4. No TA has a time conflict: for any fact f, $\forall (t_i, l_i) \in f, time\,(l_i) \notin \{time\,(c_k) \wedge c_k \in courses\,(t_i)\} \wedge time(l_i) \notin \{time(l_j) \wedge (t_i, l_j) \in f\}$

Define $Ext : \{A \rightarrow B|A, B \subseteq F\}$ where $B = A \cup C$ where C is generated by specifying an allowable time and calling *Generate* then *Combine* until that time is exceeded or until $|C| = |A|$ so that $|B| = 2|A|$. The operations used are defined as:

1. *Generate* - Do a random walk through the defined in **??**. The random walk does not compare leafs, it only tries paths at random until a solution is found. If no solution is found within the allowed time, this operation fails.
2. *Combine* - First, map each element in a fact f from a 2-tuple to a 3-tuple $(t, l) \rightarrow (t, l, b = time(l))$. Then, for each b such that $\exists (t', l', b) \in f$, match each instance of t' and l' once at random. This does not change the times that any TA teaches, it only changes which labs a TA is teaching.

$$K(s,e) = (s\text{-}A) \cup B \text{ where } A \to B \in Ext \wedge A \subseteq s \wedge \forall A' \to B' \in Ext \mid A' \subseteq s \bullet fWert(A,B,e) \leq fWert(A',B',e) \wedge A \to B = fselect(\{A' \to B' \mid \forall A'' \to B'' \in Ext \mid A'' \subseteq s \bullet fWert(A',B',e) \leq fWert(A'',B'',e)\},e)$$

Figure 1: The control K from the course rubric.

If the result violates any hard constraints, this operation fails. For the implementation, we will consider lazy evaluation of hard constraints.

3.2.2 Process

We define our process $P : (A, Env, K)$ for the set based search. The model A has already been defined. It is assumed that the environment Env is unchanging so $K : S \times Env \to S$ is just $K : S \to S$. The control K for a set based search is defined in Figure **??**.

f_{wert} is defined as $-\sum_i penalty_i(f)$ where $penalty_i$ is defined in Figure **??** as a function of a fact which is either the penalty value from the table or zero if the penalty does not apply.

f_{select} is defined as a tournament. The number of facts is "culled" down to a specified number N. This is done by repeating the following operation $|A| - N$ times. At random, two facts in $f_1, f_2 \in F$ are selected. A random number $0 < r < 1$ is generated. If $r < \frac{f_{1wert}}{f_{1wert} + f_{2wert}}$, f_1 is removed from A, otherwise f_2 is removed from A. For the implementation,

3.3 Demonstration of a small search instance

Appendices

A Hard Constraints

1. every TA is assigned at most MAX_LABS labs
   FORALL ta:TA . lab-count(ta) $\leq$ MAX_LABS)

2. every TA is assigned at least MIN_LABS labs (if the TA *has* a lab assignment)
   FORALL ta:TA . lab-count(ta) $\neq$ 0 then lab-count(ta) $\geq$ MIN_LABS)

3. no lab has more than one TA assigned to it
   FORALL course:Course, lab:Lab | has-lab(course,?,lab) . $\neg$ EXISTS ta1,ta2:TA | ta1$\neq$ ta2 . instructs(ta1,lab) $\wedge$ instructs(ta2,lab)

4. every lab has a TA assigned to it
   FORALL course:Course, lab:Lab | has-lab(course,?,lab). EXITS ta:TA . instructs(ta,course,lab)

5. no TA is assigned simultaneous labs
   FORALL ta:TA, c1,c2:Course, b1,b2:Lab | (c1=c2 =¿ b1 $\neq$ b2) $\wedge$ instructs(ta,c1,b1) $\wedge$ instructs(ta,c2,b2) . $\neg$ EXISTS t1,t2 | at(c1,b1,t1) $\wedge$ at(c2,b2,t2) . conflicts(t1,t2)

6. no TA is assigned a lab that conflicts with his/her own courses
   FORALL ta:TA, course:Course, lab:Lab | instructs(ta,course,lab) .
   (($\neg$ EXISTS c:Course, lec:Lecture | taking(ta,c,lec) . EXISTS t1,t2 | at(course,lab,t1) $\wedge$ at(c,lec,t2)) . conflicts(t1,t2)) $\wedge$
   (($\neg$ EXISTS c:Course, b:Lab | taking(ta,c,b) . EXISTS t1,t2 | at(course,lab,t1) $\wedge$ at(c,b,t2)) . conflicts(t1,t2)))

   where:
   lab-count(TA) is a function that returns the number of labs a TA instructs.

B Soft Constraints

Soft Constraints

	Constraint	Formal Expression	Penalty/ Violation
0	Each TA should be funded (that is, they should teach at least one course)	∀ta:TA • instructs(ta,?,?)	50
1	TAs should get their first choice course	∀ta:TA, c:Course • prefers1(ta,c) → instructs(ta,c,?)	5
2	TAs should get their first or second choice course		10
3	TAs should get their first or second or third choice course		10
4	TAs should have all their labs in the same course	∀ta:TA, c:Course | instructs(ta,c,?) • ~∃c2:Course | c2≠c • instructs(ta,c2,?)	20
5	TAs should have all their labs in no more than 2 courses	∀ta:TA, c,c2:Course | c2≠c /\ instructs(ta,c,?) /\ instructs(ta,c2,?) • ~∃c3:Course | c3≠c /\ c3≠c2 • instructs(ta,c3,?)	35
6	TAs should not teach a lab for a course for which they don't know the subject matter	∀ta:TA, c:Course • instructs(ta,c,?) → knows(ta,c)	30
7	TAs should not teach two labs of distinct courses at the senior level		10
8	TAs should not teach more than one more lab than the TA that teaches the least number of labs.		25
9	TAs should all teach the same number of labs.		5
10	If the instructor requested particular TAs for his/her course, each of the lecture the instructor is teaching for that course should be taught by one of the requested TAs	∀i:Instructor, c:Course • (∃ ta:TA • prefers(i,c,ta)) → ∀ lec:Lecture, lab:Lab | instructs(i,c,lec) /\ has-lab(c,lec,lab) • instructs(ta,c,lab)	10

Figure 2: The table of soft constraints with penalties given in the course assignment