TA Assignment Problem

University of Cagary - CPSC 433 - Fall 2013

Carson Tunna, Tyler Smith, Adam Thompson

October 2013

Contents

1 Preamble **3**

2 OR-Tree Based Search **3**
2.1 Explaination of Model 3
2.2 Mathematical Description of the Model 4
2.2.1 Defining the problem instance 4
2.2.2 Evaluating Solutions 4
2.2.3 When is a branch solvable or unsolvable 5
2.2.4 Branching Definition 5
2.3 Demonstration of a small search instance 5

3 Set-Based Search **5**
3.1 Explanation of Model 5
3.2 Mathematical Description of the Model 5
3.3 Demonstration of a small search instance 6

Appendices **7**

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 Explaination of Model

In the input we're given the following a set of time slots[2]. These time slots consits of a set of days MWF, MW, TR and a time in 24 hour format. Additioally were given a set of TA's that have their own schedule that must be adhered to. We're looking for the existence of as assignment(and if it exists, the optimal solution) such that all time slots are filled with available TA's. Enter OR-Tree based search.

The alforithm works as follows

1. Start with the empty set
2. Pick a TA with the greatest number of constraints to follow.[3]
3. For each time slot that the choosen TA is available to teach - add a branch to the root node. At the end of each branch are the nodes with a tuple that explains:

[1] See appendix
[2] Time slots will be denoted $TS = (D, T)$ where $D \in \{MWF, MW, TR\}$
[3] This helps us not get forced into a corner further down the tree

(a) the goodness value of this pair

(b) the time slot

(c) the TA

4. Sort the branches from the root node by goodness value in ascending order[4]

5. Set the left hand node to be your root node and recurse

6. When there are no more TA's available the left hand arm of the is a 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 T be the set of all TA's

Let $I : L \times T^n$ where I is a tuple representing an assignment of labs to at least one TA

2.2.1 Defining the problem instance

We defined our problem instance as follows

$$pr = (M, T, J)$$

Where

$$\begin{array}{rcl} M & : & \text{The list of labs not yet assigned a TA} \\ T & : & \text{The set of all TA's} \\ I & : & L \times T^N \text{ an assignment of Labs to TAs} \end{array}$$

2.2.2 Evaluating Solutions

$$f(pr) = \sum_{t \in T} Cv(t)$$

Where $Cv : T \to \mathbb{N}$ is a function that takes in a TA and returns the sum of soft constraints that TA's current assignment(s)are violating.

[4]The largest goodness value possible is 0

2.2.3 When is a branch solvable or unsolvable

A pr is unsolvable when

$$Erw_{v,wt}((pr,?),(pr,no)) \iff \forall t \in T, \exists l \in L \text{ such that } (l,t) \in M \wedge violates - hard - constraint((t,l))$$

I.e. for every assignment of a TA to a specific lab there exists no case where it doesn't violate a hard constraint. A pr is solved when

$$Erw_{v,wt}((pr,?),(pr,yes)) \iff \neg Erw_{v,wt}((pr,?),(pr,no)) \wedge \forall x \in M, length(x) > 1.$$

I.e. the problem is not unsolvable and lab has at least one assigned TA. A lab has at least one assigned TA when the length of the tuple is greater than one.

2.2.4 Branching Definition

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_{werth} 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

Let a fact F be a solution, albeit possibly very bad, to our problem - pr.

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

Let $A = (S, T)$ where naturally T is a relation defined $T : S \times S$. We will elaborate further on T below.

Define a function $Ext : A \to B$ where $A, B \subseteq F$. This function will mutate a fact A by swapping a TA assignment and time slot thus creating a new fact B.

Defininte exaclty what Ext does here

Let $T = \{(s, s') | A \to B \in Ext, A \subseteq s, s' = (s - A) \cup B\}$

The search process is defined as $P = (A, Env, K)$ where A is the model defined above, Env is the constraints put upon us and for this instance, and all problems we will be solving in class, are static.

Let $K : S \times Env \to Sand$K(s,e) = (s-A) $\cup B where$A $\to B \in Ext$ and $A \subseteq s$ and $\forall A' \to B' \in Ext$. ... This deifinition is stupid. It's saything that control function chooses the optimal swap for that fact.

$f_{werth} : 2^F \times 2^F \times Env \to \mathbb{N}$ and defined as

$$f_{werth}(x, y, e) = \sum_{i \in x} Cv(i, e) - \sum_{j \in y} Cv(j, e)$$

where $Cv : S \times Env \to \mathbb{N}$. Cv is a function that totals the number of constraint violations a state violates.

3.3 Demonstration of a small search instance

Appendices

Hard Constraints

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

//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)

// 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)

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

//no TA is assigned simultanious 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)

//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.