Πανεπιστήμιο Μακεδονίας

Τμήμα Εφαρμοσμένης Πληροφορικής

Εργασία 4 για το μάθημα 'Σχεδιασμός και Χρονοπρογραμματισμός'

Κωνσταντίνος Πασβάντης
Email: aid23005@uom.edu.gr
Αριθμός Μητρώου: aid23005

10 Μαίου, 2023

1. Compiling Multiple Servers

(A)

Σε αυτό το πρόβλημα είχαμε ένα σύνολο από αρχεία και έπρεπε να βρούμε σε ποιον από τους διαθέσιμους server έπρεπε να γίνει το compilation του καθενός, έτσι ώστε να ελαχιστοποιηθεί το συνολικό lateness, όπου εδώ μετράται ως το άθροισμα των διαφορών μεταξύ του χρόνου ολοκλήρωσης και του deadline του. Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
 % Create 2 arrays, duration and send_time, from the two columns of array timeTo
array[FILES] of int: durations = [timeTo[i,COMP] | i in FILES] ;
array[FILES] of int: send_time = [timeTo[i,REPL] | i in FILES];

set of int : TIME = 0..100;

%% Creating the arrays that contain the start and end time for each file
array[FILES] of var TIME: start;
array[FILES] of var TIME: end;

% Passing in our constraints
% First of all the most basic constraint in time programming
constraint forall(j in FILES)(end[j] = start[j] + durations[j]);

% Creating the in_server array that contains the number of the server the file will be
compiled
array[FILES] of var SERVERS: in_server;

% For each server, we have to satisfy the disjunctive constraint because a server can
compile only one file at a specific time
constraint forall(s in SERVERS) (disjunctive( [start[j]| j in FILES where
in_server[j]=s], [durations[j]| j in FILES ]));



% Finally we have to pass constraints that takes into account the dependencies of each
file
% It is obvious that a file must start compiling after all of its dependency files end
compiling
constraint forall(f in FILES)( forall(d in dependencies[f])(start[f] >= end[d])) ;

% And now the constraint about dependencies. We say that a file must begin compiling
after all of its dependencies end compiling , if they are on the same server, or else
the compilation must begin after the end time of its dependencies + the time needed for
them to arrive on the same server
constraint forall(f in FILES)(forall(d in dependencies[f])(if
in_server[f]!=in_server[d] then start[f]>= end[d]+ send_time[d] else start[f] >= end[d]
endif));


% Creating the variable points that measures the sum of lateness that is defied as
lateness = end - deadline, and we want to minimize it
var int: points;
constraint points = sum([end[i] - deadline[i]|i in FILES]);

% Basic
solve minimize points;


% Different Heuristics #1
% solve ::seq_search(
%          [int_search([in_server[f]|f in FILES], input_order, indomain_random),
%           int_search([start[f]|f in FILES], first_fail, indomain_split)])
%       minimize points;

% Different Heuristics #2
% solve ::seq_search(
```

```
%           [int_search([in_server[f]|f in FILES], first_fail, indomain_median),
%            int_search([start[f]|f in FILES], smallest, indomain_reverse_split)])
      %     minimize points;
```

Ο σχολιασμός του κώδικα περιγράφεται παραπάνω. Αρχικά χρησιμοποιώ τον πίνακα timeTo προκειμένου να δημιουργήσω δύο πίνακες οι οποίοι θα περιλαμβάνουν ο ένας την διάρκεια που χρειάζεται ένα αρχείο ώστε να γίνει compiled και ο άλλος τον χρόνο που χρειάζεται ώστε να μεταφερθεί το compiled αρχείο σε κάποιον άλλον server από αυτόν στον οποίο μεταγλωτίστηκε. Στην συνέχεια, χρησιμοποιούμε τον κλασικό περιορισμό όταν έχουμε να κάνουμε με χρονοπρογραμματισμό, ο οποίος μας λέει ότι το κάθε στοιχείο του πίνακα end θα υπολογίζεται από την σχέση τέλος = αρχή + διάρκεια. Επίσης, δημιουργούμε τον πίνακα in_server και αναφέρουμε ότι για τα αρχεία που κάνουν compile στον ίδιο server, θα πρέπει να ισχύει ο περιορισμός disjunctive, καθώς οι server έχουν χωρητικότητα 1. Έπειτα, λαμβάνουμε υπόψη τον περιορισμό που υπάρχει όσον αφορά τα dependencies κάθε αρχείου, και λέμε ότι για κάθε αρχείο το οποίο έχει κάποιο dependency, προφανώς πρέπει ο χρόνος έναρξής του να είναι μεγαλύτερος ή ίσος από τον χρόνο λήξης μεταγλώτισης όλων των dependencies του. Στην συνέχεια, αναφέρω ότι για να ξεκινήσει ένα αρχείο να μεταγλωτίζεται, πρέπει ο χρόνος εκκίνησής του να είναι μεγαλύτερος από τον χρόνο λήξης μεταγλώτισης των dependencies του, αν αυτά έχουν γίνει στον άλλον server, ή να είναι μεγαλύτερος από τον χρόνο λήξης των dependencies του, αν αυτά έχουν μεταγλωιστεί σε άλλον server.
Τέλος, δημιουργώ την μεταβλητή points, η οποία μετράει το συνολικό lateness που περιγράφηκε και προηγουμένως και θέλω η τιμή αυτής της μεταβλητής να ελαχιστοποιείται.
Πιο κάτω έχω σε σχόλια τον κώδικα που χρησιμοποίησα για τις δύο διαφορετικές ευρετικές τεχνικές που χρησιμοποιήθηκαν στην αναζήτηση λύσεων, τα οποία χρείαστηκαν για το (β) ερώτημα.

Το αποτέλεσμα του κώδικα, σε συνδυασμό με αυτόν που είχαμε εμείς διαθέσιμο, για το πρώτο αρχείο με τους δύο server, files0.dzn, μετά από 9 επαναλήψεις (θα αναφέρω μόνο τις δύο τελευταίες), καταφέρνει να βρεί την βέλτιστη λίση, η οποία είναι ίση με 48:

```
. . . . . .
----------
points = 52 ;
start = [0, 0, 23, 10, 53, 38] ;
in_server = [2, 1, 1, 1, 1, 1] ;
----------
points = 48 ;
start = [0, 0, 15, 10, 30, 65] ;
in_server = [1, 2, 1, 2, 1, 2] ;
----------
==========
```

Το αποτέλεσμα του κώδικα, σε συνδυασμό με αυτόν που είχαμε εμείς διαθέσιμο, για το δεύτερο αρχείο με τους τρεις server, files3.dzn, μετά από 15 επαναλήψεις (θα αναφέρω μόνο τις δύο τελευταίες), καταφέρνει να βρεί την βέλτιστη λίση, η οποία είναι ίση με 112:

```
. . . . . .
----------
points = 118 ;
start = [0, 6, 15, 26, 30, 65, 50, 80, 16, 14, 0, 85, 0] ;
in_server = [2, 1, 2, 1, 2, 1, 2, 1, 1, 3, 1, 1, 3] ;
----------
points = 112 ;
start = [0, 0, 15, 20, 30, 65, 50, 80, 10, 20, 0, 85, 6] ;
in_server = [2, 1, 2, 1, 2, 1, 2, 1, 1, 3, 3, 1, 3] ;
----------
==========
```

Όσον αφορά τα bugs, τα μόνα που παρατήρησα είναι πριν να ολοκληρώσω τον κώδικα, δηλαδή πριν βρω τον περιορισμό που ανέφερα και προηγουμένως. Πιο συγκεκριμένα, βλέποντας στο output τον χρόνο έναρξης του κάθε compilation, είχα κάνει ένα λάθος όσον αφορά τον περιορισμό, και δεν μπορούσα να καταλάβω ότι ενώ τα περισσότερα φαινόταν σωστά, για κάποιο λόγο μερικές φορές τα αρχεία ξεκινούσαν

από χρονικές στιγμές οι οποίες δεν ήταν λογικές. Για παράδειγμα, τα ένα αρχείο που δεν είχε κάποιο dependency ξεκινούσε την στιγμή 0, ενώ το άλλο ξεκινούσε την στιγμή 2, ενώ θα μπορούσε να ξεκινήσει και νωρίτερα. Επίσης, είμαι αρκετά σίγουρος ότι δεν είναι bug, αλλά όταν τρέχω τον κώδικα για το αρχείο files2.dzn, δεν χρησιμοποιείται καθόλου ο τρίτος server.

(B)

Παρακάτω θα παρουσιαστεί ο πίνακας από τον χρόνο ολοκλήρωσης του προβλήματος χρησιμοποιώντας τρεις διαφορετικές τεχνικές αναζήτησης. Θα αναφερόμαστε σε αυτές, όπως αναφέρονται και στον κώδικα στα σχόλια, ως Basic, Different Heuristics #1 και Different Heuristics #2. Επίσης, καθώς ο κώδικας βγάζει διαφορετικό χρόνο ολοκλήρωσης κάθε φορά που τον τρέχω, αποφάσισα για κάθε πρόβλημα και για κάθε διαφορετική περίπτωση, να τρέξω τον κώδικα τρεις φορές και να χρησιμοποιήσω τον μέσο όρο της χρονικής τους διάρκειας για κάθε περίπτωση. Τα αποτελέσματα φαίνονται παρακάτω:

Αρχείο Προβλήματος	Χρόνος επίλύσης με την μέθοδο basic	Χρόνος επίλύσης με την μέθοδο Different Heuristics 1	Χρόνος επίλύσης με την μέθοδο Different Heuristics 2
files0.dzn	478.33 msec	444 msec	523.33 msec
files1.dzn	516.66 msec	547.33 msec	689.33 msec
files2.dzn	570.66 msec	598.66 msec	661.33 msec
files3.dzn	989.33 msec	923.66 msec	1338 msec

2. Parking Trucks

Σε αυτό το πρόβλημα μας ζητήθηκε να τοποθετήσουμε τα διαθέσιμα φορτηγά που το κάθε ένα έχει συγκεκριμένο μήκος, σε λωρίδες συγκεκριμένου μήκους, ώστε να ελαχιστοποιήσουμε τον αριθμό των λωρίδων που χρησιμοποιήθηκαν. Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
% Creating allocations to know in which lane we can place a truck
array[TRUCKS] of var set of LANES: allocations;
constraint forall( i in TRUCKS)( allocations[i] = {j| j in LANES where
truck_type[i] = parking_lane_type[j]});

% The array at_parking_lane must take values from each set used in allocations
for each truck
constraint forall(i in TRUCKS)(at_parking_lane[i] in allocations[i]);

% The sum of the length of trucks in a lane must be lower than the total lane
length
constraint forall(i in LANES) ( sum( [truck_length[j]| j in TRUCKS where
at_parking_lane[j] == i]) <= parking_lane_length);
```

```
% We create a set that holds the unique numbers of lanes used
var set of int: lanes_used = {i |i in at_parking_lane};

% And now we say that the number of lanes we used is equal to the cardinality
of the above set
constraint used_lanes = card(lanes_used);

% Finally we want to minimize the used_lanes
   solve minimize used_lanes;
```

Ο σχολιασμός του κώδικα φαίνεται παραπάνω. Στην αρχή, σκεπτόμενος ότι αυτή η άσκηση θα λυθεί όπως και τα παραδείγματα των διαφανειών, δημιούργησα τον πίνακα allocations ο οποίος έχει τόσα στοιχεία όσα είναι και τα φορτηγά, και έχει για κάθε στοιχείο ένα σετ αριθμών οι οποίοι υποδηλώνουν την λωρίδα στην οποία μπορει να τοποθετηθεί το συγκεκριμένο φορτηγό. Αυτό το κατάφερα με list και sets manipulation και δήλωσα ότι για κάθε στοιχείο αυτού του πίνακα θα δημιουργηθεί ένα set, το οποίο θα έχει τον αριθμό των λωρίδων που είναι συμβατές με το εκάστοτε φορτηγό. Στην συνέχεια δήλωσα ότι το κάθε στοιχείο που θέλουμε να βρούμε του πίνακα at_parking_lane, πρέπει να έχει τιμές από το set κάθε φορτηγού που έχουμε αποθηκεύσει στον πίνακα allocations. Το τελευταίο list manipulation που έκανα, είναι για να ικανοποιήσω τον περιορισμό ότι σε κάθε λωρίδα το μήκος το φορτηγών που θα τοποθετηθούν δεν θα πρέπει να ξεπερνάνε το ανώτατο όριο του κάθε προβλήματος. Τέλος, για να βρούμε τον αριθμό των συνολικών λωρίδων που χρησιμοποιήθηκαν, μετατρέπουμε το array at_parking_lane σε set, και μετά παίρνοντας τον πληθικό αριθμό του, βρίσκω το ζητούμενο.
Για το πρώτο παράδειγμα που είχαμε στην διάθεσή μας με τον τίτλο parkingData1.dzn, ο κώδικας βρήκε την βέλτιστη λύση σε ένα βήμα, η οποία είναι η εξής:

```
at_parking_lane = [2, 3, 1, 2, 1];
used_lanes = 3;
----------
==========
```

Για το δεύτερο παράδειγμα που είχαμε με τον τίτλο parkingData2.dzn, ο κώδικας βρήκε την βέλτιστη λύση πάλι σε ένα βήμα, η οποία είναι η εξής:

```
at_parking_lane = [4, 3, 6, 4, 3, 2, 4, 4, 2, 4, 2, 2, 1, 1, 1];
used_lanes = 5;
----------
==========
```

Σε αυτόν τον κώδικα, στην αρχή νόμιζα ότI βρέθηκε ένα bug, όταν έτρεξα το τρίτο αρχείο με τίτλο parkingData3.dzn. Πιο συγκεκριμένα, ο κώδικας έτρεχε για αρκετή ώρα, αφού όμως είχε παράξει μία λύση, για να βρει την βέλτιστη δυνατή. Τελικά όμως μετά από 6 λεπτά και 51 δευτερόλεπτα ο κώδικας σταμάτησε, χωρίς να βρει κάποια άλλη λύση πέρα της πρώτης που είχε ήδη βρει. Παρόλα αυτά είμαι αρκετά σίγουρος ότι δεν υπάρχει κάποιο bug, καθώς για την διασταύρωση των αποτελεσμάτων, χρησιμοποίησα και τον πίνακα allocations για να δω ότι όλα υπολογίζονται έτσι όπως θα έπρεπε.

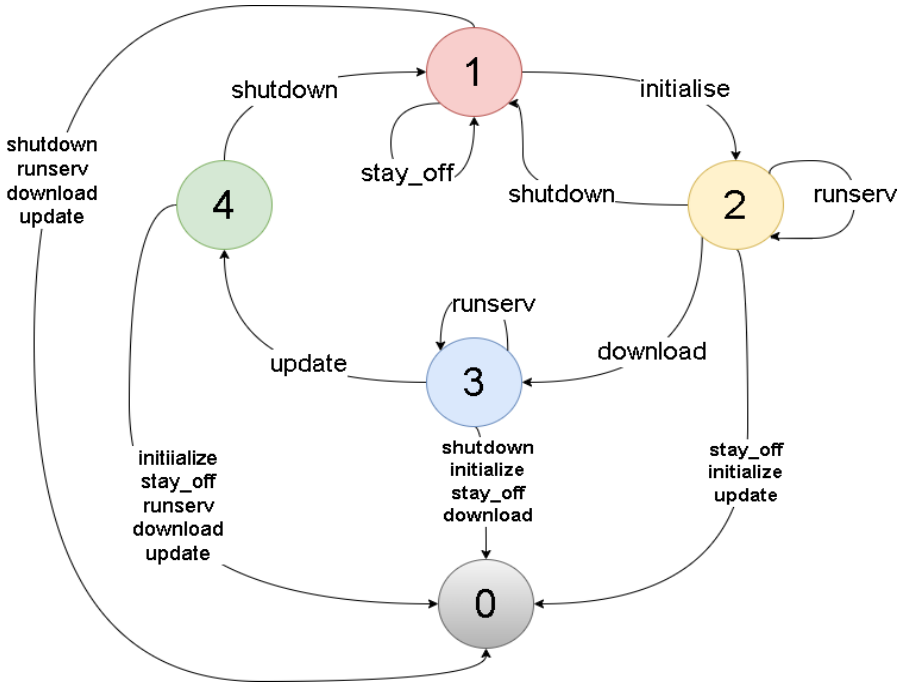
3. Server Commands

Στο συγκεκριμένο πρόβλημα έπρεπε να βρούμε την κατάλληλη ακολουθία εντολών που θα εκτελούσε ο κάθε server σε χρονική διάρκεια 7 χρονικών στιγμών, προκειμένου να ενημερωθούν όλοι, ικανοποι-ώντας συγκεκριμένους περιορισμούς. Για να γίνει πιο κατανοητός ο κώδικας, παρακάτω θα αναφέρω τις καταστάσεις που χρησιμοποίησα και το τι δηλώνει η κάθε μία. Στο παρακάτω σχήμα που μοντελοποιεί το αυτόματο του συγκεκριμένου προβλήματος βλέπουμε 4 βασικές καταστάσεις και μία απορροφητική

κατάσταση (κατάσταση 0). Οι 4 βασικές καταστάσεις που απεικονίζονται είναι οι εξής:

1. Closed – Ο server είναι κλειστός
2. Open – Ο server είναι ανοιχτός
3. Ready to update – Ο server κατέβασε τα απαραίττητα αρχεία με την ενέργεια download και βρίσκεται σε θέση να κάνει το update.
4. Updated – Ο server έκανε την ενημέρωση που έπρεπε να κάνει.

Με τις παραπάνω καταστάσεις δημιούργησα το εξής αυτόματο :



Οπότε με αυτό το σκεπτικό πήγα και δημιούργησα τον πίνακα που χρειαζόμαστε.
Ο κώδικάς μου είναι ο εξής:

```
% We have 4 states, each of them described in report
int: ns = 4;
set of int : STATE = 1..ns;

% Declaring the array
array[SERVERS,COMMANDS] of int: allowed =
                        %stay_off  initialize runserv shutdown download update
                        [|1,           2,         0,      0,       0,      0
                         |0,           0,         2,      1,       3,      0
                         |0,           0,         3,      0,       0,      4
                         |0,           0,         0,      1,       0,      0|];


% Passing in the regular constraint
constraint forall(s in SERVERS)
 (regular(row(server_commands,s),ns,card(COMMANDS), allowed, 1, STATE));

% Using function count to say that each server must be updated only once
constraint forall(s in SERVERS)
 (sum([server_commands[s,d] == update | d in TIME]) == 1);

% Now for the constraint about simultaneous downloads
constraint forall(d in TIME)
 (sum([server_commands[s,d] == download | s in SERVERS]) < 2);

% Now we say that we want each time_point to have at least one opperation runserv,
except day number one, so we must iterate from day 2
constraint forall(d in 2..no_time_points)
 (sum([server_commands[s,d] == runserv | s in SERVERS]) >= 1);
```

```
% We say that max is the number of total runserv, when we iterate over the entire the
array
constraint max = count([server_commands[s,t] | s in SERVERS, t in TIME], runserv);

% Last, we want to maximize the count of runserv.
solve maximize max;
```

Αρχικά ορίζουμε τον αριθμό των καταστάσεων και τον πίνακα μεταβάσεων των βασικών μας καταστάσεων, ώστε να χρησιμοποιήσουμε κατάλληλα τον περιορισμό regular για κάθε γραμμή του πίνακα μας. Στην συνέχεια δηλώνουμε τους υπόλοιπους περιορισμούς που χρειάζονται για να ικανοποιηθεί το πρόβλημά μας, ξεκινώντας από το ότι πρέπει ο κάθε server να κάνει update μόνο μία φορά. Στην συνέχεια αναφέρουμε ότι κάθε χρονική στιγμή πρέπει να γίνεται το πολύ μία ενέργεια download σε κάποιον server και τέλος, ότι κάθε στιγμή πρέπει να υπάρχει τουλάχιστον μία ενέργεια runserv. Αυτοί οι περιορισμοί είναι οι αντίστοιχοι του απραδείγματος που υπάρχουν στις διαφάνειες. Τελικά λέμε ότι πρέπει να μεγιστοποιήσουμε την μεταβλητή max, την οποία ορίσαμε να είναι ο συνολικός αριθμός ενεργειών runserv, χρησιμοποιώντας την συνάρτηση count. Το αποτέλεσμα του παραπάνω κώδικα σε συνδυασμό με αυτόν που ήδη υπήρχε στο αρχείο που μας δώθηκε, είναι (παραθέτω τις δύο τελευταίες επαναλήψεις από τις εννιά συνολικά) :

server_commands=array2d(1..4,1..7,[initilize, runserv, runserv, runserv, download, runserv, update, stay_off, initilize, runserv, runserv, runserv, download, update, initilize, download, runserv, update, shutdown, initilize, runserv, initilize, runserv, runserv, download, runserv, runserv, update]);
max=13;
---
server_commands=array2d(1..4,1..7,[initilize, runserv, runserv, runserv, download, runserv, update, initilize, runserv, runserv, runserv, runserv, download, update, initilize, download, runserv, update, shutdown, initilize, runserv, initilize, runserv, runserv, download, runserv, runserv, update]);
max=14;

==========

Ο συγκεκριμένος κώδικας δεν μου δημιούργησε κάποιο bug.