Ευστάθιος Ιωσηφίδης

AM: mai25017

EMAIL:
mai25017@uom.edu.gr

Άσκηση 1: Client/Server Allocation

Ο κώδικας της άσκησης 1 μοντελοποιεί ένα πρόβλημα βελτιστοποίησης κατανομής πελατών σε διακομιστές, με στόχο την ελαχιστοποίηση του κόστους, υπό τους περιορισμούς χωρητικότητας.
Ο κώδικας που χρησιμοποιήθηκε στην επίλυση της άσκησης 1 ήταν

```
%% Number of Servers and Clients and related Sets
par int: Servers;
par int: Clients;

set of int:SERVERS = 1..Servers;
set of int:CLIENTS = 1..Clients;

%% Arrays stating Problem Data
array[SERVERS] of int: Capacity;
array[SERVERS] of int: DeploymentCost;
array[CLIENTS] of int: ClientDemands ;
array[CLIENTS,SERVERS] of int: NetworkingCost;

%% Add your code here

%% Decision Variables
array[CLIENTS] of var SERVERS: client_at;
var int: total_cost;

%% Helper Variables
array[SERVERS] of var int: server_load =
   [sum(c in CLIENTS where client_at[c] ==
s)(ClientDemands[c]) | s in SERVERS];

array[SERVERS] of var 0..1: server_active;

%% Constraints
constraint forall(s in SERVERS) (
   server_load[s] <= Capacity[s]);

constraint forall(s in SERVERS) (
   server_active[s] = (server_load[s] > 0));

%% Calculate total cost
var int: total_deployment_cost = sum(s in
SERVERS)(server_active[s] * DeploymentCost[s]);
```

```
var int: total_networking_cost = sum(c in
CLIENTS)(ClientDemands[c] * NetworkingCost[c, client_at[c]]);

constraint total_cost = total_deployment_cost +
total_networking_cost;


%% Solve to minimize total cost
solve minimize total_cost;

%%% DO NOT DELETE THIS LINES
output ["client_at = \(client_at); \n total_cost=\(total_cost);\n"];
```

Ας πάρουμε τον κώδικα αναλυτικά ανά μπλοκ.

1. Number of Servers and Clients and related Sets
Κώδικας:
%% Number of Servers and Clients and related Sets
par int: Servers;
par int: Clients;

set of int:SERVERS = 1..Servers;
set of int:CLIENTS = 1..Clients;

Παράμετροι:
Servers: Ορίζει τον αριθμό των διακομιστών.
Clients: Ορίζει τον αριθμό των πελατών.

Σύνολα:
SERVERS = 1..Servers: Δημιουργεί ένα σύνολο με αναγνωριστικά διακομιστών.
CLIENTS = 1..Clients: Δημιουργεί ένα σύνολο με αναγνωριστικά πελατών.

2. Arrays stating Problem Data
Κώδικας:
%% Arrays stating Problem Data
array[SERVERS] of int: Capacity;
array[SERVERS] of int: DeploymentCost;
array[CLIENTS] of int: ClientDemands;
array[CLIENTS,SERVERS] of int: NetworkingCost;

Πίνακες Δεδομένων
Capacity: Πίνακας που καθορίζει τη μέγιστη χωρητικότητα κάθε διακομιστή.
DeploymentCost: Πίνακας με το κόστος ανάπτυξης κάθε διακομιστή.
ClientDemands: Πίνακας που ορίζει το φορτίο/ζήτηση κάθε πελάτη.
NetworkingCost: 2D πίνακας που καθορίζει το κόστος δικτύωσης ανά μονάδα φορτίου για κάθε ζεύγος (πελάτης, διακομιστής).

Ακολουθεί ο κώδικας που προστέθηκε για να απαντηθεί το ζητούμενο, να βρεθεί λύση (στον πίνακα client_at) με το ελάχιστο συνολικό κόστος (μεταβλητή total_cost)

3. Μεταβλητές απόφασης
Κώδικας:
```
%% Decision Variables
array[CLIENTS] of var SERVERS: client_at;
var int: total_cost;
```

client_at: Πίνακας που αντιστοιχεί κάθε πελάτη (client) σε έναν διακομιστή (server). Κάθε στοιχείο client_at[c] δείχνει σε ποιον διακομιστή έχει ανατεθεί ο πελάτης c.
Σημασία: Αυτή είναι η κύρια μεταβλητή απόφασης – καθορίζει τη βέλτιστη κατανομή των πελατών σε διακομιστές.

total_cost: Μεταβλητή που θα περιέχει το συνολικό κόστος της λύσης.
Σημασία: Είναι ο **στόχος βελτιστοποίησης (ελαχιστοποίησης)**.

4. Βοηθητικές Μεταβλητές
Κώδικας:
```
%% Helper Variables
array[SERVERS] of var int: server_load =
   [sum(c in CLIENTS where client_at[c] == s)(ClientDemands[c]) | s in SERVERS];

array[SERVERS] of var 0..1: server_active;
```

server_load: Πίνακας που υπολογίζει το φορτίο κάθε διακομιστή s ως το άθροισμα των αιτημάτων (ClientDemands) όλων των πελατών που του έχουν ανατεθεί.
Σημασία: Βοηθά να ελεγχθεί αν ο διακομιστής μπορεί να υποστηρίξει το φορτίο.

Σημείωση: Η χρήση where client_at[c] == s φιλτράρει τους πελάτες που χρησιμοποιούν τον διακομιστή s.

server_active: Δυαδικός πίνακας (0 ή 1) που δείχνει αν ένας διακομιστής s χρησιμοποιείται (1) ή όχι (0).
Σημασία: Χρειάζεται για τον υπολογισμό του κόστους ανάπτυξης (DeploymentCost), το οποίο πληρώνεται μόνο για ενεργούς διακομιστές.

5. Constraints
Κώδικας:
```
%% Constraints
constraint forall(s in SERVERS) (
   server_load[s] <= Capacity[s]);

constraint forall(s in SERVERS) (
   server_active[s] = (server_load[s] > 0));
```

Πρώτος Περιορισμός: Εξασφαλίζει ότι το φορτίο κάθε διακομιστή s (server_load) δεν ξεπερνά την χωρητικότητά του (Capacity[s]).

Σημασία: Διατηρεί τη λύση εφικτή, αποτρέποντας υπερφόρτωση.

Δεύτερος Περιορισμός: Συνδέει τη μεταβλητή server_active[s] με το φορτίο του διακομιστή. Αν server_load[s] > 0, τότε server_active[s] = 1 (διαφορετικά 0).
Σημασία: Ενεργοποιεί το κόστος ανάπτυξης μόνο για διακομιστές που χρησιμοποιούνται.

6. Υπολογισμός Κόστους
Κώδικας:
%% Calculate total cost
var int: total_deployment_cost = sum(s in SERVERS)(server_active[s] * DeploymentCost[s]);
var int: total_networking_cost = sum(c in CLIENTS)(ClientDemands[c] * NetworkingCost[c, client_at[c]]);

constraint total_cost = total_deployment_cost + total_networking_cost;

total_deployment_cost: Αθροίζει το κόστος ανάπτυξης (DeploymentCost) όλων των ενεργών διακομιστών.
Σημασία: Αντιπροσωπεύει το σταθερό κόστος χρήσης διακομιστών.

total_networking_cost: Αθροίζει το κόστος δικτύου (NetworkingCost) για κάθε πελάτη c, πολλαπλασιασμένο με τη ζήτησή του (ClientDemands). Το κόστος εξαρτάται από τον διακομιστή (client_at[c]) στον οποίο έχει ανατεθεί ο πελάτης.
Σημασία: Αντιπροσωπεύει το μεταβλητό κόστος επικοινωνίας.

total_cost: Το total_cost είναι το άθροισμα και των δύο παραπάνω.
Σημασία: Είναι η ποσότητα που ελαχιστοποιείται στη λύση.

7. Επίλυση
Κώδικας:
%% Solve to minimize total cost
solve minimize total_cost;

Στόχος: Ελαχιστοποίηση του total_cost (συνολικό κόστος ανάπτυξης + δικτύωσης).
Μηχανισμός: Ο solver θα αναζητήσει την κατανομή client_at που ικανοποιεί όλους τους περιορισμούς και ελαχιστοποιεί το κόστος.

7. Output
Κώδικας:
%%% DO NOT DELETE THIS LINES
output [" client_at = \(client_at); \n total_cost=\(total_cost);\n"];

Εμφανίζει τη βέλτιστη ανάθεση πελατών σε διακομιστές (client_at) και το συνολικό κόστος (total_cost).

Παραδείγματα εκτέλεσης

server_set0.dzn	server_set2.dzn	server_set4.dzn
Servers = 4; Clients = 6;	Servers = 10; Clients = 10;	Servers = 10; Clients = 20;

Running serverAlloc.mzn, server_set0.dzn
141msec
 	
client_at = [3, 3, 1, 2, 2, 1];
total_cost=7508;
---------- 	
client_at = [4, 4, 1, 2, 2, 1];
total_cost=7406;
---------- 	
client_at = [3, 3, 1, 3, 2, 1];
total_cost=7196;
---------- 	
client_at = [3, 3, 1, 2, 3, 1];
total_cost=7028;
---------- 	
client_at = [2, 3, 1, 3, 3, 1];
total_cost=6980;
---------- 	
client_at = [3, 4, 1, 3, 3, 1];
total_cost=6572;
---------- 	
client_at = [3, 3, 1, 2, 1, 2];
total_cost=6518;
---------- 	
client_at = [2, 3, 1, 3, 1, 2];
total_cost=6470;
---------- 	
client_at = [3, 3, 1, 3, 1, 2];
total_cost=6206;
---------- 	
client_at = [3, 1, 3, 1, 3, 2];
total_cost=6104;
---------- 	
client_at = [2, 3, 3, 3, 3, 2];
total_cost=6075;
---------- 	
client_at = [3, 4, 3, 3, 3, 2];
total_cost=6017;
---------- 	
client_at = [3, 4, 1, 3, 1, 3];
total_cost=5912;
---------- 	
client_at = [1, 4, 1, 1, 3, 3];
total_cost=5856;
---------- 	
client_at = [3, 4, 3, 3, 4, 3];
	Running serverAlloc.mzn, server_set2.dzn
	200msec
	
	client_at = [3, 4, 2, 2, 2, 3, 3, 1, 1, 1];
	total_cost=8745;
	---------- 	
	client_at = [3, 3, 2, 2, 2, 5, 3, 1, 1, 1];
	total_cost=8394;
	---------- 	
	client_at = [3, 3, 2, 2, 2, 9, 3, 1, 1, 1];
	total_cost=8077;
	---------- 	
	client_at = [7, 7, 2, 2, 2, 1, 7, 7, 1, 1];
	total_cost=7890;
	---------- 	
	client_at = [7, 2, 7, 2, 2, 1, 7, 7, 1, 1];
	total_cost=7869;
	---------- 	
	client_at = [7, 3, 7, 2, 2, 1, 7, 7, 1, 1];
	total_cost=7832;
	---------- 	
	client_at = [7, 7, 7, 2, 2, 1, 7, 2, 1, 1];
	total_cost=7735;
	---------- 	
	client_at = [7, 7, 7, 2, 2, 2, 1, 7, 1, 1];
	total_cost=7665;
	---------- 	
	client_at = [7, 7, 7, 2, 2, 2, 7, 1, 1, 1];
	total_cost=7662;
	---------- 	
	client_at = [2, 7, 7, 7, 2, 1, 7, 7, 1, 1];
	total_cost=7658;
	---------- 	
	client_at = [2, 7, 7, 7, 2, 2, 7, 7, 1, 1];
	total_cost=7625;
		Running serverAlloc.mzn, server_set4.dzn
		6s 254msec
		
		client_at = [4, 4, 3, 4, 4, 4, 4, 1, 4, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=14330;
		---------- 	
		client_at = [4, 4, 3, 5, 4, 4, 4, 1, 4, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=14301;
		---------- 	
		client_at = [4, 4, 3, 5, 5, 4, 4, 1, 4, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=13941;
		---------- 	
		client_at = [4, 4, 3, 7, 7, 4, 4, 1, 4, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=13819;
		---------- 	
		client_at = [4, 4, 3, 5, 5, 5, 4, 1, 4, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=13523;
		---------- 	
		client_at = [4, 4, 3, 7, 7, 7, 4, 1, 4, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=13247;
		---------- 	
		client_at = [4, 4, 3, 7, 7, 7, 4, 1, 7, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=12941;
		---------- 	
		client_at = [4, 4, 3, 7, 7, 7, 4, 1, 1, 4, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=12879;
		---------- 	
		client_at = [4, 4, 3, 7, 7, 7, 4, 1, 7, 4, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];

total_cost=5777;
---
client_at = [3, 4, 1, 3, 1, 4];
total_cost=5462;
---
client_at = [1, 4, 1, 1, 3, 4];
total_cost=5406;
---
client_at = [3, 4, 3, 3, 3, 4];
total_cost=5067;

==========
Finished in 141msec.

---
client_at = [7, 2, 7, 7, 3, 1, 2, 2, 1, 1];
total_cost=7592;
---
client_at = [3, 2, 7, 7, 3, 1, 3, 2, 1, 1];
total_cost=7520;
---
client_at = [7, 2, 7, 7, 3, 1, 3, 2, 1, 1];
total_cost=7466;
---
client_at = [7, 2, 7, 2, 3, 1, 7, 2, 1, 1];
total_cost=7452;
---
client_at = [7, 2, 3, 7, 3, 1, 7, 2, 1, 1];
total_cost=7382;
---
client_at = [3, 2, 7, 7, 3, 1, 7, 2, 1, 1];
total_cost=7376;
---
client_at = [7, 2, 7, 7, 3, 1, 7, 2, 1, 1];
total_cost=7322;
---
client_at = [7, 2, 7, 7, 3, 1, 3, 7, 1, 1];
total_cost=7321;
---
client_at = [7, 2, 7, 2, 3, 1, 7, 7, 1, 1];
total_cost=7307;
---
client_at = [7, 2, 3, 7, 3, 1, 7, 7, 1, 1];
total_cost=7237;
---
client_at = [3, 2, 7, 7, 3, 1, 7, 7, 1, 1];
total_cost=7231;
---
client_at = [3, 3, 2, 7, 3, 1, 7, 7, 1, 1];
total_cost=7144;
---
client_at = [7, 3, 2, 7, 3, 1, 7, 7, 1, 1];
total_cost=7090;
---
client_at = [7, 3, 7, 2, 3, 1,

total_cost=12771;
---
client_at = [4, 4, 3, 7, 1, 7, 4, 7, 7, 4, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12727;
---
client_at = [4, 4, 3, 7, 1, 7, 4, 7, 7, 7, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12687;
---
client_at = [5, 10, 3, 7, 1, 7, 10, 5, 7, 5, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12673;
---
client_at = [5, 10, 3, 7, 7, 7, 10, 1, 7, 7, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12632;
---
client_at = [5, 7, 3, 7, 1, 7, 10, 5, 7, 7, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12628;
---
client_at = [5, 10, 3, 7, 1, 7, 10, 5, 7, 7, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12583;
---
client_at = [7, 4, 3, 7, 7, 7, 4, 1, 4, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12581;
---
client_at = [7, 7, 3, 7, 10, 7, 10, 1, 7, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
total_cost=12483;
---
client_at = [7, 10, 3, 7, 10, 7, 10, 1, 7, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1];
total_cost=12438;
---
client_at = [7, 10, 3, 7, 7, 7, 10, 1, 10, 1, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1];
total_cost=12267;
---
client_at = [7, 4, 3, 7, 7, 7, 4, 1, 1, 4, 3, 2, 2, 1, 1, 1, 1,

	3, 7, 1, 1];
	total_cost=7064;
	---------- 	
	client_at = [3, 3, 7, 2, 3, 1, 7, 7, 1, 1];
	total_cost=6974;
	---------- 	
	client_at = [7, 3, 7, 2, 3, 1, 7, 7, 1, 1];
	total_cost=6920;
	---------- 	
	client_at = [7, 3, 7, 3, 3, 1, 7, 3, 1, 1];
	total_cost=6904;
	---------- 	
	client_at = [7, 3, 7, 3, 3, 1, 7, 7, 1, 1];
	total_cost=6739;
	---------- 	
	client_at = [7, 3, 7, 7, 3, 1, 7, 3, 1, 1];
	total_cost=6605;
	---------- 	
	client_at = [7, 3, 7, 7, 3, 1, 3, 7, 1, 1];
	total_cost=6584;
	---------- 	
	client_at = [3, 3, 7, 7, 3, 1, 7, 7, 1, 1];
	total_cost=6494;
	---------- 	
	client_at = [7, 3, 7, 7, 3, 1, 7, 1, 3, 1];
	total_cost=6473;
	---------- 	
	client_at = [3, 3, 7, 7, 3, 1, 7, 1, 7, 1];
	total_cost=6410;
	---------- 	
	client_at = [3, 3, 7, 7, 3, 1, 7, 7, 7, 1];
	total_cost=6305;
	---------- 	
	client_at = [1, 3, 3, 7, 10, 1, 7, 7, 7, 3];
	total_cost=6285;
	---------- 	
	client_at = [1, 3, 7, 7, 3, 1, 7, 7, 3, 3];
	total_cost=5930;
	---------- 	
	client_at = [1, 3, 7, 7, 3, 1, 7, 7, 7, 3];
	total_cost=5813;
		1, 1, 1];
		total_cost=12213;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 4, 7, 7, 4, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=12061;
		---------- 	
		client_at = [7, 10, 3, 7, 1, 7, 10, 7, 10, 7, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=12013;
		---------- 	
		client_at = [10, 10, 3, 7, 1, 7, 10, 7, 7, 7, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11986;
		---------- 	
		client_at = [7, 3, 7, 5, 1, 7, 3, 5, 7, 5, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11940;
		---------- 	
		client_at = [7, 3, 7, 7, 1, 5, 3, 5, 7, 5, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11938;
		---------- 	
		client_at = [7, 4, 3, 7, 7, 1, 3, 1, 7, 4, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11901;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 7, 3, 4, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11803;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 7, 3, 7, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11763;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 7, 7, 4, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11659;
		---------- 	
		client_at = [3, 4, 7, 7, 1, 7, 3, 7, 7, 4, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11653;
		---------- 	
		client_at = [7, 3, 3, 7, 1, 7,

	---------- 	
	client_at = [7, 3, 7, 7, 3, 2, 7, 2, 2, 3];
	total_cost=5770;
	---------- 	
	client_at = [7, 3, 7, 7, 3, 2, 3, 7, 2, 3];
	total_cost=5769;
	---------- 	
	client_at = [7, 3, 7, 2, 3, 2, 7, 7, 2, 3];
	total_cost=5755;
	---------- 	
	client_at = [7, 3, 7, 2, 3, 2, 7, 7, 3, 3];
	total_cost=5665;
	---------- 	
	client_at = [2, 3, 7, 7, 3, 2, 7, 7, 7, 3];
	total_cost=5616;
	---------- 	
	client_at = [9, 3, 7, 7, 3, 9, 7, 7, 7, 3];
	total_cost=5613;
	---------- 	
	==========
	Finished in 200msec.
		3, 5, 7, 5, 5, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11504;
		---------- 	
		client_at = [3, 3, 7, 7, 1, 7, 3, 5, 7, 5, 5, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11498;
		---------- 	
		client_at = [3, 10, 3, 7, 1, 7, 10, 10, 7, 7, 7, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11484;
		---------- 	
		client_at = [7, 3, 3, 5, 1, 7, 3, 5, 7, 5, 7, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11439;
		---------- 	
		client_at = [7, 3, 3, 7, 1, 5, 3, 5, 7, 5, 7, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11437;
		---------- 	
		client_at = [9, 3, 3, 7, 1, 7, 3, 9, 7, 7, 7, 2, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11248;
		---------- 	
		client_at = [2, 10, 3, 7, 1, 7, 10, 2, 7, 7, 7, 3, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11038;
		---------- 	
		client_at = [7, 4, 3, 7, 7, 1, 3, 2, 7, 4, 2, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=11029;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 2, 7, 3, 2, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10837;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 2, 3, 4, 2, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10831;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 2, 7, 4, 2, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10687;
		---------- 	

		client_at = [3, 4, 7, 7, 1, 7, 3, 2, 7, 4, 2, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10681;
		---------- 	
		client_at = [3, 4, 2, 7, 1, 7, 3, 7, 7, 4, 7, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10603;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 2, 3, 2, 7, 4, 7, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10480;
		---------- 	
		client_at = [2, 4, 3, 7, 1, 7, 3, 2, 7, 4, 7, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10392;
		---------- 	
		client_at = [2, 4, 3, 7, 1, 7, 3, 2, 7, 7, 7, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10352;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 2, 3, 4, 7, 4, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10338;
		---------- 	
		client_at = [2, 3, 3, 7, 1, 7, 3, 2, 7, 7, 7, 8, 2, 1, 1, 1, 1, 1, 1, 1];
		total_cost=10281;
		---------- 	
		client_at = [7, 4, 3, 7, 1, 7, 3, 2, 3, 4, 7, 4, 1, 2, 1, 1, 1, 1, 1, 1];
		total_cost=10202;
		---------- 	
		client_at = [2, 3, 3, 7, 1, 7, 3, 2, 7, 7, 7, 8, 1, 2, 1, 1, 1, 1, 1, 1];
		total_cost=10145;
		---------- 	
		client_at = [7, 4, 1, 7, 1, 7, 3, 2, 3, 4, 7, 4, 2, 2, 1, 1, 1, 1, 1, 1];
		total_cost=10121;
		---------- 	
		client_at = [3, 4, 1, 7, 1, 7, 3, 2, 7, 4, 7, 4, 2, 2, 1, 1, 1, 1, 1, 1];
		total_cost=10031;

		---------- 	
		client_at = [3, 4, 1, 7, 1, 7, 3, 2, 7, 7, 7, 4, 2, 2, 1, 1, 1, 1, 1, 1];
		total_cost=9991;
		---------- 	
		client_at = [1, 4, 3, 7, 1, 7, 3, 2, 7, 4, 7, 4, 2, 2, 1, 1, 1, 1, 1, 1];
		total_cost=9979;
		---------- 	
		client_at = [1, 4, 3, 7, 1, 7, 3, 2, 7, 7, 7, 4, 2, 2, 1, 1, 1, 1, 1, 1];
		total_cost=9939;
		---------- 	
		client_at = [3, 3, 1, 7, 1, 7, 3, 2, 7, 7, 7, 8, 2, 2, 1, 1, 1, 1, 1, 1];
		total_cost=9920;
		---------- 	
		client_at = [1, 3, 3, 7, 1, 7, 3, 2, 7, 7, 7, 8, 2, 2, 1, 1, 1, 1, 1, 1];
		total_cost=9868;
		---------- 	
		client_at = [7, 4, 1, 2, 1, 7, 3, 2, 7, 4, 7, 4, 2, 1, 3, 1, 1, 1, 1, 1];
		total_cost=9859;
		---------- 	
		client_at = [1, 4, 7, 2, 1, 7, 3, 2, 7, 4, 7, 4, 2, 1, 3, 1, 1, 1, 1, 1];
		total_cost=9801;
		---------- 	
		client_at = [7, 4, 1, 7, 1, 7, 3, 2, 3, 4, 7, 4, 2, 1, 3, 1, 1, 1, 1, 1];
		total_cost=9730;
		---------- 	
		client_at = [2, 3, 1, 7, 1, 7, 3, 2, 7, 7, 7, 8, 2, 1, 3, 1, 1, 1, 1, 1];
		total_cost=9673;
		---------- 	
		client_at = [1, 4, 7, 2, 1, 7, 3, 2, 7, 4, 7, 4, 1, 2, 3, 1, 1, 1, 1, 1];
		total_cost=9665;
		---------- 	
		client_at = [7, 4, 1, 7, 1, 7, 3, 2, 3, 4, 7, 4, 1, 2, 3, 1, 1, 1, 1, 1];

		total_cost=9594;
		---------- 	
		client_at = [2, 3, 1, 7, 1, 7, 3, 2, 7, 7, 7, 8, 1, 2, 3, 1, 1, 1, 1, 1, 1];
		total_cost=9537;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 3, 4, 7, 4, 2, 2, 3, 1, 1, 1, 1, 1, 1];
		total_cost=9515;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 3, 7, 7, 4, 2, 2, 3, 1, 1, 1, 1, 1, 1];
		total_cost=9475;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 7, 4, 7, 4, 2, 2, 3, 1, 1, 1, 1, 1, 1];
		total_cost=9371;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 7, 7, 7, 4, 2, 2, 3, 1, 1, 1, 1, 1, 1];
		total_cost=9331;
		---------- 	
		client_at = [1, 3, 1, 7, 1, 7, 3, 2, 7, 7, 7, 8, 2, 2, 3, 1, 1, 1, 1, 1, 1];
		total_cost=9260;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 7, 7, 7, 4, 2, 2, 3, 4, 1, 1, 1, 1, 1];
		total_cost=9259;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 1, 4, 7, 4, 2, 7, 3, 4, 1, 1, 1, 1, 1];
		total_cost=9203;
		---------- 	
		client_at = [7, 6, 1, 7, 1, 7, 3, 1, 3, 6, 6, 6, 1, 7, 3, 6, 1, 1, 1, 1, 1];
		total_cost=9177;
		---------- 	
		client_at = [1, 3, 1, 7, 1, 7, 3, 2, 7, 7, 7, 8, 2, 2, 3, 8, 1, 1, 1, 1, 1];
		total_cost=9176;
		---------- 	
		client_at = [1, 3, 1, 7, 1, 7, 3, 2, 1, 8, 7, 8, 2, 7, 3, 8, 1,

		1, 1, 1];
		total_cost=9140;
		---------- 	
		client_at = [1, 6, 1, 7, 7, 7, 3, 1, 3, 6, 6, 6, 1, 7, 3, 6, 1, 1, 1, 1];
		total_cost=9123;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 7, 4, 7, 4, 1, 2, 3, 4, 3, 1, 1, 1];
		total_cost=9082;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 2, 7, 7, 7, 4, 1, 2, 3, 4, 3, 1, 1, 1];
		total_cost=9042;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 5, 7, 4, 7, 4, 1, 5, 3, 5, 3, 1, 1, 1];
		total_cost=9023;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 5, 7, 7, 7, 4, 1, 5, 3, 5, 3, 1, 1, 1];
		total_cost=8983;
		---------- 	
		client_at = [1, 6, 1, 7, 1, 7, 3, 7, 7, 6, 7, 6, 1, 6, 3, 6, 3, 1, 1, 1];
		total_cost=8939;
		---------- 	
		client_at = [1, 6, 1, 7, 1, 7, 3, 7, 7, 6, 6, 6, 1, 7, 3, 6, 3, 1, 1, 1];
		total_cost=8837;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7, 3, 1, 4, 4, 7, 4, 1, 7, 3, 4, 3, 3, 1, 1];
		total_cost=8807;
		---------- 	
		client_at = [1, 6, 1, 7, 1, 7, 3, 1, 7, 6, 7, 6, 1, 6, 3, 6, 3, 3, 1, 1];
		total_cost=8797;
		---------- 	
		client_at = [1, 6, 1, 7, 1, 7, 3, 1, 7, 6, 6, 6, 1, 7, 3, 6, 3, 3, 1, 1];
		total_cost=8695;
		---------- 	
		client_at = [1, 4, 1, 7, 1, 7,

		3, 1, 1, 4, 7, 4, 1, 7, 3, 4, 3, 3, 4, 1];
		total_cost=8674;
		---------- 	
		client_at = [1, 6, 1, 7, 1, 7, 3, 1, 7, 6, 6, 6, 1, 7, 3, 6, 3, 3, 6, 1];
		total_cost=8655;
		---------- 	
		client_at = [1, 6, 1, 7, 1, 7, 3, 1, 1, 6, 7, 6, 1, 7, 3, 6, 3, 3, 6, 1];
		total_cost=8559;
		---------- 	
		==========
		Finished in 6s 254msec.

Στα αποτελέσματα, παρατηρούμε ότι ο solver βρίσκει διαδοχικές λύσεις με συνεχώς μειούμενο κόστος, εξισορροπώντας το κόστος ανάπτυξης (DeploymentCost) και δικτύωσης (NetworkingCost). Οι πελάτες ανατίθενται σε διακομιστές με βάση τη χωρητικότητα και το κόστος, με συχνές μετακινήσεις για βελτιστοποίηση (π.χ., στο server_set4, ο διακομιστής 7 χρησιμοποιείται συχνά λόγω χαμηλού δικτυακού κόστους).

Οι χρόνοι εκτέλεσης αυξάνονται με το μέγεθος του προβλήματος: από 141 msec (6 πελάτες, 4 διακομιστές) σε 6+ sec (20 πελάτες, 10 διακομιστές). Ο solver αρχικά βρίσκει προσωρινά καλές λύσεις και στη συνέχεια τις βελτιώνει σταδιακά. Σε μικρά προβλήματα, η βέλτιστη λύση βρίσκεται γρήγορα (αρχείο server_set0.dzn), ενώ σε μεγαλύτερα απαιτείται περισσότερος χρόνος (αρχείο server_set4.dzn) ή ευρετικές προσεγγίσεις.

Άσκηση 2: FireTrucks

Η άσκηση αφορά τον χρονοπρογραμματισμό πυροσβεστικών οχημάτων. Συγκεκριμένα, επιλύει το πρόβλημα του προγραμματισμού δύο διαδικασιών για κάθε πυροσβεστικό όχημα:

- Γέμισμα με νερό (filling)
- Εξυπηρέτηση (service)

Ο κώδικας επιχειρεί να βρει το βέλτιστο χρονοδιάγραμμα για αυτές τις εργασίες με στόχο την ελαχιστοποίηση του συνολικού χρόνου (makespan), λαμβάνοντας υπόψη τους ακόλουθους περιορισμούς:

- Υπάρχουν περιορισμένοι πόροι για το γέμισμα (3 αντλίες)
- Η εξυπηρέτηση πρέπει να γίνει μετά το γέμισμα

Οι εργασίες εξυπηρέτησης δεν μπορούν να επικαλύπτονται (απαιτείται αμοιβαίος αποκλεισμός).

Ο κώδικας που χρησιμοποιήθηκε στην επίλυση της άσκησης 1 ήταν

```
include "globals.mzn";

int: no_firetrucks;
set of int: FIRETRUCKS = 1..no_firetrucks;
array[FIRETRUCKS] of int: water_cap;
array[FIRETRUCKS] of int: service_time;

%% Decision Variables - You can change anything, BUT the
names of the variables.
array[FIRETRUCKS] of var int: start_fill;
array[FIRETRUCKS] of var int: start_service;
var int: makespan;
var int: total_service;
%%% Add your code here

% Calculate filling time for each truck
array[FIRETRUCKS] of int: fill_time = [ceil(water_cap[i] /
50.0) | i in FIRETRUCKS];

% Πρέπει να περιορίσουμε τα πλαίσια των μεταβλητών
constraint forall(i in FIRETRUCKS) (
   start_fill[i] >= 0 /\
   start_fill[i] + fill_time[i] <= 1000 /\
   start_service[i] >= start_fill[i] + fill_time[i] /\
   start_service[i] + service_time[i] <= 1000 );
```

```
% Filling constraints - using cumulative (περιορισμός πόρων)
constraint cumulative(start_fill, fill_time, [1 | i in FIRETRUCKS], 3);

% Service constraints - χρησιμοποιούμε disjunctive (αμοιβαίος αποκλεισμός)
constraint disjunctive(start_service, service_time);

% Total service time calculation
constraint total_service = max([start_service[i] + service_time[i] | i in FIRETRUCKS]) - min([start_service[i] | i in FIRETRUCKS]);

% Makespan calculation
constraint makespan = max([start_service[i] + service_time[i] | i in FIRETRUCKS]);

% Objective: Minimize makespan
solve :: seq_search([
   int_search(start_fill, smallest, indomain_min),
   int_search(start_service, smallest, indomain_min)
]) minimize makespan;


output ["start_fill=\(start_fill);\n start_service=\(start_service);\n"]
       ++ ["total_service = \(total_service);\n makespan=\(makespan);\n"]
```

1. Δήλωση Μεταβλητών και Συνόλων
Κώδικας:
int: no_firetrucks;
set of int: FIRETRUCKS = 1..no_firetrucks;
array[FIRETRUCKS] of int: water_cap;
array[FIRETRUCKS] of int: service_time;

no_firetrucks: Αριθμός πυροσβεστικών οχημάτων
FIRETRUCKS: Σύνολο ευρετηρίου για τα οχήματα

Πίνακες Δεδομένων

- **water_cap:** Χωρητικότητα νερού κάθε οχήματος
- **service_time:** Χρόνο εξυπηρέτησης κάθε οχήματος

2. Μεταβλητές Απόφασης
Κώδικας:
```
%% Decision Variables
array[FIRETRUCKS] of var int: start_fill;
array[FIRETRUCKS] of var int: start_service;
var int: makespan;
var int: total_service;
```

start_fill: Πότε ξεκινά το γέμισμα κάθε οχήματος (μεταβλητή απόφασης)
start_service: Πότε ξεκινά η εξυπηρέτηση κάθε οχήματος (μεταβλητή απόφασης)

makespan: Συνολικός χρόνος ολοκλήρωσης όλων των εργασιών (στόχος ελαχιστοποίησης)
total_service: Ο συνολικός χρόνος από την πρώτη μέχρι την τελευταία εξυπηρέτηση

3. Υπολογισμός χρόνου γεμίσματος πυροσβεστικών
Κώδικας:
```
% Calculate filling time for each truck
array[FIRETRUCKS] of int: fill_time = [ceil(water_cap[i] / 50.0) | i in FIRETRUCKS];
```

Υπολογισμός χρόνου γεμίσματος για κάθε όχημα ως την απαιτούμενη ποσότητα νερού διαιρεμένη με το ρυθμό γεμίσματος (50 μονάδες/λεπτό, εδώ kg νερού το λεπτό). Χρήση ceil για στρογγυλοποίηση προς τα πάνω (ακέραιος χρόνος)

3. Περιορισμοί Χρονοδιαγράμματος
Κώδικας:
```
% We need to narrow down the range of variables
constraint forall(i in FIRETRUCKS) (
   start_fill[i] >= 0 /\
   start_fill[i] + fill_time[i] <= 1000 /\
   start_service[i] >= start_fill[i] + fill_time[i] /\
   start_service[i] + service_time[i] <= 1000 );
```

Εφαρμόχονται περιορισμοί για κάθε όχημα:
- Το γέμισμα δεν μπορεί να ξεκινήσει πριν το χρόνο 0
- Το γέμισμα πρέπει να ολοκληρωθεί μέσα σε ένα μεγάλο χρονικό όριο (1000)
- Η εξυπηρέτηση πρέπει να ξεκινήσει μετά το τέλος του γεμίσματος
- Η εξυπηρέτηση πρέπει να ολοκληρωθεί μέσα στο χρονικό όριο

4. Περιορισμός Πόρων Γεμίσματος
Κώδικας
```
% Filling constraints - using cumulative (resource limitation)
constraint cumulative(start_fill, fill_time, [1 | i in FIRETRUCKS], 3);
```

Χρήση του περιορισμού cumulative για την μοντελοποίηση:
- Ύπαρξη μόνο 3 διαθέσιμων αντλιών (τελευταία παράμετρος)
- Κάθε εργασία γεμίσματος καταναλώνει 1 πόρο (πίνακας [1 | i in FIRETRUCKS])
- Οι εργασίες μπορούν να επικαλύπτονται αρκεί να μην ξεπερνούν τους διαθέσιμους πόρους

5. Περιορισμός Εξυπηρέτησης
Κώδικας:
% Service constraints - we use disjunctive (mutual exclusion)
constraint disjunctive(start_service, service_time);

Χρήση του περιορισμού disjunctive για να εξασφαλιστεί ότι:

- Οι εργασίες εξυπηρέτησης δεν επικαλύπτονται (αμοιβαίος αποκλεισμός)
- Κάθε εξυπηρέτηση χρειάζεται αποκλειστική πρόσβαση στον πόρο

6. Υπολογισμός Μετρικών
Κώδικας:
% Total service time calculation
constraint total_service = max([start_service[i] + service_time[i] | i in FIRETRUCKS]) - min([start_service[i] | i in FIRETRUCKS]);
constraint makespan = max([start_service[i] + service_time[i] | i in FIRETRUCKS]);

total_service: Διαφορά μεταξύ τέλους της τελευταίας και αρχής της πρώτης εξυπηρέτησης
makespan: Ο χρόνος ολοκλήρωσης της τελευταίας εξυπηρέτησης (στόχος βελτιστοποίησης)

7. Στρατηγική Επίλυσης
Κώδικας:
% Objective: Minimize makespan
solve :: seq_search([
  int_search(start_fill, smallest, indomain_min),
  int_search(start_service, smallest, indomain_min)
]) minimize makespan;

Ορίζεται η στρατηγική αναζήτησης:

- Πρώτα αναζητώνται τιμές για τα start_fill (μικρότερες τιμές πρώτα)
- Μετά αναζητώνται τιμές για τα start_service (επίσης μικρότερες τιμές πρώτα)
- Στόχος είναι η ελαχιστοποίηση του makespan

8. Εκτύπωση Αποτελεσμάτων
Κώδικας:
output [" start_fill=\(start_fill);\n start_service=\(start_service);\n"]
    ++ [" total_service = \(total_service);\n makespan=\(makespan);\n"]

Εκτύπωση των χρονοδιαγραμμάτων γεμίσματος και εξυπηρέτησης
Εμφάνιση total_service και makespan

Παραδείγματα εκτέλεσης

FireData1.dzn	**FireData2.dzn**
no_firetrucks = 6; water_cap =	no_firetrucks = 8; water_cap =

[1200,900,1500,800,1300,600]; service_time = [15,10,20,10,5,20];	[1200,700,1500,500,1300,400,300,200]; service_time = [15,10,20,10,15,20,5,5];

Running FireTrucks.mzn, FireData1.dzn
385msec
 	
start_fill=[0, 0, 0, 18, 24, 30];
start_service=[28, 18, 43, 63, 73, 78];
total_service = 80;
makespan=98;
---------- 	
start_fill=[0, 0, 16, 0, 18, 24];
start_service=[26, 41, 51, 16, 71, 76];
total_service = 80;
makespan=96;
---------- 	
start_fill=[0, 0, 12, 18, 24, 0];
start_service=[32, 47, 57, 77, 87, 12];
total_service = 80;
makespan=92;
---------- 	
==========
Finished in 385msec.
	Running FireTrucks.mzn, FireData2.dzn
	386msec
	
	start_fill=[0, 0, 0, 14, 24, 24, 30, 32];
	start_service=[24, 14, 39, 59, 69, 84, 104, 109];
	total_service = 100;
	makespan=114;
	---------- 	
	start_fill=[0, 0, 10, 0, 14, 24, 32, 38];
	start_service=[30, 20, 45, 10, 65, 80, 100, 105];
	total_service = 100;
	makespan=110;
	---------- 	
	start_fill=[0, 0, 8, 14, 24, 0, 24, 30];
	start_service=[28, 43, 53, 73, 83, 8, 98, 103];
	total_service = 100;
	makespan=108;
	---------- 	
	start_fill=[0, 0, 10, 14, 24, 24, 0, 6];
	start_service=[26, 16, 41, 61, 71, 86, 6, 11];
	total_service = 100;
	makespan=106;
	---------- 	
	start_fill=[0, 0, 10, 14, 24, 24, 4, 0];
	start_service=[25, 15, 40, 60, 70, 85, 10, 4];
	total_service = 101;
	makespan=105;
	---------- 	
	start_fill=[0, 4, 8, 18, 24, 0, 28, 0];
	start_service=[29, 44, 54, 74, 84, 9, 99, 4];
	total_service = 100;
	makespan=104;
	---------- 	
	==========
	Finished in 386msec.

Τα αποτελέσματα δείχνουν πολλαπλές βέλτιστες λύσεις με παρόμοιο makespan αλλά διαφορετική χρονική διάταξη, λόγω των περιορισμών του προβλήματος. Στο πρώτο σετ δεδομένων (6 οχήματα), το βέλτιστο makespan=92 επιτυγχάνεται όταν το γέμισμα και η εξυπηρέτηση οργανώνονται έτσι ώστε να μην υπάρχουν μεγάλες καθυστερήσεις μεταξύ των εργασιών. Παρατηρούμε ότι τα οχήματα με μεγάλο χρόνο γεμίσματος (π.χ., 1500 λίτρα → 30 λεπτά) ή εξυπηρέτησης (20 λεπτά) επηρεάζουν σημαντικά το χρονοδιάγραμμα, καθώς απαιτούν περισσότερους πόρους ή δημιουργούν μπλοκαρίσματα. Η ύπαρξη τριών αντιλήψεων επιτρέπει κάποια παραλληλία στο γέμισμα, αλλά ο disjunctive περιορισμός στις

εξυπηρετήσεις εξασφαλίζει ότι καμία δεν επικαλύπτεται, οδηγώντας σε έναν συγκεκριμένο "πυρήνα" χρονικών περιορισμών.

Στο δεύτερο σετ δεδομένων (8 οχήματα), το makespan=104 είναι μεγαλύτερο λόγω του αυξημένου αριθμού εργασιών, αλλά η παρουσία μικρών εξυπηρετήσεων (5-10 λεπτά) επιτρέπει μεγαλύτερη ευελιξία στη βελτιστοποίηση. Ο solver βρίσκει λύσεις όπου τα μικρά tasks τοποθετούνται σε κενά του χρονοδιαγράμματος, μειώνοντας το συνολικό χρόνο ολοκλήρωσης. Παράλληλα, το total_service παραμένει σταθερό (~100 λεπτά), αφού καθορίζεται από τη συνολική διάρκεια των εξυπηρετήσεων και όχι από τη σειρά εκτέλεσής τους. Η διαφορά μεταξύ των λύσεων οφείλεται στη στρατηγική αναζήτησης (smallest, indomain_min), που δοκιμάζει διαφορετικούς συνδυασμούς χωρίς να αλλάζει ουσιαστικά την απόδοση.

Προβλήματα:
Ο κώδικας που υλοποίησα αρχικά ήταν ο:

```
% Ensure at most 3 firetrucks are filling water at the
same time
constraint forall(t in 0..1000)(
   sum([bool2int(start_fill[i] <= t /\ t < start_fill[i] +
ceil(water_cap[i] / 50.0)) | i in FIRETRUCKS]) <= 3 );

% Constraints for filling water
constraint forall(i in FIRETRUCKS)(
   start_fill[i] >= 0 /\
   start_fill[i] + ceil(water_cap[i] / 50.0) <= 1000 );

% Constraints for servicing
constraint forall(i in FIRETRUCKS)(
   start_service[i] >= start_fill[i] + ceil(water_cap[i] /
50.0) /\
   start_service[i] + service_time[i] <= 1000 );

% Ensure no overlap in servicing (single mobile crew)
constraint forall(i, j in FIRETRUCKS where i != j)(
            (start_service[i]    +    service_time[i]    <=
start_service[j]) \/
            (start_service[j]    +    service_time[j]    <=
start_service[i]) );

% Total service time calculation
```

```
constraint total_service = max([start_service[i] + service_time[i] | i in FIRETRUCKS]) - min([start_service[i] | i in FIRETRUCKS]);

% Makespan calculation
constraint makespan = max([start_service[i] + service_time[i] | i in FIRETRUCKS]);

% Objective: Minimize makespan
solve minimize makespan;
```

Ο αρχικός κώδικας υλοποιούσε τους περιορισμούς με τρόπο υπολογιστικά ακριβό, χρησιμοποιώντας έναν περιορισμό με γεννήτρια που ελέγχει κάθε χρονική στιγμή από 0 έως 1000 ξεχωριστά για να διασφαλίσει ότι δεν γεμίζουν ταυτόχρονα περισσότερα από 3 πυροσβεστικά οχήματα. Επίσης, η συνθήκη αμοιβαίου αποκλεισμού των εξυπηρετήσεων υλοποιούνταν με ζεύγη περιορισμών για όλα τα ζεύγη οχημάτων, προσθέτοντας σημαντική υπολογιστική φόρτωση. Αντίθετα, ο τελικός κώδικας αξιοποιεί εξειδικευμένους global constraints, συγκεκριμένα τον **cumulative** για τα γεμίσματα και τον **disjunctive** για τις εξυπηρετήσεις, οι οποίοι εφαρμόζουν βελτιστοποιημένους αλγορίθμους για τον ίδιο σκοπό.

Η σημαντική επιβράδυνση στον αρχικό κώδικα, ιδιαίτερα στο δεύτερο σετ δεδομένων με 8 οχήματα, οφειλόταν κυρίως στην αναποτελεσματική μοντελοποίηση των περιορισμών. Ο έλεγχος των ταυτόχρονων γεμισμάτων με 1001 ξεχωριστούς ελέγχους και η τετραγωνική πολυπλοκότητα των περιορισμών μη επικάλυψης δημιουργούσαν μεγάλο υπολογιστικό κόστος. Επιπλέον, η απουσία συγκεκριμένης στρατηγικής αναζήτησης οδηγούσε τον solver σε λιγότερο αποτελεσματική εξερεύνηση του χώρου των πιθανών λύσεων και σημαντική καθυστέρηση στην εύρεση βέλτιστης λύσης.

Οι κύριες βελτιώσεις στον τελικό κώδικα περιλάμβαναν την αντικατάσταση των απλών περιορισμών με global constraints που υλοποιούν ειδικευμένους αλγορίθμους, την προσθήκη στρατηγικής αναζήτησης που καθοδηγεί τον solver να εστιάσει πρώτα στις μεταβλητές γεμίσματος και έπειτα στις εξυπηρετήσεις, καθώς και βελτιστοποιήσεις όπως ο προϋπολογισμός του χρόνου γεμίσματος για κάθε όχημα. Αυτές οι αλλαγές οδήγησαν σε σημαντική μείωση του χρόνου επίλυσης και βελτίωση της ποιότητας των λύσεων, με το σύστημα να συγκλίνει πλέον σε λιγότερο από 1 δευτερόλεπτο αντί για δεκάδες που απαιτούσε ο αρχικός κώδικας.