ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Έλεγχος ορθότητας και σύγκριση απόδοσης αλγορίθμων χρησιμοποιώντας Python

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ε. Παλαιοκρασσάς

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια ΕΜΠ

Αθήνα, Σεπτέμβριος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Έλεγχος της ορθότητας και σύγκριση απόδοσης αλγορίθμων χρησιμοποιώντας Python

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ε. Παλαιοκρασσάς

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τη 2013

.....	.....	.....
Θεοδώρα Βαρβαρίγου Καθηγήτρια ΕΜΠ	Βασίλειος Λούμος Καθηγητής ΕΜΠ	Συμεών Παπαβασιλείου Αναπληρωτής Καθηγητής ΕΜΠ

Αθήνα, 2013

.....

Γεώργιος Ε. Παλαιοκρασσάς

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της διπλωματικής αυτής εργασίας, είναι η μελέτη και σύγκριση μεθόδων εύρεσης βέλτιστης ανάθεσης φυσικών πόρων σε έναν Πάροχο Υποδομής ενός διακομιστή νέφους (cloud server).

Οι πόροι του Παρόχου Υποδομής είναι οι εξυπηρετητές που συνδέονται συνήθως σε ένα ή περισσότερα ιδιωτικά δίκτυα. Το αποτέλεσμα του ελέγχου ανάθεσης είναι ένα σύνολο αποδεκτών υπηρεσιών, η ανατιθέμενη υπολογιστική ισχύς και χωρητικότητα δικτύου για τα συστατικά που αποτελούν τις υπηρεσίες. Στόχος είναι η μεγιστοποίηση του κέρδους του Παρόχου Υποδομής από τον Πάροχο Υπηρεσιών για την εξυπηρέτηση υπηρεσιών.

Το γενικό πρόβλημα που παρουσιάζεται, μοντελοποιήθηκε κάνοντας χρήση του συστήματος GAMS (General Algebraic Modeling System) και της γλώσσας προγραμματισμού Python, τα χαρακτηριστικά των οποίων θα εξεταστούν αναλυτικά και ξεχωριστά για το κάθε ένα και ακολούθως θα γίνει σύγκριση μεταξύ τους. Επίσης θα γίνει εκτενής ανάλυση των αποτελεσμάτων των μοντέλων βελτιστοποίησης καθώς και της ορθότητας και απόδοσης αυτών κάτω από ρεαλιστικές συνθήκες.

Λέξεις Κλειδιά: GAMS, Python, Cloud Computing, Testing, Profiling, Εικονική Μηχανή (Virtual Machine- VM), Έλεγχος Ανάθεσης, Βέλτιστη Ανάθεση

Abstract

The main purpose of this project is the study of methods which compute the optimum allocation of an Infrastructure Provider's natural resources into a cloud server.

The resources of the Infrastructure Provider are the hosts that are connected to one or more private networks. The output of the admission control test is the set of accepted services, the allocated computing and networking capacity for the components that comprise the services. The model's objective is maximizing the profit of the Infrastructure Provider from the Service Provider for hosting services.

The presented problem has been implemented using GAMS (General Algebraic Modeling System) and the programming language Python, the features of which are separately examined and then are compared. An extensive analysis of the results of the implemented models and their correctness and efficiency, is performed under realistic settings .

Key words: GAMS, Python, Cloud Computing, Testing, Profiling, Virtual Machine, Admission Control, Optimum allocation

Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαίτερα την καθηγήτρια κ. Θεοδώρα Βαρβαρίγου που μου έδωσε τη δυνατότητα να ασχοληθώ με το ενδιαφέρον αυτό θέμα καθώς και για την υποστήριξη που μου παρείχε κατά τη διάρκεια εκπόνησης της εργασίας.

Επίσης θα ήθελα να εκφράσω τις ευχαριστίες μου στον υποψήφιο διδάκτορα κ. Κωνσταντίνο Ψύχα και τη διδάκτορα κ. Κλεοπάτρα Κωνσταντέλη που με βοήθησαν σε όλα τα στάδια της εργασίας.

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου για τη συμπαράσταση σε όλα τα χρόνια των σπουδών μου στο ΕΜΠ.

Περιεχόμενα

1)Εισαγωγή 15
2)Λεπτομέρειες για την υλοποίηση σε Python και GAMS 18
2.1)Συμβολισμοί 18
2.1.1) Τοπολογία Πόρων 18
2.1.2) Συμβολισμοί Υπηρεσίας 18
2.1.3) Service Level Agreement Model 19
2.1.4) Περιοριστικοί Κανόνες 19
2.2)Περιγραφή μοντέλου Python 20
2.2.1) .Περιορισμοί προβλήματος 20
2.2.2) Πληροφορίες των συνδυασμών 21
2.2.3) Εύρεση Αναθέσεων 21
2.2.4) Κόστος της ανάθεσης 24
2.2.5) Υπολογισμός Αποτελέσματος 24
2.3) Περιγραφή μοντέλου Gams 25
2.4) Παράμετροι Προβλήματος 26
3)Δυνατότητες που παρέχει η Python 28
3.1) Πιθανότητες και Κατανομές [6] [7] [8] 28
3.2) Παραγωγή random αριθμών στην Python 31
3.3) testing και ορθότητα 32
3.4) Python Profiler 33
4)Αποτελέσματα 34
4.1)Σύντομη παρουσίαση των περιπτώσεων που εξετάζονται 36
4.2) Πρώτη Κατηγορία Περιπτώσεων 38
4.2.1) Αυξανόμενος Αριθμός Συστατικών 38
4.2.2)Αυξανόμενος Αριθμός Εξυπηρετητών 41
4.2.3)Αυξανόμενος Αριθμός Υπηρεσιών 44
4.2.4)Αυξανόμενος Αριθμός Δικτύων 46
4.3) Δεύτερη Κατηγορία Περιπτώσεων 50
4.3.1) Αυξανόμενος Αριθμός Συστατικών 51
4.3.2)Αυξανόμενος Αριθμός Εξυπηρετητών 53
4.3.3) Αυξανόμενος Αριθμός Υπηρεσιών 55

4.3.4)Αυξανόμενος Αριθμός Δικτύων 58
4.4) Τρίτη Κατηγορία Περιπτώσεων 61
Αυξανόμενος Αριθμός Συστατικών 61
Αυξανόμενος Αριθμός Εξυπηρετητών 64
Αυξανόμενος Αριθμός Υπηρεσιών 67
Αυξανόμενος Αριθμός Δικτύων 70
4.5) Τέταρτη κατηγορία περιπτώσεων 73
Αυξανόμενος Αριθμός Συστατικών 74
Αυξανόμενος Αριθμός Εξυπηρετητών 75
Αυξανόμενος Αριθμός Υπηρεσιών 76
Αυξανόμενος Αριθμός Δικτύων 77
4.6) Πέμπτη κατηγορία περιπτώσεων με πολλές εκτελέσεις 78
4.6.1)Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό εξυπηρετών 79
Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό υπηρεσιών 84
Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό συστατικών 87
Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό δικτύων 90
4.7) Έκτη κατηγορία περιπτώσεων – προφίλ 94
5) Συμπεράσματα 102
6) Γλωσσάριο 103
7) Βιβλιογραφία 104
8) Παράρτημα 106

Ευρετήριο Εικόνων

Εικόνα 1 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό συστατικών 39
Εικόνα 2 : . Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό συστατικών 40
Εικόνα 3 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό συστατικών 40
Εικόνα 4 :Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών 40
Εικόνα 5 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών 41
Εικόνα 6 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών 42
Εικόνα 7 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό εξυπηρετητών 42
Εικόνα 8 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών 43
Εικόνα 9 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών 43
Εικόνα 10 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python 43
Εικόνα 11 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό υπηρεσιών 45
Εικόνα 12 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό υπηρεσιών 45
Εικόνα 13 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών 45
Εικόνα 14 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών 46
Εικόνα 15 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό υπηρεσιών 46
Εικόνα 16 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό δικτύων 47
Εικόνα 17 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό δικτύων 48
Εικόνα 18 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό δικτύων 48
Εικόνα 19 : . Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό δικτύων 48
Εικόνα 20 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό δικτύων 49

Εικόνα 21 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό συστατικών 51
Εικόνα 22 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό συστατικών 52
Εικόνα 23 : . Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό συστατικών 52
Εικόνα 24 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών 52
Εικόνα 25 : . Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών 53
Εικόνα 26 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών 54
Εικόνα 27: Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό εξυπηρετητών 54
Εικόνα 28 : Χρόνος CPU για τον υπολογισμό του ορίου της Python 54
Εικόνα 29 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών 55
Εικόνα 30 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό εξυπηρετητών 55
Εικόνα 31 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό υπηρεσιών 56
Εικόνα 32 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό υπηρεσιών 56
Εικόνα 33 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών 57
Εικόνα 34 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών 57
Εικόνα 35 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό υπηρεσιών 57
Εικόνα 36 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό δικτύων 58
Εικόνα 37 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό δικτύων 59
Εικόνα 38 : Χρόνος CPU για τον υπολογισμό του ορίου της Python για αυξανόμενο αριθμό Δικτύων 59
Εικόνα 39 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό δικτύων 59
Εικόνα 40 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό δικτύων 60
Εικόνα 41 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό συστατικών 62
Εικόνα 42 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό συστατικών 63
Εικόνα 43 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό συστατικών 63

Εικόνα 44 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών..... 63
Εικόνα 45 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών..... 64
Εικόνα 46 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών..... 65
Εικόνα 47 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό εξυπηρετητών..... 65
Εικόνα 48: Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών..... 66
Εικόνα 49 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών..... 66
Εικόνα 50 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό εξυπηρετητών..... 66
Εικόνα 51 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON..... 67
Εικόνα 52 :Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python..... 68
Εικόνα 53: Χρόνος CPU για τον υπολογισμό του ορίου της Python..... 68
Εικόνα 54: Λόγος αποτελέσματος βέλτιστης λύσης από το BARON προ το αποτέλεσμα της αντικειμενικής συνάρτησης της Python..... 68
Εικόνα 55 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python..... 69
Εικόνα 56 : Λόγος του χρόνοy υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python..... 69
Εικόνα 57: Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON..... 70
Εικόνα 58 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python..... 71
Εικόνα 59: Χρόνος CPU για τον υπολογισμό του ορίου της Python..... 71
Εικόνα 60: Λόγος αποτελέσματος βέλτιστης λύσης από το BARON προ το αποτέλεσμα της αντικειμενικής συνάρτησης της Python..... 71
Εικόνα 61: Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python..... 72
Εικόνα 62: Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python..... 72
Εικόνα 63 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών..... 74
Εικόνα 64 : Λόγος των χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών..... 75
Εικόνα 65 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών..... 75
Εικόνα 66 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό εξυπηρετητών..... 75
Εικόνα 67 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών..... 76
Εικόνα 68 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό υπηρεσιών..... 76

Εικόνα 69 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό δικτύων 77
Εικόνα 70 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό δικτύων 77
Εικόνα 71 : Μέσος χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό εξυπηρετητών 80
Εικόνα 72 : Μέσος χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών 81
Εικόνα 73 : Μέσος λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python 82
Εικόνα 74 : μέσος όρος χρόνου BARON προς χρόνο ορίου Python 83
Εικόνα 75 : Μέσος χρόνος Python με αυξανόμενο αριθμό υπηρεσιών 85
Εικόνα 76 : Μέσος Χρόνος BARON με αυξανόμενο αριθμό υπηρεσιών 85
Εικόνα 77 : Μέσος λόγος GAMS προς χρόνο Python με αυξανόμενο αριθμό υπηρεσιών 86
Εικόνα 78 : Μέσος Χρόνος BARON προς χρόνο ορίου Python 87
Εικόνα 79 : Μέσος Χρόνος Python με αυξανόμενο αριθμό συστατικών 88
Εικόνα 80 : Μέσος χρόνος BARON με αυξανόμενο αριθμό συστατικών 89
Εικόνα 81 : μέσος χρόνος GAMS προς χρόνο ορίου της Python 89
Εικόνα 82 : Μέσος χρόνος GAMS προς χρόνο Python 90
Εικόνα 83 : Μέσος Χρόνος Python με αυξανόμενο αριθμό δικτύων 91
Εικόνα 84 : Μέσος : Χρόνος του BARON με αυξανόμενο αριθμό δικτύων 92
Εικόνα 85 : Μέσος χρόνος GAMS προς το χρόνο Python 93
Εικόνα 86: Μέσος χρόνος GAMS προς το χρόνο του ορίου της Python 93
Εικόνα 87 : Μέσος χρόνος Python με αυξανόμενο αριθμό εξυπηρετητών 96
Εικόνα 88 : Μέσος χρόνος GAMS με αυξανόμενο αριθμό εξυπηρετητών 96
Εικόνα 89 : Μέσος Χρόνος Python με αυξανόμενο αριθμό εξυπηρετητών 99
Εικόνα 90 : Μέσος χρόνος GAMS με αυξανόμενο αριθμό εξυπηρετητών 99

Ευρετήριο Πινάκων

Πίνακας 1 : Συντομεύσεις ποσοτήτων που εξετάζονται 35
Πίνακας 2 : Στατιστικά Στοιχεία για 30 εκτελέσεις με 10 εξυπηρετητές 79
Πίνακας 3: Χρόνος της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό εξυπηρετητών ... 80
Πίνακας 4 : Χρόνος BARON με αυξανόμενο αριθμό εξυπηρετητών 81
Πίνακας 5: Λόγος χρόνου του BARON προς τον χρόνο της βέλτιστης της Python με αυξανόμενο αριθμό εξυπηρετητών 82
Πίνακας 6 : Λόγος χρόνου BARON προς χρόνο ορίου Python 83
Πίνακας 7 : Χρόνος Python με αυξανόμενο αριθμό υπηρεσιών 84
Πίνακας 8 : Χρόνος BARON με αυξανόμενο αριθμό υπηρεσιών 85
Πίνακας 9 : Λόγος χρόνου BARON προς χρόνο Python με αυξανόμενο αριθμό υπηρεσιών 86
Πίνακας 10 : Χρόνος BARON προς χρόνο ορίου Python 86
Πίνακας 11 : Χρόνος Python με αυξανόμενο αριθμό συστατικών 88
Πίνακας 12 : Χρόνος BARON με αυξανόμενο αριθμό συστατικών 88
Πίνακας 13 : χρόνος GAMS προς χρόνο ορίου της Python 89
Πίνακας 14 : Χρόνος GAMS προς χρόνο Python 90
Πίνακας 15 : Χρόνος Python με αυξανόμενο αριθμό δικτύων 91
Πίνακας 16 : Χρόνος του BARON με αυξανόμενο αριθμό δικτύων 92
Πίνακας 17 : Χρόνος GAMS προς το χρόνο Python 92
Πίνακας 18 : Χρόνος GAMS προς το χρόνο του ορίου της Python 93
Πίνακας 19 : Χρόνος GAMS/ Χρόνος Python προφίλ(6,2,4) 98
Πίνακας 20 : Χρόνος GAMS/ Χρόνος Python προφίλ(4,2,10) 98
Πίνακας 21 : Χρόνος GAMS/ Χρόνος Python προφίλ(1,10,3) 98
Πίνακας 22 : Χρόνος GAMS/ Χρόνος python προφίλ(10,4,5) 99
Πίνακας 23 : Ποσοστό (%) από τις 30 περιπτώσεις ανά προφίλ και αριθμό εξυπηρετητών όπου ο χρόνος GAMS ήταν μεγαλύτερος των 120 δευτερολέπτων 100
Πίνακας 24 Ποσοστό (%) από τις 30 περιπτώσεις ανά προφίλ και αριθμό εξυπηρετητών όπου ο χρόνος GAMS ήταν μεγαλύτερος των 900 δευτερολέπτων 100
Πίνακας 25 : μέσος όρος του λόγου του αποτελέσματος GAMS προς το αποτελέσματα της Python ανά προφίλ και αριθμό εξυπηρετητών 101

1)Εισαγωγή

Σε μια εικονικοποιημένη υποδομή (virtualized infrastructure) όπως το υπολογιστικό νέφος (computing Cloud) θεωρούμε κάθε υπηρεσία ως ένα σύνολο από εικονικοποιημένα συστατικά (virtualized components) δηλαδή εικονικές μηχανές (Virtual Machines) που ενεργοποιούνται ανάλογα με το μοτίβο ροής εργασίας τους (workflow pattern) κάθε φορά που ένα αίτημα φτάνει από τους τελικούς χρήστες [1]. Η χρήση εικονικοποιημένων τεχνικών επιτρέπει την αδιάκοπη ανάθεση (allocation) του κάθε συστατικού της κατανεμημένης (distributed) υπηρεσίας μέσα στο νέφος. Καθιστά επίσης ευκολότερη την διαδικασία της οριζόντιας ελαστικότητας (horizontal elasticity) δηλαδή προσθαφαίρεση επιπλέον αντιγράφων των εικονικών μηχανών (duplicate Virtual Machines) για κάθε συστατικό (component) κατά τη διάρκεια της εκτέλεσης για να διατηρήσουμε ένα συγκεκριμένο επίπεδο απόδοσης για ολόκληρη την υπηρεσία όταν υπάρχουν διακυμάνσεις στο φόρτο εργασίας .

Ένα γενικό πρόβλημα που προκύπτει για τον Πάροχος Υποδομής (Infrastructure Provider) κατά τη χρήση κέντρου δεδομένων αφορά την εύρεση της βέλτιστης ανάθεσης των πόρων, δεδομένου ενός συνόλου υπηρεσιών (services). Οι πόροι του Παρόχου Υποδομής είναι οι εξυπηρετητές (hosts) που συνδέονται συνήθως σε ένα ή περισσότερα ιδιωτικά δίκτυα (networks).

Μια υπηρεσία δίνεται από έναν Πάροχο Υπηρεσίας (Service Provider) και περιγράφεται από μια ροή εργασίας (workflow) που περιέχει προγράμματα-συστατικά (components) που εκπληρώνουν συγκεκριμένες εργασίες. Ο Πάροχος Υπηρεσίας έχει συγκεκριμένες απαιτήσεις για την υπηρεσία και τα συστατικά της και πρόκειται να πληρώσει τον Πάροχο Υποδομής μόνο αν οι απαιτήσεις αυτές ικανοποιηθούν.

Η βασική απαίτηση για μια υπηρεσία είναι ότι πρέπει να ικανοποιεί ένα περιορισμό διαθεσιμότητας (availability). Αυτός ο περιορισμός περιγράφει το ελάχιστο ποσοστό χρόνου λειτουργίας (uptime) δηλαδή αν η ελάχιστη διαθεσιμότητα είναι 99% τότε η υπηρεσία θα πρέπει να αποκρίνεται το 99% του χρόνου.

Η διαθεσιμότητα αυξάνεται αντιγράφοντας όλα ή μερικά από τα συστατικά των υπηρεσιών, κάτι που είναι πολύ εύκολο με τη χρήση ενός Διαχειριστή Εικονικής Μηχανής (Virtual Machine Manager). Αυτό σημαίνει ότι κάθε συστατικό περιέχεται σε μια Εικονική Μηχανή που έχει υπολογιστικές απαιτήσεις (computing requirements) και απαιτήσεις δικτύου (network requirements) (και πιθανώς απαιτήσεις μνήμης και αποθήκευσης

αλλά δεν εξετάζονται). Υποθέτουμε ότι μια Εικονική Μηχανή (Virtual Machine) απαιτεί υπολογιστικούς πυρήνες (computing cores) και MB εύρους δικτύου (bandwidth).

Κατά τη διάρκεια του ελέγχου ανάθεσης (admission control) και προκειμένου να υπάρξουν ισχυρές εγγυήσεις απόδοσης ο πάροχος υποδομής πρέπει να λάβει υπ' όψιν όχι μόνο τις βασικές υπολογιστικές απαιτήσεις και απαιτήσεις δικτύου αλλά και τις ελαστικές απαιτήσεις οι οποίες σε πολλές περιπτώσεις μπορεί να είναι μεγάλες συγκριτικά με τις βασικές. Για παράδειγμα, για μια υπηρεσία με υψηλή διακύμανση (variation) στον αριθμό των χρηστών ο αριθμός των VM's που πρέπει να προστεθούν κατά τη διάρκεια της εκτέλεσης (at runtime) μπορεί να είναι πολλαπλάσιος του αριθμού των βασικών. Επομένως το ποσό των ελαστικών απαιτήσεων παίζει ένα σημαντικό ρόλο στο σύνολο των απαιτήσεων και επομένως στο κόστος εξυπηρέτησης μιας υπηρεσίας και ο πάροχος υποδομής έχει μεγάλο συμφέρον στο να εξετάσει το ενδεχόμενο μείωσης των πόρων που πρέπει να δεσμευθούν εκ των προτέρων (booked) για λόγους ελαστικότητας όταν αποδεχόμαστε την υπηρεσία. Ταυτόχρονα μια τέτοια προσέγγιση μπορεί να αυξήσει την πιθανότητα απόκλισης από το συμφωνηθέν επίπεδο ποιότητας υπηρεσίας (quality of service - QoS)

Όταν μια Εικονική Μηχανή ανατίθεται σε έναν εξυπηρετητή χρησιμοποιεί τους υπολογιστικούς πυρήνες του, αν είναι επαρκώς διαθέσιμοι, και το εύρος δικτύου του. Επίσης κάθε συστατικό υπηρεσίας απαιτεί ένα σταθερό ποσό εύρους δικτύου σε κάθε δίκτυο που ανατίθεται, ανεξάρτητα από το πόσες εικονικές μηχανές ανατίθενται σε αυτό. Φυσικά σε περίπτωση που το εύρος δικτύου ή οι υπολογιστικοί πυρήνες δεν είναι επαρκείς η Εικονική Μηχανή δεν μπορεί να ανατεθεί.

Η διαθεσιμότητα μιας υπηρεσίας μπορεί να ελεγχθεί στην πράξη μετά την ανάθεση αλλά υποθέτουμε ότι εφόσον ο αριθμός των χρηστών είναι (περισσότερο ή λιγότερο) γνωστός, ο αριθμός των η Εικονικών Μηχανών που δίνουν 0 και μέγιστο (100%) διαθεσιμότητα είναι επίσης γνωστός για κάθε συστατικό. Η διαθεσιμότητα για κάθε άλλο αριθμό Εικονικών Μηχανών είναι γραμμικά συνδεδεμένη με αυτό τον αριθμό. Για παράδειγμα αν 4 Εικονικές Μηχανές δίνουν 0% διαθεσιμότητα και 8 δίνουν 100% τότε 6 δίνουν 50%. Πολλαπλασιάζοντας τις διαθεσιμότητες όλων των συστατικών μιας υπηρεσίας λαμβάνουμε τη διαθεσιμότητα της.

Αν ο Πάροχος Υποδομής αναθέτει αρκετές Εικονικές Μηχανές ώστε να ικανοποιείται η απαίτηση του Παρόχου Υπηρεσίας, ο τελευταίος πληρώνει τον Πάροχο Υποδομής. Είναι επίσης πιθανό ο Πάροχος Υποδομής να μην μπορεί να αναθέσει Εικονικές Μηχανές για όλα τα συστατικά αλλά για

κάποια από αυτά. Σε αυτήν την περίπτωση ο Πάροχος Υπηρεσίας πληρώνει ένα κλάσμα της αρχικής τιμής.

Παρ' όλα αυτά ο Πάροχος Υποδομής πληρώνει ένα ποσόν για κάθε χρησιμοποιούμενο υπολογιστικό πυρήνα ενός εξυπηρετητή και επίσης ένα ποσόν για την χρήση κάθε εξυπηρετητή (το κόστος ενός εξυπηρετητή σε κατάσταση αναμονής). Επιπρόσθετα κάθε Πάροχος Υπηρεσίας έχει ένα ποσόν εμπιστοσύνης (trust) που περιγράφεται από ένα ποσοστό και αντιπροσωπεύει την πιθανότητα ότι ο Πάροχος Υπηρεσίας θα πληρώσει. Αυτό το ποσοστό συνεπώς πολλαπλασιάζει το ποσόν του αναμενόμενου κέρδους από την υπηρεσία. Η ανάθεση είναι καλύτερη όταν μεγιστοποιεί το κέρδος του Παρόχου Υποδομής αν λάβουμε υπ' όψιν το κόστος της ανάθεσης.

Τέλος, εκτός από την απαίτηση διαθεσιμότητας ο Πάροχος Υπηρεσίας μπορεί να έχει επιπρόσθετες απαιτήσεις όπως η τοποθέτηση όλων των Εικονικών Μηχανών μιας υπηρεσίας σε ένα δίκτυο ή άλλες όμοιες.

2)Λεπτομέρειες για την υλοποίηση σε Python και GAMS

Για να επιλύσουμε το πρόβλημα χρειάζεται κάποια μοντελοποίηση στον Η/Υ. Το πρόβλημα έχει διάφορες παραμέτρους που περιγράφονται παρακάτω. Συγκρίναμε δύο μεθόδους επίλυσης του προβλήματος που βασίζονται στη συγκεκριμένη μοντελοποίηση, οι οποίες έχουν χρησιμοποιηθεί και στο project OPTIMIS[2] και έχουν λειτουργήσει επιτυχώς. Το πρόβλημα λύνει στο OPTIMIS το πρόβλημα ανάθεσης εργασιών. Θα εξετάσουμε τα χαρακτηριστικά της κάθε μιας και θα κάνουμε τη σύγκριση τους χρησιμοποιώντας πειραματικά δεδομένα σε επόμενη ενότητα.

2.1)Συμβολισμοί

2.1.1) Τοπολογία Πόρων

Οι πόροι του παρόχου υποδομής μπορούν γενικότερα να θεωρηθούν ως μια διασύνδεση (πιθανώς ετερογενών) δικτύων που συνδέουν (πιθανώς ετερογενείς) υπολογιστικούς κόμβους. Παραδείγματος χάριν πολλά LANs που εμπεριέχουν υπολογιστικούς κόμβους πολλών επεξεργαστών και είναι διασυνδεδεμένα με ένα ή περισσότερα WANs. Η τοπολογία δικτύου λοιπόν χαρακτηρίζεται από τα ακόλουθα στοιχεία:

1) Ένα σύνολο υπολογιστικών κόμβων ή εξυπηρετητών: $H=\{1,\dots,N_H\}$. Κάθε εξυπηρετητής $j \in H$ χαρακτηρίζεται από μια διαθέσιμη υπολογιστική ισχύ $\mathbf{U_j \in R^+}$ που είναι η τιμή μιας μετρικής ενός δοσμένου συστήματος
2) Ένα σύνολο από διαθέσιμα δίκτυα $N= \{1,\dots,N_N\}$. Κάθε δίκτυο $n \in N$ χαρακτηρίζεται από ένα μέγιστο εύρος δικτύου $\mathbf{U_n \in R^+}$ που εκφράζεται σε bytes/sec
3) Η πληροφορία σχετικά με την τοπολογία δικτύου που προσδιορίζει ποιοι εξυπηρετητές $H_n \subset H$ είναι συνδεδεμένοι σε κάθε δίκτυο $n \in N$.

2.1.2) Συμβολισμοί Υπηρεσίας

Οι ακόλουθοι συμβολισμοί χρησιμοποιούνται για να αναφερόμαστε στις υπηρεσίες:

1) Ένα σύνολο από στιγμιότυπα υπηρεσιών: $S=\{1,\dots,N_S\}$
2) Κάθε υπηρεσία $s \in S$ είναι ροή εργασίας από $m^{(s)}$ συστατικά ενσωματωμένα μέσα στις Εικονικές Μηχανές: $S^s = \{1,\dots,m^s\}$ που εκφράζονται επίσης ως $(\xi_1^s,\dots, \xi_{m^s}^s)$.

Κάθε component $\xi_i^s \in S^s$ χαρακτηρίζεται από τις ακόλουθες παραμέτρους:

1) Ελάχιστη **βασική υπολογιστική ισχύς** $\boldsymbol{\theta_i^s}$ και **χωρητικότητα δικτύου** $\boldsymbol{b_i^s}$ που χρειάζεται το ξ_i^s για να έχει τη βασική του λειτουργικότητα στους εξυπηρετητές ενός δοσμένου δικτύου.

2) Μέγιστη **επιπλέον υπολογιστική ισχύς** $\boldsymbol{\Theta_i^s}$ και **χωρητικότητα δικτύου** $\boldsymbol{B_i^s}$ που ονομάζονται ελαστικές απαιτήσεις που το ξ_i^s μπορεί να αξιοποιήσει.

2.1.3) Service Level Agreement Model

Ο Πάροχος Υποδομής εγκαθιστά SLA's με τους παρόχους υπηρεσιών για την εξυπηρέτηση των υπηρεσιών τους για μια περίοδο χρόνου. Το SLA για μια δεδομένη υπηρεσία sεS εμπεριέχει τις παρακάτω παραμέτρους:

1) Η περιγραφή της ροής εργασίας της υπηρεσίας S^s που περιλαμβάνει τις υπολογιστικές απαιτήσεις και τις απαιτήσεις δικτύου του κάθε συστατικού ξ_i^s: θ_i^s , Θ_i^s, b_i^s , B_i^s
2) Το χρονικό διάστημα I^s μέσα στο οποίο η υπηρεσία μπορεί να είναι ενεργή. Για χάριν απλότητας αγνοούμε την επιπλέον πολυπλοκότητα λόγω πολλαπλών χρονικών διαστημάτων με πιθανά διαφορετικές υπηρεσίες που συγκρούονται για πιθανά μη ανατεθειμένα χρονικά διαστήματα. Αυτές οι λεπτομέρειες μπορούν να προστεθούν εύκολα στο μοντέλο.
3) Μια ελάχιστη πιθανότητα ϕ^s ότι οι απαιτούμενοι πόροι είναι όντως διαθέσιμοι όταν η αίτηση φτάνει (μέσα στο I^s), ότι υπάρχουν επαρκείς υπολογιστικοί πόροι και πόροι δικτύου για την ενεργοποίηση των Εικονικών Μηχανών όταν χρειαστεί.
4) Κέρδος G^s για τον Πάροχο Υποδομής σε περίπτωση που η υπηρεσία γίνει αποδεκτή.
5) Πρόστιμο (Penalty) P^s για τον Πάροχο Υποδομής αν η υπηρεσία αποτύχει να ανταπεξέλθει στους QoS περιορισμούς.

Επιπλέον υποτίθεται ότι ο Πάροχος Υποδομής κατέχει ή είναι δυνατό να αποκτήσει πρόσβαση σε εργαλεία που του δίνουν τη δυνατότητα να λάβει γνώση σχετικά με έναν ακόμα παράγοντα που βοηθά στη διαδικασία αποδοχής ή όχι μια υπηρεσίας:

- **Trust T^s**: αυτός ο παράγοντας είναι μια μετρική του πόσο άξιος εμπιστοσύνης είναι ο Πάροχος Υπηρεσίας που του ανήκει μια υπηρεσία.

Αυτός ο παράγοντας έχει θετική ερμηνεία (δηλαδή όσο μεγαλύτερη η τιμή, τόσο μεγαλύτερη είναι η επίδραση του) και είναι ενσωματωμένος στον συνολική αντικειμενική συνάρτηση του Παρόχου Υποδομής.

2.1.4) Περιοριστικοί Κανόνες

Στο πρόβλημα λαμβάνονται υπ' όψιν κάποιοι περιοριστικοί κανόνες που αφορούν την ανάθεση. Αναλυτικότερα έχουμε τρείς περιπτώσεις:

- Κανένας περιορισμός για το συνδυασμό.
- Όλα οι εικονικές μηχανές όλου του συνδυασμού των συστατικών πρέπει να ανατεθούν στο ίδιο δίκτυο.
- Όλες οι εικονικές μηχανές όλου του συνδυασμού των συστατικών πρέπει να ανατεθούν στον ίδιο εξυπηρετητή.

2.2)Περιγραφή μοντέλου Python

Το διαμορφωμένο πρόβλημα εμπίπτει στην κατηγορία των Mixed-Integer Non-Linear Programming (MINLP) προβλημάτων βελτιστοποίησης. Πέρα από την ακριβή μαθηματική διαμόρφωση που επιλύεται από επιλυτές (solvers) με υψηλή πολυπλοκότητα, αναπτύσσεται και ένας προσεγγιστικός αλγόριθμος που μπορεί να προσεγγίσει σχεδόν βέλτιστα αποτελέσματα αλλά με πολύ μεγαλύτερη ταχύτητα από τους MINLP επιλυτές των οποίων η λύση είναι η βέλτιστη.
Τα κυριότερα βήματα αυτής της heuristic προσέγγισης είναι τα εξής:

1. Εύρεση όλων των έγκυρων συνδυασμών των συστατικών για κάθε υπηρεσία που μπορούν να ανατεθούν, βασισμένων στους περιορισμούς του προβλήματος
2. Για κάθε έγκυρο συνδυασμό υπολόγισε το κόστος, την συνολική υπολογιστική ισχύ και χωρητικότητα δικτύου που χρησιμοποιεί και οποιαδήποτε άλλη πληροφορία χρήσιμη για την ανάθεση.
3. Επανάληψη σε όλους τους διαθέσιμους συνδυασμούς για κάθε υπηρεσία και εύρεση μιας βέλτιστης ανάθεσης αν υπάρχει. Αποθήκευση των συνδυασμών των συνδυασμών για τις υπηρεσίες που μεγιστοποιούν το κέρδος για μια δεδομένη υπολογιστική ισχύ.
4. Υπολόγισε το κόστος των αποθηκευμένων αναθέσεων.
5. Δίνεται ως έξοδος η ανάθεση που μεγιστοποιεί το κέρδος μείον το κόστος

Καθένα από τα παραπάνω βήματα θα αναλυθεί παρακάτω.

2.2.1) .Περιορισμοί προβλήματος

Οι έγκυροι συνδυασμοί θα πρέπει να ικανοποιούν τους περιορισμούς δηλαδή οι Εικονικές Μηχανές θα πρέπει να καλύπτουν την ελάχιστη πιθανότητα διαθεσιμότητας (probability of availability). Δεδομένου ότι η απαιτούμενη διαθεσιμότητα ϕ^s είναι σχετικά υψηλή και ότι ο υπολογισμός της πιθανότητα χρησιμοποιεί τον κανόνα του γινομένου ο υπολογισμός των συνδυασμών είναι εφικτός σε εύλογο χρόνο.

Σε μια τυπική περίπτωση θεωρούμε τρείς διακριτές τιμές των Εικονικών Μηχανών για κάθε συστατικό, μια που δίνει μέγιστη χωρητικότητα, μια που δίνει τη ελάχιστη δυνατή χωρητικότητα που είναι αποδεκτή με μια συγκεκριμένη ϕ^s και μια ενδιάμεση τιμή. Το δίκτυο επίσης θα έχει μια χωρητικότητα που εξαρτάται από την υπολογιστική ισχύ. Για μια δεδομένη υπηρεσία s που αποτελείται από m^s συστατικά ο αριθμός των πιθανών περιορισμών είναι:

$$1 + 2 \cdot m^s + \frac{m^s(m^s-1)}{2} = \frac{(m^s+1)(m^s+2)}{2}$$

Επεξηγηματικά υπάρχει ένας συνδυασμός όπου όλα τα συστατικά έχουν μέγιστη ισχύ, $2 \cdot m^s$ συνδυασμοί με μόνο ένα συστατικό σε ελάχιστη ή ενδιάμεση τιμή και $\frac{m^s(m^s-1)}{2}$ συνδυασμοί με δύο συστατικά σε ενδιάμεσες τιμές.

2.2.2) Πληροφορίες των συνδυασμών

Για κάθε έγκυρο συνδυασμό υπολογίζουμε τον παρακάτω όρο ο οποίος αποτελεί τον πρώτο όρο της αντικειμενικής συνάρτησης.

$\sum_{s=1}^{N_s} is\_admit(s)$ * trust(s) * (gain(s) – availability_penalty(s) * (1 – product(s)))

Υπολογίζουμε αυτόν τον όρο γιατί όλες οι άλλες τιμές είναι σταθερές για κάθε υπηρεσία. Για κάθε συνδυασμό χρειαζόμαστε επίσης πληροφορίες σχετικά με τους περιοριστικούς κανόνες και τη συνολική ισχύ για κάθε συστατικό ώστε να μην παραβιάζεται κανένας περιορισμός κατά τη διάρκεια της ανάθεσης.

2.2.3) Εύρεση Αναθέσεων

Η υπολογιστική ισχύς διακριτοποιήθηκε στο μοντέλο μας π.χ. 1GHz = 1 μονάδα υπολογιστικής ισχύος. Ένας προσεγγιστικός αλγόριθμος μπορεί να αναπτυχθεί για το διαμορφωμένο πρόβλημα βελτιστοποίησης του βέλτιστου ελέγχου (ACOP) με διακριτές αναθέσεις υπολογιστικής ισχύος. Ο ακόλουθος ψευδοκώδικας τυποποιεί τον αλγόριθμο για το ACOP:

acop(S,H)
1. C:= 0
2. **for all** s ϵ S **do**
3. **for all** i ϵ S^s **do**

4. C:= C + $\theta_s^i + \Theta_i^s$
5. **end for**
6. **end for**
7. U:= 0
8. **for all** j ϵ H **do**
9. U := U + U_j
10. **end for**
11. C_{max}^s : = min(U,C)
12. **for** c=0 to C_{max}^s **do**
13. $gain[c] := 0$
14. $combs[c] := \{nil\}$
15. **end for**
16. _gain:= gain
17. $\_combs := combs$
18. **for all** s ϵ S do
19. B_v^s := get_valid_combinations(s)
20. **for all** b ϵ B_v^s **do**
21. **for** c=0 **to** C_{max}^s
22. **if** (c≥ c_b) **and** (gain[c] < gain[c-c_b] + G^s)
23. **and** is_found_allocation(combs[c-c_b] + b)
24. **then**
25. gain[c] := gain[c-c_b] + G^s
26. combs[c]:= combs[c- c_b] + b
27. **end if**
28. **end for**
29. **end for**
30. gain:=_gain
31. combs:= _combs
32. **end for**

Στις γραμμές 1-6 υπολογίζεται η συνολική απαιτούμενη υπολογιστική ισχύς για όλα τα συστατικά όλων των υπηρεσιών. Στις γραμμές 7-10 υπολογίζεται η συνολική διαθέσιμη υπολογιστική ισχύς όλων των εξυπηρετητών. Στην γραμμή 11 αναθέτουμε στην μεταβλητή C_{max}^s το ελάχιστο των δύο παραπάνω. Στις γραμμές 12- 15 γίνεται αρχικοποίηση στους πίνακες gain[c] και combs[c]. Στις γραμμές 16 και 17 αρχικοποιούνται οι βοηθητικοί πίνακες _gain και _combs . Στις γραμμές 22 και 23 ελέγχεται αν ο συνδυασμός χωράει σε ισχύ και το κέρδος του βελτιώνει το τρέχον κέρδος

και αν μια ανάθεση ενός νέου συνόλου συνδυασμών έχει βρεθεί ενημερώνονται οι βοηθητικοί πίνακες.

Ο αλγόριθμος ACOP, όπως περιγράφεται παραπάνω, δεν βρίσκει σε όλες τις περιπτώσεις, όλες τις πιθανές αναθέσεις ούτε τις πιο επικερδείς. Ακόμη και το πρόβλημα της εύρεσης αν ένα σύνολο από συστατικά ταιριάζει σε ένα Cloud είναι NP-Hard πρόβλημα και πρακτικά αδύνατο να επιλυθεί αποδοτικά όπως θα εξηγηθεί παρακάτω. Έτσι χρησιμοποιούμε First Fit λαμβάνοντας υπ' όψιν οποιουσδήποτε επιπρόσθετους περιορισμούς.

Το πρόβλημα της εύρεσης αν μια ανάθεση είναι εφικτή ανάγεται σε ένα πρόβλημα απόφασης bin packing σε περίπτωση που έχουμε περιοριστικούς κανόνες για κάθε συστατικό (κάθε συστατικό πρέπει να ανατίθεται σε ένα εξυπηρετητή) και όλοι οι εξυπηρετητές έχουν την ίδια χωρητικότητα. Το πρόβλημα μπορεί τότε να περιγραφεί όπως φαίνεται ακολούθως:

$$\sum_{i \in S^s} k_i^s \cdot \eta_{i,j}^s \le U_j, \qquad \forall j \in \mathrm{H}$$

$$\sum_{j \in \mathrm{H}} \eta_{i,j}^s = 1, \qquad \forall i \in S^s$$

$$\eta_{i,j}^s \in \{0,1\}, \qquad \forall i \in S^s,\ \forall j \in \mathrm{H}$$

όπου k_i^s είναι η συνολική υπολογιστική ισχύς ενός συστατικού (που είναι μια σταθερή ποσότητα ανάμεσα στο θ_i^s και Θ_i^s), $\eta_{i,j}^s$ είναι η λογική μεταβλητή που δείχνει αν ένα συστατικό i ανατείθεται σε έναν εξυπηρετητή j που έχει υπολογιστική ισχύς U_j . Πρέπει επίσης να σημειωθεί ότι η χωρητικότητα δικτύου λαμβάνεται επίσης υπ' όψιν στους περιορισμούς του αλγορίθομου και αυτό καθιστά το πρόβλημα ακόμα πιο δύσκολο.

Το υπόλοιπο του κώδικα είναι μια παραλλαγή του προβλήματος Knapsack που επιλύθηκε χρησιμοποιώντας δυναμικό προγραμματισμό. Ο επιπρόσθετος περιορισμός είναι ότι αν ένας συνδυασμός μιας υπηρεσίας γίνεται αποδεκτός σε ένα νέφος, κανένας συνδυασμός της ίδιας υπηρεσίας δεν μπορεί να γίνει αποδεκτός. Γι' αυτόν το λόγο οι βοηθητικοί πίνακες _gain και _combs και οι πίνακες gain[c] και combs[c] ενημερώνονταν ύστερα από την εξέταση κάθε υπηρεσίας.

Η πολυπλοκότητα του αλγορίθμου είναι:

O ([N_S* $(m^{(s)})^2$ * $2^{m^{(s)}}$] * [N_S * $(m^{(s)})^2$ * $m^{(s)}$ * N_N * N_H * max(i,$\theta_i^s + \Theta_i^s$)])

όπου: N_S ο αριθμός των υπηρεσιών

$m^{(s)}$ ο αριθμός των συστατικών των υπηρεσιών
N_N ο αριθμός των δικτύων
N_H ο αριθμός των εξυπηρετητών
max(i,$\theta_i^s + \Theta_i^s$) το μεγαλύτερο άθροισμα βασικών και ελαστικών απαιτήσεων υπολογιστικής ισχύος ανάμεσα σε όλα τα συστατικά

Επιπρόσθετα υλοποιήθηκε ένας αλγόριθμος ορίου (bound) που υπολογίζει ένα άνω φράγμα της λύσης. Βασίζεται στο ότι αν υπάρχει συνολικά Χ υπολογιστική ισχύς και Υ χωρητικότητα δικτύου και μία υπηρεσία χρειάζεται X_s και Y_s αντίστοιχα, με X_s<Χ και Y_s<Υ τότε η υπηρεσία γίνεται αποδεκτή. Αυτό δεν συμβαίνει απαραίτητα αλλά στόχος του αλγορίθμου είναι η εύρεση ενός άνω φράγματος και όχι της βέλτιστης λύσης

2.2.4) Κόστος της ανάθεσης

Το κόστος της ανάθεσης μπορεί να υπολογισθεί στο προηγούμενο βήμα του αλγορίθμου στη διαδικασία ελέγχου αν η ανάθεση είναι εφικτή. Παρ' όλα αυτά, οι συνδυασμοί που τελικά απορρίπτονται είναι πολύ περισσότεροι από αυτούς που κρατούνται και υπολογίζοντας το κόστος ανάθεσης για καθένα από αυτούς θα προσέθετε ένα επιπλέον φόρτο εργασίας. Για όλους τους συνδυασμούς που κρατήσαμε υπολογίζουμε το κόστος της ανάθεσης δηλαδή το κόστος ενεργοποίησης για όλους τους εξυπηρετητές που είναι αρχικά απενεργοποιημένοι και το κόστος χρήσης για κάθε χρησιμοποιούμενο υπολογιστικό πυρήνα κάθε εξυπηρετητή. Ο αλγόριθμος προσπαθεί να ελαχιστοποιήσει το κόστος χρησιμοποιώντας τους πιο οικονομικούς εξυπηρετητές.

2.2.5) Υπολογισμός Αποτελέσματος

Προκειμένου να υπολογίσουμε τη βέλτιστη ευρεθείσα ανάθεση πρέπει να βρούμε την τιμή της αντικειμενικής συνάρτησης για κάθε ανάθεση που βρέθηκε προηγουμένως. Αυτή ισούται με τη διαφορά του κέρδους μείον το κόστος. Το κόστος εξυπηρέτησης μιας υπηρεσίας είναι γενικά μικρότερο από το κέρδος έτσι όσο περισσότερη υπολογιστική ισχύ αναθέτουμε τόσο περισσότερο κέρδος έχουμε. Παρ' όλα αυτά μια αύξηση στην ισχύ μπορεί να προκαλέσει μικρές αλλαγές στην διαθεσιμότητα και πολύ μικρό κέρδος, γι' αυτό η αντικειμενική συνάρτηση δεν είναι απαραίτητα μεγαλύτερη όταν η μέγιστη υπολογιστική ισχύς ανατίθεται. Έτσι η αντικειμενική συνάρτηση χρειάζεται να υπολογιστεί για κάθε πιθανή τιμή της ανατεθείσας ισχύος.

2.3) Περιγραφή μοντέλου Gams

Το General Algebraic Modeling System (GAMS) είναι ένα υψηλού επιπέδου σύστημα μοντελοποίησης για μαθηματικό προγραμματισμό και βελτιστοποίηση[3][4]. Με το Gams μπορούν να μοντελοποιηθούν προβλήματα βελτιστοποίησης και εύρεσης λύσης. Το συγκεκριμένο πρόβλημα είναι βελτιστοποίησης. Το πρόβλημα μοντελοποιήθηκε στο GAMS και επιλύθηκε κάτω από ρεαλιστικές ρυθμίσεις του παρόχου που δείχνουν την αποδοτικότητα του προτεινόμενου μοντέλου

Υπάρχουν διάφοροι έτοιμοι επιλυτές (solvers) και για το πρόβλημα αυτό που είναι της κατηγορίας Mixed Integer NonLinear Programming (MINLP) επιλέχτηκε ο Branch-And-Reduce Optimization Navigator (BARON)[5]. Οι επιλυτές χρησιμοποιούν τυποποιημένους αλγορίθμους, προσπαθούν να προσεγγίσουν λύσεις με αναλυτικές μεθόδους ή μεθόδους γραμμικού προγραμματισμού και μπορεί να ρυθμιστεί η ακρίβεια της λύσης δηλαδή πόσο κοντά είναι στη βέλτιστη λύση.

Στο GAMS υπάρχουν κάποια βασικά στοιχεία και δομές που χρησιμοποιούνται και στο μοντέλο που αναπτύχθηκε:

- SETS (σύνολα): Τα σύνολα είναι μια βασική έννοια στο GAMS. Με τη βοήθεια τους, κατασκευάζουμε όλες τις συνδέσεις ανάμεσα στις μεταβλητές (variables) και τις εξισώσεις (equations) στα περισσότερα μοντέλα. Είναι παρόμοια με τα σύνολα των μαθηματικών και επιτρέπονται διάφορες πράξεις για αυτά όπως ένωση (uniting), τομή (intersection) κ.α. Στο μοντέλο που αναπτύχθηκε έχουμε πολλά σύνολα όπως π.χ. το σύνολο των υπηρεσιών, το σύνολο των εξυπηρετητών κ.α.
- Parameters (Παράμετροι): Για να ορίσουμε ένα μονοδιάστατο πίνακα χρησιμοποιήθηκαν παράμετροι. Στο μοντέλο που αναπτύχθηκε έχουμε πολλές παραμέτρους όπως ο μέγιστος αριθμός των cpus ανά εξυπηρετητή, το κέρδος ανά υπηρεσία κ.α.
- Tables(Πίνακες): Για να ορίσουμε πολυδιάστατους πίνακες χρησιμοποιήθηκαν tables. Στο μοντέλο που αναπτύχθηκε έχουμε πολλούς πίνακες όπως τους δισδιάστατους πίνακες για τις βασικές απαιτήσεις δικτύου, τις ελαστικές απαιτήσεις δικτύου κ.α.
- Binary Variables (Λογικές Μεταβλητές): μπορούν να πάρουν την τιμή 1 ή 0, όπως μια μεταβλητή που παίρνει την τιμή 1 αν ένας συγκεκριμένος εξυπηρετητής έχει χρησιμοποιηθεί στην ανάθεση, αλλιώς παίρνει την τιμή 0.

- Integer Variables (ακέραιες μεταβλητές): παίρνουν τιμές στο [0,1,2,…] όπως μια μεταβλητή που δηλώνει το πλήθος των cpus ενός συγκεκριμένου εξυπηρετητή που χρησιμοποιούνται. Στις μεταβλητές μπορούμε να θέσουμε όρια, αρχικές τιμές και σταθερές τιμές. Στην προαναφερθείσα μεταβλητή για παράδειγμα έχουμε θέσει ελάχιστη τιμή το 0 και μέγιστη τις διαθέσιμες cpus όπως καθορίζονται από την είσοδο της εκάστοτε περίπτωσης που εξετάζουμε.
- Positive Variables: πρόκειται για τις πραγματικές μεταβλητές με τιμή από 0 έως +∞.
- Equations (εξισώσεις): οι εξισώσεις αποτελούνται από δύο μέρη, την δήλωση των ονομάτων των εξισώσεων και τις εξισώσεις. Μια από τις εξισώσεις είναι και η αντικειμενική συνάρτηση του GAMS
- Model (μοντέλο): έπειτα από την δήλωση των συνόλων, παραμέτρων, μεταβλητών και εξισώσεων μπορούμε να δημιουργήσουμε ένα συγκεκριμένο μοντέλο και να του δώσουμε ένα όνομα. Το μοντέλο αυτό θα περιέχει κάποιες ή όλες τις εξισώσεις ανάλογα με τη μοντελοποίηση
- Τέλος επιλέγουμε την επίλυση του μοντέλου μεγιστοποιώντας την αντικειμενική συνάρτηση:

  “Solve model_name using minlp maximizing objective;”

2.4) Παράμετροι Προβλήματος

Στο μοντέλο GAMS και Python έχουμε ως εισόδους τις παραμέτρους του προβλήματος δηλαδή τον αριθμό των υπηρεσιών και των συστατικών τους, τον αριθμό των εξυπηρετητών και των δικτύων. Επίσης ως είσοδοι λαμβάνονται και όλες οι ποσότητες που αφορούν την μοντελοποίηση του προβλήματος. Αναλυτικά στις εισόδους αναφέρονται:

1) Ο αριθμός $m^{(s)}$ των συστατικών που αποτελούν κάθε υπηρεσία
2) Ο αριθμός των εξυπηρετητών N_H
3) Ο αριθμός των δικτύων N_N
4) Ο αριθμός των υπηρεσιών N_S
5) Η διαθεσιμότητα ϕ^s, δηλαδή η πιθανότητα για κάθε υπηρεσία να βρούμε διαθέσιμη υπολογιστική ισχύ και χωρητικότητα δικτύου.
6) Το πρόστιμο διαθεσιμότητας P^s, δηλαδή το κόστος σε περίπτωση που η υπηρεσία δεν βρει διαθέσιμους τους απαιτούμενους επιπλέον πόρους, οδηγώντας στην αναγκαιότητα να πληρώσουμε πρόστιμο πίσω στον Πάροχο Υπηρεσίας
7) Το κέρδος G^s για κάθε υπηρεσία αν αυτή εξυπηρετηθεί

8) Οι βασικές απαιτήσεις σε χωρητικότητα δικτύου b_i^s για κάθε συστατικό κάθε υπηρεσίας
9) Ο αριθμός βασικών εικονικών μηχανών θ_i^s για κάθε συστατικό κάθε υπηρεσίας
10) Το κόστος χρήσης κάθε cpu ανά εξυπηρετητή
11) Περιοριστικοί κανόνες που αφορούν την ανάθεση
12) Το κόστος ενεργοποίησης ανά εξυπηρετητή
13) Οι ελαστικές απαιτήσεις σε χωρητικότητα δικτύου B_i^s για κάθε συστατικό κάθε υπηρεσίας
14) Ο αριθμός των ελαστικών εικονικών μηχανών Θ_i^s για κάθε συστατικό κάθε υπηρεσίας
15) Ο μέγιστος αριθμός επεξεργαστών ανά εξυπηρετητή
16) Η χωρητικότητα κάθε δικτύου U_n
17) Η τοπολογία H_n ενός δικτύου n ϵ N που καθορίζει ποιοι εξυπηρετητές ανήκουν σε κάθε δίκτυο
18) Ο αριθμός δεσμευμένων επεξεργαστών ανά εξυπηρετητή
19) Η εμπιστοσύνη T^S είναι μια μετρική που εκφράζει πόσο άξιος εμπιστοσύνης είναι ο Πάροχος Υπηρεσιών που του ανήκει μια υπηρεσία
20) πόσοι υπολογιστικοί πυρήνες χρειάζονται για να τρέξει ένα στιγμιότυπο

3)Δυνατότητες που παρέχει η Python

3.1) Πιθανότητες και Κατανομές [6] [7][8]

Συνάρτηση Κατανομής Πιθανότητας

Ονομάζουμε συνάρτηση κατανομής πιθανότητας (cumulative distribution function) της τυχαίας μεταβλητής X την:

$$F(x)=P(X\leq x),\ x\epsilon R$$

Η συνάρτηση F(x) εκφράζει μια πιθανότητα συνεπώς έχει τις εξής ιδιότητες:

1) Είναι φραγμένη ανάμεσα στο μηδέν και το ένα.
2) Είναι μια μονότονη μη φθίνουσα συνάρτηση της μεταβλητής x. Δηλαδή, $F(x_1) \leq F(x_2)$ αν $x_1 < x_2$

Συνάρτηση Πυκνότητας Πιθανότητας

Ονομάζουμε συνάρτηση πυκνότητας πιθανότητας (probability density function) της τυχαίας μεταβλητής X την f με πεδίο ορισμού το R τέτοια ώστε:

$f(x)\geq 0$, για κάθε $x\epsilon R$

$F(x) = P[X\leq x] = \int_{-\infty}^{x} f(y)dy$, για κάθε $x\epsilon R$ (Σχέση 1)

Προκύπτει λοιπόν από τον παραπάνω ορισμό ότι:

$$P[\alpha\leq X\leq b] = \int_{a}^{b} f(x)dx$$

Το εμβαδόν κάτω από την καμπύλη f για το διάστημα [α,b] δίνει την πιθανότητα η τ.μ. X να λάβει τιμή στο [α,b]

Αν θεωρήσουμε το b να τείνει στο α, λαμβάνουμε

$P(X=\alpha) = 0$, για κάθε $\alpha\epsilon R$

Συνεπώς στις συνεχείς κατανομές η πιθανότητα να λάβει η τυχαία μεταβλητή X συγκεκριμένη τιμή, έστω α, είναι μηδέν, ενώ το α δεν παύει να είναι μια δυνατή τιμή της τυχαίας μεταβλητής X.

Από την (Σχέση 1) συνάγεται ότι η παράγωγος μιας συνεχούς συνάρτησης κατανομής πιθανότητας F όπου υπάρχει, συμπίπτει με τη συνάρτηση πυκνότητας πιθανότητας f.
Δηλαδή

$$f(x) = \frac{d}{dx}F(x)$$

εφόσον η παράγωγος υπάρχει.

Δεδομένου ότι ισχύει η σχέση F(+∞)=1, η οποία αντιστοιχεί στην πιθανότητα ενός βεβαίου συμβάντος και η σχέση F(-∞)=0, η οποία αντιστοιχεί στην πιθανότητα ενός αδυνάτου συμβάντος διαπιστώνουμε ότι:

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

<u>Ομοιόμορφη κατανομή</u>
Μια τυχαία μεταβλητή X λέγεται ότι είναι ομοιόμορφα κατανεμημένη σε ένα διάστημα (α,b) και γράφουμε X~ U(α,b) αν η συνάρτηση πυκνότητας πιθανότητας της είναι:

$$f(x)=\begin{cases} 0, & x \le a \\ \frac{1}{a-b}, & a < x \le b \\ 0, & x > b \end{cases}$$

Η συνάρτηση κατανομής πιθανότητας της μεταβλητής X είναι κατά συνέπεια

$$F(x)=\begin{cases} 0, & x \le a \\ \frac{x-a}{b-a}, & a < x \le b \\ 1, & x > b \end{cases}$$

Κανονική κατανομή (Gauss)

Η συνεχής τυχαία μεταβλητή X ακολουθεί την Κανονική κατανομή με παραμέτρους μ, σ ($-\infty<\mu<+\infty$, $\sigma>0$) και γράφουμε $X \sim N(\mu,\sigma^2)$ όταν έχει συνάρτηση πιθανότητας την:

$$f(x)=\frac{1}{\sqrt{2\pi}\,\sigma}\exp\left(-\frac{1}{2\sigma^2}\,(x-\mu)^2\right) \quad , \quad x\epsilon R$$

Μέση τιμή: μ
Διασπορά: σ^2

Η συνάρτηση πυκνότητας πιθανότητας της κανονικής κατανομής είναι συμμετρική ως προς τον άξονα x=μ και έχει κορυφή στο x=μ.

Η συνάρτηση κατανομής πιθανότητας της X είναι:

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}\,\sigma}\exp\left(-\frac{1}{2\sigma^2}\,(y-\mu)^2\right)dy$$

Κατανομή Γάμμα

Η κατανομή Γάμμα συμβολίζεται με G(α,b) με παραμέτρους α και b, α>0 και b>0 και έχει συνάρτηση πυκνότητας πιθανότητας:

$$f(x)=\begin{cases} \frac{x^{a-1}\,e^{-bx}\,b^{a}}{\Gamma(a)} & , \quad x>0 \\ 0 & , \quad \chi\leq 0 \end{cases}$$

όπου Γ(α) αναπαριστά τη συνάρτηση Γάμμα που ορίζεται ως

$$\Gamma(\alpha)=\int_0^{\infty}\chi^{\alpha-1}e^{-x}dx \quad ,\alpha>0$$

και ισχύει Γ(1)=1 , $\Gamma(\frac{1}{2})=\sqrt{\pi}$, Γ(n+1)=n! για nϵN

μέση τιμή: $\frac{\alpha}{b}$

Διασπορά $\frac{\alpha}{b^2}$

Εκθετική κατανομή

Είναι μια ειδική περίπτωση της κατανομής Γάμμα που προκύπτει θέτοντας α=1. Η συνάρτηση πυκνότητας πιθανότητας της εκθετικής κατανομής είναι:

$$f(x)=\begin{cases} \lambda e^{-\lambda x} , & x \geq 0 \\ 0 , & x < 0 \end{cases}$$

όπου το λ>0

Η συνάρτηση κατανομής πιθανότητας της εκθετικής κατανομής είναι:

$$F(x)=1 - e^{-\lambda x} , x \geq 0$$

$$F(x)=\begin{cases} 0 & ,x < 0 \\ 1 - e^{-\lambda x} & ,x \geq 0 \end{cases}$$

Μέση τιμή: $\frac{1}{\lambda}$

Διασπορά: $\frac{1}{\lambda^2}$

3.2) Παραγωγή random αριθμών στην Python

Για τη μοντελοποίηση του προβλήματος σε Python χρησιμοποιήθηκε η έκδοση Python 3.3.0 . Χρησιμοποιήθηκε η βιβλιοθήκη random για την παραγωγή ψευδό-τυχαίων αριθμών (pseudo-random numbers) [9] [10] . Σχεδόν όλες οι συναρτήσεις της βιβλιοθήκης αυτής στηρίζονται στην βασική συνάρτηση random() , που παράγει ένα τυχαίο αριθμό κινητής υποδιαστολής (float) ομοιόμορφα κατανεμημένο στο [0.0 , 1.0) . Η Python χρησιμοποιεί το Mersenne Twister για την παραγωγή των αριθμών. Το Mersenne Twister είναι ένας από τους υφιστάμενους παραγωγούς τυχαίων αριθμών, ελεγχθείς εκτενώς.

Αναλυτικότερα χρησιμοποιήθηκαν οι παρακάτω συναρτήσεις:

- random.randint(a,b) : επιστρέφει έναν τυχαίο ακέραιο N τέτοιο ώστε a ≤ N ≤ b
- random.randrange(start,stop[,step]) : επιστρέφει έναν τυχαίο ακέραιο N με ελάχιστη τιμή start, μέγιστη stop και βήμα step (αν υπάρχει βήμα).

- random.choice(seq) : επιστρέφει μια τυχαία επιλογή από την ακολουθία seq
- random.gauss(mu,sigma) : επιστρέφει μια τιμή που ακολουθεί την κανονική (γκαουσιανή) κατανομή όπου mu είναι η μέση τιμή και sigma είναι η τυπική απόκλιση.
- random.gammavariate(alpha,beta): επιστρέφει μια τιμή που ακολουθεί την κατανομή Γάμμα (όπου alpha>0 αι beta>0).

3.3) testing και ορθότητα

Το unittest unit testing framework της Python βασίστηκε στο JUnit και είναι παρεμφερές με αλλά unit testing frameworks σε άλλες γλώσσες[11][12] . Υποστηρίζει κάποιες βασικές ιδέες όπως το test case και το test suite. Ένα test case είναι η βασική μονάδα του testing. Ελέγχει για μια συγκεκριμένη έξοδο σε ένα συγκεκριμένο σύνολο εισόδων. Το unittest παρέχει μια βασική κλάση (class) την Testcase.

Ένα testcase δημιουργείται ως υποκλάση του unittest.Testcase . Τα διάφορα test ορίζονται με μεθόδους των οποίων το όνομα αρχίζει με τα γράμματα “test” . Αυτή η παραδοχή ενημερώνει τον test runner ποιες μέθοδοι αποτελούν tests. Η κλάση Testcase παρέχει έναν αριθμό μεθόδων για τον έλεγχο, πολλές από τις οποίες χρησιμοποιήθηκαν όπως:

- assertEqual(a , b): ελέγχει αν το a και b είναι ίσα. Αν οι τιμές δεν βρεθούν ίσες, το τεστ θα αποτύχει.
- assertNotEqual(a , b): ελέγχει αν το a και b δεν είναι ίσα. Αν οι τιμές βρεθούν ίσες, το τεστ θα αποτύχει.
- assertLessEqual(a , b): Ελέγχει αν το a είναι μικρότερο ίσο του b. Αν το a βρεθεί μεγαλύτερο το τεστ θα αποτύχει.
- assertRaises() που επιβεβαιώνει ότι μια αναμενόμενη exception προκύπτει.

Επιπρόσθετα για το testing του μοντέλου που αναπτύχθηκε ορίστηκαν οι μέθοδοι setUp() και tearDown(). Όταν ορίζεται η μέθοδος setUp() ο test runner θα εκτελέσει αυτή τη μέθοδο πριν από οποιοδήποτε test. Αντίστοιχα όταν ορίζεται η tearDown() ο test runner θα εκτελέσει αυτή τη μέθοδο μετά από κάθε test.

Ένα test suite είναι μια συλλογή από test cases, test suites ή και τα δύο. Χρησιμοποιείται για να ομαδοποιήσει tests που θα έπρεπε να εκτελούνται μαζί. Η εκτέλεση ενός στιγμιότυπου (instance) είναι ισοδύναμη με την εκτέλεση κάθε test ξεχωριστά. Υπάρχει η κλάση TestSuite του unittest. Πολύ χρήσιμες είναι οι μέθοδοι addTest(test) που προσθέτει ένα Testcase ή

TestSuite στο suite και run(result) που τρέχει τα συνδεδεμένα test με αυτό το suite, συλλέγοντας το αποτέλεσμα στο αντικείμενο test result.

Μέσω του testing ελέχθησαν αναλυτικά σχεδόν όλες οι μέθοδοι του μοντέλου σε πολλές περιπτώσεις. Στο παράρτημα φαίνεται ένα μέρος ενός testcase που αναπτύχθηκε για τον έλεγχο μιας συγκεκριμένης περίπτωσης.

3.4) Python Profiler

Ένα profile είναι ένα σύνολο στατιστικών που περιγράφουν πόσο συχνά εκτελούνται συγκεκριμένα μέρη ενός προγράμματος και πόσο διαρκεί η εκτέλεση αυτή. Η cProfile παρέχει τη δυνατότητα δημιουργίας ντετερμινιστικού profile για ένα πρόγραμμα. Έχει βασιστεί στη γλώσσα C και είναι κατάλληλη για δημιουργία profile μεγάλων προγράμματα με μεγάλους χρόνους εκτέλεσης[13][14] .

Χρησιμοποιώντας την cProfile λοιπόν έγινε καταγραφή όλων των κλήσεων των μεθόδων του προβλήματος, του συνολικού αριθμού των κλήσεων, του χρόνου που καταναλώθηκε για κάθε μέθοδο σε αρκετές περιπτώσεις. Συνεπώς στη συνέχεια με κατάλληλη βελτιστοποίηση του κώδικα μειώθηκε ο αριθμός των κλήσεων αλλά και ο χρόνος που καταναλώθηκε από κάθε μέθοδο.

4)Αποτελέσματα

Θεωρούμε ενδεικτικές περιπτώσεις όπου για τα τέσσερα μεγέθη (υπηρεσίες, συστατικά, εξυπηρετητές, δίκτυα) διατηρούμε σταθερό τον αριθμό των τριών εκ των τεσσάρων με το τέταρτο να μεταβάλλεται ώστε να διαπιστώσουμε την επίδραση της μεταβολής στα παρακάτω:

- το χρόνο υπολογισμού της βέλτιστης λύσης της Python
- το χρόνο υπολογισμού του ορίου (bound) της Python
- το χρόνο υπολογισμού της αντικειμενικής συνάρτησης στο μοντέλο του GAMS
- το αποτέλεσμα της αντικειμενικής συνάρτησης για την Python
- το αποτέλεσμα της αντικειμενικής συνάρτησης του μοντέλου του GAMS
- το αποτέλεσμα του ορίου(bound) της Python.

Ακόμη υπολογίζουμε σε κάθε περίπτωση τους παρακάτω λόγους:

- λόγος του αποτελέσματος της αντικειμενικής συνάρτησης του μοντέλου του GAMS προς το αποτέλεσμα της αντικειμενικής συνάρτησης για την Python
- λόγος του ορίου της Python προς το αποτέλεσμα της αντικειμενικής συνάρτησης του μοντέλου του GAMS
- λόγος του χρόνο υπολογισμού της αντικειμενικής συνάρτησης στο μοντέλο του GAMS προς το χρόνο υπολογισμού του ορίου της Python
- λόγος του χρόνου υπολογισμού της αντικειμενικής συνάρτησης στο μοντέλο του GAMS προς το χρόνο υπολογισμού της βέλτιστης λύσης της Python

Οι παραπάνω ποσότητες θα αναφέρονται συντομότερα όπως φαίνεται στον Πίνακας 1 :

Ποσότητες που εξετάζονται	**Συντομεύσεις**
Χρόνος cpu υπολογισμού της βέλτιστης λύσης της Python (αντικειμενικής συνάρτησης)	χρόνος Python
Χρόνος cpu υπολογισμού του ορίου (bound) της Python	Χρόνος ορίου Python
Χρόνος cpu υπολογισμού της αντικειμενικής συνάρτησης στο μοντέλο του GAMS	χρόνος GAMS ή χρόνος BARON
Αποτέλεσμα της αντικειμενικής συνάρτησης για την Python	αποτέλεσμα Python
Αποτέλεσμα της αντικειμενικής συνάρτησης του μοντέλου του GAMS	αποτέλεσμα GAMS ή αποτέλεσμα BARON
Αποτέλεσμα του ορίου(bound) της Python	όριο Python
λόγος του αποτελέσματος της αντικειμενικής συνάρτησης του μοντέλου του GAMS προς το αποτέλεσμα της αντικειμενικής συνάρτησης για την Python	Αποτέλεσμα GAMS προς αποτέλεσμα Python
λόγος του αποτελέσματος της αντικειμενικής συνάρτησης του μοντέλου του GAMS προς το αποτέλεσμα του ορίου της Python	Αποτέλεσμα GAMS προς όριο Python
Λόγος του χρόνου cpu υπολογισμού της αντικειμενικής συνάρτησης στο μοντέλο του GAMS προς το χρόνο υπολογισμού της βέλτιστης λύσης στην Python	Χρόνος GAMS προς χρόνο Python
Λόγος του χρόνου cpu υπολογισμού της αντικειμενικής συνάρτησης στο μοντέλο του GAMS προς το χρόνο υπολογισμού του ορίου της Python	Χρόνος GAMS προς χρόνο ορίου Python

Πίνακας 1 : Συντομεύσεις ποσοτήτων που εξετάζονται

4.1)Σύντομη παρουσίαση των περιπτώσεων που εξετάζονται

Α1) θεωρούμε την περίπτωση όπου έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια και 8 εξυπηρετητές σε 2 δίκτυα. Όσον αφορά τις υπόλοιπες εισόδους όλες οι μετρικές παίρνουν την ίδια σταθερή τιμή για όλα τα συστατικά, όλες τις υπηρεσίες, όλους τους εξυπηρετητές και όλα τα δίκτυα.

Α2) Έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια και 8 εξυπηρετητές σε 2 δίκτυα. Όσον αφορά τις υπόλοιπες μετρικές παίρνουν την ίδια τυχαία τιμή για όλα τα συστατικά, όλες τις υπηρεσίες, όλα τους εξυπηρετητές και όλα τα δίκτυα. Η τυχαία αυτή τιμή προκύπτει από τη βιβλιοθήκη random της Python.

Α3) Έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια και 8 εξυπηρετητές σε 2 δίκτυα. Όσον αφορά τα τις υπόλοιπες μετρικές θεωρούμε παίρνουν (ενδεχομένως[1]) διαφορετική τυχαία τιμή για όλα τα συστατικά, όλες τις υπηρεσίες, όλα τους εξυπηρετητές και όλα τα δίκτυα αντίστοιχα. Η τυχαία αυτή τιμή προκύπτει από τη βιβλιοθήκη random της Python.

Α4) Θεωρούμε την περίπτωση όπου έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια, 8 εξυπηρετητές σε 2 δίκτυα. Όλες οι υπόλοιπες μετρικές παίρνουν την ίδια σταθερή τιμή για όλα τα συστατικά, όλους τους εξυπηρετητές, όλα τις υπηρεσίες και τα όλα δίκτυα αντίστοιχα.

Α5) Θεωρούμε την περίπτωση όπου έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια και 8 εξυπηρετητές σε 2 δίκτυα. Όσον αφορά τα τις υπόλοιπες μετρικές θεωρούμε παίρνουν διαφορετική τυχαία τιμή για όλα τα συστατικά, όλες τις υπηρεσίες, όλα τους εξυπηρετητές και όλα τα δίκτυα αντίστοιχα. Η τυχαία αυτή τιμή προκύπτει από τη βιβλιοθήκη random της Python και πολλές από τις τιμές αυτές ακολουθούν συγκεκριμένες κατανομές. Για κάθε στιγμιότυπο (συγκεκριμένος αριθμό υπηρεσιών, συστατικών, εξυπηρετητών και δικτύων) γίνονται 30 εκτελέσεις απ' όπου προκύπτουν στατιστικά στοιχεία, πίνακες και διαγράμματα με μεγαλύτερη ακρίβεια.

[1] Λέγοντας ενδεχομένως εννοείται ότι υπάρχει η περίπτωση δύο τυχαίες τιμές να είναι ίσες. Το ίδιο ισχύει και για τις επόμενες περιπτώσεις όπου υπάρχουν τυχαίες τιμές

Α6) (profiles πολλές εκτελέσεις) Θεωρούμε 4 προφίλ με συγκεκριμένο αριθμό υπηρεσιών, συστατικών, δικτύων και αυξανόμενο αριθμό εξυπηρετητών. Για κάθε στιγμιότυπο γίνονται 30 εκτελέσεις.

Οι μετρήσεις έγιναν σε υπολογιστή με τα παρακάτω χαρακτηριστικά :

- λειτουργικό σύστημα: Windows 7 Home Premium
- επεξεργαστής: Intel(R) Core(TM) i5 CPU M 460 @ 2.53Ghz 2.53GHz
- μνήμη RAM: 4GB
- τύπος συστήματος: 64-bit

4.2) Πρώτη Κατηγορία Περιπτώσεων

Αρχικά θεωρούμε μια ενδεικτική περίπτωση όπου έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια, 8 εξυπηρετητές σε 2 δίκτυα. Για τα τέσσερα μεγέθη (υπηρεσίες, συστατικά, εξυπηρετητές, δίκτυα) διατηρούμε σταθερό τον αριθμό των τριών εκ των τεσσάρων με το τέταρτο να μεταβάλλεται ώστε να διαπιστώσουμε την επίδραση της μεταβολής. Όλες οι υπόλοιπες μετρικές παίρνουν την ίδια σταθερή τιμή για όλα τα συστατικά, όλους τους εξυπηρετητές, όλα τις υπηρεσίες και τα όλα δίκτυα αντίστοιχα. Εκτελούμε μια φορά κάθε στιγμιότυπο αφού όλες οι μετρικές παίρνουν την ίδια σταθερή τιμή. Εκτελώντας περισσότερες φορές ένα στιγμιότυπο, θα προέκυπταν πάλι τα ίδια αποτελέσματα.

- Πρόστιμο διαθεσιμότητας = 100 για κάθε υπηρεσία
- διαθεσιμότητα = 0.9 για κάθε υπηρεσία
- κέρδος =1000 για κάθε υπηρεσία
- βασικές απαιτήσεις δικτύου = 1 για κάθε συστατικό κάθε υπηρεσίας
- βασικές απαιτήσεις εικονικών μηχανών = 1 για κάθε συστατικό κάθε υπηρεσίας
- κόστος χρήσης = 5 για κάθε εξυπηρετητή
- κόστος ενεργοποίησης = 10 για κάθε εξυπηρετητή
- ελαστικές απαιτήσεις δικτύου = 2 για κάθε συστατικό κάθε υπηρεσίας
- ελαστικές απαιτήσεις εικονικών μηχανών = 2 για κάθε συστατικό κάθε υπηρεσίας
- μέγιστος αριθμός επεξεργαστών = 16 για κάθε εξυπηρετητή
- χωρητικότητα δικτύου = 1000 για κάθε δίκτυο
- δεσμευμένοι επεξεργαστές = 0 για κάθε εξυπηρετητή
- όχι περιοριστικούς κανόνες
- τοπολογία δικτύου = ίσος αριθμός εξυπηρετητών σε κάθε δίκτυο (σε περίπτωση περιττού αριθμού εξυπηρετητών το ένα δίκτυο έχει έναν παραπάνω εξυπηρετητή)
- εμπιστοσύνη = 1

4.2.1) Αυξανόμενος Αριθμός Συστατικών

Στην Εικόνα 1 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των συστατικών που αποτελούν την κάθε υπηρεσία σχεδιάζονται στον άξονα x με μέγιστο το 20 ενώ ο χρόνος που

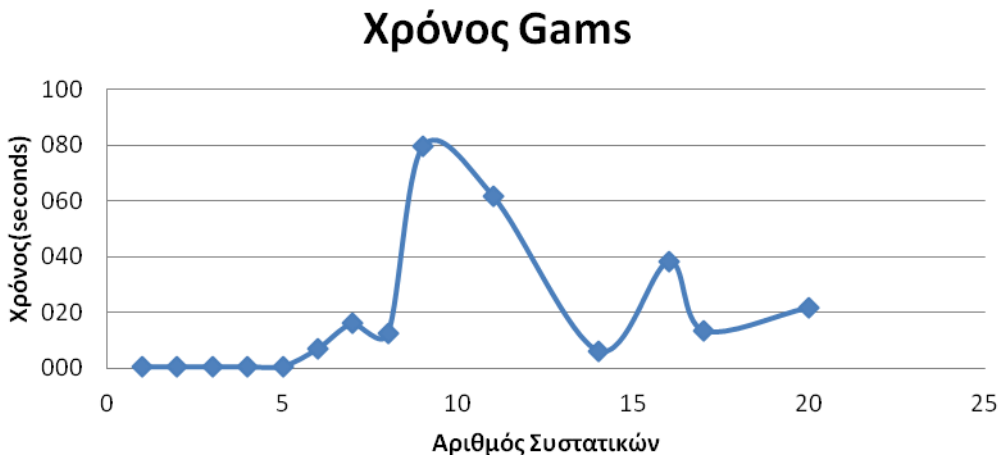
καταναλώθηκε σχεδιάζεται στον άξονα y. Στην συγκεκριμένη περίπτωση όλοι οι εξυπηρετητές ήταν ανενεργοί (και της ίδιας υπολογιστικής ισχύος). Να τονίσουμε ότι έχουμε θέσει όριο τερματισμού στο χρόνο CPU για το GAMS στα 120 δευτερόλεπτα. Παρατηρούμε ότι 6/20 περιπτώσεις, συγκεκριμένα για 10,12,13,15,18 , 19 συστατικά, ξεπέρασαν αυτό το όριο και γι' αυτό δεν εμφανίζονται στο σχήμα.

Στην Εικόνα 2 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό συστατικών.

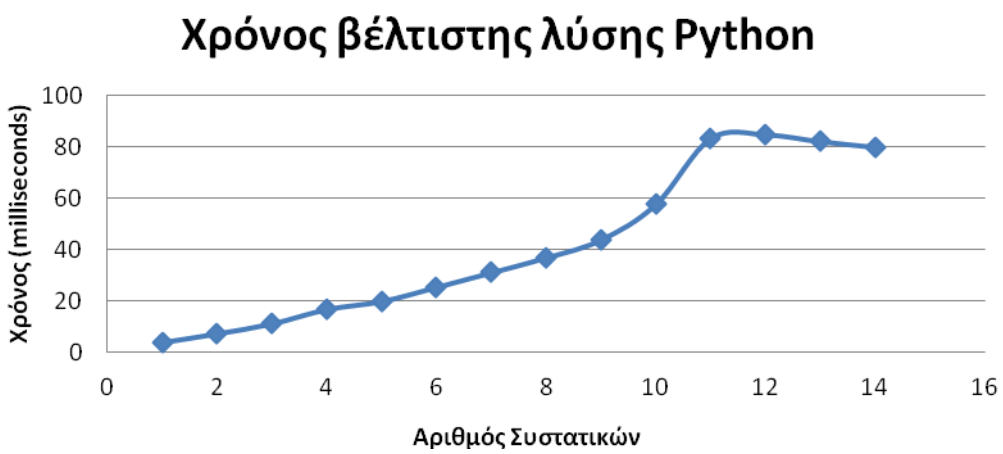
Στην Εικόνα 3 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό συστατικών.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις. Στην Εικόνα 4 και την Εικόνα 5 φαίνονται οι λόγοι των χρόνων υπολογισμού των παραπάνω αποτελεσμάτων.

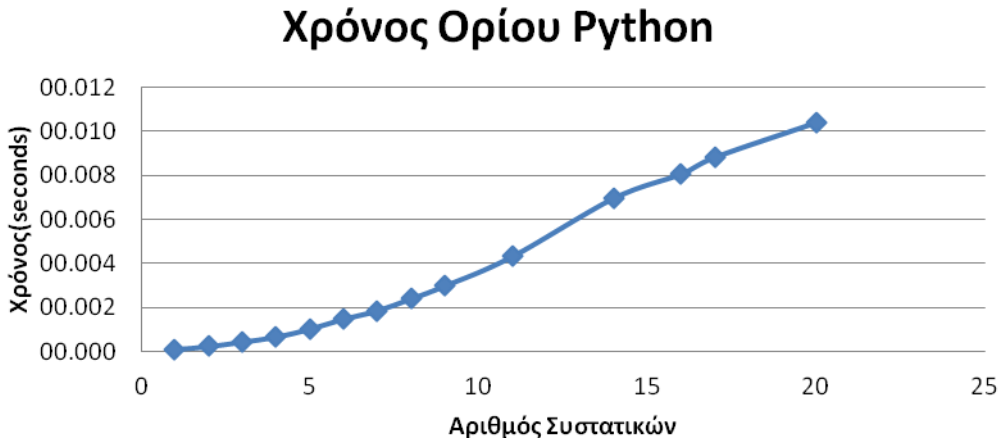
Παρατηρούμε ότι για τις παραπάνω περιπτώσεις η Python είναι 35,5 έως 1811 φορές πιο γρήγορη από το GAMS χωρίς απόκλιση στο αποτέλεσμα. Επίσης παρατηρείται αύξηση του χρόνου Python και του χρόνος ορίου Python με την αύξηση του αριθμού των συστατικών.



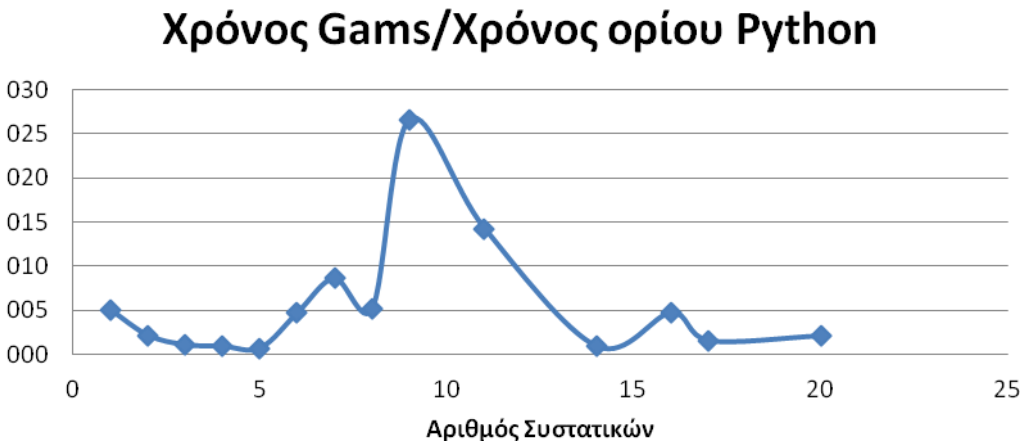
Εικόνα 1 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό συστατικών



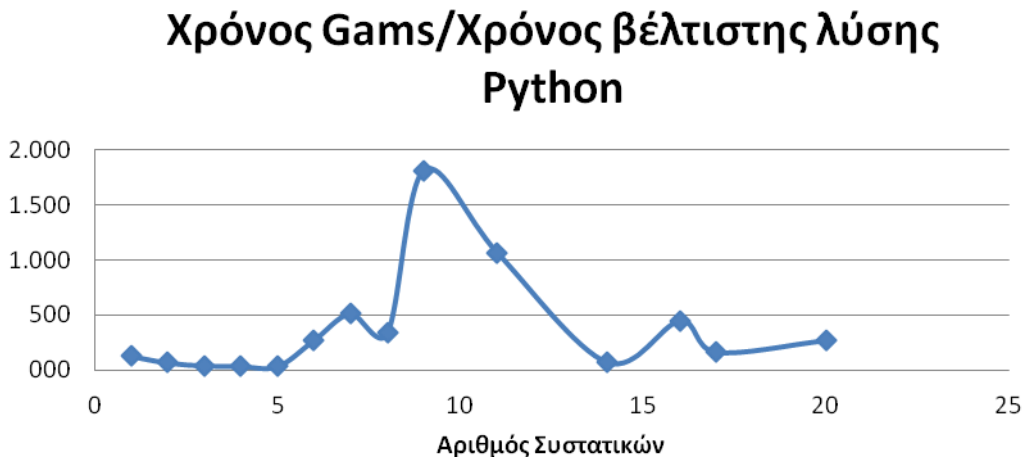
Εικόνα 2 : . Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό συστατικών



Εικόνα 3 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό συστατικών



Εικόνα 4 :Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών



Εικόνα 5 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών

4.2.2)Αυξανόμενος Αριθμός Εξυπηρετητών

Στην Εικόνα 6 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των εξυπηρετητών που θεωρούνται υποψήφιοι για ανάθεση σχεδιάζονται στον άξονα x με μέγιστο το 49 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Στην συγκεκριμένη περίπτωση όλοι οι εξυπηρετητές ήταν ανενεργοί (και της ίδιας υπολογιστικής ισχύος). Να τονίσουμε ότι έχουμε θέσει όριο τερματισμού στο χρόνο CPU για το GAMS τα 120 δευτερόλεπτα.

Στην Εικόνα 7 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό εξυπηρετητών.

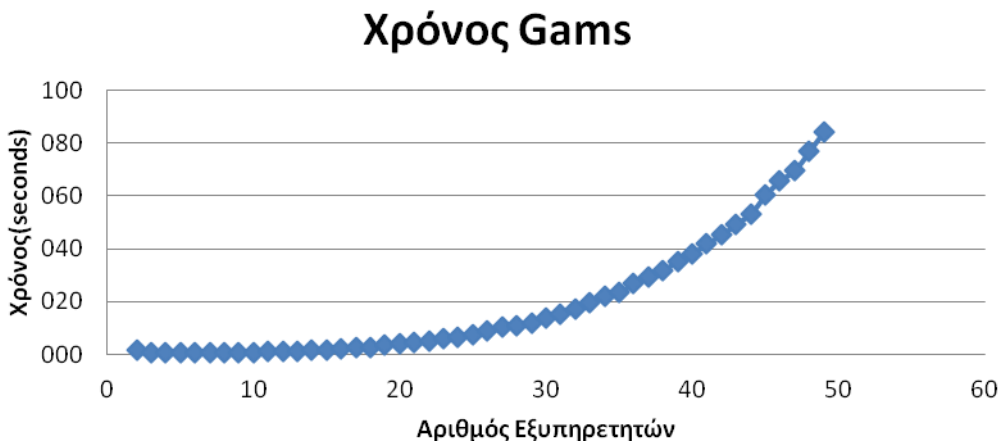
Στην Εικόνα 8 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο σε καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό εξυπηρετητών.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις.

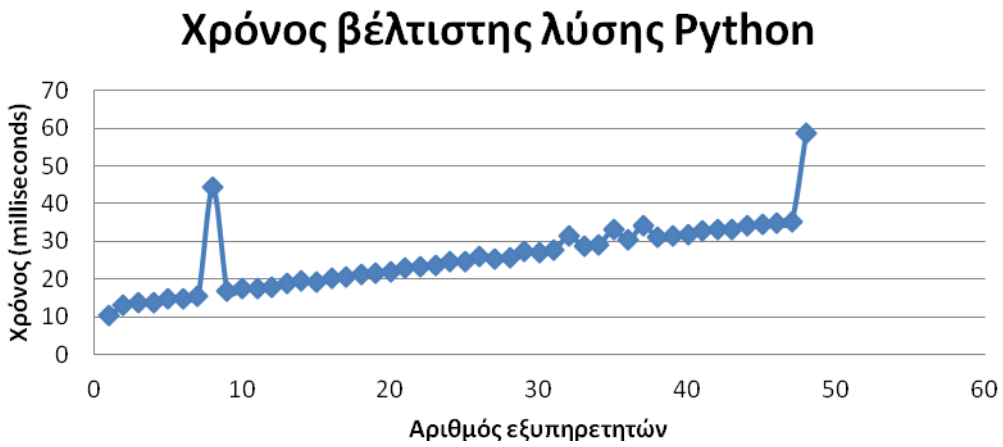
Όσον αφορά τους χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 9 και την Εικόνα 10 φαίνονται οι λόγοι των χρόνων υπολογισμού. Πιο συγκεκριμένα ο χρόνος GAMS προς το χρόνο Python κυμαίνεται από 16,7233 έως 2190 και ο χρόνος GAMS προς το χρόνο ορίου Python από 15,19 έως 47,36 .

Παρατηρούμε ότι οι χρόνοι GAMS, Python, ορίου Python αυξάνονται με την αύξηση του αριθμού των εξυπηρετητών. Επίσης η αύξηση του χρόνου

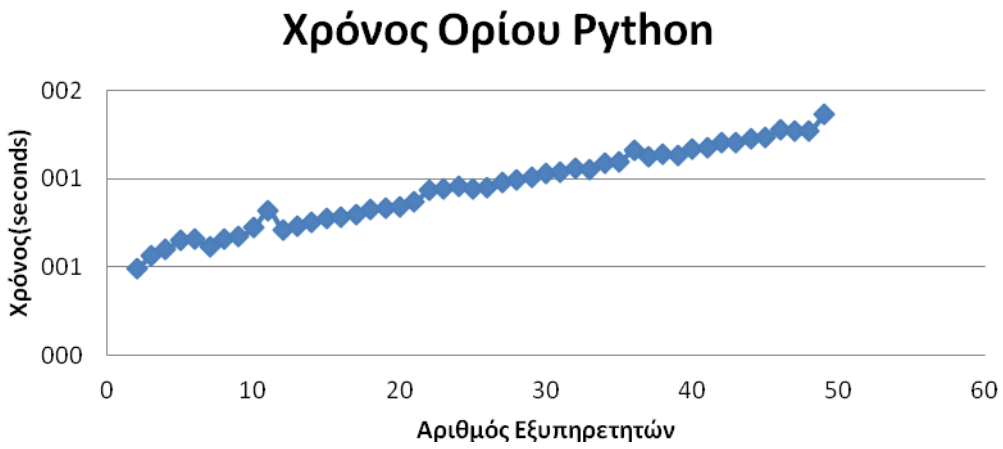
GAMS είναι μεγαλύτερη από αυτή του χρόνου Python και του ορίου της Python.



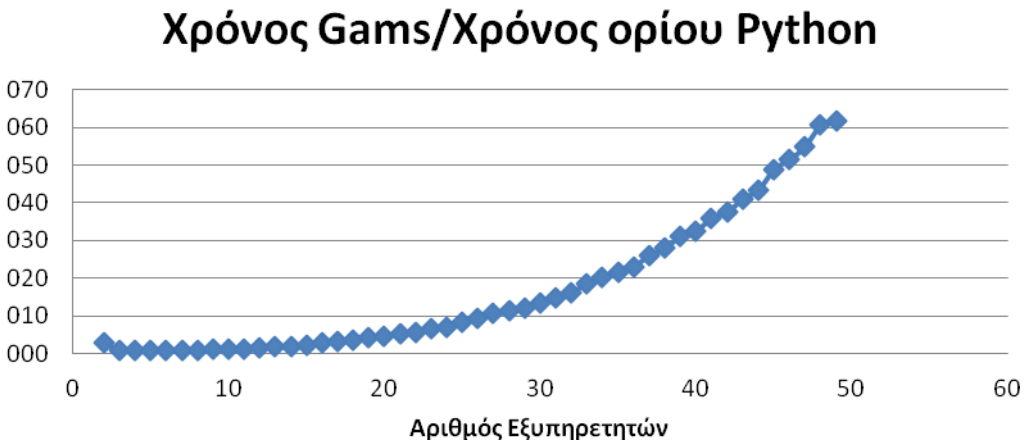
Εικόνα 6 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών



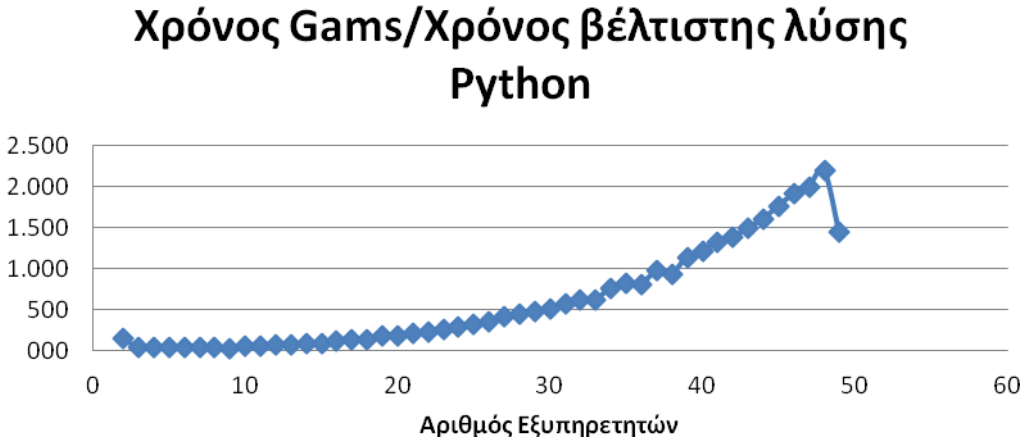
Εικόνα 7 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 8 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 9 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 10 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python

4.2.3)Αυξανόμενος Αριθμός Υπηρεσιών

Στην Εικόνα 11 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των υπηρεσιών προς ανάθεση σχεδιάζονται στον άξονα x με μέγιστο το 10 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Στην συγκεκριμένη περίπτωση όλοι οι εξυπηρετητές ήταν ανενεργοί (και της ίδιας υπολογιστικής ισχύος). Να τονίσουμε ότι έχουμε θέσει όριο τερματισμού στο χρόνο CPU για το GAMS τα 120 δευτερόλεπτα. Για περιπτώσεις με αριθμό υπηρεσίες από 11 έως 49 ο χρόνος του GAMS ξεπερνά τα 120 δευτερόλεπτα γι' αυτό και δεν σχεδιάζονται στο διάγραμμα. (39/49 περιπτώσεις ξεπέρασαν το 120 δευτερόλεπτα)

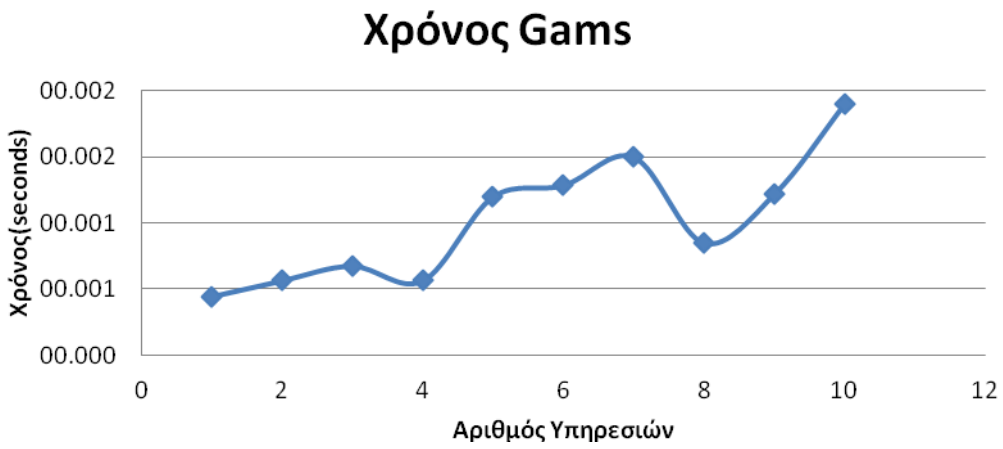
Στην Εικόνα 12 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό υπηρεσιών.

Στην Εικόνα 13 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό υπηρεσιών.

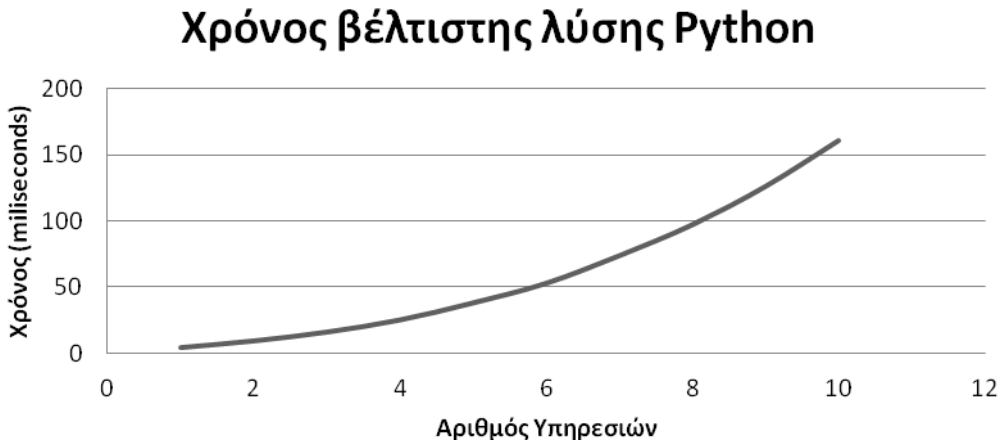
Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις εκτός από την περίπτωση με services=8 όπου ο λόγος τους είναι 1,0013

Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 14 και την Εικόνα 15 φαίνονται οι λόγοι των χρόνων υπολογισμού.

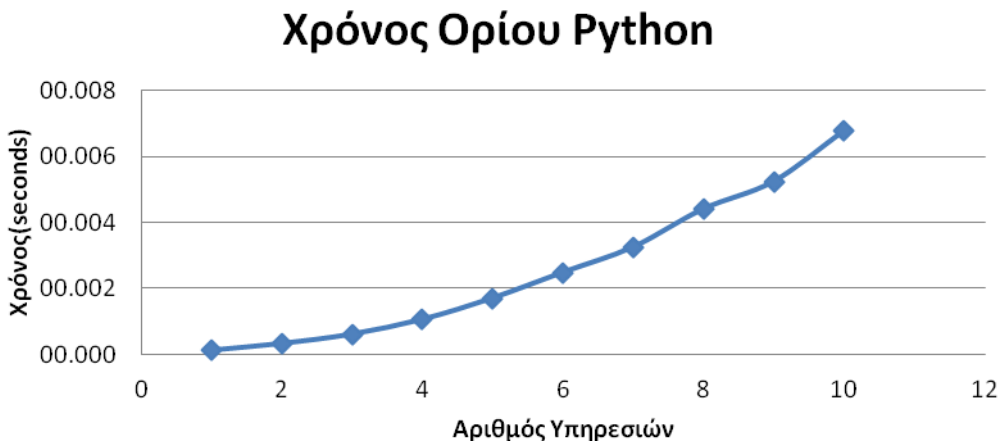
Παρατηρούμε αύξηση στου χρόνους Python και ορίου Python με την αύξηση του αριθμού των υπηρεσιών. Επίσης ο χρόνος GAMS ήταν από 9 έως 117 φορές μεγαλύτερος από τον χρόνο Python (με στρογγυλοποίηση).



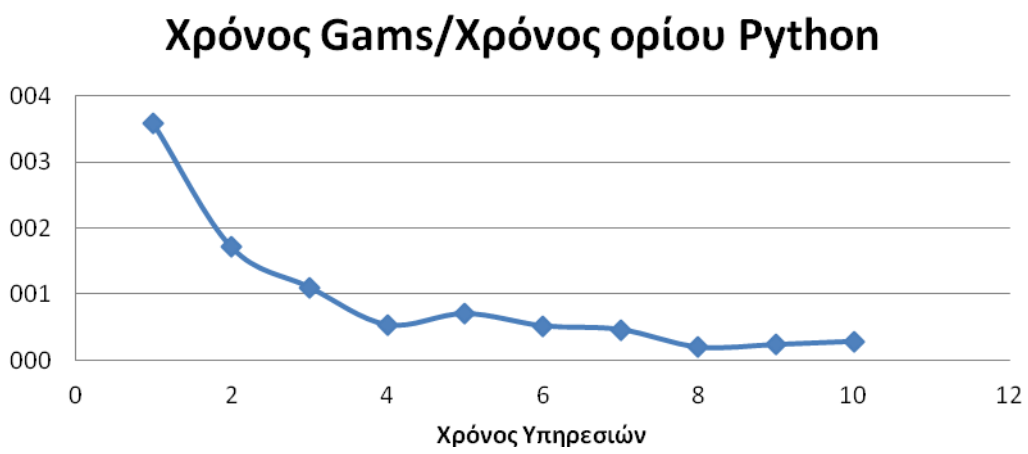
Εικόνα 11 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό υπηρεσιών



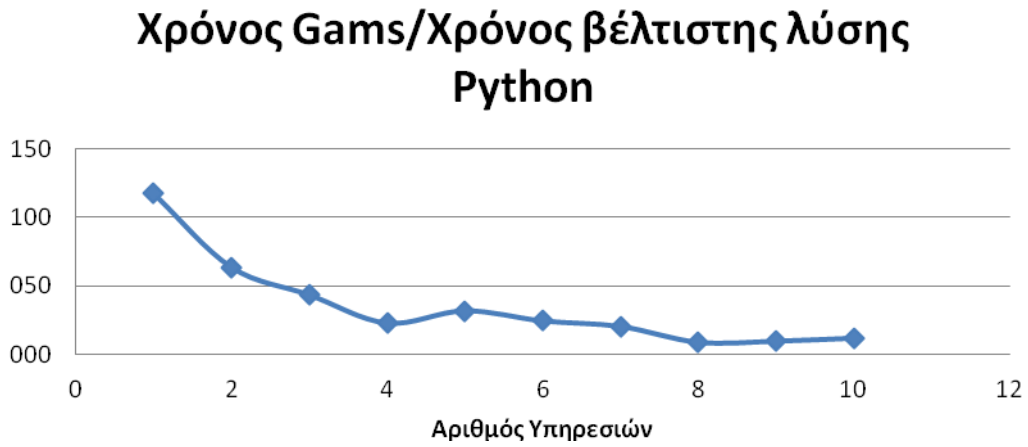
Εικόνα 12 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 13 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 14 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 15 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό υπηρεσιών

4.2.4)Αυξανόμενος Αριθμός Δικτύων

Στην Εικόνα 16 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των δικτύων σχεδιάζονται στον άξονα x με μέγιστο το 23 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Στην συγκεκριμένη περίπτωση όλοι οι εξυπηρετητές ήταν ανενεργοί (και της ίδιας υπολογιστικής ισχύος). Έχουμε αυξήσει τον αριθμό των εξυπηρετητών σε 25 ενώ η τοπολογία δικτύου που προσδιορίζει ποιοι εξυπηρετητές υπάγονται σε κάθε δίκτυο, προσπαθεί να τους ισομοιράσει, όταν είναι εφικτό. Σε κάθε περίπτωση κάποιο δίκτυο περιέχει το πολύ έναν εξυπηρετητή παραπάνω από κάποιο άλλο.

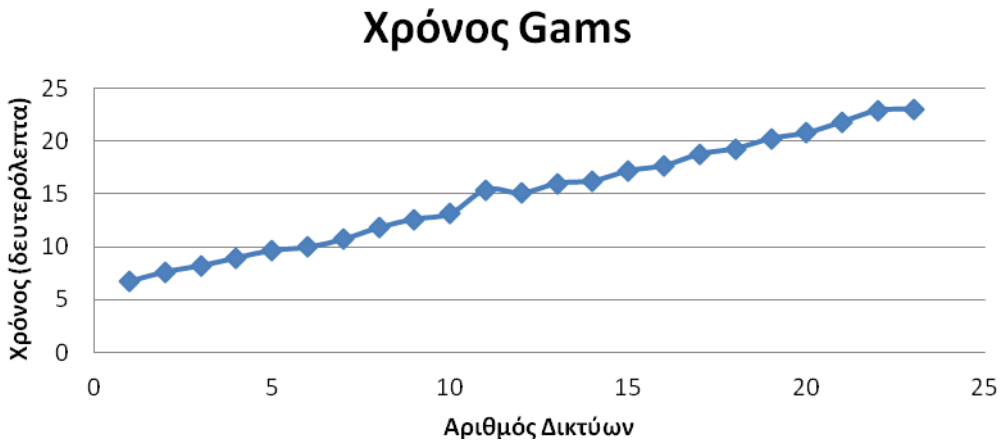
Στην Εικόνα 17 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό δικτύων.

Στην Εικόνα 18 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό δικτύων.

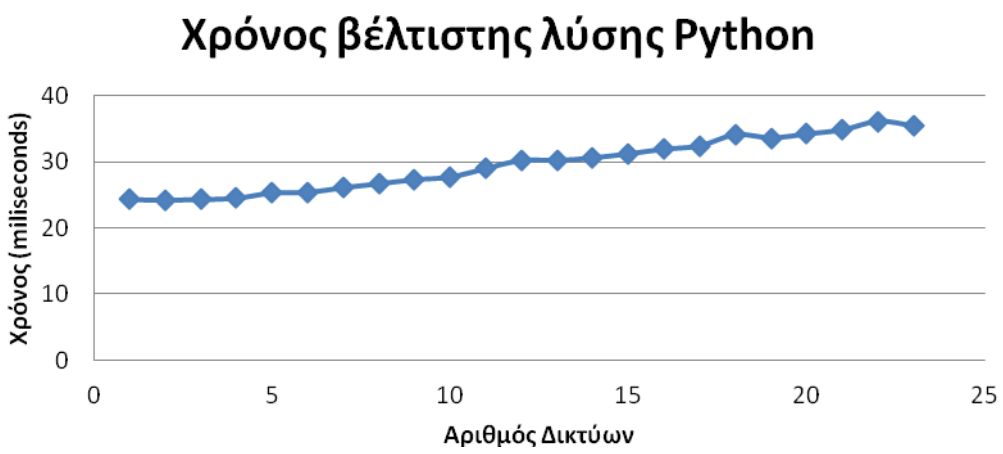
Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις εξεταζόμενες περιπτώσεις.

Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 19 και την Εικόνα 20 φαίνονται οι λόγοι των χρόνων υπολογισμού.

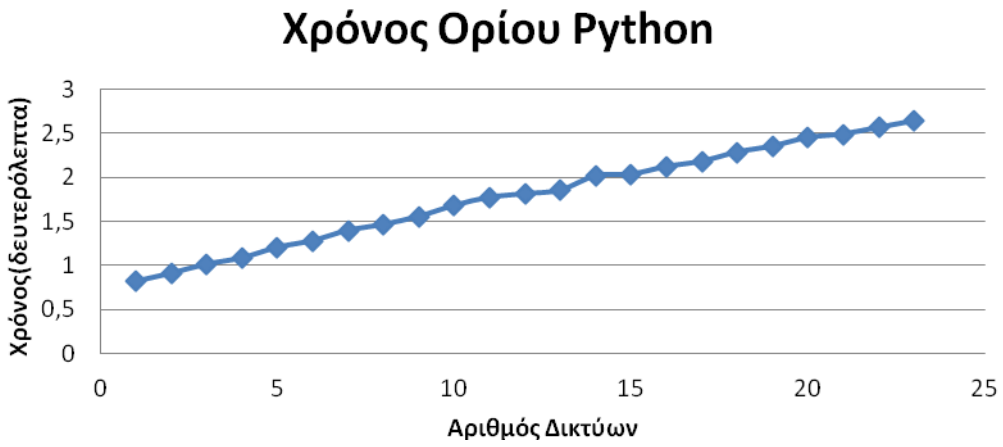
Παρατηρούμε ότι οι χρόνοι GAMS, Python και ορίου Python αυξάνονται με την αύξηση του αριθμού των εξυπηρετητών. Επίσης η αύξηση του χρόνου GAMS είναι μεγαλύτερη από αυτή του χρόνου Python.



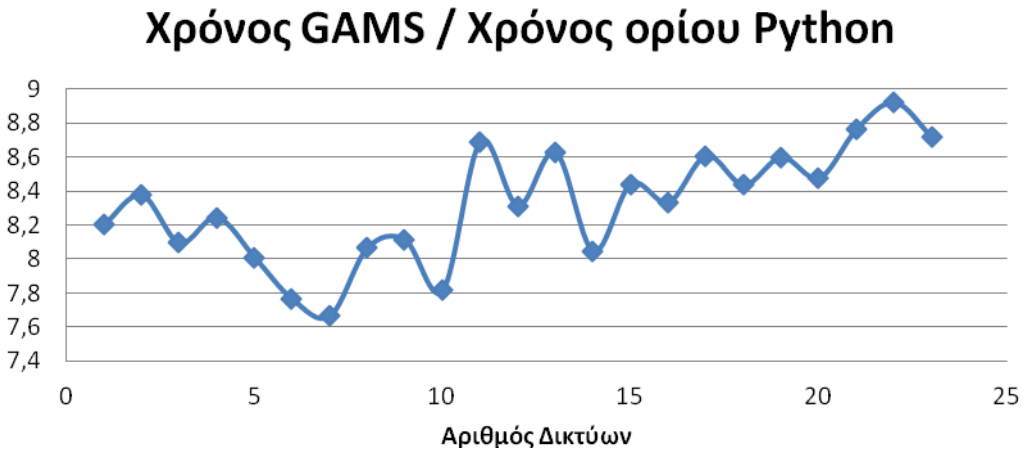
Εικόνα 16 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό δικτύων



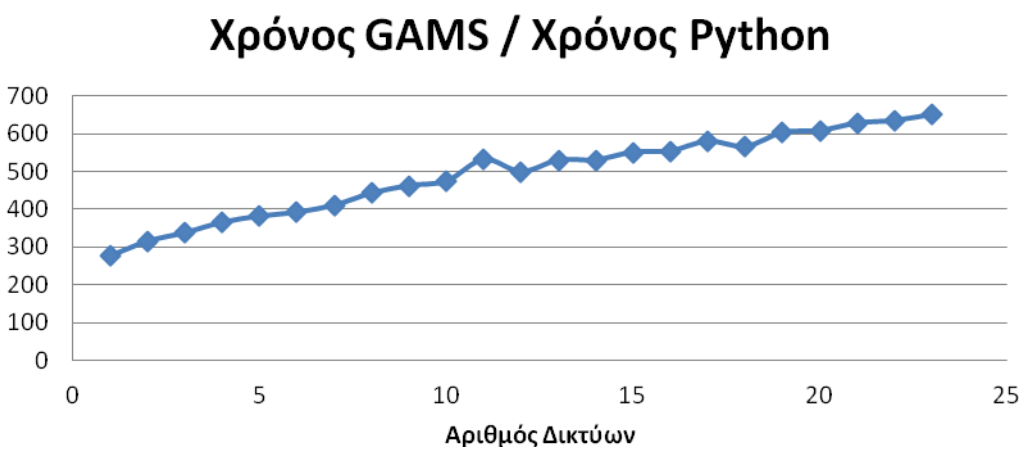
Εικόνα 17 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό δικτύων



Εικόνα 18 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό δικτύων



Εικόνα 19 : . Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό δικτύων



Εικόνα 20 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό δικτύων

4.3) Δεύτερη Κατηγορία Περιπτώσεων

Έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια, 8 εξυπηρετητές σε 2 δίκτυα. Όσον αφορά τις υπόλοιπες μετρικές παίρνουν την ίδια τυχαία τιμή για όλα τα συστατικά, όλες τις υπηρεσίες, όλους τους εξυπηρετητές και όλα τα δίκτυα. Η τυχαία αυτή τιμή προκύπτει από τη βιβλιοθήκη random της Python.

- Πρόστιμο διαθεσιμότητας = 100 έως και 200 με βήμα 20 ίδιο για όλες τις υπηρεσίες
- Διαθεσιμότητα = 0,9 έως και 1 με βήμα 0,1 ίδιο για όλες τις υπηρεσίες
- Κέρδος = 500 έως 1000 με βήμα 10 ίδιο για όλες τις υπηρεσίες
- βασικές απαιτήσεις δικτύου = ακέραιος από 1 έως 20 ίδιος για όλα τα συστατικά όλων των υπηρεσιών
- βασικές απαιτήσεις εικονικών μηχανών = ακέραιος από 1 έως 3 ίδιος για όλα τα συστατικά όλων των υπηρεσιών
- κόστος χρήσης = ακέραιος από 1 έως 20 ίδιος για όλους τους εξυπηρετητές
- κόστος ενεργοποίησης = ακέραιος από 10 έως 30 ίδιος για όλους τους εξυπηρετητές
- ελαστικές απαιτήσεις δικτύου = ακέραιος από 1 έως 20 ίδιος για όλα τα συστατικά όλων των υπηρεσιών
- ελαστικές απαιτήσεις εικονικών μηχανών = ακέραιος από 1 έως 15 ίδιος για όλα τα συστατικά όλων των υπηρεσιών
- μέγιστος αριθμός επεξεργαστών = 16 για όλους τους εξυπηρετητές
- χωρητικότητα δικτύου = τυχαία επιλογή ανάμεσα σε 500, 1000, 2000, ίδια για όλα τα δίκτυα
- δεσμευμένοι επεξεργαστές = ακέραιος από 0 έως 15 ίδιος για όλους τους εξυπηρετητές.
- τοπολογία δικτύου = ίσος αριθμός εξυπηρετητών σε κάθε δίκτυο (σε περίπτωση περιττού αριθμού εξυπηρετητών το ένα δίκτυο έχει έναν παραπάνω εξυπηρετητή)
- εμπιστοσύνη = 1
- όχι περιοριστικοί κανόνες
-

4.3.1) Αυξανόμενος Αριθμός Συστατικών

Στην Εικόνα 21 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των συστατικών σχεδιάζονται στον άξονα x με μέγιστο το 20 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Σε 2 από τις 20 περιπτώσεις και συγκεκριμένα για 16 και 20 συστατικά ο χρόνος υπολογισμού στο GAMS ξεπέρασε το όριο των 120 δευτερόλεπτων και δεν σχεδιάζεται στα σχήματα. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (για ανενεργό εξυπηρετητή) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητής

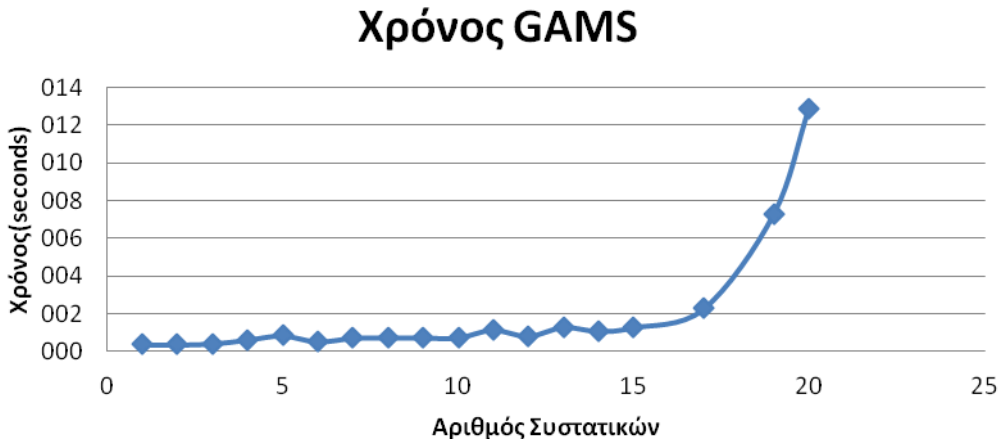
Στην Εικόνα 22 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό συστατικών.

Στην Εικόνα 23 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό συστατικών.

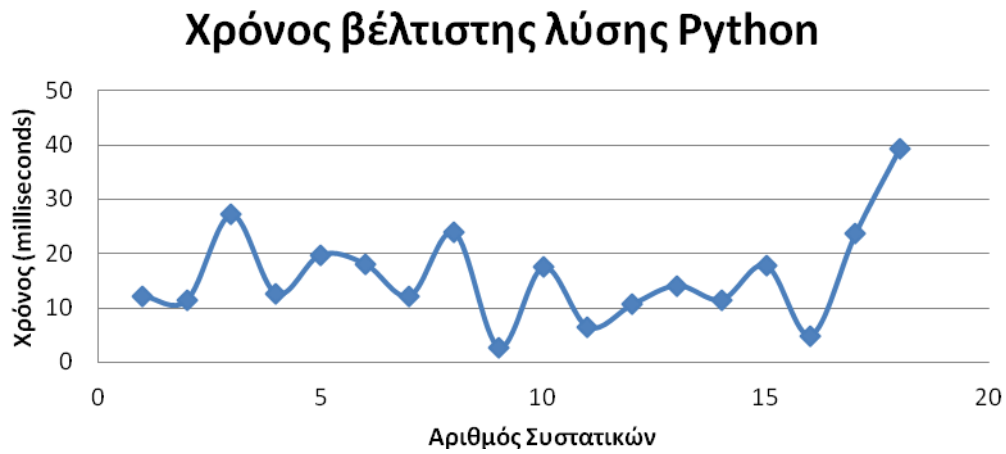
Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις

Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 24 και την Εικόνα 25 φαίνονται οι λόγοι των χρόνων υπολογισμού.

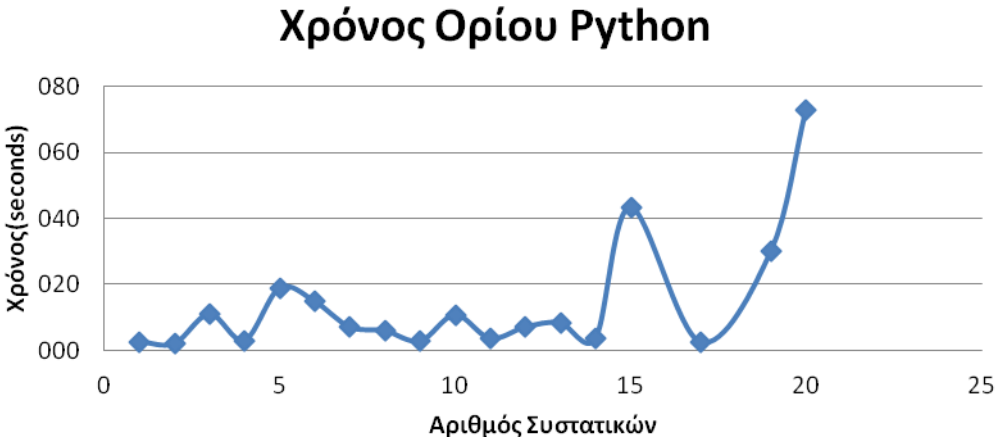
Παρατηρείται επίσης αύξηση του χρόνου GAMS με την αύξηση του αριθμού των συστατικών.



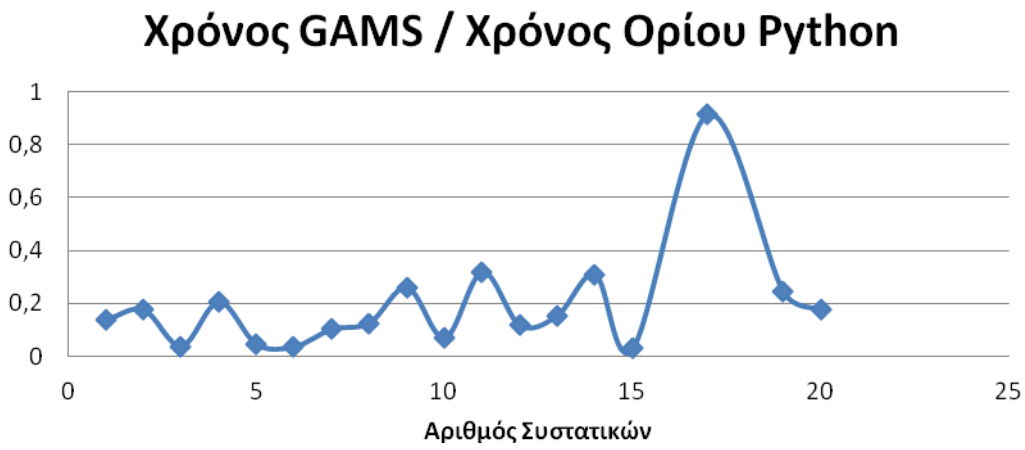
Εικόνα 21 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό συστατικών



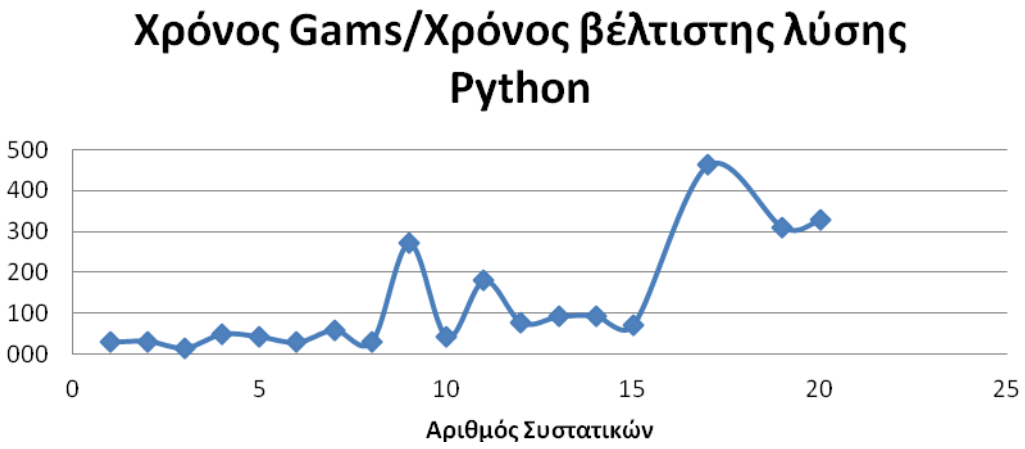
Εικόνα 22 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό συστατικών



Εικόνα 23 : . Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό συστατικών



Εικόνα 24 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών



Εικόνα 25 : . Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών

4.3.2)Αυξανόμενος Αριθμός Εξυπηρετητών

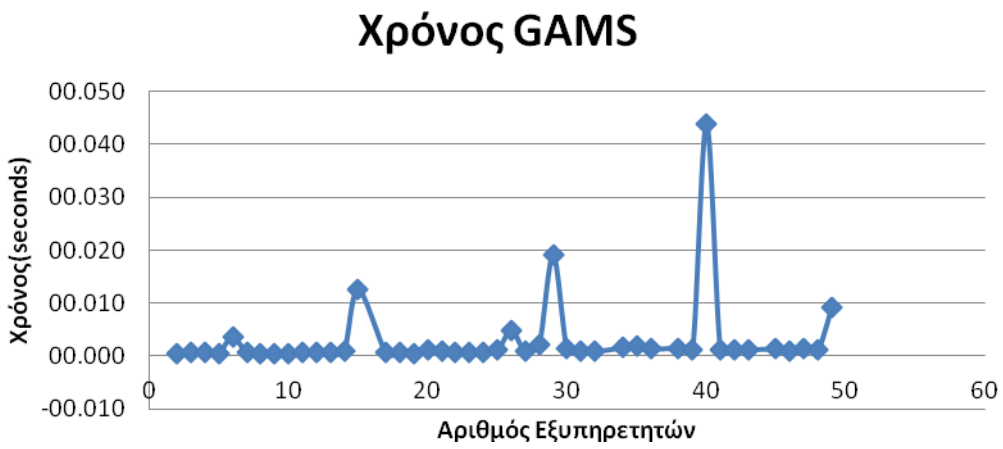
Στην Εικόνα 26 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των εξυπηρετητών σχεδιάζονται στον άξονα x με μέγιστο το 49 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Σε 4 από τις 48 περιπτώσεις και συγκεκριμένα για 16, 33, 37 και 44 συστατικά ο χρόνος υπολογισμού στο GAMS ξεπέρασε το όριο των 120 δευτερόλεπτων και δεν σχεδιάζεται στα σχήματα. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (δηλαδή για ανενεργό εξυπηρετητή) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητής

Στην Εικόνα 27 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό εξυπηρετητών.

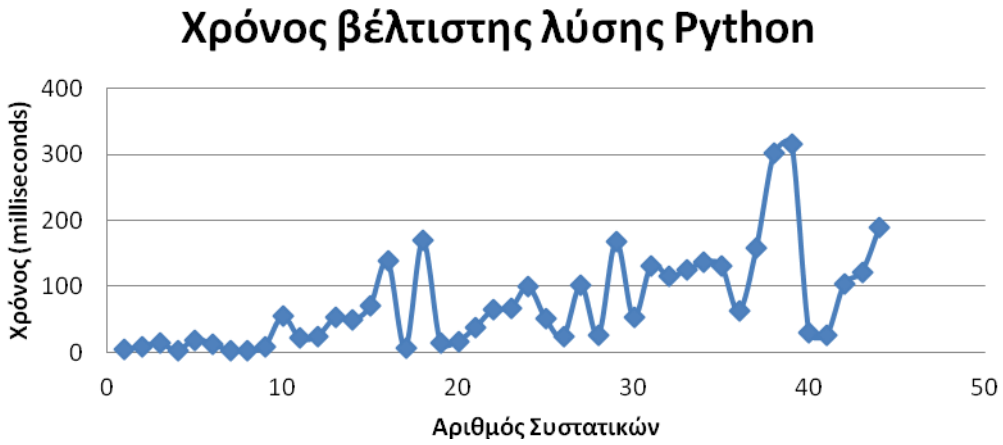
Στην Εικόνα 28 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό εξυπηρετητών.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις

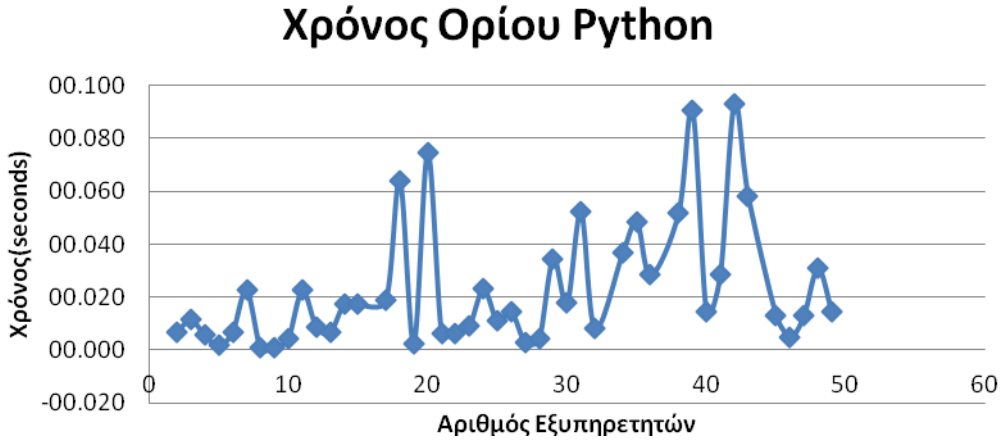
Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 29 και την Εικόνα 30 φαίνονται οι λόγοι των χρόνων υπολογισμού.



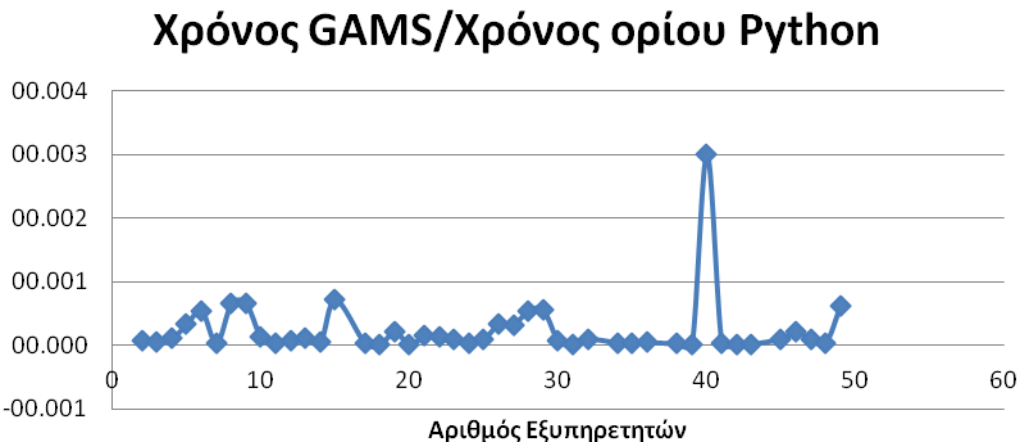
Εικόνα 26 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών



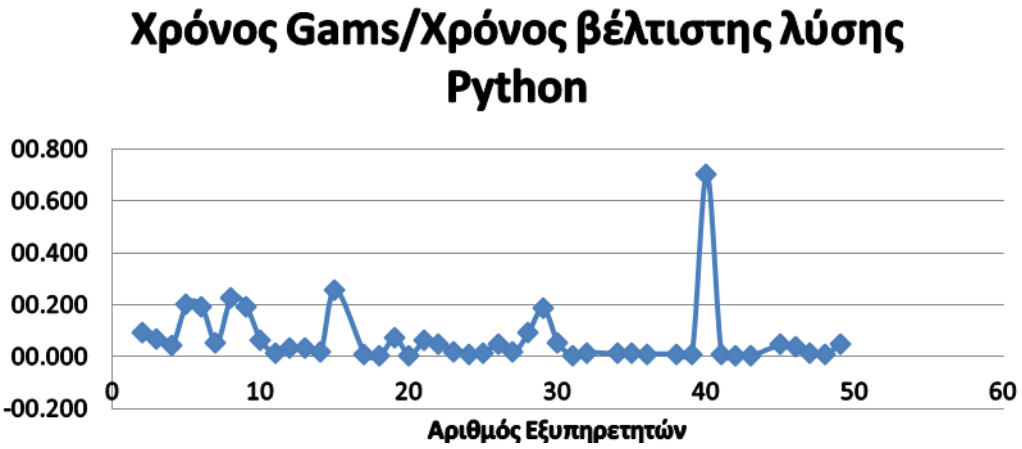
Εικόνα 27: Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 28 : Χρόνος CPU για τον υπολογισμό του ορίου της Python



Εικόνα 29 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 30 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό εξυπηρετητών

4.3.3) Αυξανόμενος Αριθμός Υπηρεσιών

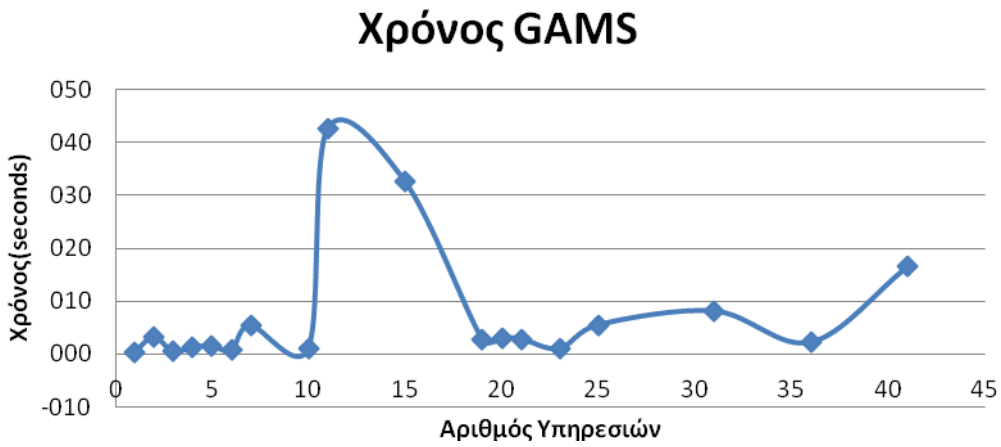
Στην Εικόνα 31 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Εκτελέσαμε περιπτώσεις για 1 έως 49 υπηρεσίες. Ο αριθμός των υπηρεσιών σχεδιάζονται στον άξονα x με μέγιστο το 49 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Σε 31 από τις 49 περιπτώσεις ο χρόνος υπολογισμού στο GAMS ξεπέρασε το όριο των 120 δευτερολέπτων και δεν σχεδιάζεται στα σχήματα. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (δηλαδή για ανενεργό εξυπηρετητή) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητής.

Στην Εικόνα 32 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό υπηρεσιών.

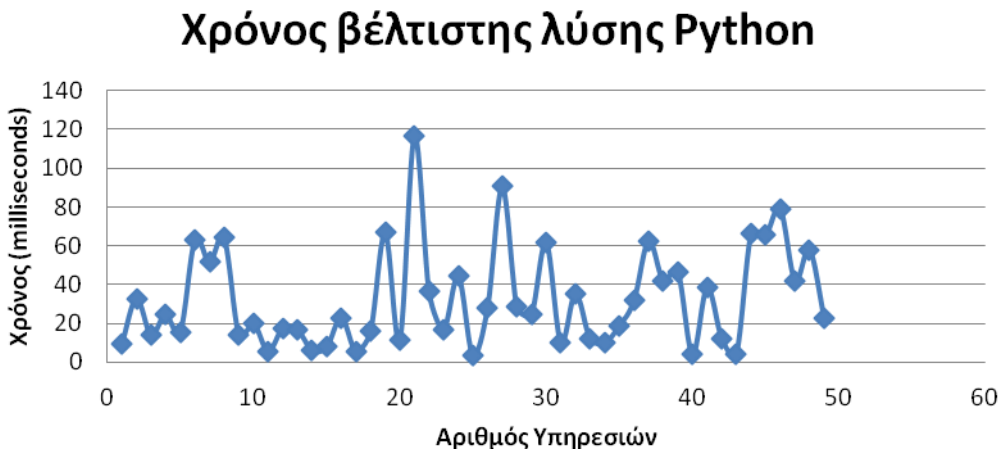
Στην Εικόνα 33 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό υπηρεσιών.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις, εκτός από μια περίπτωση που είναι 1,0126.

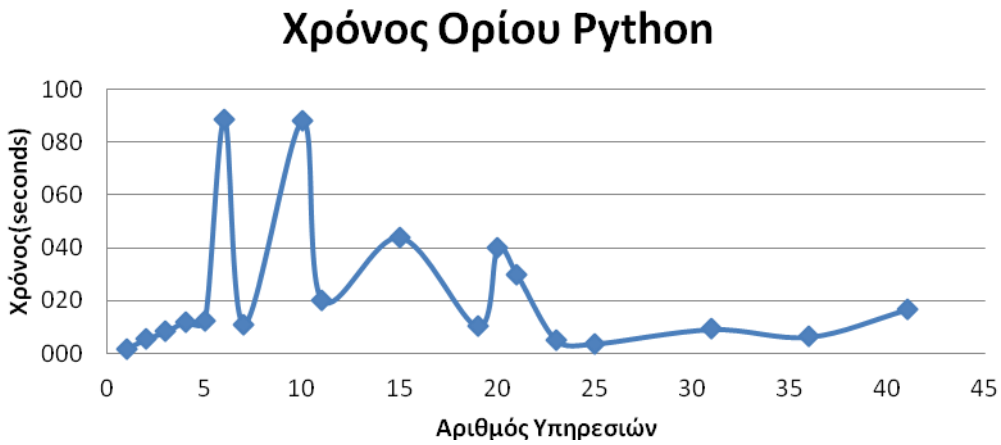
Όσον αφορά τους χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 34 και την Εικόνα 35 φαίνονται οι λόγοι των χρόνων υπολογισμού.



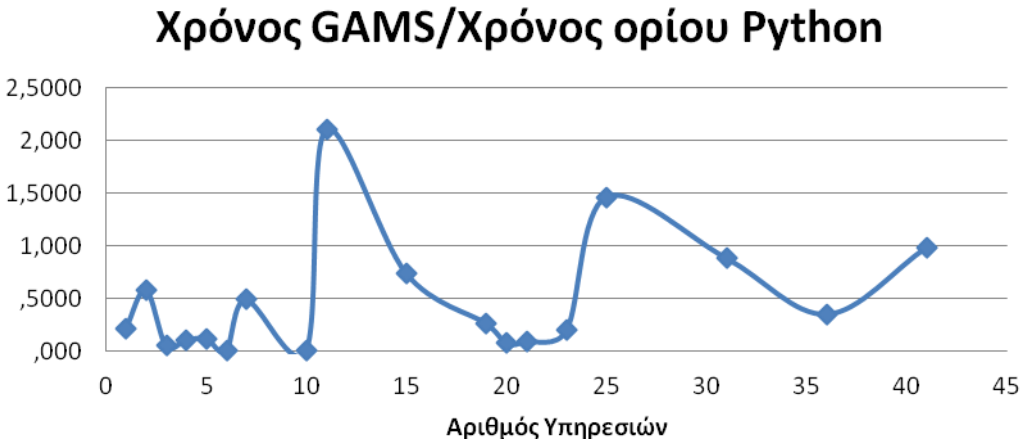
Εικόνα 31 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό υπηρεσιών



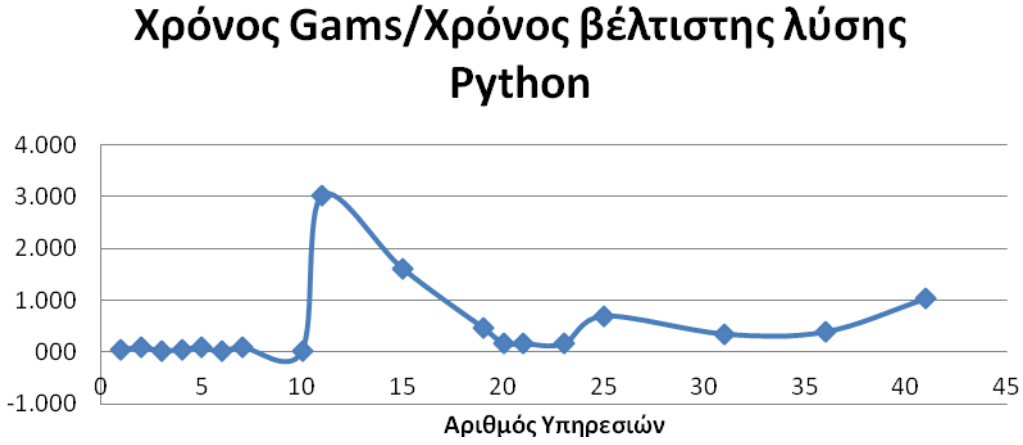
Εικόνα 32 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 33 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 34 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 35 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό υπηρεσιών

4.3.4)Αυξανόμενος Αριθμός Δικτύων

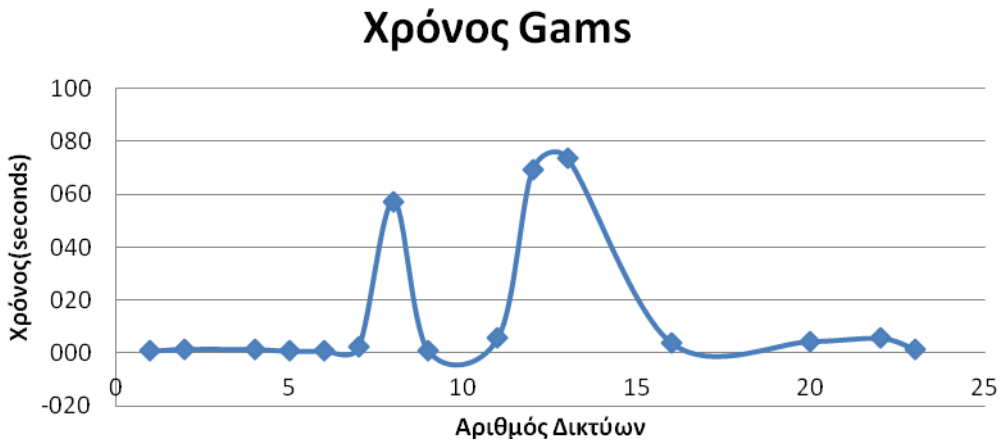
Στην Εικόνα 36 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Εκτελέσαμε περιπτώσεις για 1 έως 23 δίκτυα. Ο αριθμός των δικτύων σχεδιάζονται στον άξονα x με μέγιστο το 23 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Σε 8 από τις 23 περιπτώσεις ο χρόνος υπολογισμού στο GAMS ξεπέρασε το όριο των 120 δευτερόλεπτα και δεν σχεδιάζονται στα σχήματα. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (δηλαδή για ανενεργό εξυπηρετητή) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητής

Στην Εικόνα 37 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό δικτύων.

Στην Εικόνα 38 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό δικτύων.

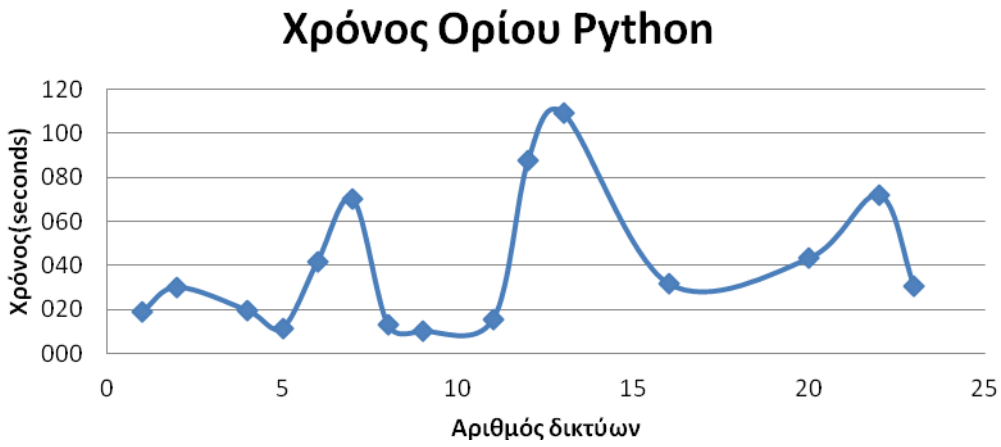
Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις.

Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 39 και την Εικόνα 40 φαίνονται οι λόγοι των χρόνων υπολογισμού.

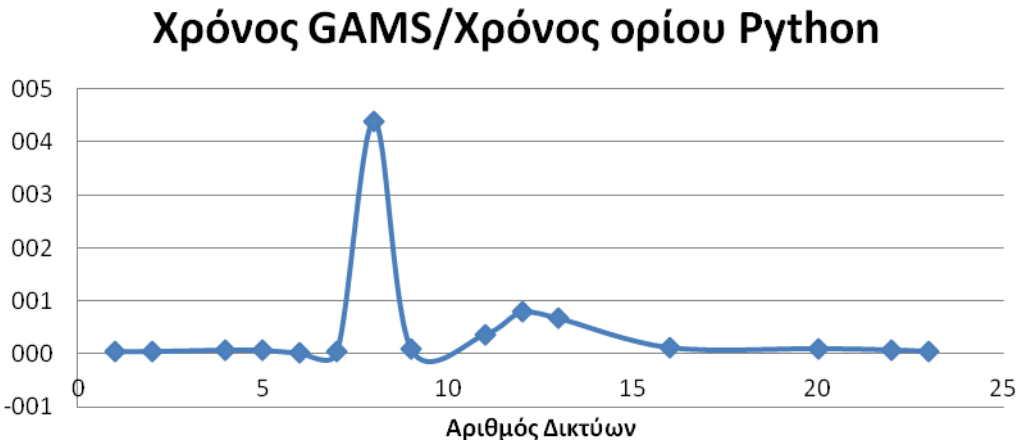


Εικόνα 36 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό δικτύων

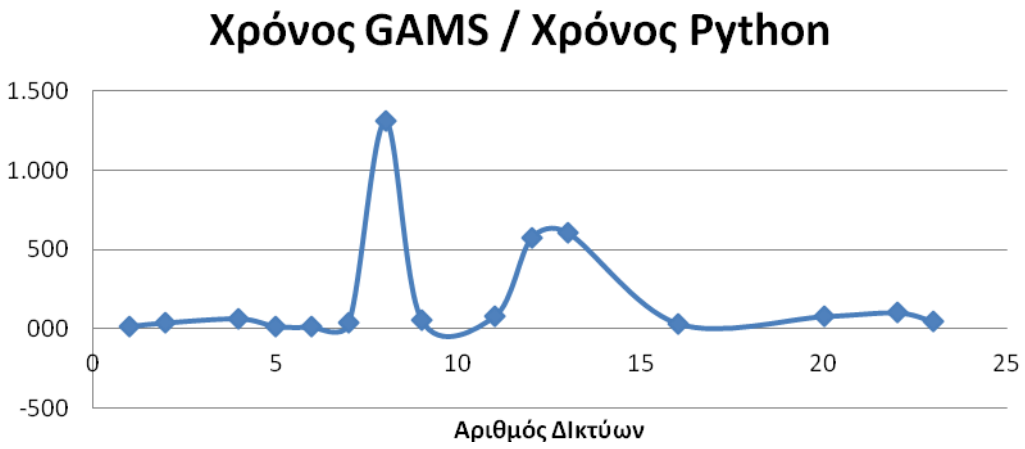
Εικόνα 37 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό δικτύων



Εικόνα 38 : Χρόνος CPU για τον υπολογισμό του ορίου της Python για αυξανόμενο αριθμό Δικτύων



Εικόνα 39 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό δικτύων



Εικόνα 40 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό δικτύων

4.4) Τρίτη Κατηγορία Περιπτώσεων

Θεωρούμε ότι έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια, 8 εξυπηρετητές σε 2 δίκτυα. Όσον αφορά τις υπόλοιπες μετρικές παίρνουν (ενδεχομένως[2]) διαφορετική τυχαία τιμή για όλα τα συστατικά, όλες τις υπηρεσίες, όλους τους εξυπηρετητές και όλα τα δίκτυα. Η τυχαία αυτή τιμή προκύπτει από τη βιβλιοθήκη random της Python.

- Πρόστιμο διαθεσιμότητας = 100 έως και 200 με βήμα 20 διαφορετικό για κάθε υπηρεσία
- Διαθεσιμότητα = 0,9 έως και 1 με βήμα 0,1 διαφορετικό για κάθε υπηρεσία
- κέρδος= 500 έως 1000 με βήμα 10 διαφορετικό για κάθε υπηρεσία
- βασικές απαιτήσεις δικτύου = ακέραιος από 1 έως 20 διαφορετικός για κάθε συστατικό κάθε υπηρεσίας.
- βασικές απαιτήσεις εικονικών μηχανών = ακέραιος από 1 έως 3 διαφορετικός για κάθε συστατικό κάθε υπηρεσίας
- κόστος χρήσης = ακέραιος από 1 έως 20 διαφορετικός για κάθε εξυπηρετητή.
- Κόστος ενεργοποίησης = ακέραιος από 10 έως 30 διαφορετικός για κάθε εξυπηρετητή
- ελαστικές απαιτήσεις δικτύου = ακέραιος από 1 έως 20 διαφορετικός για κάθε συστατικό κάθε υπηρεσίας
- ελαστικές απαιτήσεις εικονικών μηχανών = ακέραιος από 1 έως 15 διαφορετικός για κάθε συστατικό κάθε υπηρεσίας
- μέγιστος αριθμός επεξεργαστών = 16 για όλους τους εξυπηρετητές
- χωρητικότητα δικτύου = τυχαία επιλογή ανάμεσα σε 500,1000,2000 διαφορετική για κάθε δίκτυο
- δεσμευμένοι επεξεργαστές = ακέραιος από 0 έως 15 διαφορετικός για όλους τους εξυπηρετητές
- virtual_cpus = 1 για κάθε συστατικό κάθε υπηρεσίας
- τοπολογία δικτύου =

Αυξανόμενος Αριθμός Συστατικών

Στην Εικόνα 41 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των συστατικών σχεδιάζονται στον άξονα x με μέγιστο το 20 ενώ ο χρόνος cpu που

[2] Βλέπε σημείωση 1 σελίδα 36. Το ενδεχομένως θα παραλείπεται στη συνέχεια

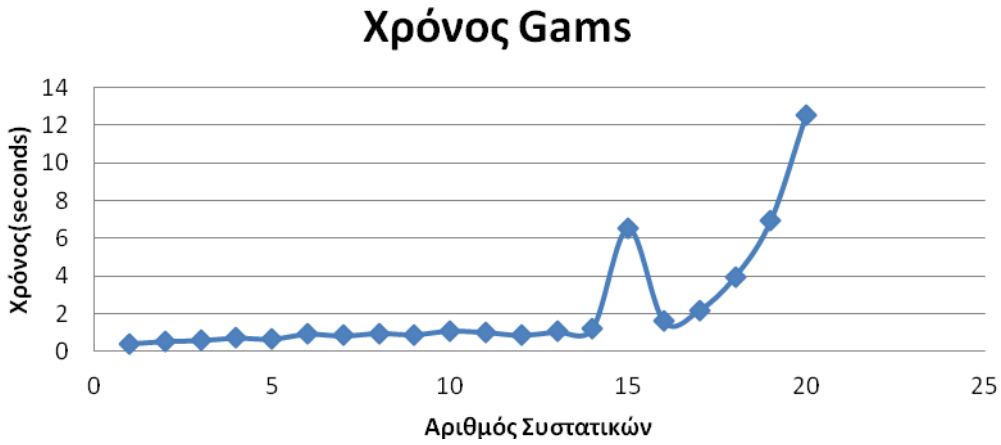
καταναλώθηκε σχεδιάζεται στον άξονα y. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (δηλαδή για ανενεργό εξυπηρετητή) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητής.

Στην Εικόνα 42 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθένα από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό συστατικών.

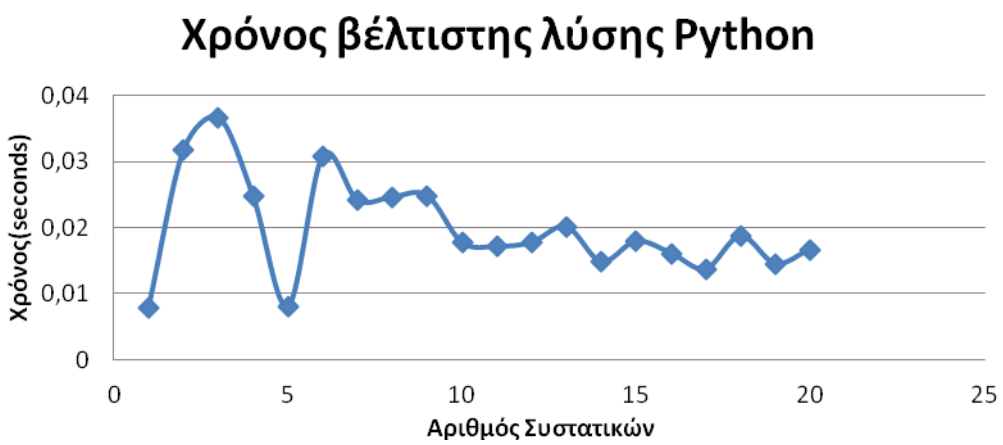
Στην Εικόνα 43 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό συστατικών.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις

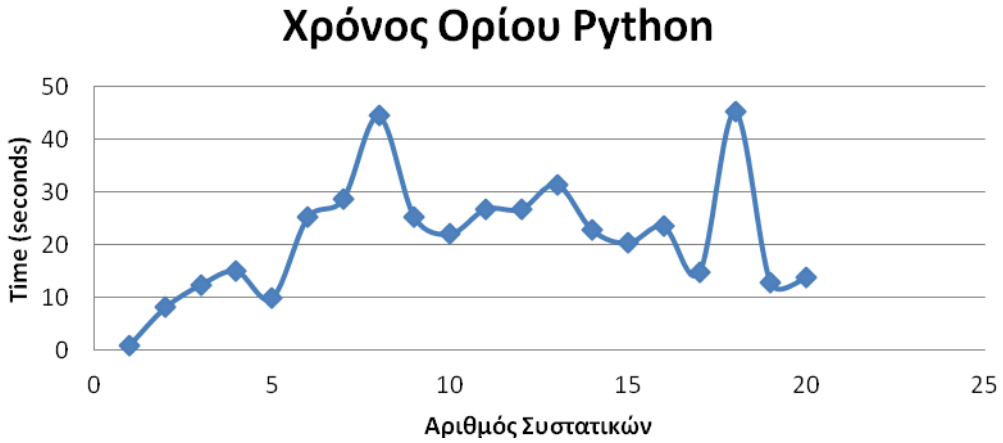
Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 44 και την Εικόνα 45 φαίνονται οι λόγοι των χρόνων υπολογισμού.



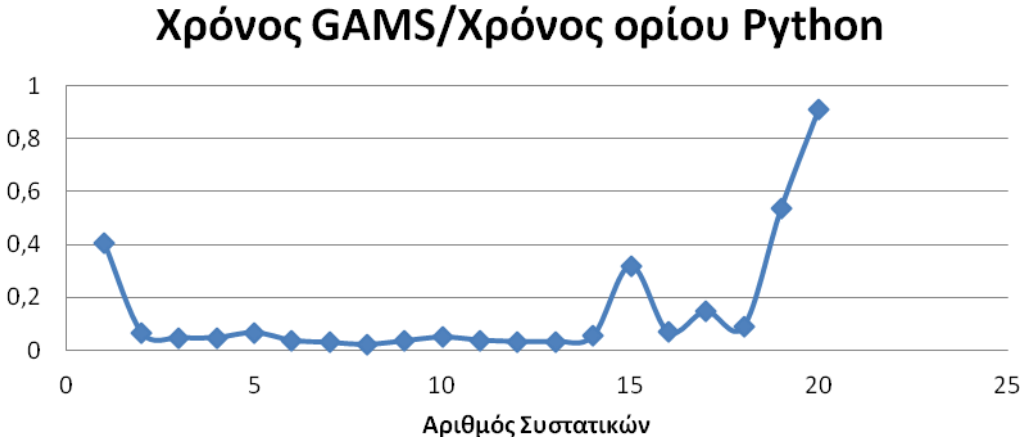
Εικόνα 41 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό συστατικών



Εικόνα 42 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό συστατικών



Εικόνα 43 : Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό συστατικών



Εικόνα 44 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών

Εικόνα 45 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών

Αυξανόμενος Αριθμός Εξυπηρετητών

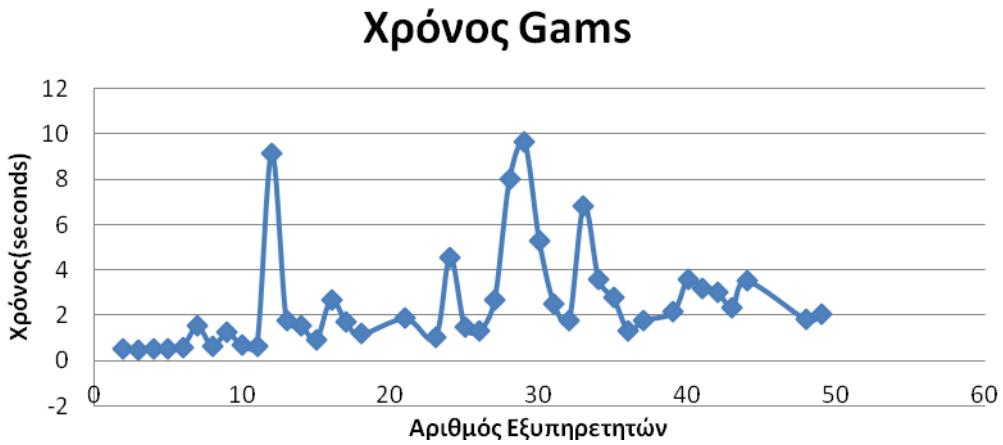
Στην Εικόνα 46 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των εξυπηρετητών σχεδιάζονται στον άξονα x με μέγιστο το 49 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (δηλαδή για ανενεργό εξυπηρετητή) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητής. Σε 7 από τις 49 περιπτώσεις και συγκεκριμένα για 19, 20, 22, 38, 45, 46, 47 εξυπηρετητές, ο χρόνος CPU του GAMS ξεπέρασε τα 120 δευτερόλεπτα και η εκτέλεση τερματίστηκε. Οι παραπάνω 7 περιπτώσεις δεν εμφανίζονται στα σχήματα.

Στην Εικόνα 47 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό εξυπηρετητών.

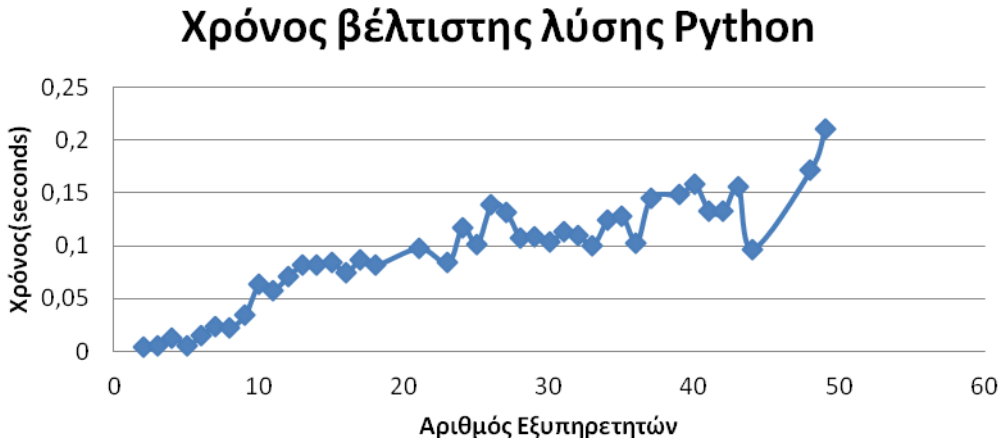
Στην Εικόνα 48 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό εξυπηρετητών.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλα τις περιπτώσεις

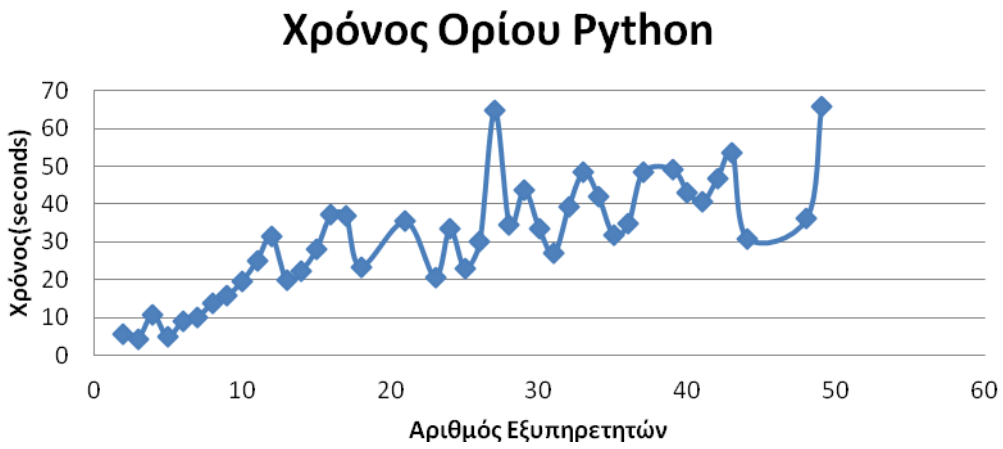
Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 49 και την Εικόνα 50 φαίνονται οι λόγοι των χρόνων υπολογισμού.



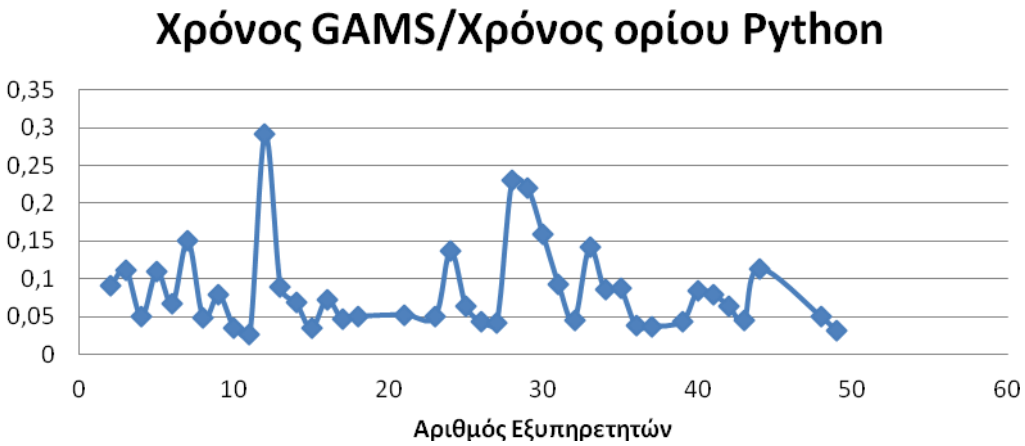
Εικόνα 46 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών



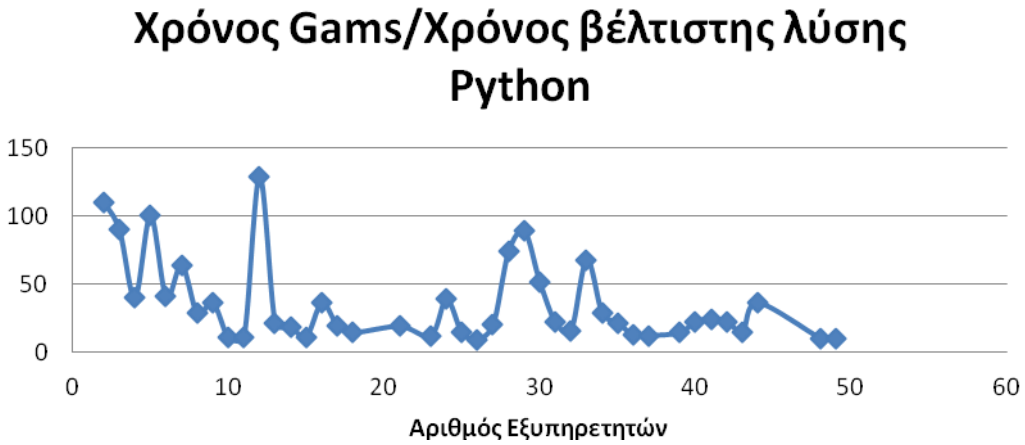
Εικόνα 47 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 48: Χρόνος CPU για τον υπολογισμό του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 49 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 50 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό εξυπηρετητών

Αυξανόμενος Αριθμός Υπηρεσιών

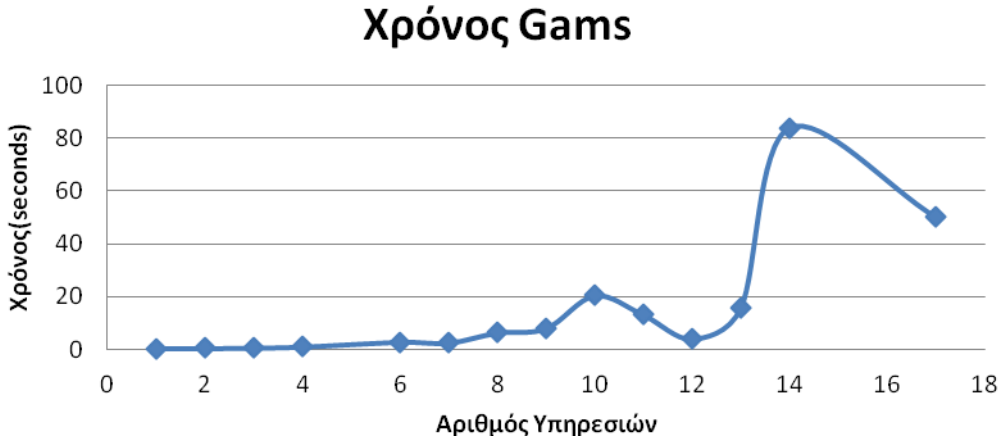
Στην Εικόνα 51 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των υπηρεσιών σχεδιάζονται στον άξονα x με μέγιστο το 17 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (δηλαδή για ανενεργό host) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητές. Σε 35 από τις 49 περιπτώσεις ο χρόνος CPU του GAMS ξεπέρασε τα 120 δευτερόλεπτα και η εκτέλεση τερματίστηκε. Οι παραπάνω 35 περιπτώσεις δεν εμφανίζονται στα σχήματα.

Στην Εικόνα 52 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό υπηρεσιών.

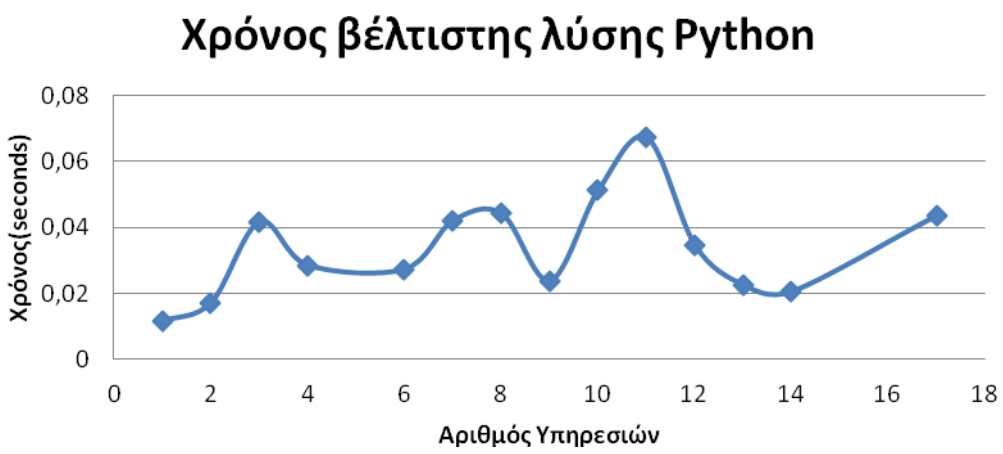
Στην Εικόνα 53 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό υπηρεσιών.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλα τις περιπτώσεις.

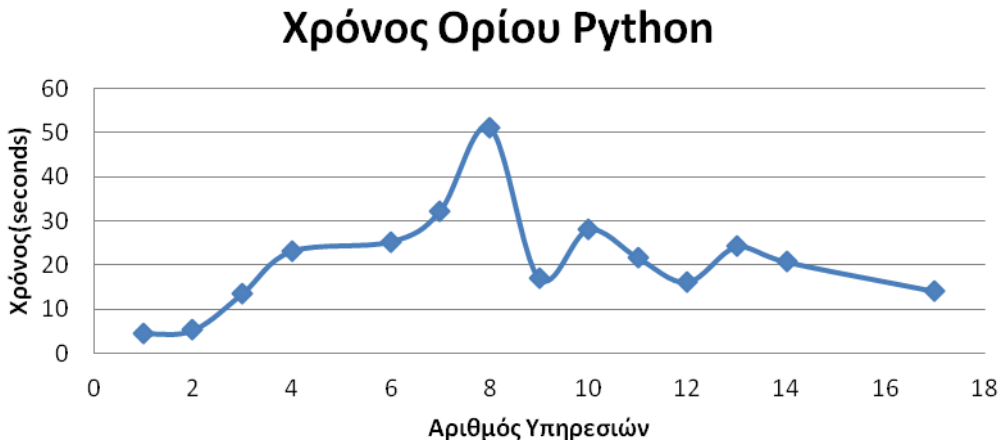
Όσον αφορά του χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 55 και την Εικόνα 56Εικόνα 55 φαίνονται οι λόγοι των χρόνων υπολογισμού.



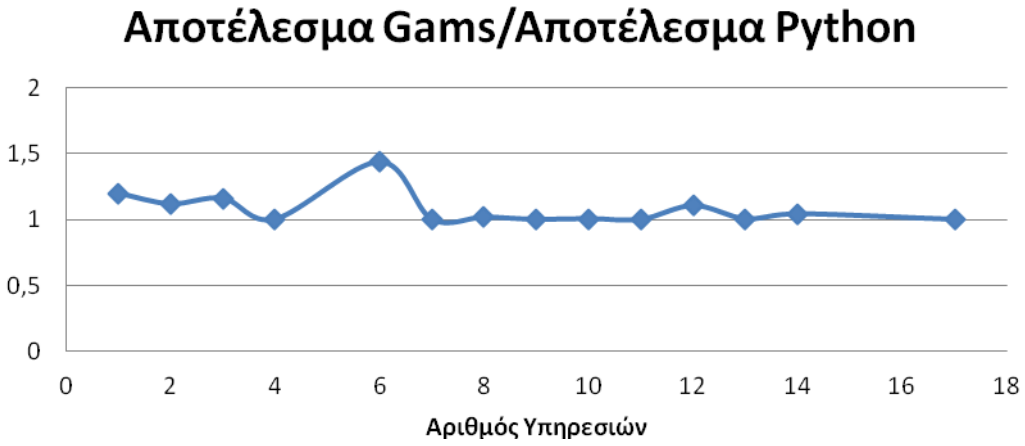
Εικόνα 51 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON



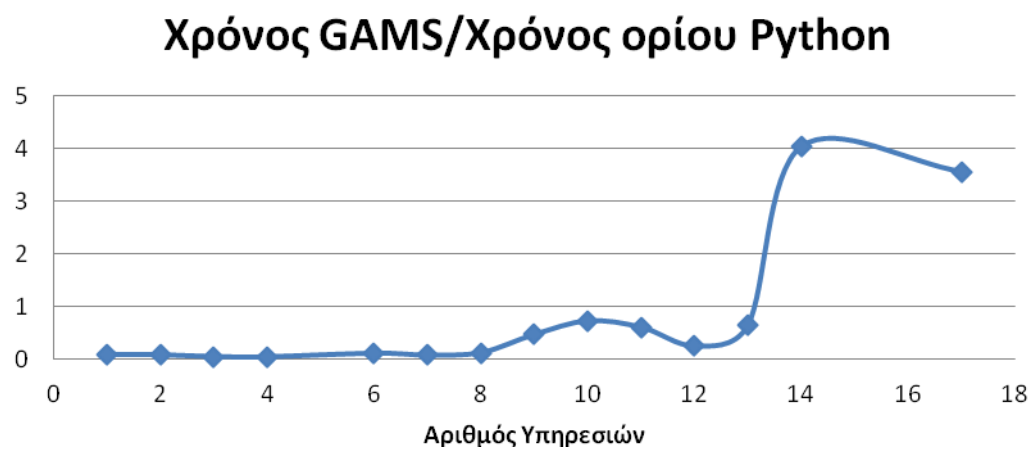
Εικόνα 52 :Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python



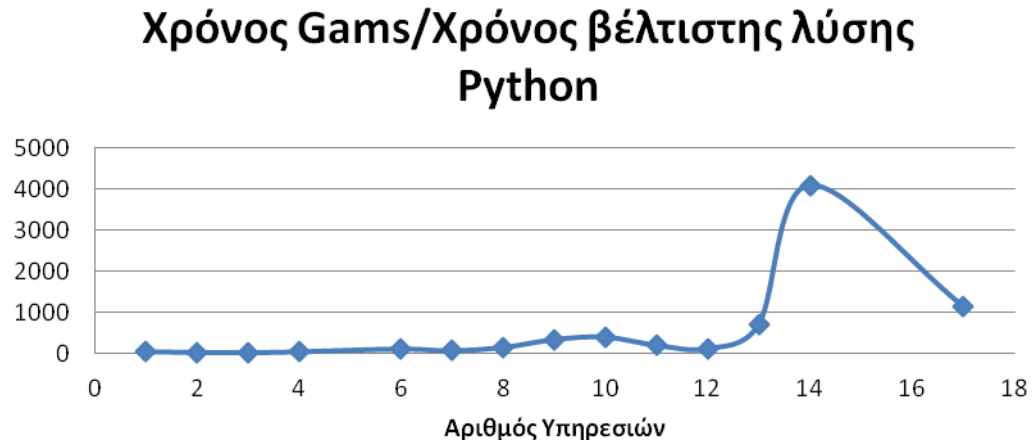
Εικόνα 53: Χρόνος CPU για τον υπολογισμό του ορίου της Python



Εικόνα 54: Λόγος αποτελέσματος βέλτιστης λύσης από το BARON προ το αποτέλεσμα της αντικειμενικής συνάρτησης της Python



Εικόνα 55 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python

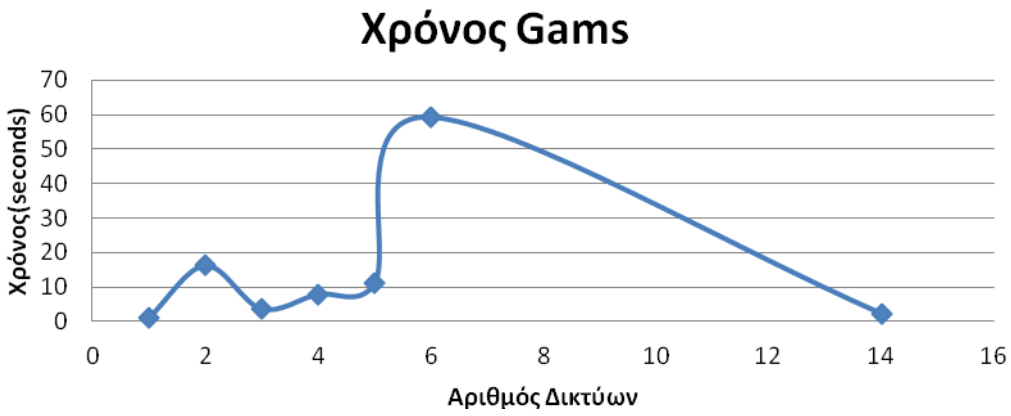


Εικόνα 56 : Λόγος του χρόνοy υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python

Αυξανόμενος Αριθμός Δικτύων

Στην Εικόνα 57 φαίνεται ο χρόνος CPU που καταναλώθηκε από το BARON για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών. Ο αριθμός των δικτύων σχεδιάζονται στον άξονα x με μέγιστο το 14 ενώ ο χρόνος που καταναλώθηκε σχεδιάζεται στον άξονα y. Οι δεσμευμένες cpus σε κάθε εξυπηρετητή καθορίζονται από μια random επιλογή και είναι από 0 (δηλαδή για ανενεργό εξυπηρετητή) έως 15 στις 16 cpus που έχει κάθε εξυπηρετητής. Σε 16 από τις 23 περιπτώσεις ο χρόνος CPU του GAMS ξεπέρασε τα 120 δευτερόλεπτα και η εκτέλεση τερματίστηκε. Οι παραπάνω 16 περιπτώσεις δεν εμφανίζονται στα σχήματα.

Στην Εικόνα 58 φαίνεται ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί η βέλτιστη λύση για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό δικτύων.

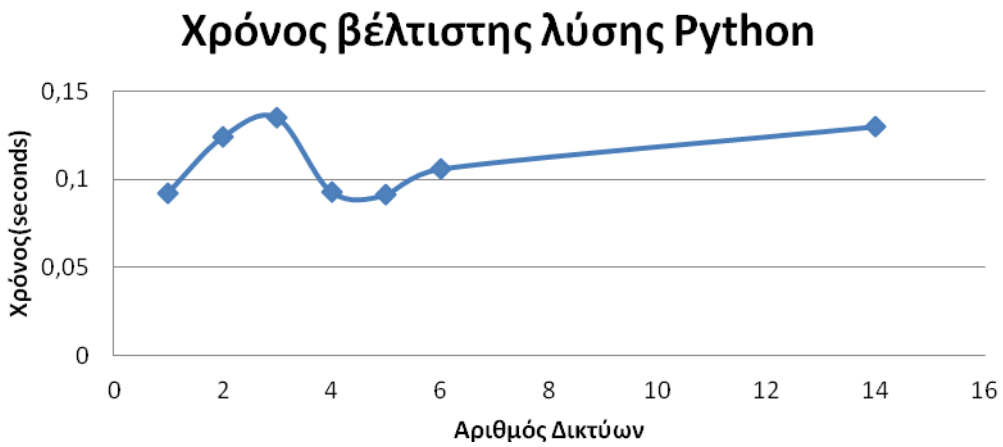


Εικόνα 57: Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON

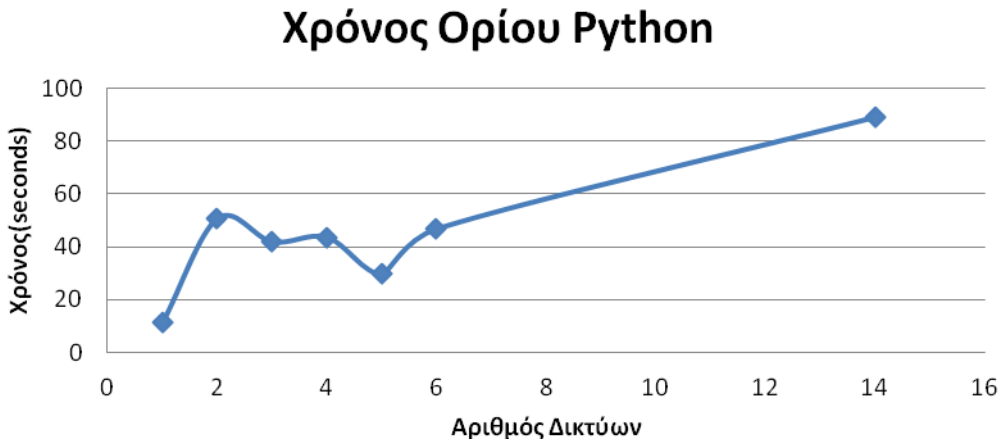
Στην Εικόνα 59 φαίνεται όμοια με παραπάνω ο χρόνος CPU που καταναλώθηκε από την Python για να ευρεθεί το όριο για καθεμία από τις εξεταζόμενες περιπτώσεις διαφορετικών μεγεθών με αυξανόμενο αριθμό δικτύων.

Παρατηρήσαμε επίσης ότι το αποτέλεσμα της αντικειμενικής συνάρτησης του GAMS είναι ίσο με το αποτέλεσμα της αντικειμενικής συνάρτησης της Python και με το όριο της Python για όλες τις περιπτώσεις.

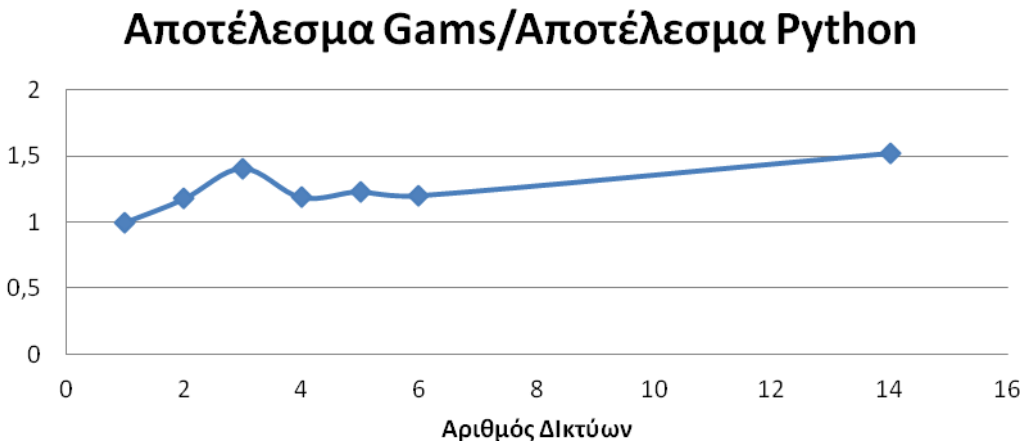
Όσον αφορά τους χρόνους υπολογισμού των παραπάνω αποτελεσμάτων στην Εικόνα 61 και την Εικόνα 62 φαίνονται οι λόγοι των χρόνων υπολογισμού.



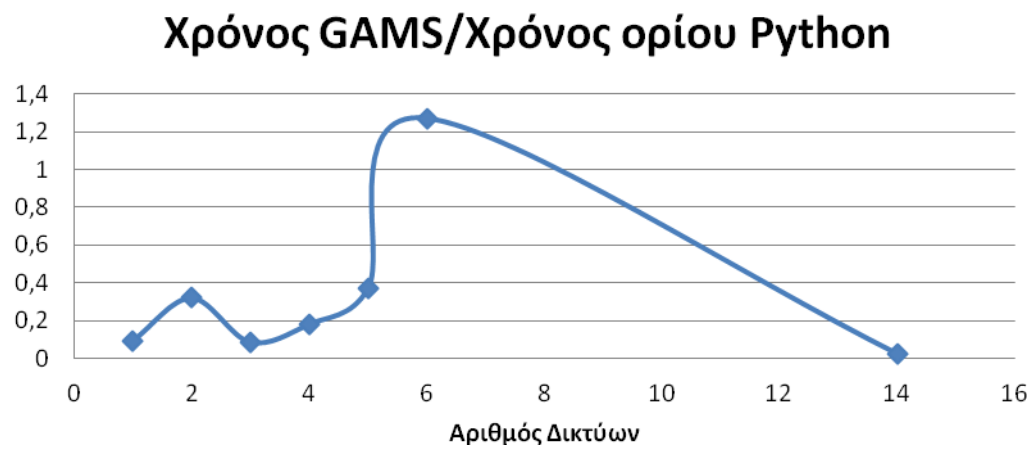
Εικόνα 58 : Χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από τη Python



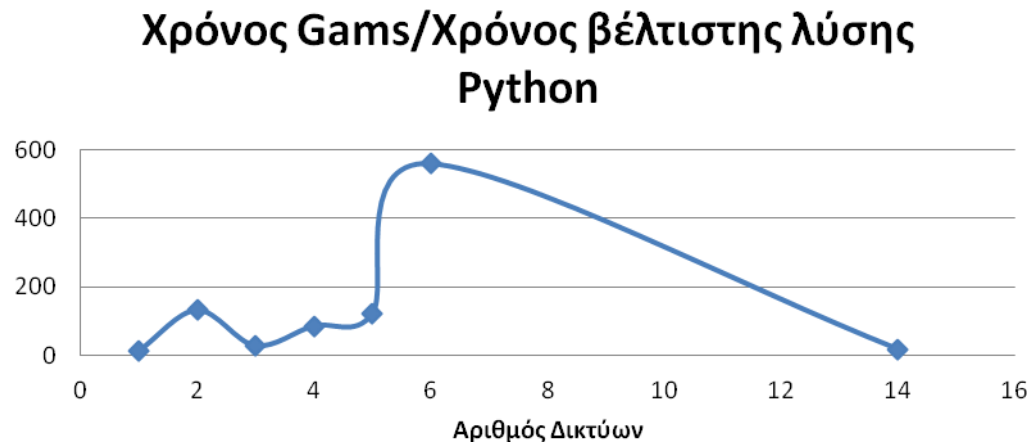
Εικόνα 59: Χρόνος CPU για τον υπολογισμό του ορίου της Python



Εικόνα 60: Λόγος αποτελέσματος βέλτιστης λύσης από το BARON προ το αποτέλεσμα της αντικειμενικής συνάρτησης της Python



Εικόνα 61: Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python



Εικόνα 62: Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python

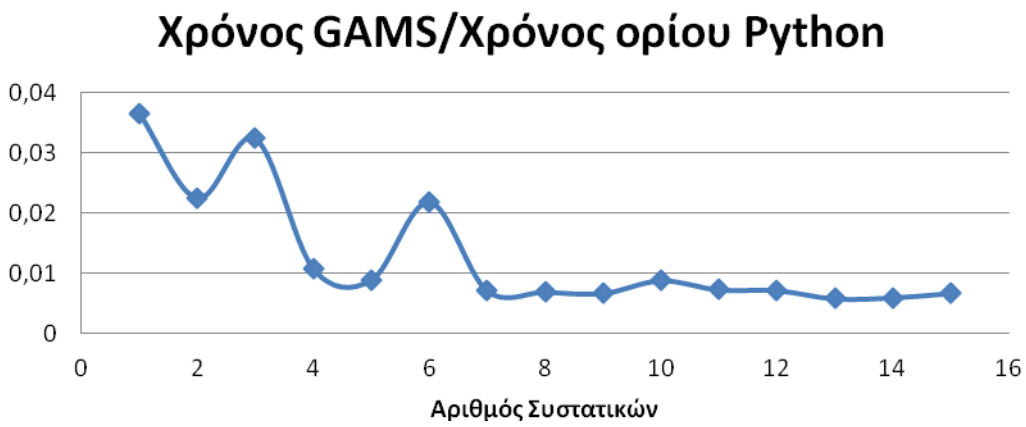
4.5) Τέταρτη κατηγορία περιπτώσεων

θεωρούμε την περίπτωση όπου έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια, 8 εξυπηρετητές σε 2 δίκτυα. Για τα τέσσερα μεγέθη (υπηρεσίες, συστατικά, εξυπηρετητές, δίκτυα) διατηρούμε σταθερό τον αριθμό των τριών εκ των τεσσάρων με το τέταρτο να μεταβάλλεται ώστε να διαπιστώσουμε την επίδραση της μεταβολής. Όλες οι υπόλοιπες μετρικές παίρνουν την ίδια σταθερή τιμή για όλα τα συστατικά, όλους τους εξυπηρετητές, όλες τις υπηρεσίες και τα όλα δίκτυα αντίστοιχα. Εκτελούμε μια φορά κάθε στιγμιότυπο αφού όλες οι μετρικές παίρνουν την ίδια σταθερή τιμή. Εκτελώντας περισσότερες φορές ένα στιγμιότυπο, θα προέκυπταν πάλι τα ίδια αποτελέσματα. Στην περίπτωση αυτή θα εξεταστούν μόνο οι λόγοι των χρόνων για να διαπιστωθεί κυρίως πόσες φορές είναι ταχύτερο το μοντέλο της Python από το μοντέλο του GAMS.

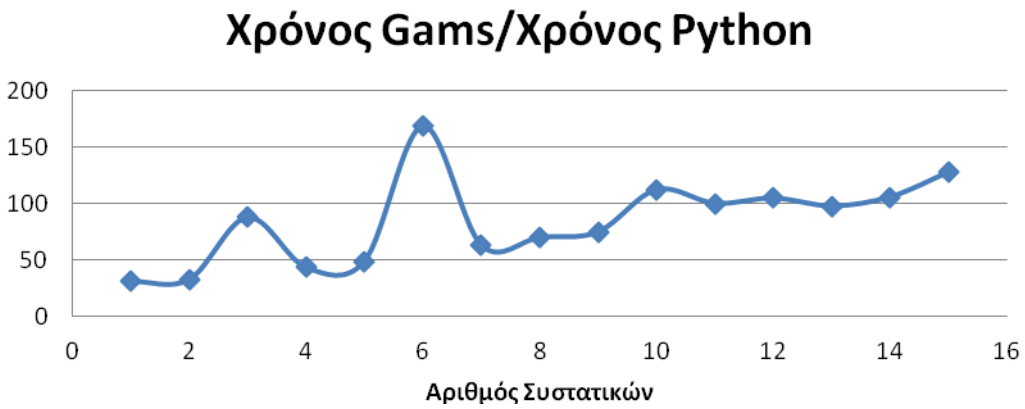
- Πρόστιμο διαθεσιμότητας = 800 για κάθε υπηρεσία
- διαθεσιμότητα = 0.95 για κάθε υπηρεσία
- κέρδος =1000 για κάθε υπηρεσία
- βασικές απαιτήσεις δικτύου = 0 για κάθε συστατικό κάθε υπηρεσίας
- βασικές απαιτήσεις εικονικών μηχανών = 1 για κάθε συστατικό κάθε υπηρεσίας
- κόστος χρήσης = 5 για κάθε εξυπηρετητή
- κόστος ενεργοποίησης = 10 για κάθε εξυπηρετητή
- ελαστικές απαιτήσεις δικτύου = 100 για κάθε συστατικό κάθε υπηρεσίας
- ελαστικές απαιτήσεις εικονικών μηχανών = 2 για κάθε συστατικό κάθε υπηρεσίας
- μέγιστος αριθμός επεξεργαστών = 16 για κάθε εξυπηρετητή
- χωρητικότητα δικτύου = 16000 για κάθε δίκτυο
- δεσμευμένοι επεξεργαστές = 11 για κάθε εξυπηρετητή
- όχι περιοριστικοί κανόνες
- τοπολογία δικτύου = ίσος αριθμός εξυπηρετητών σε κάθε δίκτυο (σε περίπτωση περιττού αριθμού εξυπηρετητών το ένα δίκτυο έχει έναν παραπάνω εξυπηρετητή)
- εμπιστοσύνη = 1

Αυξανόμενος Αριθμός Συστατικών

Παρακάτω έχουμε τα διαγράμματα του χρόνου GAMS προς το χρόνο Python καθώς και του χρόνου GAMS προς το χρόνο ορίου Python με αυξανόμενο αριθμό συστατικών. Παρατηρούμε ότι ο Χρόνος GAMS είναι μέχρι και 168,44 φορές μεγαλύτερος από το χρόνο Python.

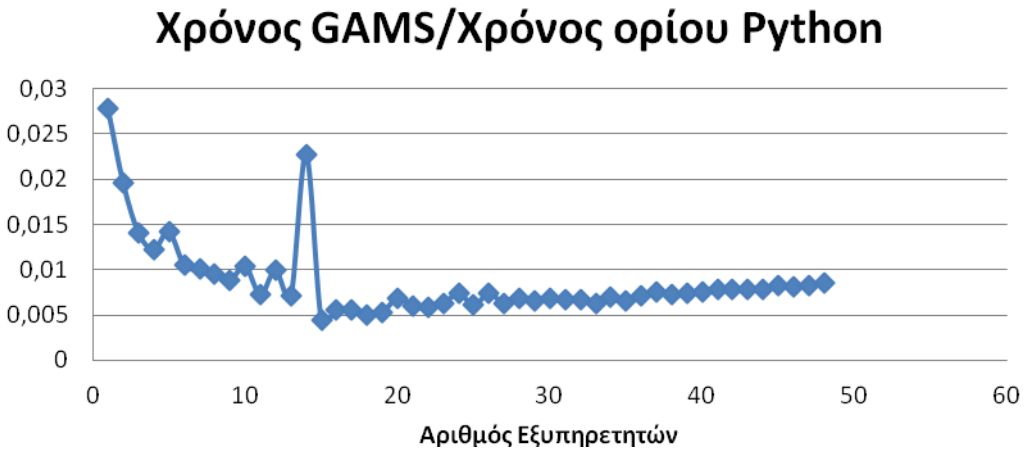


Εικόνα 63 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών

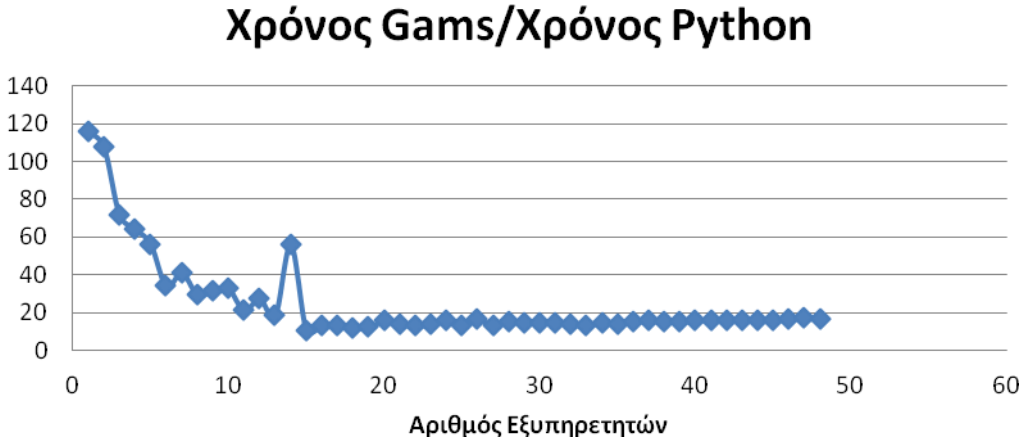


Εικόνα 64 : Λόγος των χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών

Αυξανόμενος Αριθμός Εξυπηρετητών



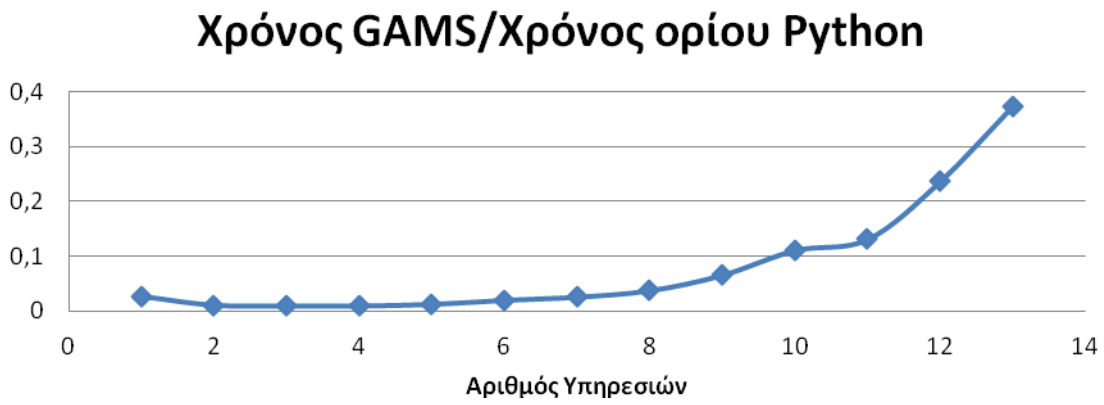
Εικόνα 65 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών



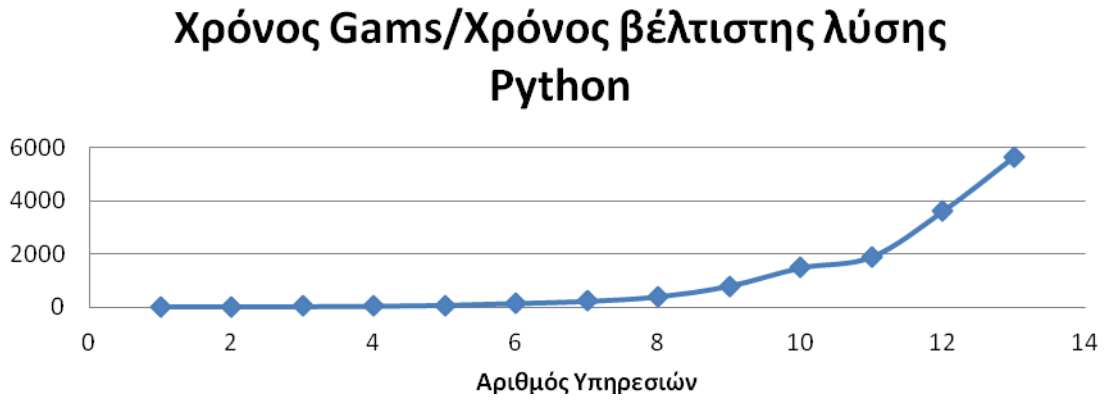
Εικόνα 66 : Λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό εξυπηρετητών

Αυξανόμενος Αριθμός Υπηρεσιών

Παρατηρείται ότι για αυξανόμενο αριθμό υπηρεσιών οι λόγοι των χρόνων αυξάνονται και αυτό οφείλεται στην μεγαλύτερη αύξηση του χρόνου GAMS σε σχέση με αυτή του χρόνου Python ή του χρόνου ορίου Python.

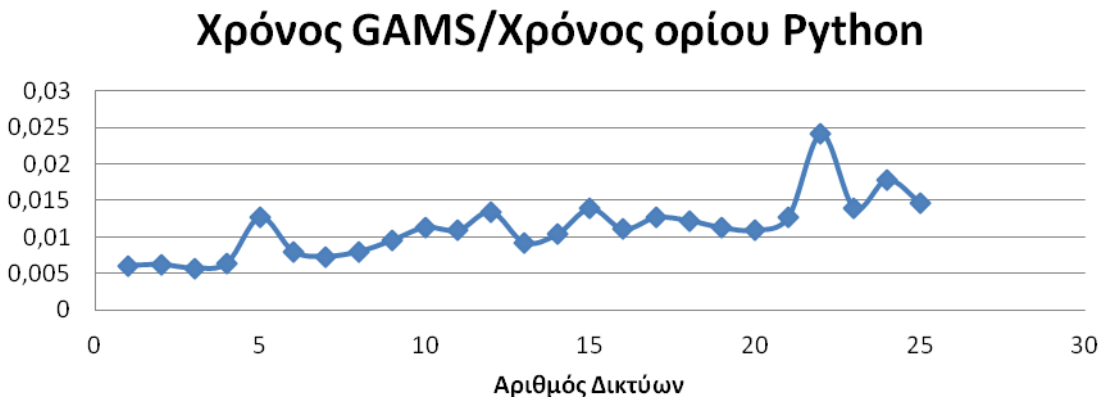


Εικόνα 67 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών

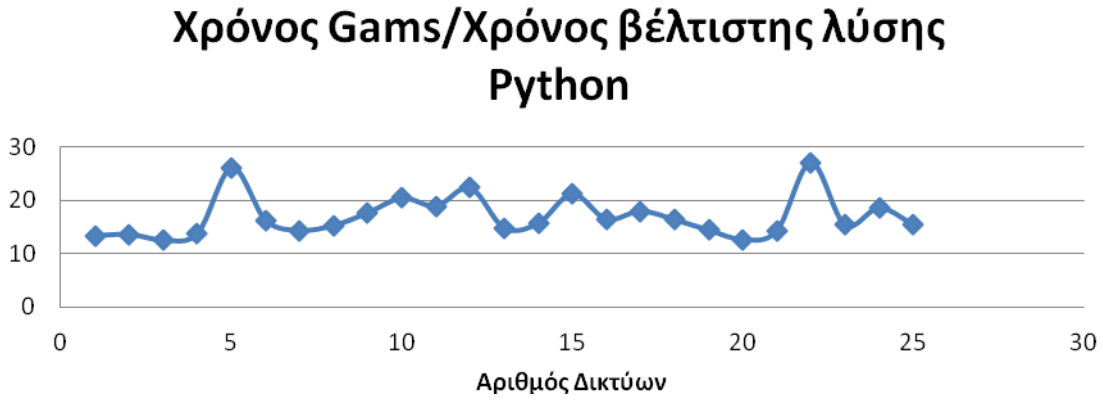


Εικόνα 68 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό υπηρεσιών

Αυξανόμενος Αριθμός Δικτύων



Εικόνα 69 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό δικτύων



Εικόνα 70 : Λόγος του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό δικτύων

4.6) Πέμπτη κατηγορία περιπτώσεων με πολλές εκτελέσεις

Θεωρούμε την περίπτωση όπου έχουμε 3 υπηρεσίες με 4 συστατικά η κάθε μια, 8 εξυπηρετητές σε 2 δίκτυα. Για κάθε στιγμιότυπο (συγκεκριμένος αριθμό υπηρεσιών, συστατικών, εξυπηρετητών και δικτύων) γίνονται 30 εκτελέσεις απ' όπου προκύπτουν στατιστικά στοιχεία, πίνακες και διαγράμματα με μεγαλύτερη ακρίβεια. Στις περιπτώσεις με αυξανόμενο αριθμό δικτύων θεωρήθηκαν 24 εξυπηρετητές.

- Πρόστιμο διαθεσιμότητας = (βασικές και ελαστικές απαιτήσεις σε εικονικές μηχανές) * 10
- διαθεσιμότητα = κατανομή Gauss με μέση τιμή 0,95 και διασπορά 0,015 με μέγιστο το 1 και (ενδεχομένως[3]) διαφορετικό για κάθε υπηρεσία.
- Κέρδος = (βασικές και ελαστικές απαιτήσεις σε εικονικές μηχανές) * 10 * κατανομή Gauss (μέσης τιμής 1,2 και διασποράς 0,1)
- Βασικές εικονικές μηχανές = τυχαία επιλογή ανάμεσα στο 1,2,3 για κάθε συστατικό κάθε υπηρεσίας
- Ελαστικές εικονικές μηχανές = κατανομή gamma(2,2) για κάθε συστατικό κάθε υπηρεσίας
- Βασικές απαιτήσεις δικτύου = 0 για κάθε συστατικό κάθε υπηρεσίας
- Ελαστικές απαιτήσεις δικτύου = τυχαία επιλογή ανάμεσα στο 50, 100, 200 για κάθε συστατικό κάθε υπηρεσίας
- Μέγιστος αριθμός επεξεργαστών = 16 για κάθε εξυπηρετητή
- Δεσμευμένοι επεξεργαστές = 50% πιθανότητα για τιμή μηδέν και 50% στην κατανομή (16-gamma(2,2)) με μέγιστο το 15 διαφορετικό για κάθε εξυπηρετητή
- Χωρητικότητα δικτύου = τυχαία επιλογή ανάμεσα στο 8000, 16000, 24000 διαφορετικό για κάθε εξυπηρετητή
- virtual_cpus= 1 για κάθε συστατικό κάθε υπηρεσίας
- κόστος χρήσης = ακολουθεί εκθετική κατανομή με τιμές στο διάστημα [2,6] για κάθε εξυπηρετητή
- κόστος ενεργοποίησης = επιλογή ανάμεσα στο 5,10,15 για κάθε εξυπηρετητή.

[3] Βλέπε σημείωση 1 σελ. 36 Το "ενδεχομένως" παραλείπεται στο εξής

4.6.1)Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό εξυπηρετών

Αρχικά αυξάνουμε τον αριθμό των εξυπηρετητών με βήμα 1 ξεκινώντας από 10 εξυπηρετητές δηλαδή εκτελούμε την περίπτωση με 3 υπηρεσίες, 4 συστατικά, 10 εξυπηρετητές, 2 δίκτυα. Θα εκτελέσουμε 30 περιπτώσεις με 3 υπηρεσίες, 4 συστατικά, 10 εξυπηρετητές, 2 δίκτυα και από τα αποτελέσματα εξάγουμε στατιστικές πληροφορίες για τις ποσότητες και τους λόγους που προαναφέραμε. Ενδεικτικά για 10 εξυπηρετητές προκύπτει ο παρακάτω πίνακας:

	Average	min	max
gams_time/bound_time	0,006418289	0,003536198	0,011940628
gams_time/best_tme	15,25884612	8,978021978	27,66509434
python_best_time	0,048448276	0,0309	0,0752
gams_time	0,710641379	0,6078	1,173

Πίνακας 2 : Στατιστικά Στοιχεία για 30 εκτελέσεις με 10 εξυπηρετητές

Αντίστοιχα εκτελούμε 30 περιπτώσεις για 11 εξυπηρετητές εξάγοντας τον πίνακα που προκύπτει, για 12 εξυπηρετητές κ.ο.κ. μέχρι και 30 εξυπηρετητές.

Με αυτόν τον τρόπο προκύπτουν οι παρακάτω πίνακες και σχήματα.
Στις περιπτώσεις όπου ο χρόνος CPU GAMS έχει ξεπεράσει το όριο των 120 δευτερολέπτων η εκτέλεση διακόπηκε και τα αποτελέσματα αυτών δεν συνυπολογίζονται στους πίνακες και τα σχήματα.

Στους πίνακες που ακολουθούν παρουσιάζονται τα συγκεντρωτικά στατιστικά στοιχεία:

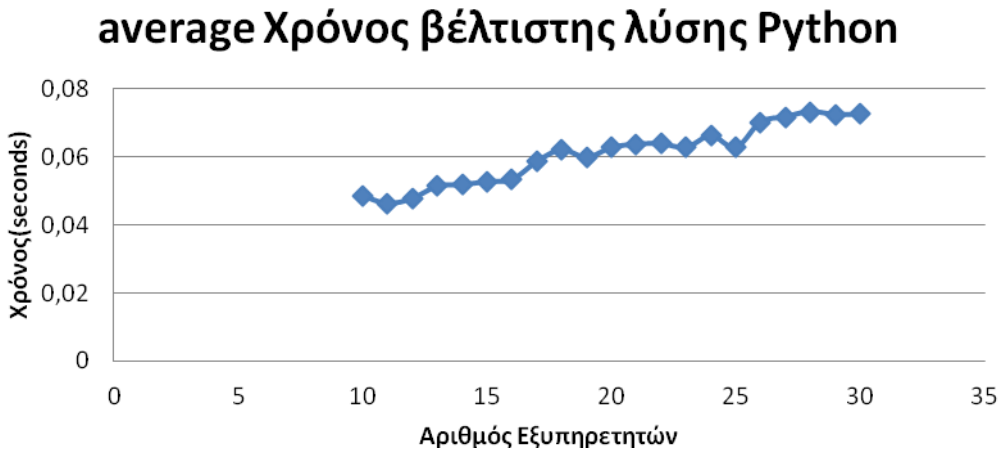
- Στον Πίνακας 3 για τον χρόνο υπολογισμού της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό εξυπηρετητών.
- Στον Πίνακας 4 για τον υπολογισμό της βέλτιστης λύσης από τον BARON με αυξανόμενο αριθμό εξυπηρετητών
- Στον Πίνακας 5 για τον λόγο του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό εξυπηρετητών
- Στον Πίνακας 6 για το λόγο του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό εξυπηρετητών

Παρατηρούμε ότι ο μέσος όρος του χρόνου Python και ο μέσος όρος του χρόνου GAMS αυξάνονται με την αύξηση του αριθμού των εξυπηρετητών.

Χρόνος Python

Αριθμός Εξυπηρετητών	Average	min	max
10	0,048448276	0,0309	0,0752
11	0,046313333	0,0325	0,0644
12	0,047786207	0,0333	0,0738
13	0,051368966	0,0362	0,0757
14	0,051806667	0,034	0,0669
15	0,052696667	0,0351	0,072
16	0,05327	0,042	0,0686
17	0,058603448	0,044	0,079
18	0,06204	0,0486	0,0844
19	0,059753333	0,0353	0,0866
20	0,062716667	0,0419	0,0907
21	0,06365	0,0451	0,0848
22	0,06384	0,0439	0,0875
23	0,0627	0,0472	0,0882
24	0,066173333	0,0454	0,0945
25	0,062723333	0,0419	0,081
26	0,070166667	0,0462	0,1172
27	0,071724138	0,0509	0,1051
28	0,073237931	0,0488	0,1027
29	0,072344828	0,045	0,1194
30	0,07255	0,0516	0,0997

Πίνακας 3: Χρόνος της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό εξυπηρετητών

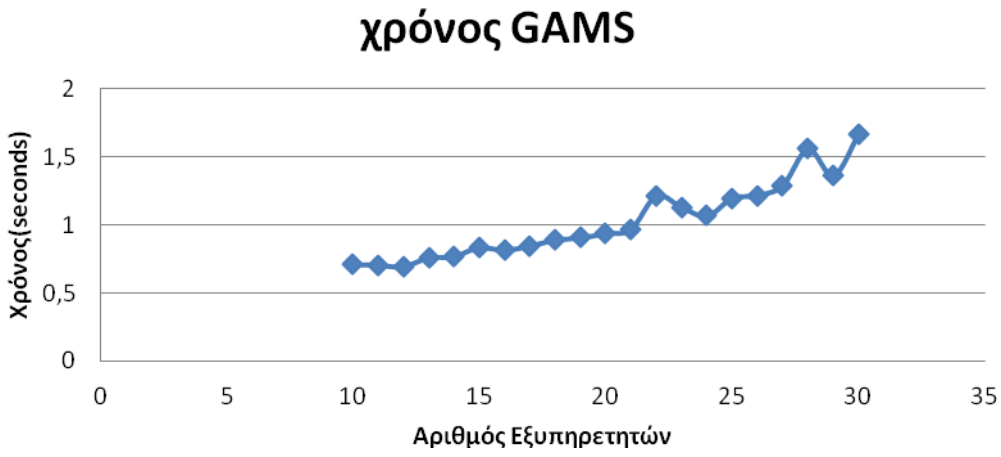


Εικόνα 71 : Μέσος χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό εξυπηρετητών

Χρόνος GAMS

Αριθμός Εξυπηρετητών	Average	min	max
10	0,710641379	0,6078	1,173
11	0,70675	0,5976	0,9775
12	0,694172414	0,6262	0,7763
13	0,759044828	0,6623	1,3776
14	0,76822	0,6771	0,9753
15	0,834466667	0,7055	1,977
16	0,81558	0,7349	0,984
17	0,846189655	0,7504	1,0866
18	0,88879	0,7675	1,1403
19	0,90562	0,8007	1,3419
20	0,937	0,8496	1,2171
21	0,967656667	0,8247	1,3706
22	1,21397	0,8455	8,329
23	1,131106667	0,8723	2,6667
24	1,066206667	0,9311	1,4314
25	1,196736667	0,9365	3,9222
26	1,216856667	0,974	1,9522
27	1,29091	1,061	1,8709
28	1,558082759	1,0988	4,0413
29	1,362186207	1,1067	1,8433
30	1,6674	1,127	6,1702

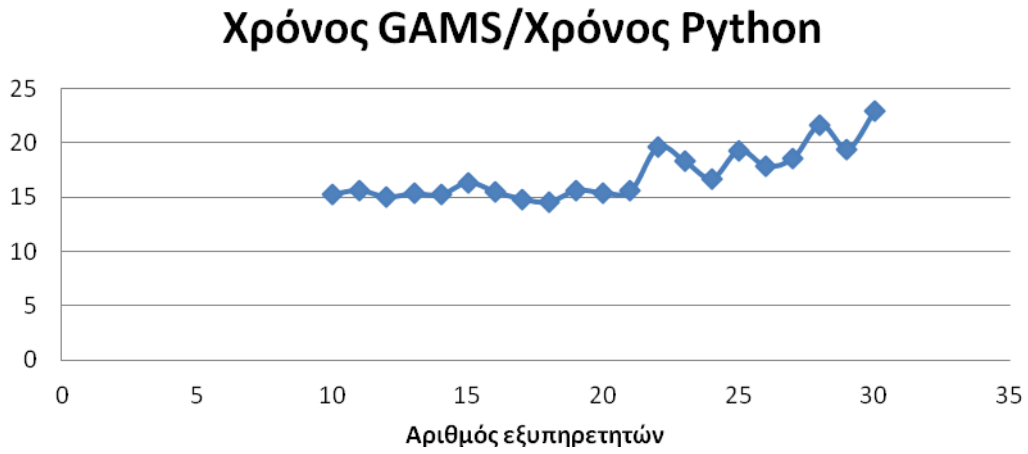
Πίνακας 4 : Χρόνος BARON με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 72 : Μέσος χρόνος CPU για τον υπολογισμό της βέλτιστης λύσης από το BARON με αυξανόμενο αριθμό εξυπηρετητών

Χρόνος_GAMS/ Χρόνος_Python Αριθμός εξυπηρετητών	Average	min	max
10	15,25884612	8,978	27,6651
11	15,62738215	11,791	22,7738
12	15,01382863	10,098	21,3857
13	15,34327604	9,6566	38,0552
14	15,22047111	10,35	21,3511
15	16,35738777	11,044	36,9533
16	15,50837952	12,245	19,0143
17	14,82325013	10,428	22,7322
18	14,56728883	10,203	20,2661
19	15,63245676	10,619	25,1864
20	15,4069905	10,412	21,5035
21	15,57073895	9,7252	23,0353
22	19,58619995	9,9474	137,442
23	18,28612609	11,389	44,5936
24	16,72588741	10,662	23,6542
25	19,29119664	12,604	56,4345
26	17,83883206	12,161	25,4545
27	18,60498965	12,556	30,1272
28	21,66737121	12,674	46,7743
29	19,4498998	9,2688	29,6828
30	22,89842015	12,911	76,7438

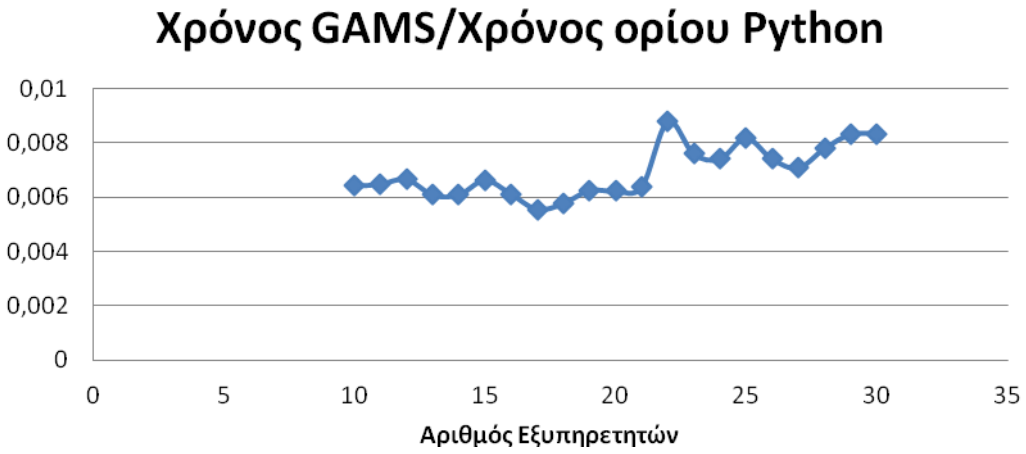
Πίνακας 5: Λόγος χρόνου του BARON προς τον χρόνο της βέλτιστης της Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 73 : Μέσος λόγος του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python

Χρόνος GAMS/ Χρόνος Ορίου Python			
Αριθμός Εξυπηρετητών	Average	min	max
10	0,006418	0,003536	0,011941
11	0,006493	0,002996	0,010549
12	0,006674	0,002931	0,016497
13	0,006091	0,002739	0,022862
14	0,006099	0,002969	0,012832
15	0,006634	0,002509	0,013525
16	0,006128	0,003055	0,013254
17	0,005564	0,002399	0,009238
18	0,005778	0,002801	0,010787
19	0,006262	0,002788	0,01212
20	0,006231	0,002917	0,013744
21	0,006376	0,003069	0,0154
22	0,008787	0,003089	0,070083
23	0,007598	0,003748	0,013285
24	0,007417	0,003195	0,017696
25	0,008186	0,004684	0,022993
26	0,00741	0,003089	0,015108
27	0,00709	0,00365	0,014249
28	0,007796	0,003404	0,012976
29	0,00832	0,002571	0,022136
30	0,008352	0,004406	0,015409

Πίνακας 6 : Λόγος χρόνου BARON προς χρόνο ορίου Python



Εικόνα 74 : μέσος όρος χρόνου BARON προς χρόνο ορίου Python

Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό υπηρεσιών

Επαναλαμβάνουμε την παραπάνω διαδικασία για αυξανόμενο αριθμό υπηρεσιών από 4 έως και 12. Στις περιπτώσεις όπου ο χρόνος CPU GAMS έχει ξεπεράσει το όριο των 120 δευτερολέπτων η εκτέλεση διακόπηκε και τα αποτελέσματα αυτών δεν συνυπολογίζονται στους πίνακες και τα σχήματα. Στους πίνακες που ακολουθούν παρουσιάζονται τα συγκεντρωτικά στατιστικά στοιχεία:

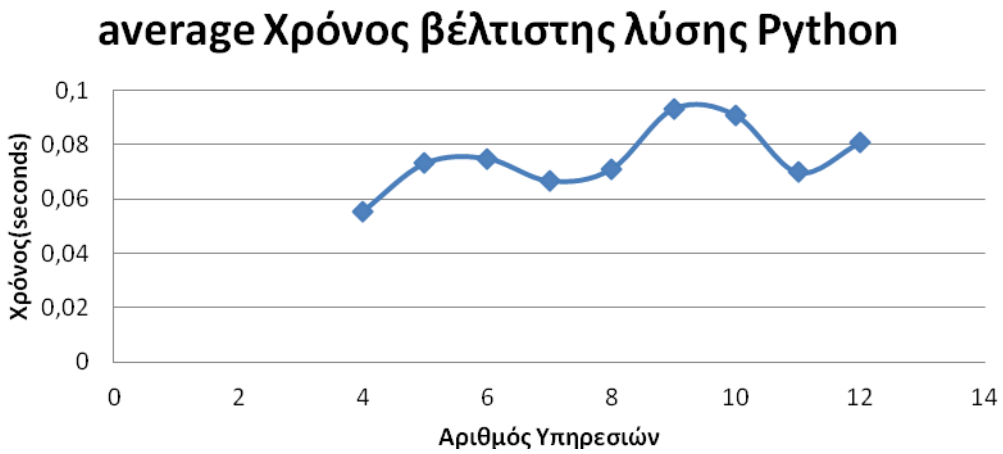
- Στον Πίνακας 7 για το χρόνο cpu τια τον υπολογισμό της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό υπηρεσιών
- Στον Πίνακας 8 για τον υπολογισμό της βέλτιστης λύσης από τον BARON με αυξανόμενο αριθμό υπηρεσιών
- Στον Πίνακας 9 για τον λόγο του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό υπηρεσιών
- Στον Πίνακας 10 για το λόγο του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό υπηρεσιών

Παρατηρούμε ότι ο μέσος χρόνος GAMS προς το μέσο χρόνο Python αυξάνονται με την αύξηση του αριθμού των υπηρεσιών. Αυτό συμβαίνει διότι η αύξηση στο χρόνο GAMS είναι πολύ μεγαλύτερη από την αύξηση στο χρόνο Python.

Χρόνος Python

Αριθμός Υπηρεσιών	Average	min	max
4	0,055551852	0,0258	0,0844
5	0,073151724	0,026	0,1123
6	0,074989286	0,0222	0,1281
7	0,066748148	0,0276	0,1128
8	0,071066667	0,0158	0,1248
9	0,0931	0,0374	0,2668
10	0,090733333	0,0613	0,1635
11	0,069865217	0,0224	0,1634
12	0,080725	0,0311	0,1593

Πίνακας 7 : Χρόνος Python με αυξανόμενο αριθμό υπηρεσιών

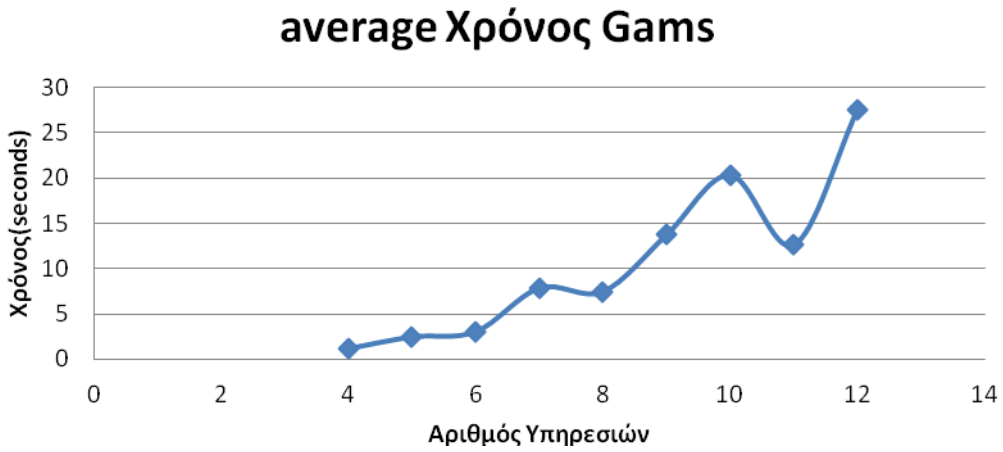


Εικόνα 75 : Μέσος χρόνος Python με αυξανόμενο αριθμό υπηρεσιών

Χρόνος Gams

Αριθμός Υπηρεσιών	Average	min	max
4	1,156792593	0,7319	4,0984
5	2,457268966	0,8826	20,347
6	3,037867857	0,9857	18,418
7	7,867985185	1,1577	40,646
8	7,477490476	1,3844	27,015
9	13,76572273	2,7054	66,735
10	20,36226667	3,1084	94,501
11	12,64163043	2,3555	41,553
12	27,5928625	2,9906	102,31

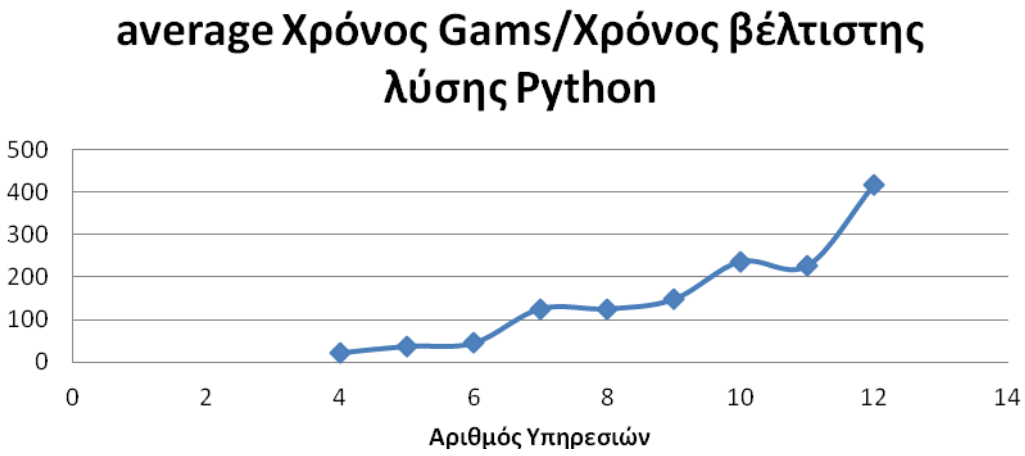
Πίνακας 8 : Χρόνος BARON με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 76 : Μέσος Χρόνος BARON με αυξανόμενο αριθμό υπηρεσιών

Χρόνος GAMS/ Χρόνο Python Αριθμός Υπηρεσιών	Average	min	max
4	22,00336705	9,5829	62,9555
5	37,53913147	8,2253	289,842
6	45,01882442	7,7127	237,969
7	125,6328403	12,652	454,846
8	125,471421	11,093	543,563
9	147,8218496	11,136	487,472
10	236,4584965	37,798	1118,36
11	226,2830147	28,21	721,403
12	416,0865476	42,88	2402,73

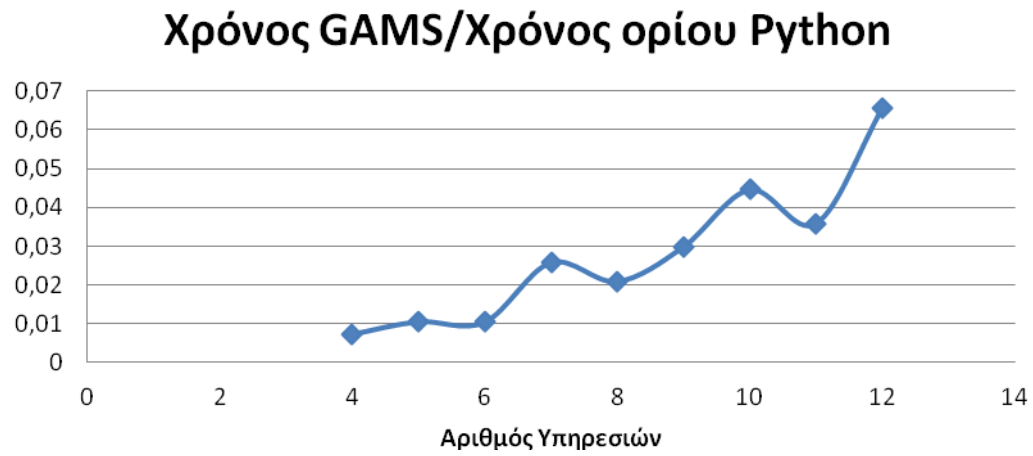
Πίνακας 9 : Λόγος χρόνου BARON προς χρόνο Python με αυξανόμενο αριθμό υπηρεσιών



Εικόνα 77 : Μέσος λόγος GAMS προς χρόνο Python με αυξανόμενο αριθμό υπηρεσιών

Χρόνος GAMS/ Χρόνο ορίου Python Αριθμός Υπηρεσιών	Average	min	max
4	0,007281	0,002813	0,026073
5	0,010555	0,002587	0,089218
6	0,010421	0,002351	0,047999
7	0,025789	0,002513	0,10487
8	0,020924	0,002908	0,070033
9	0,029849	0,004853	0,126801
10	0,04462	0,005961	0,179776
11	0,035913	0,004616	0,107255
12	0,06569	0,006011	0,220426

Πίνακας 10 : Χρόνος BARON προς χρόνο ορίου Python



Εικόνα 78 : Μέσος Χρόνος BARON προς χρόνο ορίου Python

Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό συστατικών

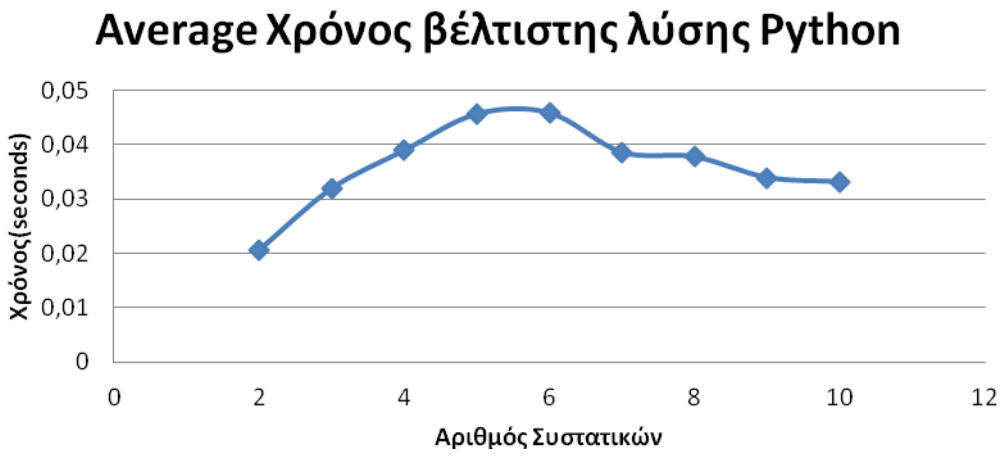
Επαναλαμβάνουμε την παραπάνω διαδικασία για αυξανόμενο αριθμό συστατικών από 4 έως και 10. Στις περιπτώσεις όπου ο χρόνος CPU GAMS έχει ξεπεράσει το όριο των 120 δευτερολέπτων η εκτέλεση διακόπηκε και τα αποτελέσματα αυτών δεν συνυπολογίζονται στους πίνακες και τα σχήματα. Στους πίνακες που ακολουθούν παρουσιάζονται τα συγκεντρωτικά στατιστικά στοιχεία:

- Στον Πίνακας 11 για τον χρόνο υπολογισμού της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό συστατικών.
- Στον Πίνακας 12 για τον υπολογισμό της βέλτιστης λύσης από τον BARON με αυξανόμενο αριθμό συστατικών
- Στον Πίνακας 13 για το λόγο του χρόνου υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό συστατικών
- Στον Πίνακας 14 για τον λόγο του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό συστατικών

Χρόνος βέλτιστης λύσης Python

Αριθμός Συστατικών	Average	min	max
2	0,020608824	0,0119	0,031
3	0,031923077	0,0208	0,05
4	0,038935714	0,0112	0,0501
5	0,045617241	0,0265	0,0646
6	0,045825	0,0091	0,0891
7	0,0386125	0,0139	0,0843
8	0,037760714	0,0143	0,0628
9	0,033966667	0,0087	0,0728
10	0,03319	0,0138	0,0547

Πίνακας 11 : Χρόνος Python με αυξανόμενο αριθμό συστατικών

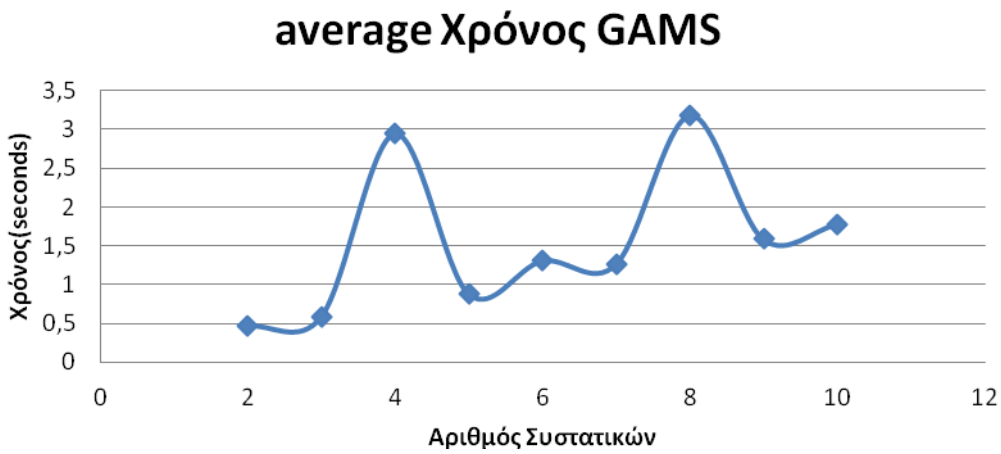


Εικόνα 79 : Μέσος Χρόνος Python με αυξανόμενο αριθμό συστατικών

Χρόνος GAMS

Αριθμός Συστατικών	Average	min	max
2	0,473588235	0,3977	0,6514
3	0,587396154	0,5053	0,9296
4	2,947367857	0,5726	56,801
5	0,883434483	0,6913	1,5498
6	1,309	0,7504	8,2753
7	1,265041667	0,9921	1,8039
8	3,182785714	1,1694	22,582
9	1,58626	0,8776	4,1367
10	1,78093	1,3651	2,6107

Πίνακας 12 : Χρόνος BARON με αυξανόμενο αριθμό συστατικών

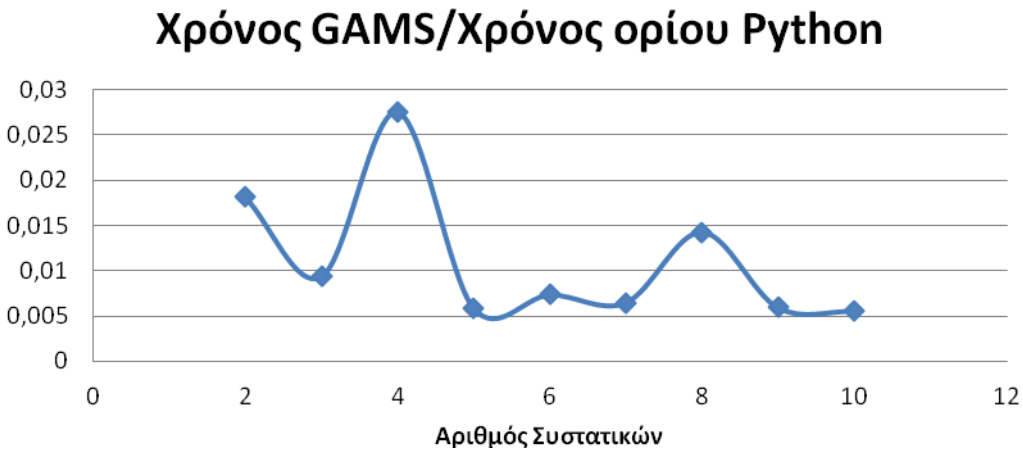


Εικόνα 80 : Μέσος χρόνος BARON με αυξανόμενο αριθμό συστατικών

Χρόνος GAMS/ Χρόνος ορίου Python

Αριθμός Συστατικών	Average	min	max
2	0,018141	0,008537	0,044113
3	0,009331	0,00443	0,01907
4	0,027497	0,004482	0,545705
5	0,005896	0,003064	0,011721
6	0,007344	0,003347	0,0368
7	0,006474	0,003237	0,012354
8	0,014272	0,003301	0,106041
9	0,005931	0,002831	0,01854
10	0,005523	0,003528	0,007917

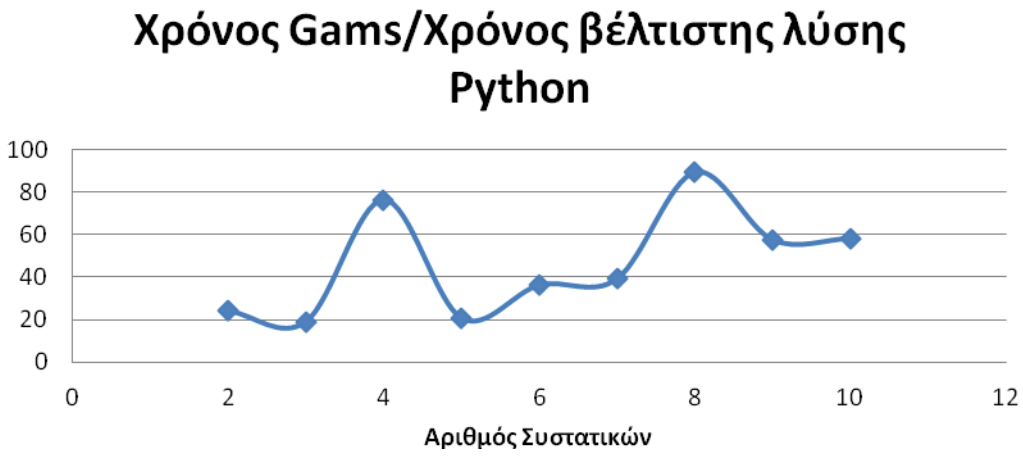
Πίνακας 13 : χρόνος GAMS προς χρόνο ορίου της Python



Εικόνα 81 : μέσος χρόνος GAMS προς χρόνο ορίου της Python

Χρόνος GAMS/ Χρόνος Python Αριθμός Συστατικών	Average	min	max
2	24,04000874	14,703	34,5839
3	18,8953767	11,318	25,3989
4	76,25658956	13,225	1434,36
5	20,5838086	10,947	44,1538
6	36,12073618	11,798	152,681
7	39,12047868	15,17	118,014
8	89,2142485	22,96	482,519
9	57,34896065	18,239	315,779
10	58,07066529	29,042	119,757

Πίνακας 14 : Χρόνος GAMS προς χρόνο Python



Εικόνα 82 : Μέσος χρόνος GAMS προς χρόνο Python

Εκτέλεση 30 περιπτώσεων με αυξανόμενο αριθμό δικτύων

Επαναλαμβάνουμε την παραπάνω διαδικασία για αυξανόμενο αριθμό δικτύων και συγκεκριμένα για 2,3,4,6,8,12,24 και αυξάνουμε τους εξυπηρετητές σε 24 . Στις περιπτώσεις όπου ο χρόνος CPU GAMS έχει ξεπεράσει το όριο των 120 δευτερολέπτων η εκτέλεση διακόπηκε και τα αποτελέσματα αυτών δεν συνυπολογίζονται στους πίνακες και τα σχήματα. Στους πίνακες που ακολουθούν παρουσιάζονται τα συγκεντρωτικά στατιστικά στοιχεία:

- Στον Πίνακας 15 για το χρόνο υπολογισμού της βέλτιστης λύσης από την Python με αυξανόμενο αριθμό δικτύων.

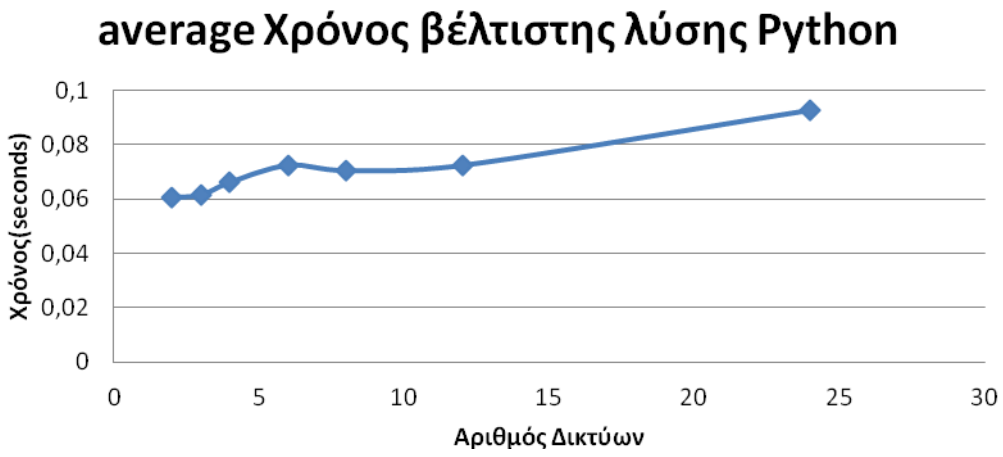
- Στον Πίνακας 16 για τον υπολογισμό της βέλτιστης λύσης από τον BARON με αυξανόμενο αριθμό δικτύων.
- Στον Πίνακας 17 για τον λόγο του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού της αντικειμενικής συνάρτησης της Python με αυξανόμενο αριθμό δικτύων.
- Στον Πίνακας 18 για τον λόγο του χρόνων υπολογισμού του GAMS προς τον χρόνο υπολογισμού του ορίου της Python με αυξανόμενο αριθμό δικτύων

Παρατηρούμε ότι ο μέσος όρος του χρόνου Python αυξάνεται με την αύξηση του αριθμού των δικτύων.

Χρόνος Python

Αριθμός Δικτύων	Average	min	max
2	0,060841379	0,0378	0,1137
3	0,061666667	0,0434	0,0922
4	0,066296667	0,0479	0,1057
6	0,072604	0,0551	0,1023
8	0,0706875	0,0491	0,0909
12	0,072522222	0,0514	0,1056
24	0,09275	0,0701	0,1326

Πίνακας 15 : Χρόνος Python με αυξανόμενο αριθμό δικτύων

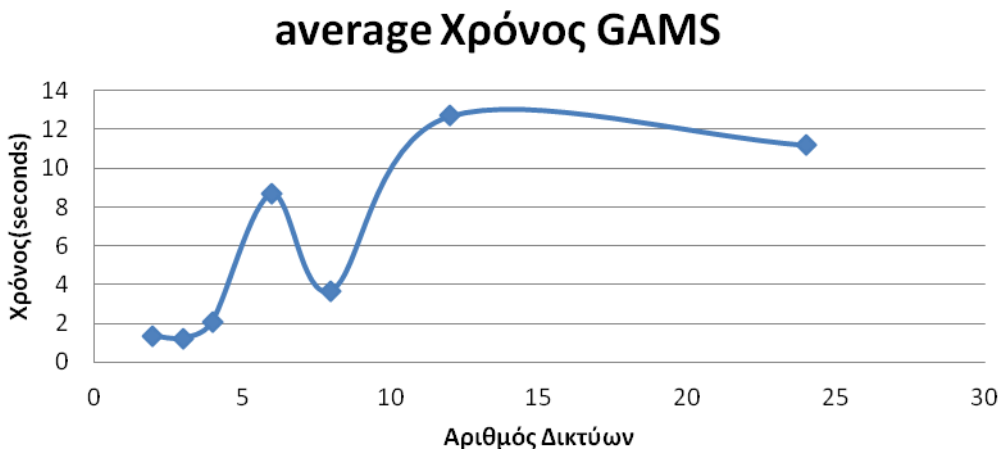


Εικόνα 83 : Μέσος Χρόνος Python με αυξανόμενο αριθμό δικτύων

Χρόνος GAMS

Αριθμός Δικτύων	Average	min	max
2	1,3367	0,9261	4,138
3	1,2337	0,9926	1,596
4	2,0573	1,1056	6,501
6	8,6737	1,3052	111,5
8	3,6761	1,5199	12,01
12	12,69	2,0069	51,6
24	11,19	3,4996	18,68

Πίνακας 16 : Χρόνος του BARON με αυξανόμενο αριθμό δικτύων

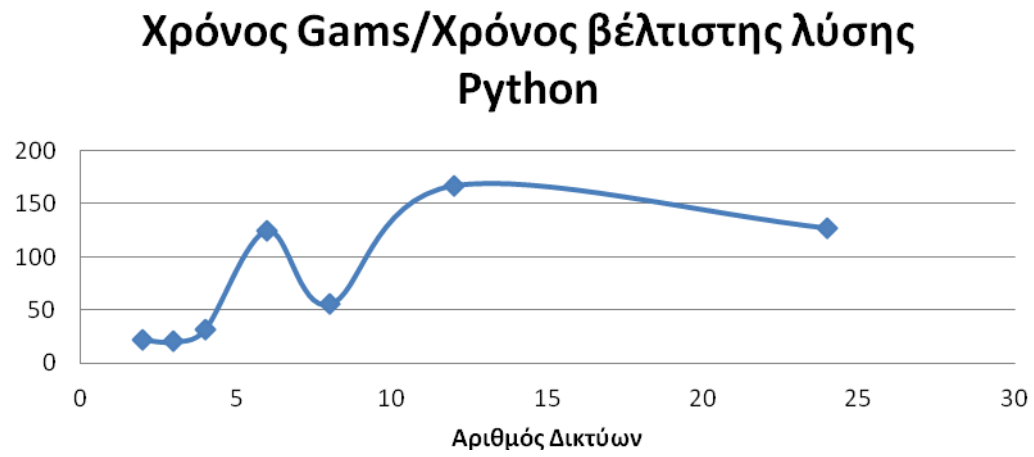


Εικόνα 84 : Μέσος : Χρόνος του BARON με αυξανόμενο αριθμό δικτύων

Χρόνος GAMS/
Χρόνος Python

Αριθμός Δικτύων	Average	min	max
2	22,346	9,72383	51,5268
3	20,403	13,2408	28,4964
4	31,438	13,2535	84,2098
6	124,34	15,4585	1654,23
8	55,388	22,4645	244,697
12	166,43	28,3061	785,411
24	126,79	39,678	230,108

Πίνακας 17 : Χρόνος GAMS προς το χρόνο Python

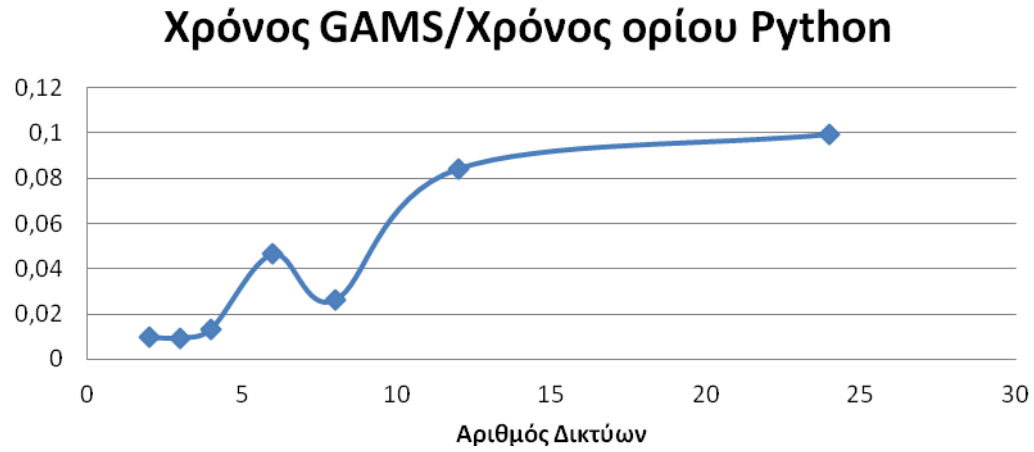


Εικόνα 85 : Μέσος χρόνος GAMS προς το χρόνο Python

Χρόνος GAMS/ Χρόνος Ορίου Python

Αριθμός δικτύων	Average	min	max
2	0,009634	0,003986	0,021724
3	0,009482	0,003962	0,025691
4	0,013376	0,003793	0,0485
6	0,046447	0,007785	0,640305
8	0,026055	0,007654	0,131836
12	0,084228	0,010547	0,423129
24	0,099411	0,02329	0,251851

Πίνακας 18 : Χρόνος GAMS προς το χρόνο του ορίου της Python



Εικόνα 86: Μέσος χρόνος GAMS προς το χρόνο του ορίου της Python

4.7) Έκτη κατηγορία περιπτώσεων – προφίλ

Προκειμένου να εξετάσουμε περαιτέρω την επίδοση των μοντέλων βελτιστοποίησης δημιουργήθηκαν διαφορετικά προφίλ μοντέλων (model profiles) για διαφορετικό αριθμό υπηρεσιών, συστατικών και υποδικτύων. Για κάθε προφίλ επαναλήφθηκαν 30 εκτελέσεις και καταγράφηκαν όλοι οι χρόνοι και τα αποτελέσματα, προκειμένου να αυξηθεί η ακρίβεια των αποτελεσμάτων.

Ο αριθμός των εξυπηρετητών που θεωρούνται ως υποψήφιοι για ανάθεση σχεδιάζεται στον άξονα x με μέγιστο το 1000 ενώ ο χρόνος CPU που καταναλώνεται σχεδιάζεται στον άξονα y. Συνολικά εξετάζονται 4 προφίλ με αυξανόμενο αριθμό εξυπηρετητών. Αναλυτικότερα:

- το προφίλ(6,2,4) αντιστοιχεί σε ένα μοντέλο που θεωρεί 6 υπηρεσίες με 2 συστατικά η κάθε μια σε 4 δίκτυα
- Το προφίλ(4,2,10) αντιστοιχεί σε ένα μοντέλο που θεωρεί 4 υπηρεσίες με 2 συστατικά η κάθε μια και 10 δίκτυα
- Το προφίλ(1,10,3) αντιστοιχεί σε ένα μοντέλο που θεωρεί 1 υπηρεσία με 10 συστατικά η κάθε μια και 3 δίκτυα
- Το προφίλ(10,4,5) αντιστοιχεί σε ένα μοντέλο που θεωρεί 10 υπηρεσίες με 4 συστατικά η κάθε μια και 5 δίκτυα.

Για κάθε προφίλ εκτελέστηκαν 30 εκτελέσεις για 5 εξυπηρετητές, 30 εκτελέσεις για 10 εξυπηρετητές και αντίστοιχα για 20, 50, 100, 150, 200, 500, 1000 αντίστοιχα. Θα λάβουμε υπ’ όψιν για το χρόνο του GAMS δύο όρια τα 120 δευτερόλεπτα καθώς τα 900 δευτερόλεπτα που αν ξεπεραστούν τερματίζεται το GAMS. Οι εκτελέσεις στις οποίες ο χρόνος ξεπέρασε το όριο των 900 δευτερολέπτων δεν συμπεριλήφθησαν στα διαγράμματα και τα στατιστικά στοιχεία. Επιπρόσθετα στους πίνακες που ακολουθούν αναφέρονται αναλυτικά τα ποσοστά των περιπτώσεων που ξεπέρασαν τα παραπάνω όρια για κάθε προφίλ. Οι απαιτήσεις των υπηρεσιών και των συστατικών τους καθώς και όλες οι μετρικές παίρνουν τιμές όμοια με το A5 δηλαδή:

- Πρόστιμο διαθεσιμότητας = (βασικές και ελαστικές απαιτήσεις σε εικονικές μηχανές) * 10
- διαθεσιμότητα = κατανομή Gauss με μέση τιμή 0,95 και διασπορά 0,015 με μέγιστο το 1 και (ενδεχομένως[4]) διαφορετικό για κάθε υπηρεσία.

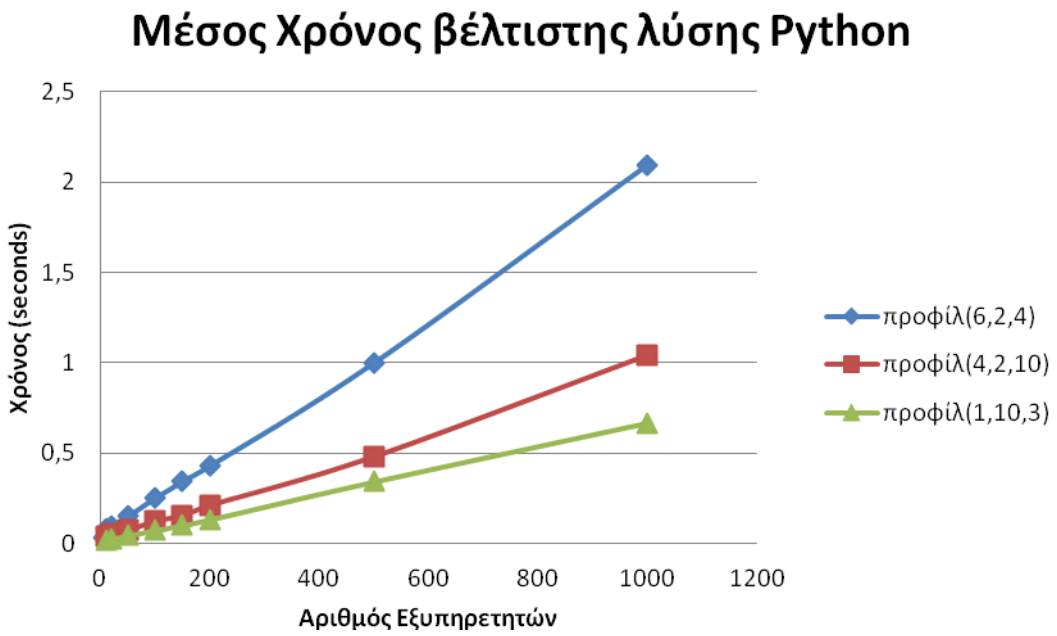
[4] Βλέπε σημείωση 1 σελ. 36

- Κέρδος = (βασικές και ελαστικές απαιτήσεις σε εικονικές μηχανές) * 10 * κατανομή Gauss (μέσης τιμής 1,2 και διασποράς 0,1)
- Βασικές εικονικές μηχανές = τυχαία επιλογή ανάμεσα στο 1,2,3 για κάθε συστατικό κάθε υπηρεσίας
- Ελαστικές εικονικές μηχανές = κατανομή gamma(2,2) για κάθε συστατικό κάθε υπηρεσίας
- Βασικές απαιτήσεις δικτύου = 0 για κάθε συστατικό κάθε υπηρεσίας
- Ελαστικές απαιτήσεις δικτύου = τυχαία επιλογή ανάμεσα στο 50, 100, 200 για κάθε συστατικό κάθε υπηρεσίας
- Μέγιστος αριθμός επεξεργαστών = 16 για κάθε εξυπηρετητή
- Δεσμευμένοι επεξεργαστές = 50% πιθανότητα για τιμή μηδέν και 50% στην κατανομή (16-gamma(2,2)) με μέγιστο το 15 διαφορετικό για κάθε εξυπηρετητή
- Χωρητικότητα δικτύου = τυχαία επιλογή ανάμεσα στο 8000, 16000, 24000 διαφορετικό για κάθε εξυπηρετητή
- virtual_cpus= 1 για κάθε συστατικό κάθε υπηρεσίας
- κόστος χρήσης = ακολουθεί εκθετική κατανομή με τιμές στο διάστημα [2,6] για κάθε εξυπηρετητή
- κόστος ενεργοποίησης = επιλογή ανάμεσα στο 5,10,15 για κάθε εξυπηρετητή.

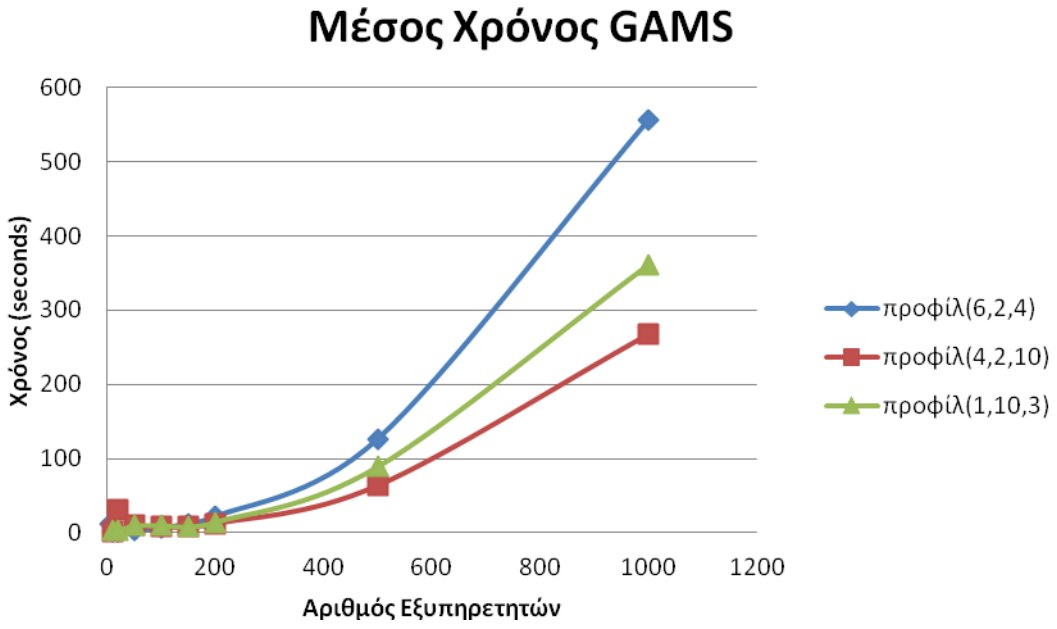
Στην Εικόνα 87 φαίνεται ο χρόνος CPU για τον υπολογισμό της αντικειμενικής συνάρτησης από την Python με αυξανόμενο αριθμό εξυπηρετητών για το προφίλ(6,2,4), το προφίλ(4,2,10) και το προφίλ(1,10,3).

Στην Εικόνα 88 φαίνεται ο χρόνος CPU για τον υπολογισμό της αντικειμενικής συνάρτησης από το GAMS με αυξανόμενο αριθμό εξυπηρετητών για το προφίλ(6,2,4), το προφίλ(4,2,10) και το προφίλ(1,10,3). Επισημαίνουμε ότι στις περιπτώσεις στις οποίες ο χρόνος αυτός ξεπέρασε τα 900 δευτερόλεπτα η εκτέλεση του GAMS τερματίστηκε και οι περιπτώσεις αυτές δεν συνυπολογίστηκαν στα διαγράμματα.

Τα αντίστοιχα διαγράμματα για το προφίλ(10,4,5) φαίνονται παρακάτω στην Εικόνα 89 και την Εικόνα 90.



Εικόνα 87 : Μέσος χρόνος Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 88 : Μέσος χρόνος GAMS με αυξανόμενο αριθμό εξυπηρετητών

Στους πίνακες Πίνακας 19 , Πίνακας 20 , Πίνακας 21 Πίνακας 22 φαίνονται ο μέσος χρόνος CPU για τον υπολογισμό της αντικειμενικής συνάρτησης από το GAMS προς τον μέσο χρόνο CPU για τον υπολογισμό της αντικειμενικής συνάρτησης από την Python για τα προφίλ(6,2,4) , (4,2,10) , (1,10,3) και (10,4,5) αντίστοιχα. Παρατηρούμε ότι και στα 4 προφίλ το μοντέλο της Python είναι πολύ πιο γρήγορο από το αντίστοιχο του GAMS.

Στον Πίνακας 23 βλέπουμε το ποσοστό των 30 εκτελέσεων ανά προφίλ και ανά αριθμό εξυπηρετητών που ο χρόνος CPU του GAMS ξεπέρασε τα 120 δευτερόλεπτα. Παρατηρούμε ότι σε όλα τα προφίλ για 1000 εξυπηρετητές το ποσοστό των εκτελέσεων ήταν 100%.

Στον Πίνακας 24 βλέπουμε το ποσοστό των 30 εκτελέσεων ανά προφίλ και ανά αριθμό εξυπηρετητών που ο χρόνος CPU του GAMS ξεπέρασε τα 900 δευτερόλεπτα και διακόπηκε η εκτέλεση. Παρατηρούμε ότι σε όλα τα προφίλ υπήρξαν περιπτώσεις για κάποιο αριθμό εξυπηρετητών που το όριο αυτό ξεπεράστηκε. Επίσης στο προφίλ(10,4,5) για 500 και 1000 εξυπηρετητές το GAMS δεν κατάφερε να τερματίσει εντός του ορίου των 900 δευτερολέπτων σε καμία από τις εκτελέσεις και να επιστρέψει αποτέλεσμα. Αντίθετα το μοντέλο της Python τερμάτισε επιτυχώς σε όλες τις εκτελέσεις με μέσους χρόνους 5,6515 δευτερόλεπτα και 11,477 δευτερόλεπτα για 500 και 1000 εξυπηρετητές αντίστοιχα.

Επιπρόσθετα όσον αφορά την διαφορά των αποτελεσμάτων της αντικειμενικής συνάρτησης των δυο μοντέλων πρέπει να αναφερθεί ότι ήταν πολύ μικρή. Υπενθυμίζουμε ότι το αποτέλεσμα του GAMS είναι μεγαλύτερο ή ίσιο αυτού της Python.

Στον Πίνακας 25 φαίνεται ο μέσος όρος των λόγων των δύο αποτελεσμάτων ανά προφίλ και αριθμό εξυπηρετητών. Τα δυο κενά κελιά στον πίνακα προκύπτουν επειδή το GAMS δεν επέστρεψε κανένα αποτέλεσμα σε αυτές τις περιπτώσεις αφού ξεπέρασε το χρονικό όριο των 900 δευτερολέπτων και η εκτέλεση διακόπηκε.

Χρόνος GAMS/Χρόνος Python προφίλ (6,2,4)

Αριθμός Εξυπηρετητών	Average	min	max
5	321,16	13,75	2668,30
10	14,38	7,08	45,40
20	14,19	10,15	22,66
50	22,16	16,27	39,08
100	26,71	16,54	73,50
150	37,49	28,42	47,22
200	51,67	36,95	67,79
500	129,92	91,48	169,78
1000	269,61	211,73	357,63

Πίνακας 19 : Χρόνος GAMS/ Χρόνος Python προφίλ(6,2,4)

Χρόνος GAMS/Χρόνος Python προφίλ(4,2,10)

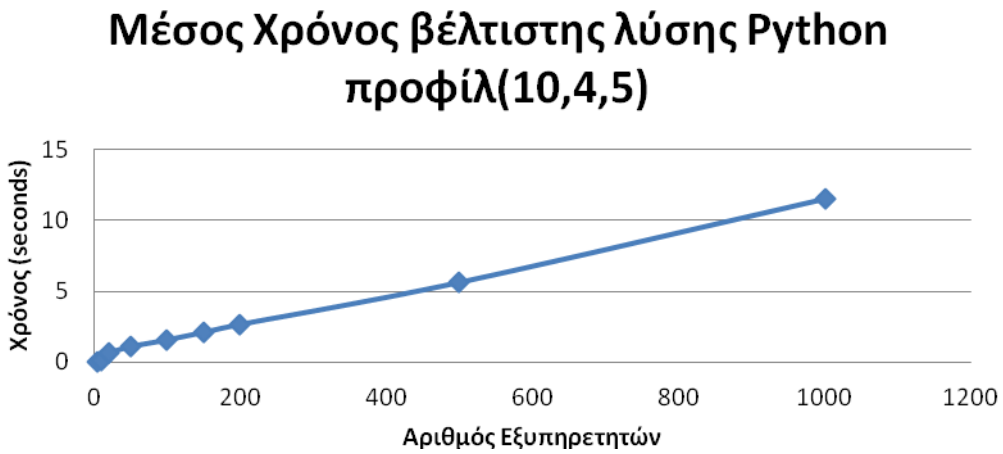
Αριθμός Εξυπηρετητών	Average	min	max
10	42,12	18,02	188,87
20	590,04	18,14	4904,33
50	139,79	19,32	753,87
100	66,69	22,67	512,77
150	52,68	32,81	98,42
200	60,11	36,68	94,23
500	136,17	104,25	192,58
1000	263,25	201,24	355,41

Πίνακας 20 : Χρόνος GAMS/ Χρόνος Python προφίλ(4,2,10)

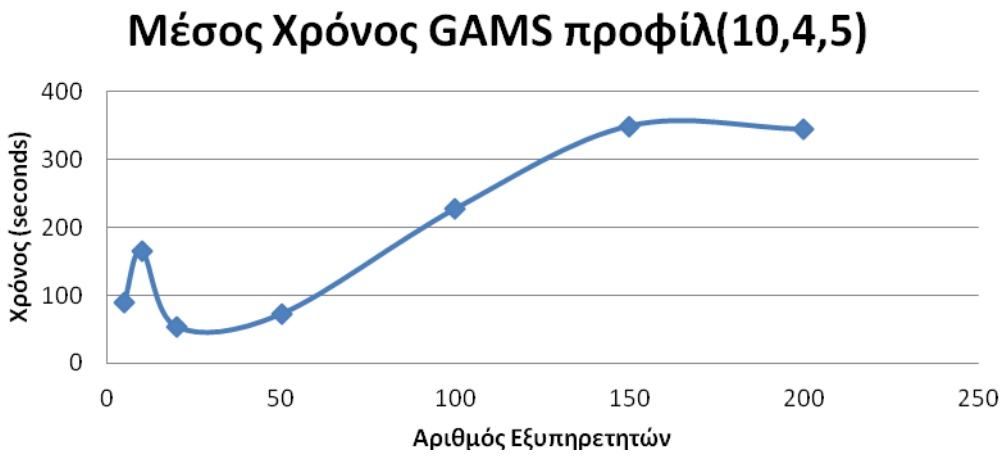
Χρόνος GAMS/Χρόνος Python Προφίλ(1,10,3)

Αριθμός Εξυπηρετητών	Average	min	max
10	142,04	19,29	2237,30
20	92,49	25,54	587,20
50	214,26	32,67	2638,96
100	129,92	44,64	1158,97
150	80,47	59,61	106,97
200	109,52	74,02	147,76
500	265,97	215,88	330,56
1000	557,46	404,26	832,26

Πίνακας 21 : Χρόνος GAMS/ Χρόνος Python προφίλ(1,10,3)



Εικόνα 89 : Μέσος Χρόνος Python με αυξανόμενο αριθμό εξυπηρετητών



Εικόνα 90 : Μέσος χρόνος GAMS με αυξανόμενο αριθμό εξυπηρετητών

Χρόνος GAMS/Χρόνος Python προφίλ(10,4,5)

Αριθμός Εξυπηρετητών	Average	min	max
5	2519,69	227,02	9391,77
10	1284,50	44,74	3625,38
20	90,32	10,16	447,64
50	66,46	17,74	277,96
100	145,64	49,15	414,06
150	167,47	65,33	353,24
200	129,65	87,87	274,03
500			
1000			

Πίνακας 22 : Χρόνος GAMS/ Χρόνος python προφίλ(10,4,5)

Χρόνος cpu GAMS >
120 δευτερόλεπτα

Αριθμός Εξυπηρετητών	προφίλ(6,2,4)	προφίλ(4,2,10)	προφίλ(1,10,3)	προφίλ(10,4,5)
5	3,33			46,6
10	0	30	0	66,6
20	0	40	0	43,3
50	3,33	20	3,33	23,3
100	0	10	6,66	70
150	0	3,33	0	100
200	0	0	0	100
500	0	0	3,33	100
1000	0	100	100	100

Πίνακας 23 : Ποσοστό (%) από τις 30 περιπτώσεις ανά προφίλ και αριθμό εξυπηρετητών όπου ο χρόνος GAMS ήταν μεγαλύτερος των 120 δευτερολέπτων

Χρόνος cpu GAMS >
900 δευτερόλεπτα

Αριθμός Εξυπηρετητών	προφίλ(6,2,4)	προφίλ(4,2,10)	προφίλ(1,10,3)	προφίλ(10,4,5)
5	3,33			26,6
10	0	30	0	36,6
20	0	26,6	0	30
50	3,33	20	0	10
100	0	10	3,33	30
150	0	0	0	30
200	0	0	0	20
500	0	0	0	100
1000	0	0	0	100

Πίνακας 24 Ποσοστό (%) από τις 30 περιπτώσεις ανά προφίλ και αριθμό εξυπηρετητών όπου ο χρόνος GAMS ήταν μεγαλύτερος των 900 δευτερολέπτων

Αποτέλεσμα GAMS /
Αποτέλεσμα Python

Αριθμός Εξυπηρετητών	προφίλ(6,2,4)	προφίλ(4,2,10)	προφίλ(1,10,3)	προφίλ(10,4,5)
10	1,015885627	1,069962695	1,028222096	1,085261892
20	1,006526534	1,01534133	1,041379306	1,023527359
50	1,006299438	1,016557002	1,023038964	1,004418891
100	1,003054398	1,011510143	1,008715936	1,003077353
150	1,001410347	1,005649214	1,002605647	1,002535958
200	1,001071201	1,003962331	1,00052519	1,001628051
500	1	1,000125891	1	
1000	1	1	1	

Πίνακας 25 : μέσος όρος του λόγου του αποτελέσματος GAMS προς το αποτελέσματα της Python ανά προφίλ και αριθμό εξυπηρετητών

5) Συμπεράσματα

Από όλες τις παραπάνω περιπτώσεις καταλήγουμε ότι η μοντελοποίηση της Python είναι πολύ πιο αποδοτική από την αντίστοιχη του GAMS. Σε πολλές περιπτώσεις μάλιστα και ειδικά σε αυτές με μεγάλο αριθμό εξυπηρετητών, δικτύων, υπηρεσιών και συστατικών το GAMS απέτυχε να επιστρέψει αποτέλεσμα εντός πολύ μεγάλου χρονικού περιθωρίου, ενώ η Python τερμάτισε επιτυχώς σε όλες τις περιπτώσεις και επέστρεψε αποτέλεσμα σε πολύ σύντομο χρονικό διάστημα. Παρατηρήθηκε επίσης από τους λόγους των χρόνων ότι σε πολλές περιπτώσεις ο χρόνος cpu στο GAMS αυξάνεται πολύ γρηγορότερα από το χρόνο cpu στην Python καθώς αυξάνεται ο αριθμός ενός εκ των εξυπηρετητών, δικτύων, υπηρεσιών και συστατικών.

Παράλληλα γνωρίζοντας ότι το GAMS επιστρέφει τη βέλτιστη λύση, παρατηρώντας τους λόγους των αποτελεσμάτων του GAMS και της Python προκύπτει ότι το σφάλμα στην αντικειμενική συνάρτηση στην πλειοψηφία των περιπτώσεων είναι πάρα πολύ μικρό. Δεν θα πρέπει να παραλείψουμε να αναφέρουμε ότι σημειώθηκαν και περιπτώσεις, σε μικρό ποσοστό, που το σφάλμα έφτασε το 15% .

Εν κατακλείδι η μοντελοποίηση στην Python εμφανίζεται πολύ πιο γρήγορη από την μοντελοποίηση του GAMS με μικρό σφάλμα και πιο αξιόπιστη στο να επιστρέψει αποτέλεσμα σε ρεαλιστικούς χρόνους. Παρ’ όλα αυτά θα πρέπει να χρησιμοποιήσουμε το GAMS αν χρειαζόμαστε όσο το δυνατόν πιο ακριβή λύση.

6) Γλωσσάριο

Availability	διαθεσιμότητα
Allocation	ανάθεση
Bandwidth	Εύρος δικτύου
Cloud Computing	Υπολογιστικό νέφος
Component	πρόγραμμα-συστατικό
Computing requirements	υπολογιστικές απαιτήσεις
Cost	κόστος
Distributed	κατανεμημένη
Gain	κέρδος
Horizontal Elasticity	Οριζόντια Ελαστικότητα
Host	εξυπηρετητής
Infrastructure Provider	Πάροχος Υποδομής
Network	Δίκτυο
Network requirements	απαιτήσεις δικτύου
Service	Υπηρεσία
Service Provider	Πάροχος Υπηρεσίας
Solvers	Επιλυτές
Uptime	χρόνος λειτουργίας
Virtual Machine	Εικονική Μηχανή
Virtual Machine Manager	Διαχειριστής Εικονικής Μηχανής
Virtualized Infrastructure	Εικονικοποιημένη Υποδομή

7) Βιβλιογραφία

[1] Konstanteli, K.; Cucinotta, T.; Psychas, K.; Varvarigou, T., "Admission Control for Elastic Cloud Services," Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on , vol., no., pp.41,48, 24-29 June 2012

[2] *Optimis-Optimized Infrastructure Services* (n.d.). Retrieved from OPTIMIS Home Page: http://www.optimis-project.eu/

[3] Rosenthal, R. E. (n.d). GAMS | A User's Guide. Ανάκτηση από http://www.gams.com/dd/docs/bigdocs/GAMSUsersGuide.pdf

[4] *The General Algebraic Modeling System (GAMS) Website*. (n.d). Retrieved from GAMS Home Page: http://www.gams.com/

[5] Sahinidis, N (n.d.). BARON Ανάκτηση από http://www.gams.com/dd/docs/solvers/baron.pdf

[6] Γ.Κοκολάκης και Ι.Σπηλιώτης, " Εισαγωγή στη θεωρία Πιθανοτήτων και Στατιστική" , Εκδόσεις Συμεών, 1999

[7] A. Papoulis και S. Unnikrishna Pillai, "Πιθανότητες Τυχαίες Μεταβλητές & Στοχαστικές Διαδικασίες", Εκδόσεις Τζιόλα , 2007

[8] S.Haykin και M.Moher, "Συστήματα Επικοινωνίας" , Εκδόσεις Παπασωτηρίου , 2010

[9] Python 3.3.2 Documentation 9.6 random – Generate pseudo-random numbers. Ανάκτηση από http://docs.python.org/3.3/library/random.html

[10] T. Gaddis "Starting out with Python", Pearson Education, 2011

[11] Python 3.3.2 Documentation 26.3 unittest – Unit testing framework. Ανάκτηση από http://docs.python.org/3.3/library/unittest.html#module-unittest

[12] M. L. Hetland "Beginning Python From Novice to Professional", Apress, 2005

[13] Python 3.3.2 Documentation 27.4 The Python Profilers. Ανάκτηση από http://docs.python.org/3.3/library/profile.html#the-python-profilers

[14] M. Lutz, “Learning Python”, O’Reilly Media, 2009

8) Παράρτημα

1) Πηγαίος κώδικας από το testing.py (απόσπασμα). Υλοποιήθηκαν αντίστοιχα τα AdmissionTest1 και AdmissionTest3

```
class AdmissionTest2(unittest.TestCase):
    def setUp(self):
        in_folder = "Θέση φακέλου"
        fi.readinput(in_folder)

    def test_1(self):
        host_num=4
        service_num=3

        self.assertEqual(fc.get_total_bandwidth(),220)
        self.assertEqual(fc.get_total_cpus(),25)
        self.assertEqual(fc.get_total_cost(),405)
        self.assertEqual((fc.get_mincost_cpunet())[0][0],0)
        self.assertEqual((fc.get_maxgain_cpunet())[0][0],0)
        self.assertEqual(fc.get_maxgain_cpu_found_alloc(4)[0][0],0)


        r1 = fr.get_best_result(4)
        cpus_tou_h=[0]*(host_num+1)
        for i in range(1,host_num+1):
            cpus_tou_h[i]=tc.check_cpus_host(r1[3],'h'+str(i))
        for i in range(1,host_num+1):
            self.assertLessEqual(0,cpus_tou_h[i])
            self.assertLessEqual(cpus_tou_h[i],fi.dic_hos['h'+str(i)].maxcpus)

        total_cpus_allocated=0
        for i in range(1,host_num+1):
             total_cpus_allocated=cpus_tou_h[i]
        total_cpus_needed=0
        for i in range(1,service_num+1):
            total_cpus_needed=fi.dic_ser['s'+str(i)].get_cpus()
        self.assertLessEqual(total_cpus_allocated,total_cpus_needed)


    def test_2(self):
        self.assertLessEqual(fr.get_best_result(4), fr.get_best_result_bound())


    def test_3(self):

        #Service
        self.assertEqual(len(fi.dic_ser['s1'].get_comb()),6)
        self.assertEqual(len(fi.dic_ser['s1'].get_components()),2)
        self.assertEqual(len(fi.dic_ser['s2'].get_components()),2)
        self.assertEqual(len(fi.dic_ser['s3'].get_components()),2)
        av1=(fi.dic_ser['s1'].avail)/100
        av2=(fi.dic_ser['s2'].avail)/100
        av3=(fi.dic_ser['s3'].avail)/100
        self.assertEqual(round(fi.dic_ser['s1'].get_gain(av1)),960)
        self.assertEqual(round(fi.dic_ser['s2'].get_gain(av2)),760)
        self.assertEqual(round(fi.dic_ser['s3'].get_gain(av3)),960)
        self.assertEqual(fi.dic_ser['s1'].get_avail(960),av1)
        self.assertEqual(fi.dic_ser['s2'].get_avail(760),av2)
        self.assertEqual(fi.dic_ser['s3'].get_avail(960),av3)
        self.assertEqual(fi.dic_ser['s1'].get_cpus(),12)
        self.assertEqual(fi.dic_ser['s2'].get_cpus(),12)
        self.assertEqual(fi.dic_ser['s3'].get_cpus(),12)
        net1=len(fi.dic_net)
```

```
        self.assertEqual(fi.dic_ser['s1'].get_net(net1),320)
        self.assertEqual(fi.dic_ser['s2'].get_net(net1),32)
        self.assertEqual(fi.dic_ser['s3'].get_net(net1),32)

        #Host
        self.assertEqual(fi.dic_hos['h1'].is_off(),False)
        self.assertEqual(fi.dic_hos['h2'].is_off(),True)
        self.assertEqual(fi.dic_hos['h3'].is_off(),True)
        self.assertEqual(fi.dic_hos['h4'].is_off(),False)
        self.assertEqual(fi.dic_hos['h1'].get_avail_cpus(),7)
        self.assertEqual(fi.dic_hos['h2'].get_avail_cpus(),6)
        self.assertEqual(fi.dic_hos['h3'].get_avail_cpus(),6)
        self.assertEqual(fi.dic_hos['h4'].get_avail_cpus(),6)
        avail_cpus_h1=fi.dic_hos['h1'].get_avail_cpus()
        avail_cpus_h2=fi.dic_hos['h2'].get_avail_cpus()
        avail_cpus_h3=fi.dic_hos['h3'].get_avail_cpus()
        avail_cpus_h4=fi.dic_hos['h4'].get_avail_cpus()
        self.assertEqual(fi.dic_hos['h1'].get_cost(avail_cpus_h1),35)
        self.assertEqual(fi.dic_hos['h2'].get_cost(avail_cpus_h2),35)
        self.assertEqual(fi.dic_hos['h3'].get_cost(avail_cpus_h3),35)
        self.assertEqual(fi.dic_hos['h4'].get_cost(avail_cpus_h4),300)

        #Network
        self.assertEqual(fi.dic_net['n1'].get_totalcpus(),13)
        self.assertEqual(fi.dic_net['n2'].get_totalcpus(),12)
        self.assertEqual(fi.dic_net['n1'].get_totalcost(),70)
        self.assertEqual(fi.dic_net['n2'].get_totalcost(),335)

        #Component
        el_s1_c1=fi.dic_ser['s1'].dic_com['c1'].elastic_vms
        el_s1_c2=fi.dic_ser['s1'].dic_com['c2'].elastic_vms
        el_s2_c1=fi.dic_ser['s2'].dic_com['c1'].elastic_vms
        el_s2_c2=fi.dic_ser['s2'].dic_com['c2'].elastic_vms
        el_s3_c1=fi.dic_ser['s3'].dic_com['c1'].elastic_vms
        el_s3_c2=fi.dic_ser['s3'].dic_com['c2'].elastic_vms
        self.assertEqual(fi.dic_ser['s1'].dic_com['c1'].get_cpus(el_s1_c1),6)
        self.assertEqual(fi.dic_ser['s1'].dic_com['c2'].get_cpus(el_s1_c2),6)
        self.assertEqual(fi.dic_ser['s2'].dic_com['c1'].get_cpus(el_s2_c1),6)
        self.assertEqual(fi.dic_ser['s2'].dic_com['c2'].get_cpus(el_s2_c2),6)
        self.assertEqual(fi.dic_ser['s3'].dic_com['c1'].get_cpus(el_s3_c1),6)
        self.assertEqual(fi.dic_ser['s3'].dic_com['c2'].get_cpus(el_s3_c2),6)

        nets2=len(fi.dic_net)
        self.assertEqual(fi.dic_ser['s1'].dic_com['c1'].get_net(el_s1_c1,nets2),160)
        self.assertEqual(fi.dic_ser['s1'].dic_com['c2'].get_net(el_s1_c2,nets2),160)
        self.assertEqual(fi.dic_ser['s2'].dic_com['c1'].get_net(el_s2_c1,nets2),16)
        self.assertEqual(fi.dic_ser['s2'].dic_com['c2'].get_net(el_s2_c2,nets2),16)
        self.assertEqual(fi.dic_ser['s3'].dic_com['c1'].get_net(el_s3_c1,nets2),16)
        self.assertEqual(fi.dic_ser['s3'].dic_com['c2'].get_net(el_s3_c2,nets2),16)


    def test_4(self):
        """ Test other"""
        self.assertEqual(len(fi.dic_ser),3)
        self.assertEqual(len(fi.dic_hos),4)
        self.assertEqual(len(fi.dic_net),2)
        self.assertEqual(len(fi.dic_ser['s1'].dic_com),2)
        self.assertEqual(len(fi.dic_ser['s2'].dic_com),2)
        self.assertEqual(len(fi.dic_ser['s3'].dic_com),2)

    def tearDown(self):
        fi.dic_ser['s1'].dic_com.clear()
        fi.dic_ser['s2'].dic_com.clear()
        fi.dic_ser['s3'].dic_com.clear()
        fi.dic_ser.clear()
        fi.dic_comb.clear()
```

```
        fi.dic_net.clear()
        fi.dic_hos.clear()


suite = unittest.TestLoader().loadTestsFromTestCase(AdmissionTest1)
suite.addTests(unittest.TestLoader().loadTestsFromTestCase(AdmissionTest2))
suite.addTests(unittest.TestLoader().loadTestsFromTestCase(AdmissionTest3))
unittest.TextTestRunner(verbosity=2).run(suite)
```

2) Από το αρχείο όπου λαμβάνουν τιμές οι παράμετροι του προβλήματος: Στο συγκεκριμένο απόσπασμα αρχικοποιούνται κάποιες από τις παραμέτρους της περίπτωσης που αναλύθηκε στην παράγραφο 4.6)

```
availability = [min(int(random.gauss(95,1.5)),100)/100 for _ in range(servs)]
basic_vms = [[int(random.choice([1,2,3])) for _ in range(servs)] for _ in range(comps)]
elastic_vms = [[1+int(random.gammavariate(2,2)) for _ in range(servs)] for _ in
range(comps)]
basic_cost = [int(round(((sum([sum(a) for a in zip(*basic_vms)])+sum([sum(a) for a in
zip(*elastic_vms)]))*10*random.gauss(1.2,0.1)),-1)) for _ in range(servs)]
basic_net = [[ 0 for _ in range(servs)] for _ in range(comps)]
elastic_net = [[random.choice([50,100,200]) for _ in range(servs)] for _ in range(comps)]
max_cpus=[16 for _ in range(hosts)]
network_capacity = [random.choice([8000,16000,24000]) for _ in range(nets)]
res_cpus = [min(af.epilogh_res_cpus(),15) for _ in range(hosts)]
virtual_cpus = [[1 for _ in range(servs)] for _ in range(comps)]
availability_penalty = [int((sum([sum(a) for a in zip(*basic_vms)])+sum([sum(a) for a in
zip(*elastic_vms)]))*10) for _ in range(servs)]
```