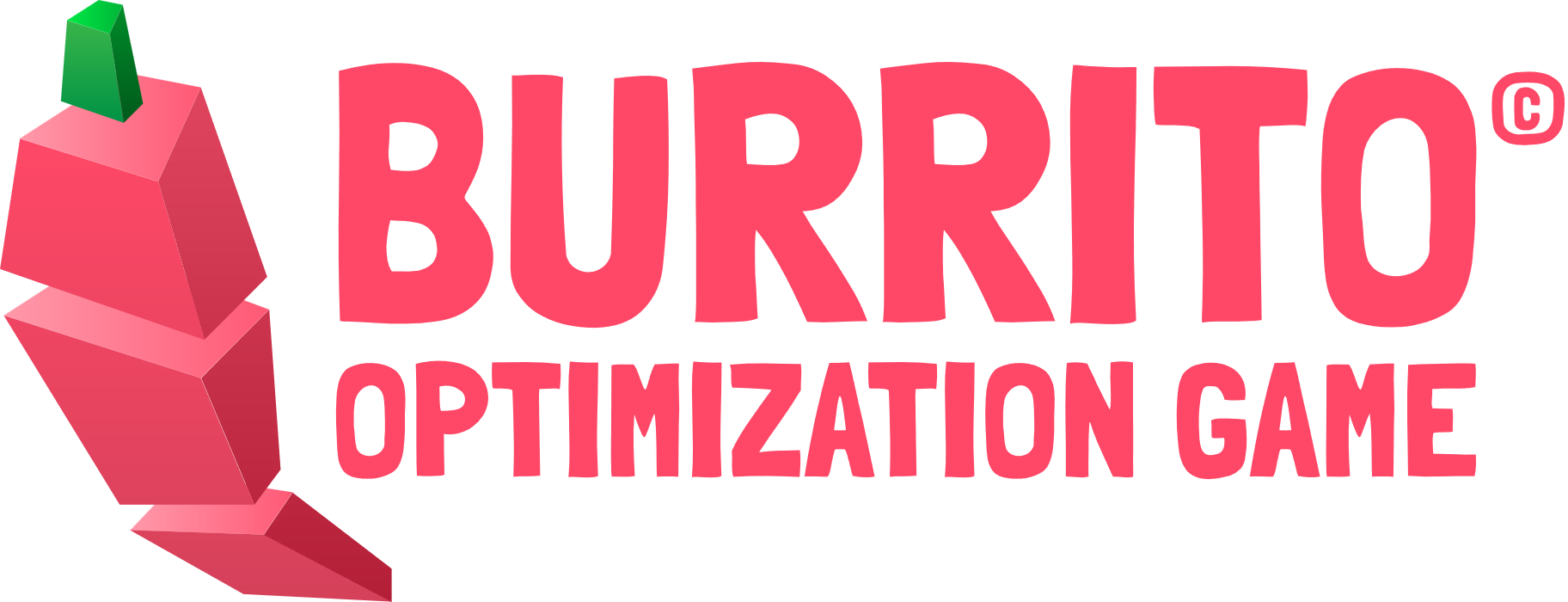
A bird with a head

AI-generated content may be incorrect.

**Τεχνική Αναφορά**  
Επιχειρησιακή Έρευνα  
Αναστάσιος Βήττας (ΑΜ: 197)  
18 Μαΐου 2025

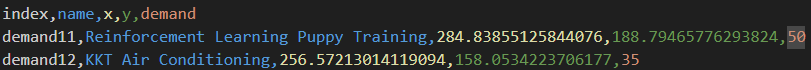
**Εισαγωγή**  
Το Burrito Optimization Game[[1]](#footnote-1) είναι ένα εκπαιδευτικό παιχνίδι βελτιστοποίησης που αναπτύχθηκε από τον Larry Snyder και την ομάδα του, με την υποστήριξη της Gurobi. Στόχος του είναι να εισάγει τους παίκτες σε έννοιες της επιχειρησιακής έρευνας και να αναδείξει τις δυνατότητες σύγχρονων λογισμικών επίλυσης μαθηματικών μοντέλων. Το παιχνίδι μπορεί να παιχτεί ατομικά ή σε μορφή πρωταθλήματος. Στην παρούσα εργασία, καλούμαστε να αναπτύξουμε λύσεις με χρήση δύο διαφορετικών εργαλείων βελτιστοποίησης όπως τα OR-Tools και πιο συγκεκριμένα τον επιλυτή CP-SAT αλλά και με τον Gurobi, έναν από τους πιο ισχυρούς και εμπορικά διαθέσιμους επιλυτές. Με τις λύσεις που θα προκύψουν από την επίλυση του μοντέλου με Gurobi και CP-SAT, θα αξιολογήσουμε την απόδοσή τους και θα καταθέσουμε τις αντίστοιχες απαντήσεις απευθείας στο πρωτάθλημα.



**Ερώτημα1 – Dataset**  
Κάθε μια από τις πέντε ημέρες του παιχνιδιού συνοδεύεται από τέσσερα αρχεία .csv που περιέχουν όλες τις απαραίτητες πληροφορίες για τη διαμόρφωση του προβλήματος.

1. scenDE4BD4\_round1\_day(1-5)\_demand\_node\_data.csv:

Το πρώτο αρχείο αφορά τους κόμβους ζήτησης, δηλαδή τους πελάτες που ζητούν burrito. Για κάθε πελάτη, δίνονται συντεταγμένες στο επίπεδο (x, y), ένα όνομα κτιρίου και το πόσα burrito χρειάζεται. Το σύνολο αυτών των πληροφοριών μάς λέει πού βρίσκεται κάθε πελάτης και βάση του demand πρέπει να παρθεί απόφαση για το αν θα μας προσφέρει μεγάλο κέρδος ή όχι το συγκεκριμένο κτήριο αν χτίσουμε κοντά του.



1. scenDE4BD4\_round1\_day(1-5)\_demand\_truck\_data.csv:

Το δεύτερο αρχείο άφορα τις πιθανές θέσεις της καντίνας, δηλαδή που μας αφήνει το παιχνίδι την κάθε μέρα ξεχωριστά να χτίσουμε σε σχέση με την ζήτηση. Ουσιαστικά το dataset περιλαμβάνει τέσσερις στήλες, η πρώτη στήλη (demand\_node\_index) αφορά το αναγνωριστικό του πελάτη δηλαδή του κάθε κτηριού από το προηγούμενο dataset, η δεύτερη στήλη (truck\_node\_index) το αναγνωριστικό του φορτηγού, η τρίτη στήλη (distance) την απόσταση μεταξύ τους και η τέταρτη στήλη (scaled\_demand) πόσα burrito από τη ζήτηση του συγκεκριμένου πελάτη μπορεί να καλύψει το φορτηγό.

A screenshot of a computer code

AI-generated content may be incorrect.

1. scenDE4BD4\_round1\_day(1-5)\_problem\_data.csv:

Το τρίτο αρχείο αφορά την τιμή πώλησης του burrito, το κόστος παραγωγής του και το κόστος τοποθέτησης κάθε καντίνας ξεχωριστά. Οι τιμές αυτές είναι διαφορετικές σε κάθε μια από τις πέντε μέρες.



1. scenDE4BD4\_round1\_day(1-5)\_truck\_node\_data.csv  
   Το τελεύταιο άρχειο μας δίνει τις συντεταγμένες (x,y) της κάθε καντίνας που μας επιτρέπει το παιχνίδι να στήσουμε την κάθε μέρα ξεχωριστά. Παρολα αυτά δεν χρησιμοποιείται στο Ερώτημα 2 της εργασίας καθώς δεν μας έχει ζητηθεί να υλοποιήσουμε γραφικά το παιχνίδι παρά μόνο να κάνουμε την μοντελοποίηση.

A screenshot of a computer

AI-generated content may be incorrect.

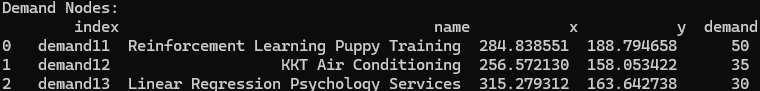
**Ερώτημα 2 – Επίλυση του προβλήματος με τον CP-SAT και τον Gurobi**

Στο δεύτερο ερώτημα της εργασίας υλοποιήθηκαν δυο μοντέλα (Gurobi και CP-SAT) που επιλύουν το Burrito Optimization Game[[2]](#footnote-2).

Τα αρχεία που δημιουργήθηκαν είναι 3 και είναι τα εξής:

1. read\_dataset.py: Ανοίγει το dataset, το διαβάζει και κάνει parse τα δεδομένα όταν καλέσουμε την συνάρτηση load\_data() που έχει γραφτεί εντός του αρχείου.
2. cpsat\_burrito.py: Σε αυτό το αρχείο υπάρχει η μοντελοποίηση του προβλήματος με την χρήση του επιλυτή CP-SAT.
3. gurobi\_burrito.py: Σε αυτό το αρχείο υπάρχει η μοντελοποίηση του προβλήματος με την χρήση του Gurobi.

Κατά την εκτέλεση των μοντέλων βελτιστοποίησης, χρησιμοποιούνται δεδομένα που προέρχονται από τρία διαφορετικά αρχεία csv για κάθε ημέρα ξεχωριστά. Αυτά τα δεδομένα διαβάζονται και μετατρέπονται σε δομές τύπου DataFrame μέσω της βιβλιοθήκης pandas της Python. Στην συνάρτηση load\_data() δημιουργούμαι δηλαδή τρια DataFrames, πρώτα το demand\_nodes, το truck\_assignments, και το problem\_data, όπως φαίνονται και στα Screenshots παρακάτω.



A screen shot of a computer

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

Έπειτα κάνουμε split τα δεδομένα όπως τα θέλουμε εμείς, δηλαδή την κάθε στήλη που χρειαζόμαστε ξεχωριστά.

    price = problem\_data['burrito\_price'][0]

    cost = problem\_data['ingredient\_cost'][0]

    truck\_cost = problem\_data['truck\_cost'][0]

    demands = demand\_nodes['index'].unique()

    trucks = truck\_assignments['truck\_node\_index'].unique()

Στην συνέχεια ορίζουμε μεταβλητές απόφασης κάθε τέτοια μεταβλητή είναι δυαδική και παίρνει τιμή 1 μόνο όταν πραγματοποιείται η αντίστοιχη ανάθεση. Δηλαδή δηλώνουμε με 0 ή 1 αν η κάθε καντίνα χρησιμοποιείται στη λύση ή όχι αλλά και αν η σύνδεση με καντίνας με ζήτηση είναι οκ.





Χρειαζόμαστε επίσης και περιορισμούς έτσι ώστε να περιορίσουμε κάθε ζήτηση να αφορά μόνο μια καντίνα και το αντίστροφο.



Σκοπός μας είναι να πετύχουμε το μεγαλύτερο κέρδος με την τοποθέτηση της κάθε καντίνας. Η αντικειμενική συνάρτηση[[3]](#footnote-3) που φαίνεται και στην εικόνα παρακάτω στοχεύει στην εύρεση του μέγιστου κέρδους.

A black text on a white background

AI-generated content may be incorrect.

* CP-SAT:

model.Maximize((price - cost) \* total\_units - total\_truck\_costs)

* Gurobi:

profit = (burrito\_price - ingredient\_cost) \* total\_units - total\_truck\_costs

model.setObjective(profit, GRB.MAXIMIZE)

Ουσιαστικά όπου είναι η τιμή των burrito πλην το κόστος παραγωγής τους, επείτα το πολλαπλασιάζουμε με το total\_units δηλαδή , όπου αν ο πελάτης εξυπηρετείται από το φορτηγό στη θέση , πολλαπλασιασμένο με την ζήτηση του κάθε πελάτη και τον συντελεστή , που εξαρτάται από την απόσταση της καντίνας με την ζήτηση, αν δηλαδή ο πελάτης είναι κοντά στην εκάστοτε καντίνα. Τέλος αφαιρούμε από το (price – cost) \* total\_units το total\_truck\_costs δηλαδή το , όπου f είναι η σταθερή τιμή που κοστίζει η κάθε καντίνα για να το τοποθετηθεί πολλαπλασιασμένη με 1 αν το φορτηγό j χρησιμοποιείται, αλλιώς 0.

**Ερώτημα 3 – Σύγκριση επιλύσεων, αξιολόγηση αποτελεσμάτων  
**A screenshot of a computer game

AI-generated content may be incorrect.

Στη συνέχεια,

|  |  |  |
| --- | --- | --- |
| **Ημέρα** | **Gurobi** | **CP-SAT** |
| 1 | 945.00 | 945.00 |
| 2 | 4660.00 | 4660.00 |
| 3 | 1632.00 | 1632.00 |
| 4 | 2415.00 | 2415.00 |
| 5 | 7925.00 | 7925.00 |
| **Σύνολο** | **17,577.00** | **17,577.00** |

A screen shot of a computer program

AI-generated content may be incorrect.

**Συμπεράσματα**

Η μαθηματική μοντελοποίηση με την χρήση των εργαλείων OR-Tools λύνει το πρόβλημα με ακρίβεια αλλά είναι πιο αργή σε μεγάλα δεδομένα σε αντίθεση με τον Ουγγρικό αλγόριθμο που είναι πολύ πιο γρήγορος. Το μοντέλο με περιορισμό ανάθεσης σε ομάδες έχει δικαίως πιο αυξημένο κόστος καθώς οι περιορισμοί «δυσκολεύουν» τον επιλυτή.

**Πόροι Συστήματος**

CPU: AMD Ryzen 7 7730U

RAM: 16GB unified DDR4

**Οδηγίες Εκτέλεσης Κώδικα**

Προαπαιτούμενα:

Python >= 3.8

git clone <https://github.com/tasosvittas/Operations-Research.git>

Δημιουργία και ενεργοποίηση περιβάλλοντος:

Windows:

cd Operations-Research\Ergasia2\_OS

python -m venv erg2env

erg2env\Scripts\activate

Linux / macos:

cd Operations-Research\Ergasia2\_OS

python3 -m venv erg2env

source erg2env/bin/activate

Εγκατάσταση βιβλιοθηκών:

pip install -r requirements.txt

Εκτέλεση των ερωτημάτων:

python cpsat\_burrito.py

python gurobi\_burrito.py

1. <https://www.gurobi.com/burrito-optimization-game/> [↑](#footnote-ref-1)
2. <https://www.gurobi.com/burrito-optimization-game/> [↑](#footnote-ref-2)
3. https://www.gurobi.com/burrito-optimization-game-guide/#the\_ip\_formulation [↑](#footnote-ref-3)