POST-DISASTER SUPPLY DELIVERY BY DRONES USING LINEAR PROGRAMMING

Γραμμική και Συνδυαστική Βελτιστοποίηση [EE916]

Γέροντας Νικόλαος - Α.Μ.: 1092813

Πίνακας περιεχομένων

GitHub: https://github.com/Nick-744/LinearProgramming

Εισαγωγή

Σε κρίσεις (όπως οι σεισμοί και οι πλημμύρες) η γρήγορη διανομή τροφίμων, φαρμάκων και νερού σε δύσβατα σημεία σώσει ζωές. Τα επανδρωμένα μέσα, ωστόσο, συχνά καθυστερούν λόγω πρόσβασης ή συντονισμού. Οι ημιαυτόνομοι δρόνοι (Μη Επανδρωμένα Αεροσκάφη) αποτελούν μία ασφαλή και ευέλικτη λύση στο πρόβλημα. Ωστόσο, το ερώτημα που πρέπει να απαντηθεί είναι:

Ποιος δρόνος πρέπει να σταλεί, από ποιο σημείο εφοδιασμού, προς ποιον προορισμό και με πόσο φορτίο, ώστε να ελαχιστοποιηθεί το συνολικό «κόστος ανθρωπιστικής παράδοσης»;

Σκοπιμότητα και Εφαρμογές

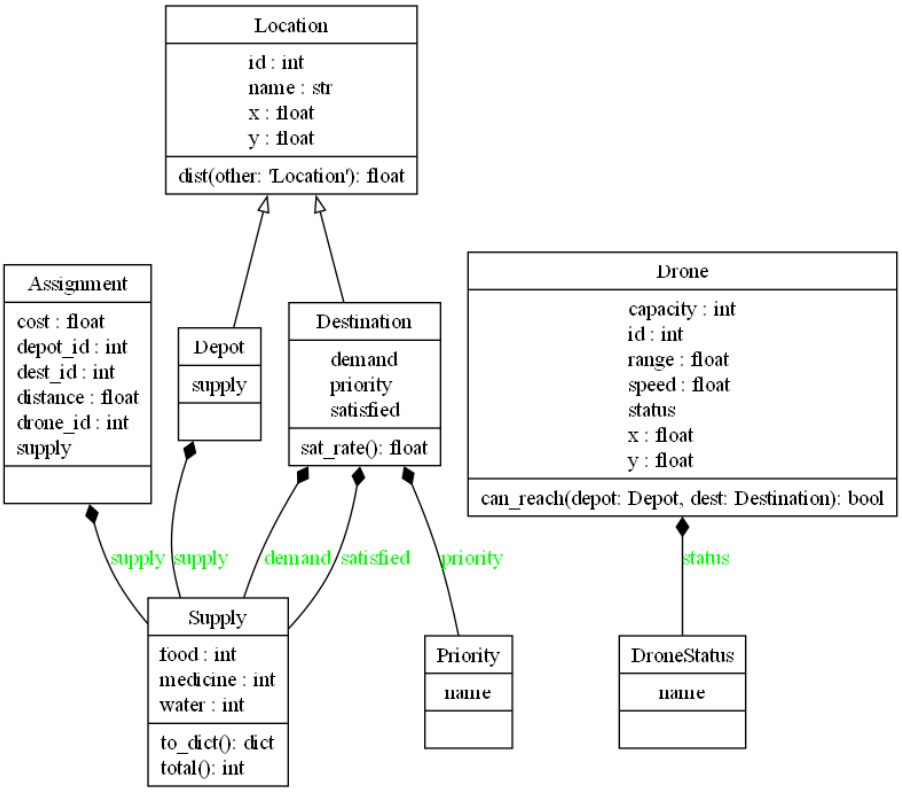
Η παρούσα προσέγγιση έχει σαφείς και ποικίλες εφαρμογές:

- ✓ Ανθρωπιστικές αποστολές, για στοχευμένες διανομές φαρμάκων και τροφίμων σε πραγματικό χρόνο.
- ✓ Έξυπνες πόλεις, με αυτόματα κιβώτια πρώτων βοηθειών που μεταφέρονται σε σημεία ανάγκης.
- ✓ Βιομηχανικά ή ενεργειακά campus, όπου απαιτείται άμεση διανομή ανταλλακτικών ή υλικών για τη διατήρηση της λειτουργικότητας.
- ✓ Στρατιωτικές επιχειρήσεις

Παράλληλα, η μελέτη αναδεικνύει τον τρόπο με τον οποίο εργαλεία Γραμμικού Προγραμματισμού μπορούν να μεταφραστούν σε λειτουργικές λύσεις με πρακτικό αντίκτυπο.

Μοντελοποίηση

Το UML διάγραμμα στα δεξιά, συνοψίζει τη δομή των βασικών κλάσεων που υλοποιούν το πρόβλημα. Αναπαρίστανται τα αντικείμενα του συστήματος (δρόνοι, σημεία εφοδιασμού, σημεία ανάγκης, προμήθειες), οι μεταξύ τους σχέσεις, καθώς και βασικές μέθοδοι που υποστηρίζουν τη λειτουργικότητα του αλγορίθμου (όπως ο υπολογισμός απόστασης και η κάλυψη αναγκών):



UML διάγραμμα του αρχείου models.py

Σύνδεση models.py - lp_solver.py / build_model() συνάρτηση

Μεταβλητές και Περιορισμοί

UML Οντότητα – Κλάση/Γνώρισμα/Μέθοδος	Μεταβλητή ή Παράμετρος MILP	Ερμηνεία
Drone.capacity	C_d	Μέγιστο φορτίο δρόνου d - **Περιορισμός**
Drone.range + can_reach()	Φίλτρο διαδρομών	Έγκυρες αποστολές – **Έμμεσος περιορισμός**
Depot.supply	$S_{i,s}$	Διαθέσιμη ποσότητα τύπου s στο σημείο εφοδιασμού i - **Περιορισμός**
Destination.demand	$D_{j,s}$	Απαίτηση τύπου s στο σημείο ανάγκης j - **Περιορισμός**
$y_{d_{ij}}$	Δυαδική	Ανάθεση διαδρομής/αποστολής σε έναν δρόνο
$x_{d_{ijs}}$	Συνεχής	Μονάδες τύπου s που μεταφέρονται
$unmet_{js}$	Slack μεταβλητές	Ανάγκη που δεν καλύπτεται

Αντικειμενική συνάρτηση:

$$\mathbf{Z = min}\left\{\sum_{\mathbf{d,i,j}}\left[\mathbf{dist(i,j)}\cdot \mathbf{w_{priority}(j)}\cdot \mathbf{y_{d_{ij}}}\right] + \sum_{\mathbf{j,s}}\left[\mathbf{UNMET\_PENALTY}\cdot \mathbf{w_{priority}(j)}\cdot \mathbf{unmet_{js}}\right]\right\}$$

Το **γνώρισμα priority** των σημείων ανάγκης επηρεάζει το κόστος στην αντικειμενική συνάρτηση (με βάρη `priority_w = {Priority.HIGH: 3, Priority.MEDIUM: 2, Priority.LOW: 1}`) ώστε να προτιμώνται κρίσιμες αποστολές!

Σημείωση:

Για τη σύνδεση της επιλογής αποστολής (δυαδική μεταβλητή y) με την ποσότητα φορτίου x, χρησιμοποιείται **η τεχνική Big-M**. Ο περιορισμός x ≤ M * y εξασφαλίζει ότι δεν μπορεί να σταλεί φορτίο αν δεν έχει επιλεγεί η αντίστοιχη αποστολή, περιορίζοντας το πεδίο λύσεων και βελτιώνοντας τη λογική συνέπεια του μοντέλου.

Δομή της λύσης - συνάρτηση solve() του lp_solver.py

Η solve():

- Κατασκευάζει το μαθηματικό μοντέλο/πρόβλημα (μέσω της build_model()) που περιεγράφηκε παραπάνω.
- Επιλύει το μοντέλο χρησιμοποιώντας τον MILP solver (CBC).
- Εξάγει την τελική λίστα αποστολών (assignments) που υλοποιούν την προτεινόμενη στρατηγική διανομής.

Δημιουργία των σεναρίων - scenario.py

Το αρχείο scenario.py λειτουργεί ως γεννήτρια δεδομένων εισόδου για το μοντέλο βελτιστοποίησης. Παρέχει έτοιμες συναρτήσεις δημιουργίας διαφορετικών σεναρίων παράδοσης, με ελεγχόμενο αριθμό δρόνων, αποθηκών και προορισμών. Η συναρτησιακή δομή είναι επεκτάσιμη, επιτρέποντας τη διαμόρφωση σύνθετων περιπτώσεων με διαφορετικές γεωμετρίες, απαιτήσεις προμηθειών και προτεραιότητες. Παράλληλα, η συνάρτηση print_scenario_info() παρέχει έναν πρακτικό τρόπο οπτικής επιβεβαίωσης των δεδομένων που χρησιμοποιούνται στην επίλυση. Το αρχείο αυτό διευκολύνει τον πειραματισμό και τη σύγκριση στρατηγικών χωρίς τροποποίηση του πυρήνα του αλγορίθμου.

- ✓ Το κάθε σενάριο πρέπει να ακολουθεί την εξής δομή:

```
def sample_scenario():
    drones = [...]

    depots = [...]

    dests  = [...]

    print_scenario_info(drones, depots, dests)

    return (drones, depots, dests);
```

Με βάση τις δοκιμές και την οπτικοποίηση μέσω animation, **το σενάριο big_city_scenario() αποδείχθηκε το πιο κατανοητό και ενδιαφέρον**. Η γεωγραφική διασπορά των προορισμών, η παρουσία 2 αποθηκών και η ποικιλία σε απαιτήσεις και προτεραιότητες δημιουργούν ένα ρεαλιστικό και οπτικά πλούσιο περιβάλλον. Το σενάριο αυτό αναδεικνύει καθαρά τη συμπεριφορά του αλγορίθμου, την απόδοση των δρόνων σε αποστολές διαφορετικής κρισιμότητας και την αξία της βελτιστοποίησης υπό περιορισμούς. **Το σενάριο περιλαμβάνει 3 δρόνους, 2 σημεία εφοδιασμού και 6 σημεία ανάγκης με διαφορετικές απαιτήσεις και προτεραιότητες.**

Απλή εκτέλεση σε τερματικό - main.py

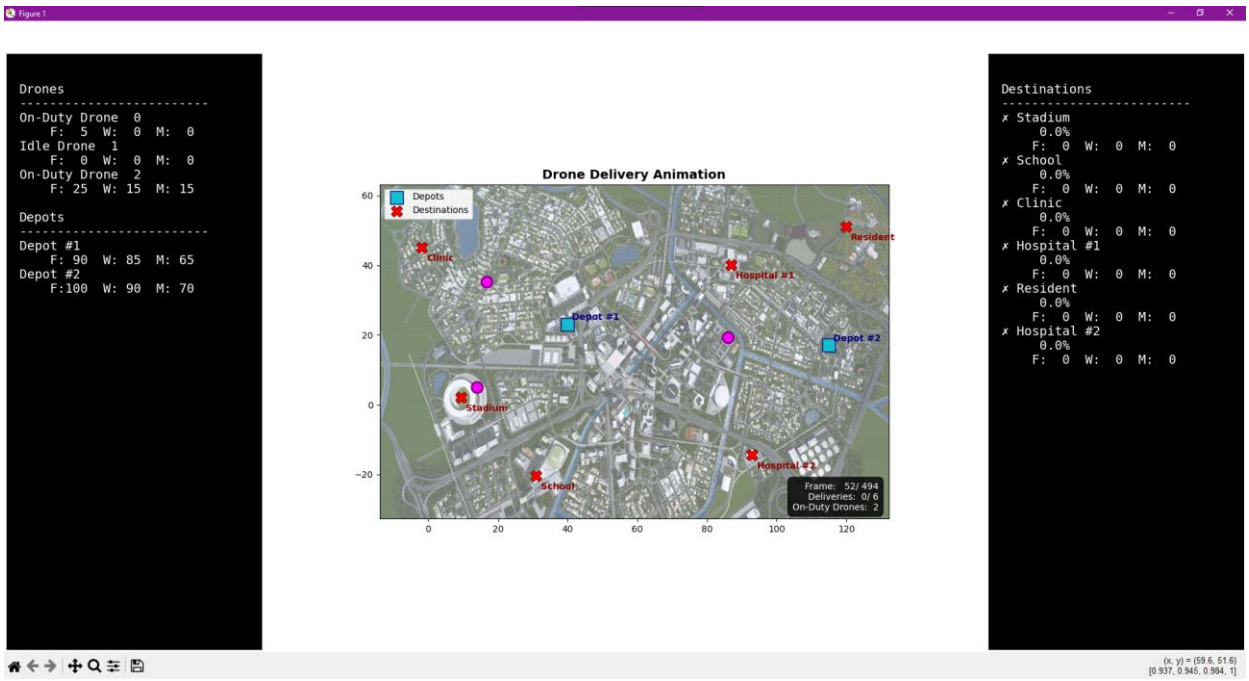
Εκτελώντας το αρχείο main.py (με το σενάριο sample_scenario()) προκύπτει το εξής αποτέλεσμα:

```
-> Πληροφορίες για το δοκιμαστικό σενάριο:
Αποθήκη Depot      (0) - Προμήθεια: 100 τ,  80 ν,  50 φ
Δρόνος  0 - Θέση: (  0.0,   0.0) - Χωρητικότητα: 100 - Εμβέλεια: 300.0 - Ταχύτητα:  50 - Κατάσταση: idle
Δρόνος  1 - Θέση: (  0.0,   0.0) - Χωρητικότητα:  80 - Εμβέλεια: 250.0 - Ταχύτητα:  60 - Κατάσταση: idle
Δρόνος  2 - Θέση: (  0.0,   0.0) - Χωρητικότητα: 120 - Εμβέλεια: 350.0 - Ταχύτητα:  40 - Κατάσταση: idle
Προορισμός Hospital     (0) - Ανάγκη:  40 τ,  30 ν,  10 φ - Προτεραιότητα: HIGH
Προορισμός Community    (1) - Ανάγκη:  50 τ,  40 ν,  20 φ - Προτεραιότητα: MEDIUM
Προορισμός School       (2) - Ανάγκη:  30 τ,  20 ν,  10 φ - Προτεραιότητα: LOW
Προορισμός Clinic       (3) - Ανάγκη:  20 τ,  10 ν,   5 φ - Προτεραιότητα: HIGH

 - Χρόνος επίλυσης: 0.17 sec -

Βέλτιστες αναθέσεις αποστολών:
Δρόνος 0 -> Hospital   | Απόσταση:  70.7 | Φορτίο:  40 τρόφιμα,  30 νερό,  10 φάρμακα
Δρόνος 0 -> School     | Απόσταση: 128.1 | Φορτίο:   0 τρόφιμα,   0 νερό,  10 φάρμακα
Δρόνος 1 -> Clinic     | Απόσταση:  76.2 | Φορτίο:  20 τρόφιμα,  10 νερό,   5 φάρμακα
Δρόνος 2 -> Community  | Απόσταση: 100.0 | Φορτίο:  40 τρόφιμα,  40 νερό,  20 φάρμακα

Ποσοστά κάλυψης προμηθειών ανά προορισμό:
  Hospital  : 100.0%
  Community : 90.9%
  School    : 16.7%
  Clinic    : 100.0%
```

Οπτικοποίηση με animation_2D.py

Το αρχείο animation_2D.py υλοποιεί μια πλήρως δυναμική γραφική αναπαράσταση των αποστολών των δρόνων, βάσει της λύσης του μαθηματικού μοντέλου. **Χρησιμοποιώντας εξ ολοκλήρου τη βιβλιοθήκη matplotlib**, προβάλλει σε πραγματικό χρόνο τις διαδρομές των δρόνων, τις παραλαβές από τις αποθήκες, τις παραδόσεις στους προορισμούς, καθώς και την κατάσταση ικανοποίησης κάθε σημείου ανάγκης.

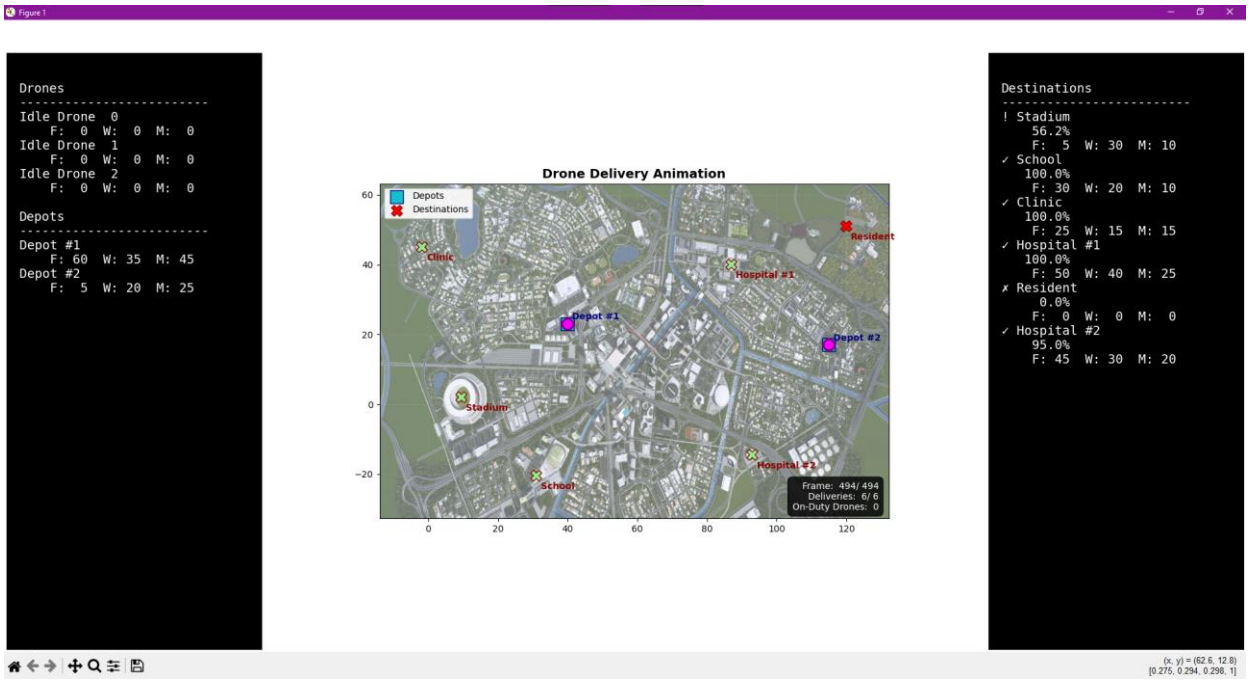
Παράδειγμα εκτέλεσης/animation:



Τυχαίο στιγμιότυπο #1 κατά την εκτέλεση

Τυχαίο στιγμιότυπο #2 κατά την εκτέλεση

Τελική κατάσταση:



Βίντεο στο YouTube από την εκτέλεση:
https://youtu.be/PEMtnI7_ko8

Αποτέλεσμα στο τερματικό:

```
-> Πληροφορίες για το δοκιμαστικό σενάριο:
Αποθήκη Depot #1   (0) - Προμήθεια: 120 τ, 100 ν,  80 φ
Αποθήκη Depot #2   (1) - Προμήθεια: 100 τ,  90 ν,  70 φ
Δρόνος  0 - Θέση: ( 40.0,  23.0) - Χωρητικότητα: 120 - Εμβέλεια: 300.0 - Ταχύτητα:  60 - Κατάσταση: idle
Δρόνος  1 - Θέση: (115.0,  17.0) - Χωρητικότητα: 100 - Εμβέλεια: 280.0 - Ταχύτητα:  55 - Κατάσταση: idle
Δρόνος  2 - Θέση: ( 40.0,  23.0) - Χωρητικότητα: 150 - Εμβέλεια: 350.0 - Ταχύτητα:  50 - Κατάσταση: idle
Προορισμός Stadium      (0) - Ανάγκη:  40 τ,  30 ν,  10 φ - Προτεραιότητα: MEDIUM
Προορισμός School       (1) - Ανάγκη:  30 τ,  20 ν,  10 φ - Προτεραιότητα: HIGH
Προορισμός Clinic       (2) - Ανάγκη:  25 τ,  15 ν,  15 φ - Προτεραιότητα: HIGH
Προορισμός Hospital #1  (3) - Ανάγκη:  50 τ,  40 ν,  25 φ - Προτεραιότητα: HIGH
Προορισμός Resident     (4) - Ανάγκη:  60 τ,  50 ν,  30 φ - Προτεραιότητα: LOW
Προορισμός Hospital #2  (5) - Ανάγκη:  45 τ,  35 ν,  20 φ - Προτεραιότητα: MEDIUM

-> Λύση σεναρίου:

Δρόνος 0:
  -> Stadium      | Απόσταση:  37.0 | Φορτίο:  5F  0W  0M
  -> Hospital #1  | Απόσταση:  36.2 | Φορτίο: 50F 40W 25M
  Συνολική απόσταση: 73.3

Δρόνος 1:
  -> Stadium      | Απόσταση:  37.0 | Φορτίο:  0F 30W 10M
  -> School       | Απόσταση:  44.4 | Φορτίο: 30F 20W 10M
  Συνολική απόσταση: 81.5

Δρόνος 2:
  -> Clinic       | Απόσταση:  47.2 | Φορτίο: 25F 15W 15M
  -> Hospital #2  | Απόσταση:  38.4 | Φορτίο: 45F 30W 20M
  Συνολική απόσταση: 85.7

Χρόνος επίλυσης: 0.938s

Ποσοστά κάλυψης προμηθειών ανά προορισμό:
  ! Stadium     :  56.2%
  ✓ School      : 100.0%
  ✓ Clinic      : 100.0%
  ✓ Hospital #1 : 100.0%
  ✗ Resident    :   0.0%
  ✓ Hospital #2 :  95.0%

Μέση κάλυψη: 75.2%
```

Ερμηνεία αποτελεσμάτων - γιατί ο κόμβος «Resident» έμεινε στο 0%;

Η λύση MILP επέλεξε να μη στείλει κανέναν δρόνο στον κόμβο Resident, με αποτέλεσμα ποσοστό κάλυψης 0%. Αυτό δεν αποτελεί «σφάλμα» του αλγορίθμου, αλλά ορθολογική συνέπεια της αντικειμενικής συνάρτησης: **ο κόμβος φέρει προτεραιότητα LOW (βάρος = 1)** και βρίσκεται σε σχετικά μεγάλη απόσταση από τα κοντινότερα depots. Για να τον εξυπηρετήσει κάποιος δρόνος θα έπρεπε να δεσμευθεί σχεδόν ολόκληρη η χωρητικότητά του και να καταναλωθεί σημαντικό τμήμα του διαθέσιμου range, στερώντας πόρους από κόμβους HIGH ή MEDIUM προτεραιότητας που κοστίζουν λιγότερο ανά μονάδα ωφέλειας. Με το ισχύον `UNMET_PENALTY = 1_000` η «ποινή» μη κάλυψης είναι μικρότερη από το πρόσθετο κόστος (απόσταση x βάρος x προτεραιότητας) που θα πρόκυπτε εάν τελικά εκτελούνταν η αποστολή, έτσι το μοντέλο «προτιμά» να πληρώσει το penalty!

Οδηγίες εγκατάστασης και εκτέλεσης

☑ Πριν την εκτέλεση του έργου, **είναι απαραίτητη η εγκατάσταση των παρακάτω βιβλιοθηκών Python**:

```
pip install pulp matplotlib numpy
```

✍ **Εκτέλεση του Αλγορίθμου (μέσω τερματικού)** - Για να εκτελεστεί το μοντέλο βελτιστοποίησης και να εμφανιστούν τα αποτελέσματα σε μορφή κειμένου:

```
python main.py
```

✍ **Οπτικοποίηση της Λύσης (2D Animation)** - Για να προβληθεί η λύση σε μορφή κινούμενης αναπαράστασης:

```
python animation_2D.py
```

Βιβλιογραφία

[1]. PuLP Documentation – https://coin-or.github.io/pulp/

[2]. https://www.quantagonia.com/post/last-mile-delivery-drone-example