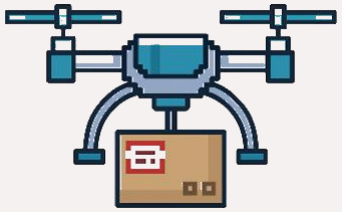
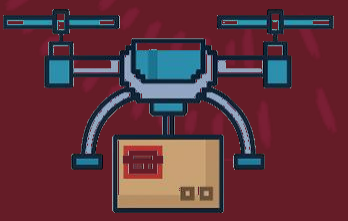
POST-DISASTER SUPPLY DELIVERY BY DRONES USING LINEAR PROGRAMMING



Γραμμική και Συνδυαστική Βελτιστοποίηση [ΕΕ916]

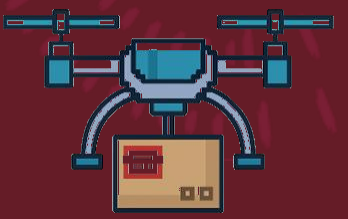
Γέροντας Νικόλαος

Εισαγωγή



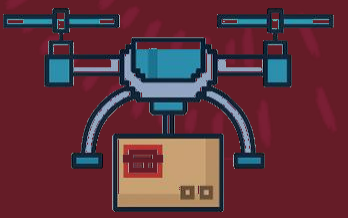
- Σε κρίσεις (όπως οι σεισμοί και οι πλημμύρες) η γρήγορη διανομή τροφίμων, φαρμάκων και νερού σε δύσβατα σημεία σώσει ζωές. Τα επανδρωμένα μέσα, ωστόσο, συχνά καθυστερούν λόγω πρόσβασης ή συντονισμού. Οι ημιαυτόνομοι δρόνοι (Μη Επανδρωμένα Αεροσκάφη) αποτελούν μία ασφαλή και ευέλικτη λύση στο πρόβλημα.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

Η ερώτηση...



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

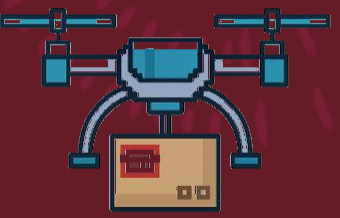
… η απάντηση

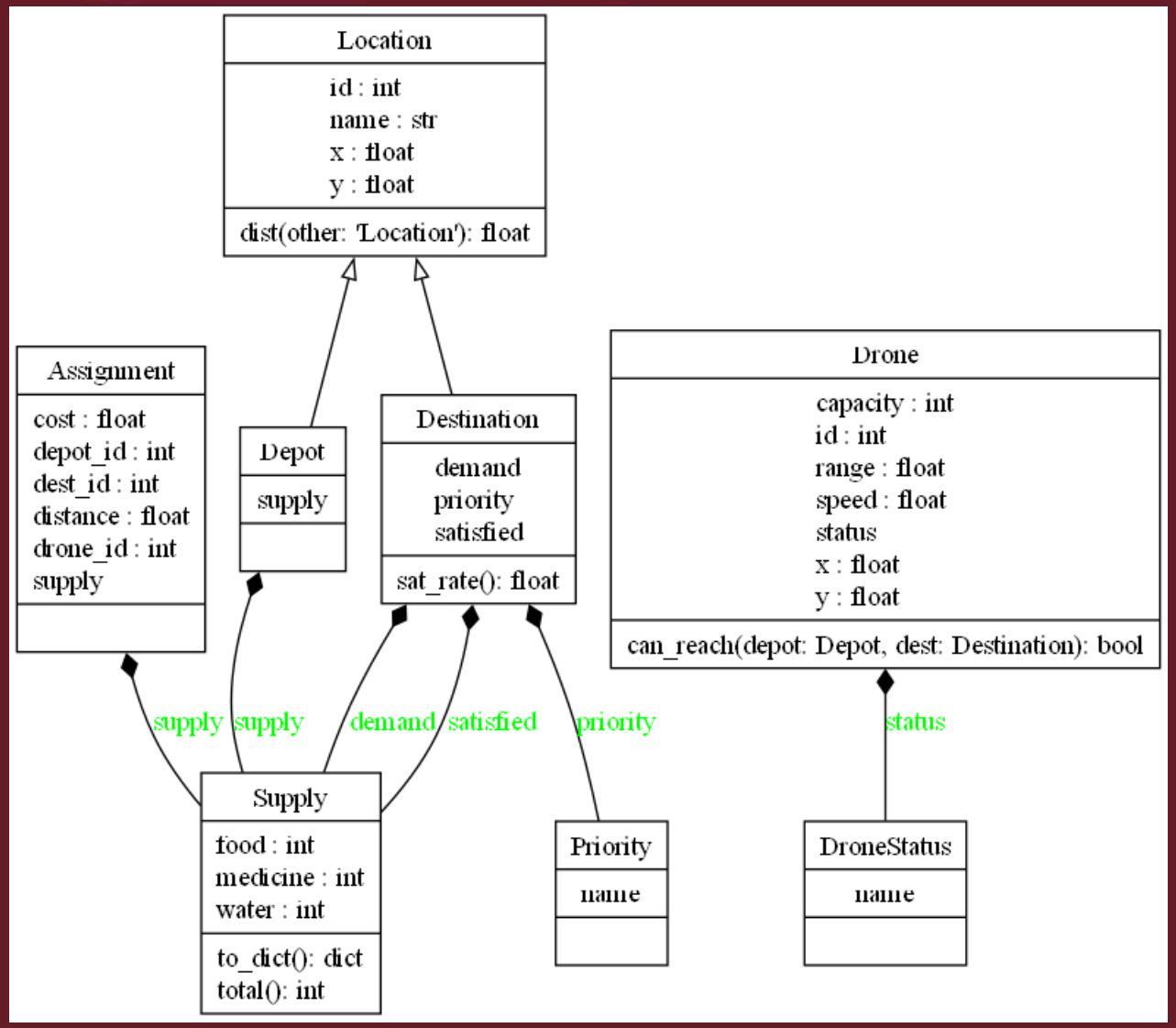


- Η απάντηση στο ερώτημα δίνεται με γραμμικό προγραμματισμό!

- Συγκεκριμένα, αναπτύχθηκε ένα πρόγραμμα σε Python που υπολογίζει τη βέλτιστη ανάθεση αποστολών από αποθήκες σε προορισμούς, με χρήση γραμμικού προγραμματισμού, λαμβάνοντας υπόψη τις ανάγκες, τη διαθεσιμότητα προμηθειών και τις δυνατότητες των δρόνων.

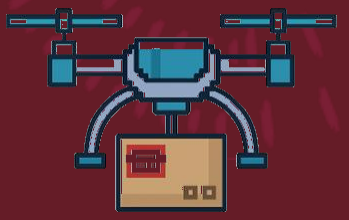
Μοντελοποίηση





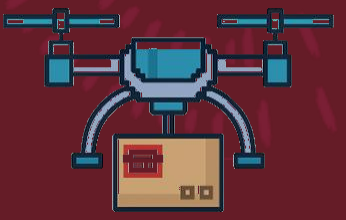
- UML διάγραμμα του αρχείου models.py
- Συνοψίζει τη δομή των βασικών κλάσεων που υλοποιούν το πρόβλημα.

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



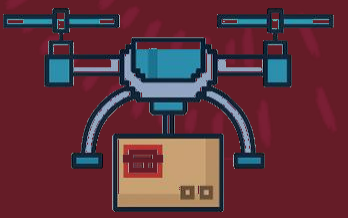
$$Z = \min \left\{ \sum_{d,i,j} \left[\text{dist}(i,j) \cdot w_{\text{priority}}(j) \cdot y_{d_{ij}} \right] + \sum_{j,s} \left[\text{UNMET\_PENALTY} \cdot w_{\text{priority}}(j) \cdot \text{unmet}_{js} \right] \right\}$$

- 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}

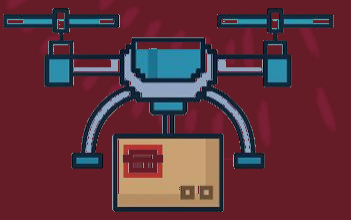
  ώστε να προτιμώνται κρίσιμες αποστολές!

Δημιουργία των σεναρίων



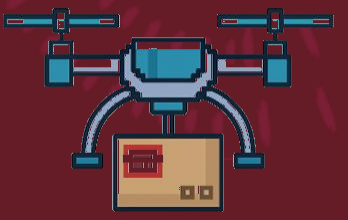
- Με βάση τις δοκιμές και την οπτικοποίηση μέσω animation, το σενάριο big_city_scenario() αποδείχθηκε το πιο κατανοητό και ενδιαφέρον. Το σενάριο περιλαμβάνει 3 δρόνους, 2 σημεία εφοδιασμού και 6 σημεία ανάγκης με διαφορετικές απαιτήσεις και προτεραιότητες.

Animation εκτέλεση



```python
    ];


def main():
    my_scenario = big_city_scenario()

    animator = DroneAnimator(
        scenario = my_scenario,
        map_path = os.path.join(base_dir, 'maps', 'map_backgroun
        dt       = 0.01
    )

    animator.run()

    return;

if __name__ == '__main__':
    main()
```

Αποτέλεσμα στο τερματικό



```
-> 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%
```

- 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!

Ευχαριστώ πολύ για την προσοχή σας!