

Εξελικτική Υπολογιστική- Εργασία

Θεόδωρος- Π. Βαγενάς Α.Ε.Μ : 410

02/05/2019

1 Εισαγωγή

Στόχος της εργασίας είναι η βελτιστοποίηση της αξιοποίησης του αποθέματος ξύλου του εργοστασίου προκειμένου το απόθεμα μετά από την εκπλήρωση κάποιας παραγγελίας να μπορεί να αξιοποιηθεί σε επόμενη παραγγελία χωρίς να χάνεται μεγάλο εμβαδό ξύλου. Επομένως, απαιτείται να μένουν συμπαγή κομμάτια ξύλου μετά από την κοπή. Το πρόβλημα που αντιμετωπίζεται περιλαμβάνει 3 παραγγελίες που έρχονται με τη σειρά και κάποια κομμάτια Stock που μειώνονται ανάλογα.

Χρησιμοποιήθηκαν δύο εξελικτικοί αλγόριθμοι ο PSO και ο DEGL σύμφωνα με τις παραλλαγές που δόθηκαν και αναπτύχθηκε η *FitnessFunction* για την επίτευξη του στόχου όπως θα περιγραφεί σε επόμενο κεφάλαιο. Επιπλέον έγινε σύγκριση και με άλλους αλγόριθμους βελτιστοποίησης και παρουσιάζονται τα αποτελέσματα στο τελευταίο κεφάλαιο. Για την ποσοτικοποίηση του πόσου συμπαγές είναι ένα πολύγωνο χρησιμοποιήθηκε ο τύπος που δίνεται $f_{sm}(\lambda) = 1/(1 + a\lambda)$ όπου $\lambda = (\Omega(CH(O)))/(\Omega(O)) - 1$

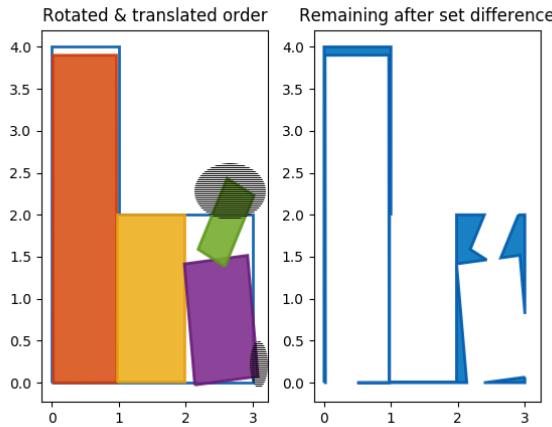
Για την κωδικοποίηση του προβλήματος το οποίο απαιτεί την τοποθέτηση των σχημάτων μέσα στα στοκ χρειάστηκε αυτή η τοποθέτηση να οριστεί σε κάποια μορφή. Επομένως, για κάθε σχήμα η λύση περιλαμβάνει τη θέση x, y και τη γωνία θ κατά την οποία θα μετατοπιστεί και θα στραφεί το εκάστοτε πολύγωνο. Επειδή έχουμε πολλά σχήματα σε κάθε παραγγελία, το πλήθος των μεταβλητών είναι ίσο με το γινόμενο $3 * length(Order)$ και ανάλογα με τον αλγόριθμο φτιάχνεται ο πληθυσμός. Στο τέλος, προκύπτει η τελική λύση η οποία είναι αυτή που έδωσε αποδεκτό αποτέλεσμα όπως θα δειχθεί παρακάτω και περιλαμβάνει τα $[x, y, \theta]$ για κάθε σχήμα κατά τα οποία πρέπει να μετατοπιστεί από την αρχή του θέση (αρχή των αξόνων) και να περιστραφεί προκειμένου να χωρέσει η παραγγελία και μάλιστα με αποδοτικό τρόπο.

2 Περιγραφή σχεδίασης συνάρτησης καταλληλότητας

Παρακάτω παρουσιάζονται οι όροι της συνάρτησης καταλληλότητας που δοκιμάστηκαν σε διάφορες προσπάθειες καθώς και η τελική συνάρτηση. Για την υλοποίηση της χρησιμοποίηθηκε κυρίως σύνδεση των όρων με άθροισμα (+) αλλά έγιναν και προσπάθειες χρήσης πολλαπλασιασμού. Τα τελικά βάρη κάθε όρου δίνονται στη συνολική συνάρτηση.

- Αρχικά χρειάζεται ένας όρος για να αποτρέπει την κοπή πολυγώνων που βρίσκονται έξω από τα όρια του εκάστοτε στοκ. Επειδή ακριβώς ο αλγόριθμος έχει ως στόχο την ελαχιστοποίηση της συνάρτησης τίθεται ένας όρος ο οποίος ισούται με το άθροισμα του εμβαδού του τμήματος των σχημάτων που βρίσκεται έξω από το στοκ. Για να γίνει αυτό και με όσο το δυνατόν λιγότερο χρόνο υπολογίζεται αρχικά η ένωση των σχημάτων με τις νέες θέσεις και στροφές και στη συνέχεια η διαφορά της ένωσης με το στοκ όπως φαίνεται παρακάτω. Το εμβαδόν αυτού του σχήματος είναι αυτό που περιεγράφηκε.

```
unionNewOrder = shapely.ops.cascaded_union(newOrder)
outOfStock = unionNewOrder.difference(Stock)
```

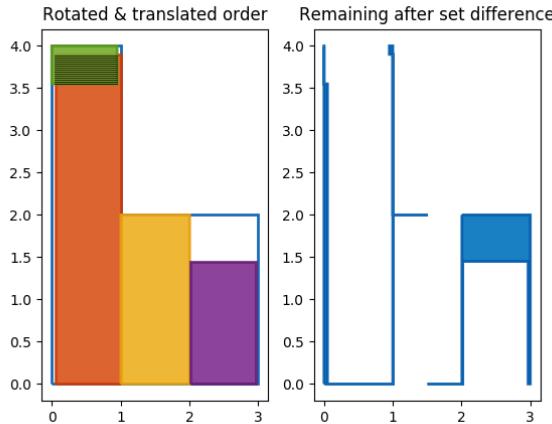


Σχήμα 1: Τοποθέτηση σχημάτων έξω από το στοκ

- Ο δεύτερος στόχος είναι να μην υπάρχει επικάλυψη μεταξύ των πολυγώνων που κόβονται από το ξύλο. Για τον λόγο αυτόν υπολογίζεται το εμβαδόν των σχημάτων που επικαλύπτεται από πολλά σχήματα. Ο όρος που εισάγεται μπορεί να υπολογιστεί με δύο τρόπους. Αρχικά, χρειάζεται το άθροισμα των εμβαδών των σχημάτων της εκάστοτε παραγγελίας και το εμβαδόν την ένωσης των επιμέρους σχημάτων της παραγγελίας με τις τοποθετήσεις από την προτεινόμενη λύση (αυτό έγινε για την επαναχρησιμοποίηση της προηγούμενης ένωσης και ώρα τη μείωση του χρόνου). Στη συνέχεια δημιουργείται ο όρος από την αφάρεση του εμβαδού του άθροισματος μείον το εμβαδόν της ένωσης όπως φαίνεται παρακάτω (αν δηλαδή δεν υπάρχει επικάλυψη η διαφορά πρέπει να βγαίνει μηδενική ενώ αν υπάρχει η διαφορά αυτή εκφράζει το τμήμα που αλληλοεπικαλύπτεται):

$$\begin{aligned} \text{areaSum} &= \sum([newOrder[w].area for w in range(0, len(newOrder))]) \\ \text{overlapArea} &= \text{areaSum} - \text{unionNewOrder.area} \end{aligned}$$

Ο δεύτερος τρόπος είναι ο υπολογισμός του άθροισματος των εμβαδών των τομών των σχημάτων μεταξύ τους αλλά χρησιμοποιήθηκε ο πρώτος για λόγους επιτάχυνσης της εκτέλεσης.



Σχήμα 2: Τοποθέτηση σχημάτων με επικάλυψη

- Ένας άλλος όρος που δοκιμάστηκε είχε ως στόχο την μείωση των αποστάσεων μεταξύ των σχημάτων είτε με τυχαίο τρόπο είτε με φύλνουσα σειρά κατά εμβαδό. Αυτό γινόταν για να έρθουν πιο κοντά τα σχήματα (με σειρά εμβαδού). Η απόσταση αυτή ήταν είτε η μικρότερη απόσταση είτε η απόσταση των centroids που υπολογίζει η βιβλιοθήκη είτε η απόσταση hausdorff_distance της ίδιας βιβλιοθήκης. Τέλος, ο όρος στη συνάρτηση καταλληλότητας ήταν το άθροισμα αυτών των αποστάσεων. Ο υπολογισμός γινόταν ως εξής:

$$\begin{aligned} \text{sortedOrder} &= np.argsort((-1) * np.array([newOrder[i].area for i in range(0, len(newOrder))])) \\ \text{sortedArray} &= [newOrder[w] for w in sortedOrder] \\ \text{dist_sum} &= \sum([sortedArray[i].distance(sortedArray[i + 1]) for i in range(0, len(sortedArray) - 1)]) \end{aligned}$$

- Ένας ακόμη στόχος που υλοποιήθηκε είναι η έλξη των σχημάτων προς την αρχή των αξόνων με χρήση μάλιστα των εμβαδών καθώς με δοκιμές έδωσε καλύτερα οπτικά αποτελέσματα. Για κάθε σχήμα της παραγγελίας όπως διαμορφώνεται από τις ύσεις του *psd*, *degl* υπολογίζεται το άνθροισμα των x και y του *centroid* των σχημάτων πολλαπλασιασμένο με το εμβαδόν του σχήματος. Και στη συνέχεια το άνθροισμα των επιμέρους αιθροισμάτων εισάγεται ως όρος. Με τη χρήση των συντεταγμένων αποφεύχθηκε η χρήση της απόστασης με την αντικατάσταση της με πιο απλή μορφή που εκφράζει και αυτήν την έλξη προς την αρχή των αξόνων.

$$dist_from_zero = sum([newOrder[i].area * (newOrder[i].centroid.x + newOrder[i].centroid.y) \text{ for } i \text{ in range}(0, len(newOrder))])$$

- Επειτα χρησιμοποιήθηκε ο τύπος του *fsm* για το πόσο συμπαγές είναι το υπόλοιπο του στοκ μετά την αφάρεση της παραγγελίας με τις νέες θέσεις. Ο τύπος αυτός διατηρήθηκε και στην τελική συνάρτηση καταλληλότητας.

$$\begin{aligned} ch &= (remaining.convex_hull) \\ lamda &= (ch.area)/(remaining.area) - 1 \\ alpha &= 1.11 \\ fsm &= 1/(1 + alpha * lamda) \end{aligned}$$

Μετά από δοκιμές με διάφορους συνδυασμούς από τους παραπάνω όρους και διαφορετικά βάρη στον καθένα (δοκιμάστηκε και η χρήση κανονικοποίησης για τον καθορισμό βαρών άλλα δεν βοήθησε ιδιαιτέρως) προέκυψε η βασική τελικά συνάρτηση καταλληλότητας. Σε αυτές τις δοκιμές για κάθε μορφή της συνάρτησης εκτελούνταν μερικές εκτελέσεις με διαφορετικό *seed* καθώς ο αλγόριθμος εμπεριέχει τυχαιότητα. Όπως φαίνεται παρακάτω για τους δύο πρώτους όρους που αφορούν την απαίτηση να μην υπάρχει πολύγωνο εκτός του στοκ και να μην υπάρχει επικάλυψη δίνεται μεγάλο βάρος καθώς αποτελούν απαραίτητη προϋπόθεση για να δώσει αποδεκτό αποτέλεσμα ο αλγόριθμος. Αυτό ενισχύεται με το γεγονός ότι για να γίνει αποδεκτή μία προτεινομένη τοποθέτηση υπάρχει ανοχή στο εμβαδόν έξω από το στοκ και για τις επικάλυψεις της τάξεως του 10^{-4} . Στη συνέχεια το επόμενο μεγαλύτερο βάρος δίνεται στον όρο για την ίσο πιο συμπαγή τοποθέτηση των πολυγώνων ενώ δίνεται και ένα μικρότερο βάρος για την “απόσταση από το μηδέν” (όπως αυτή ορίστηκε παραπάνω) καθώς αυτή αποτελεί προτίμηση και όχι απαίτηση. Θεωρήθηκε ότι η τοποθέτηση είναι καλύτερη όταν έλκεται προς μια αρχή στην περίπτωση αυτή στο $(0,0)$.

Παρακάτω παρουσιάζονται οι τελικές συναρτήσεις καταλληλότητας που προέκυψαν με βάση τα παραπάνω χαρακτηριστικά.

- Για τον *PSO* θεωρήθηκε σκόπιμο να παρουσιαστούν δύο παραλλαγές της συνάρτησης καταλληλότητας. Και οι δύο περιλαμβάνουν τους δύο όρους για την αποτροπή της τοποθέτησης εκτός του στοκ και της επικάλυψης. Επίσης περιλαμβάνουν και τον όρο για την μείωση της “απόστασης” από το μηδέν των σχημάτων με βάση το εμβαδό, που αναλύθηκε προηγουμένως. Η 1η παραλλαγή αρκείται σε αυτούς τους 3 όρους και είναι η εξής:

$$res = outOfStock.area * 10000 + overlapArea * 10000 + dist_from_zero * 10$$

Αυτή η συνάρτηση αναφέρεται καθώς παρά την απλότητα της παρατηρήθηκε ότι τα αποτελέσματα της έχουν αρκετό ενδιαφέρον. Σε πολλές περιπτώσεις οπτικά φαίνεται ότι έχουν τοποθετηθεί αρκετά καλά τα σχήματα. Το μειονέκτημα της από την επόμενη παραλλαγή είναι ότι είναι λιγότερη σταθερή ως προς τα αποτελέσματα ενώ η 2η πετυχαίνει και κάποιες μικροβελτιώσεις στην τοποθέτηση που μπορεί να γίνει ακόμα σημαντικότερες καθώς αυξάνονται τα μεγέθη των παραγγελιών. Η 2η παραλλαγή προσθέτει ως τελευταίο όρο τη μετρική για το πόσο συμπαγές τμήμα αφήνει πίσω του το κόψιμο της παραγγελίας. Ετσι με μικρό σχετικά υπολογιστικό κόστος που προστέθηκε τα αποτελέσματα βελτιώνονται με την τοποθέτηση των σχημάτων πιο κοντά και καταλαμβάνουν λιγότερο χώρο.

$$res = outOfStock.area * 10000 + overlapArea * 10000 + dist_from_zero/10 * 10 + 100 * fsm$$

Περισσότερα αποτελέσματα στο αντίστοιχο κεφάλαιο.

- Για τον *DEGL* χρησιμοποιήθηκε και πάλι η 2η μορφή της συνάρτησης καταλληλότητας με λίγο διαφορετικά βάρη στους όρους προκειμένου να επιτευθεί όσο καλύτερο αποτέλεσμα γίνεται. Εδώ δόθηκε λίγο μεγαλύτερο βάρος στα συμπαγή υπόλοιπα πράγμα που το εκμεταλλεύτηκε και καλύτερα ο αλγόριθμος του *DEGL* χρησιμοποιώντας τη γειτονιά:

$$res = outOfStock.area * 1000 + overlapArea * 1000 + dist_from_zero * 1 + 300 * fsm$$

- Τέλος για τους υπόλοιπους αλγόριθμους βελτιστοποίησης χρησιμοποιήθηκε η ίδια συνάρτηση καταλληλότητας με τον *DEGL*:

3 Περιγραφή σχεδίασης ροής τοποθέτησης όλων των παραγγελιών στα στοκ

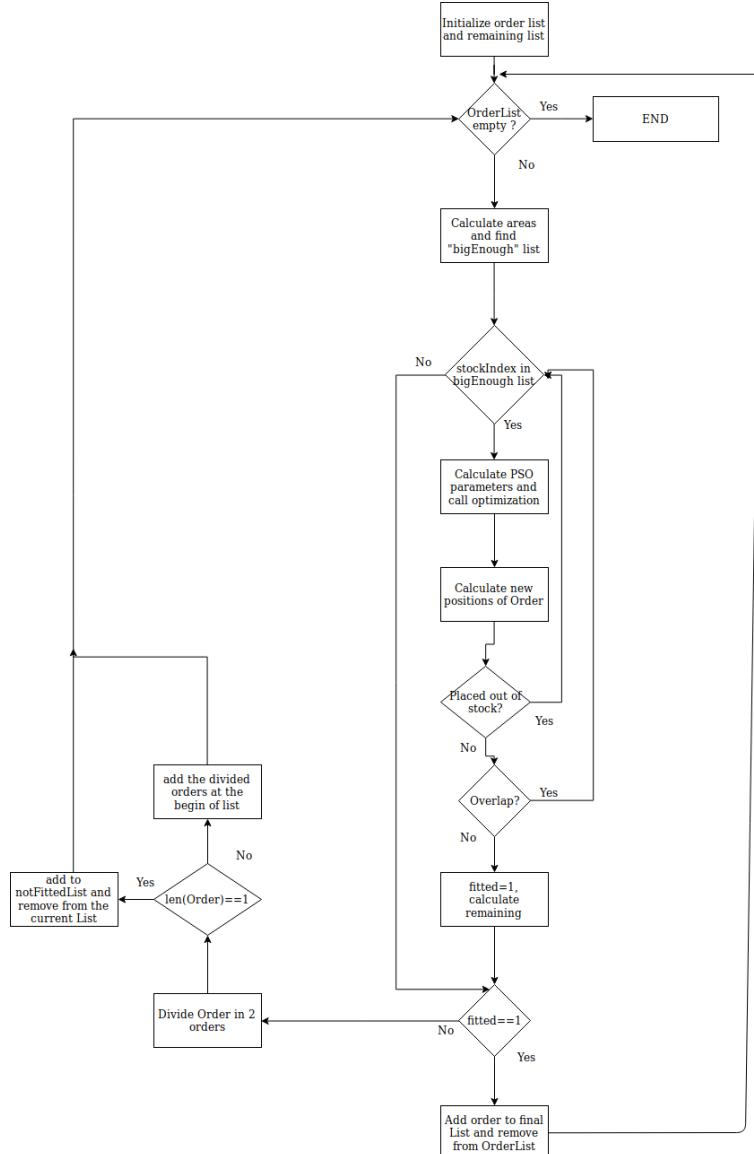
Αρχικά τοποθετούνται όλες οι παραγγελίες σε μία λίστα και αντιγράφονται τα στοκ στη μεταβλητή *remaining*. Θα χρειαστεί μία μεταβλητή για να μετρηθούν πόσα πολύγωνα χώρεσαν μέσα στα στοκ κάθε φορά (για λόγους αξιολόγησης). Κανώς και μία σημαία *fitted* που δηλώνει αν χώρεσε η παραγγελία. Η διαδικασία ξεκινάει με μία επαναληπτική διαδικασία *while* η οποία εκτελείται όσο η λίστα των παραγγελιών δεν είναι άδεια. Αυτή περιλαμβάνει τη λειτουργικότητα της εφαρμογής. Μέσα σε αυτήν τίθεται η 1η παραγγελία ως τρέχουσα και ξεκινούν οι υπολογισμοί. Υπολογίζεται το άθροισμα των εμβαδών των σχημάτων της παραγγελίας (*currentOrderArea*) και ένας πίνακας με τα εμβαδά των στοκ (ή ότι έχει απομείνει από αυτά *remainingsArea*). Για να αποφασιστεί σε ποια στοκ θα δοκιμαστεί αν χωράει η παραγγελία χρησιμοποιείται το εξής σκεπτικό. Δημιουργείται μία λίστα *bigEnough* με τους δείκτες των στοκ που έχουν μεγαλύτερο εμβαδό (*remainingsArea*) από το εμβαδό της παραγγελίας(*currentOrderArea*) που υπολογίστηκε. Στη συνέχεια τα ικάνα στοκ ταξινομούνται σε αύξουσα σειρά με βάση το εμβαδό. Έτσι δημιουργείται μία λίστα με τα στοκ που έχουν μεγαλύτερο εμβαδό από την παραγγελία και με σειρά από το μικρότερο δηλαδή αυτό που αν η παραγγελία τελικά χωρέσει θα αφήσει λιγότερο αναζιοποίητο χώρο στο στοκ προς το μεγαλύτερο.

Το 2o βασικό μέρος είναι μία επαναληπτική διαδικασία *for* που σκανάρει τα στοκ που βρίσκονται στη λίστα *bigEnough* . Μέσα σε αυτή τίθεται το εκάστοτε στοκ (ή υπόλοιπο) ως τρέχον στοκ (*currentStock*) για το οποίο θα γίνουν οι πράξεις στη συνέχεια. Επίσης, ορίζονται με τον τρόπο που φαίνονται στον κώδικα τα άνω και κάτω όρια των μεταβλητών ($[x, y, theta]$) με βάση το τρέχον στοκ. Στη συνέχεια εκτελείται ο αλγόριθμος με βάση τον οποίον γίνονται οι τοποθετήσεις της παραγγελίας στο εκάστοτε διαθέσιμο στοκ. Ο αλγόριθμος μπορεί να είναι ένας από τους παρακάτω: *PSO*, *DEGL*, *Optimize library(Nelder – Mead, L – BFGS – B, SLSQP)* . Μετά την εκτέλεση οποιουδήποτε από τους παραπάνω έχουν υπολογιστεί οι πιθανές θέσεις τοποθέτησης των σχημάτων της παραγγελίας *resultPositions* και με την εφαρμογή των μετασχηματισμών προχύπτει η παραγγελία με τις νέες θέσεις (*newOrder*).

Αναφέρονται ως πιθανές γιατί οι παραπάνω αλγόριθμοι τερματίζονται όταν η συνάρτηση καταλληλότητας δεν βελτιωθεί αρκετά για έναν αριθμό επανολήψεων αλλά αυτό δεν σημαίνει ότι έχουν καταλήξει σε αποτέλεσμα αποδεκτό. Αυτό συμβαίνει είτε γιατί μπορεί να μην χωράει η παραγγελία και άρα η προτεινόμενη λύση να έχει επικαλύψεις ή να βρίσκεται εκτός στοκ, είτε γιατί μπορεί να έγινε νωρίς η σύγκλιση του αλγορίθμου πριν ακόμα βρεί τη βέλτιστη ή κάποια αποδεκτή λύση. Επομένως, μετά από κάθε τέτοια δοκιμή γίνεται έλεγχος αν είναι αποδεκτή η λύση. Ο έλεγχος περιλαμβάνει αρχικά το αν η παραγγελία βρίσκεται εντός του στοκ. Αυτός όπως προαναφέρθηκε ορίζεται με το εμβαδό της παραγγελίας που βρίσκεται εκτός στοκ. Τίθεται μία ανοχή (εδώ 0.0001) όπου αν αυτό το εμβαδό είναι μεγαλύτερο από την ανοχή τότε η τρέχουσα λύση δεν είναι αποδεκτή και η επανάληψη συνεχίζει για την ίδια παραγγελία με το επόμενο στοκ της λίστας *bigEnough* . Ο 2ος έλεγχος είναι για την ύπαρξη επικάλυψης ο οποίος ορίζεται ως η διαφορά του εμβαδού (άθροισμα εμβαδών) των σχημάτων της παραγγελίας μείον το εμβαδό της ένωσης των σχημάτων της παραγγελίας. Έτσι υπολογίζεται το εμβαδόν της επικάλυψης και τίθεται και πάλι ένα πολύ μικρό όριο ανοχής (0.0001) γιατί δεν επιτρέπονται λύσεις με τέτοιες επικαλύψεις. Αν τηρείται και αυτό το κριτήριο σημαίνει ότι η παραγγελία με τις θέσεις όπως προέκυψαν έχει τοποθετηθεί αποδεκτά μέσα στο στοκ. Αυτό δηλώνεται με την τιμή της μεταβλητής *fitted* = 1, αφαιρούνται από το στοκ τα τμήματα της παραγγελίας και η εκτέλεση βγαίνει από την εσωτερική επανάληψη με *break*.

Όταν η εκτέλεση έχει βγει από την επανάληψη *for* υπάρχουν δύο περιπτώσεις είτε να έχει χωρέσει η παραγγελία είτε όχι. Στην περίπτωση που δεν έχει χωρέσει (σε κανένα από τα πιθανά στοκ) η παραγγελία διαιρείται στη μέση σε 2 παραγγελίες και ξανά αρχίζει η μεγάλη επανάληψη αυτήν την φορά προσπαθώντας πρώτα να χωρέσει τη νέα 1η παραγγελία (δηλαδή τη μισή προηγούμενη) και στη συνέχεια τη 2η δηλαδή την υπόλοιπη μισή. Στην περίπτωση που οι τοποθετήσεις δεν έχουν γίνει σωστά τότε μπορεί κάποια παραγγελία με ένα σχήμα να μη χωρέσει τότε αυτό τοποθετείται σε λίστα σχημάτων που δεν χώρεσαν (*notFittedList*) και αφαιρείται από τις παραγγελίες. Αυτό έγινε για να καλυψθούν και περιπτώσεις πραγματικής χρήσης όπου τα στοκ δεν επαρκούν για κάποια παραγγελία οπότε πρέπει να ενισχυθούν με νέα στοκ. Επίσης για τον ίδιο λόγο επιλέχθηκε αν κάποιο σχήμα δεν χωρέσει στα τρέχοντα στοκ αφού μπει στην προηγούμενη λίστα να συνεχίζεται η εκτέλεση των υπόλοιπων παραγγελιών. Αν η *for* έχει τελειώσει με θετικό αποτέλεσμα και έχει τοποθετηθεί σωστά η τρέχουσα παραγγελία αυτή αποθηκεύεται σε μία λίστα με τις θέσεις που έφεραν σωστό αποτέλεσμα για κάθε σχήμα καθώς επίσης και σε ποιο στοκ. Έτσι ώστε να μπορεί να αξιοποιηθεί αυτό το αποτέλεσμα για πραγματική χρήση. Ενώ ταυτόχρονα διαγράφεται η παραγγελία που ικανοποιήθηκε από την λίστα παραγγελιών. Επειτα συνεχίζεται η επανάληψη *while*, τίθεται η επόμενη παραγγελία της λίστας ως η τρέχουσα (θα είναι και πάλι στη θέση μηδέν αφού διαγράφεται η παραγγελία που ικανοποιήθηκε ή αν πρόκειται για σχήμα που δεν χώρεσε πάλι διαγράφεται και μπαίνει στη λίστα *notFittedList*) και η διαδικασία επαναλαμβάνεται μέχρι να αδειάσει η λίστα των παραγγελιών.

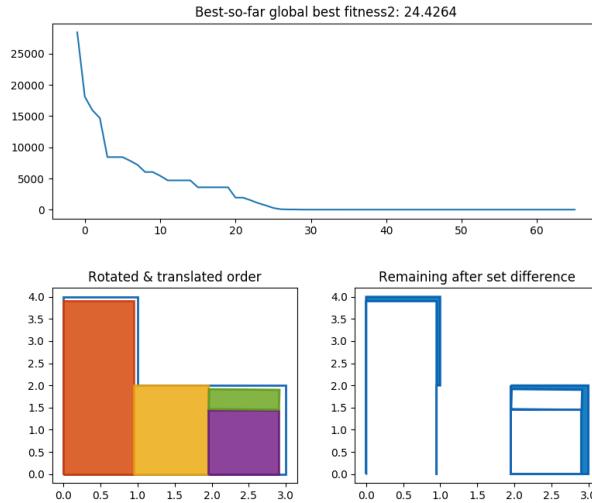
Τέλος για λόγους αξιολόγησης αποθηκεύεται ο χρόνος εκτέλεσης, πόσα πολύγωνα χώρεσαν (για την αξιολόγηση της περίπτωσης που δεν χώρεσαν όλα) και ο μέσος, ελάχιστος και μέγιστος αριθμός επαναλήψεων σε κάθε εκτέλεση των επιμέρους αλγορίθμων (*PSO*, *DEGL* κτλ.). Για τον ίδιο λόγο εκτυπώνεται και ένα σχήμα με όλα τα υπόλοιπα των στοκ όπου οπτικοποιείται η τοποθέτηση των σχημάτων.



Σχήμα 3: Simple Flowchart of the overall process

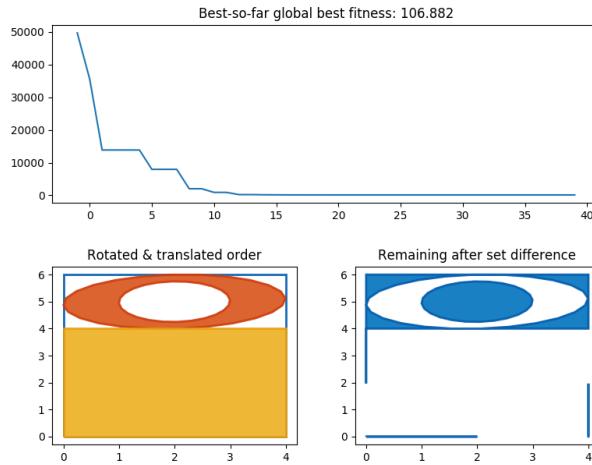
4 Συμπεράσματα- Αποτελέσματα

Αρχικά παρατίθενται κάποια στιγμιότυπα κατά τη διάρκεια της εκτέλεσης του *PSO* για την ικανοποίηση της λίστας των 3 παραγγελιών, παρόμοια είναι και για τους υπόλοιπους αλγορίθμους. Στο 1ο σχήμα (Σχήμα 4) φαίνεται η τελική τοποθέτηση της 1η παραγγελίας στο στοκ μηδέν μαζί με την συνάρτησης καταλληλότητας όπως εξελίχθηκε μέσα από τις επαναλήψεις.



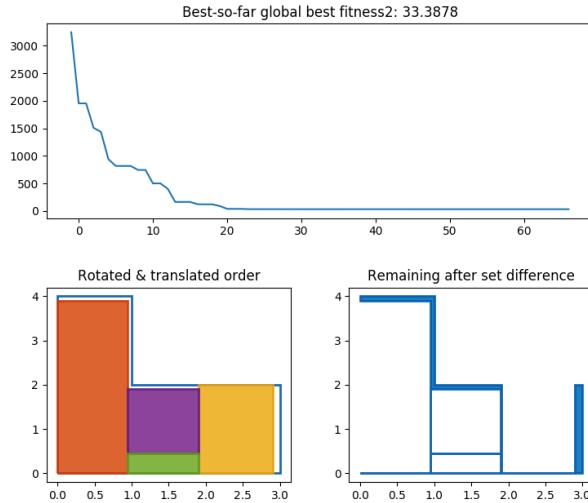
Σχήμα 4: Snapshot1 PSO

Στο 2o σχήμα (Σχήμα 5) παρατηρείται η τοποθέτηση δύο σχημάτων του ενός τμήματος μιας παραγγελίας η οποία λόγω του ότι δεν χώρεσε ολόκληρη σε κάποιο στοκ διασπάστηκε σε δύο παραγγελίες. Όπως επίσης γίνεται αντιληπτή η σταδιακή μείωση της συνάρτησης καταλληλότητας.



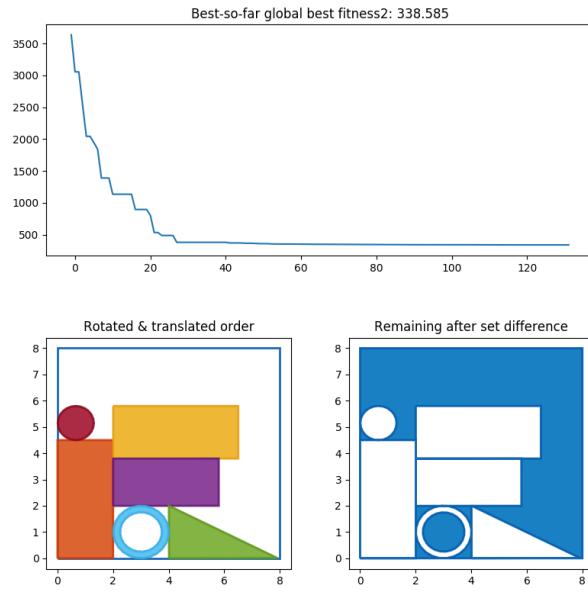
Σχήμα 5: Snapshot2 PSO

Στο 3o σχήμα (Σχήμα 6) φαίνεται και πάλι η ίδια παραγγελία με το 1o αλλά αυτήν την φορά το στιγμιότυπο είναι από τον αλγόριθμο *DEGL*. Και αυτός κάνει πολύ καλή τοποθέτηση με τη διαφορά ότι στον *DEGL* τα βήματα μείωσης της συνάρτησης είναι πιο ομαλά και εδώ συγκλίνει στις ίδιες περίπου επαναλήψεις.



Σχήμα 6: Snapshot3 DEGL

Στο 4ο σχήμα (Σχήμα 7) με τον αλγόριθμο *DEGL* παρουσιάζεται μια άλλη επιμέρους τοποθέτηση. Σε αυτήν ο αλγόριθμος άργησε αρκετά να συγκλίνει με σχεδόν 130 επαναλήψεις αλλά το αποτέλεσμα που πέτυχε είναι αρκετά καλό καθώς τα σχήματα τοποθετήθηκαν αρκετά κοντά και η παραγγελία αυτή χώρεσε ολόκληρη σε ένα στοκ. Η συνάρτηση καταλληλότητας αρχικά μειώνεται αισθητά και μετά τις 30 επαναλήψεις γίνονται μικροβελτιώσεις που άμως οδηγούν σε αυτό το αποτέλεσμα.



Σχήμα 7: Snapshot4 DEGL

4.1 Τρόπος αξιολόγησης

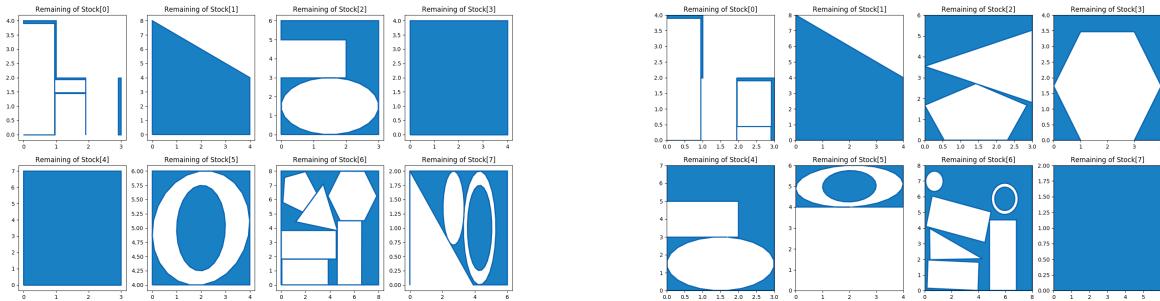
Για την τελική αξιολόγηση των αποτελεσμάτων για τους 2 βασικούς αλγόριθμους (*PSO*, *DEGL*) με την επιλεγμένη συνάρτηση καταλληλότητας πραγματοποιήθηκαν 10 εκτελέσεις και καταγράφηκε σε πόσες από αυτές οι παραγγελίες τοποθετήθηκαν μέσα στα στοκ αλλά και πόσα σχήματα δεν τοποθετήθηκαν στις περιπτώσεις όπου ο αλγόριθμος δεν πέτυχε το επιθυμητό αποτέλεσμα. Για κάθε εκτέλεση αποθηκεύτηκε στο τέλος η εμφάνιση των εναπομείναντων στοκ μετά την τοποθέτηση των σχημάτων μέσω του αντίστοιχου αλγόριθμου, προκειμένου να έχουμε μια εικόνα των τοποθετήσεων μέσα στα στοκ. Αποθηκεύτηκαν επίσης και ο χρόνος εκτέλεσης και κάποια στοιχεία των επιμέρους επαναλήψεων. Στο τέλος παρουσιάζονται κάποια από αυτά που έχουν μεγαλύτερο ενδιαφέρον ενώ το σύνολο των 10 δοκιμών για τους 2 πρώτους αλγόριθμους και των 5 για τους αλγόριθμους *optimization* της *python* δίνονται ολόκληρα στο Παράρτημα.

4.2 Αποτελέσματα

4.2.1 PSO

Μερικές τοποθετήσεις που έχουν ιδιαίτερο ενδιαφέρον βρίσκονται παρακάτω. Μπορεί να παρατηρηθεί ότι η 1η παραγγελία μπορεί να τοποθετηθεί στο 1ο στοκ ενώ η 2η χωράει στο 7ο στοκ ($Stock[6]$) και θεωρητικά η 3η δεν χωράει και πρέπει να διασπαστεί για να τοποθετηθεί. Παρόλα αυτά υπάρχουν και λύσεις όπου μπορεί να μην τοποθετηθεί όλη η παραγγελία που χωράει σε ένα στοκ και πάλι λόγω της προσπάθειας του αλγορίθμου για συμπαγή τμήματα και της έλξης κοντά στο μηδέν να παραχθεί αποτέλεσμα που οπτικά φαίνεται καλύτερο.

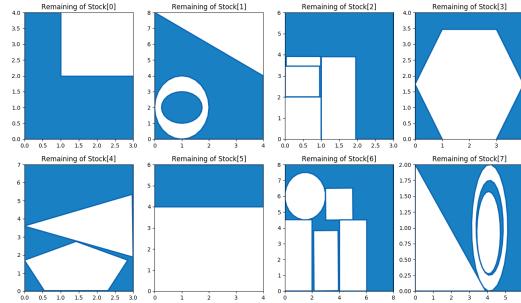
Για την 1η παραλλαγή (πιο απλή) του *PSO* στην 1η εικόνα (Σχήμα 8α) φαίνεται μια τοποθέτηση που παράχθηκε η οποία είναι ίσως από τις καλύτερες που έχουν παραχθεί καθώς αν και δεν μπήκαν ολόκληρες οι δύο παραγγελίες σε 2 στοκ έχουν τοποθετηθεί τα επιμέρους τμήματα τους σε συμπαγή δομή αφήνοντας 3 στοκ τελείως ελεύθερα και γεμίζοντας αρκετά καλά τα υπόλοιπα. Στην 2η εικόνα (Σχήμα 8β) βλέπουμε την περίπτωση όπου τοποθετούνται οι 2 ολόκληρες παραγγελίες σε 2 στοκ και η 3η μοιράζεται. Στην 3η εικόνα (Σχήμα 9) η τοποθέτηση αυτή φαίνεται λιγότερο αποδοτική καθώς έχουν χρησιμοποιηθεί περισσότερα στοκ αν και πάλι τα υπόλοιπα είναι αρκετά συμπαγή σε βαθμό που θα μπορούσαν να αξιοποιηθούν. Έχει εμφανιστεί μια περίπτωση όπου ο αλγόριθμος δεν κατάφερε να τα τοποθετήσει (χώρεσαν 16/17), πράγμα δύνατος που είναι εξαιρετικά σπάνιο καθώς έχει συμβεί μια φορά στο σύνολο των δοκιμών και σε καμία από τις τελικές 10 εκτελέσεις που παρουσιάζονται και στο παρότρημα.



(α') Trial PSO without f

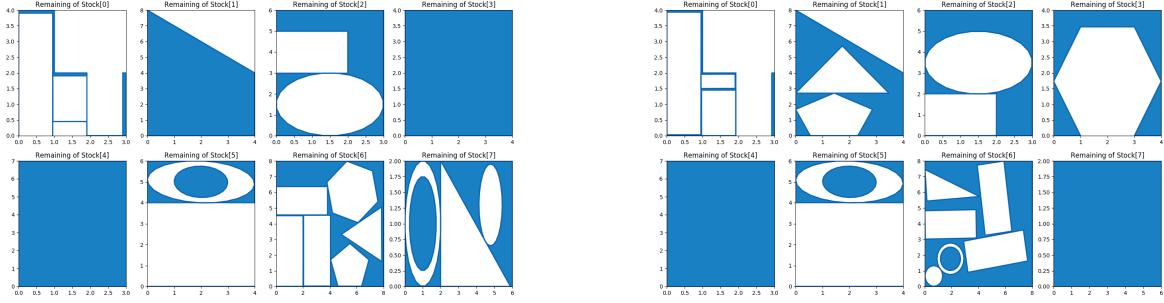
(β') Trial PSO without f

Σχήμα 8: Trial PSO without f (1 and 2)



Σχήμα 9: Trial PSO without f 3

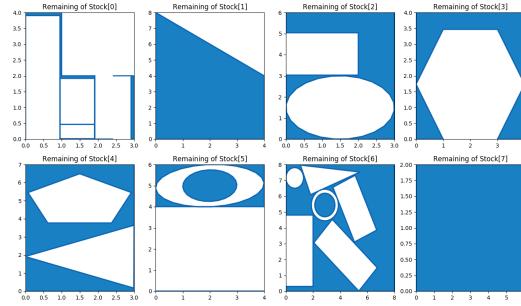
Για την 2η παραλλαγή (με χρήση και της μετρικής για το συμπαγές τμήμα f) τώρα που είναι και πιο αποτελεσμαστική όπως θα γίνει φανερό παρακάτω παρουσιάζονται και εδώ 3 περιπτώσεις. Η 1η περίπτωση είναι μια πολύ καλή τοποθέτηση καθώς περισσέουν 3 στοκ ολόκληρα ενώ ταυτόχρονα οι τοποθετήσεις είναι αρκετά συμπαγείς και χωρίς να αφήνουν αναξιοποίητο χώρο. Η 2η και η 3η περίπτωση μπορούμε να δούμε ότι τοποθετούν τις 2 παραγγελίες στα στοκ που χωράνε και αφήνουν 2 στοκ ελεύθερα πράγμα που δείχνει σε ένα μεγάλο βαθμό ότι ήταν αποδοτικές.



(α') Trial PSO with f

(β') Trial PSO with f

Σχήμα 10: Trial PSO with f (1 and 2)

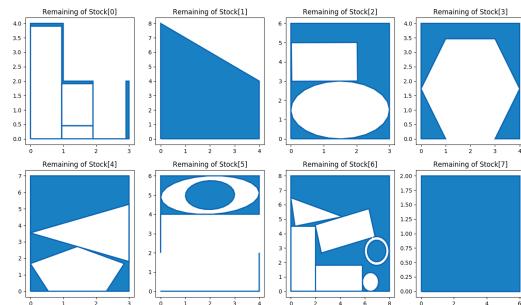


Σχήμα 11: Trial PSO with f 3

Τέλος, όσων αφορά τις 2 συναρτήσεις η 2η περίπτωση με τη χρήση του f αποτελεί καλύτερη επιλογή και επομένως θα χρησιμοποιηθεί για τις συγχρίσεις παράκατω. Αυτό μπορεί να επιβεβαιωθεί χοιτώντας συνολικά τα αποτελέσματα από τις 10 εκτελέσεις και των 2 περιπτώσεων (που υπάρχουν στο παράτημα). Μπορεί να παρατηρηθεί ότι η 2η περίπτωση παράγει πιο σταθερά αποτελέσματα καθώς στις 8/10 φορές αφήνει 2 στοκ ελεύθερα (τη 1 μάλιστα 3) πράγμα που συμβαίνει στις 4/10 φορές της απλής περίπτωσης. Επιπλέον και οι τοποθετήσεις της 2η περίπτωσης είναι αρκετά παρόμοιες μεταξύ τους δείχνοντας ότι δίνει αρκετά σταθερά αποτελέσματα χωρίς ιδιαίτερη επιβάρυνση σε χρονικό κόστος.

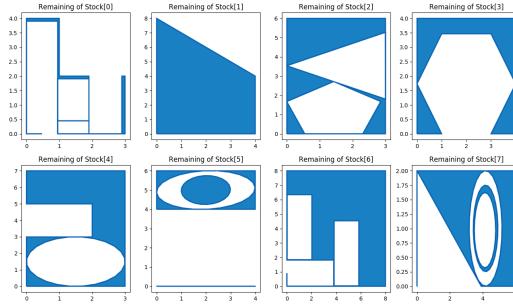
4.2.2 DEGL

Οι πιο ενδιαφέρουσες τοποθετήσεις παρουσιάζονται παρακάτω. Στην 1η (Σχήμα 12) βλέπουμε ότι οι 2 παραγγελίες χώρεσαν στα 2 στοκ και μάλιστα η 2η παραγγελία έχει τοποθετηθεί αρκετά συμπυκνωμένα ώστε να μην αφήνει μεγάλα κενά αφήνοντας χώρο και για κάποιο πιθανό σχήμα που θα έρθει μετά ενώ μένουν τελείως άδεια 2 άλλα στοκ.



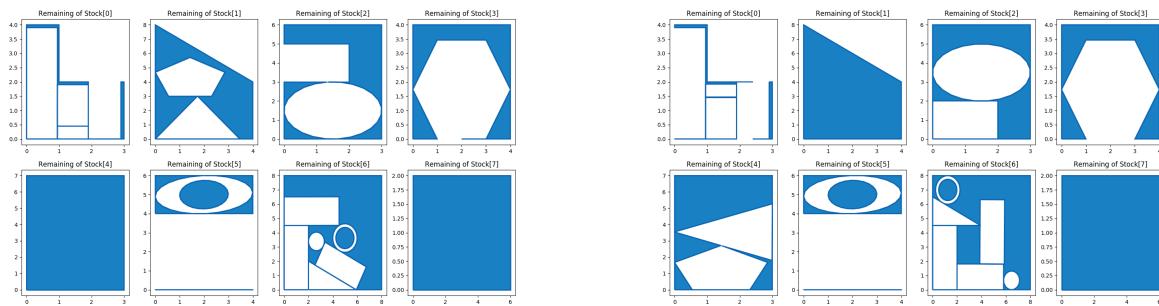
Σχήμα 12: Trial DEGL 1

Στη 2η εικόνα (Σχήμα 13) φαίνεται μία περίπτωση όπου η τοποθέτηση είναι αποδεκτή και αν και έχει χωρίσει και τη 2η παραγγελία πάλι πετυχαίνει ένα αποτέλεσμα όπου τα σχήματα είναι τοποθετημένα αρκετά συμπυκνωμένα και αφήνουν χώρο στα στοκ για τυχόν επόμενη παραγγελία. Σε αυτήν την περίπτωση το μειονέκτημα είναι ότι αφήνει απείραχτο ένα μόνο στοκ αλλά και πάλι το κάνει αφήνοντας χώρο στακ για μικρότερα σχήματα.



Σχήμα 13: Trial DEGL 2

Στη 3η εικόνα (Σχήμα 14) φαίνονται 2 τοποθετήσεις ίσως οι καλύτερες του αλγορίθμου στις 10 προσπάθειες. Και στις δύο μένουν δύο στοκ ολόχληρα ενώ ειδικά στην 1η τοποθέτηση κοιτώντας την 2η παραγγελία τα σχήματα έχουν έρθει αρκετά κοντά με αποτέλεσμα να εξοικονομείται χρήσιμος χώρος και να μειώνονται οι απώλειες δηλαδή τα μικρά τμήματα που προκύπτουν ανάμεσα στις κοπές και δεν μπορούν να αξιοποιηθούν. Και στην 2η περίπτωση οι τοποθετήσεις είναι αρκετά συμπυκνωμένες και μπορεί να ειπωθεί ότι το κενό ανάμεσα στο Stock[6] έχει σχήμα που μπορεί να εκμεταλευθερίζεται.



(α') Trial DEGL 3

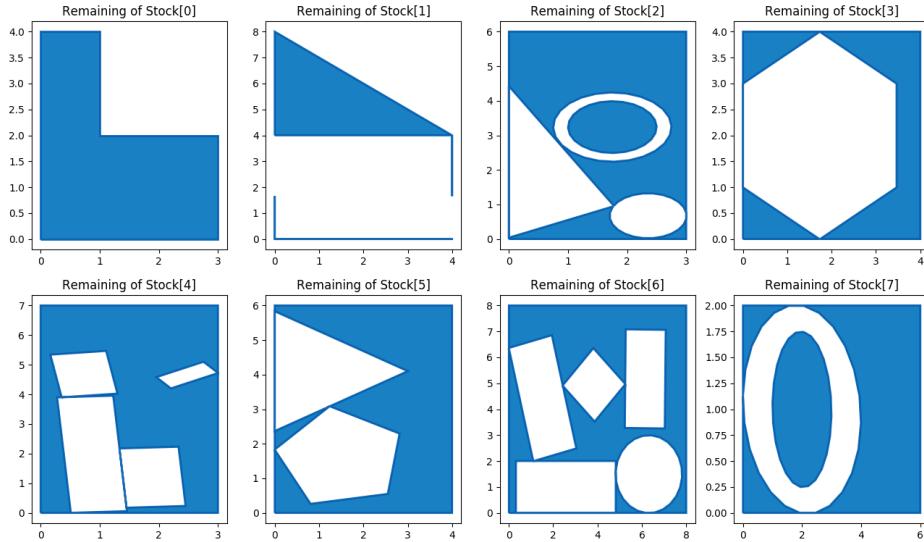
(β') Trial DEGL 4

Σχήμα 14: Trials DEGL (3 and 4)

4.2.3 Other optimize functions

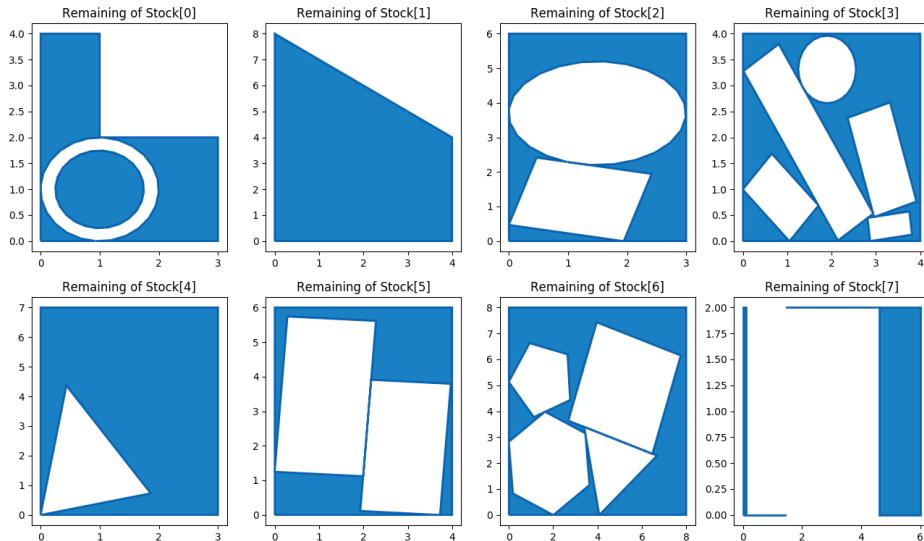
Παρακάτω παρουσιάζονται κάποιες τοποθετήσεις όπως δόθηκαν μέσω των αλγορίθμων Nelder-Mead, L-BFGS-B, SLSQP της βιβλιοθήκης της *python* και του *patternsearch-noisyopt*.

Όσων αφορά τον Nelder – Mead η παραγγελία χώρεσε στις 2 από τις 5 φορές ενώ στις υπόλοιπες 3 δεν χώρεσε ένα σχήμα. Παρακάτω φαίνεται μία εκτέλεση που χώρεσαν όλα αλλά μπορεί να παρατηρηθεί ότι το αποτέλεσμα δεν είναι ικανοποιητικό καθώς περίσσεψε ένα μόνο ολόχληρο στοκ και στα υπόλοιπα οι τοποθετήσεις είναι αραιές αφήνοντας κενά μεταξύ τους που δεν είναι εύκολο να αξιοποιηθούν.



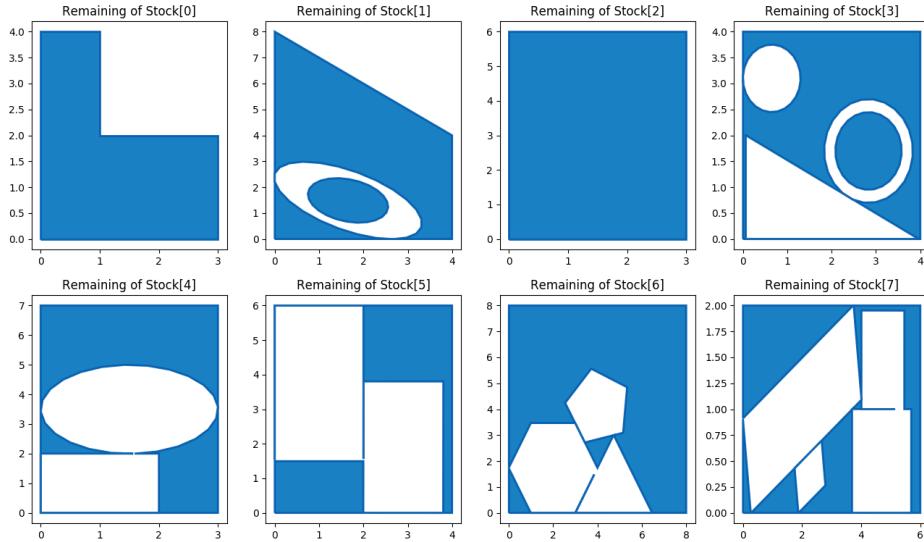
Σχήμα 15: Nelder- Mead

Όσων αφορά τον $L - BFGS - B$ οι παραγγελίες δεν χώρεσαν σε καμία από τις 5 φορές (περίσσεψαν 1-2 σχήματα κάθε φορά). Επομένως, το αποτέλεσμα δεν είναι ικανοποιητικό καθώς δεν χώρεσαν οι παραγγελίες ενώ παρακάτω παρουσιάζεται μία περίπτωση που περίσσεψε 1 σχήμα με μέτριο αποτέλεσμα τοποθετήσεων.



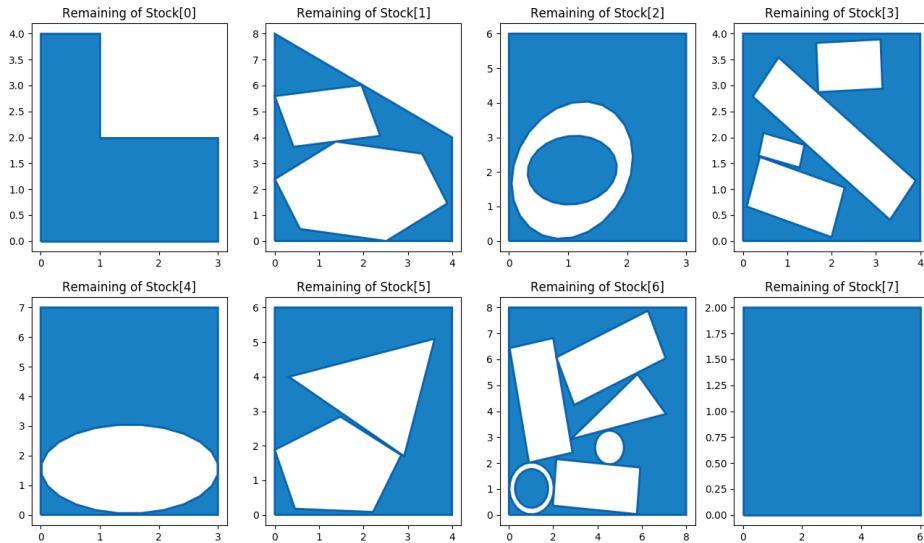
Σχήμα 16: L-BFGS-B

Όσων αφορά τον $SLSQP$ οι παραγγελίες δεν χώρεσαν σε καμία από τις 5 φορές (περίσσεψαν 2-4 σχήματα κάθε φορά). Επομένως, το αποτέλεσμα είναι χειρότερο από τους προηγούμενους καθώς δεν χώρεσαν οι παραγγελίες με περισσότερα σχήματα εκτός του στοκ ενώ παρακάτω παρουσιάζεται μία περίπτωση που περίσσεψε 2 σχήματα με μέτριο αποτέλεσμα τοποθετήσεων.



Σχήμα 17: SLSQP

Όσων αφορά τον *patternsearch – noisyopt* οι παραγγελίες δεν χώρεσαν σε καμία από τις 5 φορές (περίσσεψαν 1-2 σχήματα κάθε φορά). Επομένως, το αποτέλεσμα δεν είναι ούτε εδώ ικανοποιητικό καθώς δεν χώρεσαν οι παραγγελίες ενώ παρακάτω παρουσιάζεται μία εκτέλεση που περίσσεψε 1 σχήμα.



Σχήμα 18: Pattern search (noisyopt)

4.2.4 Comparison

Αφού έχουν παρουσιασθεί και συγχριθεί κάποιες αντιπροσωπευτικές τοποθετήσεις που προέκυψαν από τους παραπάνω αλγόριθμους θα αναλυθούν κάποια αποτελέσματα από τις μετρήσεις που αποθηκεύτηκαν και θα συγχριθούν οι αλγόριθμοι ως προς αυτά. Εκτός από την ποιότητα του αποτελέσματος ως προς τις τοποθετήσεις αξίζει να μελετηθεί και ο χρόνος μέσα στον οποίο εξέρχεται ένα αποτέλεσμα. Στον πίνακα φαίνονται οι χρόνοι εκτέλεσης της διπλής *for* για την παραγωγή αποτελέσματος. Επίσης, καταγράφεται ο μέσος, ο ελάχιστος και ο μέγιστος αριθμός των επαναλήψεων του επικερίου αλγορίθμου (*PSO, DEGL, other*) για να δειχθεί σε πόσες επαναλήψεις παράγεται κάποια τοποθέτηση είτε αποδεκτή είτε όχι. Ο χρόνος για τους *PSO, DEGL* περιλαμβάνει και την παρουσίαση σε

figure των τοποθετήσεων σε πραγματικό χρόνο επομένως ένα σημαντικό μέρος του χρόνου εκτέλεσης αφορά τις εμφανίσεις αυτές και δεν μπορεί να συγχριθεί με το χρόνο των βιβλιοθηκών.

Comparison			
Algorithm / Metrics	Polygons_Fitted (Mean)	Iterations (Mean, Min, Max)	Time(sec) (Mean, Min, Max)
<i>PSO</i>	17	(45,16,126)	(211.5,151,283)*
<i>DEGL</i>	17	(58,16,193)	(421,322,536)*
<i>Optimize (Nelder – Mead)</i>	16.4	-	(308,238,367)
<i>Optimize (L-BFGS-B)</i>	15.8	-	(122,86,141)
<i>Optimize SLSQP</i>	13.6	-	(88,59,117)
<i>PatternSearch Noisyopt</i>	15.4	-	(153,83,349)

* Οι χρόνοι εκτέλεσης περιλαμβάνουν και την εκτύπωση διαγραμμάτων σε πραγματικό χρόνο και των υπολογισμών μεγεθών που παρουσιάστηκαν.

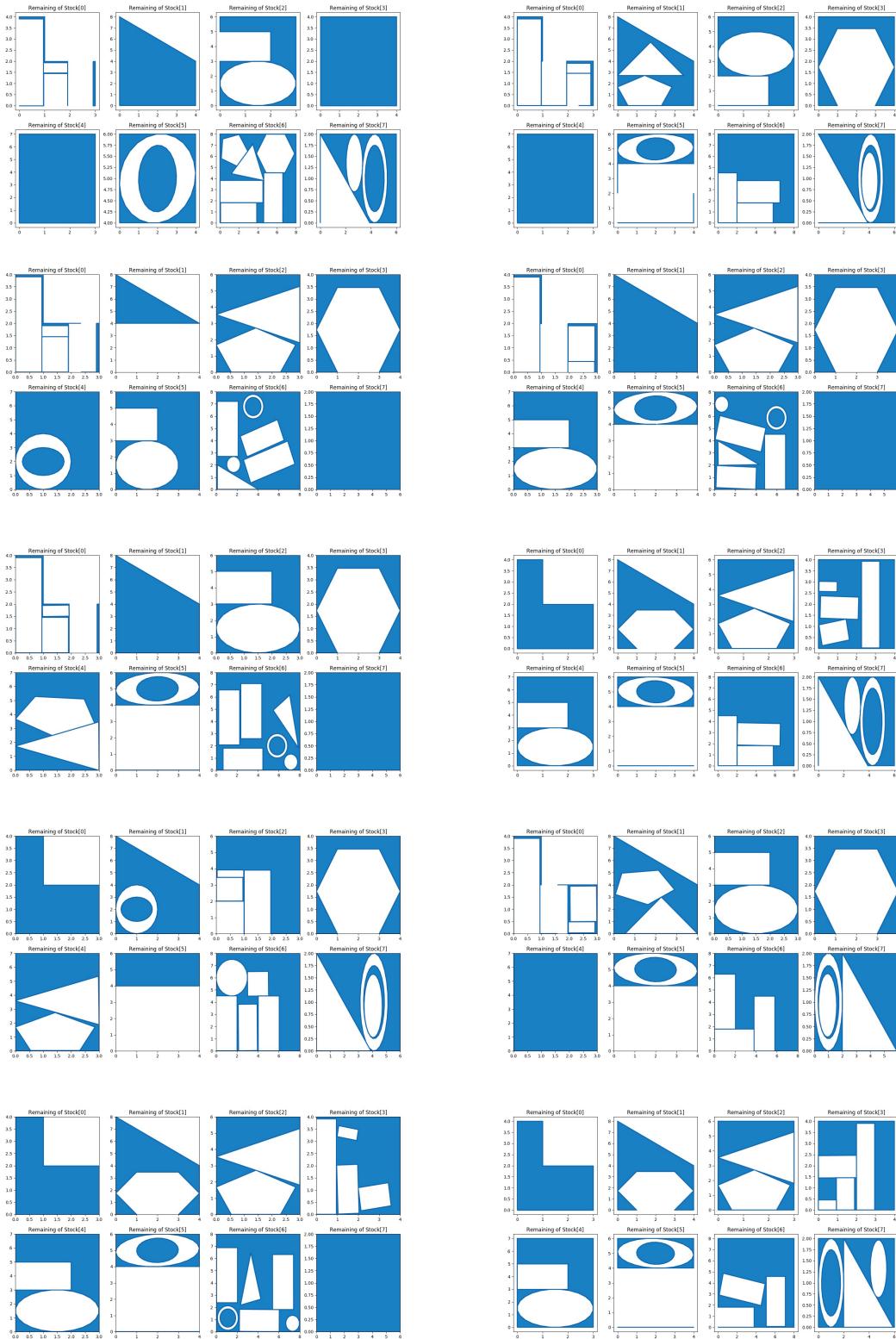
Όσων αφορά την 1η παράμετρο αξιολόγησης που φάνεται στον πίνακα μπορούμε να πούμε ότι ο *PSO*, *DEGL* έχουν πολύ μεγάλα ποσοστά επιτυχίας, στις 10 τουλάχιστον συνεχόμενες τυχαίες δοκιμές μπαίνουν και τα 17 σχήματα μέσα στα στοκ ενώ στους υπόλοιπους αλγόριθμους βλετιστοποίησης οι μέσοι όροι παρουσιάζονται στον πίνακα με χειρότερο τον *SLSQP*. Για τις επαναλήψεις που κάνει κάθε επιμέρους εκτέλεση των *PSO*, *DEGL* για την τοποθέτηση της εκάστοτε παραγγελίας μέσα στο τρέχον στοκ μπορεί να ειπωθεί ότι ο *DEGL* χρειάζεται συνήθως περισσότερες επαναλήψεις όχι 13 παραπάνω επαναλήψεις κατά μέσο όρο πράγμα που αιτιολογείται καθώς κάνει περισσότερες μικρο-βελτιώσεις στην τοποθέτηση. Ο ελάχιστος αριθμός επαναλήψεων είναι ίδιος ενώ ο μέγιστος είναι από τον *DEGL* με αρκετή διαφορά. Οι χρόνοι εκτέλεσης μεταξύ των *PSO* και *DEGL* δείχνουν ότι ο *PSO* είναι αρκετά πιο γρήγορος (ο *DEGL* κάνει κατά μέσο όρο διπλάσιο χρόνο). Οι υπόλοιποι αλγόριθμοι εκτελούνται αρκετά πιο γρήγορα αλλά όπως προαναφέρθηκε δεν μπορούν να συγχριθούν με τους παραπάνω γιατί δεν περιλαμβάνουν εκτύπωση *figures* και επίσης σε αυτόν τον χρόνο και με τις υπάρχουσες ρυθμίσεις δεν καταφέρουν να πετύχουν αποδεκτή λύση που να χωράει όλες τις παραγγελίες (Μόνο ο *Nelder – Mead* τοποθέτησης μία φορά όλα τα στοιχεία μέσα στα στοκ χωρίς καλή τοποθέτηση όμως).

Για τους δύο κύριους αλγόριθμους (*PSO*, *DEGL*) όσων αφορά τις τοποθετήσεις που πέτυχαν μπορεί να ειπωθεί ότι ο *DEGL* πέτυχε κάποιες πιο συμπαγείς τοποθετήσεις σε ορισμένα στοκ που θα εξοικονομούσαν περισσότερο χρήσιμο χώρο. Αυτό όμως δεν είναι απόλυτο καθώς ως προς το σύνολο και ο *PSO* ήταν εξίσου αποτελεσματικός. Ο *PSO* έχει επίσης το θετικό ότι είναι αρκετά πιο γρήγορος από τον *DEGL*. Ο *DEGL* ενδεχομένως δίνει κάποιες βελτιώσεις λόγω της δομής του που αξιοποιεί την τοπικότητα καθώς κοντά σε μία καλή λύση μπορεί να υπάρχει καλύτερη. Μια παρατήρηση που μπορεί να γίνει είναι ότι μεταξύ των εκτέλεσεων υπάρχει μια σταθερότητα στα αποτελέσματα για παράδειγμα στο *Stock*[5] 19/20 φορές (συνολικά από τους 2 αλγόριθμους) τοποθετούνται τα ίδια σχήματα. Και γενικότερα παρά τις αλλαγές στις τοποθετήσεις υπάρχουν κάποια μοτίβα που επαναλαμβάνονται πράγμα που δείχνει από τη μία κάποια θετική σταθερότητα καθώς δίνονται οι ίδιες παραγγελίες κάθε φορά με τα ίδια στοκ ενώ από την άλλη έχουν κάποια ποικιλία προκειμένου να ανακαλύπτονται διάφορες αποδοτικές λύσεις. Αυτό μπορεί να αξιοποιηθεί σε πιθανή ρεαλιστική εφαρμογή με την εκτέλεση του συνόλου μερικές φορές και την επιλογή έπειτα του καλύτερου αποτελέσματος. Αξίζει να σημειωθεί ότι με τον τρόπο που έχει δομηθεί η συνολική επίλυση των τοποθετήσεων το πρόγραμμα μπορεί να παράγει και γενικές λύσεις για διαφορετικό αριθμό ή σειρά παραγγελιών και στοκ.

5 ΠΑΡΑΡΤΗΜΑ

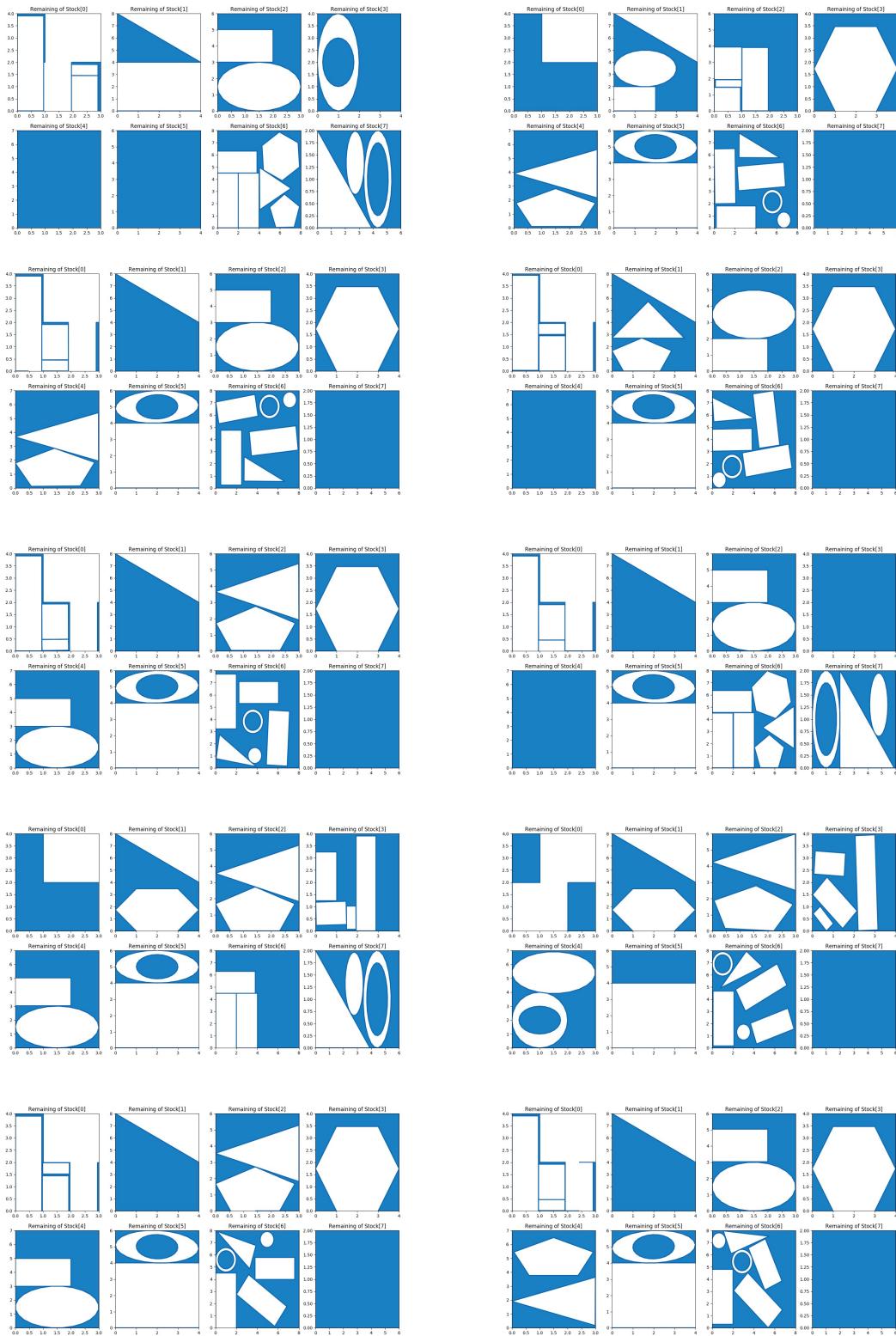
5.1 PSO

PSO ONLY ZERO DISTANCE



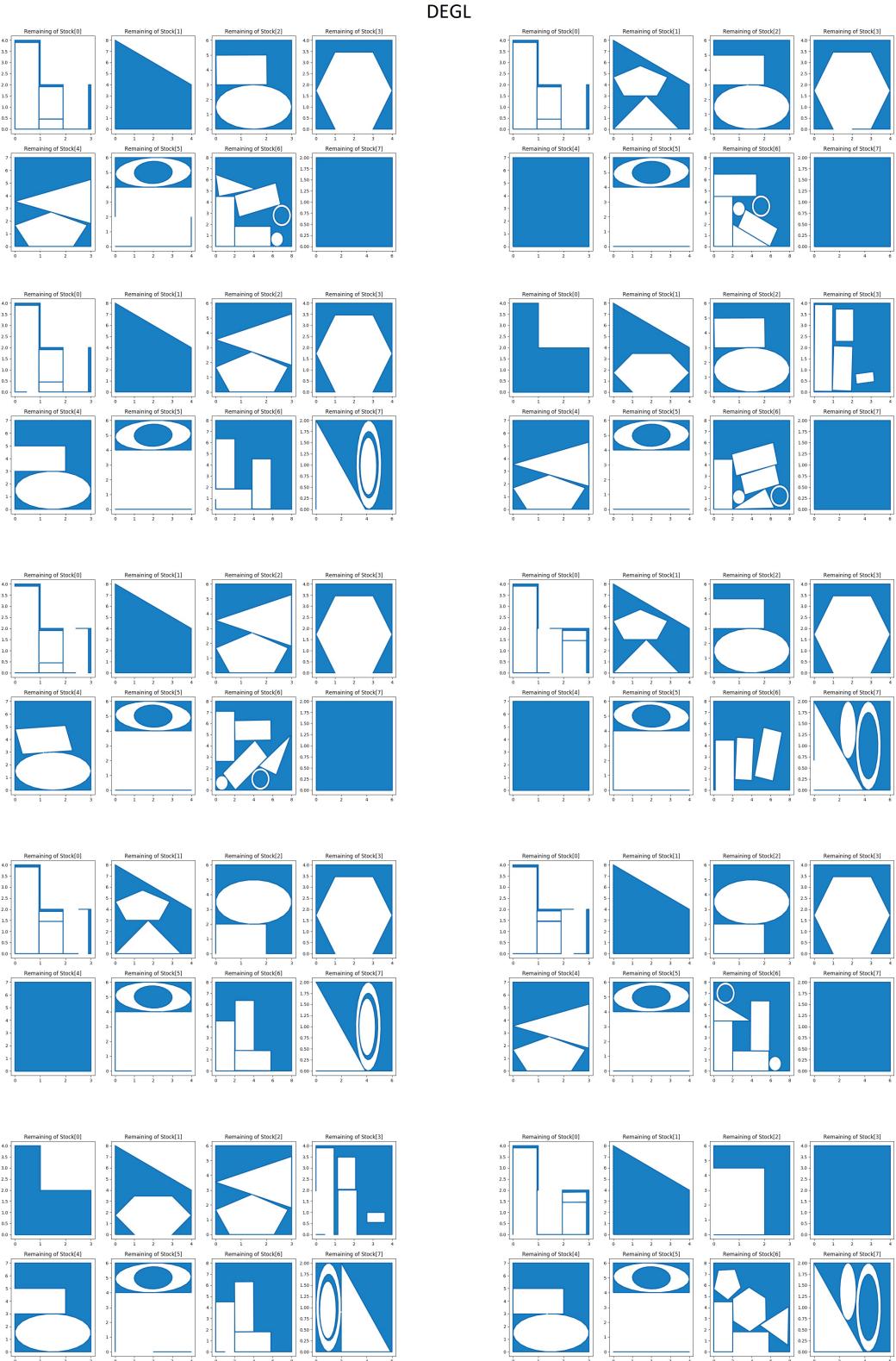
$\Sigma\chi\mu\alpha$ 19: PSO (with only zero distance) 10 results

PSO WITH EXTRA f



$\Sigma\chi\rho\alpha$ 20: PSO (with zero distance and f measure) 10 results

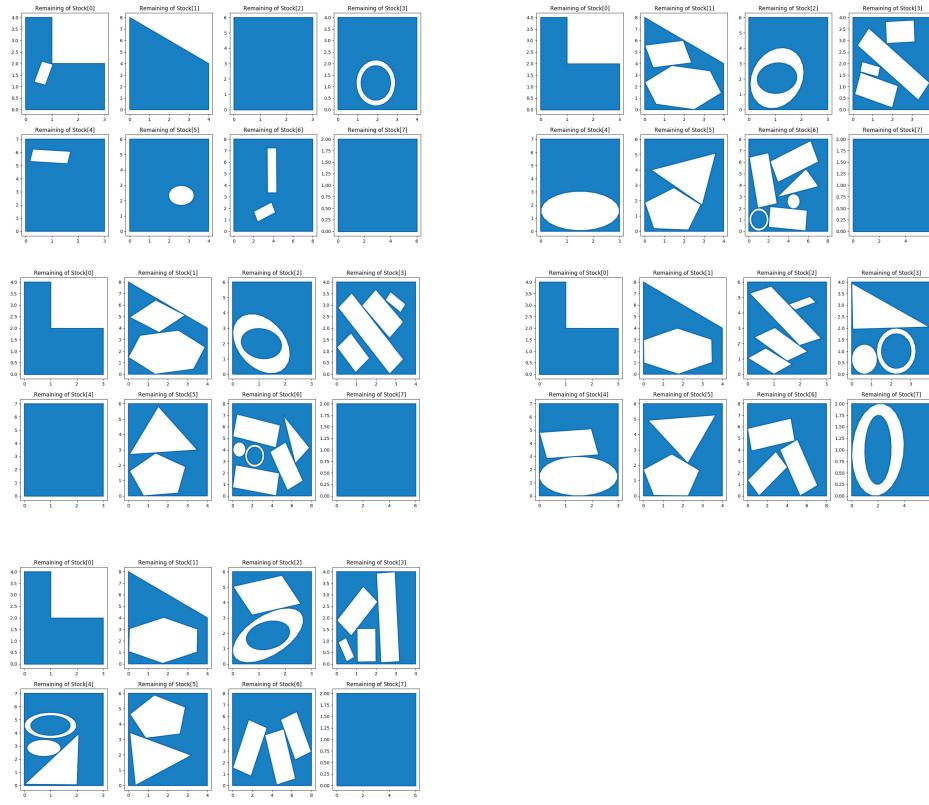
5.2 DEGL



$\Sigma\chi\mu\alpha$ 21: DEGL 10 results

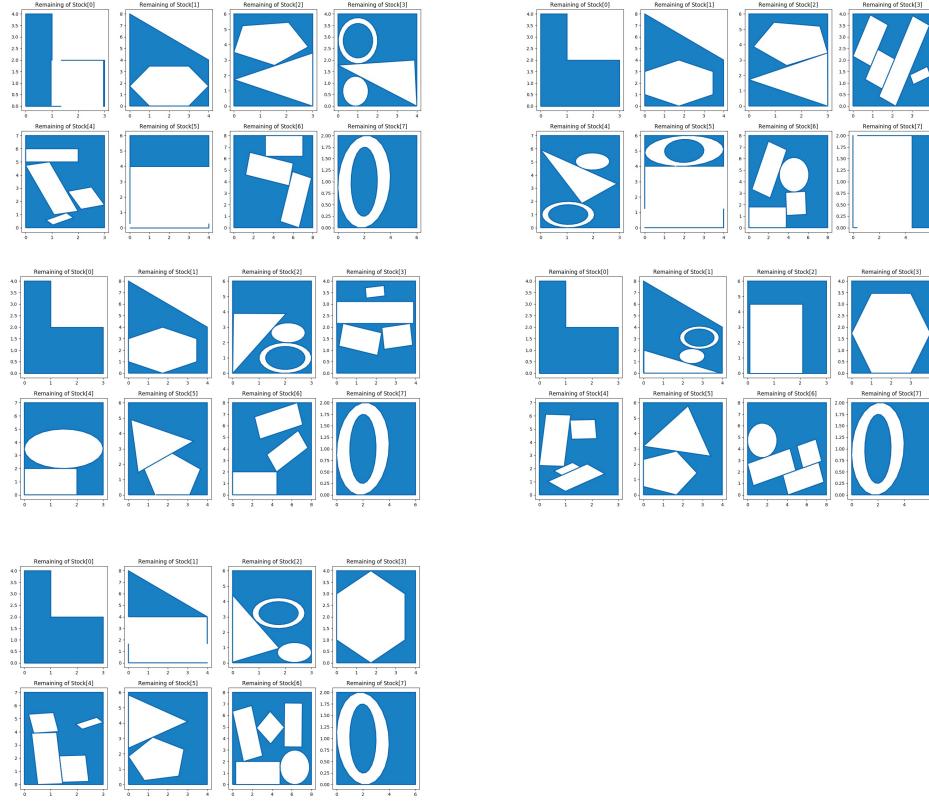
5.3 Optimize

Pattern Search (Noisy Opt)



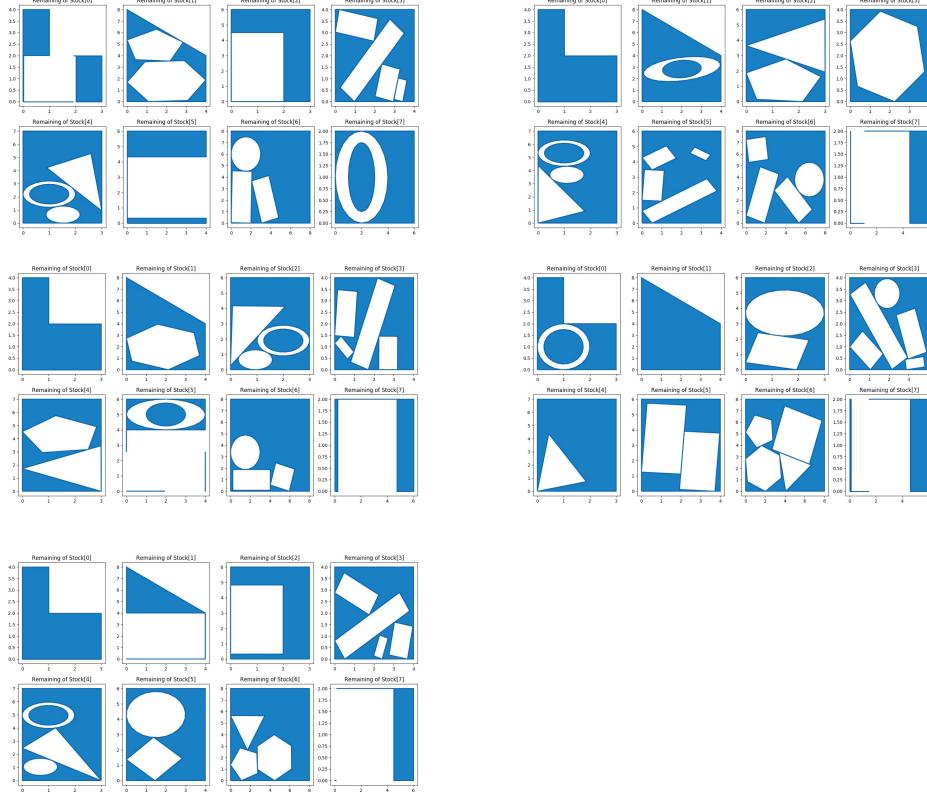
Σχήμα 22: Pattern Search- noisyopt 5 results

Nelder Mead



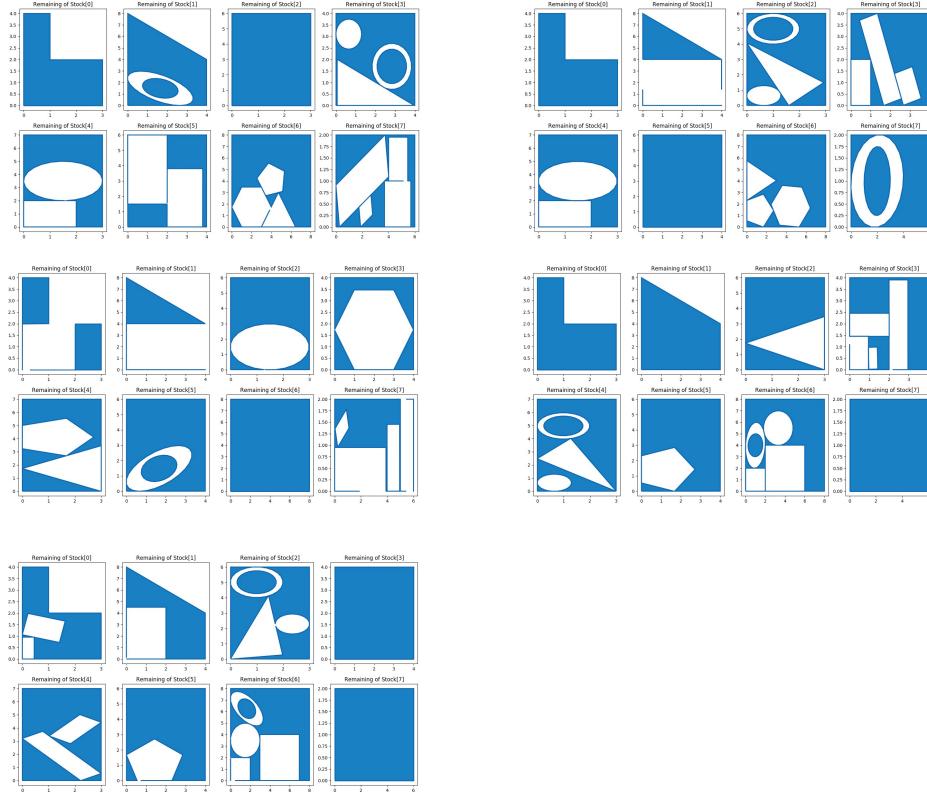
Σχήμα 23: Nelder-Mead 5 results

L-BFGS-B



$\Sigma\chi\gamma\mu\alpha$ 24: L-BFGS-B 5 results

SLSQP



$\Sigma\chi\gamma\mu\alpha$ 25: SLSQP 5 results