ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ευφυείς Μέθοδοι Εξερεύνησης Χώρου με Πολλαπλούς Πράκτορες

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

Αναστασίας Β. Ντόγκα

Επιβλέποντες : Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής
Γεώργιος Σιόλας
Ε.Δ.Ι.Π.

Αθήνα, Σεπτέμβριος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΈΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ευφυείς Μέθοδοι Εξερεύνησης Χώρου με Πολλαπλούς Πράκτορες

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αναστασίας Β. Ντόγκα

Επιβλέποντες : Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής
Γεώργιος Σιόλας
Ε.Δ.Ι.Π..

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Σεπτεμβρίου 2015.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Ανδρέας-Γεώργιος Σταφυλοπάτης	Στέφανος Κόλλιας	Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2015

(Υπογραφή)

.....

Αναστασία Β. Ντόγκα

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

Ευχαριστίες

Θα ήθελα να ευαριστήσω τον επιβλέποντα καθηγητή μου κ. Ανδρέα-Γεώργιο Σταφυλοπάτη για την ευκαιρία που μου έδωσε να ασχοληθώ με αυτή τη διπλωματική εργασία, και την καθοδήγηση που μου παρείχε τόσο στην επιλογή του θέματος όσο και κατά την εκπόνησή της. Θα ήθελα επίσης να ευχαριστήσω τον ερευνητή κ. Γεώργιο Σιόλα για την πολύ καλή συνεργασία που είχαμε, για το ενδιαφέρον του, και για τις εξαιρετικά χρήσιμες συμβουλές του. Ένα τεράστιο ευχαριστώ οφείλω τέλος, στην οικογένειά μου που με στηρίζει με κάθε τρόπο σε όλες μου τις προσπάθειες.

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η εξερεύνηση ενός αγνώστου περιβάλλοντος με χρήση πολλών πρακτόρων-ρομπότ. Για τον σκοπό αυτό θα χρησιμοποιηθεί μια παραλλαγή του αλγορίθμου PSO (Particle Swarm Optimization), ο RDPSO (Robotic Darwinian Particle Swarm Optimization), και η υλοποίησή του θα γίνει στο περιβάλλον Matlab. Στη συνέχεια προστίθεται ο περιορισμός να διατηρείται η επικοινωνία μεταξύ των πρακτόρων (MANET – Mobile Ad hoc NETwork), οπότε προχωράμε σε μία βελτίωση του παραπάνω αλγορίθμου, ώστε να ανταποκρίνεται καλύτερα σε πραγματικά δεδομένα και συνθήκες. Παράλληλα (και στις δύο περιπτώσεις) καταγράφονται οι χαρακτηρηστικές τιμές του περιβάλλοντος (συνάρτηση κόστους) στα σημεία που έχουν εξερευνηθεί προκειμένου να έχουμε και μια εκτίμηση για τις ανεξερεύνητες “γειτονιές”.

Λέξεις Κλειδιά: <<PSO, RDPSO, darwnian, Matlab, ευφυείς αλγόριθμοι, ad hoc, MANET >>

Abstract

The goal of this diploma thesis is to use multiple agents-robots in order to explore an unknown environment. To achieve that, a modification of the PSO (Particle Swarm Optimization) algorithm shall be used, whixh is named RDPSO (Robotic Darwinian Particle Swarm Optimization). The implementation will be in Matlab. A further constraint is also added later on, that requires to maintain communication between the particles (MANET – Mobile Ad hoc NETwork). This leads to a further adjustment of the original algorithm, so that it can respond better to data and conditions closer to reality. In the meantime, in both cases, we record the characteristic values of the environment (cost function) at the points that have been explored, so that we can obtain an estimation for the unexplored "neighborhood".

Keywords: <<PSO, RDPSO, darwnian, Matlab, intelligent algorithms, ad hoc, MANET>>

Πίνακας περιεχομένων

1 Εισαγωγή

1.1 Ορισμοί της Τεχνητής Νοημοσύνης

Η Τεχνητή Νοημοσύνη είναι ένα πεδίο της επιστήμης των υπολογιστών που έχει γνωρίσει μεγάλη άνθιση τις τελευταίες δεκαετίες. Πρόκειται επίσης για μία έννοια με πολύ ευρύ αντικείμενο, κάτι που καθιστά την περιγραφή της μέσω ενός και μόνο ορισμού εξαιρετικά δύσκολη. Για το λόγο αυτό παρατίθενται κάποιοι από τους πιο δημοφιλείς ορισμούς, χωρισμένοι σε τέσσερις κατηγορίες, όπως αυτές ορίστηκαν από τους S. Russel και P. Norvig στην εισαγωγή του βιβλίου τους "Τεχνητή Νοημοσύνη – Μια σύγχρονη προσέγγιση" [1].

Κατηγορία 1 – Συστήματα που σκέπτονται σαν τον άνθρωπο

"[Η αυτοματοποίηση των] δραστηριοτήτων που συσχετίζουμε με την ανθρώπινη σκέψη, όπως η λήψη αποφάσεων, η επίλυση προβλημάτων, η μάθηση..." (Bellman, 1978)

"Η συναρπαστική νέα προσπάθεια για να κάνουμε τους υπολογιστές να σκέπτονται … μηχανές με νόηση, με την πλήρη και κυριολεκτική έννοια." (Haugeland, 1985)

Κατηγορία 2 – Συστήματα που ενεργούν σαν τον άνθρωπο

"Η τέχνη της δημιουργίας μηχανών που πραγματοποιούν λειτουργίες, οι οποίες απαιτούν νοημοσύνη όταν πραγματοποιούνται από ανθρώπους." (Kurzweil, 1990)

"Η μελέτη τους πώς μπορούμε να κάνουμε τους υπολογιστές να κάνουν πράγματα στα οποία, προς το παρόν, οι άνθρωποι είναι καλύτεροι" (Rich και Night, 1991)

Κατηγορία 3 – Συστήματα που σκέπτονται ορθολογικά

"Η μελέτη των νοητικών ικανοτήτων με τη χρήση υπολογιστικών μοντέλων." (Carniak και McDermott, 1985)

"Η μελέτη των υπολογιστικών εργασιών που μας δίνουν τη δυνατότητα να αντιλαμβανόμαστε, να συλλογιζόμαστε, και να ενεργούμε." (Winston, 1992)

Κατηγορία 4 – Συστήματα που ενεργούν ορθολογικά

"Υπολογιστική Νοημοσύνη είναι η μελέτη της σχεδίασης ευφυών πρακτόρων." (Poole κ.ά., 1998)

"Η τεχνητή Νοημοσύνη ασχολείται με την ευφυή συμπεριφορά των τεχνουργημάτων." (Nilsson, 1998)

Υπάρχουν φυσικά πολλοί ακόμη ορισμοί, αλλά οι παραπάνω είναι ενδεικτικοί της σημασίας και του φάσματος των θεμάτων με τα οποία ασχολείται ο κλάδος της Τεχνητής Νοημοσύνης.

1.2 Ιστορική Αναδρομή

Η Τεχνητή Νοημοσύνη ώς έννοια έκανε την εμφάνισή της ανάμεσα στα τέλη της δεκαετίας του 1930 και στις αρχές της δεκαετίας του 1950. Έναυσμα ήταν οι πρόσφατες ανακαλύψεις

της νευρολογίας που μαρτυρούσαν την δομή του εγκεφάλου ως ηλεκτρικό δίκτυο. Με τη συμβολή κορυφαίων προσωπικοτήτων των εμπλεκόμενων πεδίων, όπως ήταν ο Claude Shannon στην Θεωρία Πληροφορίας και ο Alan Turing στην Θεωρία Υπολογισμού, γεννήθηκε για πρώτη φορά η ιδέα της κατασκευής ενός ηλεκτρονικού εγκεφάλου. Καταλυτική επίδραση στον τομέα αυτόν είχε ο Turing με το γνωστό Τεστ του Turing (Turing's Test), σύμφωνα με το οποίο αν μια μηχανή μπορούσε να πραγματοποιήσει μια συζήτηση με έναν άνθρωπο, χωρίς αυτός να μπορεί να ξεχωρίσει αν πρόκειται για μηχανή ή όχι, τότε η μηχανή αυτή “σκέφτεται”.

Επίσημα, η Τεχνητή Νοημοσύνη ως όρος θεμελιώθηκε στο Συνέδριο του Dartmouth το 1956. Το συνέδριο αυτο διοργάνωσαν οι Marvin Minsky και John McCarthy μαζί με τους Claude Shannon και Nathan Rochester της IBM, ενώ έλαβαν μέρος μερικά από τα σημαντικότερα άτομα στον χώρο αυτό τα μετέπειτα χρόνια.

Χαρακτηριστικό στην εξέλιξη του κλάδου αυτού είναι οι περίοδοι καλοκαιριού και χειμώνα (AI Summer and Winter) όπως αποκαλούνται, περίοδοι δηλαδή έντονου ερευνητικού ενδιαφέροντος, υπέρμετρης αισιοδοξίας για τα αποτελέσματα που θα επιτευχθούν και αυξημένης χρηματοδότησης, και περίοδοι αδράνειας, λόγω της διάψευσης των πρότερων ελπίδων.

Τα πρώτα χρόνια του κλάδου, 1956-1974, χαρακτηρίζονται ως η “χρυσή εποχή”. Το διάστημα αυτό αναπτύχθηκαν προγράμματα που για τους περισσότερους ανθρώπους ήταν κάτι παραπάνω από εντυπωσιακά, καθώς έβλεπαν υπολογιστές να λύνουν προβλήματα άλγεβρας, να αποδεικνύουν θεωρήματα γεωματρίας και να μαθαίνουν να μιλούν Αγγλικά. Οι ερευνητές διατύπωναν με έντονη αισιοδοξία την πρόβλεψη ότι σε λιγότερο από 20 χρόνια θα είχε φτιαχτεί ένα πλήρως νοήμον μηχάνημα, και κυβερνητικές υπηρεσίες (κυρίως των Ηνωμένων Πολιτειών) προσέφεραν άφθονα χρήματα στο νέο αυτό επιστημονικό πεδίο.

Ο πρώτος “χειμώνας” της Τεχνητής Νοημοσύνης ήρθε στη δεκαετία του '70 (1974-1980). Την εποχή αυτή ασκήθηκε κριτική στην πρόοδο του πεδίου και μειώθηκε σημαντικά η οικονομική ενίσχυση, λόγω της μεγάλης απόκλισης των προβλέψεων και των στόχων των προηγούμενων χρόνων έναντι των πραγματικών αποτελεσμάτων. Το σημαντικότερο πρόβλημα όμως που είχαν να αντιμετωπίσουν οι ερευνητές ήταν η περιορισμένη

υοπολογιστική ισχύς που είχαν στην διάθεσή τους την εποχή αυτή. Χαρακτηριστική είναι η αναλογία που χρησιμοποίησε ο Hans Moravec το 1976 για να τονίσει την αδυναμία των τότε υπολογιστών να επιδείξουν νοημοσύνη. Συγκεκριμένα αναφέρει ότι η τεχνητή νοημοσύνη έχει τόση ανάγκη από αυξημένη υπολογιστική ισχύ όσο και η αεροπλοΐα την ιπποδύναμη των κινητήρων. Ανάλογα περιορισμένη ήταν και η διαθέσιμη μνήμη για να αποθηκευτούν όλες οι πληροφορίες που απαιτεί ένα σύστημα για να βγάλει συμπεράσματα ή/και για να δράσει σύμφωνα με το συγκεκριμένο περιβάλλον στο οποίο βρίσκεται. Αυτό ήταν αισθητό σε προβλήματα όρασης υπολογιστών και επεξεργασίας φυσικής γλώσσας, όπου το σύνολο των δεδομένων ήταν πραγματικά υπέρογκο. Εξαιτίας, λοιπόν, των ανυπέρβλητων αυτών δυσκολιών (και των αυξημένων προσδοκιών που προαναφέρθηκαν) σταμάτησε το μεγαλύτερο μέρος της χρηματοδότησης προς προγράμματα με μη συγκεκριμένους στόχους, και ανακατευθύνθηκε προς στρατιωτικά κυρίως προγράμματα με άμεσα αποτελέσματα. Αξίζει ωστόσο να σημειωθεί ότι ένα από τα επιτεύγματα της περιόδου αυτής είναι η ανάπτυξη της λογικής και της γλώσσας Prolog.

Την δεκαετία του '80 παρατηρείται ευρεία χρήση των “έμπειρων συστημάτων” (expert systems) από εταιρίες, με αποτέλεσμα η έρευνα της Τεχνητής Νοημοσύνης να κατευθυνθεί προς το χώρο της γνώσης. Τα συστήματα αυτά απαντούν σε ερωτήσεις ή επιλύουν προβλήματα σε ένα συγκεκριμένο γνωσιακό πεδίο, με τη χρήση λογικών κανόνων που προκύπτουν από τις γνώσεις ειδημόνων (experts) στο αντικείμενο αυτό. Πρώιμα τέτοια συστήματα γνώσεις ήταν το Dendral (1965) που αναγνώριζε συστατικά ενός μείγματος από φασματικές μετρήσεις και το MYCIN (1972) που έκανε διαγνώσεις για λοιμώδεις ασθένειες του αίματος, αλλά η μεγαλύτερη επιτυχία ήρθε με το XCON, το οποίο αναπτύχθηκε το 1980 από το CMU για την Digital Equipment Corporation και κατόρθωσε μέχρι το 1986 να εξοικονομεί για την εταιρία 40 εκατομμύρια δολάρια ετησίως.

Έτσι, με την ανάπτυξη των ευφυών αυτών συστημάτων και την αυξημάνεη πάλι χρηματοδότηση, έχουμε για ακόμα μία φορά ανάπτυξη στον κλάδο της Τεχνητής Νοημοσύνης, η οποία χρονικά τοποθετείται στην περίοδο 1980-1987. Το διάστημα αυτό, ο Hopfield απέδειξε ότι ένα ιδιαίτερο είδος νευρωνικών δικτύων (τα δίκτυα Hopfield) μπορούν να μαθαίνουν και να επεξεργάζονται πληροφορίες με έναν εντελώς καινούργιο τρόπο, και ο Rumelhart γνωστοποίησε τη μέθοδο backpropagation για την εκπαίδευση νευρωνικών δικτύων.

Ο επόμενος “χειμώνας” διήρκησε από το 1987 έως το 1993 και οφειλόταν για ακόμα μια φορά κυρίως στις αυξημένες προσδοκίες και υποσχέσεις των επιστημόνων για το τι ήταν εφικτό. Επιπροσθέτως, οι υπολογιστές της IBM και της Apple κέρδιζαν συνεχώς έδαφος στο κομμάτι της ταχύτητας και της ισχύος και το 1987 ξεπέρασαν τα “Lisp machines” που κατασκευάζονταν από την Symbolics για προβλήματα ΤΝ και ήταν κατά πολύ πιο ακριβά. Χωρίς να υπάρχει συγκεκριμένος λόγος να τα αγοράζει κανείς πλέον, μια ολόκληρη βιομηχανία αξίας μισού δισεκατομμυρίου δολαρίων κατέρρευσε εν μία νυκτί. Παράλληλα τα expert systems αποδείχθηκαν ασύμφορα να συντηρηθούν, όχι ικανά να μαθαίνουν και επιρρεπή σε σφάλματα όταν η είσοδος που δινόταν γινόταν απρόβλεπτη.

Στα τέλη της δεκαετίας του '80 εισήχθη και μία νέα προσέγγιση στο θέμα της ΤΝ, η οποία βασιζόταν στην ρομποτική. Υποστηρίχθηκε ότι προκειμένου να υπάρξει ουσιαστική νοημοσύνη, το μηχάνημα θα πρέπει να διαθέτει “σώμα” - έναν τρόπο δηλαδή να αντιλαμβάνεται το περιβάλλον του, να κινείται και να επιβιώνει μέσα σε αυτό, και γενικότερα να αλληλεπιδρά. Με την άποψη αυτή να κυριαρχεί στους επιστημονικούς κύκλους, η έρευνα κατευθύνθηκε προς αυτό το σημείο για τα επόμενα χρόνια, αναθεωρώντας τα μέχρι τότε δεδομένα.

Σήμερα η ΤΝ, μετρώντας πάνω από μισό αιώνα ζωής, έχει επιτύχει επιτέλους κάποιους από τους παλαιότερους στόχους της, εν μέρει λόγω της αυξανόμενης υπολογιστικής ισχύος και εν μέρεις λόγω του ότι οι ερευνητές επικεντρώθηκαν σε συγκεκριμένα και μεμονωμένα προβλήματα, προσπαθώντας να τα εξετάσουν με γνώσεις που αποκτήθηκαν από διάφορα επιστημονικά πεδία. Έτσι, το 1997 ο Deep Blue έγινε ο πρώτος υπολογιστής που νίκησε στο σκάκι έναν παγκόσμιο πρωταθλητή, τον Garry Gasparov, το 2005 και το 2007 δύο ρομπότ κέρδισαν σε διαγωνισμούς της DARPA οδηγώντας αυτόνομα σε έναν δρόμο στην έρημο και σε ένα αστικό περιβάλλον αντίστοιχα, ενώ το 2011 ο Watson της IBM κέρδισε σε έναν δημοφιλή αμερικανικό διαγωνισμό γνώσεων, το “Jeopardy!”, τους δύο μεγαλύτερους “πρωταθλητές”. Παράλληλα η έννοια του “ευφυούς πράκτορα” γινόταν ολοένα ευρύτερα αποδεκτή, προσφέροντας νέες δυνατότητες στην επίλυση προβλημάτων.

Πλέον η τεχνητή νοημοσύνη κρύβεται στα παρασκήνια, με τους αλγορίθμους της να αποτελούν κομμάτι μεγαλύτερων συστημάτων σε κλάδους όπως η βιομηχανική ρομποτική, η αναγνώριση φωνής και ομιλίας, το data mining, οι ιατρικές διαγνώσεις, το λογισμικό τραπεζών, η επιχειρησιακή έρευνα και οι μηχανές αναζήτησης όπως αυτή της Google.

2 *Η νοημοσύνη στη*

Ρομποτική

Η λέξη ρομπότ (και η ρομποτική, ως παράγωγο της πρώτης) προέρχεται από τη λέξη robot που εισήχθη για πρώτη φορά από τον Τσέχο συγγραφέα Karel Capek σε μία παράστασή του με θέμα επστημονικής φαντασίας που δημοσιεύτηκε το 1920. Ρίζα αυτής είναι εν τέλει η σλαβική λέξη robota που σημαίνει “εργασία”. Όταν αναφερόμαστε επομένως σε ρομπότ μιλάμε ουσιαστικά για εργάτες με συγκεκριμένη αποστολή.

Η λέξη “ρομποτική” χρησιμοποιήθηκε για πρώτη φορά από τον Isaac Asimov το 1941, ο οποίος ήταν επίσης συγγραφέας έργων επιστημονικής φαντασίας. Χωρίς να το γνωρίζει ότι ο όρος αυτός δεν είχε χρησιμοποιηθεί πρωτύτερα δημιούργησε τη λέξη για την επιστήμη που ασχολείται με τα ρομπότ, ενώ το 1942 δημιούργησε τους Τρεις Νόμους της Ρομποτικής για την σύντομη ιστορία του με τίτλο “Runaround“.

Οι Τρεις Νόμοι της Ρομποτικής διατυπώνονται ως εξής:

1. Ένα ρομπότ δεν επιτρέπεται να βλάψει έναν ένθρωπο ή μέσω της αδράνειάς του να επιτρέψει να προκληθεί σωματική βλάβη σε έναν άνθρωπο.

2. Ένα ρομπότ πρέπει να υπακούει στις εντολές που του δίνονται από τους ανθρώπους, εκτός και αν οι εντολές αυτές έρχονται σε σύγκρουση με τον πρώτο νόμο.

3. Ένα ρομπότ πρέπει να προστατεύει την ακεραιότητά του, εφόσον αυτό δεν αντίκειται στον πρώτο και τον δεύτερο νόμο.

Είναι εμφανές από τα παραπάνω (αν και ελαφρώς ειρωνικό) το γεγονός ότι η ρομποτική, ένας κλάδος που συνδυάζει έμπρακτα πολλούς επιστημονικούς τομείς – τα μαθηματικά, τη φυσική, την μηχανολογία, την ηλεκτρονική, το λογισμικό κ.ά. – απέκτησε το όνομά του και κάποια από τα χαρακτηριστικά που τον διακρίνουν χάρη σε λογοτέχνες και όχι σε επιστήμονες.

Μπορεί επίσης κανείς να διακρίνει χωρίς δυσκολία ότι η ρομποτική από την αρχή της ήταν άρρηκτα συνδεδεμένη με την έννοια της τεχνητής νοημοσύνης. Ένα ρομπότ που δεν του έχει δοθεί η δυνατότητα να δρα αυτόνομα και χωρίς την παρουσία κάποιου χειριστή που θα καθορίζει τις κινήσεις του, καταφέρνει κατά το ήμισυ μόνο να επιτυγχάνει τον αρχικό σκοπό του, ο οποίος είναι να μη χρειάζεται να συμμετέχει ο άνθρωπος σε μία εργασία τετριμένη για τις δυνατότητές του (πχ επαναλαμβανόμενες κινήσεις στη γραμμή παραγωγής ενός εργοστασίου) ή να μην απαιτείται ανθρώπινη παρουσία σε συνθήκες που εγκυμονούν κινδύνους για την ασφάλεια ή και τη ζωή (πχ εξερεύνηση ενός κατεστραμμένου πυρηνικού αντιδραστήρα για την αξιολόγηση της μόλυνσης και των καταστροφών γενικίοτερα).

Για να μπορέσουμε να επεκταθούμε περισσότερο στις διάφορες μορφές νοημοσύνης που μπορούν να εφαρμοστούν σε ένα ρομπότ, θα πρέπει κατ' αρχήν να αναφέρουμε τις βασικές κατηγορίες ρομπότ και πού αυτές χρησιμοποιούνται.

Η πιο συνηθισμένη ίσως κατηγορία είναι αυτή των ρομποτικών βραχιώνων που χρησιμοποιούνται στη βιομηχανία. Τα ρομπότ αυτά είναι προσαρμοσμένα σε μία βάση και άρα καταλαμβάνουν μια σταθερό θέση στο χώρο χωρίς δυνατότητα μετακίνησης. Επομένως, ο ρόλος τους είναι να εκτελούν μια συγκεκριμένη ή ένα πλήθος συγκεκριμένων εργασιών εντός της ακτίνας στην οποία έχουν πρόσβαση. Βασικό χαρακτηριστικό που τα διακρίνει μεταξύ τους είναι το είδος του εργαλείου που είναι προσαρμοσμένο στο άκρο τους και το οποίο καθορίζει τον τομέα χρήσης τους. Παρότι είναι πολύ χρήσιμα στην καθημερινή βιομηχανική ζωή και έχουν τη δυνατότητα να επαναπρογραμματίζονται για να αλλάζουν χώρο εφαρμογής και να βελτιώνουν την απόδοσή τους, τα ρομπότ αυτά δεν παρουσιάζουν ιδιαίτερο ενδιαφέρον στην παρούσα εργασία.

Μια δεύτερη ενδιαφέρουσα κατηγορία είναι τα εκπαιδευτικά ρομπότ, τα οποία χρησιμοποιούνται κυρίως κατά την σχολική εκπαίδευση και έχουν στόχο να κινήσουν το ενδιαφέρον των μαθητών για την φυσική, τα μαθηματικά, τον προγραμματισμό και την ηλεκτρονική. Σε αντίθεση με τα βιομηχανικά δεν είναι συνήθως σταθερά στον χώρο, αλλά έχουν περιορισμένες τις περισσότερες φορές δυνατότητες, οι οποίες όμως επαρκούν για τις εκπαιδευτικές ανάγκες. Καθώς το ερευνητικό ενδιαφέρον εδώ επικεντρώνεται κυρίως στο να παρέχει στους χρήστες-μαθητές την ευελιξία να τα χρησιμοποιούν για ποικίλες εφαρμογές, ενώ παράλληλα το κόστος θα είναι χαμηλό σχετικά για να είναι προσιτό, προκειται για ακόμη μία κατηγορία που δεν θα μας απασχολήσει εδώ.

Η τρίτη βασική κατηγορία, και αυτή με την οποία θα ασχοληθούμε στο υπόλοιπο της παρούσας εργασίας είναι τα κινητά ρομπότ (mobile robots). Τα συγκεκριμένα έχουν τη δυνατότητα να αλληλεπιδρούν με το περιβάλλον τους και να το αντιλαμβάνονται με τέτοιο τρόπο ώστε να πλοηγούνται μέσα σε αυτό επιτυγχάνοντας έτσι και την αποστολή τους. Τα ρομπότ αυτά χρησιμοποιούνται κυρίως σε βιομηχανικές εφαρμογές, σε στρατιωτικά προγράμματα και σε περιβάλλοντα όπου τίθενται θέματα ασφαλείας, όπως πχ στην συντήρηση ενός πυρηνικού αντιδραστήρα. Σχετικά συνηθισμένη επίσης εφαρμογή τους ώς καταναλωτικό αγαθό είναι τα ρομποτάκια ηλεκτρικές σκούπες. Πέραν αυτού, η συνηθέστερη χρήση τους είναι σε ελεγχόμενα περιβάλλοντα, όπως οι γραμμές παραγωγής, γιατί υπάρχει μεγάλη δυσκολία να ανταποκριθούν σε μη προβλεπόμενες καταστάσεις και παρεμβάσεις, και αυτός είναι ο λόγος που οι περισσότεροι άνθρωποι δεν τα συναντούν στην καθημερινότητά τους.

Όπως ήδη αναφέρθηκε ο προγραμματισμός τους αποτελεί πρόκληση, γιατί όταν κάτι κινείται αυτόνομα στον χώρο οι καταστάσεις στις οποίες μπορεί να βρεθεί είναι πολλές και αρκετά συχνά απρόβλεπτες. Εμπόδια, άνθρωποι που λειτουργούν ταυτόχρονα στο ίδιο ή σε συγγενικό αντικείμενο και πρέπει να αλληλεπιδράσουν με το ρομπότ, καθώς και άλλα μηχανήματα που ενεργούν παράλληλα και επηρεάζουν την αλληλουχία των ενεργειών που πρέπει να ακολουθηθούν, πρέπει να ληφθούν υπ' όψιν και να συνυπολογίζονται ανά πάσα στιγμή προκειμένου οι “αποφάσεις” που λαμβάνονται για περαιτέρω ενέργειες να μην διακινδυνεύουν τον τελικό στόχο και να μην παρεμποδίζουν την λειτουργικότητα των ανθρώπων και των υπόλοιπων μηχανημάτων.

Για να συμβεί αυτό θα πρέπει πρώτα να επιλεγεί ανάλογα με τη χρήση το κατάλληλο σύστημα αναγνώρισης του περιβάλλοντος (στερεοσκοπική όραση, αισθητήρες υπερύθρων, laser, υπέρηχοι κτλ) και στη συνέχεια να γίνεται μια πρώτη επεξεργασία των τιμών που λαμβάνονται από το σύστημα αυτό ώστε να αποθηκεύονται ατα δεδομένα σε κατάλληλη μορφή για περαιτέρω ανάλυση. (Αυτό δε θα μας απασχολήσει εδώ, καθώς θεωρούμε ότι έχουμε τις τιμές και γνωρίζουμε τι αντιπροσωπεύουν αυτές – πχ εμπόδια ή την τιμή της συνάρτησης κόστους σε ένα συγκεκριμένο σημείο, οπότε επικεντρωνόμαστε στο κομμάτι της πλοήγησης.)

Μια διαφορετική προσέγγιση στην αναγνώριση του περιβάλλοντος, είναι η προσθήκη στο ρομπότ της δυνατότητας αναγνώρισης φωνής. Καθώς όμως αυτή η τεχνολογία είναι ακόμα σε σχετικά πρώιμο στάδιο και δεν είναι επαρκώς αξιόπιστη για εφαρμογές που απαιτούν ακρίβεια, δε θα επεκταθούμε περαιτέρω σε αυτό.

Στη συνέχεια, θα πρέπει να συγκεκριμενοποιηθεί και να αναλυθεί ο βασικός στόχος της εφαρμογής μας και στη συνέχεια να εξεταστεί το περιβάλλον στο οποίο θα λειτουργεί (ή αν αυτό δεν είναι σταθερό, οι ακραίες περιβαλλοντικές συνθήκες που μπορεί να επικρατήσουν, ώστε να μην προκύψουν απρόοπτα). Ο αλγόριθμος που θα επιλέξουμε για τον βασικό στόχο έχει άμεση σχέση τις περισσότερες φορές με το περιβάλλον και τις προκλήσεις που αυτό παρουσιάζει για την πλοήγηση, κάνοντας αυτά τα δύο στοιχεία πολύ στενά συνδεδεμένα από την αρχή κάθε τέτοιου project.

Βασικό επίσης χαρακτηριστικό μιας εφαρμογής με κινητά ρομπότ είναι το αν θα χρησιμοποιηθεί ένα μόνο “δυνατό” ρομπότ (με αυξημένες δυνατότητες, όπως υπολογιστική ισχύς, ιδιιαίτερα ακριβείς αισθητήρες, και φτιαγμένο έτσι ώστε να μην τίθεται εύκολα εκτός λειτουργίας – λόγω ατυχημάτων κτλ – για να έχει αυξημένες πιθανότητες να φέρει εις πέρας την αποστολή που του ανατέθηκε παρά τα απρόοπτα) ή μία ομάδα από πιο “απλά”, τα οποία θα συνεργαστούν για να έχουν ένα ικανοποιητικό αποτέλεσμα.

Στα επόμενα κεφάλαια θα αναπτυχθούν και οι δύο προσεγγίσεις, προκειμένου να διαπιστωθούν τα οφέλη της καθεμιάς στο πρόβλημα της πλοήγησης σε άγνωστο χώρο.

3 *Αλγόριθμοι ενός πράκτορα (Single Agent Algorithms) και πολλαπλών πρακτόρων (Multiple Agent Algorithms)*

Ένα σύστημα ενός μόνο πράκτορα μπορεί να περιλαμβάνει πολλές οντότητες – πολλά τελικά εργαλεία ή και πολλά ρομπότ. Ωστόσο αν κάθε μία από αυτές τις οντότητες στέλνει τα ερεθίσματα που δέχεται και λαμβάνει οδηγίες για το πώς να δράσει από μία κεντρική οντότητα , τότε υπάρχει στην ουσία ένας μόνο πράκτορας, η κεντρική διεργασία.

Παρότι μπορεί κανείς να σκεφτεί ότι ένα σύστημα ενός πράκτορα θα έπρεπε να είναι πιο απλό από ένα σύστημα πολλών πρακτόρων, στην πραγματικότητα όταν καλούμαστε να αντιμετωπίσουμε ένα σταθερό και σύνθετο έργο, τότε το αντίθετο ισχύει συνήθως. Η κατανομή ελέγχου σε πολλούς πράκτορες επιτρέπει στον κάθε ένα να είναι απλούστερος, καθώς κανένας πράκτορας δεν χρειάζεται να επιλύσει το πρόβλημα μόνος του.

Ειδικά κατά την πλοήγηση μια ομάδα από ρομπότ έχει πλεονέκτημα έναντι ενός που λειτουργεί μόνο του, γιατί μπορούν να εκμεταλλευτούν τη γεωμετρική κατανομή. Τη στιγμή που ένα ρομπότ μόνο του μπορεί να αντιληφθεί τον κόσμο από ένα μόνο σημείο, ένα

σύστημα πολλών πρακτόρων μπορεί να τον παρακολουθεί από πολλές τοποθεσίες ταυτόχρονα.

Ακόμα και αφότου όμως επιλέξουμε αν το σύστημά μας θα είναι ενός ή περισσότερων πρακτόρων, εξακολουθούμε να έχουμε αρκετή ελευθερία να διαμορφώνουμε το σύστημα όπως μας εξυπηρετεί καλύτερα, καθώς μπορούμε να επιλέξουμε από μία μεγάλη σχετικά ποικιλία αλγορίθμων που καθορίζουν την πορεία του συστήματος στο περιβάλλον.

Έχουμε έτσι τους αιτιοκρατικούς (ντετερμινιστικούς) αλγορίθμους, οι οποίοι με ίδιες αρχικές συνθήκες όσες φορές και αν εκτελεστούν θα δώσουν τα ίδια αποτελέσματα, και τους ευφυείς αλγορίθμους οι οποίοι, καθώς εκτελούνται αποκτούν έναν βαθμό γνώσης για το σύστημα και λειτουργούν έτσι αποδοτικότερα. Ένας τέτοιος ευφυής αλγόριθμος μπορεί για παράδειγμα να χρησιμοποιεί τεχνικές από reinforcement learning ή από εξελικτικούς αλγορίθμους. Μεγαλύτερο ενδιαφέρον όμως, όπως σχεδόν πάντα άλλωστε έχει η προσέγγιση που ακολουθείται από τη φύση, η οποία με την πάροδο των χρόνων έχει κατορθώσει στις περισσότερες περιπτώσεις να βελτιστοποιήσει την απόδοση ελαχιστοποιώντας παράλληλα το κόστος. Για λόγους καθαρότητας της παρουσίασης, οι εμπνευσμένοι από τη φύση αλγόριθμοι αναπτύσσονται στο επόμενο κεφάλαιο.

4 Ευφυείς αλγόριθμοι εμπνευσμένοι από τη φύση

Το σημείο στο οποίο διαφέρουν οι εμπνευσμένοι από τη φύση αλγόριθμοι σε σχέση με την παραδοσιακή τεχνητή νοημοσύνη είναι στο ότι ακολουθούν μία πιο εξελικτική προσέγγιση ως προς τη μάθηση. Στην παραδοσιακή τεχνητή νοημοσύνη, η ευφυία ακολουθεί μια πορεία από πάνω προς τα κάτω (top-down method): ο προγραμματιστής είναι ο δημιουργός και εμποτίζει με τη δική του νοημοσύνη τα προγράμματα. Αντιθέτως, ο προγραμματισμός που βασίζεται σε βιολογικές λειτουργίες χρησιμοιποιεί μια περισσότερο από κάτω προς τα πάνω προσέγγιση (bottom-up approach). Οι τεχνικές αυτές συνήθως επικεντρώνονται στη διατύπωση τριών βασικών πραγμάτων: ενός συνόλου απλών κανόνων, ενός συνόλου απλών οργανισμών που θα υπακούουν τους κανόνες αυτούς και μίας μεθόδου η οποία επαναλαμβανόμενα θα εφαρμόζει τους κανόνες αυτούς.

Είναι σημαντικό στο σημείο αυτό, πριν επεισέλθουμε σε περισσότερες λεπτομέρειες για τους αλγορίθμους αυτούς και τις εφαρμογές τους, να δούμε τον λόγο που αναπτύχθηκε εξαρχής ενδιαφέρον προς την κατεύθυνση αυτή.

4.1 Η νοημοσύνη της φύσης

Παρατηρώντας κανείς τον τρόπο με τον οποίο λειτουργούν όλοι οι ζωντανοί οργανισμοί, εντοπίζει κάποια αξιοσημείωτα κοινά χαρακτηριστικά. Το σημαντικότερο ίσως είναι η προσαρμοστικότητα που επιδεικνύουν όταν το περιβάλλον εντός του οποίου βρίσκονται

μεταβάλλεται. Οι νόμοι της εξέλιξης είναι τέτοιοι που επιβάλλουν σε όλα τα πλάσματα προσαρμογή ή εξαφάνιση. Προκειμένου επομένως να διασφαλιστεί η επιβίωση, όλοι οι οργανισμοί, από τους πιο μικρούς ως τους πιο μεγάλους και σύνθετους, παρακολουθούν το περιβάλλον τους και ανιχνεύουν τυχόν διαταραχές, ενώ στη συνέχεια τις λαμβάνουν υπ' όψιν τους στη λήψη αποφάσεων, όπως είναι για παράδειγμα η επιλογή του χρόνου αναπαραγωγής, η αναζήτηση τροφής σε άλλο μέρος ή η αν έχει φτάσει η εποχή για να περιέλθουν σε χειμερία νάρκη, και να προστατευτούν έτσι απο την έλλειψη τροφής και το κρύο. Καθώς εγγενής αποστολή όλων των ζώων, φυτών και μικροοργανισμών είναι η αυτοσυντήρηση, η διαιώνιση του είδους και η διατήρηση της συνοχής της ομάδας αν πρόκειται για κοινωνικά όντα, η συνεχής επανεκτίμηση των περιβαλλοντικών συνθηκών είναι πρωταρχικό μέλημα στις φυσικές διαδικασίες.

Καθοριστικής σημασίας είναι επίσης η εμφανής στις περισσότερες περιπτώσεις ικανότητα των οργανισμών να επιτύγχάνουν με το ελάχιστο δυνατό κόστος το επιθυμητό αποτέλεσμα. Ένα ενδεικτικό παράδειγμα τέτοιας συμπεριφοράς βλέπουμε στα μυρμήγκια και τον τρόπο με τον οποίο εξερευνούν μία περιοχή για τροφή, ενημερώνοντας παράλληλα με μονοπάτια από φερομόνες τα υπόλοιπα μυρμήγκια σχετικά με τις τοποθεσίες όπου βρίσκεται απόθεμα. Έτσι συνολικά σπαταλάται πολύ λιγότερη ενέργεια και φυσικά λιγότερος χρόνος για να συλλεχθεί η τροφή, ενώ παράλληλα καλύπτεται μεγαλύτερη έκταση.

Στο παράδειγμα των μυρμηγκιών βλέπουμε επίσης την σημασία που έχει η ομαδικότητα έναντι της ατομικότητας. Ο χρόνος που απαιτείται για την επίτευξη του στόχου μειώνεται δραστικά και βελτιστοποιείται η σχέση ενέργειας-αποτελέσματος. Γίνεται έτσι εμφανής ο λόγος που η επιστήμη έχει στραφεί στην μελέτη των φυσικών διεργασιών προκειμένου να αντλήσει έμπνευση για την επίλυση προβλημάτων, και ειδικότερα στη μελέτη της λειτουργίας κοινωνικών δομών όπως για παράδειγμα των μυρμηγκιών, των μελισσών, των τερμιτών κ.ά.

Εφαρμόζοντας επομένως τις βασικές αρχές λειτουργίας που ακολουθούν οι κοινωνικές ομάδες των εντόμων και των ζώων στην φύση μπορούμε να διδαχθούμε πολλά για τα συστήματα που κατασκευάζουμε και τις μεθοδολογίες που μπορούμε να ακολουθήσουμε ώστε να φτάνουμε σε αποδοτικότερες λύσεις με μικρότερο κόστος. Θα πρέπει να αξιοποιήσουμε αυτή τη γνώση που είναι κρυμμένη σε κοινή θέα, και προέρχεται από χιλιετίες εξελικτικής πορείας – τα μεγαλύτερα ανθρώπινα επιτεύγματα άλλωστε έχουν ως

πηγή έμπνευσης κάποιον ζώντα οργανισμό ή μι (πχ τα αεροπλάνα δημιουργήθηκαν μετά από προσεκτική παρατήρηση των πτηνών

5 *Χρήση του αλγορίθμου PSO (Particle Swarm Optimization) για εξερεύνηση με πολλούς πράκτορες*

5.1 Περιγραφή του αλγορίθμου

Βάση του αλγορίθμου RDPSO είναι ο PSO, γι' αυτό θα ξεκινήσουμε με μια εισαγωγή σε αυτόν.

Στον αλγόριθμο PSO (Particle Swarm Optimization) θεωρούμε ότι έχουμε N particles – στο εξής όταν λέμε particles εννοούμε τα μικρά (σχετικά) αυτόνομα ρομπότ, που αποτελούν τις μικρότερες οντότητες στο σύστημά μας με τη δυνατότητα να δρουν ανεξάρτητα. Τα particles αυτά κινούνται στον χώρο με σκοπό να εντοπίσουν το ολικό βέλτιστο, το οποίο εδώ είναι το ολικό ελάχιστο, μιας συνάρτησης κόστους, που θεωρούμε ότι έχει να κάνει με μια μετρήσιμη ποσότητα που τα ρομποτάκια αντιλαμβάνονται μέσω των αισθητήρων τους. Σε κάθε χρονική στιγμή, τα particles χαρακτηρίζονται από τη θέση τους στον χώρο και την τιμή που λαμβάνει η συνάρτηση κόστους στη θέση αυτή. Για τον υπολογισμό της επόμενης θέσης του κάθε particle χρησιμοποιούμε τις σχέσεις

$$v_n[t+1] = w \cdot v_n[t] + c_1 \cdot r_1 \cdot (\breve{g}_n[t] - x_n[t]) + c_2 \cdot r_2 \cdot (\breve{x}_n[t] - x_n[t]) \quad (1)$$

$$x_n[t+1]=x[n]+v_n[t+1] \quad (2)$$

όπου:

$v_n[t]$: η ταχύτητα του particle n τη χρονική στιγμή t

$\breve{g}_n[t]$: η βέλτιστη θέση που έχει παρατηρηθεί από όλα τα particles συνολικά από τη χρονική στιγμή 0 μέχρι την t

$\breve{x}_n[t]$: η βέλτιστη θέση στο περιβάλλον του particle n κατά τη χρονική στιγμή t όπως το αντιλαμβάνεται μέσα από τους αισθητήρες που διαθέτει.

w : το βάρος που δίνουμε στην προηγούμενη ταχύτητα για τον υπολογισμό της νέας

c_1, c_2 : τα βάρη που δίνουμε στο ολικό και το τοπικό μέγιστο ώστε να επηρεάζουν και να διαμορφώνουν την νέα ταχύτητα

r_1, r_2 : τυχαίοι συντελεστές που λαμβάνουν τιμή μεταξύ 0 και 1

Μια επέκταση του PSO αλγορίθμου είναι ο RPSO (Robotic Particle Swarm Optimization) και η βασική διαφορά τους είναι ότι ο τελευταίος υπεισέρχεται σε λεπτομέρειες της υλοποίησης, ώστε να λαμβάνονται υπόψιν πραγματικά σενάρια και προβλήματα, όπως είναι η αποφυγή εμποδίων για παράδειγμα. Έτσι η συνάρτηση κόστους που αναφέρθηκε και πιο πριν τροποποιείται ώστε να καθοδηγεί το κάθε ένα από τα ρομπότ στον στόχο τους ενώ παράλληλα θα αποφεύγει και τα εμπόδια, τόσο τα στατικά όσο και τα δυναμικά. Αυτό φυσικά προϋποθέτει την τοποθέτηση αισθητήρων ικανών για τον εντοπισμό τους εντός μίας πεπερασμένης ακτίνας r_s . Πρέπει επιπλέον να οριστεί μια μονότονη, θετική συνάρτηση $g(x_n[t])$ (sensing function), η τιμή της οποίας να εξαρτάται από τα δεδομένα που λαβάνει από τους αισθητήρες, την απόσταση δηλαδή από το εμπόδιο. (Η συνάρτηση αυτή μπορεί να είναι είτε αύξουσα είτε φθίνουσα, αναλόγως από το είδος των αισθητήρων που επιλέγονται, και φυσικά δε μας επηρεάζει σχεδόν καθόλου στην υλοποίηση του συστήματος.)

Έτσι η νέα ταχύτητα με τις προσθήκες αυτές θα δίνεται από τη σχέση:

$$v_n[t+1]=w\cdot v_n[t]+c_1\cdot r_1\cdot(\breve{g}_n[t]-x_n[t])+c_2\cdot r_2\cdot(\breve{x}_n[t]-x_n[t])+c_3\cdot r_3\cdot(\breve{x}_n^g[t]-x_n[t])$$

(3)

όπου

c_3 : το βάρος που επιλέγουμε να έχει ο παράγοντας που ρυθμίζει την αποφυγή εμποδίων

r_3 : ο τυχαίος παράγοντας με τιμή μεταξύ 0 και 1

$\breve{x}_n^g[t]$: μία θέση του ρομπότ n στην οποία βελτιστοποιείται η συνάρτηση κόστους $g(x_n[t])$ (εντός μίας συγκεκριμένης ακτίνας, γιατί η εμβέλεια των αισθητήρων είναι πεπερασμένη)

Η επιλογή των βαρών c_1, c_2, c_3 έχει να κάνει με την προτεραιότητα που θα δίνει το κάθε particle στη βασική του αποστολη (εύρεση ολικού βελτίστου) έναντι της αποφυγής των εμποδίων που συναντάει. Έτσι, επιλογή μιας τιμής του c_3 κατά πολύ μικρότερης της ελάχιστης τιμής των c_1, c_2 θα μπορούσε να οδηγήσει σε ταχύτερη σύγκλιση στην επιθυμητή λύση, κάνοντας παράλληλα το ρομπότ πιο επιρρεπές σε συγκρούσεις. Αντίθετα, αν το c_3 υπερβαίνει κατά πολύ τις τιμές των c_1, c_2 , μπορεί να αυξηθεί η επιτυχής αποφυγή των εμποδίων αλλά η σύγκλιση να καθυστερήσει σημαντικά.

Επέκταση του RPSO είναι ο RDPSO (Robotic Darwinian Particle Swarm Optimization) χαρακτηριστικό του οποίου είναι πως τα particles δεν αποτελούν ένα ενιαίο σύνολο, αλλά χωρίζονται σε swarms καθένα από τα οποία έχει μια σχετική αυτονομία. Βασικό επίσης χαρακτηριστικό της επέκτασης αυτής είναι η ύπαρξη μεθόδου “κοινωνικού αποκλεισμού” κάποιου particle αν αυτό δεν είναι αρκετά αποδοτικό. Περισσότερες λεπτομέρειες επ' αυτού μπορεί να βρει κανείς στη επόμενη παράγραφο, με την περιγραφή της υλοποίησης, ώστε να γίνει πλήρως κατανοητός ο μηχανισμός που χρησιμοποιήθηκε.

5.2 Λεπτομέρειες της υλοποίησης

Οι συναρτήσεις και τα scripts που δημιούργησα στο περιβάλλον του Matlab για να υλοποιήσω τον αλγόριθμο αυτό είναι οι ακόλουθες (σε αλφαβητική σειρά):

1. add_particle_from_excluded
2. calc_opt_pos_sensing_function

3. calc_opt_pos_sensing_function_excl

4. create_space

5. create_swarms

6. delete_swarm

7. evolve_excluded

8. evolve_swarm

9. exclude_particle

10. main

11. main_rdpso

12. rand_num

13. show_visual_result

14. spawn_swarm

15. update_local_optimum

16. update_observations

17. update_velocity

Λειτουργία των παραπάνω συναρτήσεων:

1. add_particle_from_excluded:

Ένα συγκεκριμένο particle από το excluded swarm, το best_particle (το οποίο υπολογίστηκε πριν να κληθεί το script αυτό), προστίθεται στο swarm s (όπου και πάλι το s υπολογίστηκε πριν την κλήση του script αυτού) και στη συνέχεια αφαιρείται από το excluded swarm. Προφανώς όταν μιλάμε για προσθήκη και διαγραφή εννοούμε όλα τα χαρακτηριστικά που περιγράφουν ένα particle και όχι μόνο τη θέση του.

2. calc_opt_pos_sensing_function:

Πρόκειται για συνάρτηση με τα ακόλουθα ορίσματα:

α. τη δομή swarms που περιέχει όλα τα χαρακτηριστικά των particles που ανήκουν στα ενεργά swarms

β. τον πίνακα obstacles με τις συντεταγμένες των σημείων στα οποία βρίσκονται τα εμπόδια

γ,δ. τα s, n αντιστοιχούν στο n-οστό particle του swarm s το οποίο και θέλουμε να εξετάσουμε

ε. rs είναι η ακτίνα "αντίληψης" των αισθητήρων για την ανίχνευση εμποδίων, η ακτίνα δηλαδή εντός της οποίας αν βρεθεί ένα εμπόδιο τότε αυτό θα γίνει αντιληπτό από το συγκεκριμένο ρομπότ

Οι αισθητήρες θεωρούμε ότι λειτουργούν με βάση μια φθίνουσα sensing function η οποία εδώ για λόγους απλότητας (χωρίς όμως και να προκαλείται κάποια αλλοίωση των αποτελεσμάτων) θεωρούμε ότι είναι η 1/x. Έτσι το αποτέλεσμα της συνάρτησης αυτής είναι το g_min_pos, η θέση δηλαδή στο χώρο (εντός όμως μια ακτίνας rc) στην οποία η sensing function ελαχιστοποιείται.

Για να το επιτύχουμε αυτό κρατάμε σε έναν πίνακα visible_obst τα εμπόδια και τα particles που βρίσκονται εντός της rc ακτίνας και άρα γίνονται αντιληπτά από το υπό εξέταση particle. Μετά φτιάχνουμε για κάθε σημείο εντός της rc έναν πίνακα g στον οποίο αποθηκεύεται το άθροισμα των αντιστρόφων αποστάσεων του σημείου αυτου από κάθε ορατό εμπόδιο. Ως g_min_pos θέτουμε τελικά το σημείο εκείνο στο οποίο ο πίνακας g έχει ελάχιστη τιμή.

3. calc_opt_pos_sensing_function_excl

Η συνάρτηση αυτή είναι εντελώς όμοια με την προηγούμενη, με μόνη διαφορά το ότι καθώς αναφέρεται στο excluded swarm δεν παίρνει ως όρισμα τιμή s, και άρα το υπό εξέταση particle είναι το n-οστό του excluded swarm.

4. create_space

Πρόκειται για script και όχι για συνάρτηση. Σημαντικές τιμές που χρησιμοποιούνται σε αυτή και δίνονται από την χρήστη ως σταθερές κατά κάποιον τρόπο είναι το num_Gaussians και το L, τα οποία δίνουν τον αριθμό των Gaussians που θα αθροίσουμε για να προκύψει η συνάρτηση κόστους του χώρου και η διάσταση του χώρου (με εμβαδό LxL) αντίστοιχα.

Στο πρώτο for-loop δημιουργούμε τις Gaussians με κέντρο m (ένα τυχαίο σημείο του χώρου) και πίνακα Sigma κοινό για όλες τις Γκαουσιανές που εξαρτάται κάθε φορά από το L. Η συνάρτηση κόστους που προκείπτει από το άθροισμα αυτό, αποθηκεύεται στον πίνακα cost_func και κατόπιν αντιστρέφεται ώστε να παρουσιάζει μοναδικό τοπικό ελάχιστο (αντί για μέγιστο που ήταν αρχικά). Ακολουθεί μία αναπαράσταση της συνάρτησης κόστους σε γράφημα και μετά δημιουργούμε τα εμπόδια με τυχαίο τρόπο (ο αριθμός No των εμποδίων δίνεται στην main.m, η οποία καλεί την create_space). Τέλος, δημιουργούμε και ένα plot για να δούμε την κατανομή των εμποδίων στον χώρο.

5. create_swarms

Το αρχείο “create_swarms.m” περιέχει ένα script το οποίο δημιουργεί όλες τις απαραίτητες αρχικοποιήσεις για τα particles του κάθε swarm ώστε να ξεκινήσει η εκτέλεση του κυρίως αλγορίθμου. Οι αρχικοποιήσεις αυτές συνίστανται στα παρακάτω:

- δημιουργούμε τη δομή cell μεγέθους Si x 1 που θα περιέχει όλα τα απαραίτητα στοιχεία

- swarms{s}.positions: αρχικοποιούμε τις θέσεις των particles με τυχαίες τιμές (έτσι κάθε φορά που θα εκτελείται ο αλγόριθμος οι αρχικές συνθήκες θα είναι τυχαίες και τα αποτελέσματα πιο αξιόπιστα)

- swarms{s}.velocity: οι αρχικές ταχύτητες είναι τυχαίας κατεύθυνσης και μέτρου (μέγιστο μέτρο όμως είναι η μονάδα)

- swarms{s}.x_opt: θέτουμε ως θέσεις βέλτιστης τιμής της συνάρτησης κόστους κατά την αρχική αυτή χρονική στιγμή τις θέσεις των particles.

- swarms{s}.xg_opt: στη συνέχεια υπολογίζεται με τη βοήθεια της calc_opt_pos_sensing_func το σημείο του γειτονικού χώρου του κάθε particle όπου η sensing function μηδενίζετα

- gn είναι η θέση γενικού βελτίστου και gn_cost η τιμή του βελτίστου αυτού (όπως υπολογίζονται σε κάθε κύκλο του αλγορίθμου – όχι το πραγματικό ολικό βέλτιστο του χώρου) – στην τιμή αρχικοποίησής τους καταλήγουμε αφού βρούμε την τιμή της συνάρτησης κόστους για την παρούσα θέση του κάθε particle και εξετάσουμε ποια από αυτές είναι η βέλτιστη

- οι μεταβλητές condition_1, condition_2 περιέχουν αντίστοιχα την προηγούμενη τιμή του ολικού βελτίστου και την τρέχουσα. Αφαιρώντας τις δύο λαμβάνουμε την αναγκαία

συνθήκη τερματισμού του loop στο script main_rdpso, και τις αρχικοποιούμε με τον πλησιέστερο αριθμό στο μηδέν που μπορεί να αναγνωρίσει το περιβάλλον του Matlab και την τρέχουσα τιμή γενικού βελτίστου (αντίστοιχα).

-το swarm των excluded particles δεν θα περιέχει αρχικά κάποιο particle, οπότε δημιουργούμε τη δομή του και το αφήνουμε κενό

- ο πίνακας cost_obst_fig περιέχει τις τιμές της συνάρτησης κόστους στα σημεία όπου βρίσκονται τα εμπόδια – αυτό γίνεται μόνο για να φαίνεται οπτικά ομοιόμορφο το αποτέλεσμα στα plots όταν απεικονίζονται τα εμπόδια

6. delete_swarm

Όταν, υπό κατάλληλες συνθήκες καλείται αυτό το script, τότε το s-οστό swarm διαγράφεται και τα particles που το απαρτίζουν μεταφέρονται στο excluded swarm, μαζί με όλα τα χαρακτηριστικά τους.

7. evolve_excluded

Πρόκειται για συνάρτηση με μόνο όρισμα το excluded και επιστρεφόμενη τιμή πάλι το excluded, αυτή τη φορά μετά τις αλλαγές που έχει υποστεί εντός της συνάρτησης.

Η μέθοδος που ακολουθούμε για να “εξελίξουμε” σε κάθε κύκλο του αλγορίθμου το swarm των excluded είναι σαφώς απλούστερη αυτής που ακολουθούμε για τα υπόλοιπα swarms (το γιατί γίνεται απόλυτα σαφές αμέσως μόλις δει κανείς την υλοποίηση του evolve_swarm.m). Δίνουμε στα particles τυχαία, μοναδιαία ταχύτητα. Έπειτα, βρίσκουμε την καινούργια θέση με βάση την προηγούμενη και την νέα (τυχαία) ταχύτητα, πολλαπλασιασμένη όμως επί 4 (προσοχή: στο evolve_swarm στο αντίστοιχο σημείο η ταχύτητα πολλαπλασιάζεται επί 2 – ο λόγος είναι ότι τα particles στο swarm των excluded πρέπει να είναι αισθητό ότι κινούνται πιο απρόβλεπτα και ταχύτερα, προκειμένου να σημειωθεί κάποια βελτίωση αν έχει κολλήσει ο αλγόριθμος σε κάποιο τοπικό μέγιστο. Αυτό ακολουθεί λογική που οι άνθρωποι σε κοινωνικά αποκλεισμένες ομάδες “κινούνται” σχετικά ανεξέλεγκτα και παρουσιάζουν σε κάποιες περιπτώσεις εντυπωσιακά αποτελέσματα σε κάποιον τομέα, πχ στην τέχνη. Ελέγχουμε επίσης οι συντεταγμένες της νέας θέσης να μην ξεφεύγουν από τα όρια του χώρου με τέσσερα if. Τέλος υπολογίζουμε

την τιμή της συνάρτησης κόστους για τις νέες θέσεις, και δίνουμε μια τιμή στο improvement του κάθε particle (το οποίο πρακτικά δεν έχει κάποια χρησιμότητα, παρά μόνο αν το particle το συγκεκριμένο μεταφερθεί σε κάποιο άλλο swarm).

8. evolve_swarm

Πρόκειται για συνάρτηση με τα ακόλουθα ορίσματα:

-swarms: η δομή που περιέχει όλα τα particles που ανήκουν στα "ενεργά" swarms και τα χαρακτηριστικά τους

- s: το swarm που εξετάζουμε

- gn: τη θέση του γενικού βελτίστου που έχουν εντοπίσει τα particles μέχρι στιγμής κατά την εκτέλεση του κώδικα

- SC: ο δείκτης στασιμότητας (stagnancy index) – ένας δείκτης που εξετάζει αν έχει ξεπεραστεί το κατώφλι στασιμότητας (stagnancy threshold), η τιμή δηλαδή που καθορίζει πόσοι κύκλοι επιτρέπεται να περάσουν κατά τους οποίους το συγκεκριμένο swarm δεν κατέληξε σε κάποια βελτίωση. Όταν το κατώφλι αυτό ξεπεραστεί τότε θα επέλθει κάποια προκαθορισμένη ποινή στο swarm ή στο particle αναλόγως, όπως θα φανεί και στη συνέχεια.

-N_kill: είναι ένας πίνακας-στήλη με τόσες θέσεις όσα είναι και τα swarms και σε κάθε θέση περιέχει πόσα particles από το αντίστοιχο swarm έχουν αποκλειστεί (και έχουν μεταφερθεί στο excluded δηλαδή)

Οι τιμές που επιστρέφονται είναι οι εξής:

- swarms: η ανανεωμένη δομή που περιέχει τα χαρακτηριστικά για τα particles που συναποτελούν το σύστημά μας

- SC: η νέα τιμή του δείκτη στασιμότητας, η οποία ενδέχεται να είχε μεταβληθεί κατά την εκτέλεση της συνάρτησης.

- N_kill: η νέα τιμή για τον πίνακα που προαναφέρθηκε στα ορίσματα

- next_swarm: ο μόνος λόγος ύπαρξης αυτής της τιμής ως έξοδος της συνάρτησης αυτής είναι η περίπτωση κατά την οποία έχουμε διαγραφή ενός swarm – τότε θα πρέπει να επιστραφεί η τιμή του επόμενου swarm που θα υποστεί επεξεργασία (διαφορετικά προκύπτουν ποικίλα σφάλματα λόγω της μη αναμενόμενης διάστασης του cell-structure swarm.

Ακολουθεί η περιγραφή της λειτουργίας της συνάρτησης:

Χρησιμοποιώντας τη συνάρτηση update_velocity, της οποίας η λειτουργία θα περιγραφεί παρακάτω, βρίσκουμε τη νέα ταχύτητα των particles και την εισάγουμε στο αντίστοιχο πεδίο του swarms{s}. Βρίσκουμε την νέα θέση του κάθε particle – η σχέση που χρησιμοιποιείται είναι ανάλογη με αυτή του evolve_excluded, με τη διαφορά όμως που επισημάνθηκε ως προς τον συντελεστή που επιλέξαμε για να πολλαπλασιαστεί η ταχύτητα, ο οποίος είναι 2 εδώ αντί για 4 στα excluded. Ελέγχουμε και εδώ με τέσσερα if, ώστε οι νέες συντεταγμένες να βρίσκονται εντός του χώρου.

Βασικό μέγεθος της συνάρτησης αυτής είναι το improvement, το οποίο υποδηλώνει τη συνολική βελτίωση του swarm αυτού και το κατά πόσο δηλαδή πλησιάζει στον επιθυμητό στόχο, και προκύπτει ως το άθροισμα του πεδίου improvement για κάθε ένα particle του συγκεκριμένου swarm.

Αν το improvement είναι θετικό, τότε ελέγχουμε αν υπάρχουν excluded particles και αν το πλήθος των particles στο swarm αυτό δεν ξεπερνά το μέγιστο. Αν αυτά ισχύουν τότε προσθέτουμε στο τρέχον swarm το particle εκείνο από τα excluded που έχει την καλύτερη επίδοση (best performing).

Αν το improvement είναι αρνητικό τότε αυξάνουμε τον δείκτη στασιμότητας, SC, κατά 1. Αν η τιμή του τότε ξεπερνάει το κατώφλι στασιμότητας SCmax, θα εξετάσουμε το πλήθος των particles στο swarm αυτό. Αν υπερβαίνει την ελάχιστη επιτρεπτή τιμή τότε αποκλείουμε από αυτό το particle με τη χειρότερη επίδοση. Διαφορετικά, υπάρχει μια μικρή πιθανότητα (η οποία είναι ανάλογη του πλήθους των swarms που υπάρχουν προς το μέγιστο επιτρεπτό πλήθος, αλλά είναι και ως ένα βαθμό τυχαία) το παρόν swarm να διαγραφεί και τα particles που το απαρτίζουν να μεταφερθούν στο excluded swarm.

Τέλος, αν κανένα particle δεν έχει διαγραφεί από το swarm αυτό, το συνολικό πλήθος των swarms δεν υπερβαίνει το μέγιστο και το excluded swarm περιέχει τουλάχιστον Ni particles (το ελάχιστο δηλαδή πλήθος ανά swarm) υπάρχει μια (μικρή) πιθανότητα να δημιουργηθεί καινούργιο swarm.

9. exclude_particle

Κατ' αναλογία με την περίπτωση του script “add_particle_from_excluded“, μεταφέρουμε ένα particle από το τρέχον swarm (αυτό με τη χειρότερη επίδοση για την ακρίβεια) στο swarm των socially excluded.

10. main

Πρόκειται για το βασικότερο script όλου του κώδικα. Αυτό είναι που καλούμε και αυτόματα γίνονται τα υπόλοιπα. Επίσης εδώ είναι οι περισσότερες παράμετροι που επηρεάζουν τον αλγόριθμο και στις οποίες θέλουμε να επεμβαίνουμε για να εξετάζουμε την απόδοση υπό διαφορετικές συνθήκες. Οι λειτουργίες των παραμέτρων αυτών θα επεξηγηθούν στη συνέχεια (παρότι μια σύντομη περογραφή εμφανίζεται και στις συναρτήσεις/scripts όπου χρησιμοποιούνται).

- fig: πρόκειται για την παράμετρο που καθορίζει αν τα plots θα υπολογίζονται ή όχι (1 ή 0 αντίστοιχα)
- vis: πρόκειται για την παράμετρο που καθορίζει αν τα plots θα είναι εμφανή στον χρήστη ή όχι ('on'=εμφανή, 'off'=κρυφά). Για να έχει νόημα αυτή τη παράμετρος θα πρέπει το fig να έχει τιμή 1.
- temp_vis: βοηθητική μεταβλητή – σημαντική μόνο για τεχνικές λεπτομέρειες της υλοποίησης σε matlab, χωρίς κάποια αξία για τον χρήστη
- Ni: το πλήθος των particles που θα περιέχεται σε κάθε swarm κατά την εκκίνηση του αλγορίθμου (τιμή αρχικοποίησης)
- Nmin: ο ελάχιστος επιτρεπτός αριθμός particles ανά swarm
- Nmax: ο μέγιστος επιτρεπτός αριθμός particles ανά swarm
- Si: ο αρχικός αριθμός των swarms που θα περιέχονται στο σύστημα
- Smin: ο ελάχιστος επιτρεπτός αριθμός swarms κάθε χρονική στιγμή
- Smax: ο μέγιστος επιτρεπτός αριθμός swarms κάθε χρονική στιγμή
- Np: ο συνολικός αριθμός particles – υπολογίζεται άμεσα από τις τιμές των παραπάνω παραμέτρων

- SCmax: η παράμετρος αυτή ονομάζεται κατώφλι στασιμότητας, και ρυθμίζει τον μέγιστο αριθμό κύκλων που επιτρέπεται να περάσουν, χωρίς το κάθε swarm να επιδείξει πρόοδο.

- SC: ο δείκτης στασιμότητας αρχικοποιείται με έναν πίνακα στήλη, Si θέσεων, οι οποίες περιέχουν μηδενικά

- N_kill: αρχικοποιούμε τον αριθμό των particles που έχουν αποκλειστεί από κάθε swarm με μηδενικά

- No: ο αριθμός των εμποδίων που υπάρχουν στον χώρο

- rs: η ακτίνα αντίληψης των αισθητήρων (καθορίζει στην ουσία την ακτίνα εντός της οποίας κάθε ρομπότ-particle μπορεί να αντιληφθεί ένα εμπόδιο

- r: η ακτίνα παρατηρήσεων (observation radius), εντός της οποίας κάθε particle μπορεί να υπολογίσει την τιμή της συνάρτησης κόστους (οι παρατηρήσεις αυτές θα προστίθενται κάθε φορά στον πίνακα observations)

-ev_step: καθορίζει κάθε πόσους κύκλους θα γίνεται η εκτίμηση για την συνάρτηση κόστους στον χώρο συνολικά με βάση τα δεδομένα του πίνακα observations

- color_matrix: πρόκειται για τον πίνακα που περιέχει τους κωδικούς των χρωμάτων που θα χρησιμοποιηθούν για την αναπαράσταση των swarms στα plots.

Στη συνέχεια καλούνται τα scripts create_space, create_swarms, main_rdpso και show_visual_result των οποίων η λειτουργία είτε περιγράφηκε ήδη είτε θα αναλυθεί παρακάτω.

Τέλος αποθηκεύεται σε μορφή βίντεο avi το οπτικό αποτέλεσμα της εκτέλεσης του αλγορίθμου, με τίτλο την ημερομηνία και την ώρα εκτέλεσης/αποθήκευσης.

11. main_rdpso

Το script αυτό αναλαμβάνει τη δρομολόγηση της εκτέλεσης του κυρίως αλγορίθμου, ενώ εδώ είναι που εφαρμόζεται και το κριτήριο τερματισμού στη συνθήκη του while-loop της 6ης γραμμής. Η μεταβλητή counter υπολογίζει πόσες φορές έχει εκτελεστεί συνολικά ο αλγόριθμος και η no_change_counter πόσες φορές έχει εκτελεστεί ο αλγόριθμος χωρίς κάποια μεταβολή στο παρατηρούμενο γενικό μέγιστο.

Κατά την εκτέλεση του loop η πρώτη ενέργεια που εκτελείται είναι η κλήση της calc_opt_pos_sensing_func για κάθε particle και η αποθήκευση της προκύπτουσας τιμής στο αντίστοιχο xg_opt πεδίο. Ακολούθως καλούμε την evolve_swarm για κάθε swarm. Όμοια διαδικασία ακολουθείται και για το excluded swarm. Ανανεώνουμε στη συνέχεια των πίνακα observations, και υπολογίζουμε το γενικό ελάχιστο ως το ελάχιστο του πίνακα αυτού. Ελέγχουμε με τις μεταβλητές condition_1, condition_2 αν έχει σημειωθεί κάποια μεταβολή στο γενικό βέλτιστο και ανανεώνουμε τον μετρητή no_change_counter αντίστοιχα. Τέλος, φτιάχνουμε το plot που δείχνει την συνάρτηση κόστους στον χώρο και τις θέσεις των particles και των εμποδίων, ενώ κάθε ev_step βήματα απεικονίζουμε και τον πίνακα observations σε σχέση με ολόκληρη τη συνάρτηση κόστους σε ξεχωριστό plot.

12. rand_num

Πρόκειται για μια συνάρτηση με τα εξής ορίσματα:

- a, b: δείχνουν το διάστημα [a,b] εντός του οποίου θα βρίσκονται οι τυχαίες τιμές
- i, j: υποδηλώνουν τις διαστάσεις του πίνακα, του οποίου τα στοιχεία θα είναι τυχαία

Προφανώς έξοδος της συνάρτησης είναι ο πίνακας r με διάσταση και τιμές αυτές που περιγράφονται πιο πάνω.

13. show_visual_result

Το script αυτό απλά ενώνει τα plots με τα στιγμιότυπα από τη main_rdpso και τα περνάει στη μεταβλητή visual_result ώστε να μπορέσουμε άμα θελήσουμε είτε να το αναπαράγουμε εντός matlab είτε να το αποθηκεύσουμε ως αρχείο βίντεο.

14. spawn_swarm

Ο σκοπός αυτού του script είναι να δημιουργήσει ένα καινούργιο swarm από τα υπάρχοντα particles του excluded. Έτσι, παίρνει τα πρώτα Ni particles από το excluded και τα προσθέτει στο cell-structure swarms ως μία νέα ομάδα. Στη συνέχεια αφαιρούνται τα στοιχεία τους από το excluded.

15. update_local_optimum

Η συνάρτηση αυτή που δέχεται ως ορίσματα το cell-structure swarms και την ακτίνα παρατήρησης r, επιστρέφει το swarms αφού πρώτα έχει βρει τις καινούργιες τιμές για καθένα από τα πεδία x_opt, τα σημεία δηλαδή στη "γειτονιά" του κάθε particle όπου η συνάρτηση κόστους ελαχιστοποιείται.

16. update_observations

Η συνάρτηση αυτή παίρνει ως ορίσματα τον πίνακα των observations, το cell-structure swarms, τα excluded και την ακτίνα παρατήρησης r, και με βάση αυτά βρίσκει τον νέο πίνακα observations και τον επιστρέφει. Ο υπολογισμός αυτό γίνεται κρατώντας τις τιμές της συνάρτησης κόστους cost_func εντός ακτίνας r από κάθε particle, τόσο για τα swarms όσο και για τα excluded.

17. update_velocity

Η συνάρτηση αυτή υπολογίζει την νέα τιμή της ταχύτητας για ένα συγκεκριμένο particle. Έχει τα εξής ορίσματα:

- xn: η θέση του particle στον χώρο
- x_opt: η θέση όπου η συνάρτηση κόστους (εντός της ακτίνας παρατήρησης) ελαχιστοποιείται
- xng_opt: το σημείο όπου η sensing function βελτιστοποιείται
- vel: η τρέχουσα ταχύτητας
- gn: το γενικό βέλτιστο (ελάχιστο)

Η νέα ταχύτητα υπολογίζεται με βάση τον τύπο (3) που αναφέρθηκε κατά την θεωρητική περιγραφή του αλγορίθμου, και διαιρείται με το μέτρο της, ώστε η επιστρεφόμενη ταχύτητα να έχει μοναδιαίο μέτρο τελικά.

6 *Επέκταση του αλγορίθμου RDPSO (Robotic Darwinian Particle Swarm Optimization) για πλοήγηση και εξερεύνηση, διατηρώντας παράλληλα ad hoc connectivity.*

6.1 Περιγραφή του αλγορίθμου

Στην πρώτη υλοποίηση θεωρήσαμε ότι τα ρομπότ (particles) διατηρούν ανά πάσα χρονική στιγμή επικοινωνία μεταξύ τους. Σε πραγματικά σενάρια όμως, κάτι τέτοιο δεν είναι πάντα εφικτό. Θα πρέπει επομένως να δοθεί κάποια λύση ώστε να επιτυγχάνεται ο στόχος του συστήματος χωρίς όλοι να επικοινωνούν με όλους. Η παρακάτω υλοποίηση κινείται προς αυτή την κατεύθυνση, και προτείνει να διατηρείται ανά πάσα στιγμή η επικοινωνία μεταξύ των particles κάθε swarm διασφαλίζοντας ότι δε θα χαλάει το MANET (Mobile Ad hoc NETwork) που θα δημιουργούν αυτά. Επίσης θα πρέπει η αρχική τοποθέτηση στον χώρο να γίνεται με τέτοιο τρόπο ώστε να μην παραβιάζεται αυτή η νέα συνθήκη. Τα δύο αυτά προβλήματα και οι αλγόριθμοι που τα αντιμετωπίζουν παρουσιάζονται αμέσως μετά

Διασφάλιση του MANET connectivity σε κάθε swarm:

Θα δημιουργήσουμε αρχικά έναν τετραγωνικό και συμμετρικό πίνακα A που η διάστασή του δίνεται από το πλήθος των particles στο συγκεκριμένο swarm. Στην θέση (i,j) θα έχει την τιμή 1 αν τα particles i,j συνδέονται μεταξύ τους άμεσα, διαφορετικά θα είναι 0. Θα πρέπει να τονιστεί εδώ ότι δύο ρομπότ-particles επικοινωνούν άμεσα αν βρίσκονται το πολύ σε απόσταση dmax το ένα από το άλλο – το dmax είναι μία από τις σημαντικές παραμέτρους του συστήματός μας και εισάγεται κατά την προσομοίωση από τον χρήστη, ενώ σε πραγματική υλοποίηση εξαρτάται προφανώς από την εμβέλεια του συστήματος επικοινωνίας του κάθε ρομπότ.

Ο πίνακας που μας ενδιαφέρει εδώ είναι ο C, ο connectivity matrix. Ο C υπολογίζεται επαναληπτικά σε N-1 βήματα, όπου N είναι το πλήθπς των particles στο swarm, και για τον υπολογισμό χρησιμοποιείται και ένας βοηθητικός πίνακας B. Ο αλγόριθμος υπολογισμού του C είναι ο εξής:

Αρχικοποιούμε τον $C^{(1)}=A$. Ακολουθούν N-2 επαναλήψεις συνολικά, όπου κατά την k-οστή υπολογίζεται ο πίνακας $B^{(k)}$, του οποίου τα στοιχεία δίνονται από τη σχέση

$$b_{ij}^{(k)}=\begin{cases}0, & c_{ij}^{(k-1)}>0\\ k, & \sum_{t=1}^{N} c_{it}^{(k-1)}\cdot b_{tj}^{(k-1)}>0 \;\kappa\alpha\iota\; c_{ij}^{(k-1)}=0\end{cases} \quad (4)$$

Αφού υπολογιστεί ο B, βρίσκουμε τον C από την σχέση

$C^{(k)}=C^{(k-1)}+B^{(k)}$ (5).

Η φυσική σημασία του C είναι ότι η τιμή του (i,j) στοιχείου του δείχνει αν τα στοιχεία i,j συνδέονται μεταξύ τους, και με πόσες παρεμβολές άλλων στοιχείων (0→ δεν συνδέονται, 1→ συνδέονται άμεσα, κτλ).

Ο πίνακας αυτός μας βοηθάει να διαπιστώνουμε ότι η συνδεσιμότητα (connectivity) μεταξύ των particles διατηρείται ενώ παράλληλα μπορεί να χρησιμεύσει για να βρούμε τον πλησιέστερο “γείτονα” για κάθε particle.

Η νέα ταχύτητα υπολογίζεται από την ακόλουθη σχέση:

$$v_n[t+1] = w \cdot v_n[t] + c_1 \cdot r_1 \cdot (\breve{g}_n[t] - x_n[t]) + c_2 \cdot r_2 \cdot (\breve{x}_n[t] - x_n[t]) + c_3 \cdot r_3 \cdot (\breve{x}_n^g[t] - x_n[t]) + c_4 \cdot r_4 \cdot (\breve{x}_n^m[t] - x_n[t])$$

(6)

όπου

c_4 : το βάρος που επιλέγουμε να έχει ο παράγοντας που διασφαλίζει το connectivity

r_4 : ο τυχαίος παράγοντας με τιμή μεταξύ 0 και 1

$\breve{x}_n^m[t]$: η θέση του πλησιέστρου γείτονα για το particle n αυξημένη κατά τη μέγιστη απόσταση επικοινωνίας dmax προς την κατεύθυνση της τρέχουσας θέσης του particle.

Για να γίνει καλύτερα κατανοητός ο τρόπος με τον οποίο λειτουργεί η παραπάνω μεθοδολογία θα είναι ίσως καλύτερο να προσωρήσουμε στην περιγραφή της υλοποίησης.

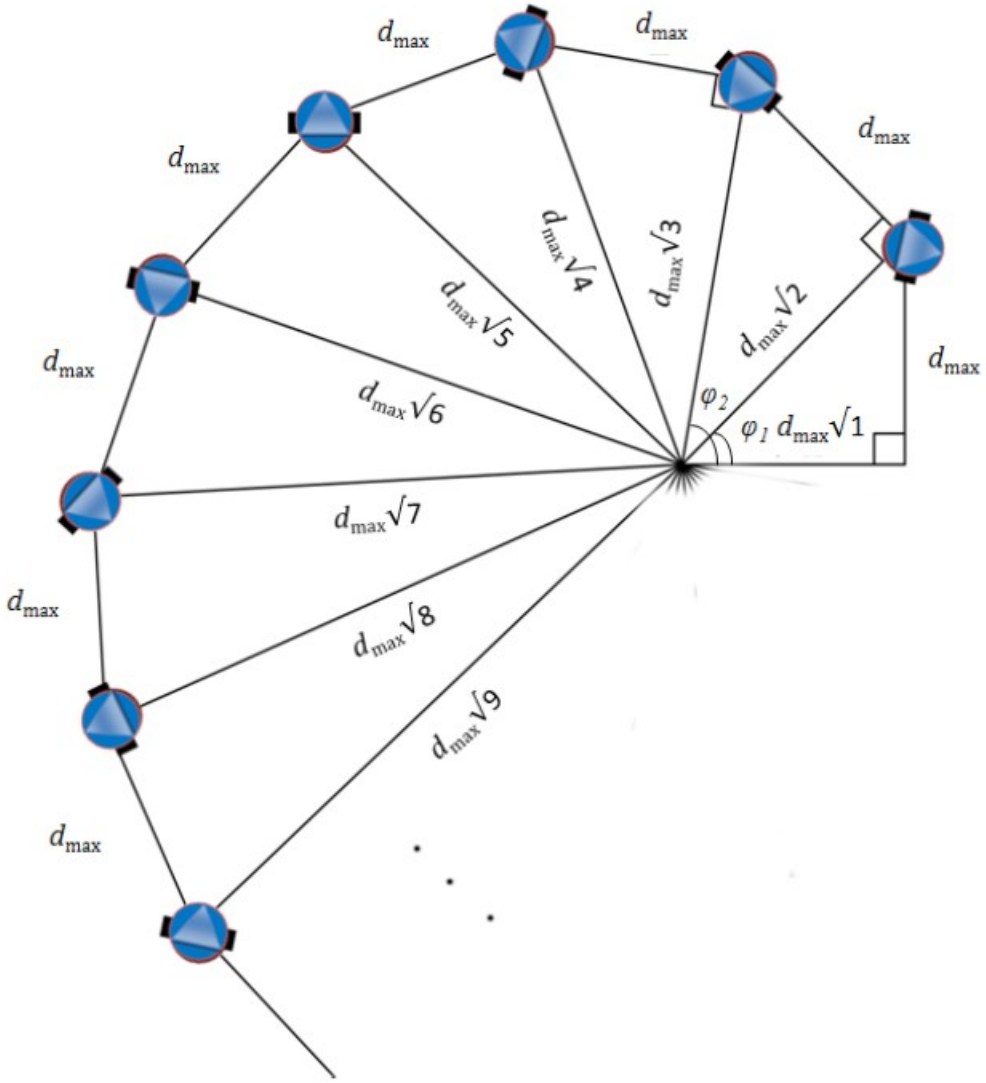
(Σε πολλά σημεία είναι όμοια με την προηγούμενη, όπου όλα τα particles επικοινωνούσαν με όλα, αλλά για λόγους πληρότητας της περιγραφής θα τα επαναλάβω.

Μεγάλη προσοχή θέλει επίσης και ο τρόπος με τον οποίο θα τοποθετηθούν τα particles κατά την έναρξη του αλγορίθμου. Μία πρακτική λύση είναι η χρήση του σπιράλ του Θεόδωρου. Αυτό έχει τη μορφή που φαίνεται στην ακόλουθη εικόνα και υπολογίζεται με βάση τους τύπους:

$$\varphi_\kappa = \arctan\left(\frac{1}{\sqrt{\kappa}}\right) \quad (7)$$

$$\varphi_n = \sum_{\kappa=1}^{n} \varphi_\kappa \quad (8)$$

$$x_n[0] = x_o + \begin{bmatrix} d_{max} \cdot \sqrt{n+1} \cdot \cos(\varphi_n) \\ d_{max} \cdot \sqrt{n+1} \cdot \sin(\varphi_n) \end{bmatrix} \quad (9)$$



Το x_o είναι το κέντρο του κάθε τέτοιου σπιράλ και η θέση του επιλέγεται τυχαία ώστε να διασφαλίζεται ο σωστός έλεγχος της αποδοτικότητας του στοχαστικού αλγορίθμου.

6.2 Λεπτομέρειες της υλοποίησης

Οι συναρτήσεις και τα scripts που δημιούργησα στο περιβάλλον του Matlab για να υλοποιήσω τον αλγόριθμο αυτό είναι οι ακόλουθες (σε αλφαβητική σειρά):

1. add_particle_from_excluded
2. calc_closest_neighbor_v2

3. calc_connectivity_matrices

4. calc_opt_pos_sensing_function

5. calc_opt_pos_sensing_function_excl

6. create_space

7. create_swarms_ad_hoc_con

8. delete_swarm

9. evolve_excluded

10. evolve_swarm

11. exclude_particle

12. main_ad_hoc_con

13. main_rdpso

14. rand_num

15. show_visual_result

16. spawn_swarm

17. test_connectivity

18. test_distances

19. update_local_optimum

20. update_observations

21. update_velocity

Λειτουργία των παραπάνω συναρτήσεων:

1. add_particle_from_excluded:

Ένα συγκεκριμένο particle από το excluded swarm, το best_particle (το οποίο υπολογίστηκε πριν να κληθεί το script αυτό), προστίθεται στο swarm s (όπου και πάλι το s υπολογίστηκε πριν την κλήση του script αυτού) και στη συνέχεια αφαιρείται από το excluded swarm. Προφανώς όταν μιλάμε για προσθήκη και διαγραφή εννοούμε όλα τα χαρακτηριστικά που περιγράφουν ένα particle και όχι μόνο τη θέση του.

Σε αντίθεση όμως με την απλούστερη υλοποίηση, εδώ θα πρέπει να ελέγξουμε αν το particle αυτό βρίσκεται εντός ακτίνας επικοινωνίας με τα υπόλοιπα paricles του swarm στο οποίο επιθυμούμε να το προσθέσουμε. Μόνο αν τηρείται το κριτήριο αυτό προχωράμε στην προσθήκη του στο swarm.

2. calc_closest_neighbor_v2

Η συνάρτηση αυτή λαμβάνει ως ορίσματα το cell-structure swarms, τα s, n που δείχνουν σε ποιο swarm και σε ποια θέση είναι το particle που θέλουμε να εξετάσουμε καθώς και τον πίνακα C_b, που μας δείχνει την τοπολογία του συστήματος. Βρίσκουμε για το ζητούμενο particle ποιο particle (ή ποια, αν είναι περισσότερα από ένα) βρίσκεται σε ακτίνα άμεσης επικοινωνίας από αυτό, και αποθηκεύουμε τις αντίστοιχες συσντεταγμένες. Αν είναι παραπάνω από ένα, τότε βρίσκουμε ανάμεσα σε αυτά ποιο είναι το πλησιέστερο. Επιστρέφουμε ως xm_opt για το συγκεκριμένο particle τη θέση του πλησιέστρου αυτού γείτονα αυξημένη κατά τη μέγιστη απόσταση επικοινωνίας dmax προς την κατεύθυνση της τρέχουσας θέσης του particle, όπως ακριβώς περιγράφηκε και πιο πάνω στην θεωρητική ανάλυση του αλγορίθμου.

3. calc_connectivity_matrices

Πρόκειται για script το οποίο υπολογίζει τους αναγκαίους πίνακες που μας δείχνουν την τοπολογία και το connectivity του εκάστοτε swarm του συστήματος. Στον πίνακα A υπολογίζουμε τα particles εκείνα που έχουν άμεση επικοινωνία μεταξύ τους. Στη συνέχεια, υπολογίζουμε με βάση τους τύπους (4) και (5) τους πίνακες B, C. Τέλος στον πίνακα C_b αποθηκεύουμε το $C^{(n-1)}$ για να χρησιμοποιηθεί παρακάτω.

4. calc_opt_pos_sensing_function:

Πρόκειται για συνάρτηση με τα ακόλουθα ορίσματα:

α. τη δομή swarms που περιέχει όλα τα χαρακτηριστικά των particles που ανήκουν στα ενεργά swarms

β. τον πίνακα obstacles με τις συντεταγμένες των σημείων στα οποία βρίσκονται τα εμπόδια

γ,δ. τα s, n αντιστοιχούν στο n-οστό particle του swarm s το οποίο και θέλουμε να εξετάσουμε

ε. rs είναι η ακτίνα “αντίληψης” των αισθητήρων για την ανίχνευση εμποδίων, η ακτίνα δηλαδή εντός της οποίας αν βρεθεί ένα εμπόδιο τότε αυτό θα γίνει αντιληπτό από το συγκεκριμένο ρομπότ

Οι αισθητήρες θεωρούμε ότι λειτουργούν με βάση μια φθίνουσα sensing function η οποία εδώ για λόγους απλότητας (χωρίς όμως και να προκαλείται κάποια αλλοίωση των αποτελεσμάτων) θεωρούμε ότι είναι η 1/x. Έτσι το αποτέλεσμα της συνάρτησης αυτής είναι το g_min_pos, η θέση δηλαδή στο χώρο (εντός όμως μια ακτίνας rc) στην οποία η sensing function ελαχιστοποιείται.

Για να το επιτύχουμε αυτό κρατάμε σε έναν πίνακα visible_obst τα εμπόδια και τα particles που βρίσκονται εντός της rc ακτίνας και άρα γίνονται αντιληπτά από το υπό εξέταση particle. Μετά φτιάχνουμε για κάθε σημείο εντός της rc έναν πίνακα g στον οποίο αποθηκεύεται το άθροισμα των αντιστρόφων αποστάσεων του σημείου αυτου από κάθε ορατό εμπόδιο. Ως g_min_pos θέτουμε τελικά το σημείο εκείνο στο οποίο ο πίνακας g έχει ελάχιστη τιμή.

5. calc_opt_pos_sensing_function_excl

Η συνάρτηση αυτή είναι εντελώς όμοια με την προηγούμενη, με μόνη διαφορά το ότι καθώς αναφέρεται στο excluded swarm δεν παίρνει ως όρισμα τιμή s, και άρα το υπό εξέταση particle είναι το n-οστό του excluded swarm.

6. create_space

Πρόκειται για script και όχι για συνάρτηση. Σημαντικές τιμές που χρησιμοποιούνται σε αυτή και δίνονται από την χρήστη ως σταθερές κατά κάποιον τρόπο είναι το num_Gaussians και το L, τα οποία δίνουν τον αριθμό των Gaussians που θα αθροίσουμε για να προκύψει η συνάρτηση κόστους του χώρου και η διάσταση του χώρου (με εμβαδό LxL) αντίστοιχα.

Στο πρώτο for-loop δημιουργούμε τις Gaussians με κέντρο m (ένα τυχαίο σημείο του χώρου) και πίνακα Sigma κοινό για όλες τις Γκαουσιανές που εξαρτάται κάθε φορά από το L. Η συνάρτηση κόστους που προκείπτει από το άθροισμα αυτό, αποθηκεύεται στον πίνακα cost_func και κατόπιν αντιστρέφεται ώστε να παρουσιάζει μοναδικό τοπικό ελάχιστο (αντί για μέγιστο που ήταν αρχικά). Ακολουθεί μία αναπαράσταση της συνάρτησης κόστους σε γράφημα και μετά δημιουργούμε τα εμπόδια με τυχαίο τρόπο (ο αριθμός No των εμποδίων δίνεται στην main.m, η οποία καλεί την create_space). Τέλος, δημιουργούμε και ένα plot για να δούμε την κατανομή των εμποδίων στον χώρο.

7. create_swarms_ad_hoc_con

Το αρχείο "create_swarms_ad_hoc_con.m" περιέχει ένα script το οποίο δημιουργεί όλες τις απαραίτητες αρχικοποιήσεις για τα particles του κάθε swarm ώστε να ξεκινήσει η εκτέλεση του κυρίως αλγορίθμου. Οι αρχικοποιήσεις αυτές συνίστανται στα παρακάτω:

- δημιουργούμε τη δομή cell μεγέθους Si x 1 που θα περιέχει όλα τα απαραίτητα στοιχεία

- swarms{s}.positions: αρχικοποιούμε τις θέσεις των particles με τρόπο που να διασφαλίζει τυχαιότητα αλλά και επικοινωνία μεταξύ των particles του κάθε swarm. Για τον λόγο αυτό χρησιμοποιούμε τις εξισώσεις (7), (8), (9) ενώ παράλληλα δίνουμε και τυχαίες τιμές στο xo ώστε να είναι κατανεμημένα στον χώρο τα swarms με τρόπο που να ευνοεί την αποτελεσματική όσο το δυνατόν εξερεύνησή του.

- swarms{s}.velocity: οι αρχικές ταχύτητες είναι τυχαίας κατεύθυνσης και μέτρου (μέγιστο μέτρο όμως είναι η μονάδα)

- swarms{s}.x_opt: θέτουμε ως θέσεις βέλτιστης τιμής της συνάρτησης κόστους κατά την αρχική αυτή χρονική στιγμή τις θέσεις των particles.

- swarms{s}.xg_opt: στη συνέχεια υπολογίζεται με τη βοήθεια της calc_opt_pos_sensing_func το σημείο του γειτονικού χώρου του κάθε particle όπου η sensing function μηδενίζετα

- gn είναι η θέση γενικού βελτίστου και gn_cost η τιμή του βελτίστου αυτού (όπως υπολογίζονται σε κάθε κύκλο του αλγορίθμου – όχι το πραγματικό ολικό βέλτιστο του χώρου) – στην τιμή αρχικοποίησής τους καταλήγουμε αφού βρούμε την τιμή της συνάρτησης κόστους για την παρούσα θέση του κάθε particle και εξετάσουμε ποια από αυτές είναι η βέλτιστη

- οι μεταβλητές condition_1, condition_2 περιέχουν αντίστοιχα την προηγούμενη τιμή του ολικού βελτίστου και την τρέχουσα. Αφαιρώντας τις δύο λαμβάνουμε την αναγκαία συνθήκη τερματισμού του loop στο script main_rdpso, και τις αρχικοποιούμε με τον πλησιέστερο αριθμό στο μηδέν που μπορεί να αναγνωρίσει το περιβάλλον του Matlab και την τρέχουσα τιμή γενικού βελτίστου (αντίστοιχα).

-το swarm των excluded particles δεν θα περιέχει αρχικά κάποιο particle, οπότε δημιουργούμε τη δομή του και το αφήνουμε κενό

- ο πίνακας cost_obst_fig περιέχει τις τιμές της συνάρτησης κόστους στα σημεία όπου βρίσκονται τα εμπόδια – αυτό γίνεται μόνο για να φαίνεται οπτικά ομοιόμορφο το αποτέλεσμα στα plots όταν απεικονίζονται τα εμπόδια

8. delete_swarm

Όταν, υπό κατάλληλες συνθήκες καλείται αυτό το script, τότε το s-οστό swarm διαγράφεται και τα particles που το απαρτίζουν μεταφέρονται στο excluded swarm, μαζί με όλα τα χαρακτηριστικά τους.

9. evolve_excluded

Πρόκειται για συνάρτηση με μόνο όρισμα το excluded και επιστρεφόμενη τιμή πάλι το excluded, αυτή τη φορά μετά τις αλλαγές που έχει υποστεί εντός της συνάρτησης.

Η μέθοδος που ακολουθούμε για να “εξελίξουμε” σε κάθε κύκλο του αλγορίθμου το swarm των excluded είναι σαφώς απλούστερη αυτής που ακολουθούμε για τα υπόλοιπα swarms (το γιατί γίνεται απόλυτα σαφές αμέσως μόλις δει κανείς την υλοποίηση του evolve_swarm.m). Δίνουμε στα particles τυχαία, μοναδιαία ταχύτητα. Έπειτα, βρίσκουμε την καινούργια θέση με βάση την προηγούμενη και την νέα (τυχαία) ταχύτητα, πολλαπλασιασμένη όμως επί 4 (προσοχή: στο evolve_swarm στο αντίστοιχο σημείο η ταχύτητα πολλαπλασιάζεται επί 2 – ο λόγος είναι ότι τα particles στο swarm των excluded πρέπει να είναι αισθητό ότι κινούνται πιο απρόβλεπτα και ταχύτερα, προκειμένου να σημειωθεί κάποια βελτίωση αν έχει κολλήσει ο αλγόριθμος σε κάποιο τοπικό μέγιστο. Αυτό ακολουθεί λογική που οι άνθρωποι σε κοινωνικά αποκλεισμένες ομάδες “κινούνται” σχετικά ανεξέλεγκτα και παρουσιάζουν σε κάποιες περιπτώσεις εντυπωσιακά

αποτελέσματα σε κάποιον τομέα, πχ στην τέχνη. Ελέγχουμε επίσης οι συντεταγμένες της νέας θέσης να μην ξεφεύγουν από τα όρια του χώρου με τέσσερα if. Τέλος υπολογίζουμε την τιμή της συνάρτησης κόστους για τις νέες θέσεις, και δίνουμε μια τιμή στο improvement του κάθε particle (το οποίο πρακτικά δεν έχει κάποια χρησιμότητα, παρά μόνο αν το particle το συγκεκριμένο μεταφερθεί σε κάποιο άλλο swarm).

10. evolve_swarm

Πρόκειται για συνάρτηση με τα ακόλουθα ορίσματα:

-swarms: η δομή που περιέχει όλα τα particles που ανήκουν στα "ενεργά" swarms και τα χαρακτηριστικά τους

- s: το swarm που εξετάζουμε

- gn: τη θέση του γενικού βελτίστου που έχουν εντοπίσει τα paricles μέχρι στιγμής κατά την εκτέλεση του κώδικα

- SC: ο δείκτης στασιμότητας (stagnancy index) – ένας δείκτης που εξετάζει αν έχει ξεπεραστεί το κατώφλι στασιμότητας (stagnancy threshold), η τιμή δηλαδή που καθορίζει πόσοι κύκλοι επιτρέπεται να περάσουν κατά τους οποίους το συγκεκριμένο swarm δεν κατέληξε σε κάποια βελτίωση. Όταν το κατώφλι αυτό ξεπεραστεί τότε θα επέλθει κάποια προκαθορισμένη ποινή στο swarm ή στο particle αναλόγως, όπως θα φανεί και στη συνέχεια.

-N_kill: είναι ένας πίνακας-στήλη με τόσες θέσεις όσα είναι και τα swarms και σε κάθε θέση περιέχει πόσα particles από το αντίστοιχο swarm έχουν αποκλειστεί (και έχουν μεταφερθεί στο excluded δηλαδή)

Οι τιμές που επιστρέφονται είναι οι εξής:

- swarms: η ανανεωμένη δομή που περιέχει τα χαρακτηριστικά για τα particles που συναποτελούν το σύστημά μας

- SC: η νέα τιμή του δείκτη στασιμότητας, η οποία ενδέχεται να είχε μεταβληθεί κατά την εκτέλεση της συνάρτησης.

- N_kill: η νέα τιμή για τον πίνακα που προαναφέρθηκε στα ορίσματα

- next_swarm: ο μόνος λόγος ύπαρξης αυτής της τιμής ως έξοδος της συνάρτησης αυτής είναι η περίπτωση κατά την οποία έχουμε διαγραφή ενός swarm – τότε θα πρέπει να

επιστραφεί η τιμή του επόμενου swarm που θα υποστεί επεξεργασία (διαφορετικά προκύπτουν ποικίλα σφάλματα λόγω της μη αναμενόμενης διάστασης του cell-stusture swarm.

Ακολουθεί η περιγραφή της λειτουργίας της συνάρτησης:

Υπολογίζουμε αρχικά τους απαραίτητους πίνακες για την τοπολογία του swarm με το calc_connectivity_matrices και στη συνέχεια, χρησιμοποιώντας τη συνάρτηση update_velocity, της οποίας η λειτουργία θα περιγραφεί παρακάτω, βρίσκουμε τη νέα ταχύτητα των particles και την εισάγουμε στο αντίστοιχο πεδίο του swarms{s}. Βρίσκουμε την νέα θέση του κάθε particle – η σχέση που χρησιμοιποιείται είναι ανάλογη με αυτή του evolve_excluded, με τη διαφορά όμως που επισημάνθηκε ως προς τον συντελεστή που επιλέξαμε για να πολλαπλασιαστεί η ταχύτητα, ο οποίος είναι 2 εδώ αντί για 4 στα excluded. Αποθηκεύουμε προσωρινα την προηγούμενη θέση στη μεταβλητή temp_prev_position και υπολογίζουμε την νέα θέση. Επανεκτιμώντας τους πίνακες A, B, C με την calc_connectivity_matrices ελέγχουμε αν με την νέα θέση διατηρείται το connectivity ψάχνοντας τον πλησιέστερο γείτονα στο particle αυτό. Αν δεν διατηρείται τότε θέτουμε ως νέα θέση την μέση τιμή της προηγούμενης με αυτή που υπολογίσαμε τελευταία. Μετά θα ελέγξουμε πάλι με τέσσερα if, ώστε οι νέες συντεταγμένες να βρίσκονται εντός του χώρου.

Βασικό μέγεθος της συνάρτησης αυτής είναι το improvement, το οποίο υποδηλώνει τη συνολική βελτίωση του swarm αυτού και το κατά πόσο δηλαδή πλησιάζει στον επιθυμητό στόχο, και προκύπτει ως το άθροισμα του πεδίου improvement για κάθε ένα particle του συγκεκριμένου swarm.

Αν το improvement είναι θετικό, τότε ελέγχουμε αν υπάρχουν excluded particles και αν το πλήθος των particles στο swarm αυτό δεν ξεπερνά το μέγιστο. Αν αυτά ισχύουν τότε προσθέτουμε στο τρέχον swarm το particle εκείνο από τα excluded που έχει την καλύτερη επίδοση (best performing).

Αν το improvement είναι αρνητικό τότε αυξάνουμε τον δείκτη στασιμότητας, SC, κατά 1. Αν η τιμή του τότε ξεπερνάει το κατώφλι στασιμότητας SCmax, θα εξετάσουμε το πλήθος των particles στο swarm αυτό. Αν υπερβαίνει την ελάχιστη επιτρεπτή τιμή τότε αποκλείουμε από αυτό το particle με τη χειρότερη επίδοση. Διαφορετικά, υπάρχει μια μικρή πιθανότητα

(η οποία είναι ανάλογη του πλήθους των swarms που υπάρχουν προς το μέγιστο επιτρεπτό πλήθος, αλλά είναι και ως ένα βαθμό τυχαία) το παρόν swarm να διαγραφεί και τα particles που το απαρτίζουν να μεταφερθούν στο excluded swarm.

Τέλος, αν κανένα particle δεν έχει διαγραφεί από το swarm αυτό, το συνολικό πλήθος των swarms δεν υπερβαίνει το μέγιστο και το excluded swarm περιέχει τουλάχιστον Ni particles (το ελάχιστο δηλαδή πλήθος ανά swarm) υπάρχει μια (μικρή) πιθανότητα να δημιουργηθεί καινούργιο swarm.

11. exclude_particle

Κατ' αναλογία με την περίπτωση του script “add_particle_from_excluded“, μεταφέρουμε ένα particle από το τρέχον swarm (αυτό με τη χειρότερη επίδοση για την ακρίβεια) στο swarm των socially excluded.

12. main_ad_hoc_con

Πρόκειται για το βασικότερο script όλου του κώδικα. Αυτό είναι που καλούμε και αυτόματα γίνονται τα υπόλοιπα. Επίσης εδώ είναι οι περισσότερες παράμετροι που επηρεάζουν τον αλγόριθμο και στις οποίες θέλουμε να επεμβαίνουμε για να εξετάζουμε την απόδοση υπό διαφορετικές συνθήκες. Οι λειτουργίες των παραμέτρων αυτών θα επεξηγηθούν στη συνέχεια (παρότι μια σύντομη περογραφή εμφανίζεται και στις συναρτήσεις/scripts όπου χρησιμοποιούνται).

- fig: πρόκειται για την παράμετρο που καθορίζει αν τα plots θα υπολογίζονται ή όχι (1 ή 0 αντίστοιχα)
- vis: πρόκειται για την παράμετρο που καθορίζει αν τα plots θα είναι εμφανή στον χρήστη ή όχι ('on'=εμφανή, 'off'=κρυφά). Για να έχει νόημα αυτή τη παράμετρος θα πρέπει το fig να έχει τιμή 1.
- temp_vis: βοηθητική μεταβλητή – σημαντική μόνο για τεχνικές λεπτομέρειες της υλοποίησης σε matlab, χωρίς κάποια αξία για τον χρήστη

- Ni: το πλήθος των particles που θα περιέχεται σε κάθε swarm κατά την εκκίνηση του αλγορίθμου (τιμή αρχικοποίησης)

- Nmin: ο ελάχιστος επιτρεπτός αριθμός particles ανά swarm

- Nmax: ο μέγιστος επιτρεπτός αριθμός particles ανά swarm

- Si: ο αρχικός αριθμός των swarms που θα περιέχονται στο σύστημα

- Smin: ο ελάχιστος επιτρεπτός αριθμός swarms κάθε χρονική στιγμή

- Smax: ο μέγιστος επιτρεπτός αριθμός swarms κάθε χρονική στιγμή

- Np: ο συνολικός αριθμός particles – υπολογίζεται άμεσα από τις τιμές των παραπάνω παραμέτρων

- SCmax: η παράμετρος αυτή ονομάζεται κατώφλι στασιμότητας, και ρυθμίζει τον μέγιστο αριθμό κύκλων που επιτρέπεται να περάσουν, χωρίς το κάθε swarm να επιδείξει πρόοδο.

- SC: ο δείκτης στασιμότητας αρχικοποιείται με έναν πίνακα στήλη, Si θέσεων, οι οποίες περιέχουν μηδενικά

- N_kill: αρχικοποιούμε τον αριθμό των particles που έχουν αποκλειστεί από κάθε swarm με μηδενικά

- No: ο αριθμός των εμποδίων που υπάρχουν στον χώρο

- rs: η ακτίνα αντίληψης των αισθητήρων (καθορίζει στην ουσία την ακτίνα εντός της οποίας κάθε ρομπότ-particle μπορεί να αντιληφθεί ένα εμπόδιο

- dmax: η μέγιστη επιτρεπτή απόσταση από το πλησιέστερο particle του ίδιου swarm ώστε να διασφαλίζεται το connectivity

- r: η ακτίνα παρατηρήσεων (observation radius), εντός της οποίας κάθε particle μπορεί να υπολογίσει την τιμή της συνάρτησης κόστους (οι παρατηρήσεις αυτές θα προστίθενται κάθε φορά στον πίνακα observations)

-ev_step: καθορίζει κάθε πόσους κύκλους θα γίνεται η εκτίμηση για την συνάρτηση κόστους στον χώρο συνολικά με βάση τα δεδομένα του πίνακα observations

- color_matrix: πρόκειται για τον πίνακα που περιέχει τους κωδικούς των χρωμάτων που θα χρησιμοποιηθούν για την αναπαράσταση των swarms στα plots.

Στη συνέχεια καλούνται τα scripts create_space, create_swarms_ad_hoc_con, main_rdpso και show_visual_result των οποίων η λειτουργία είτε περιγράφηκε ήδη είτε θα αναλυθεί παρακάτω.

Τέλος αποθηκεύεται σε μορφή βίντεο avi το οπτικό αποτέλεσμα της εκτέλεσης του αλγορίθμου, με τίτλο την ημερομηνία και την ώρα εκτέλεσης/αποθήκευσης.

13. main_rdpso

Το script αυτό αναλαμβάνει τη δρομολόγηση της εκτέλεσης του κυρίως αλγορίθμου, ενώ εδώ είναι που εφαρμόζεται και το κριτήριο τερματισμού στη συνθήκη του while-loop της 6ης γραμμής. Η μεταβλητή counter υπολογίζει πόσες φορές έχει εκτελεστεί συνολικά ο αλγόριθμος και η no_change_counter πόσες φορές έχει εκτελεστεί ο αλγόριθμος χωρίς κάποια μεταβολή στο παρατηρούμενο γενικό μέγιστο.

Κατά την εκτέλεση του loop η πρώτη ενέργεια που εκτελείται είναι η κλήση της calc_opt_pos_sensing_func για κάθε particle και η αποθήκευση της προκύπτουσας τιμής στο αντίστοιχο xg_opt πεδίο. Ακολούθως καλούμε την evolve_swarm για κάθε swarm. Όμοια διαδικασία ακολουθείται και για το excluded swarm. Ανανεώνουμε στη συνέχεια των πίνακα observations, και υπολογίζουμε το γενικό ελάχιστο ως το ελάχιστο του πίνακα αυτού. Ελέγχουμε με τις μεταβλητές condition_1, condition_2 αν έχει σημειωθεί κάποια μεταβολή στο γενικό βέλτιστο και ανανεώνουμε τον μετρητή no_change_counter αντίστοιχα. Τέλος, φτιάχνουμε το plot που δείχνει την συνάρτηση κόστους στον χώρο και τις θέσεις των particles και των εμποδίων, ενώ κάθε ev_step βήματα απεικονίζουμε και τον πίνακα observations σε σχέση με ολόκληρη τη συνάρτηση κόστους σε ξεχωριστό plot.

14. rand_num

Πρόκειται για μια συνάρτηση με τα εξής ορίσματα:

- a, b: δείχνουν το διάστημα [a,b] εντός του οποίου θα βρίσκονται οι τυχαίες τιμές
- i, j: υποδηλώνουν τις διαστάσεις του πίνακα, του οποίου τα στοιχεία θα είναι τυχαία

Προφανώς έξοδος της συνάρτησης είναι ο πίνακας r με διάσταση και τιμές αυτές που περιγράφονται πιο πάνω.

15. show_visual_result

Το script αυτό απλά ενώνει τα plots με τα στιγμιότυπα από τη main_rdpso και τα περνάει στη μεταβλητή visual_result ώστε να μπορέσουμε άμα θελήσουμε είτε να το αναπαράγουμε εντός matlab είτε να το αποθηκεύσουμε ως αρχείο βίντεο.

16. spawn_swarm

Ο σκοπός αυτού του script είναι να δημιουργήσει ένα καινούργιο swarm από τα υπάρχοντα particles του excluded. Έτσι, στην απλούστερη παίρνει τα πρώτα Ni particles από το excluded και τα προσθέτει στο cell-stusture swarms ως μία νέα ομάδα. Στη συνέχεια αφαιρούνται τα στοιχεία τους από το excluded.

Επειδή όμως μας ενδιαφέρει παράλληλα να υπάρχει επικοινωνία μεταξύ των στοιχείων του καινούργιου αυτού swarm θα πρέπει να κάνουμε πρώτα έναν έλεγχο για να διασφαλίσουμε ότι βρίσκονται αρκετά κοντά μεταξύ τους. Έτσι υπολογίζουμε τον πίνακα A όπως ακριβώς και για τον υπολογισμό του connectivity matrix C_b, και στη συνέχεια ελέγχουμε ανά γραμμή τις τιμές. Αν το άθροισμα κάποιας γραμμής είναι 0 τότε πάει να πει ότι δεν έχει επικοινωνία με κάποιο άλλο particle από το excluded group και άρα το εξαιρούμε απο τα υποψήφια για το νέο swarm. Αν συνολικά έχουμε κρατήσει τουλάχιστον Ni particles (γιατί με λιγότερα δε μπορούμε να έχουμε swarm) τότε προχωράμε στον σχηματισμό του, όπως και πριν.

17. test_connectivity

Το script αυτό εξυπηρετεί στο να εντοπίζει αν την στιγμή της κλήσης του έχει χαθεί η επικοινωνία ενός από τα particles του swarm με τα υπόλοιπα. Είναι ένα καθαρά βοηθητικό script.

18. test_distances

Το script αυτό καλείται κατά την αρχική τοποθέτηση των particles στον χώρο, και εντοπίζει αν χάνεται η επικοινωνία μεταξύ των particles καθώς τα τοποθετούμε στις διακριτές θέσεις

του grid που υπάρχει στον χώρο. Αν συμβεί αυτό τότε μειώνουμε την μεταξύ τους απόσταση και επαναλαμβάνουμε την διαδικασία μέχρις ότου να έχουμε αποδεκτές αποστάσεις ανάμεσα στα particles.

19. update_local_optimum

Η συνάρτηση αυτή που δέχεται ως ορίσματα το cell-stusture swarms και την ακτίνα παρατήρησης r, επιστρέφει το swarms αφού πρώτα έχει βρει τις καινούργιες τιμές για καθένα από τα πεδία x_opt, τα σημεία δηλαδή στη "γειτονιά" του κάθε particle όπου η συνάρτηση κόστους ελαχιστοποιείται.

20. update_observations

Η συνάρτηση αυτή παίρνει ως ορίσματα τον πίνακα των observations, το cell-stusture swarms, τα excluded και την ακτίνα παρατήρησης r, και με βάση αυτά βρίσκει τον νέο πίνακα observations και τον επιστρέφει. Ο υπολογισμός αυτό γίνεται κρατώντας τις τιμές της συνάρτησης κόστους cost_func εντός ακτίνας r από κάθε particle, τόσο για τα swarms όσο και για τα excluded.

21. update_velocity

Η συνάρτηση αυτή υπολογίζει την νέα τιμή της ταχύτητας για ένα συγκεκριμένο particle. Έχει τα εξής ορίσματα:

- xn: η θέση του particle στον χώρο
- x_opt: η θέση όπου η συνάρτηση κόστους (εντός της ακτίνας παρατήρησης) ελαχιστοποιείται
- xng_opt: το σημείο όπου η sensing function βελτιστοποιείται
- xm_opt: η θέση του πλησιέστρου αυτού γείτονα αυξημένη κατά τη μέγιστη απόσταση επικοινωνίας dmax προς την κατεύθυνση της τρέχουσας θέσης του particle
- vel: η τρέχουσα ταχύτητας
- gn: το γενικό βέλτιστο (ελάχιστο)

Η νέα ταχύτητα υπολογίζεται με βάση τον τύπο (6) που αναφέρθηκε κατά την θεωρητική περιγραφή του αλγορίθμου, και διαιρείται με το μέτρο της, ώστε η επιστρεφόμενη ταχύτητα να έχει μοναδιαίο μέτρο τελικά.

7 Εκτέλεση του κώδικα σε Matlab – Έλεγχος αποτελεσμάτων

7.1 Αποτελέσματα του RDPSO αλγορίθμου (χωρίς να μας ενδιαφέρει το ad hoc connectivity)

Βασικός παράγοντας που επηρεάζει την επιτυχία του αλγορίθμου είναι η κατανομή των particles στον χώρο. Καθότι αυτή γίνεται εντελώς τυχαία ενώ ταυτόχρονα και η μορφολογία της συνάρτησης κόστους είναι σε πολύ μεγάλο βαθμό τυχαία επίσης, δεν έχουμε κάποιο τρόπο να επηρεάσουμε την απόδοση ως προς αυτό το χαρακτηριστικό. Η αύξηση του αριθμού των συνολικών particles μπορούμε να πούμε ότι βοηθάει κάπως, καθώς αυξάνονται οι πιθανότητες να "καλύψουμε" όλο τον χώρο, αλλά αυτό σε καμία περίπτωση δε μας διασφαλίζει.

Επίσης βασικός παράγοντας που επηρεάζει την απόδοση, κυρίως ως προς την ταχύτητα σύγκλισης όμως, είναι η σχέση ανάμεσα στο μέγεθος του χώρου (πλευράς L) και ο συνολικός αριθμός των particles στο σύστημα. Αν το πλήθος των particles δεν επαρκεί (πρόκειται για διαισθητική έννοια, καθώς δεν είναι μόνο αυτή η αναλογία που επηρεάζει τα αποτελέσματά μας, και άρα δεν είναι εύκολο ούτε και ακριβές να μετρηθεί ως συντελεστής επίδρασης) τότε ο χώρος δεν εξερευνείται πλήρως και το ολικό βέλτιστο ως σημείο μπορεί να παραβλεφθεί από τον αλγόριθμο.

Παρακάτω θα εκτελέσουμε τον αλγόριθμο για κάποιες διαφορετικές τιμές των συντελεστών αυτών, ώστε τα παραπάνω να γίνουν εμφανή και απτά για τον αναγνώστη. Πριν όμως προχωρήσουμε περισσότερο, καλό θα είναι να γίνουν κάποιες διευκρινίσεις ώστε να είναι κατανοητά τα στιγμιότυπα που παρατίθενται.

- Στον χώρο απεικονίζεται η συνάρτηση κόστους υπό τη μορφή 2D κυματομορφής στον χώρο μας.

- Τα εμπόδια συμβολίζονται με τετράγωνα μαύρου χρώματος. Παρότι είναι σημειακά τα εμπόδια, στην αναπαράστασή τους φαίνεται να καταλαμβάνουν περισσότερο χώρο – αυτό συμβαίνει απλά για να είναι ορατά και ευδιάκριτα στον αναγνώστη.

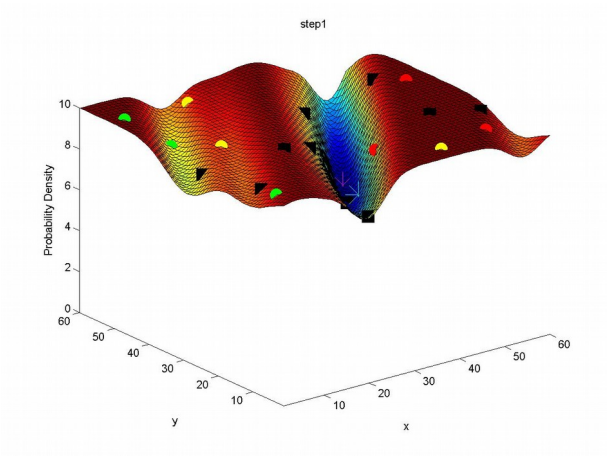
- Τα particles συμβολίζονται με κυκλάκια διαφορετικού χρώματος ανάλογα με το swarm στο οποίο ανήκουν. Η αντιστοιχία χρωμάτων είναι η ακόλουθη: 1ο swarm → κόκκινο, 2ο swarm πράσινο, 3ο swarm → κίτρινο. (Υπάρχει δυνατότητα για χρήση περισσότερων swarms και άρα περισσότερων χρωμάτων, αλλά οι εκτελέσεις του κώδικα που παρουσιάζονται εδώ περιορίζονται σε τρία)

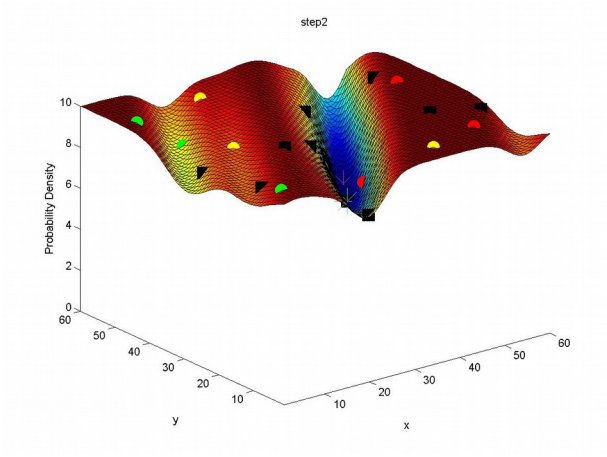
- Το ολικό βέλτιστο του χώρου συμβολίζεται με ένα αστεράκι χρώματος μωβ και παραμένει, όπως είναι λογικό, σταθερό καθόλη τη διάρκεια της εκτέλεσης.

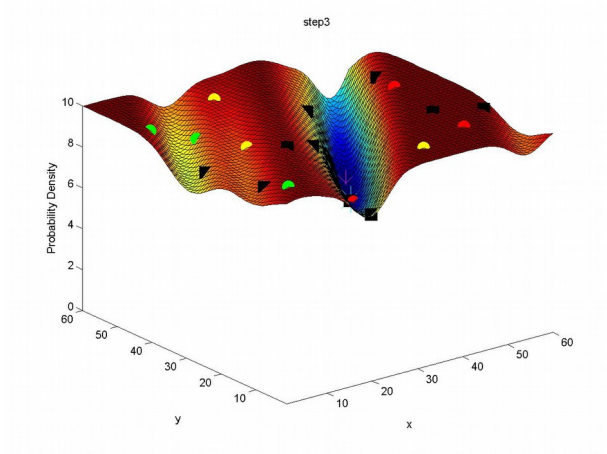
- Το γενικό βέλτιστο, με βάση τις παρατηρούμενες τιμές, συμβολίζεται με ένα αστεράκι γαλάζιου χρώματος και μεταβάλλεται καθώς τα particles κινούνται και στο σύστημα εισάγονται οι νέες παρατηρήσεις. Στόχος του είναι, όπως είναι εμφανές, να ταυτιστεί η θέση του με τη θέση ολικού βελτίστου που περιγράψαμε αμέσως πριν.

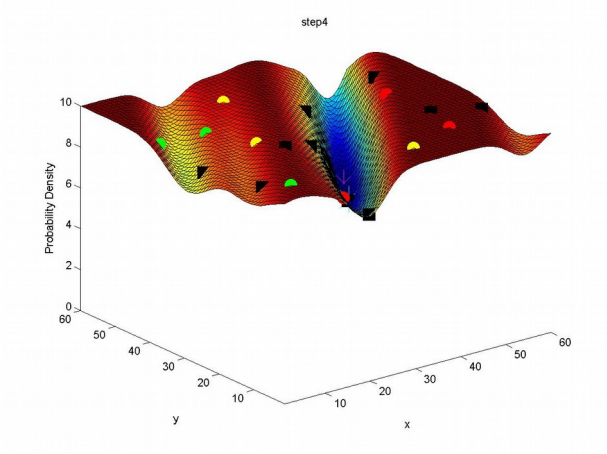
Επίσης, το κάθε στιγμιότυπο – plot έχει ως τίτλο το βήμα κατά το οποίο ελήφθη, ενώ ο χώρος που ορίσαμε έχει εμβαδό 60 x 60.

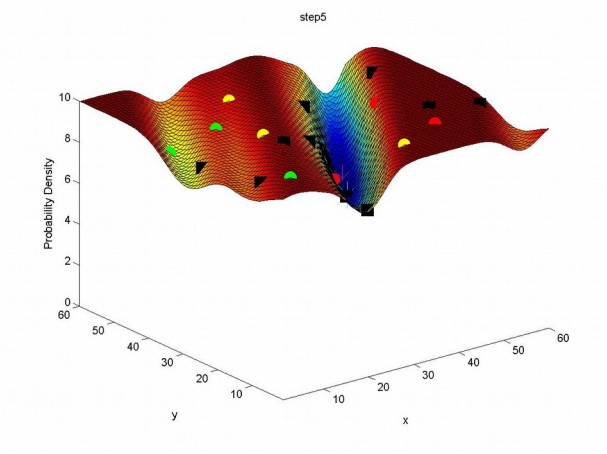
Εκτέλεση 1η:

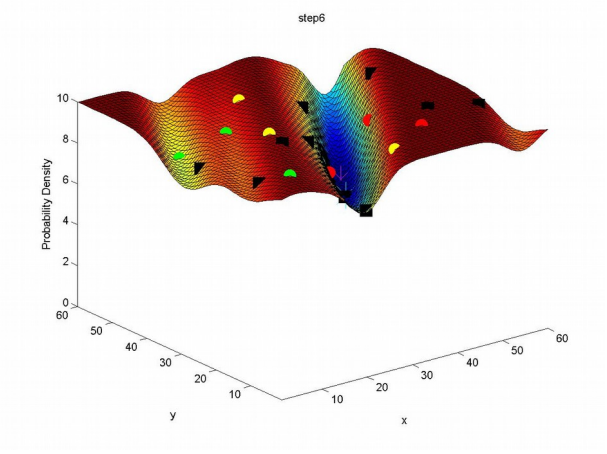

step1
Probability Density
y
x

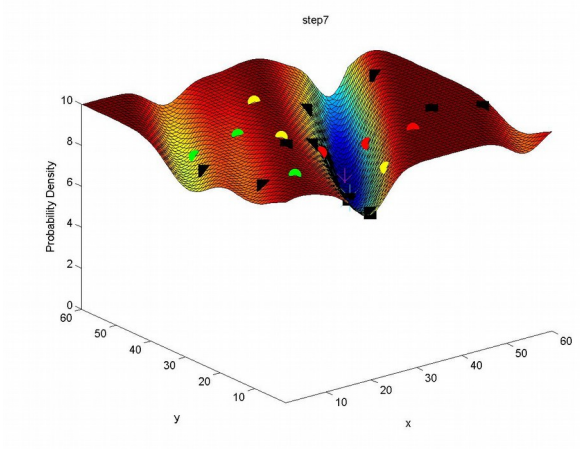

step2
Probability Density
y
x

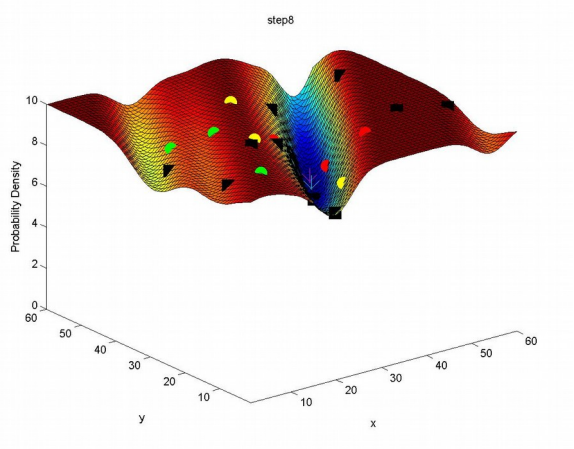

step3
Probability Density
y
x

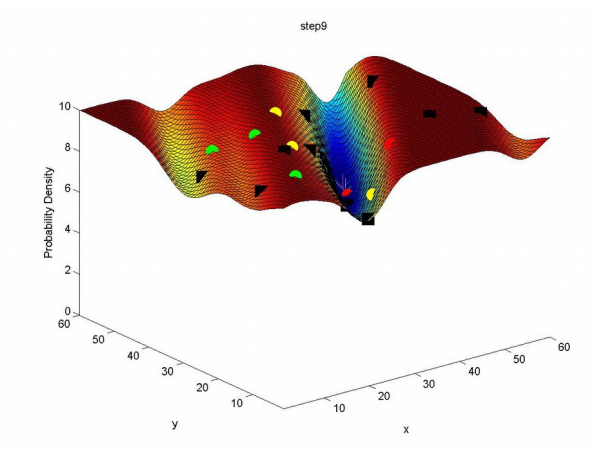

step4
Probability Density
y
x

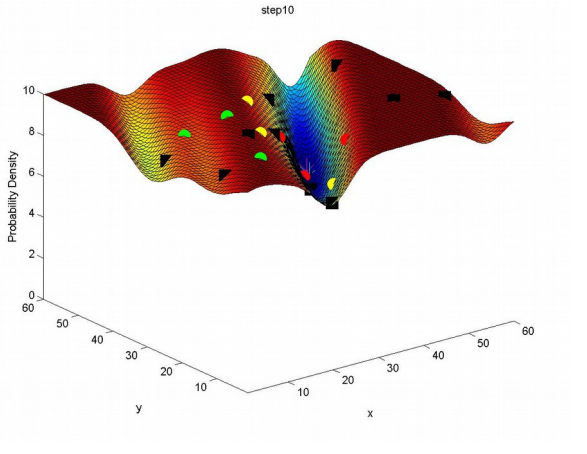

step5
Probability Density
y
x

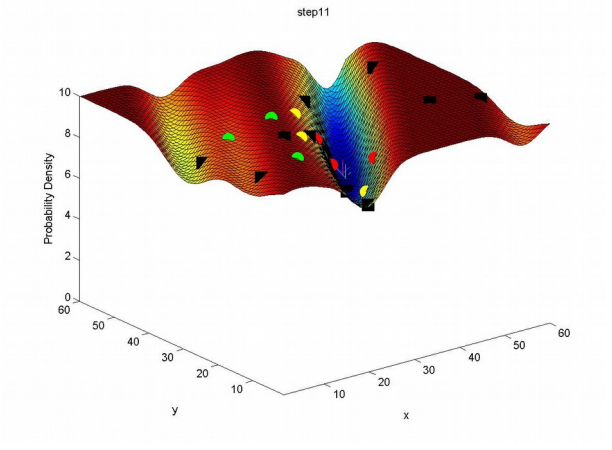

step6
Probability Density
y
x

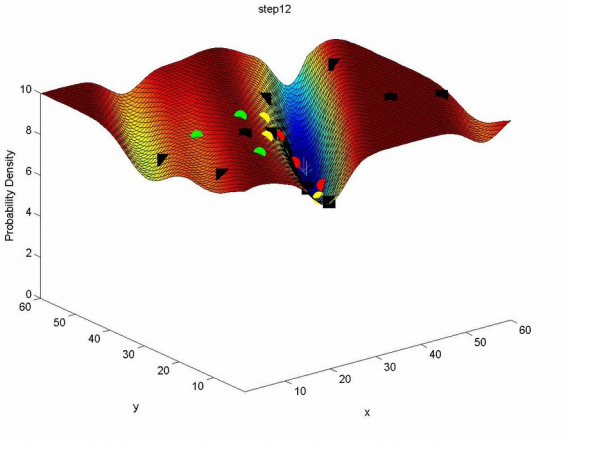

step7
Probability Density
y
x

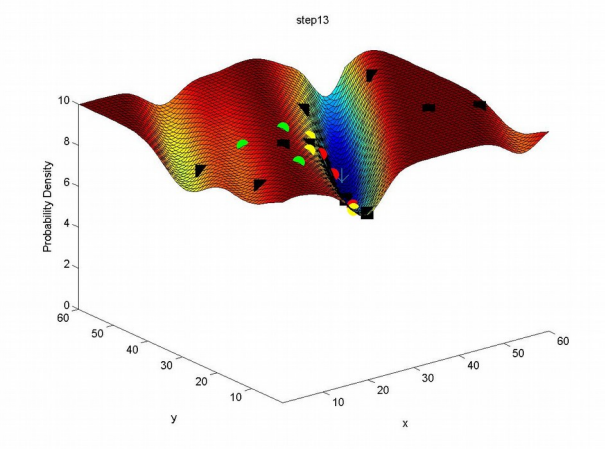

step8
Probability Density
y
x


step9
Probability Density
y
x


step10
Probability Density
y
x


step11
Probability Density
y
x

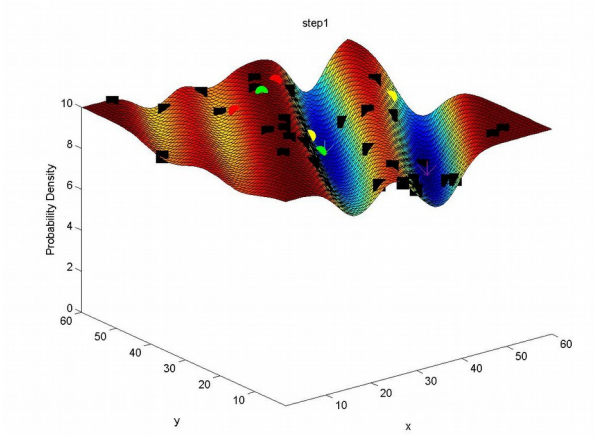

step12
Probability Density
y
x

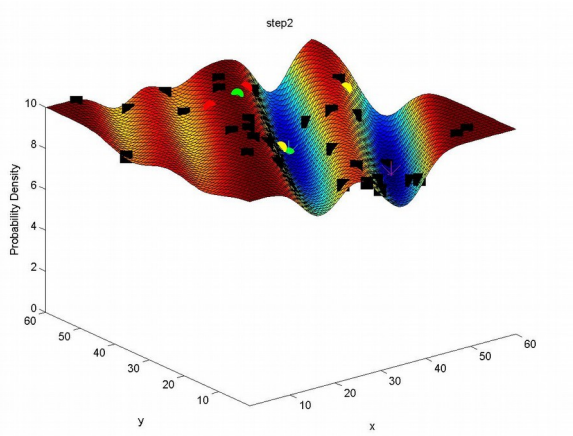


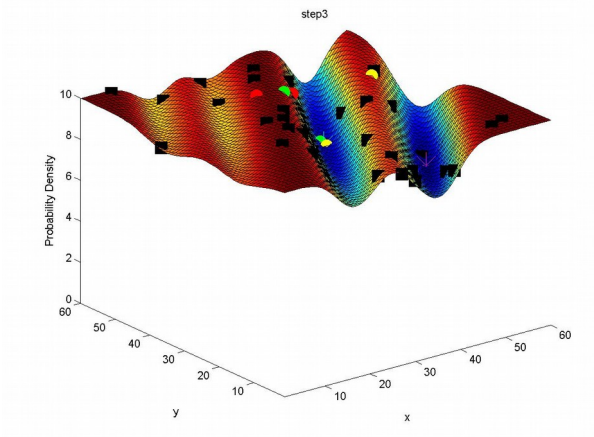
Στα παραπάνω στιγμιότυπα βλέπει κανείς την εξέλιξη του συστήματός μας με την πάροδο του χρόνου.

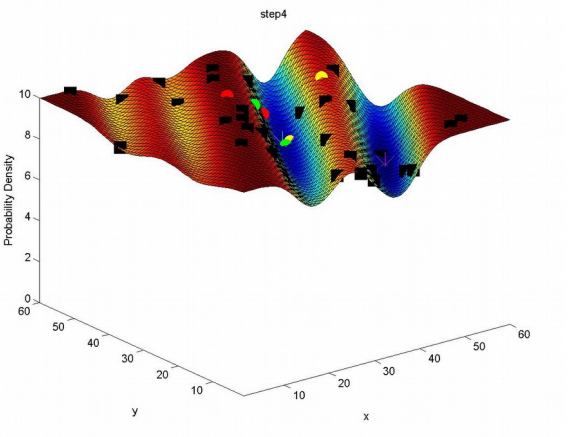
Ξεκινήσαμε την εκτέλεση με 3 swarms καθένα εκ των οποίων έχει 3 particles, και η κατανομή τους στον χώρο αν και τυχαία, είναι ομοιόμορφη. Καθώς ο χρόνος κυλάει τα βλέπουμε να κινούνται προς το σημείο στο οποίο εντοπίστηκε το γενικό ελάχιστο και κατά την κίνηση αυτή επαναπροσδιορίζεται η θέση του γενικού ελαχίστου μέχρις ότου αυτό να ταυτιστεί με το (πραγματικό) ολικό ελάχιστο. Η σύγκλιση των δύο σημείων επήλθε μετά από 13 επαναλήψεις του RDPSO αλγορίθμου.

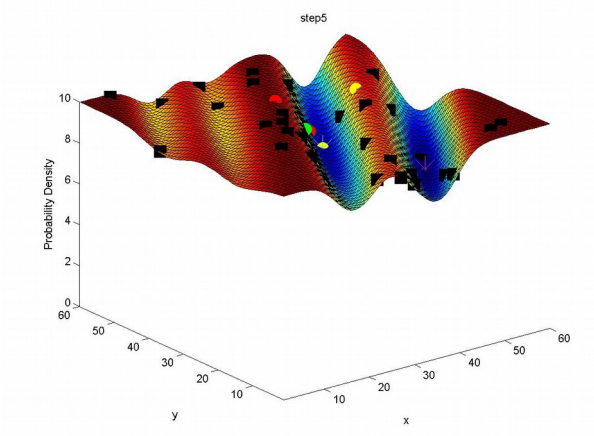
Εκτέλεση 2η:

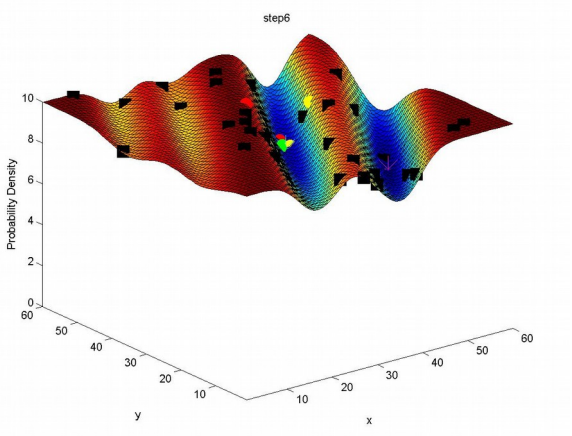

step1
Probability Density
y
x

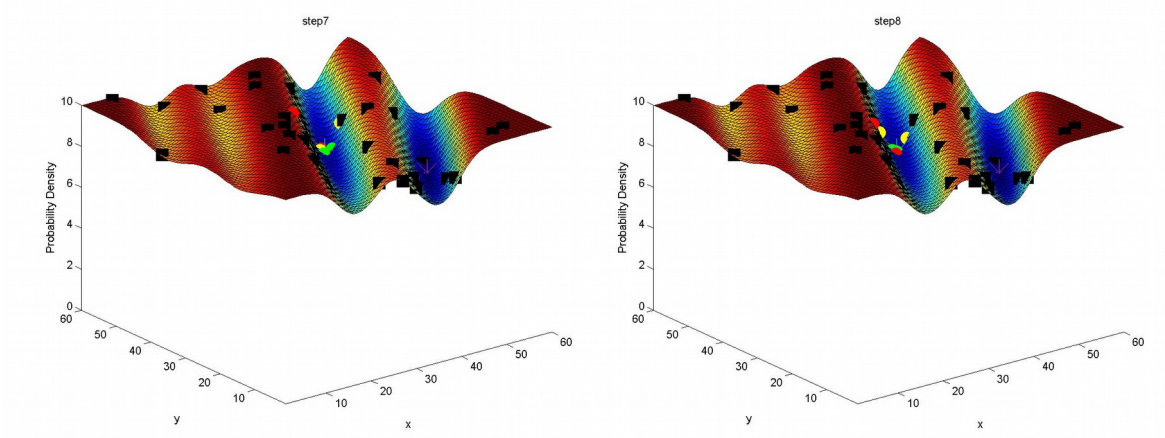

step2
Probability Density
y
x

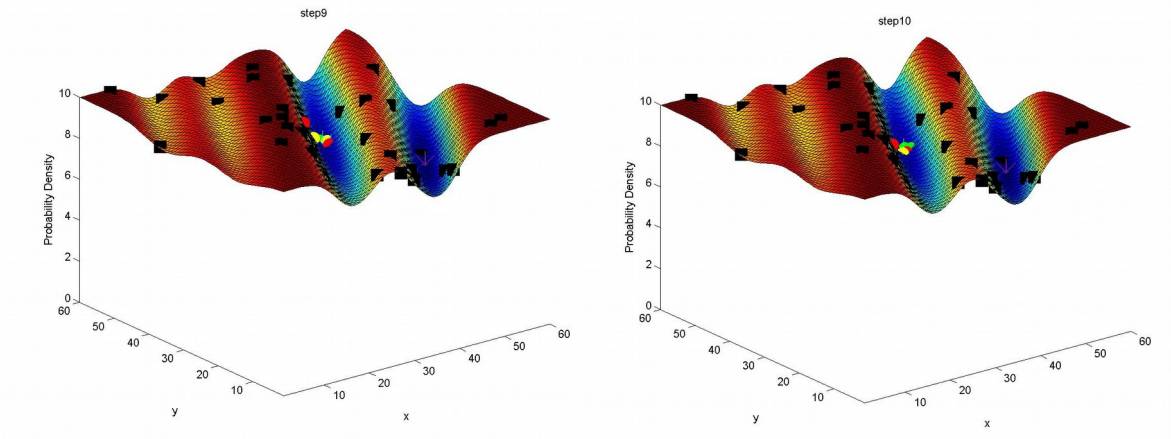

step3
Probability Density
y
x

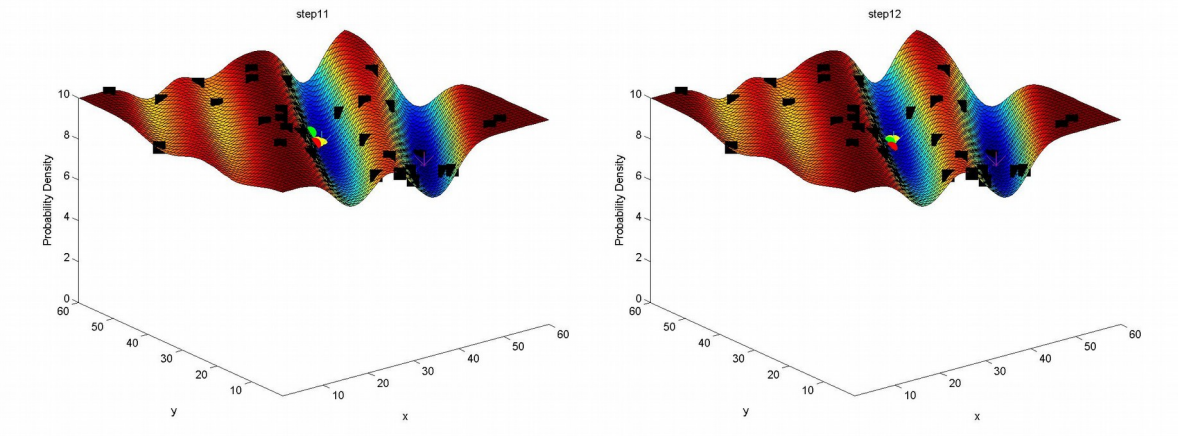

step4
Probability Density
y
x


step5
Probability Density
y
x


step6
Probability Density
y
x

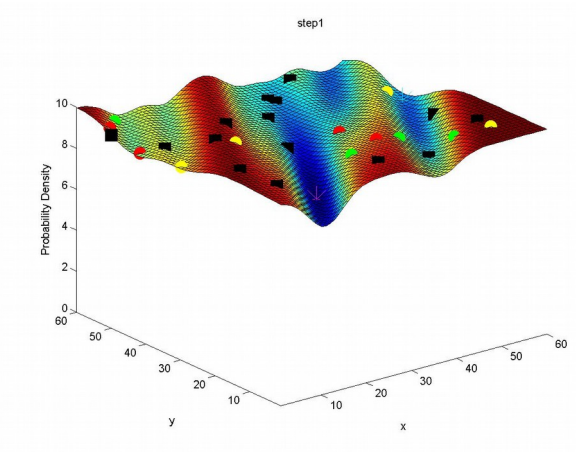

step7
step8
Probability Density
x
y

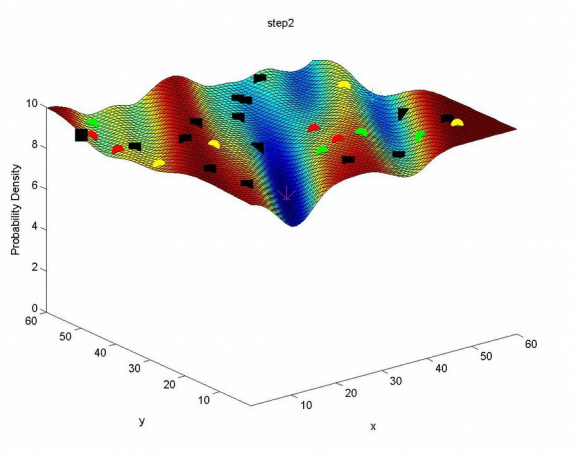

step9
step10
Probability Density
x
y

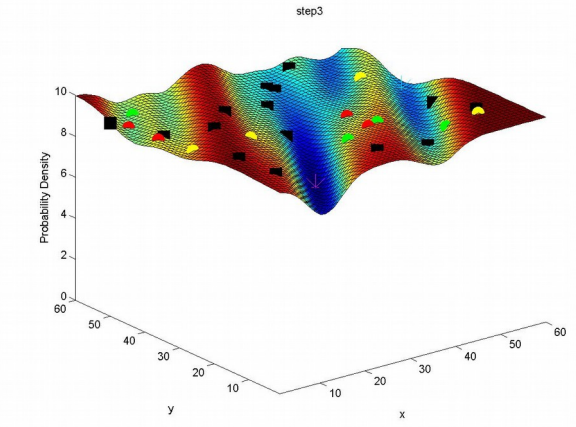

step11
step12
Probability Density
x
y

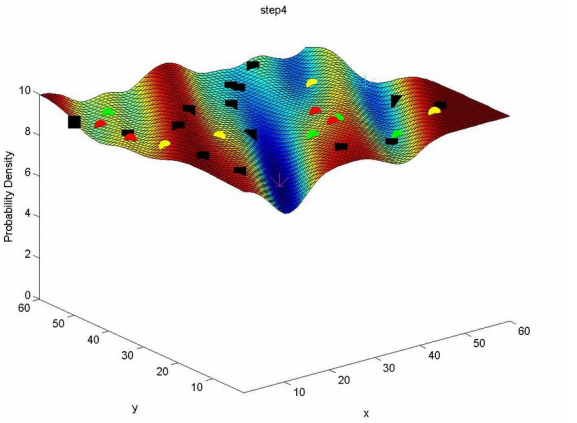
Στην δεύτερη εκτέλεση χρησιμοποιήσαμε 3 swarms των 2 particles το καθένα. Ο χώρος είναι έντονα “ανάγλυφος” ως προς τη συνάρτηση κόστους, έχει δηλαδή αρκετά τοπικά μέγιστα και ελάχιστα. Ο αλγόριθμός μας δεν έχει πάντοτε άμυνες απέναντι σε τέτοια φαινόμενα, όπως φαίνεται και από την παραπάνω εκτέλεσή του. Έτσι, λόγω του ότι τα paricles δεν τοποθετήθηκαν αρχικά αρκετά “απλωμένα” στον χώρο, δεν καταφέρνουμε να εντοπίσουμε τη θέση ολικού ελαχίστου, παρά μόνο ένα τοπικό. Αν αυξήσουμε τον αριθμό των συνολικών particles στον χώρο τότε οι πιθανότητες να συμβεί αυτό μειώνονται. Εδώ ωστόσο, επίτηδες χρησιμοποιήσαμε λιγότερα προκειμένου να καταδειχθεί ο κίνδυνος αυτός.

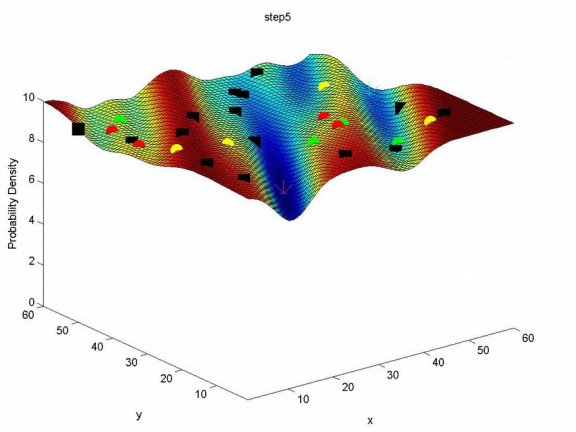
Εκτέλεση 3η:

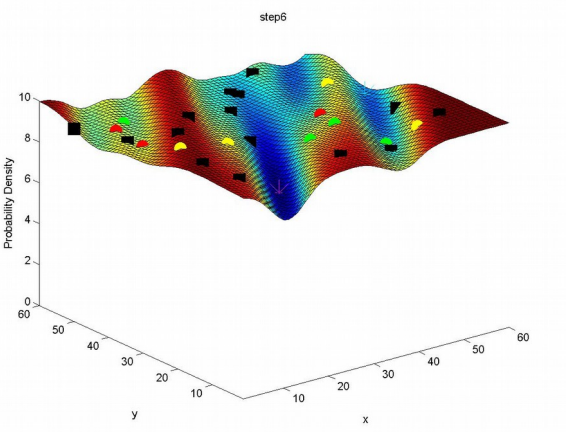

step1
Probability Density
y
x

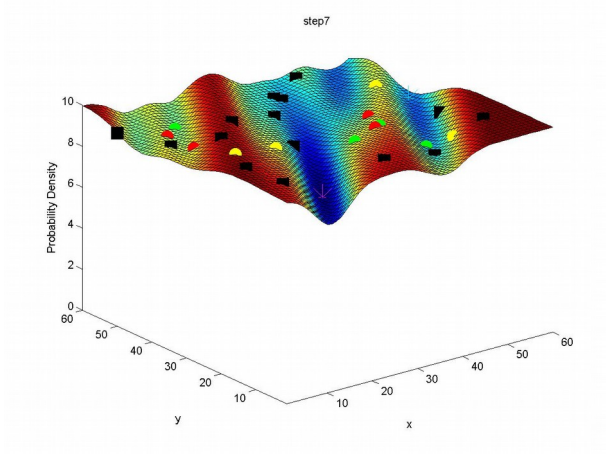

step2
Probability Density
y
x

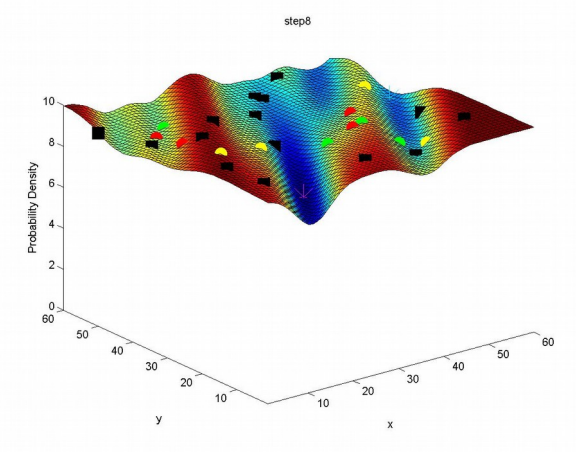

step3
Probability Density
y
x

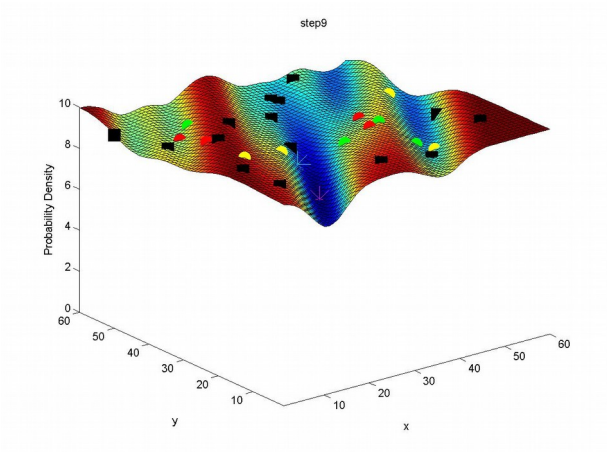

step4
Probability Density
y
x

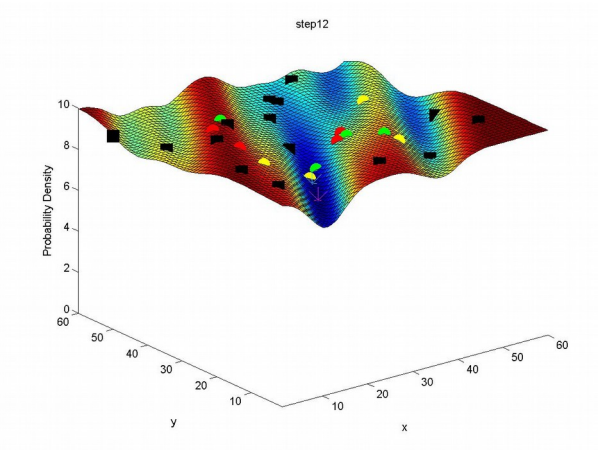

step5
Probability Density
y
x

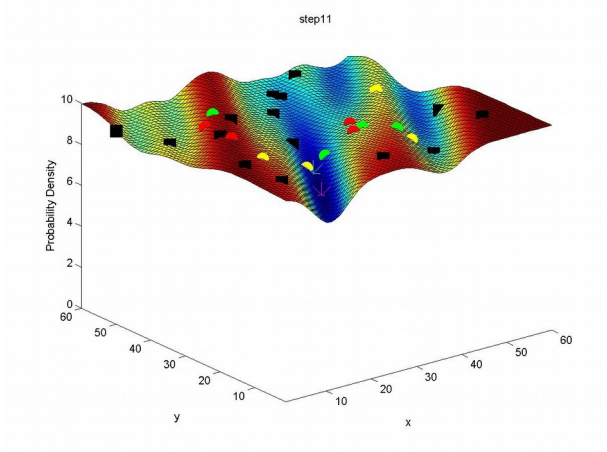

step6
Probability Density
y
x

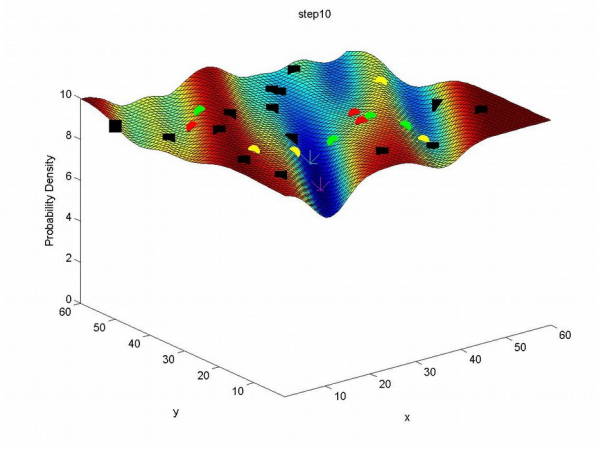

step7
Probability Density
y
x

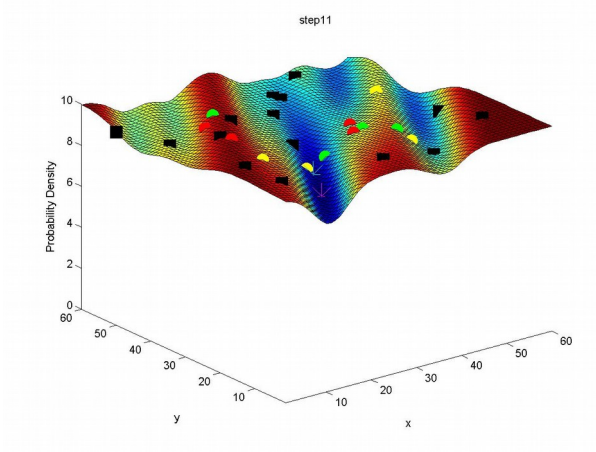

step8
Probability Density
y
x

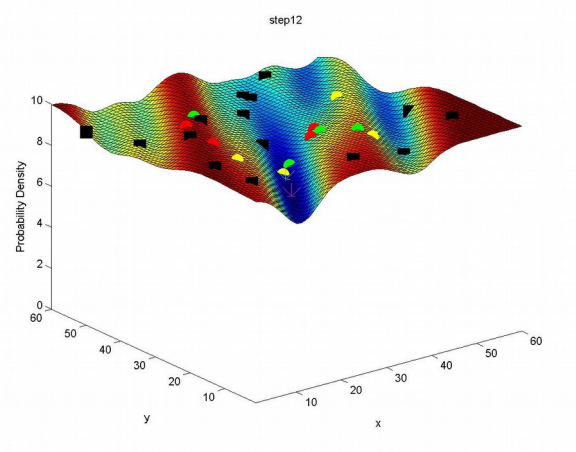

step9
Probability Density
y
x

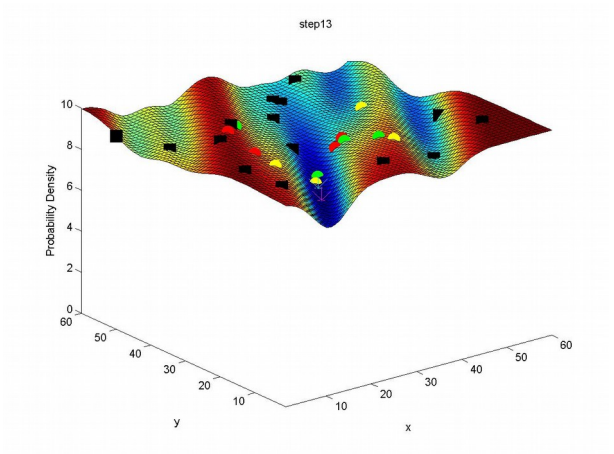

step12
Probability Density
y
x

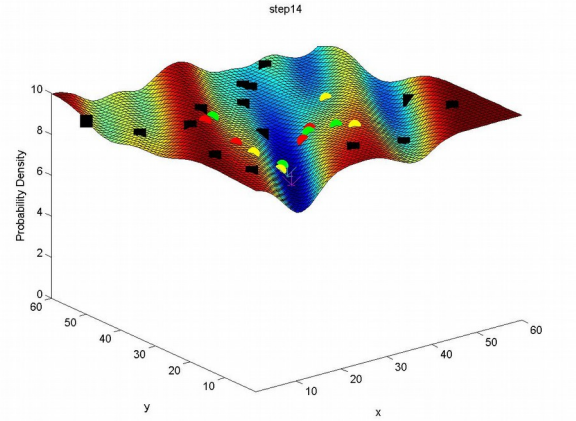

step11
Probability Density
y
x

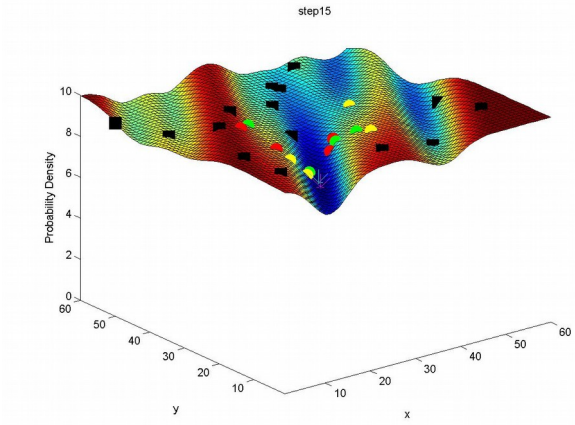

step10
Probability Density
y
x

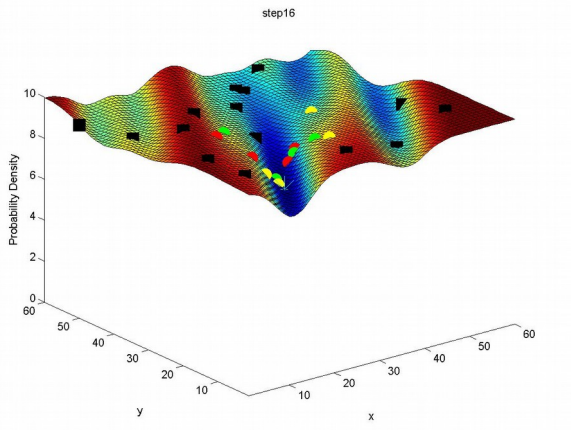

step11
Probability Density
y
x

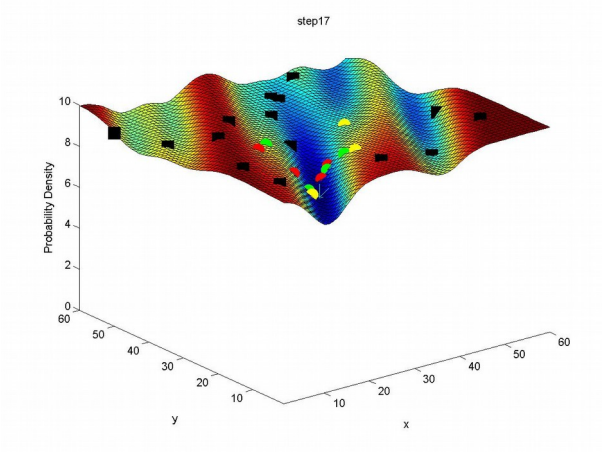

step12
Probability Density
y
x

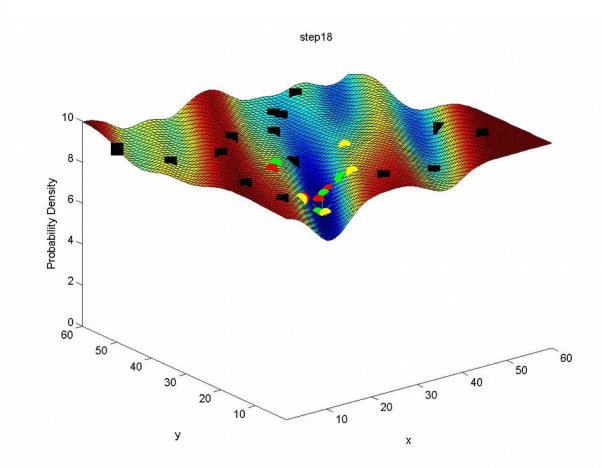

step13
Probability Density
y
x


step14
Probability Density
y
x


step15
Probability Density
y
x


step16
Probability Density
y
x



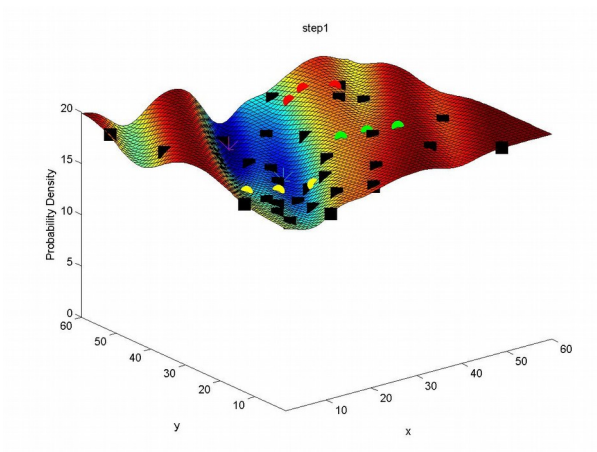


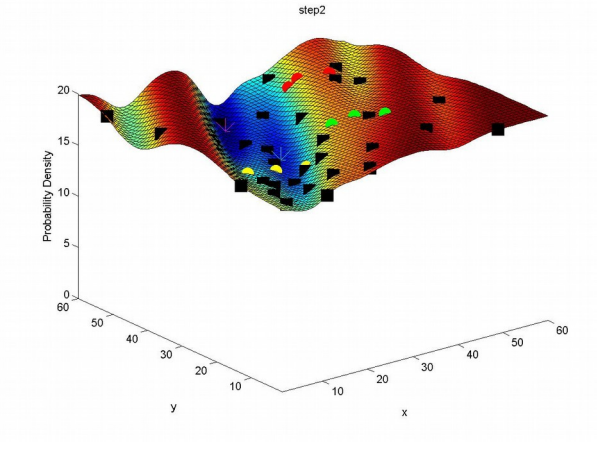
Στην εκτέλεση αυτή βλέπουμε ότι το σύστημά μας κατορθώνει να υπερπηδήσει τον σκόπελο των πολλών τοπικών ελαχίστων, και να εντοπίσει εν τέλει το ολικό. Αυτό επιτεύχθη με τρία swarms των τεσσάρων particles έκαστο, και σε χρόνο 16 επαναλήψεων.

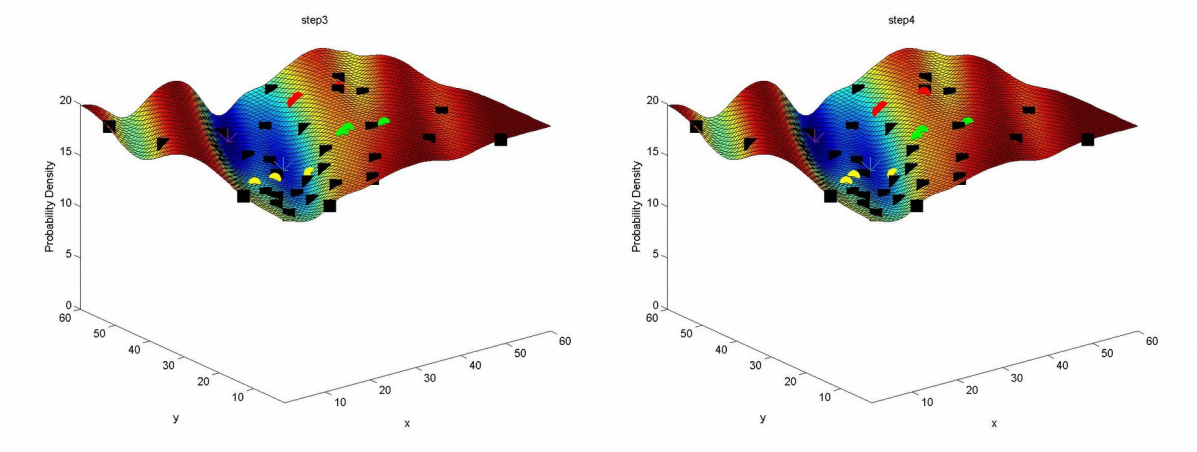
7.2 Αποτελέσματα του RDPSO αλγορίθμου (με διατήρηση του ad hoc connectivity)

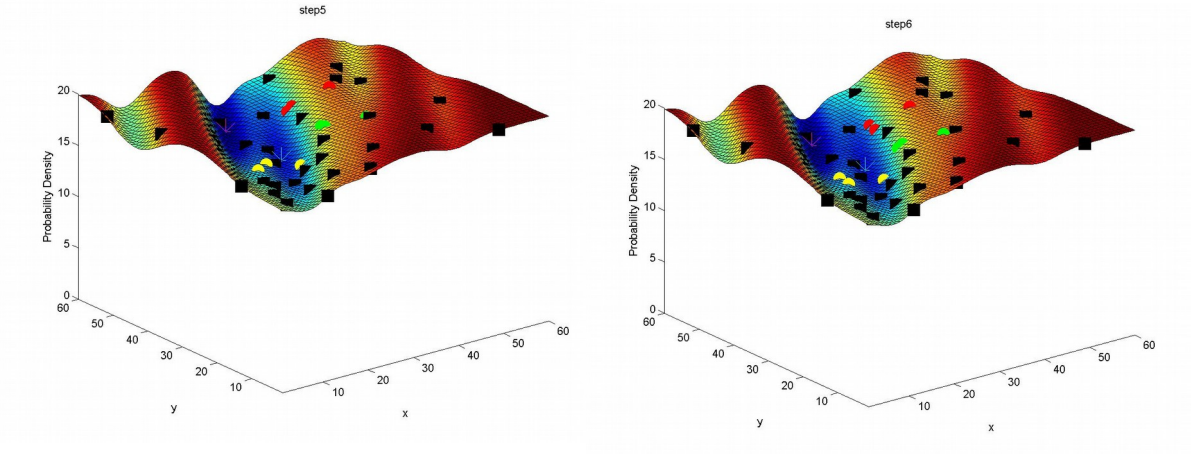
Εκτελούμε τον κώδικα σύμφωνα με όσα έχουν περιγραφεί στα προηγούμενα κεφάλαια και λαμβάνουμε τα ακόλουθα αποτελέσματα.

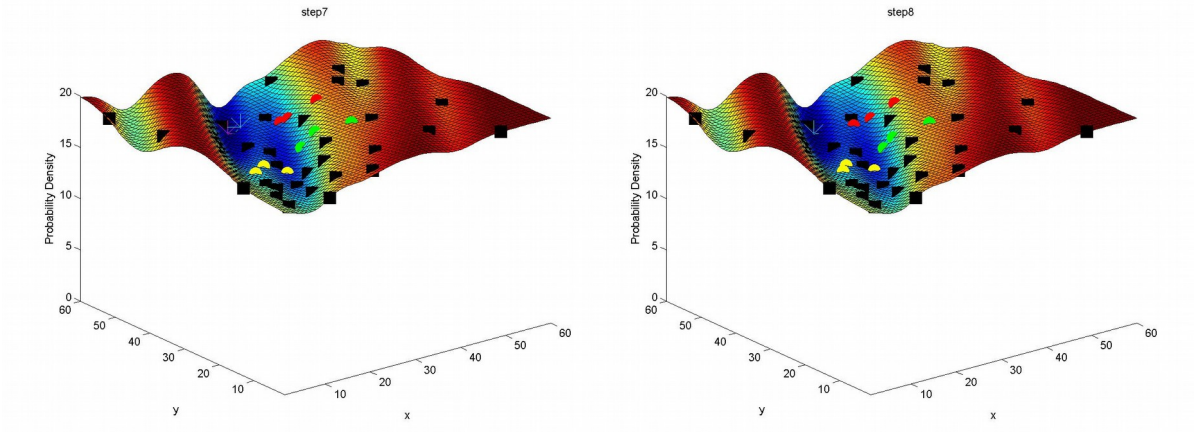
Εκτέλεση 1η:





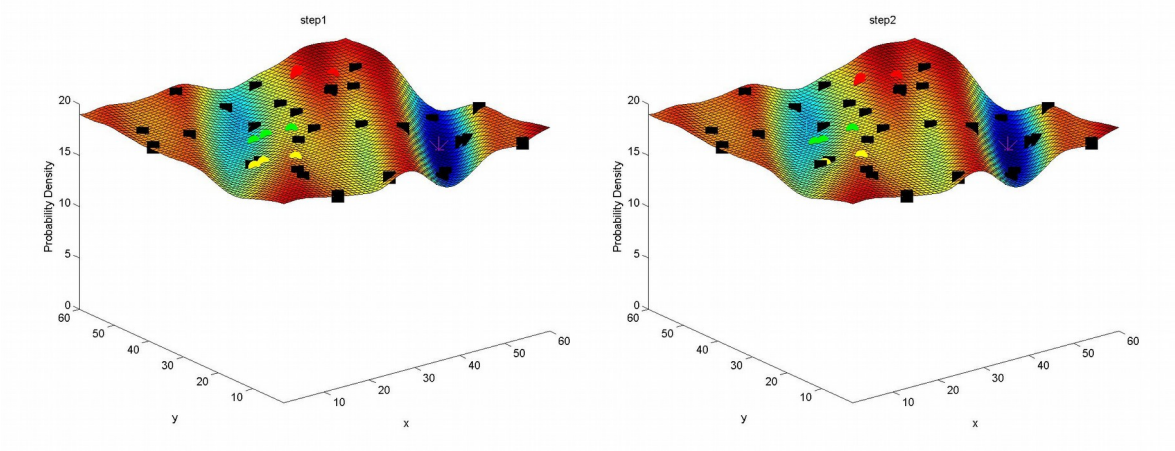


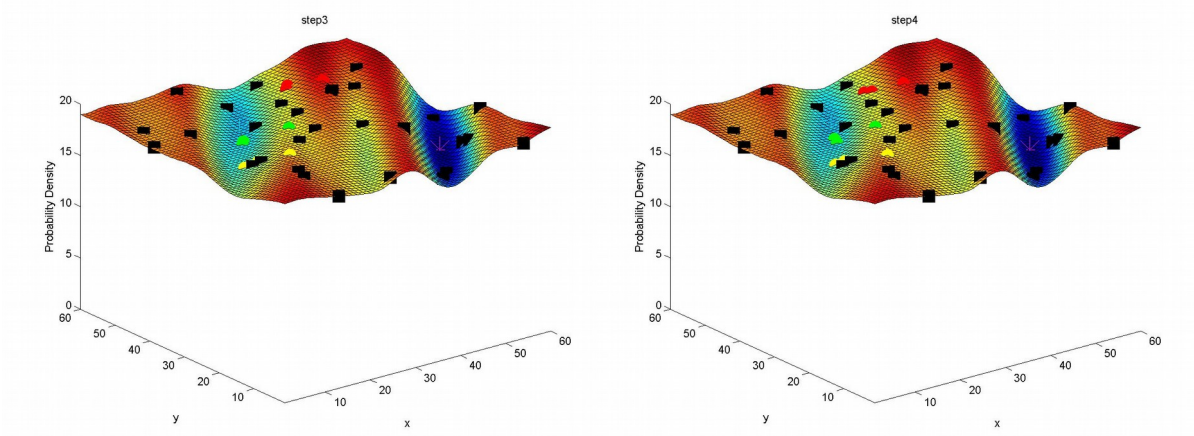


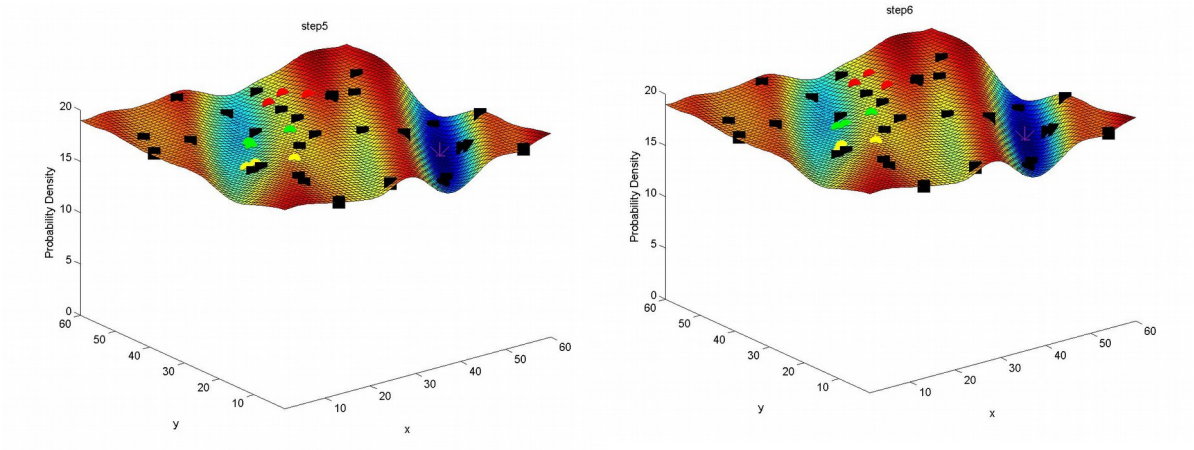


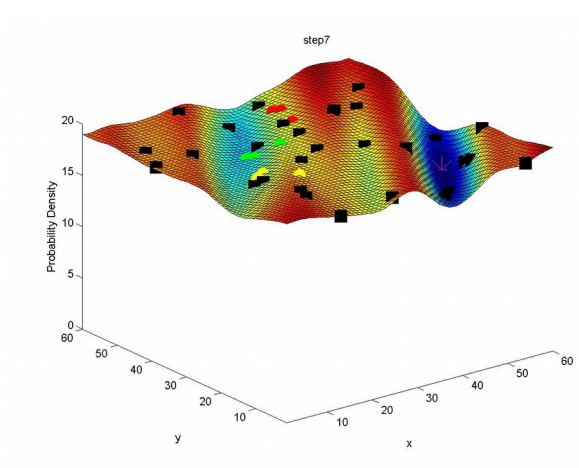
Εδώ παρατηρούμε ότι το σύστημά μας καταφέρνει να συγκλίνει στο ολικό ελάχιστο επιτυχώς και αρκετά γρήγορα (εντός 6 κύκλων), χρησιμοποιώντας 3 swarms με 3 particles το καθένα.

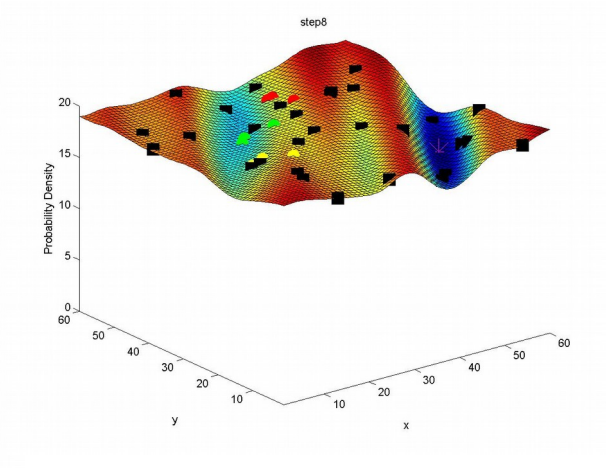
Εκτέλεση 2η:


step1
step2
Probability Density
x
y


step3
step4
Probability Density
x
y

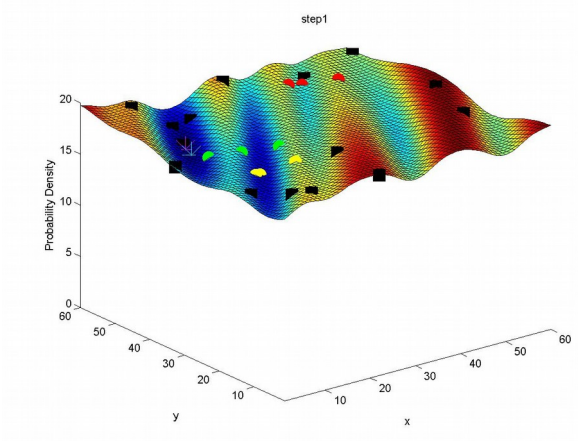

step5
step6
Probability Density
x
y

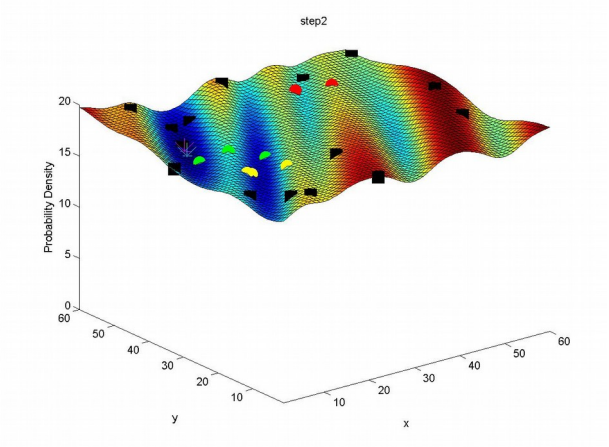


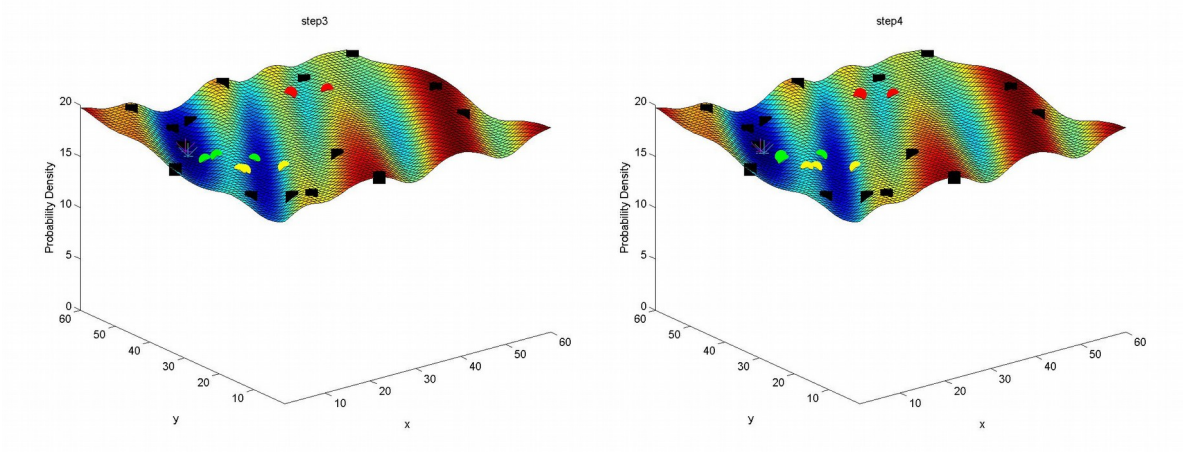


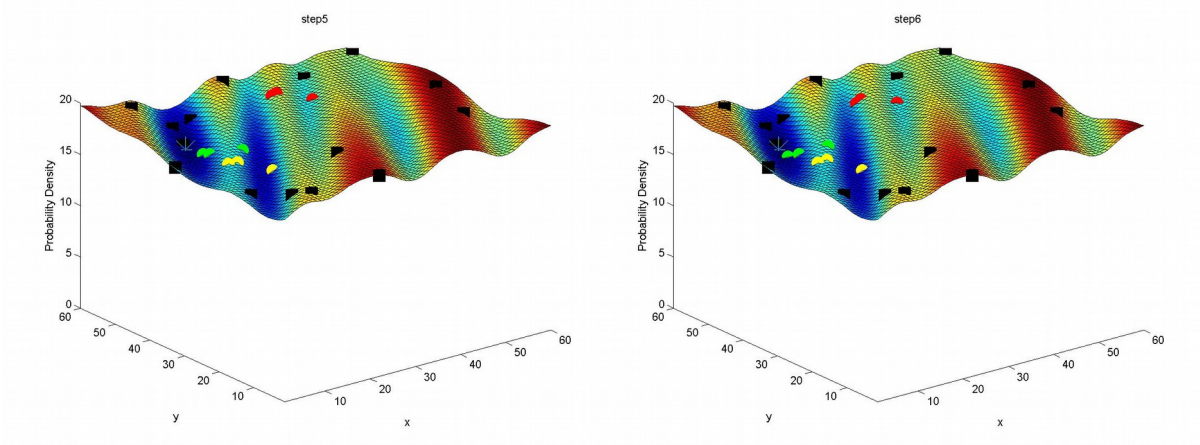
Στην δεύτερη αυτή εκτέλεση του αλγορίθμου βλέπουμε ότι το σύστημα είναι επιρρεπές στο να “κολλάει” σε τοπικά ελάχιστα, όταν τα particles δεν είναι αρκετά απλωμένα στον χώρο. Όσο και αν το αφήναμε να τρέξει δεν θα κατόρθωνε ποτέ υπό αυτές τις συνθήκες να εντοπίσει το πραγματικό ολικό ελάχιστο.

Εκτέλεση 3η:









Εδώ η συγκυρία της κατανομής των particles στον χώρο είναι ευνοϊκή και παρατηρούμε ταχύτατη σύγκλιση στο ολικό ελάχιστο, με 3 swarms απο 3 particles το καθένα.

ΣΥΜΠΕΡΑΣΜΑΤΑ:

Από τις παραπάνω εκτελέσεις του αλγορίθμου (και στις δύο εκδοχές του) παρατηρούμε ότι σε μεγάλο ποσοστό συγκλίνει επιτυχώς. Οι περπτώσεις που αποτυγχάνει οφείλονται στην στοχαστικότητα κατά τη δημιουργία τόσο κατά τη δημιουργία του χώρου όσο και κατά την αρχική τοποθέτηση των particles. Η απόδοσή του μπορεί να θεωρηθεί πολύ ικανοποιητική, αν και μπορούν να γίνουν πολλαπλές και πολύπλευρες βελτιώσεις και προσθήκες ώστε να ανταποκρίνεται με μεγαλύτερο ρεαλισμό σε πραγματικες συνθήκες και παράλληλα να ξεπερνιούνται και κάποιες από τις δυσκολίες που προαναφέρθηκαν.

8 Παράρτημα

8.1 Κώδικας για RPDSO χωρίς να διατηρείται το ad hoc connectivity

1. add_particle_from_excluded

```
global fig excluded

% we move all the particle data from the socially excluded group to the
% swarm s

p = size(swarms{s}.positions,1)+1;

swarms{s}.positions = [swarms{s}.positions; excluded.positions(best_particle,:)];
swarms{s}.cost_func = [swarms{s}.cost_func; excluded.cost_func(best_particle,:)];
swarms{s}.x_opt = [swarms{s}.x_opt; excluded.x_opt(best_particle,:)];
swarms{s}.velocity = [swarms{s}.velocity; excluded.velocities(best_particle,:)];
swarms{s}.xg_opt = [swarms{s}.xg_opt; excluded.xg_opt(best_particle,:)];
swarms{s}.improvement = [swarms{s}.improvement; excluded.improvement(best_particle,:)];
```

```
% then we remove the particle (and all its info) from the swarm

excluded.positions = removerows(excluded.positions, 'ind', best_particle);
excluded.cost_func = removerows(excluded.cost_func, 'ind', best_particle);
excluded.x_opt = removerows(excluded.x_opt, 'ind', best_particle);
excluded.velocities = removerows(excluded.velocities, 'ind', best_particle);
excluded.xg_opt = removerows(excluded.xg_opt, 'ind', best_particle);
excluded.improvement = removerows(excluded.improvement, 'ind', best_particle);
```

2. calc_opt_pos_sensing_function

```
function [g_min_pos] = calc_opt_pos_sensing_func(swarms, obstacles, s, n, rs)

  global L
%    s is the swarm whose n-th particle is to be examined
  x = swarms{s}.positions(n,:);

  rc = 3; %rc is the radius from x within which we look for the optimum point

  k=0;
  visible_obst = zeros(1,2);
  for i=1:size(obstacles,1)
    d = sqrt(sum((x-obstacles(i,:)).^2));
    if (d<=rs)
      k=k+1;
      visible_obst(k,:) = obstacles(i,:);
    end
  end

  for i=1:size(swarms,1)
    for j=1:size(swarms{i},1)
      d = sqrt(sum((x-swarms{i}.positions(j,:).^2)));
      if (d<=rs && d>0)
        k=k+1;
        visible_obst(k,:) = swarms{i}.positions(j,:);
      end
    end
  end

  foo1x = x(:,1) - 2*rc;
  foo2x = x(:,1) + 2*rc;
  foo1y = x(:,2) - 2*rc;
  foo2y = x(:,2) + 2*rc;
  if (foo1x < 0)
    foo1x = 0;
```

```
    end
    if (foo2x>L)
        foo2x = L;
    end
    if (foo1y < 0)
        foo1y = 0;
    end
    if (foo2y>L)
        foo2y = L;
    end

    [X, Y] = meshgrid(foo1x:foo2x, foo1y:foo2y);
    exp_space = [X(:) Y(:)];
    x_exp = rangesearch(exp_space, x, rc); %rangesearch returns a 1x1 cell
    x_exp = x_exp{1};

    g = zeros(size(x_exp,1),1);
    for i=1:size(x_exp,1)
        x = exp_space(x_exp(i),:);
        for j=1:size(visible_obst,1)
            d = sqrt(sum((x-visible_obst(j,:)).^2));
            g(i) = g(i) + 1/d;
        end
    end


    g_min = find(g==min(g));
    g_min_pos = exp_space(x_exp(g_min),:);

end
```

3. calc_opt_pos_sensing_function_excl

```
function [g_min_pos] = calc_opt_pos_sensing_func_excl(excluded, obstacles, n, rs)

    global L
%     s is the swarm whose n-th particle is to be examined
    x = excluded.positions(n,:);

    rc = 3; %rc is the radius from x within which we look for the optimum point

    k=0;
    visible_obst = zeros(1,2);
    for i=1:size(obstacles,1)
        d = sqrt(sum((x-obstacles(i,:)).^2));
        if (d<=rs)
```

```
      k=k+1;
      visible_obst(k,:) = obstacles(i,:);
   end
end


for j=1:size(excluded,1)
   d = sqrt(sum((x-excluded.positions(j,:).^2)));
   if (d<=rs && d>0)
      k=k+1;
      visible_obst(k,:) = excluded.positions(j,:);
   end
end

foo1x = x(:,1) - 2*rc;
foo2x = x(:,1) + 2*rc;
foo1y = x(:,2) - 2*rc;
foo2y = x(:,2) + 2*rc;
if (foo1x < 0)
   foo1x = 0;
end
if (foo2x>L)
   foo2x = L;
end
if (foo1y < 0)
   foo1y = 0;
end
if (foo2y>L)
   foo2y = L;
end

[X, Y] = meshgrid(foo1x:foo2x, foo1y:foo2y);
exp_space = [X(:) Y(:)];
x_exp = rangesearch(exp_space, x, rc); %rangesearch returns a 1x1 cell
x_exp = x_exp{1};

g = zeros(size(x_exp,1),1);
for i=1:size(x_exp,1)
   x = exp_space(x_exp(i),:);
   for j=1:size(visible_obst,1)
      d = sqrt(sum((x-visible_obst(j,:)).^2));
      g(i) = g(i) + 1/d;
   end
end
```

```
   g_min = find(g==min(g));
   g_min_pos = exp_space(x_exp(g_min),:);

end
```

4. create_space

```
global fig vis temp_vis L cost_func global_minimum

num_Gaussians = 10; %number of Gaussians involved

% The space will be a rectangle with L-1 meters length on each side, from 1
% to L
L=60;
cost_func = zeros(L); %initializations

for i=1:num_Gaussians
   m = [rand_num(1,L) rand_num(1,L)];

   Sigma = [L/6 L/5; L/3 L/4];
%    Sigma = rand_num(1,L,2,2); %we can use this definition for a random
%    %variance matrix, but the space will likely have no local optima
   Sigma = Sigma*Sigma';

   x = 1:L; y = 1:L;
   [X,Y] = meshgrid(x,y);

   F = mvnpdf([X(:) Y(:)],m,Sigma);
   F = reshape(F,length(x),length(y))/max(max(F));
   cost_func = cost_func + F;
end

cost_func = num_Gaussians - cost_func;

% the global minimum and the position in which it occurs
[global_minimum, idx]=min(cost_func(:));
[min_row,min_col]=ind2sub(size(cost_func),idx);


observations = ones(size(cost_func))*NaN;

if (fig==1)
   figure('visible', vis);
   vis = temp_vis;
   surf(cost_func);
```

```
   axis([1 L 1 L 0 num_Gaussians])
   xlabel('x'); ylabel('y'); zlabel('Probability Density');

   figure('visible', vis);
   vis = temp_vis;
   imagesc(x,y,cost_func);
   axis([1 L 1 L])
   xlabel('x'); ylabel('y');

end;


%%% create the obstacles %%%

obstacles = randi(L-1,No,2);

if fig==1
   figure('visible', vis);
   vis = temp_vis;
   plot(obstacles(:,1),obstacles(:,2),'ks', 'markerfacecolor', 'k');
   axis([0 L 0 L])

end;
```

5. create_swarms

```
% First I create the particles (the particle matrix), and place them randomly
% within the search space. Then I pick Si particles (the first Si elements
% of the particle matrix, where Si is the number of the swarms).
% The remaining particles are assigned to the swarm nearest to hem - where
% the swarm is represented by its first particle.

global fig vis temp_vis excluded


swarms = cell(Si,1);

for s=1:Si
   swarms{s}.positions = randi(L-1,Ni,2);
   swarms{s}.velocity = round(rand_num(-1,1,Ni,2));
   swarms{s}.x_opt = swarms{s}.positions;
```

```
   for n=1:size(swarms{s}.positions,1)
      temp(n,:) = calc_opt_pos_sensing_func(swarms, obstacles, s, n, rs);
   end
   swarms{s}.xg_opt = zeros(Si,2);
   swarms{s}.xg_opt = temp;

end


gn = [0,0];
gn_cost = 1000;

for i=1:size(swarms,1)
   s = size(swarms{i}.positions,1);
   for j=1:s
      swarms{i}.cost_func(j) = cost_func(swarms{i}.positions(j,2),swarms{i}.positions(j,1));
      swarms{i}.improvement(j) = -swarms{i}.cost_func(j);
      if (swarms{i}.cost_func(j)<gn_cost)
         gn_cost = swarms{i}.cost_func(j);
         gn = swarms{i}.positions(j,:);
      end
   end
   swarms{i}.cost_func = swarms{i}.cost_func';
end

condition_1 = eps;
condition_2 = gn_cost;
% condition_1 contains the previous value of the global optimum (minimum)
% and condition_2 the second the current one.
% Subtraction of the two leads to the necessary condition to terminate the
% loop in the main_rdpso.m script. We initialize it with the values eps
% and gn, where gn is the current value of the global optimum.


excluded.positions = []; %we only define the socially excluded group,
%without assigning to it any initial values (particles) since all the
%particles belong to a swarm at first

excluded.x_opt = [];
excluded.velocities = [];
excluded.xg_opt = [];
excluded.cost_func = [];
excluded.improvement = [];

observations = update_observations(observations,swarms, excluded, r);
```

```
%%%%%%%%%%%%%%%%%%%%%%%

cost_obst_fig = zeros(No,1);
for i=1:No
   cost_obst_fig(i) = cost_func(obstacles(i,2),obstacles(i,1));
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot of the search space cost, along with obstacles and particles for Si=3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if fig==1
   figure('visible', vis);
   vis = temp_vis;
   for s = 1:size(swarms,1)
         plot(swarms{s}.positions(:,1),swarms{s}.positions(:,2),'o',   'Color',   color_matrix(s,:),
'markerfacecolor', color_matrix(s,:));
      hold on;
   end
   plot(obstacles(:,1),obstacles(:,2),'ks', 'markerfacecolor', 'k');
   axis([0 L 0 L])


   figure('visible', vis);
   vis = temp_vis;
   for s=1:size(swarms,1)
         plot3(swarms{s}.positions(:,1),swarms{s}.positions(:,2),swarms{s}.cost_func,'o',  'Color',
color_matrix(s,:), 'markerfacecolor', color_matrix(s,:), 'MarkerSize', 12);
         hold on;
   end
   plot3(obstacles(:,1),obstacles(:,2),cost_obst_fig,'ks', 'markerfacecolor', 'k', 'MarkerSize', 15);
   plot3(min_col, min_row, global_minimum, 'm*', 'markerfacecolor', 'm', 'MarkerSize', 25);
   plot3(gn(1),gn(2),gn_cost, 'c*', 'markerfacecolor', 'c', 'MarkerSize', 25);
   if size(excluded.positions,1)>0
      plot3(excluded.positions(:,1), excluded.positions(:,2), excluded.cost_func, 'ko', 'markerfacecolor',
'w', 'MarkerSize', 12);
   end
   surf(cost_func);
   axis([1 L 1 L 0 num_Gaussians]);
   xlabel('x'); ylabel('y'); zlabel('Probability Density');


end;
```

<u>6. delete_swarm</u>

```
global fig excluded

% we move all the particle data to the socially excluded group

p = size(excluded.positions,1)+1;
q = size(swarms{s}.positions,1);
excluded.positions(p:p+q-1,:) = swarms{s}.positions(:,:);
excluded.cost_func(p:p+q-1,:) = swarms{s}.cost_func(:,:);
excluded.x_opt(p:p+q-1,:) = swarms{s}.x_opt(:,:);
excluded.velocities(p:p+q-1,:) = swarms{s}.velocity(:,:);
excluded.xg_opt(p:p+q-1,:) = swarms{s}.xg_opt(:,:);
excluded.improvement(p:p+q-1,:) = swarms{s}.improvement(:,:);


% then we remove the particle (and all its info) from the swarm

swarms(s,:) = [];
```

7. evolve_excluded

```
function [excluded] = evolve_excluded (excluded)

  global L cost_func


  for n = 1:size(excluded.positions,1)
    new_vel = rand(1,2);
    if (norm(new_vel)~=0)
      new_vel = new_vel./norm(new_vel);
    end

    excluded.velocity(n,:) = new_vel;
    excluded.positions(n,:) = floor(excluded.positions(n,:) + 4*excluded.velocity(n,:));
    if excluded.positions(n,1) == 0
      excluded.positions(n,1) = 1;
    end
    if excluded.positions(n,2) == 0
      excluded.positions(n,2) = 1;
    end
    if excluded.positions(n,1) > L
      excluded.positions(n,1) = L;
    end
    if excluded.positions(n,2) > L
      excluded.positions(n,2) = L;
    end
```

```
    excluded.cost_func(n) = cost_func(excluded.positions(n,2),excluded.positions(n,1));
    excluded.improvement(n) = excluded.improvement(n) + excluded.cost_func(n);


    excluded.improvement = -excluded.cost_func;

  end

end
```

8. evolve_swarm

```
function [swarms, SC, N_kill, next_swarm] = evolve_swarm (swarms, s, gn, SC, N_kill)

  global L cost_func SCmax Ni Nmin Nmax Smax excluded


 next_swarm = s+1;
  for n=1:size(swarms{s}.positions,1)
    xn = swarms{s}.positions(n,:);
    x_opt = swarms{s}.x_opt(n,:);
    xng_opt = swarms{s}.xg_opt(n,:);
    vel = swarms{s}.velocity(n,:);
    swarms{s}.velocity(n,:) = update_velocity(xn, x_opt, xng_opt, vel, gn);
    swarms{s}.positions(n,:) = floor(swarms{s}.positions(n,:) + 2*swarms{s}.velocity(n,:));
    if swarms{s}.positions(n,1) == 0
      swarms{s}.positions(n,1) = 1;
    end
    if swarms{s}.positions(n,2) == 0
      swarms{s}.positions(n,2) = 1;
    end
    if swarms{s}.positions(n,1) > L
      swarms{s}.positions(n,1) = L;
    end
    if swarms{s}.positions(n,2) > L
      swarms{s}.positions(n,2) = L;
    end
    swarms{s}.cost_func(n) = cost_func(swarms{s}.positions(n,2),swarms{s}.positions(n,1));
    swarms{s}.improvement(n) = swarms{s}.improvement(n) + swarms{s}.cost_func(n);
  end

  improvement = sum(swarms{s}.improvement);

%    swarms_improvement{s}
```

```
  if (improvement>0)
    if (size(excluded.positions,1)~=0 && size(swarms{s}.positions,1)<Nmax)
%         if the socially excluded group is not empty add to the swarm the
%         best performing robot
      best_particle = find(excluded.cost_func==min(excluded.cost_func));
      add_particle_from_excluded
    end
  else
    SC(s) = SC(s) + 1;
    if SC(s)>SCmax
      if (size(swarms{s}.positions,1)>Nmin)
        if (improvement ~= 0)
          worst_particle = find(swarms{s}.improvement == min(swarms{s}.improvement));
        else
          worst_particle =  round(rand_num(1,size(swarms{s},1)));
        end
  %         exclude worst performing robot
        exclude_particle
        clear worst_particle
        N_kill(s) = N_kill(s) + 1;
        SC(s) = SCmax * (1 - 1/(N_kill(s)+1));
        swarms{s}.improvement = -swarms{s}.cost_func;
      else
        f = rand(1);
        prob = 0.75*rand(1)*size(swarms,1)/Smax;
        if f<=prob
          disp('delete');
          next_swarm = s;
          delete_swarm
        end
      end
    end
  end

  if (N_kill(s)==0 && size(swarms{s}.positions,1)<Smax && size(excluded.positions,1)>=Ni)
%       we compute the probability prob of spawning a new swarm
    f = rand(1);
    prob = f/size(swarms{s}.positions,1);
    if rand(1)<=prob
      disp('spawn');
      spawn_swarm
    end
  end

end
```

9. exclude_particle

```
global fig excluded

% we move all the particle data to the socially excluded group

p = size(excluded.positions,1)+1;

excluded.positions(p,:) = swarms{s}.positions(worst_particle,:);
excluded.cost_func(p,:) = swarms{s}.cost_func(worst_particle,:);
excluded.x_opt(p,:) = swarms{s}.x_opt(worst_particle,:);
excluded.velocities(p,:) = swarms{s}.velocity(worst_particle,:);
excluded.xg_opt(p,:) = swarms{s}.xg_opt(worst_particle,:);
excluded.improvement(p,:) = swarms{s}.improvement(worst_particle,:);


% then we remove the particle (and all its info) from the swarm

swarms{s}.positions = removerows(swarms{s}.positions, 'ind', worst_particle);
swarms{s}.cost_func = removerows(swarms{s}.cost_func, 'ind', worst_particle);
swarms{s}.x_opt = removerows(swarms{s}.x_opt, 'ind', worst_particle);
swarms{s}.velocity = removerows(swarms{s}.velocity, 'ind', worst_particle);
swarms{s}.xg_opt = removerows(swarms{s}.xg_opt, 'ind', worst_particle);
swarms{s}.improvement = removerows(swarms{s}.improvement, 'ind', worst_particle);
```

10. main

```
close all
clear all

global fig vis temp_vis Ni Nmin Nmax SCmax Si Smin Smax vmin vmax

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% initialization of important parameters %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fig = 1; % if fig is 1 then the various plots at the different stages of
% the algorithm will be computed if fig is 0 then the plots remain unseen

vis = 'on'; % (for vis to be of any significance, fig must be 1) if vis is
% on, the plots computed along the way are visible to the user, else (if it
% is off, they remain invisible

temp_vis = vis; %every time vis is used in a figure its value is
```

```
% automatically switched to 'on', so we keep the desired value in temp_vis
% so vis will be set back to it each time (some plots are visible
% regardless of vis's value, in order for the final avi movie to be
% produced)

Ni = 4; %initial number of particles (robots) per swarm
Nmin = 2; %minimum number of particles per swarm
Nmax = 6; %maximum number of particles per swarm

Si = 3; %initial number of swarms
Smin = 1; %minimum number of swarms
Smax = 5; %maximum number of swarms

Np = Ni*Si; %total number of particles

SCmax = 10; %stagnancy threshold <-------need to figure out the parameter's value
SC = zeros(Si,1); %we initialize the search counter for each swarm with zero

N_kill = zeros(Si,1); %we initialize the number of particles that have been
%excluded from each swarm

No = 15; %number of obstacles

rs = 5; %sensing radius (depends on the sensors' type and range)

r = 2; %observation radius

ev_step = 5; %evaluation step

color_matrix = [1 0 0; 0 1 0; 1 1 0; 0.2 0 0; 0.5 0.5 0.5; 1 1 0.5];



%% end of initializations %%

create_space

create_swarms

% swarms are placed randomly in the experimentation space



% For each particle calculate the new speed and position, update the swarms
% matrix, calculate the Nkill(i) for each swarm, i, and redefine the respective
% search counter SC(i). If the critical threshold SCmax is exceeded, then we
```

```
% update SC(i) to SCmax*(1-1/(NKill(i)+1)) and the worst performing robot
% is excluded and moved to the outcast particles' group. If the swarm
% improves its objective function then the best performing robot of the
% outcast group is transferred to this swarm.

main_rdpso


show_visual_result

name = strcat(datestr(clock, 30),'.avi');
movie2avi(visual_result, name, 'compression', 'none');
```

11. main_rdpso

```
global fig vis temp_vis

counter = 0;
no_change_counter = 0;
while counter <=30 && no_change_counter<=25 % && gn_cost>global_minimum

  counter = counter + 1

  for s=1:size(swarms,1)
    for n=1:size(swarms{s}.positions)
      temp(n,:) = calc_opt_pos_sensing_func(swarms, obstacles, s, n, rs);
    end
    swarms{s}.xg_opt = zeros(Si,2);
    swarms{s}.xg_opt = temp;
    clear temp;
  end

        swarms = update_local_optimum(swarms, r);

  s=1;
  cont = 1;
  while cont == 1
    [swarms, SC, N_kill,next_swarm] = evolve_swarm(swarms, s, gn, SC, N_kill);
    if next_swarm>size(swarms,1)
      cont = 0;
    else
      s=next_swarm;
    end
```

```
    end


          if size(excluded.positions,1)>0
       for n=1:size(excluded.positions,1)
          temp(n,:) = calc_opt_pos_sensing_func_excl(excluded, obstacles, n, rs);
       end
       excluded.xg_opt = zeros(Si,2);
       excluded.xg_opt = temp;
       clear temp;

       excluded = evolve_excluded(excluded);
    end

          observations = update_observations(observations, swarms, excluded, r);



    % in the following lines we calculate the  global optimum (minimum)

    gn_cost = min(min(observations));
    [xtemp, ytemp] = find(observations==gn_cost);
    gn = [ytemp xtemp];

    clear xtemp ytemp
    condition_1 = condition_2;
    condition_2 = gn_cost;

    if condition_1 == condition_2
       no_change_counter = no_change_counter + 1;
    else
       no_change_counter = 0;
    end

    clear temp


    if fig==1
       f = figure('visible', vis);
       for s=1:size(swarms,1)
                   plot3(swarms{s}.positions(:,1),swarms{s}.positions(:,2),swarms{s}.cost_func,'o',  'Color',
color_matrix(s,:), 'markerfacecolor', color_matrix(s,:), 'MarkerSize', 12);
          hold on;
       end
       plot3(obstacles(:,1),obstacles(:,2),cost_obst_fig,'ks', 'markerfacecolor', 'k', 'MarkerSize', 15);
```

```
        plot3(min_col, min_row, global_minimum, 'm*', 'markerfacecolor', 'm', 'markeredgecolor', 'm',
'MarkerSize', 25);
    plot3(gn(1),gn(2),gn_cost, 'c*', 'markerfacecolor', 'c', 'markeredgecolor', 'c', 'MarkerSize', 25);
    if size(excluded.positions,1)>0
                        plot3(excluded.positions(:,1),  excluded.positions(:,2),  excluded.cost_func,  'ko',
'markerfacecolor', 'w', 'MarkerSize', 12);
    end
    surf(cost_func);
    axis([1 L 1 L 0 num_Gaussians]);
    xlabel('x'); ylabel('y'); zlabel('Probability Density');
    t = strcat('step ', int2str(counter));
    title(t);
    saveas(f, strcat(int2str(counter),'.jpg'));

    visual_result(counter) = getframe();

                    if mod(counter,ev_step)==0
                            f = figure('visible', vis);
                            vis = temp_vis;
                            subplot(2,1,1);
                            imagesc(1:L,1:L,observations);
                            axis([1 L 1 L])
                            xlabel('x'); ylabel('y');
                            title('observations');
                            subplot(2,1,2);
                            surf(1:L,1:L,cost_func);
                            axis([1 L 1 L]);
                            xlabel('x'); ylabel('y');
%       saveas(f, strcat(int2str(counter),'.jpg'));
                    end
  end

end
```

12. rand_num

```
function [r] = rand_num (a,b,i,j)

  if (nargin<2)
    error('too few input parameters')
  elseif (nargin==2)
      i=1; j=1;
  end;
  r = (b-a)*rand(i,j) + a;

end
```

13. show_visual_result

```
numtimes=1;
fps=2; %frames per second
figure;
movie(visual_result,numtimes,fps)
```

14. spawn_swarm

```
global fig excluded Ni

% we move all the particle data from the socially excluded group to the
% swarm s

s = size(swarms,1)+1;


swarms{s}.positions(1:Ni,:) = excluded.positions(1:Ni,:);
swarms{s}.cost_func(1:Ni,:) = excluded.cost_func(1:Ni,:);
swarms{s}.x_opt(1:Ni,:) = excluded.x_opt(1:Ni,:);
swarms{s}.velocity(1:Ni,:) = excluded.velocities(1:Ni,:);
swarms{s}.xg_opt(1:Ni,:) = excluded.xg_opt(1:Ni,:);
swarms{s}.improvement(1:Ni,:) = excluded.improvement(1:Ni,:);


% then we remove the particle (and all its info) from the swarm
if size(exluded.positions,1)==Ni
   excluded.positions = [];
   excluded.cost_func = [];
   excluded.x_opt = [];
   excluded.velocities = [];
   excluded.xg_opt = [];
   excluded.improvement = [];
else
   excluded.positions = removerows(excluded.positions, 'ind', 1:Ni);
   excluded.cost_func = removerows(excluded.cost_func, 'ind', 1:Ni);
   excluded.x_opt = removerows(excluded.x_opt, 'ind', 1:Ni);
   excluded.velocities = removerows(excluded.velocities, 'ind', 1:Ni);
   excluded.xg_opt = removerows(excluded.xg_opt, 'ind', 1:Ni);
   excluded.improvement = removerows(excluded.improvement, 'ind', 1:Ni);
end
```

15. update_local_optimum

```
function [swarms] = update_local_optimum (swarms, r)
```

```
global cost_func L global_minimum

temp = ones(L)*NaN;

        for s = 1:size(swarms,1)
                for n=1:size(swarms{s}.positions,1)
                        x = swarms{s}.positions(n,1);
                        y = swarms{s}.positions(n,2);
                        for i = x-r:x+r
                                if i<=0 || i>L
                                        continue;
                                end
                                for j = y-r:y+r
                                        if j<=0 || j>L
                                                continue;
          end
          temp(j,i) = cost_func(j,i);
                                end
                        end
                        [x,y] = find(temp==min(min(temp)));
                        swarms{s}.x_opt(n,:) = [x,y];
                        temp = ones(L)*NaN;
                end
  end
```

16. update_observations

```
function [observations] = update_observations (observations, swarms, excluded, r)

global cost_func L global_minimum

        for s = 1:size(swarms,1)
                for n=1:size(swarms{s}.positions,1)
                        x = swarms{s}.positions(n,1);
                        y = swarms{s}.positions(n,2);
                        for i = x-r:x+r
                                if i<=0 || i>L
                                        continue;
                                end
                                for j = y-r:y+r
                                        if j<=0 || j>L
                                                continue;
          end
          observations(j,i) = cost_func(j,i);
```

```
%
                                        end
                                end

                        end
    end

    for n=1:size(excluded.positions,1)
                                x = excluded.positions(n,1);
                                y = excluded.positions(n,2);
                                for i = x-r:x+r
                                        if i<=0 || i>L
                                                continue;
                                        end
                                        for j = y-r:y+r
                                                if j<=0 || j>L
                                                        continue;
                end
                observations(j,i) = cost_func(j,i);
%
                                        end
                                end

                        end

end
```

17. update_velocity

```
function [new_vel] = update_velocity(xn, x_opt, xng_opt, vel, gn)

%    global vmin vmax

%    initializations
    w = 1;
    c1 = 2;
    r1 = rand;
    c2 = 1;
    r2 = rand;
    c3 = 0.5;
    r3 = rand;
    if xn==gn
                        new_vel = rand(1,2);
```

```
        else
                new_vel = w*vel + c1*r1*(gn - xn) + c2*r2*(x_opt - xn) + c3*r3*(xng_opt - xn);
  end

  if (norm(new_vel)~=0)
     new_vel = new_vel./norm(new_vel);
  end

end
```

8.2 Κώδικας για RDPSO με επέκταση για διατήρηση του ad hoc connectivity

1. add_particle_from_excluded

```
global fig excluded dmax

found = 0;
for n=1:size(swarms{s}.positions,1)
        temp = sqrt(sum( (swarms{s}.positions(n,:)-excluded.positions(best_particle,:)).^2, 2) );
        if temp<=dmax
                found = 1; break
        end
end

if found == 1
        % we move all the particle data from the socially excluded group to the
        % swarm s

        p = size(swarms{s}.positions,1)+1;

        swarms{s}.positions(p,:) = excluded.positions(best_particle,:);
        swarms{s}.cost_func(p,:) = excluded.cost_func(best_particle,:);
        swarms{s}.x_opt(p,:) = excluded.x_opt(best_particle,:);
        swarms{s}.velocity(p,:) = excluded.velocities(best_particle,:);
        swarms{s}.xg_opt(p,:) = excluded.xg_opt(best_particle,:);
        swarms{s}.improvement(p,:) = excluded.improvement(best_particle,:);


        % then we remove the particle (and all its info) from the swarm
```

```
        excluded.positions = removerows(excluded.positions, 'ind', best_particle);
        excluded.cost_func = removerows(excluded.cost_func, 'ind', best_particle);
        excluded.x_opt = removerows(excluded.x_opt, 'ind', best_particle);
        excluded.velocities = removerows(excluded.velocities, 'ind', best_particle);
        excluded.xg_opt = removerows(excluded.xg_opt, 'ind', best_particle);
        excluded.improvement = removerows(excluded.improvement, 'ind', best_particle);

end
```

2. calc_closest_neighbor_v2

```
function [xm_opt] = calc_closest_neighbor_v2 (swarms, s, n, excluded, C_b)

global dmax



  x = swarms{s}.positions(n,:);
  a = find(C_b(n,:)==1);
  if size(a,2)>0
    clear xm
    for i=1:size(a,1)
      xm(i,:) = swarms{s}.positions(a(i),:);
    end
                if size(xm,1)>1
                        for i=1:size(a,1)
                                d = sqrt(sum((xm(i,:)-x(:)).^2));
                        end
                        temp = xm(find(d==min(d)),:);
                        clear xm
                        xm = temp;
                end

                vect = (x-xm)./norm(x-xm);
                xm_opt = round(xm + 0.7*dmax * vect);
  else
    xm_opt = [-1,-1];
  end
end
```

3. calc_connectivity_matrices

```
% calculate the adjacency matrix A - formula (3) of the respective paper
global dmax
```

```
clear A B C C_b

x = [];
x = swarms{s}.positions;

A = zeros(size(x,1));

for i = 1 : size(x,1)
   for j = i+1 : size(x,1)
      d = sqrt(sum((x(i,:)-x(j,:)).^2));
      if d<=dmax
         A(i,j) = 1;
         A(j,i) = 1;
      end
   end
end

B = zeros(size(x,1), size(x,1), size(x,1));
C = zeros(size(x,1), size(x,1), size(x,1));

B(1,:,:) = A;
C(1,:,:) = A;

for k = 2 : size(x,1)-1
   temp = squeeze(C(k-1,:,:))*squeeze(B(k-1,:,:));
   for i = 1 : size(x,1)
      for j = 1 : i-1
         if C(k-1,i,j) == 0 && temp(i,j) > 0
            B(k,i,j) = k;
            B(k,j,i) = k;
         end
      end
   end
   clear temp;
   C(k,:,:) = C(k-1,:,:) + B(k,:,:);
end

C_b = squeeze(C(size(x,1)-1,:,:));
```

4. calc_opt_pos_sensing_function

```
function [g_min_pos] = calc_opt_pos_sensing_func(swarms, obstacles, s, n, rs)

   global L
%     s is the swarm whose n-th particle is to be examined
```

```
x = swarms{s}.positions(n,:);

rc = 3; %rc is the radius from x within which we look for the optimum point

k=0;
visible_obst = zeros(1,2);
for i=1:size(obstacles,1)
   d = sqrt(sum((x-obstacles(i,:)).^2));
   if (d<=rs)
      k=k+1;
      visible_obst(k,:) = obstacles(i,:);
   end
end

for i=1:size(swarms,1)
   for j=1:size(swarms{i},1)
      d = sqrt(sum((x-swarms{i}.positions(j,:).^2)));
      if (d<=rs && d>0)
         k=k+1;
         visible_obst(k,:) = swarms{i}.positions(j,:);
      end
   end
end

foo1x = x(:,1) - 2*rc;
foo2x = x(:,1) + 2*rc;
foo1y = x(:,2) - 2*rc;
foo2y = x(:,2) + 2*rc;
if (foo1x < 0)
   foo1x = 0;
end
if (foo2x>L)
   foo2x = L;
end
if (foo1y < 0)
   foo1y = 0;
end
if (foo2y>L)
   foo2y = L;
end

[X, Y] = meshgrid(foo1x:foo2x, foo1y:foo2y);
exp_space = [X(:) Y(:)];
x_exp = rangesearch(exp_space, x, rc); %rangesearch returns a 1x1 cell
x_exp = x_exp{1};
```

```
  g = zeros(size(x_exp,1),1);
  for i=1:size(x_exp,1)
     x = exp_space(x_exp(i),:);
     for j=1:size(visible_obst,1)
        d = sqrt(sum((x-visible_obst(j,:)).^2));
        g(i) = g(i) + 1/d;
     end
  end


  g_min = find(g==min(g));
  g_min_pos = exp_space(x_exp(g_min),:);

end
```

5. calc_opt_pos_sensing_function_excl

```
function [g_min_pos] = calc_opt_pos_sensing_func_excl(excluded, obstacles, n, rs)

  global L

  x = excluded.positions(n,:);

  rc = 3; %rc is the radius from x within which we look for the optimum point

  k=0;
  visible_obst = zeros(1,2);
  for i=1:size(obstacles,1)
     d = sqrt(sum((x-obstacles(i,:)).^2));
     if (d<=rs)
        k=k+1;
        visible_obst(k,:) = obstacles(i,:);
     end
  end


  for j=1:size(excluded,1)
     d = sqrt(sum((x-excluded.positions(j,:).^2)));
     if (d<=rs && d>0)
        k=k+1;
        visible_obst(k,:) = excluded.positions(j,:);
     end
  end

  foo1x = x(:,1) - 2*rc;
  foo2x = x(:,1) + 2*rc;
```

```
  foo1y = x(:,2) - 2*rc;
  foo2y = x(:,2) + 2*rc;
  if (foo1x < 0)
     foo1x = 0;
  end
  if (foo2x>L)
     foo2x = L;
  end
  if (foo1y < 0)
     foo1y = 0;
  end
  if (foo2y>L)
     foo2y = L;
  end

  [X, Y] = meshgrid(foo1x:foo2x, foo1y:foo2y);
  exp_space = [X(:) Y(:)];
  x_exp = rangesearch(exp_space, x, rc); %rangesearch returns a 1x1 cell
  x_exp = x_exp{1};

  g = zeros(size(x_exp,1),1);
  for i=1:size(x_exp,1)
     x = exp_space(x_exp(i),:);
     for j=1:size(visible_obst,1)
        d = sqrt(sum((x-visible_obst(j,:)).^2));
        g(i) = g(i) + 1/d;
     end
  end



  g_min = find(g==min(g));
  g_min_pos = exp_space(x_exp(g_min),:);

end
```

6. create_space

```
global fig vis temp_vis L cost_func

num_Gaussians = 20; %number of Gaussians involved

% The space will be a rectangle with L-1 meters length on each side, from 1
% to L
L=60;
cost_func = zeros(L); %initializations
```

```
for i=1:num_Gaussians
   m = [rand_num(1,L) rand_num(1,L)];

   Sigma = [L/6 L/5; L/3 L/4];
%    Sigma = rand_num(1,L,2,2); %we can use this definition for a random
%    %variance matrix, but the space will likely have no local optima
   Sigma = Sigma*Sigma';

   x = 1:L; y = 1:L;
   [X,Y] = meshgrid(x,y);

   F = mvnpdf([X(:) Y(:)],m,Sigma);
   F = reshape(F,length(y),length(x))/max(max(F));
   cost_func = cost_func + F;
end

cost_func = num_Gaussians - cost_func;

% the global minimum and the position in which it occurs
[global_minimum, idx]=min(cost_func(:));
[min_row,min_col]=ind2sub(size(cost_func),idx);


observations = ones(size(cost_func))*NaN;

if (fig==1)
   figure('visible', vis);
   vis = temp_vis;
   surf(cost_func);
   axis([1 L 1 L 0 num_Gaussians])
   xlabel('x'); ylabel('y'); zlabel('Probability Density');

   figure('visible', vis);
   vis = temp_vis;
   imagesc(x,y,cost_func);
   axis([1 L 1 L])
   xlabel('x'); ylabel('y');

end;

%%% create the obstacles %%%

obstacles = randi(L-1,No,2);

if fig==1
   figure('visible', vis);
```

```
   vis = temp_vis;
   plot(obstacles(:,1),obstacles(:,2),'ks', 'markerfacecolor', 'k');
   axis([0 L 0 L])

end;
```

7. create_swarms_ad_hoc_con

```
global fig vis temp_vis excluded

swarms = cell(Si,1);

for s=1:Si
   clear temp;
   clear phi_k phi_n xo

   for k=1:Ni
      phi_k(k) = atan(1/sqrt(k));
   end

   for n=1:Ni
      phi_n(n) = sum(phi_k(1:n));
   end

%           we place the (random) centers of the various spirals of Theodorus we create
%   in such a way that the swarms are relatively well distributed in the plain


   if mod(s,3) == 0
      xo = rand_num(L/4,L/4,1,2);
           end
   if mod(s,3) == 1
      xo = rand_num(3*L/4,3*L/4,1,2);
   end
   if mod(s,3) == 2
      xo = rand_num(L/4,3*L/4,1,2);
   end


   for n=1:Ni
                      temp(n,:) = round([xo(1) + factor*dmax*sqrt(n+1)*cos(phi_n(n)), xo(2) +
factor*dmax*sqrt(n+1)*sin(phi_n(n))]);
   end
```

```
    swarms{s}.positions = zeros(Ni,2);
    swarms{s}.positions = temp;

    swarms{s}.x_opt = swarms{s}.positions;
    swarms{s}.velocity = round(rand_num(-1,1,Ni,2));
    clear temp;
    for n=1:size(swarms{s}.positions,1)
        temp(n,:) = calc_opt_pos_sensing_func(swarms, obstacles, s, n, rs);
    end
    swarms{s}.xg_opt = zeros(Si,2);
    swarms{s}.xg_opt = temp;

end

gn = [0,0];
gn_cost = 1000;

for i=1:size(swarms,1)
    s = size(swarms{i}.positions,1);
    for j=1:s
        swarms{i}.cost_func(j) = cost_func(swarms{i}.positions(j,2),swarms{i}.positions(j,1));
        swarms{i}.improvement(j) = -swarms{i}.cost_func(j);
        if (swarms{i}.cost_func(j)<gn_cost)
            gn_cost = swarms{i}.cost_func(j);
            gn = swarms{i}.positions(j,:);
        end
    end
    swarms{i}.cost_func = swarms{i}.cost_func';
end


condition_1 = eps;
condition_2 = gn_cost;
% condition is a 2x1 matrix, whose first element contains the previous
% value of the global optimum (minimum) and the second the current one.
% Subtraction of the two leads to the necessary condition to terminate the
% loop in the main_rdpso.m script. We initialize it wit h the values [0,
% gn], where gn is the current value of the global optimum.


excluded.positions = []; %we only define the socially excluded group,
%without assigning to it any initial values (particles) since all the
%particles belong to a swarm at first

excluded.x_opt = [];
excluded.velocities = [];
```

```
excluded.xg_opt = [];
excluded.cost_func = [];
excluded.improvement = [];

%%%%%%%%%%%%%%%%%%%%%%

cost_obst_fig = zeros(No,1);
for i=1:No
   cost_obst_fig(i) = cost_func(obstacles(i,2),obstacles(i,1));
end;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot of the search space cost, along with obstacles and particles for Si=3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if fig==1
   figure('visible', vis);
   vis = temp_vis;
   for s = 1:size(swarms,1)
                     plot(swarms{s}.positions(:,1),swarms{s}.positions(:,2),'o',    'Color',    color_matrix(s,:),
'markerfacecolor', color_matrix(s,:));
      hold on;
   end
   plot(obstacles(:,1),obstacles(:,2),'ks', 'markerfacecolor', 'k');
   axis([0 L 0 L])


   figure('visible', vis);
   vis = temp_vis;
   for s=1:size(swarms,1)
                  plot3(swarms{s}.positions(:,1),swarms{s}.positions(:,2),swarms{s}.cost_func,'o',  'Color',
color_matrix(s,:), 'markerfacecolor', color_matrix(s,:), 'MarkerSize', 12);
         hold on;
   end
   plot3(obstacles(:,1),obstacles(:,2),cost_obst_fig,'ks', 'markerfacecolor', 'k', 'MarkerSize', 15);
   plot3(min_col, min_row, global_minimum, 'm*', 'markerfacecolor', 'm', 'MarkerSize', 25);
   plot3(gn(1),gn(2),gn_cost, 'c*', 'markerfacecolor', 'c', 'MarkerSize', 25);
   if size(excluded.positions,1)>0
       plot3(excluded.positions(:,1), excluded.positions(:,2), excluded.cost_func, 'ko', 'markerfacecolor',
'w', 'MarkerSize', 12);
   end
   surf(cost_func);
   axis([1 L 1 L 0 num_Gaussians]);
   xlabel('x'); ylabel('y'); zlabel('Probability Density');
```

```
end;
```

test_distances

8. delete_swarm

```
global fig excluded

% we move all the particle data to the socially excluded group
s
p = size(excluded.positions,1)+1;
q = size(swarms{s}.positions,1);
excluded.positions(p:p+q-1,:) = swarms{s}.positions(:,:);
excluded.cost_func(p:p+q-1,:) = swarms{s}.cost_func(:,:);
excluded.x_opt(p:p+q-1,:) = swarms{s}.x_opt(:,:);
excluded.velocities(p:p+q-1,:) = swarms{s}.velocity(:,:);
excluded.xg_opt(p:p+q-1,:) = swarms{s}.xg_opt(:,:);
excluded.improvement(p:p+q-1,:) = swarms{s}.improvement(:,:);


% then we remove the particle (and all its info) from the swarm

swarms(s,:) = [];
```

9. evolve_excluded

```
function [excluded] = evolve_excluded (excluded)

   global L cost_func


   for n = 1:size(excluded.positions,1)
      new_vel = rand(1,2);
      if (norm(new_vel)~=0)
         new_vel = new_vel./norm(new_vel);
      end

      excluded.velocity(n,:) = new_vel;
      excluded.positions(n,:) = floor(excluded.positions(n,:) + 4*excluded.velocity(n,:));
      if excluded.positions(n,1) == 0
         excluded.positions(n,1) = 1;
      end
      if excluded.positions(n,2) == 0
```

```
        excluded.positions(n,2) = 1;
    end
    if excluded.positions(n,1) > L
        excluded.positions(n,1) = L;
    end
    if excluded.positions(n,2) > L
        excluded.positions(n,2) = L;
    end
    excluded.cost_func(n) = cost_func(excluded.positions(n,2),excluded.positions(n,1));



    excluded.improvement = -excluded.cost_func;

  end

end
```

10. evolve_swarm

```
function [swarms, SC, N_kill, next_swarm] = evolve_swarm (swarms, s, gn, SC, N_kill)

  global L cost_func SCmax Ni Nmin Nmax Smax excluded

  calc_connectivity_matrices
  next_swarm = s+1;
  n=1;
  while n<=size(swarms{s}.positions,1)

    xn = swarms{s}.positions(n,:);
    x_opt = swarms{s}.x_opt(n,:);
    xng_opt = swarms{s}.xg_opt(n,:);
    vel = swarms{s}.velocity(n,:);
    xm_opt = calc_closest_neighbor_v2 (swarms, s, n, excluded, C_b);
    swarms{s}.velocity(n,:) = update_velocity_ad_hoc(xn, x_opt, xng_opt, xm_opt, vel, gn);
    temp_prev_position = swarms{s}.positions(n,:);
    swarms{s}.positions(n,:) = floor(swarms{s}.positions(n,:) + 2*swarms{s}.velocity(n,:));

    calc_connectivity_matrices
    xm_opt = calc_closest_neighbor_v2 (swarms, s, n, excluded, C_b);
    if xm_opt == [-1,-1]
      pause;
      swarms{s}.positions(n,:) = floor((swarms{s}.positions(n,:) + temp_prev_position)./2);
      calc_connectivity_matrices
    else
      clear temp_prev_position;
```

```
    end

    if swarms{s}.positions(n,1) == 0
      swarms{s}.positions(n,1) = 1;
    end
    if swarms{s}.positions(n,2) == 0
      swarms{s}.positions(n,2) = 1;
    end
    if swarms{s}.positions(n,1) > L
      swarms{s}.positions(n,1) = L;
    end
    if swarms{s}.positions(n,2) > L
      swarms{s}.positions(n,2) = L;
    end
    swarms{s}.cost_func(n) = cost_func(swarms{s}.positions(n,2),swarms{s}.positions(n,1));
    swarms{s}.improvement(n) = swarms{s}.improvement(n) + swarms{s}.cost_func(n);
    n=n+1;
    test_connectivity
  end

  improvement = sum(swarms{s}.improvement);



  if (improvement>0)
    if (size(excluded.positions,1)~=0 && size(swarms{s}.positions,1)<Nmax)
%       if the socially excluded group is not empty add to the swarm the
%       best performing robot
      best_particle = find(excluded.cost_func==min(excluded.cost_func));
      add_particle_from_excluded
    end
  else
    SC(s) = SC(s) + 1;
    if SC(s)>SCmax
%       size(swarms{s}.positions,1)
%       Nmin
      if (size(swarms{s}.positions,1)>Nmin)
        if (improvement ~= 0)
          worst_particle = find(swarms{s}.improvement == min(swarms{s}.improvement));
        else
          worst_particle =  round(rand_num(1,size(swarms{s},1)));
        end
  %       exclude worst performing robot
        exclude_particle
        clear worst_particle
        N_kill(s) = N_kill(s) + 1;
```

```
            SC(s) = SCmax * (1 - 1/(N_kill(s)+1));
            swarms{s}.improvement = -swarms{s}.cost_func;
          else
            f = rand(1);
            prob = 0.75*rand(1)*size(swarms,1)/Smax;
            if f<=prob
              disp('delete');
              next_swarm = s;
              delete_swarm
            end
          end
        end
      end

      if (N_kill(s)==0 && size(swarms{s}.positions,1)<Smax && size(excluded.positions,1)>=Ni)
%       we compute the probability prob of spawning a new swarm
        f = rand(1);
        prob = f/size(swarms{s}.positions,1);
        if rand(1)<=prob
          disp('spawn');
          spawn_swarm
        end
      end


end
```

11. exclude_particle

```
global fig excluded

% we move all the particle data to the socially excluded group

p = size(excluded.positions,1)+1;

excluded.positions(p,:) = swarms{s}.positions(worst_particle,:);
excluded.cost_func(p,:) = swarms{s}.cost_func(worst_particle,:);
excluded.x_opt(p,:) = swarms{s}.x_opt(worst_particle,:);
excluded.velocities(p,:) = swarms{s}.velocity(worst_particle,:);
excluded.xg_opt(p,:) = swarms{s}.xg_opt(worst_particle,:);
excluded.improvement(p,:) = swarms{s}.improvement(worst_particle,:);


% then we remove the particle (and all its info) from the swarm
```

```
swarms{s}.positions = removerows(swarms{s}.positions, 'ind', worst_particle);
swarms{s}.cost_func = removerows(swarms{s}.cost_func, 'ind', worst_particle);
swarms{s}.x_opt = removerows(swarms{s}.x_opt, 'ind', worst_particle);
swarms{s}.velocity = removerows(swarms{s}.velocity, 'ind', worst_particle);
swarms{s}.xg_opt = removerows(swarms{s}.xg_opt, 'ind', worst_particle);
swarms{s}.improvement = removerows(swarms{s}.improvement, 'ind', worst_particle);
```

12. main_ad_hoc_con

```
close all
clear all

global fig vis temp_vis Ni Nmin Nmax SCmax Si Smin Smax vmin vmax dmax

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% initialization of important parameters %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fig = 1; % if fig is 1 then the various plots at the different stages of
% the algorithm will be computed if fig is 0 then the plots remain unseen

vis = 'on'; % (for vis to be of any significance, fig must be 1) if vis is
% on, the plots computed along the way are visible to the user, else (if it
% is off, they remain invisible

temp_vis = vis; %every time vis is used in a figure its value is
% automatically switched to 'on', so we keep the desired value in temp_vis
% so vis will be set back to it each time (some plots are visible
% regardless of vis's value, in order for the final avi movie to be
% produced)

Ni = 3; %initial number of particles (robots) per swarm
Nmin = 1; %minimum number of particles per swarm
Nmax = 3; %maximum number of particles per swarm

Si = 3; %initial number of swarms
Smin = 2; %minimum number of swarms
Smax = 5; %maximum number of swarms

Np = Ni*Si; %total number of particles

SCmax = 10; %stagnancy threshold <-------need to figure out the parameter's value
SC = zeros(Si,1); %we initialize the search counter for each swarm with zero

N_kill = zeros(Si,1); %we initialize the number of particles that have been
```

```
% excluded from each swarm

No = 15; %number of obstacles

rs = 5; %sensing radius (depends on the sensors' type and range)

dmax = 10; %the maximum distance that ensures ad hoc connectivity

r = 5; %observation radius

ev_step = 5;

color_matrix = [1 0 0; 0 1 0; 1 1 0; 0.2 0 0; 0.5 0.5 0.5; 1 1 0.5];


%% end of initializations %%

create_space

factor = 0.5; %this is a factor used to calculate the initial
% positions of the particles

create_swarms_ad_hoc_con

% swarms are placed according to the spiral of Theodorus in the experimentation space

% outcast_particles are the socially excluded ones



% For each particle calculate the new speed and position, update the swarms
% matrix, calculate the Nkill(i) for each swarm, i, and redefine the respective
% search counter SC(i). If the critical threshold SCmax is exceeded, then we
% update SC(i) to SCmax*(1-1/(NKill(i)+1)) and the worst performing robot
% is excluded and moved to the outcast particles' group. If the swarm
% improves its objective function then the best performing robot of the
% outcast group is transferred to this swarm.

main_rdpso


show_visual_result

name = strcat(datestr(clock, 30),'.avi');
movie2avi(visual_result, name, 'compression', 'none');
```

13. main_rdpso

```
global fig vis temp_vis

counter = 0;
no_change_counter = 0;
while counter <=30 && no_change_counter<=25

  counter = counter + 1

  for s=1:size(swarms,1)
    for n=1:size(swarms{s}.positions)
      temp(n,:) = calc_opt_pos_sensing_func(swarms, obstacles, s, n, rs);
    end
    swarms{s}.xg_opt = zeros(Si,2);
    swarms{s}.xg_opt = temp;
  end


  s=1;
  cont = 1;
  while cont == 1
    [swarms, SC, N_kill,next_swarm] = evolve_swarm(swarms, s, gn, SC, N_kill);
    if next_swarm>size(swarms,1)
      cont = 0;
    else
      s=next_swarm;
    end
  end


  if size(excluded.positions,1)>0
    for n=1:size(excluded.positions,1)
      temp(n,:) = calc_opt_pos_sensing_func_excl(excluded, obstacles, n, rs);
    end
    excluded.xg_opt = zeros(Si,2);
    excluded.xg_opt = temp;
    clear temp;

    excluded = evolve_excluded(excluded);
  end

        observations = update_observations(observations,swarms, excluded,r);
```

```
% in the following lines we calculate the  global optimum (minimum)

gn_cost = min(min(observations));
[xtemp, ytemp] = find(observations==gn_cost);
gn = [ytemp xtemp];

clear xtemp ytemp
condition_1 = condition_2;
condition_2 = gn_cost;

if condition_1 == condition_2
   no_change_counter = no_change_counter + 1;
else
   no_change_counter = 0;
end

clear temp


if fig==1
   f = figure('visible', vis);
   vis = temp_vis;
   for s=1:size(swarms,1)
            plot3(swarms{s}.positions(:,1),swarms{s}.positions(:,2),swarms{s}.cost_func,'o',  'Color',
color_matrix(s,:), 'markerfacecolor', color_matrix(s,:), 'MarkerSize', 12);
      hold on;
   end
   plot3(obstacles(:,1),obstacles(:,2),cost_obst_fig,'ks', 'markerfacecolor', 'k', 'MarkerSize', 15);
   plot3(min_col, min_row, global_minimum, 'm*', 'markerfacecolor', 'm', 'MarkerSize', 25);
   plot3(gn(1),gn(2),gn_cost, 'c*', 'markerfacecolor', 'c', 'MarkerSize', 25);
   if size(excluded.positions,1)>0
            plot3(excluded.positions(:,1),   excluded.positions(:,2),   excluded.cost_func,   'ko',
'markerfacecolor', 'w', 'MarkerSize', 12);
   end
   surf(cost_func);
   axis([1 L 1 L 0 num_Gaussians]);
   xlabel('x'); ylabel('y'); zlabel('Probability Density');

   t = strcat('step ', int2str(counter));
   title(t);
   saveas(f, strcat(int2str(counter),'.jpg'));

   visual_result(counter) = getframe();
```

```
                    if mod(counter,ev_step)==0
                            f = figure('visible', vis);
                            vis = temp_vis;
                            subplot(2,1,1);
                            imagesc(1:L,1:L,observations);
                            axis([1 L 1 L])
                            xlabel('x'); ylabel('y');
                            title('observations');
                            subplot(2,1,2);
                            surf(1:L,1:L,cost_func);
                            axis([1 L 1 L]);
                            xlabel('x'); ylabel('y');
%           saveas(f, strcat(int2str(counter),'.jpg'));
                    end

   end


end
```

14. rand_num

```
function [r] = rand_num (a,b,i,j)

   if (nargin<2)
      error('too few input parameters')
   elseif (nargin==2)
         i=1; j=1;
   end;
   r = (b-a)*rand(i,j) + a;

end
```

15. show_visual_result

```
numtimes=1;
fps=2; %frames per second
figure;
movie(visual_result,numtimes,fps)
```

16. spawn_swarm

```
global fig excluded Ni

% we move all the particle data from the socially excluded group to the
```

```
% swarm s


x = [];
x = excluded.positions;
temp = excluded.positions;
indices = [];
A = zeros(size(x,1));

for i = 1 : size(x,1)
   for j = i+1 : size(x,1)
      d = sqrt(sum((x(i,:)-x(j,:)).^2));
      if d<=dmax
         A(i,j) = 1;
         A(j,i) = 1;
      end
   end
end

for i = 1:size(excluded.positions,1)
            if sum(A(i,:) > 0
                        indices = [indices; i];
            end
end

if size(indices,1)>=Ni
            s = size(swarms,1)+1;

            swarms{s}.positions(1:Ni,:) = excluded.positions(indices(1:Ni),:);
            swarms{s}.cost_func(1:Ni,:) = excluded.cost_func(indices(1:Ni),:);
            swarms{s}.x_opt(1:Ni,:) = excluded.x_opt(indices(1:Ni),:);
            swarms{s}.velocity(1:Ni,:) = excluded.velocities(indices(1:Ni),:);
            swarms{s}.xg_opt(1:Ni,:) = excluded.xg_opt(indices(1:Ni),:);
            swarms{s}.improvement(1:Ni,:) = excluded.improvement(indices(1:Ni),:);


            % then we remove the particle (and all its info) from the swarm
            if size(exluded.positions,1)==Ni
                        excluded.positions = [];
                        excluded.cost_func = [];
                        excluded.x_opt = [];
                        excluded.velocities = [];
                        excluded.xg_opt = [];
                        excluded.improvement = [];
            else
                        excluded.positions = removerows(excluded.positions, 'ind', indices(1:Ni));
```

```
                excluded.cost_func = removerows(excluded.cost_func, 'ind', indices(1:Ni));
                excluded.x_opt = removerows(excluded.x_opt, 'ind', indices(1:Ni));
                excluded.velocities = removerows(excluded.velocities, 'ind', indices(1:Ni));
                excluded.xg_opt = removerows(excluded.xg_opt, 'ind', indices(1:Ni));
                excluded.improvement = removerows(excluded.improvement, 'ind', indices(1:Ni));
        end

end
```

17. test_connectivity

```
for p=1:3
  a=find(C_b(p,:)==1);
  if size(a,2) == 0
    disp('NO CONNECTIVITY');
    pause;
  end
end
```

18. test_distances

```
global dmax

for s=1:size(swarms,1)
  temp_test = [];
  temp_test = swarms{s}.positions;

  for i=2:Ni
    test(i-1) = sqrt(sum( (temp_test(i-1,:)-temp_test(i,:)).^2, 2) );
  end

  if find(test>dmax)
  %   disp('ERROR');
    factor = factor - 0.005;
    clear swarms
    create_swarms_ad_hoc_con
    break;
  end

% condition that a particle does not coincide with obstacle
end
```

19. update_local_optimum

```
function [swarms] = update_local_optimum (swarms, r)
```

```
global cost_func L global_minimum

temp = ones(L)*NaN;

        for s = 1:size(swarms,1)
                for n=1:size(swarms{s}.positions,1)
                        x = swarms{s}.positions(n,1);
                        y = swarms{s}.positions(n,2);
                        for i = x-r:x+r
                                if i<=0 || i>L
                                        continue;
                                end
                                for j = y-r:y+r
                                        if j<=0 || j>L
                                                continue;
          end
          temp(j,i) = cost_func(j,i);
                                end
                        end
                        [x,y] = find(temp==min(min(temp)));
                        swarms{s}.x_opt(n,:) = [x,y];
                        temp = ones(L)*NaN;
                end
  end
```

20. update_observations

```
function [observations] = update_observations (observations, swarms, excluded, r)

global cost_func L global_minimum

        for s = 1:size(swarms,1)
                for n=1:size(swarms{s}.positions,1)
                        x = swarms{s}.positions(n,1);
                        y = swarms{s}.positions(n,2);
                        for i = x-r:x+r
                                if i<=0 || i>L
                                        continue;
                                end
                                for j = y-r:y+r
                                        if j<=0 || j>L
                                                continue;
          end
          observations(j,i) = cost_func(j,i);
```

```
%
                                        end
                                end

                        end
    end

    for n=1:size(excluded.positions,1)
                                x = excluded.positions(n,1);
                                y = excluded.positions(n,2);
                                for i = x-r:x+r
                                        if i<=0 || i>L
                                                continue;
                                        end
                                        for j = y-r:y+r
                                                if j<=0 || j>L
                                                        continue;
                end
                observations(j,i) = cost_func(j,i);
%
                                        end
                                end

                        end

end
```

21. update_velocity

```
function [new_vel] = update_velocity_ad_hoc(xn, x_opt, xng_opt, xm_opt, vel, gn)

%    initializations
    w = 1;
    c1 = 1;
    r1 = rand;
    c2 = 1;
    r2 = rand;
    c3 = 0.5;
    r3 = rand;
    c4 = 3;
    r4 = rand;

    if xn==gn
                        new_vel = rand(1,2);
    else
```

```
      new_vel = w*vel + c1*r1*(gn - xn) + c2*r2*(x_opt - xn) + c3*r3*(xng_opt - xn) + c4*r4*(xm_opt -
xn);
   end

  if (norm(new_vel)~=0)
      new_vel = new_vel./norm(new_vel);
   end

end
```

9 *Βιβλιογραφία*

[1] "A novel Multi-Robot Exploration Approach based on Particle Swarm Optimization Algorithms", Couceiro, Rocha, et al, 2011

[2] "Ensuring Ad Hoc Connectivity in Distributed Search with Robotic Darwinian Particle Swarms", Couceiro, Rocha, et al, 2011

[3] „Τεχνητή Νοημοσύνη – Μια σύγχρονη προσέγγιση", S. Russel, P. Norvig, Δεύτερη Αμερικανική Έκδοση, Εκδόσεις Κλειδάριθμος

[4] https://en.wikipedia.org/wiki/History_of_artificial_intelligence

[5] https://en.wikipedia.org/wiki/Bio-inspired_computing

[6] http://www.cleveralgorithms.com/nature-inspired/index.html