ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Δημιουργία Συμμαχιών σε Πολυπρακτορικά Συστήματα

Διπλωματική Εργασία

Ηλίας Φλόκας

Επιβλέπων: Δασκαλοπούλου Ασπασία

Βασιλακόπουλος Μιχαήλ

Βόλος 2020

Σχηματισμός Συμμαχιών σε Πολυπρακτορικά Συστήματα

Λέξεις Κλειδιά: Πράκτορας, Συμμαχία, Πολυπρακτορικό Σύστημα, Τεχνητή Νοημοσύνη, Ευφυής Πράκτορας, Πράκτορας Λογισμικού, Πολυπλοκότητα, Αλγόριθμος, Βέλτιστη Δομή

ΠΕΡΙΛΗΨΗ

Ο σχηματισμός συμμαχιών είναι μια πολύ σημαντική διαδικασία όχι μόνο στην καθημερινότητά μας και στην δουλειά μας αλλά και στην τεχνητή νοημοσύνη. Όταν μιλάμε για σχηματισμό συμμαχιών στην τεχνητή νοημοσύνη αναφερόμαστε στην διαδικασία μέσω της οποία περισσότεροι από ένας πράκτορες λογισμικού σταματούν να λειτουργούν ατομικά κοιτώντας μόνο το δικό τους όφελος και ξεκινάνε να λειτουργούν σαν μέλη μιας ομάδας. Αυτό πρακτικά σημαίνει ότι ξεκινάνε να λειτουργούνε έχοντας έναν κοινό στόχο, δουλεύουν όλοι μαζί ώστε να πετύχουν κάτι από κοινού, αυτό το πετυχαίνουν προσφέροντας ο καθένας στην συμμαχία τους πόρους που έχει στην κατοχή του με σκοπό να επωφεληθεί το σύνολο. Η συμμαχία δημιουργείται και λειτουργεί για σύντομο χρονικό διάστημα, μέχρι δηλαδή να επιτευχθεί ο στόχος. Αυτήν η διαδικασία ιδανικά θέλουμε να εκτελείται στον ελάχιστο δυνατό χρόνο και όσο πιο αποδοτικά γίνεται, ιδιαίτερα όταν έχουμε να λύσουμε ένα πρόβλημα που ατομικά θα ήταν αδύνατο να λυθεί και επομένως η συμμαχία είναι ο μόνος βιώσιμος τρόπος να το φέρουμε εις πέρας. Είναι πολύ σημαντικό επομένως να διερευνηθούν οι αλγόριθμοι σχηματισμού δομής συμμαχίας με απώτερο σκοπό την εύρεση και υπολογισμό του αποδοτικότερου αλγορίθμου για την εκτέλεση αυτής της διαδικασίας.

Coalition Formation in Multi-Agent Systems

Keywords: Agent, Coalition, Multi-Agent System, Artificial Intelligence, AI, Intelligent Agent, Software Agent, Algorithm Complexity, Algorithm, Best Coalition Structure

ABSTRACT

Coalition formation is a very important process, not only when it comes to our everyday life or our work, but in Artificial Intelligence as well. In artificial intelligence the term coalition formation and coalition structure generation refers to the process during which a group of intelligent software agents stop working individually minding only their own benefit, they form a group and they start working together. This means that they share one common goal and they strive to achieve it just like a team. This is achievable when the agents co-operate together and share their resources, acting like one agent, when everyone's goal becomes the common goal of the coalition. Coalitions have a short lifespan, starting when the need arises and lasting until this need or goal is satisfied. It is critical that this process is executed using the bare minimum when it comes to time and resources, especially when we have a problem which is impossible to solve with just a single agent, where forming a coalition is our only viable choice. It is important to research our possible "state-of-the-art" algorithm choices and discover which one of them is the optimal to use when we face the problem of generating a coalitional structure in a multi-agent system.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

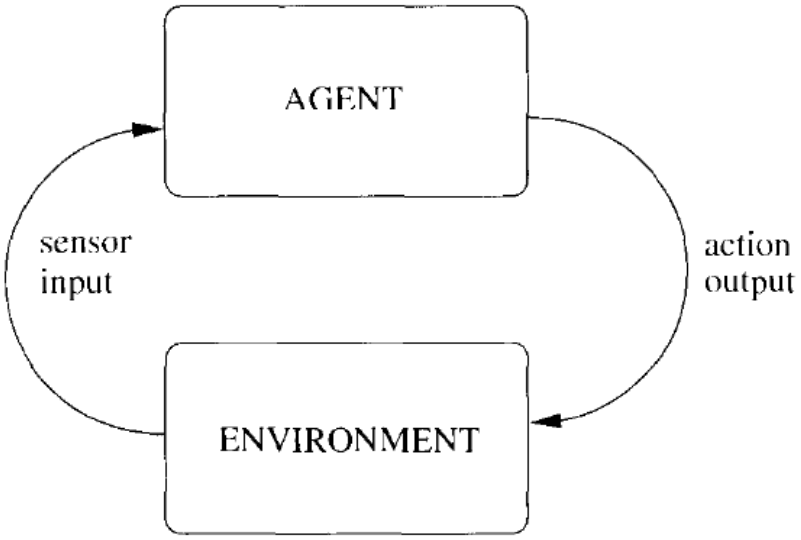
1.1 Βασικό Θεωρητικό Υπόβαθρο

1.1.1 Τεχνητή Νοημοσύνη

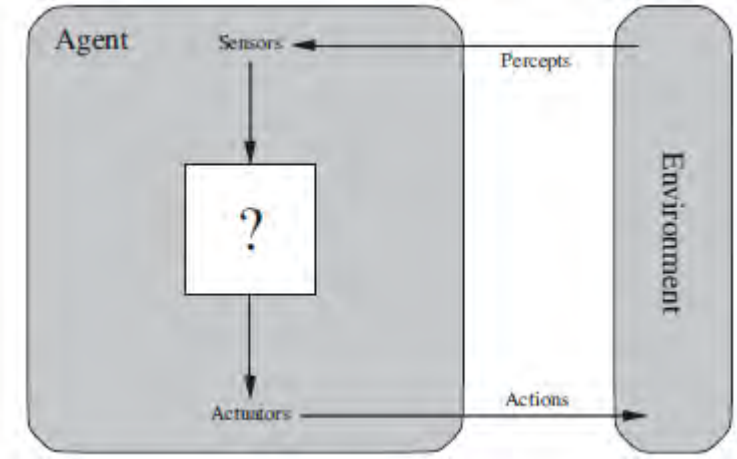
Ένας απ' τους βασικούς κλάδους της πληροφορικής είναι και η τεχνητή νοημοσύνη. «Τεχνητή Νοημοσύνη είναι εκείνος ο κλάδος της επιστήμης των υπολογιστών που ασχολείται με το σχεδιασμό ευφυών υπολογιστικών συστημάτων, δηλαδή συστημάτων με χαρακτηριστικά τα οποία σχετίζονται με την ευφυΐα στην ανθρώπινη συμπεριφορά (μάθηση, αιτίαση, επίλυση προβλημάτων, κατανόηση φυσικής γλώσσας, αναγνώριση αντικειμένων κτλ.).» [1] Από την δεκαετία του 1940 όπου παρουσιάστηκε η πρώτη μαθηματική περιγραφή ενός τεχνητού νευρωνικού δικτύου μέχρι και σήμερα η τεχνολογία έχει αναπτυχθεί αρκετά και η τεχνητή νοημοσύνη έχει επεκταθεί σε πολλούς τομείς όπως φιλοσοφία, μαθηματικά, οικονομικά, νευροεπιστήμη, ψυχολογία, στην επιστήμη των μηχανικών καθώς και στην γλωσσολογία. Διάφορες εφαρμογές της τεχνητής νοημοσύνης στην σύγχρονη τεχνολογία αποτελούν ρομποτικά οχήματα, αναγνώριση ομιλίας, ρομποτική, καταπολέμηση του «spamming», αυτόματη μετάφραση. [2]

1.1.2. Ευφυείς πράκτορες

Τα ευφυή αυτά υπολογιστικά συστήματα ονομάζονται ευφυείς πράκτορες. Βάσει Wooldridge για τον όρο ευφυής πράκτορας δεν υπάρχει γενικά καθολικός ορισμός, ένα χαρακτηριστικό που φαίνεται να είναι σημαντικό για τους πράκτορες είναι η αυτονομία ενώ για παράδειγμα η δυνατότητα των πρακτόρων να μαθαίνουν από τις εμπειρίες τους σε κάποιες περιπτώσεις είναι ένα πολύ σημαντικό χαρακτηριστικό ενώ σε άλλες όχι απλά είναι ασήμαντο αλλά και ανεπιθύμητο. Ένας γενικά αποδεκτός ορισμός για το τι είναι ευφυής πράκτορας είναι ο ακόλουθος «Ο πράκτορας είναι ένα υπολογιστικό σύστημα που βρίσκεται σε κάποιο περιβάλλον μέσα στο οποίο είναι ικανός για αυτόνομη δράση προκειμένου να πετύχει τους σχεδιαστικούς του στόχους.» [3] Ακολουθούν δύο εικόνες που περιγράφουν την λειτουργία του πράκτορα που βρίσκεται μέσα σε κάποιο περιβάλλον



Εικόνα 1.1: Ένας πράκτορας στο περιβάλλον του. Λαμβάνει είσοδο απ' τους αισθητήρες του και παράγει ως έξοδο δράσεις που επηρεάζουν το περιβάλλον του [3]



Εικόνα 1.2: Ένας πράκτορας στο περιβάλλον του. Λίγη παραπάνω ανάλυση του σχεδιασμού ενός πράκτορα. [2]

Ένα πολύ απλό παράδειγμα πράκτορα αποτελεί ο θερμοστάτης ο οποίος ακολουθεί τους εξής κανόνες:

Πολύ κρύο → ενεργοποίηση θέρμανσης

Κατάλληλη θερμοκρασία → απενεργοποίηση θέρμανσης

Ένας ευφυής πράκτορας θα πρέπει να διαθέτει τα εξής είδη ικανοτήτων αντιδραστικότητα (reactivity), ενεργητικότητα (proactiveness) και κοινωνική ικανότητα (social ability). [3] Στην παρούσα διπλωματική εργασία θα μας απασχολήσει κυρίως η τρίτη ικανότητα δηλαδή η ικανότητα του πράκτορα να δημιουργήσει συμμαχίες με άλλους πράκτορες που ανήκουν στο περιβάλλον του με απώτερο σκοπό την επίτευξη ενός ατομικού ή συλλογικού στόχου.

Τα περιβάλλοντα βάσει των ιδιοτήτων τους χωρίζονται σε πλήρως παρατηρήσιμο ή μερικώς παρατηρήσιμο, μονοπρακτορικό ή πολυπρακτορικό, αιτιοκρατικό ή μη αιτιοκρατικό, επεισοδιακό ή ακολουθιακό, στατικό ή δυναμικό, διακριτό ή συνεχές, προσιτό ή απρόσιτο. [2] Ακολουθεί μια εικόνα με παραδείγματα για καλύτερη κατανόηση των χαρακτηριστικών.

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Εικόνα 1.3: Παραδείγματα περιβαλλόντων πρακτόρων και χαρακτηριστικά τους. [2]

Οι ευφυείς πράκτορες μπορούν να κατηγοριοποιηθούν βάσει της γενικής δομής του κόσμου σε τρεις κλάσεις:

- Physical Agents
- Mental Agents
- Structural or Information Agents

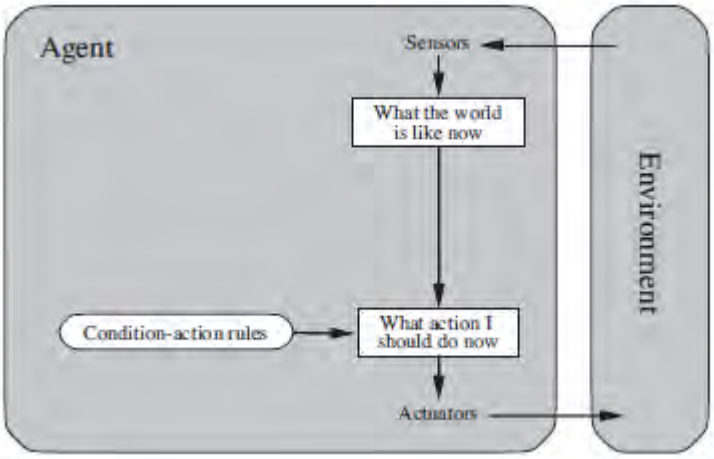
Όπου στην κλάση των Physical Agents ανήκουν άνθρωποι ζώα και ρομπότ στην κλάση των Mental Agents ανήκουν πράκτορες λογισμικού (Software Agents) ενώ η κεφαλή ενός Turing Machine είναι ένας Structural Agent. Περαιτέρω οι Physical Agents μπορούν να χωριστούν στις παρακάτω τρεις κλάσεις:

- Biological Agents
- Artificial Agents
- Hybrid Agents

Άνθρωποι ζώα και μικροοργανισμοί ανήκουν στην κατηγορία των Biological Agents τα ρομπότ ανήκουν στην κατηγορία των Artificial Agents ενώ στους Hybrid Agents ανήκουν πράκτορες που έχουν και βιολογικά και τεχνητά μέρη. [4]
Μια άλλη κατηγοριοποίηση των ευφυών πρακτόρων είναι αυτήν που προτείνεται από τους Russel and Norvig [2] βάσει της οποίας οι πράκτορες χωρίζονται σε τέσσερις βασικές κλάσεις:

- Simple Reflex Agents
- Model-based Reflex Agents
- Goal-based Agents
- Utility-based Agents

Οι simple reflex agents διαλέγουν την δράση τους βάσει της τωρινής τους αντίληψης αγνοώντας παρελθοντικές αντιλήψεις. Για παράδειγμα ο πράκτορας καθαριστής είναι ένας simple reflex agent ο οποίος επιλέγει δράση βασιζόμενος μόνο στην θέση την οποία ελέγχει αυτήν την στιγμή και στο αν αυτήν είναι καθαρή η βρώμικη την παρούσα στιγμή.



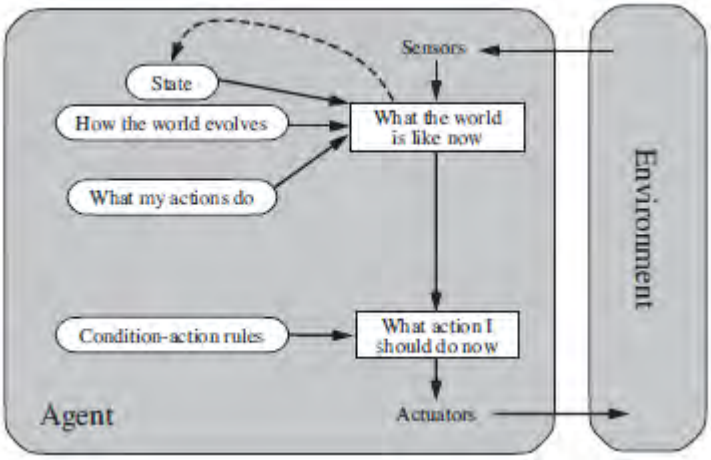
Εικόνα 1.4: Παραστατικό διάγραμμα ενός Simple Reflex Agent [2]

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Εικόνα 1.5: Ψευδοκώδικας υλοποίησης ενός Simple Reflex Agent [2]

Οι Model-based Reflex Agents κρατάνε μια εσωτερική κατάσταση η οποία εξαρτάται από το ιστορικό αντίληψης βάσει της οποίας κατάστασης λαμβάνουν την απόφαση. Όταν μιλάμε για έναν πράκτορα που αποφασίζει αν θα φρενάρει η όχι ένα όχημα για παράδειγμα μας αρκεί ένα στιγμιότυπο απ' την κάμερα στο οποίο να φαίνεται ότι τα κόκκινα φώτα στα άκρα του αυτοκινήτου είναι αναμμένα ή όχι. Αυτήν η εσωτερική κατάσταση προφανώς πρέπει να αναβαθμίζεται.



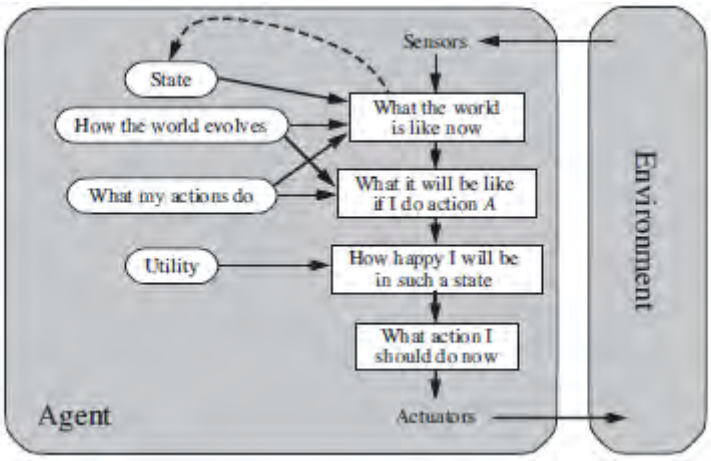
Εικόνα 1.6: Παραστατικό διάγραμμα ενός Model-based Reflex Agent [2]

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Εικόνα 1.7: Ψευδοκώδικας υλοποίησης ενός Model-based Reflex Agent [2]

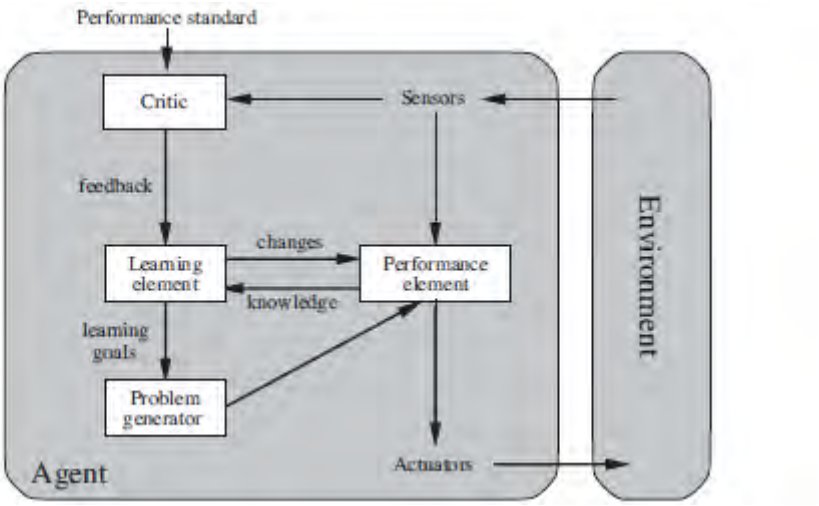
Οι Goal-based Agents χρειάζονται πέραν μιας εσωτερικής κατάστασης και ένα στόχο με βάση τον οποίο θα αποφασίσουν πως θα ενεργήσουν. Αν ένα όχημα είναι σε μια διασταύρωση δεν του αρκεί μια κατάσταση για να αποφασίσει αν θα πάει αριστερά δεξιά η ευθεία, πρέπει να έχει έναν στόχο βάσει του οποίου θα μπορέσει να καταλήξει στην απόφαση. Για παράδειγμα αν αυτό το όχημα θέλει να πάει στο αεροδρόμιο και το αεροδρόμιο είναι ευθεία θα επιλέξει να συνεχίσει να πηγαίνει ευθεία. Πολλές φορές όμως ένας στόχος δεν είναι αρκετός για να παράξει μια συμπεριφορά υψηλής ποιότητας στα περισσότερα περιβάλλοντα. Ένα ταξί μπορεί να φτάσει στον προορισμό του ακολουθώντας πολλές διαδρομές εκ των οποίων κάποιες είναι περισσότερο ασφαλείς γρηγορότερες ή αποτελεσματικότερες. Οι στόχοι απλά σε βοηθάνε να κατηγοριοποιήσεις δυαδικά αν επιτεύχθηκε η όχι κάτι μέσω μιας ενέργειας και όχι πόσο καλά επιτεύχθηκε. Σε αυτήν την περίπτωση για να βρούμε δηλαδή πόσο καλά επιτεύχθηκε ένας στόχος αναγκαζόμαστε να χρησιμοποιήσουμε Utility-based Agents.



Εικόνα 1.8: Παραστατικό διάγραμμα ενός Goal-based, Utility-based Agent. [2]

Υπάρχει άλλη μια κατηγορία πρακτόρων οι Learning Agents. Το να μαθαίνει ένας πράκτορας έχει το πλεονέκτημα ότι μπορεί αρχικά να ξεκινά την λειτουργία του σε άγνωστα περιβάλλοντα και να γίνεται όλο και πιο ικανός απ' ότι αρχικά θα του επιτρεπόταν. Ο πιο διαδεδομένος διαχωρισμός είναι μεταξύ του στοιχείου της μάθησης που είναι υπεύθυνο για την βελτίωση και του στοιχείου της απόδοσης που είναι υπεύθυνο για την επιλογή δράσεων. Το στοιχείο της εκμάθησης ανατροφοδοτείται από την κριτική που λαμβάνει για το πως ο πράκτορας πρέπει να αλλάξει το στοιχείο της απόδοσης έτσι ώστε να υπάρξει μελλοντική βελτίωση. Το στοιχείο απόδοσης είναι αυτό που προηγουμένως είχαμε ονομάσει ευφυή πράκτορα. Λαμβάνει μια αντίληψη και αποφασίζει αναλόγως να δράσει. Το τελευταίο στοιχείο ενός Learning Agent είναι

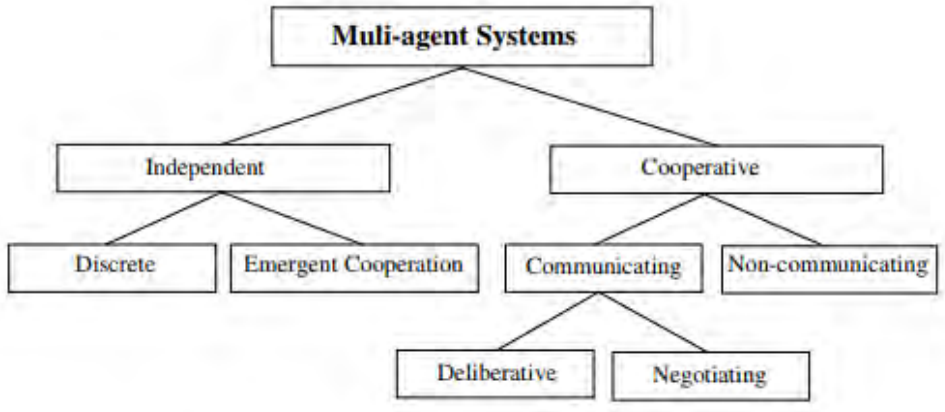
μια «γεννήτρια προβλημάτων», αυτό το στοιχείο είναι υπεύθυνο να προτείνει μελλοντικές δράσεις που μπορούν να οδηγήσουν σε νέες και πληροφοριακές εμπειρίες.



Εικόνα 1.9: Παραστατικό παράδειγμα ενός learning agent [2]

1.2 Πολυπρακτορικά Συστήματα

Όπως με τον ορισμό των πρακτόρων αντίστοιχα και με τα πολυπρακτορικά συστήματα υπάρχουν πολλοί ορισμοί . Ένας γενικά αποδεκτός ορισμός είναι ο ακόλουθος: «Ένα πολυπρακτορικό σύστημα είναι ένα ελαφρώς συνδεδεμένο δίκτυο από οντότητες επίλυσης προβλημάτων (πράκτορες) οι οποίοι συνεργάζονται για να βρούνε απαντήσεις σε προβλήματα που ξεπερνάνε τις ατομικές δυνατότητες ή γνώσεις μιας οντότητας (πράκτορα).» Το γεγονός ότι οι πράκτορες δουλεύουν μεταξύ τους δείχνει ότι υπάρχει κάποιου είδους συνεργασία. Όμως στα πολυπρακτορικά συστήματα η λογική της ισοτιμίας είναι στην καλύτερη ασαφής, στην χειρότερη ασυνεπής. [5]



Εικόνα 1.10: Αναπαράσταση τυπολογίας συνεργασίας για πολυπρακτορικά συστήματα [5]

Σε ένα μονολιθικό σύστημα έχουμε είτε έναν πράκτορα ή περισσότερους από έναν πράκτορες εκ των οποίων ένας έχει ηγετικό ρόλο και έχει πλήρη γνώση και δυνατότητες επίλυσης του προβλήματος και συνήθως αφορούν απλά προβλήματα αντίθετα με τα πολυπρακτορικά συστήματα τα οποία μπορούν να αναλάβουν πιο πολύπλοκα προβλήματα. Τα βασικά πλεονεκτήματα ενός πολυπρακτορικού

συστήματος είναι η δυνατότητα διαμοίρασης του ελέγχου και ευθυνών μεταξύ των πρακτόρων με σκοπό το σύστημα να μπορεί να ανεχτεί σφάλματα από έναν ή περισσότερους πράκτορες καθώς και η επεκτασιμότητα, ότι δηλαδή πρέπει να είναι ευκολότερο να προσθέσουμε έναν νέο πράκτορα στο πολυπρακτορικό σύστημα από την προσθήκη δυνατοτήτων σε ένα μονολιθικό σύστημα.

Τα χαρακτηριστικά ενός πολυπρακτορικού συστήματος βάσει [6] είναι τα εξής:

- Κάθε πράκτορας έχει ελλιπή πληροφορία ή δυνατότητες για την επίλυση του συνολικού προβλήματος και έτσι περιορισμένη άποψη
- Δεν υπάρχει κάποιος πράκτορας που έχει καθολική εξουσία, αλλιώς θα μιλούσαμε για μονολιθικό σύστημα
- Κάθε πράκτορας είναι τουλάχιστον μερικώς ανεξάρτητος έχει αυτογνωσία και είναι αυτόνομος
- Οι υπολογισμοί είναι ασύγχρονοι

Η δημιουργία συμμαχιών είναι μια σημαντική δυνατότητα αυτοματοποιημένης διαπραγμάτευσης μεταξύ πρακτόρων. Η δημιουργία μιας συμμαχίας γίνεται με σκοπό το σύνολο των αποτελεσμάτων όλων των συμμαχιών να είναι το μέγιστο δυνατό και να ξεπερνά το αποτέλεσμα το οποίο θα ήταν εφικτό εάν κάθε πράκτορας δούλευε ατομικά. Αυτό επιτυγχάνεται χωρίζοντας τους πράκτορες σε τμήματα βάσει του παραπάνω κριτηρίου. Καθένα απ' αυτά τα τμήματα ονομάζεται δομή της συμμαχίας.

1.3 Γιατί είναι σημαντικό να βρεθεί γρήγορος αλγόριθμος δημιουργίας συμμαχιών;

Η δημιουργία συμμαχιών είναι ένα πρόβλημα μεγάλης σημασίας τόσο στην καθημερινή μας ζωή όσο και στην τεχνητή νοημοσύνη. Μέσω αυτής της διαδικασίας είναι δυνατόν αυτόνομοι λογικοί ευφυείς πράκτορες να συνεργαστούν με σκοπό την δημιουργία μιας ομάδας πρακτόρων η μιας συμμαχίας. Ο σκοπός αυτής της διπλωματικής εργασίας είναι η εύρεση του γρηγορότερου καθολικά ή ανά περίπτωση αλγορίθμου δημιουργίας συμμαχιών. Εφόσον η δημιουργία συμμαχιών είναι μια σημαντική διαδικασία είναι σημαντικό να εκτελείτε όσο το δυνατόν γρηγορότερα για μεγάλο αριθμό πρακτόρων ώστε να μπορεί να χρησιμοποιηθεί και σε ρεαλιστικά προβλήματα όπου πιθανώς να απαιτούνται πολλοί πράκτορες ταυτόχρονα. Θέλουμε να εκτελείται σε αρκετά σύντομο χρονικό διάστημα για απαιτητικούς υπολογισμούς γιατί η δημιουργία συμμαχίας είναι ένα απ' τα πρώτα στάδια στην επίλυση του γενικού προβλήματος και ο συνολικός χρόνος επίλυσης του προβλήματος εξαρτάται άμεσα απ' τον χρόνο που χρειάστηκε για την δημιουργία της.

Ένας καθολικά γρηγορότερος αλγόριθμος πιθανόν να είναι δυσκολότερο να βρεθεί και λογικά θα χρειαστεί να βρούμε γρηγορότερο αλγόριθμο για συγκεκριμένες περιπτώσεις προβλημάτων.

Παραδείγματα ρεαλιστικών προβλημάτων που βασίζονται στην δημιουργία συμμαχιών είναι μια ταχυδρομική εταιρία η οποία έχει να παραδώσει γράμματα σε διάφορους προορισμούς διαθέτοντας περιορισμένο αριθμό ταχυδρόμων στην οποία περίπτωση οι ταχυδρόμοι πρέπει να μοιράσουν την δουλειά με τον πιο αποδοτικό τρόπο ώστε να μπορέσουν να παραδώσουν όσο το δυνατόν περισσότερα γράμματα στον ελάχιστο χρόνο, ή μια υπηρεσία που παρέχει μέσα μεταφοράς, π.χ. (ταξί, λεωφορείο, ελικόπτερο,

αεροπλάνο) όπου θα πρέπει σε κάθε περίπτωση ανάλογα με το πλήθος των πελατών και τον προορισμό αυτών να παρέχει το κατάλληλο μέσο μετακίνησης ώστε να φτάσουν όλοι στον προορισμό τους στην ώρα τους και η εταιρία να έχει το μέγιστο κέρδος απ' αυτό, για παράδειγμα για 1-4 πελάτες θα χρησιμοποιούσε ένα ταξί ενώ για 40 πελάτες που ο προορισμός τους ήταν στην άλλη άκρη του ωκεανού θα χρησιμοποιούσε ένα αεροπλάνο.

1.4 Χάρτης Πτυχιακής

Η παρούσα διπλωματική αποτελείτε από τα εξής κεφάλαια:

Κεφάλαιο 1, Εισαγωγή, παρουσιάζεται το βασικό θεωρητικό υπόβαθρο που χρειάζεται για την κατανόηση της διπλωματική εργασίας και απαντάται το ερώτημα «Γιατί είναι σημαντική η εύρεση του γρηγορότερου και αποδοτικότερου αλγορίθμου σχεδιασμού συμμαχιών;»

Κεφάλαιο 2, Σχηματισμός Συμμαχιών, Υπόβαθρο και ορισμοί σχετικά με την δημιουργία συμμαχιών.

Κεφάλαιο 3, Αλγόριθμοι σχηματισμού, παρουσιάζονται οι τέσσερις αλγόριθμοι που θα συγκρίνουμε DP, IP, D-SlyCE & DyCE καθώς και ένα παράδειγμα τρεξίματος για τον καθένα.

Κεφάλαιο 4, Σύγκριση και συζήτηση, Γίνεται η σύγκριση των τεσσάρων αλγορίθμων καθώς και συζήτηση για τα Κεφάλαια 2 και 3.

Κεφάλαιο 5, Συμπεράσματα και μελλοντική έρευνα, παρουσιάζονται τα συμπεράσματα καθώς και προτάσεις για πιθανή μελλοντική έρευνα.

ΚΕΦΑΛΑΙΟ 2: ΔΗΜΙΟΥΡΓΙΑ ΣΥΜΜΑΧΙΩΝ

2.1 Συμμαχίες στην κοινωνία

Στην καθημερινότητα μας πολύ συχνά βλέπουμε άτομα να δημιουργούν συμμαχίες είτε είναι στα πλαίσια μιας ομαδικής εργασίας, δουλειάς ή ακόμα και ενός παιχνιδιού.

Ακολουθούν διάφοροι ορισμοί για τον όρο συμμαχία

«Συμμαχία ονομάζεται ή προσωρινή ένωση μεταξύ δύο ομάδων με σκοπό να κερδίσουν περισσότερη δύναμη ή επιρροή απ' ότι αν οι ομάδες δούλευαν ξεχωριστά. Με το να συγκεντρωθούν σε έναν κοινό στόχο όλα τα μέλη της ομάδας μπορούν να επωφεληθούν απ' τα δυνατά σημεία των υπολοίπων μελών καθώς και να έχουν πλεονέκτημα όσων αφορά τα κοινά ζητήματα. Συνήθως η συμμαχία διαλύεται μόλις επιτευχθεί ο στόχος της.» [13]

«Συμμαχία ονομάζεται μια ομάδα όταν άνθρωποι ή ομάδες ανθρώπων συνεργάζονται με στόχο την επίτευξη ενός κοινού σκοπού. Το να φτιάξεις μια συμμαχία μεταξύ δύο ή περισσότερων ομάδων με κοινές αξίες σκοπούς ή ενδιαφέροντα επιτρέπει την καλύτερη συνεργασία μεταξύ της ομάδας και ίσως να έχει και θετική επιρροή μεταξύ των μελών αυτών των ομάδων απ' το να δούλευαν ξεχωριστά οι δυο ομάδες.» [14]

Μια πολύ σημαντική ερώτηση που πρέπει να λυθεί είναι η εξής:

«Γιατί να δημιουργήσω μια συμμαχία με ένα άλλο άτομο ή ομάδα;»

Όπως αναφέρθηκε και στον ορισμό μια συμμαχία κυρίως δημιουργείται με σκοπό την επίτευξη κάποιου κοινού στόχου. Θα πρέπει να υπάρχουν ξεκάθαροι λόγοι για την δημιουργία μιας συμμαχίας. Βάσει [13] «Είναι διαφορετικό να δημιουργηθεί μια συμμαχία για την προσφορά περισσότερων ψήφων σε ένα κόμμα και άλλο η δημιουργία μιας συμμαχίας για την δημιουργία κυβέρνησης. Σε κάποιες περιπτώσεις αυτές οι δύο συμμαχίες μπορεί να αποτελούνται από ακριβώς τα ίδια άτομα ή ομάδες αλλά οι στόχοι παραμένουν διαφορετικοί.»

Ακολουθούν μερικά ενδεικτικά πλεονεκτήματα και μειονεκτήματα του να είναι κανείς μέλος μιας συμμαχίας:

<u>Πλεονεκτήματα</u>:

- Μια συμμαχία μπορεί να προσφέρει στα μέλη ή τις ομάδες της την δυνατότητα να αναπτυχθούν και να διευρύνουν την βάση υποστήριξής τους.
- Τα μέλη μιας συμμαχίας μπορούν να επικεντρωθούν στις δυνάμεις τους και να στηριχθούν στις δυνάμεις των συνεργατών τους σε άλλους τομείς και να μοιράζονται τα κέρδη.
- Κέρδος γνώσεων και δεξιοτήτων των μελών από άλλα μέλη ή ομάδες.
- Περισσότεροι άνθρωποι μπορούν να ανταπεξέλθουν καλύτερα σε έλλειψη χρόνου και χρημάτων, περισσότερα άτομα θα τελειώσουν μια δουλειά γρηγορότερα και με μικρότερο κόστος με λίγη εθελοντική προσπάθεια. [13]

<u>Μειονεκτήματα</u>:

- Για να βρεθούν κοινοί στόχοι και επομένως κέρδος για όλους τους εμπλεκόμενους θα πρέπει να υπάρξουν συμβιβασμοί. Τα πλεονεκτήματα απ' τους συμβιβασμούς επιβάλλεται να είναι περισσότερα απ' το χάσιμο.
- Με το να είναι μέρος μια συμμαχίας τα μέλη μπορεί να χάσουν τον πλήρη έλεγχο σε κάποιες αποφάσεις, όπως επίσης μπορεί να χάσουν και την ταυτότητά τους κατά την διάρκεια ύπαρξης της συμμαχίας. Αυτό μπορεί να οδηγήσει σε διαφωνίες και διαμάχες μεταξύ των μελών της συμμαχίας.
- Με την συσχέτισή τους με άλλες ομάδες μπορεί να συσχετιστούν και με τον αρνητικό προσανατολισμό των υπολοίπων ομάδων. [13]

Ακολουθούν μερικές δυσκολίες τις οποίες μπορεί να αντιμετωπίσει μια συμμαχία: [13]

- Συγκέντρωση, στόχοι και όρια
  Πολλές φορές μεταξύ συνεργατών στην συμμαχία δεν υπάρχει συμφωνία όσων αφορά τους κοινούς στόχους είτε λόγω παράλειψης ή λόγω κακής συνεννόησης. Αυτήν η ασυμφωνία μπορεί να προκαλέσει διαφωνίες και χάσματα στην συμμαχία
- Κίνητρο
  Κάποια μέλη της συμμαχίας μπορεί να μην έχουν τόσο μεγάλο κίνητρο να ασχοληθούν με τους κοινούς στόχους μέσα στην συμμαχία και έτσι να αποδειχτεί ότι κάποια μέλη κάνουν περισσότερη δουλειά ενώ άλλοι είναι πιο χαλαροί
- Λήψη αποφάσεων
  Πρέπει να υπάρχει κατανόηση μεταξύ μελών της συμμαχίας για το πως παίρνονται αποφάσεις στην συμμαχία καθώς και για το ποια άτομα έχουν δικαιοδοσία σε ποιους πόρους
- Διατήρηση εμπιστοσύνης
  Είναι σημαντικό τα πράγματα μέσα σε μια συμμαχία να είναι φανερά και ξεκάθαρα έτσι ώστε να μην υπάρχουν αμφιβολίες μεταξύ των μελών διότι η ύπαρξη αμφιβολιών μπορεί να διαλύσει μια συμμαχία ακόμα και στην περίπτωση που οι αμφιβολίες είναι μη βάσιμες
- Μοιρασμένο φόρτο εργασίας
  Οι συνεργάτες πρέπει να κατανοούν το γεγονός ότι το κάθε μέλος πρέπει να κάνει αυτό που του αναλογεί και ότι ο καθένας προσφέρει στην συμμαχία ότι παραπάνω μπορεί
- Αδυναμίες
  Όπως και σε κάθε οργανισμό είναι δεδομένο ότι και στην συμμαχία θα υπάρχουν αδυναμίες. Είναι καλό αυτές οι αδυναμίες να είναι γνωστές απ' την αρχή έτσι ώστε να μπορέσουν να βρεθούν τρόποι να αντιμετωπιστούν
- Μηχανισμοί επίλυσης διαφωνιών
  Πρέπει να υπάρχει ένας δεδομένος τρόπος βάσει του οποίου να επιλύονται οι διαφορές μεταξύ μελών της συμμαχίας και είναι καλό αυτός ο τρόπος να καθιερωθεί πριν αρχίσουν να προκύπτουν προβλήματα

- Επικοινωνία

Τα μέλη μιας συμμαχίας πρέπει να αναπτύξουν επικοινωνιακές σχέσεις μεταξύ τους αλλά και με το κοινό

- Αναγνώριση
  Τα μέλη της συμμαχίας είναι σημαντικό να λαμβάνουν την αναγνώριση που τους αξίζει βάσει αυτών που έχουν προσφέρει στην συμμαχία. Σε σπάνιες περιπτώσεις ίσως τα μέλη να θέλουν να προσφέρουν ανώνυμα αλλά στην μεγάλη πλειοψηφία είναι καλό να αναγνωρίζεται και να ανταμείβεται η όποια προσφορά
- Αποφυγή κοινών προτεραιοτήτων
  Διαφορετικοί συνεργάτες μεταξύ μιας συμμαχίας μπορεί να έχουν διαφορετικούς στόχους πολλές φορές μπορεί ακόμα και αυτοί οι ρόλοι να είναι αντικρουόμενοι μεταξύ τους. Είναι σημαντικό όλοι οι συνεργάτες να συμφωνήσουν να έχουν κοινές προτεραιότητες και στόχους και να συνδράμουν αντίστοιχα στην συμμαχία
- Κρυφές ατζέντες
  Μεταξύ συνεργατών είναι υγιές να υπάρχει ειλικρίνεια και όχι να κρύβουν πράγματα απ' τους υπόλοιπους συνεργάτες όπως π.χ. ενδιαφέροντα ή προτεραιότητες στόχων, ή το τι αποσκοπούν να πετύχουν μέσω της συμμαχίας

Μια ακόμα μεγάλη δυσκολία είναι το να διατηρηθεί ζωντανή η συμμαχία. Κάποια απ' τα προβλήματα που αναφέρονται παραπάνω μπορούν να καταστρέψουν μια συμμαχία. Έχουν προταθεί από διάφορους ερευνητές κάποιοι τρόποι που βοηθούν στην συντήρηση μιας συμμαχίας όπως για παράδειγμα στο [13]

- Διατήρηση της εμπιστοσύνης
  Πρέπει να γίνονται συχνή έλεγχοι ότι συνεχίζει να υπάρχει εμπιστοσύνη και άνεση μεταξύ των μελών της συμμαχίας ιδιαίτερα μετά την πρώτη «χαρούμενη» φάση. Επιβεβαίωση ότι όλα τα μέλη βλέπουν τα οφέλη που έχει το να είσαι μέλος μιας συμμαχίας και εργάζονται με σκοπό να πετύχουν τους στόχους της συμμαχίας
- Συναντήσεις
  Πρέπει να διοργανώνονται συχνές συναντήσεις στις οποίες το θέμα συζήτησης να είναι η επίλυση προβλημάτων ή διαμαχών καθώς και μια γενική ενημέρωση του που βρίσκεται η συμμαχία όσων αφορά τους στόχους της
- Εσωτερική Επικοινωνία
  Οι συνεργάτες πρέπει να διατηρούν επικοινωνία μεταξύ τους και με τις διάφορες ομάδες με τις οποίες συνεργάζονται εντός της συμμαχίας
- Εξωτερική Επικοινωνία
  Οι συνεργάτες θα πρέπει να διατηρούν έναν κοινώς αποδεκτό τρόπο επικοινωνίας με το κοινό (πελάτες, εξωτερικούς συνεργάτες)

Επιπροσθέτως στο [17] αναφέρονται τα εξής:

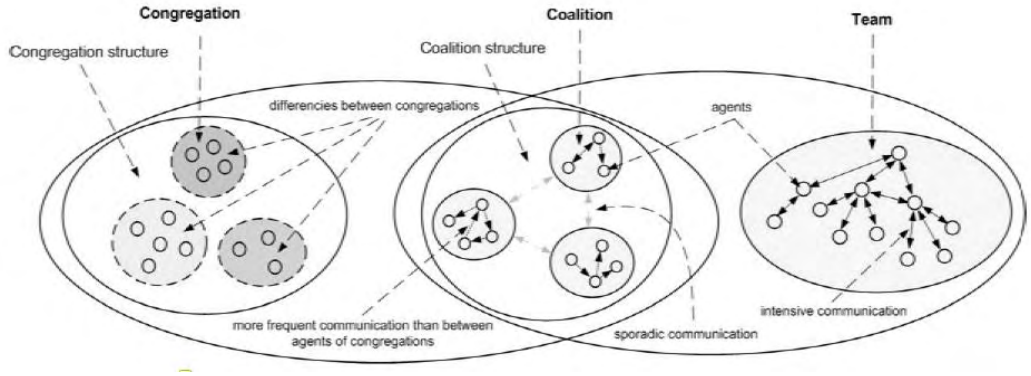
- Γνωστοποίηση προβλημάτων της συμμαχίας
  Παρόλο που πρόωρα σημάδια είναι δύσκολο να αντιμετωπιστούν και μπορεί να είναι βαρετές συναντήσεις, έλλειψη ενθουσιασμού, συνεχείς αποτυχίες, διαμάχες μεταξύ μελών πρέπει να αντιμετωπίζονται με σκοπό την αρμονική

λειτουργία μιας συμμαχίας. Το καλύτερο feedback σε αυτήν την περίπτωση προέρχεται απ' τα ίδια τα μέλη της ομάδας

- Καταμερισμός της ηγεσίας και δύναμης
  Διαπραγμάτευση μη ισορροπίας δύναμης στην λήψη αποφάσεων, ιδιαίτερα όταν μια συμμαχία έχει φτάσει αυτά τα επίπεδα ωριμότητας, απαιτεί ευαισθησία και πιθανώς και λίγο επιπλέον χρόνο ώστε να διευκρινιστεί
- Πρόσληψη νέων ατόμων
  Νέα μέλη προσφέρουν ενθουσιασμό και ενέργεια στις δραστηριότητες μιας συμμαχίας. Είναι ιδιαίτερα σημαντικό να δοθεί η απαραίτητη προσοχή ώστε τα νέα αυτά μέλη να είναι ευπρόσδεκτα και να οδηγηθούν σωστά ως προς την επίτευξη των στόχων
- Ενθάρρυνση αλλαγών, εύρεση νέων ανταγωνιστικών ιδεών για την ομάδα
  Το κάθε μέλος δίνει τον δικό του τρόπο σκέψης στην συμμαχία, επομένως θα πρέπει να υπάρχει απ' την αρχή ένα κοινός τρόπος συνεννόησης και κοινές αρχές που να ακολουθούν όλα τα μέλη. Επίσης θα πρέπει να υπάρχει η δυνατότητα να εκπαιδεύονται τα νέα μέλη ώστε να τους δίνεται η ευκαιρία να καλύψουν το χαμένο έδαφος. Τέλος όλοι μπορούν να επωφεληθούν απ' αυτές τις επανεκπαιδεύσεις, ή τις συζητήσεις για το τι δουλεύει σωστά και τι όχι στην συμμαχία, δεδομένου ότι το να κάνεις μόνο δουλειά για την συμμαχία μπορεί να καταντήσει ιδιαίτερα κουραστικό και να οδηγήσει σε εξάντληση κάποιων μελών
- Ανταμοιβές και αναγνώριση για τις επιτυχίες
  Είναι ιδιαίτερα σημαντικό να υπάρχουν διαλείμματα στα οποία θα επιβραβεύονται οι επιτυχίες της συμμαχίας πράγμα που θα δείξει σε πολλά άτομα ότι η συμμαχία στην οποία είναι μέλη κάνει κάτι σημαντικό. Πολύ συχνά οι συμμαχίες επικεντρώνονται μόνο στην επίλυση προβλημάτων καθώς και στο να προχωρήσουν στο επόμενο βήμα που ξεχνάνε ότι είναι σημαντικές οι σύντομες παύσεις ώστε να εκτιμηθεί η δουλειά που έχει επιτευχθεί μέχρι τώρα

2.2 Συμμαχίες Πρακτόρων

Τα περισσότερα απ' αυτά που είδαμε σχετικά με τις συμμαχίες στην κοινωνία ισχύουν και όταν μιλάμε για συμμαχίες μεταξύ ευφυών πρακτόρων. Σύμφωνα με [16] εκτός από συμμαχίες μεταξύ πρακτόρων μπορούν να δημιουργηθούν ομάδες ή και συγκεντρώσεις. Υπάρχουν κάποια στοιχεία τα οποία διαχωρίζουν τους τρεις αυτούς όρους.



Εικόνα 2.1: Διαφορές μεταξύ συγκέντρωσης, συμμαχίας και ομάδας [16]

Στοιχείο	**Συμμαχία**	**Συγκέντρωση**	**Ομάδα**
Διάρκεια	Κυρίως σύντομη	Μακράς διάρκειας	Κυρίως σύντομη
Στόχος	Ένας κοινός	Διαφορετικοί	Ένας κοινός
Αλληλεπικαλυπτόμενες ομάδες	Πιθανό	Πιθανό	Πιθανό (ρόλοι)
Ελεύθερη είσοδος/έξοδος	Μη κοινό	Συχνό	Μη κοινό
Ιδιότητες των δομών	Ομογενείς ομάδες	Ετερογενείς Ομάδες	Ομογενείς/Ετερογενείς
Συχνότητα επικοινωνίας μεταξύ πρακτόρων σε μια ομάδα	Έμμεση, όχι ιδιαίτερα έντονη	Χωρίς Επικοινωνία	Άμεση ή έμμεση συχνή
Συχνότητα επικοινωνίας μεταξύ ομάδων	Σποραδική	Χωρίς Επικοινωνία	Συχνή

Πίνακας 2.1: Διαφορές μεταξύ συγκέντρωσης, συμμαχίας και ομάδας [16]

Στα πλαίσια αυτής της διπλωματικής εμείς θα ασχοληθούμε μόνο με τις συμμαχίες.

Λόγω των περιορισμένων πόρων ή ικανοτήτων ένας πράκτορας είναι αδύνατον να φέρει εις πέρας πολλά έργα μόνος του. Πολλές φορές όταν μόλις ένας πράκτορας πρέπει μόνος του να φέρει εις πέρας ένα στόχο οι επιδόσεις του είναι αρκετά χαμηλές για να θεωρηθούν αποδεκτές. Σε αυτήν την περίπτωση οι πράκτορες καλούνται να δημιουργήσουν ομάδες, γνωστές ως συμμαχίες έτσι ώστε να λύσουν το πρόβλημα γρηγορότερα σε συνεργασία με άλλους πράκτορες. [7]

Όπως για τους πράκτορες έτσι και για τις συμμαχίες δεν υπάρχει κάποιος καθολικός ορισμός, ακολουθούν διάφοροι ορισμοί οι οποίοι ορίζουν πλήρως μια συμμαχία. Αρχικά ο ορισμός που φαίνεται ο πιο ολοκληρωμένος απ' όλους από τους Shehory και Kraus

«Συμμαχίες πρακτόρων ονομάζονται οι ομάδες που δημιουργούνται από τους πράκτορες με σκοπό την επίτευξη κάποιου στόχου που είτε δεν είναι δυνατόν να ολοκληρωθεί ατομικά από έναν πράκτορα ή δεν μπορεί να επιτευχθεί ικανοποιητικά. Επίσης αν δύο συμμαχίες δοκιμάσουν να επιτύχουν τον ίδιο στόχο τα αποτελέσματά τους θα διαφέρουν λόγω των διαφορετικών ικανοτήτων των μελών της συμμαχίας. Σκοπός δημιουργίας μιας συμμαχίας είναι η μεγιστοποίηση των οφειλών μέσω των επιδόσεων των μελών της» [8]

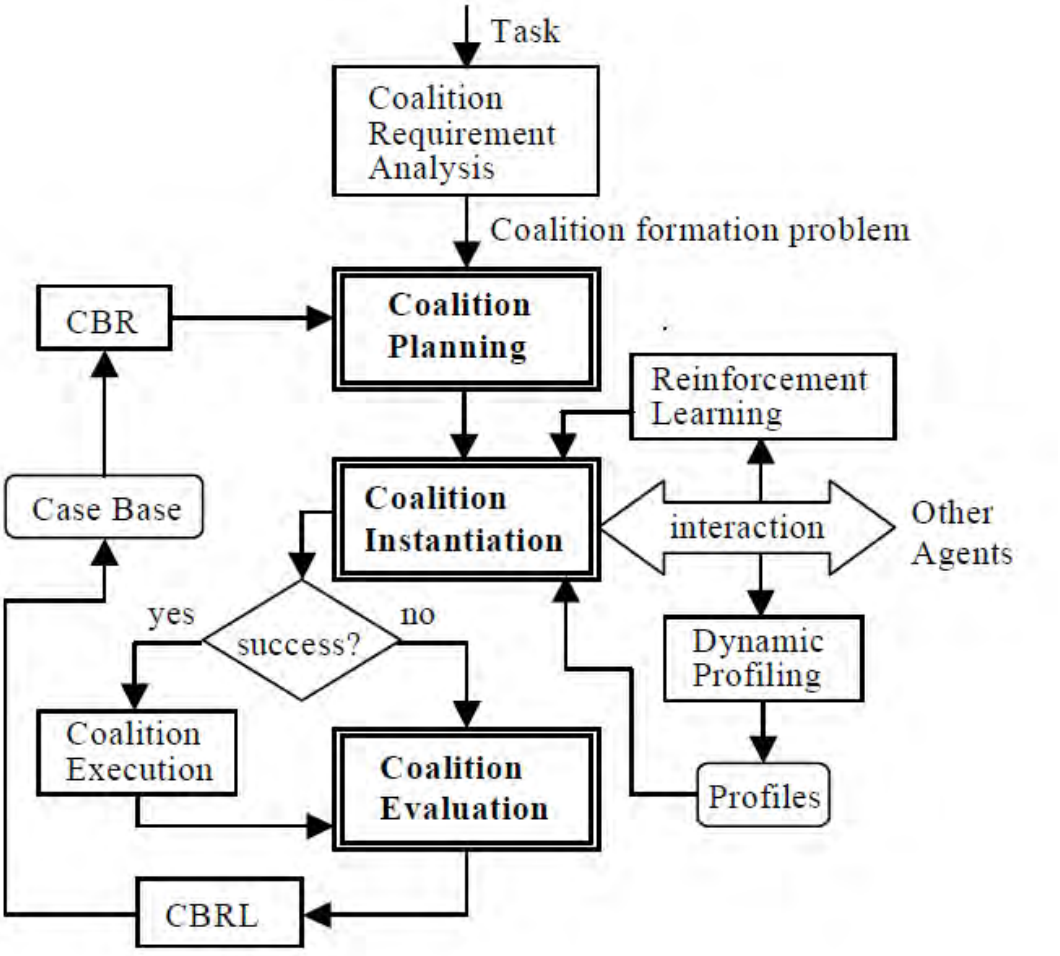
Ακολουθούν και διάφοροι άλλοι ορισμοί για τον όρο Συμμαχία Πρακτόρων

«Συμμαχία πρακτόρων είναι μια ομάδα που αποτελείτε από πράκτορες που συνεργάζονται μεταξύ τους δουλεύοντας μαζί με σκοπό την επίτευξη ενός στόχου, χαρακτηρίζονται από σύντομη διάρκεια ζωής και οδηγούνται καθαρά από τον στόχο για τον οποίο δημιουργήθηκαν, αυτός τις καθιστά μια επίπεδη δομή.» [7]

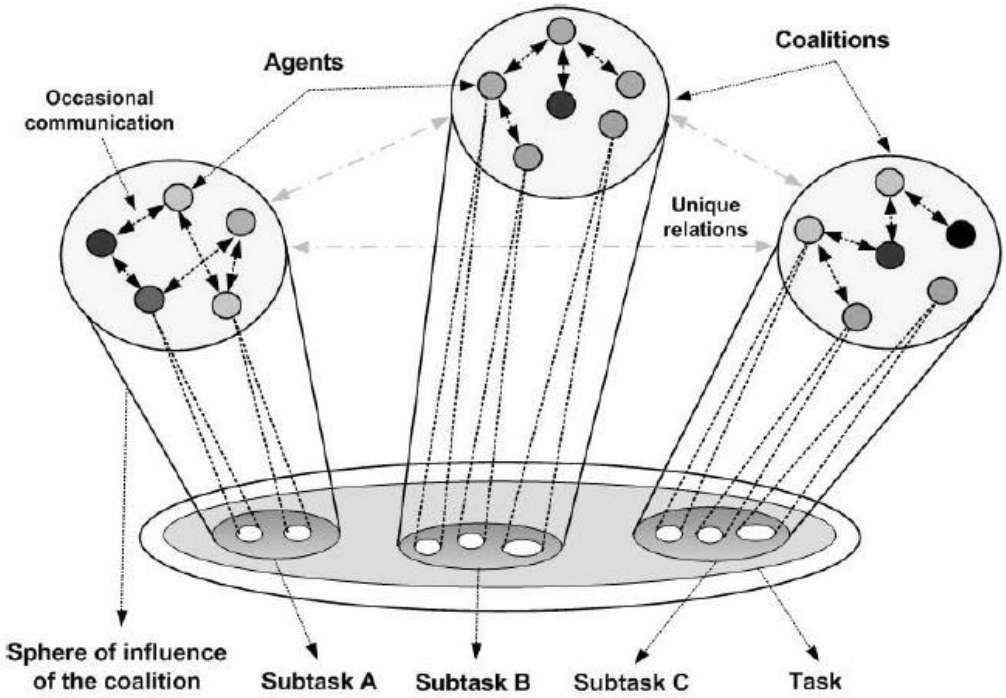
«Συμμαχία πρακτόρων είναι μια προσωρινή σχέση μεταξύ συνεργαζόμενων πρακτόρων με σκοπό να επιτευχθεί από κοινού ένα περίπλοκο πρόβλημα. Ο στόχος είναι η καλύτερη κατανομή των ικανοτήτων των πρακτόρων έτσι ώστε μέσω της συνεργασίας τους να μπορέσει αποτελεσματικά να επιτευχθεί η λύση του περίπλοκου προβλήματος.» [9]

«Δεδομένου ενός κόσμου με πολλές εσωτερικές καταστάσεις μια εκ των οποίων είναι η αρχική κατάσταση του κόσμου. Δεδομένου ότι σε αυτό τον κόσμο υπάρχει ένα σύνολο n πρακτόρων και μιας συνάρτησης για τον κόσμο η οποία έχει δύο ορίσματα το πρώτο είναι η τωρινή κατάσταση του κόσμου s_i ενώ το δεύτερο είναι το σετ δράσεων για κάθε έναν από τους n πράκτορες $<a_1, \dots, a_n>$. Η συνάρτηση του κόσμου επιστρέφει την νέα εσωτερική κατάσταση η οποία προκύπτει απ' τα δύο ορίσματα. Αν λοιπόν ορίσουμε ως s_0 την αρχική κατάσταση τότε για την κατάσταση i+1 έχουμε: $s_{i+1} = \text{World}(s_i, <a_1(i), \dots, a_n(i)>)$ όπου το $a_j(i)$ είναι η ενέργεια του j-στου πράκτορα την στιγμή i. Όταν μια ομάδα πρακτόρων έχει μια στρατηγική για να επιτευχθεί ή να διαχειριστεί μια συγκεκριμένη συνθήκη έχουμε την δημιουργία μιας συμμαχίας μεταξύ αυτών των πρακτόρων.» [10]

Ακολουθούν μερικές γραφικές αναπαραστάσεις δημιουργίας συμμαχιών.



Εικόνα 2.2: Μοντέλο Πολυφασικής Δημιουργίας Συμμαχιών MPCF βασισμένο στην μάθηση [11]



Εικόνα 2.3: Σχηματισμός συμμαχίας [16]

2.2.1 Δραστηριότητες συμμαχιών

Σε κάθε συμμαχία συναντώνται οι εξής δραστηριότητες [12]:

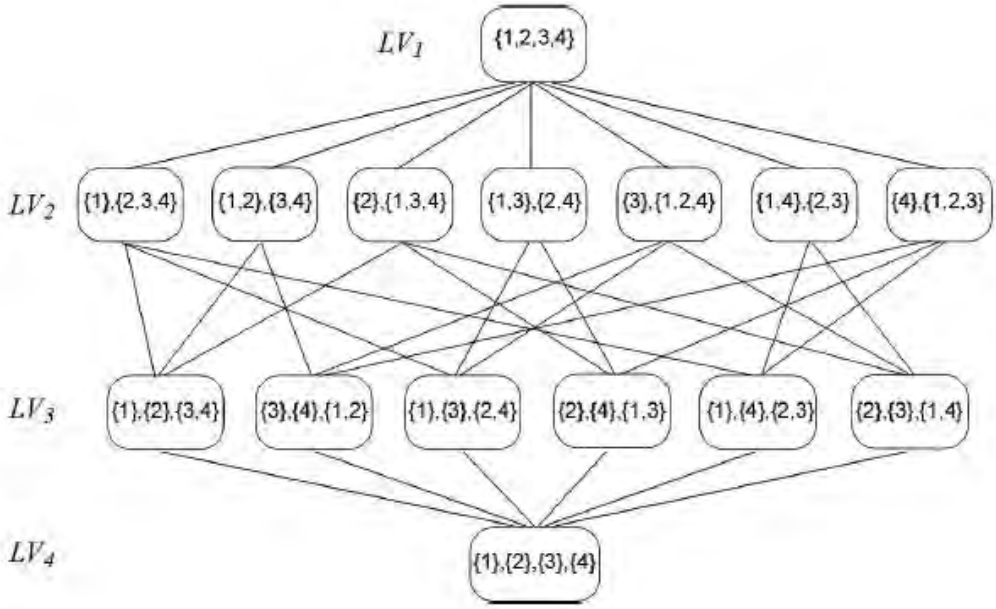
- Παραγωγή δομής συμμαχίας
- Λύση του προβλήματος βελτιστοποίησης
- Διαίρεση της αξίας της παραγόμενης λύσης μεταξύ των πρακτόρων

Παραγωγή δομής συμμαχίας: Οι πράκτορες συμμετέχουν σε μια συμμαχία και συντονίζουν τις δραστηριότητες τους σε αυτήν την συμμαχία και όχι μεταξύ άλλων συμμαχιών. Δομή συμμαχίας ονομάζεται η διαίρεση του συνόλου πρακτόρων λαμβάνοντας υπόψιν κάθε μέλος του συνόλου χωρίς διπλότυπα, για παράδειγμα εάν έχουμε τρεις πράκτορες {1}, {2}, {3} έχουμε εφτά πιθανές συμμαχίες τις εξής:

({1}, {2}, {3}, {1,2}, {2,3}, {3,1}, {1,2,3})

και πέντε πιθανές δομές:

{{1}, {2}, {3}}, {{1}, {2,3}}, {{2}, {3,1}}, {{3}, {1,2}}, {{1,2,3}}



Εικόνα 2.4: Γράφημα Δομής συμμαχίας για τέσσερις πράκτορες [7]

Λύση του προβλήματος βελτιστοποίησης: Συγκέντρωση των πόρων και εργασιών κάθε πράκτορα με σκοπό την από κοινού επίλυση του συλλογικού προβλήματος. Στόχος της συμμαχίας είναι η μεγιστοποίηση αξίας: κέρδη που έχει από εξωτερικούς παράγοντες λόγω της ολοκλήρωσης εργασιών μείον τα κόστη χρήσης πόρων. Σε κάποια προβλήματα μπορεί να μην χρειάζεται να χειριστούν όλες οι εργασίες στην οποία περίπτωση θα πρέπει να αναθέσουμε σε κάθε εργασία και ένα κόστος.

Διαίρεση της αξίας της παραγόμενης λύσης μεταξύ των πρακτόρων: Αυτήν η αξία μπορεί να είναι και αρνητική λόγω του ότι για την χρήση πόρων οι πράκτορες υφίστανται κόστη.

Αυτές οι δραστηριότητες μπορεί να είναι παρεμβαλλόμενες και όχι απαραίτητα ανεξάρτητες για παράδειγμα το πόσο θέλει ένας πράκτορας να γίνει μέλος μιας συμμαχίας εξαρτάται απ' το ποσοστό της αξίας που θα αντιστοιχιστεί στον πράκτορα.

2.3 Υπολογισμός προσφοράς αξίας κάθε πράκτορα στην συμμαχία

Ένα πολύ σημαντικό κομμάτι που πρέπει να λαμβάνεται υπόψιν όταν δημιουργείται η δομή μιας συμμαχίας, είναι η συγκεκριμένη δομή να είναι η αποδοτικότερη δομή. Αυτό μπορούμε να το ελέγξουμε μέσω μιας μεταβλητής που ονομάζεται "Coalitional Value" και είναι ουσιαστικά η απάντηση στην ερώτηση «Τι προσφέρει ένας πράκτορας στην συμμαχία;» Θα πρέπει λοιπόν να υπάρχει ένας αποδοτικός αλγόριθμος ο οποίος θα είναι υπεύθυνος για τον υπολογισμό αυτής της τιμής για κάθε πιθανό συνδυασμό μεταξύ των ενδεχόμενων μελών της συμμαχίας. Προφανώς αυτό είναι μια δύσκολη διαδικασία και αυξάνεται εκθετικά όσο μιλάμε για περισσότερους πράκτορες στην συμμαχία. Για παράδειγμα για μια πιθανή συμμαχία μεταξύ τριών πρακτόρων {a1, a2, a3} υπάρχουν μόλις πέντε πιθανές δομές για τις οποίες πρέπει να υπολογιστεί η

παραπάνω τιμή. Όσο περισσότερο αυξάνεται ο αριθμός των πρακτόρων, ο αριθμός των πιθανών δομών μεταξύ τους αυξάνεται εκθετικά βάσει του τύπου:

$$Αριθμός\ Δομών = 2^{Αριθμός\ Πρακτόρων} - 1$$

Στο [18] παρουσιάζονται τρεις εκδοχές ενός αλγορίθμου του DCVC ο οποίος είναι υπεύθυνος για τον υπολογισμό και απόδοση του "Coalitional Value" στους πράκτορες μια συμμαχίας. Αρχικά η βασική εκδοχή αυτού του αλγορίθμου στην οποία θεωρούμε ως δεδομένο το ότι οι διαφορές στην χρήση των πόρων απ' τους πράκτορες είναι οι ελάχιστες δυνατές, έπειτα παρουσιάζει το πως αυτός ο χρόνος μπορεί να μειωθεί ορίζοντας ποιες τιμές θα υπολογίζει κάθε πράκτορας και τέλος δείχνει πως μπορεί να τροποποιηθεί ώστε να λαμβάνει υπόψιν και διαφορετικές ταχύτητες υπολογισμού απ' τον κάθε πράκτορα.

L_1	L_2	L_3	L_4	L_5	L_6
6	5, 6	4, 5, 6	3, 4, 5, 6	2, 3, 4, 5, 6	1, 2, 3, 4, 5, 6
5	4, 6	3, 5, 6	2, 4, 5, 6	1, 3, 4, 5, 6	
4	4, 5	3, 4, 6	2, 3, 5, 6	1, 2, 4, 5, 6	
3	3, 6	3, 4, 5	2, 3, 4, 6	1, 2, 3, 5, 6	
2	3, 5	2, 5, 6	2, 3, 4, 5	1, 2, 3, 4, 6	
1	3, 4	2, 4, 6	1, 4, 5, 6	1, 2, 3, 4, 5	
	2, 6	2, 4, 5	1, 3, 5, 6		
	2, 5	2, 3, 6	1, 3, 4, 6		
	2, 4	2, 3, 5	1, 3, 4, 5		
	2, 3	2, 3, 4	1, 2, 5, 6		
	1, 6	1, 5, 6	1, 2, 4, 6		
	1, 5	1, 4, 6	1, 2, 4, 5		
	1, 4	1, 4, 5	1, 2, 3, 6		
	1, 3	1, 3, 6	1, 2, 3, 5		
	1, 2	1, 3, 5	1, 2, 3, 4		
		1, 3, 4			
		1, 2, 6			
		1, 2, 5			
		1, 2, 4			
		1, 2, 3			

Εικόνα 2.5: Διαχωρισμός πιθανών συμμαχιών 6 πρακτόρων σε επίπεδα με τον DCVC [18]

Όπως αναλύεται στα [18] και [19] σχετικά με τον DCVC αλγόριθμο, χωρίζουμε το σύνολο των συμμαχιών σε επιμέρους επίπεδα/υποσύνολα καθένα απ' τα οποία περιέχει πιθανές συμμαχίες ενός συγκεκριμένου μεγέθους. Αυτό έχει δύο πλεονεκτήματα:

- Αύξηση του μεγέθους της συμμαχίας συνήθως συνεπάγεται περισσότερες δράσεις απ' τους πράκτορες ώστε να υπολογιστεί η αξία. Στην συγκεκριμένη περίπτωση, διανέμοντας ακριβώς τον ίδιο αριθμό συμμαχιών σε κάθε πράκτορα, όχι μόνο υπολογίζεται ο ίδιος αριθμός "Coalitional Value", αλλά εκτελείται και ο ίδιος αριθμός δράσεων

- Μπορούν να μπουν και όρια σχετικά με το μέγεθος της επιθυμητής συμμαχίας. Για παράδειγμα αν δεν θέλουμε να δημιουργηθεί συμμαχία κάποιου μεγέθους έστω X οι πράκτορες δεν διανέμουν συμμαχίες μεγέθους X μεταξύ τους. Σε αυτές τις περιπτώσεις υπολογίζεται ο ίδιος αριθμός "Coalitional Value" αλλά κάνει τον DCVC να ισχύει για αλγορίθμους δημιουργίας συμμαχιών με μειωμένο χρόνο αναζήτησης περιορίζοντας το μέγεθος της συμμαχίας

Παρακάτω παρουσιάζονται οι αλγόριθμοι για δύο εκδοχές του DCVC, η βασική εκδοχή του, καθώς και αυτήν που λαμβάνει υπόψιν και διαφορετικούς χρόνους υπολογισμού από πράκτορα σε πράκτορα

Each agent a_i should perform the following:

- Sort the set of agents based on the agents' UID in an ascending order.
- Set: $\alpha = 1$.
- For every $s \in S$, do the following:

1. If $(N_s \geq n)$ then:

1.1. Calculate the size of your share: $N_{s,i} = \lfloor N_s/n \rfloor$

1.2. Calculate the index of the last coalition in your share: $index_{s,i} = i \times N_{s,i}$

1.3. Calculate the values of each coalition in your share.

1.4. Calculate the number of additional values that need to be calculated: $N' = N_s - (n \times N_{s,i})$

Otherwise:

1.5. Calculate the number of additional values that need to be calculated: $N' = N_s$

2. If $(N' > 0)$ then:

2.1. Find the sequence of agents A' in which each agent should calculate one additional value, and if you are a member of A', then calculate the appropriate value. This is done as follows:

- If $(\alpha + N' - 1 \leq n)$ then: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_{\alpha+N'-1})$
  else: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_n, a_1, \ldots, a_{(\alpha+N'-1)-n})$
- If $(a_i \in A')$ then calculate one of the additional values based on your position in A'
- If $(\alpha + N' \leq n)$ then: $\alpha = \alpha + N'$, else: $\alpha = \alpha + N' - n$

Εικόνα 2.6: Η βασική εκδοχή του αλγορίθμου DCVC [18]

Each agent a_i should first perform the following once:

- Sort the set of agents based on the agents' UID.
- Set: $\alpha = 1$.
- If $(equal\_computational\_speeds = false)$ then:
  1. Calculate t_i, and send it to every other agent.

For every coalition that needs to be formed, each agent a_i should perform the following:

- For every $(s \in S, s \leq n)$ do the following:

1. If $(N_s^* \geq n)$ then:

1.1. If $(equal\_computational\_speeds)$:

- Calculate the size of your share: $N_{s,i} = \lfloor N_s^*/n \rfloor$
- Calculate the size of both sub-lists: $N_{s,i}^1 = \lfloor N_{s,i} \times 0.4 \rfloor, N_{s,i}^2 = \lceil N_{s,i} \times 0.6 \rceil$
- Calculate the number of additional values that need to be calculated: $N' = N_s^* - n \times N_{s,i}$
- Calculate the index of the last coalition of each sub-list: $index_{s,i}^1 = i \times N_{s,i}^1$
  $index_{s,i}^2 = N_s^* - N' - ((i-1) \times N_{s,i}^2)$

Otherwise:

- Calculate the size of every agent's share: $N_{s,j} = \left\lfloor \frac{N_s^*}{t_j \times \sum_{k=1}^{n} 1/t_k} \right\rfloor : j = 1, \ldots, n$
- Calculate the size of both sub-lists: $N_{s,j}^1 = \lfloor N_{s,j} \times 0.4 \rfloor, N_{s,j}^2 = \lceil N_{s,j} \times 0.6 \rceil : j = 1, \ldots, n$
- Calculate the number of additional values that need to be calculated: $N' = N_s^* - \sum_{j=1}^{n} N_{s,j}$
- Calculate the index of the last coalition of each sub-list: $index_{s,i}^1 = \sum_{j=1}^{i} N_{s,j}^1$
  $index_{s,i}^2 = N_s^* - N' - \sum_{j=1}^{i-1} N_{s,j}^2$

1.2. Calculate the values of each coalition in your share.

Otherwise:

1.3. Calculate the number of additional values that need to be calculated: $N' = N_s^*$

2. If $(N' > 0)$ then:

2.1. Find the sequence of agents A' in which each agent should calculate one additional value, and if you are a member of A', then calculate the appropriate value. This is done as follows:

- If $(\alpha + N' - 1 \leq n)$ then: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_{\alpha+N'-1})$
  else: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_n, a_1, \ldots, a_{(\alpha+N'-1)-n})$
- If $(a_i \in A')$ then calculate one of the additional values based on your position in A'
- If $(\alpha + N' \leq n)$ then: $\alpha = \alpha + N'$, else: $\alpha = \alpha + N' - n$

Εικόνα 2.7: Η τελική εκδοχή του αλγορίθμου DCVC, διαφορετικές ταχύτητες υπολογισμού μεταξύ πρακτόρων [18]

Αντίθετα με άλλους αλγόριθμους, με τους οποίους μπορεί να έχουμε έως και εκθετικά μεγάλο αριθμό περιττών υπολογισμών, με τον αλγόριθμο DCVC δεν υπάρχουν περιττοί υπολογισμοί διότι κάθε πράκτορας ξέρει ακριβώς τα όρια των υπολογισμών που πρέπει να εκτελέσει.

ΚΕΦΑΛΑΙΟ 3: ΑΛΓΟΡΙΘΜΟΙ ΣΧΗΜΑΤΙΣΜΟΥ ΣΥΜΜΑΧΙΩΝ

Ο σχηματισμός συμμαχιών είναι μια σημαντική διαδικασία και σε κάποιες περιπτώσεις πιθανώς ακόμα και αναγκαία, ώστε να φέρουν εις πέρας οι πράκτορες έναν κοινό στόχο που υπό άλλες συνθήκες δεν θα μπορούσε να ολοκληρωθεί. Έτσι λοιπόν η ύπαρξη ενός γρήγορου ακριβή και αποτελεσματικού αλγορίθμου είναι σημαντική. Στα πλαίσια αυτής της διπλωματικής θα μελετήσουμε τους παρακάτω αλγορίθμους:

- DP (Dynamic Programming)
- IP (Integer Partitioning)
- D-SlyCE (Sequentially connected Coalition Enumeration)
- DyCE (Dynamic programming for optimal connected Coalition structure Evaluation)

3.1 DP Algorithm

Ο αλγόριθμος βάσει [20] δίνει σαν είσοδο σε κάθε πιθανή συμμαχία μια τιμή v[C], αν αυτήν η τιμή δεν οριστεί θεωρείτε v[C] = 0. Στην συνέχεια υπολογίζει δύο πίνακες, τους f1 και f2. Για να υπολογίσει το f1[C] συγκρίνει την αξία της συμμαχίας (την v[C] της συμμαχίας) με την τιμή του κάθε πράκτορα μέλους της συμμαχίας σε περίπτωση που δούλευε μόνος του. Η μεγαλύτερη απ' τις δύο τιμές είναι και η αποδοτικότερη και το f1[C] ισούται με το αν θα δημιουργηθεί τελικά συμμαχία μεταξύ αυτών των πρακτόρων ή αν θα δουλέψουν μόνοι τους. Το f2[C] αντίθετα είναι η αποδοτικότερη απ' τις δυο τιμές. Αυτός ο αλγόριθμος δεν θα ξεκινήσει να εκτελεί πράξεις για συμμαχίες μεγέθους τριών πρακτόρων και πάνω πριν να τελειώσει με όλες τις συμμαχίες δύο πρακτόρων. Ακολουθεί ένας πίνακας με παραδείγματα υπολογισμού των τιμών f1 και f2 για τέσσερις πράκτορες, δεδομένου ότι A = {1, 2, 3, 4}. Από [21] φαίνεται ότι ο αλγόριθμος DP έχει πολυπλοκότητα $O(3^n)$, όπου n είναι ο αριθμός των πρακτόρων.

size	Coalition	The evaluations that are performed before setting f_1 and f_2	f_1	f_2
1	{1}	v[{1}]=30	{1}	30
	{2}	v[{2}]=40	{2}	40
	{3}	v[{3}]=25	{3}	25
	{4}	v[{4}]=45	{4}	45
2	{1,2}	v[{1,2}]=50 f_2[{1}]+f_2[{2}]=70	{1} {2}	70
	{1,3}	v[{1,3}]=60 f_2[{1}]+f_2[{3}]=55	{1,3}	60
	{1,4}	v[{1,4}]=80 f_2[{1}]+f_2[{4}]=75	{1,4}	80
	{2,3}	v[{2,3}]=55 f_2[{2}]+f_2[{3}]=65	{2} {3}	65
	{2,4}	v[{2,4}]=70 f_2[{2}]+f_2[{4}]=85	{2} {4}	85
	{3,4}	v[{3,4}]=80 f_2[{3}]+f_2[{4}]=70	{3,4}	80
3	{1,2,3}	v[{1,2,3}]=90 f_2[{1}]+f_2[{2,3}]=95 f_2[{2}]+f_2[{1,3}]=100 f_2[{3}]+f_2[{1,2}]=95	{2} {1,3}	100
	{1,2,4}	v[{1,2,4}]=120 f_2[{1}]+f_2[{2,4}]=115 f_2[{2}]+f_2[{1,4}]=110 f_2[{4}]+f_2[{1,2}]=115	{1,2,4}	120
	{1,3,4}	v[{1,3,4}]=100 f_2[{1}]+f_2[{3,4}]=110 f_2[{3}]+f_2[{1,4}]=105 f_2[{4}]+f_2[{1,3}]=105	{1} {3,4}	110
	{2,3,4}	v[{2,3,4}]=115 f_2[{2}]+f_2[{3,4}]=120 f2[{3}]+f_2[{2,4}]=110 f_2[{4}]+f_2[{2,3}]=110	{2} {3,4}	120
4	{1,2,3,4}	v[{1,2,3,4}]=140 f_2[{1}]+f_2[{2,3,4}]=150 f_2[{2}]+f_2[{1,3,4}]=150 f_2[{3}]+f_2[{1,2,4}]=145 f_2[{4}]+f_2[{1,2,3}]=145 f_2[{1,2}]+f_2[{3,4}]=150 f_2[{1,3}]+f_2[{2,4}]=145 f_2[{1,4}]+f_2[{2,3}]=145	{1,2} {3,4}	150

Εικόνα 3.1: Υπολογισμός τιμών f1[C] και f2[C] για κάθε πιθανή συμμαχία με τέσσερις πράκτορες [20, 22]

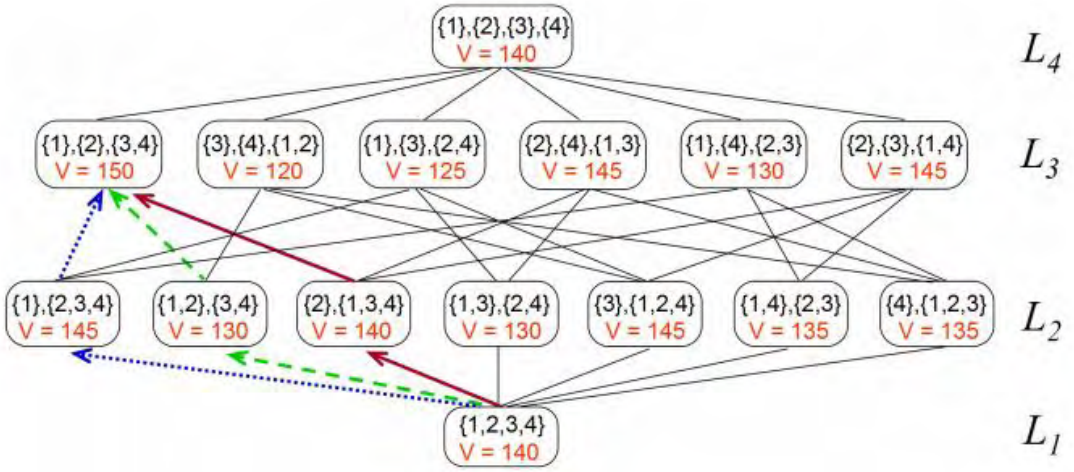
Για παράδειγμα, δίνεται σαν είσοδος ότι το v[{1,2}] = 50 και γνωρίζουμε ότι v[{1}] = 30 και v[{2}] = 40 επομένως το v[{1\} + {2}] = 30 + 40 => v[{1}, {2}] = 70 και συγκρίνοντας τις δύο τιμές v[{1, 2}] & v[{1}, {2}] έχουμε f1 = {1}, {2} και f2 = 70

Επομένως οι πράκτορες {1} και {2} είναι προτιμότερο να δουλέψουν χώρια απ' ότι να δημιουργήσουν συμμαχία μεταξύ τους. Αντίθετα για την περίπτωση του {1}, {3}. Έχουμε δεδομένο ότι v[{1, 3}] = 60 και επίσης ότι v[{1}] = 30, v[{3}] = 25 επομένως v[{1}, {3}] = 55. Συγκρίνοντας τα βρίσκουμε ότι v[{1, 3}] > v[{1}, {3}], άρα f1 = {1, 3} και f2 = 60. Συνεχίζει να υπολογίζει αυτό για κάθε πιθανή συμμαχία τελειώνοντας με όλες τις πιθανές συμμαχίες μεγέθους δύο προτού ξεκινήσει να υπολογίζει τιμές για συμμαχίες μεγέθους >= 3. Τέλος, αφού υπολογιστεί f1, f2 για κάθε πιθανή συμμαχία, η βέλτιστη δομή συμμαχίας CS* υπολογίζεται αναδρομικά όπως φαίνεται παρακάτω.

Input: $v[C]$ for all $C \subseteq A$. If no $v[C]$ is specified then $v[C] = 0$.
Output: the optimal coalition structure, CS^*.

1. For all $i \in \{1, \ldots, n\}$, set $f_1[\{a_i\}] := \{a_i\}, f_2[a_i] := v[\{a_i\}]$
2. For $s := 2$ to n, do:
   For all $C \subseteq A$ such that $|C| = s$, do:
   2.1. $f_2[C] := max\{f_2[C'] + f_2[C \backslash C'] : C' \subset C$ and $1 \leq |C'| \leq 1/2\,|C|\}$
   2.2. If $f_2[C] \geq v[C]$, then set $f_1[C] := C^*$ where C^* maximizes the right hand side of the equation in step 2.1.
   2.3. If $f_2[C] < v[C]$, then set $f_1[C] := C$, and $f_2[C] := v[C]$.
3. Set $CS^* := \{A\}$.
4. For every $C \in CS^*$, do:
   If $f_1[C] \neq C$, then:
   4.1. Set $CS^* := (CS^* \backslash \{C\}) \cup \{f_1[C], S \backslash f_1[C]\}$.
   4.2. Goto 4 and start with new CS^*.

Εικόνα 3.2: Αλγόριθμος σχηματισμού δομής συμμαχίας DP [20, 22]



Εικόνα 3.3: Γράφημα δομής συμμαχίας για τέσσερις πράκτορες συμπεριλαμβανομένης και της αξίας κάθε συμμαχίας [20, 22]

Παράδειγμα εκτέλεσης του αλγορίθμου DP για 3 πράκτορες

Έστω A = { a1, a2, a3 }

Πρώτο βήμα είναι απόδοση τιμής v σε κάθε πιθανή συμμαχία όπως φαίνεται στον πίνακα που ακολουθεί:

Μέγεθος / Συμμαχίες			
1	V{a1} = 30	V{a2} = 40	V{a3} = 25
2	V{a1, a2} = 50 V{a1, a3} = 60	V{a2, a3} = 55	

3	V{a1, a2, a3} = 50		

Αυτές λοιπόν είναι οι αρχικές μας τιμές. Στην συνέχεια πρέπει να υπολογίσουμε το αν οι πράκτορες μας βολεύει να δουλεύουν μαζί ή ο καθένας μόνος του. Για να το υπολογίσουμε αυτό πρέπει να δούμε τι αξία προσφέρει ο κάθε πράκτορας μόνος του σε μια συμμαχία όπως φαίνεται παρακάτω. Σε αυτό το βήμα δεν υπολογίζουμε κάτι για συμμαχίες μεγέθους |C| = 1.

|C| = 2

f({a1} , {a2}) = V{a1} + V{a2} = 30 + 40 = 70 > V{a1, a2} => BestSplit = { {a1} , {a2} }

f({a1} , {a3}) = V{a1} + V{a3} = 30 + 25 = 55 < V{a1, a3} => BestSplit = { {a1, a3} }

f({a2} , {a3}) = V{a2} + V{a3} = 40 + 25 = 65 > V{a2, a3} => BestSplit = { {a2} , {a3} }

|C| = 3

f({a1}, {a2, a3}) = V{a1} + V{a2, a3} = 30 + 65 = 95 > V{a1, a2, a3}

f({a2}, {a1, a3}) = V{a2} + V {a1, a3} = 40 + 60 = 100 > V{a1, a2, a3} =>

f({a3}, {a1, a2}) = V{a3} + V{a1, a2} = 25 + 70 = 95 > V{a1, a2, a3}

Απ' τα παραπάνω 3 βγαίνει ότι ο τρόπος που μας βολεύει να δημιουργηθεί η συμμαχία είναι BestSplit = { {a2}, {a1, a3} }

Για την τελική μας απάντηση πρέπει να απαντήσουμε σε ένα τελευταίο ερώτημα. Το αποτέλεσμα μας δείχνει ότι ο καλύτερος διαχωρισμός είναι BestSplit = { {a2}, {a1, a3} }.

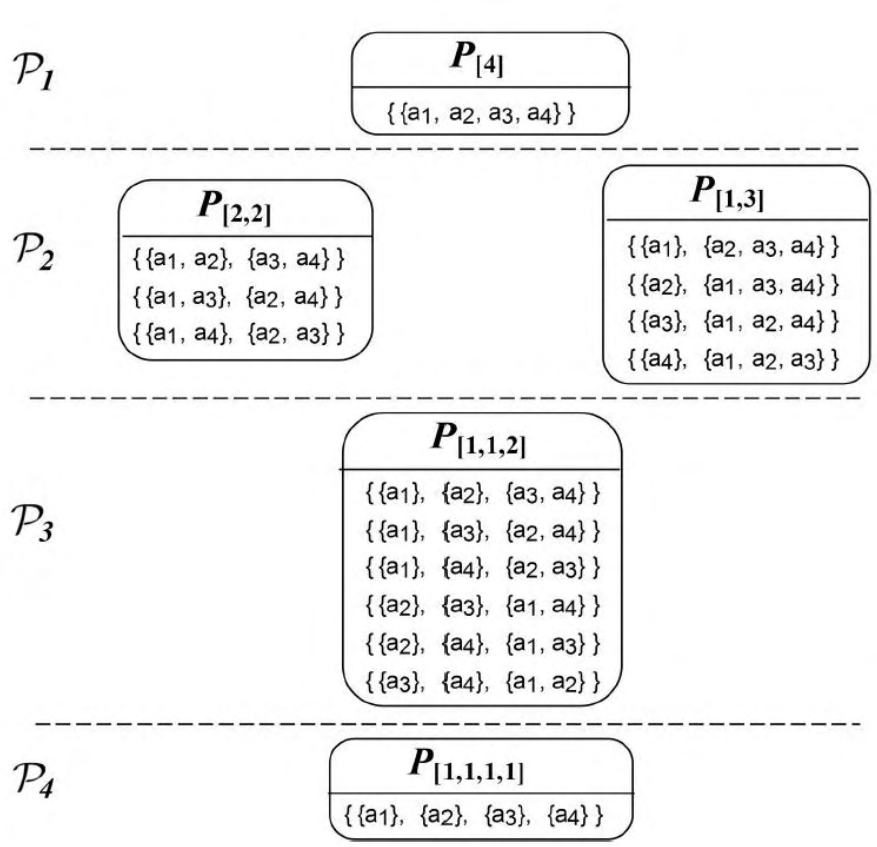
Πρέπει να ελέγξουμε αν μας βολεύει ο {a1, a3} ή ο {a1}, {a3}. Αναδρομικά μπορούμε να δούμε ότι μας βολεύει να συνεργαστούν ο a1 με τον a3 και επομένως ο καλύτερος διαχωρισμός είναι πράγματι ο BestSplit = { {a2}, {a1, a3} }.

Όπως φαίνεται απ' το παράδειγμα εκτέλεσης για 3 πράκτορες αυτήν η μέθοδος είναι αποτελεσματική αλλά απαιτεί πολλές πράξεις, ιδιαίτερα όσο αυξάνεται ο αριθμός των πρακτόρων, στην οποία περίπτωση οι πράξεις αυξάνονται εκθετικά.

3.2 IP Algorithm

Αφού τρέξαμε το παράδειγμα του DP φάνηκε ότι επιστρέφει σίγουρα την βέλτιστη λύση, αλλά χρειάζεται να εκτελέσει πολλές πράξεις ακόμα και για μικρό αριθμό πρακτόρων. Αυτό έρχεται να το διορθώσει ο αλγόριθμος IP (Integer Partition) του οποίου η βασική ιδέα είναι να μην υπολογίζει τις παραπάνω τιμές για όλες τις πιθανές συμμαχίες και επομένως να μην εκτελεί περιττές πράξεις. Το πρώτο βήμα αυτού του αλγορίθμου είναι ο διαχωρισμός του χώρου αναζήτησης. Από [23] λογική με την οποία διαχωρίζουμε τον χώρο βασίζεται σε ακέραιες διαμερίσεις του αριθμού των πρακτόρων

n. Για παράδειγμα για 4 πράκτορες {a1, a2, a3, a4} έχουμε τις εξής διαμερίσεις [4], [1, 3], [2, 2], [1, 1, 2], [1, 1, 1, 1]. Έτσι η συμμαχία {a1, a2, a3, a4} κατατάσσεται στο ακέραιο διαμέρισμα [4] ενώ η { {a1}, {a2}, {a3, a4} } στο ακέραιο διαμέρισμα [1, 1, 2] λόγω του ότι περιέχει δύο συμμαχίες μεγέθους 1 καθώς και μία μεγέθους 2. Στην παρακάτω εικόνα φαίνεται ο πλήρης διαχωρισμός του χώρου για το πρόβλημα των τεσσάρων πρακτόρων καθώς και ποιοι ακριβώς συμμετέχουν σε κάθε διαμέρισμα.



Εικόνα 3.4: Διαχωρισμός χώρου αναζήτησης για το πρόβλημα των τεσσάρων πρακτόρων. [23]

Αφού ολοκληρωθεί ο διαχωρισμός στα ακέραια διαμερίσματα του χώρου αναζήτησης το επόμενο βήμα είναι ο υπολογισμός ορίων και συγκεκριμένα των τιμών MAX & AVG για κάθε πιθανή συμμαχία πρακτόρων. Με την ίδια λογική όπως και στον DP δίνονται αρχικές τιμές v[C] στις συμμαχίες και βάσει αυτών υπολογίζονται τα παραπάνω όρια. Στον πίνακα που ακολουθεί φαίνονται οι αρχικές τιμές καθώς και τα max, avg για κάθε πιθανή συμμαχία.

P1	v[C]	P2	v[C]	P3	v[C]	P4	v[C]
{a1}	125	{a1, a2}	175	{a1, a2, a3}	200	{a1, a2, a3, a4}	425
{a2}	110	{a1, a3}	150	{a1, a2, a4}	150		
{a3}	75	{a1, a4}	100	{a1, a3, a4}	300		
{a4}	150	{a2, a3}	150	{a2, a3, a4}	150		
		{a2, a4}	200				
		{a3, a4}	125				
	Max = 150		Max = 200		Max = 300		Max = 425
	Avg = 115		Avg = 150		Avg = 200		Avg = 425

Όπου max είναι απλώς η μέγιστη τιμή ενώ avg είναι η μέση τιμή. Για παράδειγμα στις συμμαχίες ενός πράκτορα έχουμε $avg = \frac{125+110+75+150}{4} = \frac{460}{4} = 115$.

Το επόμενο βήμα είναι βάσει των παραπάνω υπολογισμών να βρούμε τα όρια για κάθε μια απ' τις διαχωρίσεις. Επομένως έχουμε

Upper Bound = max (Max), Lower Bound = max (Avg)

$P_{[4]}$: Max = 425, Avg = 425

$P_{[1, 3]}$: Max = $Max_{[1]}$ + $Max_{[3]}$ = 450, Avg = $Avg_{[1]}$ + $Avg_{[3]}$ = 315

$P_{[2, 2]}$: Max = 400, Avg = 300 Upper Bound = 600, Lower Bound = 460

$P_{[1, 1, 2]}$: Max = 500, Avg = 380

$P_{[1, 1, 1, 1]}$: Max = 600, Avg = 460

Όπου στις $P_{[1, 1, 2]}$, $P_{[1, 1, 1, 1]}$ συμπεριλαμβάνονται οι συμμαχίες όπως εμφανίζονται στην Εικόνα 3.4.

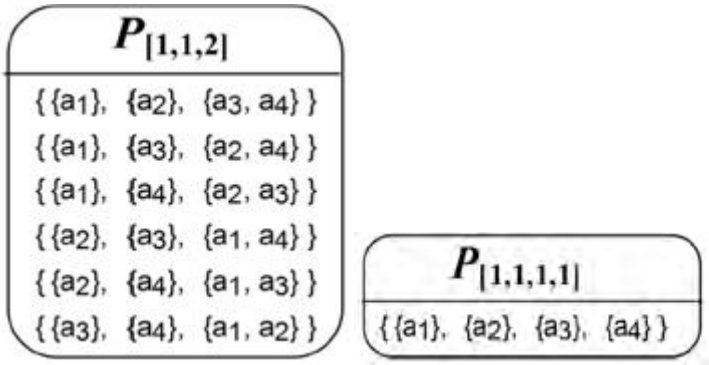
Αφού έχουμε υπολογίσει και τα όρια αρχίζουμε τους ελέγχους για να βρούμε πιθανές συμμαχίες που δεν μας χρειάζονται. Αρκεί να ελέγξουμε το Max μιας συμμαχίας με το Avg των υπολοίπων. Αν το Max κάποιας είναι μικρότερο από το Avg μιας άλλης αυτομάτως μπορούμε να την διαγράψουμε μιας και η δεύτερη μας συμφέρει περισσότερο. Στο παραπάνω παράδειγμα και με τα όρια που έχουμε υπολογίσει βλέπουμε ότι η $P_{[1, 3]}$ μας συμφέρει περισσότερο και απ' την $P_{[4]}$ και απ' την $P_{[2, 2]}$.

Μετά την διαγραφή μας απομένουν οι παρακάτω:

$P_{[1, 1, 2]}$: Max = 500, Avg = 380 Upper Bound = 600, Lower Bound = 460

$P_{[1, 1, 1, 1]}$: Max = 600, Avg = 460

Απ' τις οποίες καμία δεν μπορεί να παραληφθεί, επομένως σε αυτό το σημείο πρέπει να υπολογίσουμε τις τιμές για ότι έμεινε.



Εικόνα 3.5: Υπολειπόμενες συμμαχίες.

{{a1}, {a2}, {a3, a4}} = 125 + 110 + 125 = 360

{{a1}, {a3}, {a2, a4}} = 125 + 75 + 200 = 400

{{a1}, {a4}, {a2, a3}} = 125 + 150 + 150 = 425

{{a2}, {a3}, {a1, a4}} = 110 + 75 + 100 = 285

{{a2}, {a4}, {a1, a3}} = 110 + 150 + 150 = 410

{{a3}, {a4}, {a1, a2}} = 75 + 150 + 175 = 400

{{a1}, {a2}, {a3}, {a4}} = 125 + 110 + 75 + 150 = 460

Άρα η καλύτερη συμμαχία είναι η {{a1}, {a2}, {a3}, {a4}}, δηλαδή κάθε πράκτορας να δουλεύει μόνος του. Αν βρίσκαμε τιμή μεγαλύτερη του Lower Bound θα έπρεπε να αυξήσουμε το όριο του Lower Bound, αντίστοιχα αν δεν βρίσκαμε καμία τιμή ίση με το Upper Bound θα έπρεπε να το μειώσουμε ανάλογα. Σε περίπτωση προσαρμογής του Lower Bound θα έπρεπε να ελέγξουμε αυτόματα και τις υπόλοιπες διαμερίσεις για τυχόν νέες διαγραφές. Στην περίπτωση μας αν βρίσκαμε μια τιμή μεγαλύτερη του 500 θα μπορούσαμε να διαγράψουμε και την $P_{[1, 1, 2]}$ όπου το max της είναι 500 και σε αυτήν την περίπτωση θα γνωρίζαμε αυτόματα ότι η λύση μας βρίσκεται στην $P_{[1, 1, 1, 1]}$. Γενικά ο στόχος είναι να στενεύει το όριο των αριθμών (Upper Bound, Lower Bound).

Στην χειρότερη περίπτωση ο IP έχει πολυπλοκότητα $O(n^n)$, όπου n ο αριθμός των πρακτόρων που συμμετέχουν, παρόλα αυτά έχει αποδειχθεί [24] ότι είναι αρκετά γρηγορότερος και ελαφρύτερος, όσων αφορά υλικούς πόρους, απ᾽ τους αντίστοιχους DP-based αλγόριθμους

Ακολουθεί ενδεικτικά μια σειρά συναρτήσεων σε ψευδοκώδικα για τον IP:

Algorithm 1 : `scanAndSearch()` – scan the input, generate initial solutions and bounds.

Require: n, $\{\mathbf{v}(L_s)\}_{s\in\{1,2,\ldots,n\}}$

$CS' \leftarrow \arg\max_{CS\in\{\{a_1,\ldots,a_n\},\{\{a_1\},\ldots,\{a_n\}\}\}} V(CS)$
for $s = 1$ to $\lfloor \frac{n}{2} \rfloor$ **do**
  $\widehat{s} \leftarrow n - s$
  if $s = \widehat{s}$ {if cycling through the same list.} **then**
    $end = \lfloor |\mathbf{v}(L_s)|/2 \rfloor$
  else
    $end = |\mathbf{v}(L_s)|$
  end if
  Set max_s, $max_{\widehat{s}}$, v_{max} to $-\infty$, and set sum_s, $sum_{\widehat{s}}$ to 0
  for $x = 1$ to end {cycle through the lists $\mathbf{v}(L_s)$ and $\mathbf{v}(L_{\widehat{s}})$.} **do**
    $\widehat{x} \leftarrow |\mathbf{v}(L_s)| - x + 1$
    $v \leftarrow \mathbf{v}(L_s)^x$, $\widehat{v} \leftarrow \mathbf{v}(L_{\widehat{s}})^{\widehat{x}}$ {extract element at $x,\widehat{x}$ from $\mathbf{v}(L_s),\mathbf{v}(L_{\widehat{s}})$.}
    if $v_{max} < v + \widehat{v}$ **then**
      $v_{max} \leftarrow v + \widehat{v}$
      $x_{max} = x$ {record the index in $\mathbf{v}(L_s)$ at which v is located.}
    end if
    if $max_s < v$ **then**
      $max_s \leftarrow v$ {record the maximum value in $\mathbf{v}(L_s)$.}
    end if
    if $max_{\widehat{s}} < \widehat{v}$ **then**
      $max_{\widehat{s}} \leftarrow \widehat{v}$ {record the maximum value in $\mathbf{v}(L_{\widehat{s}})$.}
    end if
    $sum_s \leftarrow sum_s + v$, $sum_{\widehat{s}} \leftarrow sum_{\widehat{s}} + \widehat{v}$
  end for
  $\widehat{x}_{max} \leftarrow |\mathbf{v}(L_s)| - x_{max} + 1$
  if $V(CS') < V(\{L_s^{x_{max}}, L_{\widehat{s}}^{\widehat{x}_{max}}\})$ **then**
    $CS' \leftarrow \{L_s^{x_{max}}, L_{\widehat{s}}^{\widehat{x}_{max}}\}$ {update the best coalition structure found so far.}
  end if
  $avg_s \leftarrow sum_s/|\mathbf{v}(L_s)|$, $avg_{\widehat{s}} \leftarrow sum_{\widehat{s}}/|\mathbf{v}(L_{\widehat{s}})|$ {compute averages.}
end for
$\mathcal{G}' \leftarrow \mathcal{G} \setminus \mathcal{G}^2$
for $G \in \mathcal{G}'$ {compute upper and lower bounds for each sub-space in $\mathcal{G}'$.} **do**
  $MAX_G \leftarrow \sum_{s\in G} max_s \cdot G(s)$
  $AVG_G \leftarrow \sum_{s\in G} avg_s \cdot G(s)$
end for
$UB^* \leftarrow \max[\, V(CS'),\ \max_{G\in\mathcal{G}'}[MAX_G]\,]$
$LB^* \leftarrow \max[\, V(CS'),\ \max_{G\in\mathcal{G}'}[AVG_G]\,]$
$\mathcal{G}' \leftarrow$ `prune`$(\mathcal{G}', \{MAX_G\}_{G\in\mathcal{G}'}, LB^*)$ {prune the sub-spaces that have an upper bound lower than LB^*.}
$\beta \leftarrow \min[\, n/2\ ,\ UB^*/V(CS')\,]$ {compute a worst-case bound on $V(CS')$.}
return $CS', \beta, \{max_s\}_{s\in\{1,\ldots,n\}}, \mathcal{G}', \{MAX_G\}_{G\in\mathcal{G}'}, \{AVG_G\}_{G\in\mathcal{G}'}$

Εικόνα 3.6: Αλγόριθμος Scan and Search για τον IP. Ελέγχει την είσοδο και παράγει τις αρχικές λύσεις και όρια [23]

Algorithm 2 : `prune` $(\mathcal{G}', \{MAX_G\}_{G\in\mathcal{G}'}, v)$ – prune sub-spaces.

1: **for** $G \in \mathcal{G}'$ **do**
2: **if** $MAX_G \leq v$ {if the upper bound of P_G is lower than v.} **then**
3: $\mathcal{G}' \leftarrow \mathcal{G}' \setminus G$ {remove G.}
4: **end if**
5: **end for**
6: **return** $\mathcal{G}'$

Εικόνα 3.7: Αλγόριθμος Prune για τον IP. Μετά από κατάλληλους ελέγχους διαγράφει τους υποχώρους για τους οποίους το Max τους είναι χαμηλότερο από κάποιου άλλου το Avg. [23]

Algorithm 3 : `searchSpace()` – search, or prune, the remaining sub-spaces.

Require: $\mathcal{G}', \{MAX_G\}_{G\in\mathcal{G}'}, A, \beta^*$
1: **while** $\mathcal{G}' \neq \emptyset$ **do**
2: **Select** G'' {select the integer partition that represents the next sub-space to be searched.}
3: $CS' \leftarrow \texttt{searchList}(G'', 1, 1, A, CS', \overrightarrow{CS})$ {search within $P_{G''}$ and update CS'.}
4: $\mathcal{G}' \leftarrow \mathcal{G}' \setminus G''$ {remove $P_{G''}$ from the list of sub-spaces that are yet to be searched.}
5: **if** $CS' \in P_{G''}$ {If CS' has been modified while searching $P_{G''}$.} **then**
6: $\mathcal{G}' \leftarrow \texttt{prune}(\mathcal{G}', \{MAX_G\}_{G\in\mathcal{G}'}, V(CS'))$ {prune the sub-spaces that have upper bounds lower than $V(CS')$.}
7: **end if**
8: $UB^* \leftarrow \max[\, V(CS'),\ \max_{G\in\mathcal{G}'}[MAX_G]\,]$ {update the upper bound on the value of the optimal coalition structure(s).}
9: $\beta \leftarrow \min[\frac{UB^*}{V(CS')}, \beta]$ {update the worst-case bound on $V(CS')$.}
10: **if** $\beta \leq \beta^*$ {if CS' is within the specified bound from the optimal.} **then**
11: **return** CS'
12: **end if**
13: **end while**
14: **return** CS'

Εικόνα 3.8: Αλγόριθμος searchSpace για τον IP. Ελέγχει τους εναπομείναντες υποχώρους και διαγράφει αν γίνεται κάποιον απ' αυτούς. [23]

Algorithm 4 : `searchList`$(G, k, \alpha, A_k, CS', \overrightarrow{CS})$ – search a sub-space.

Require: $\{max_s\}_{s\in\{1,\cdots,n\}}, \{\mathbf{v}(L_s)\}_{s\in\{1,\cdots,n\}}, UB^*, \beta^*$

1: **if** $k > 1$ and $g_k \neq g_{k-1}$ {if the size is not being repeated.} **then**
2: $\alpha \leftarrow 1$ {reset α.}
3: **end if**
4: **for** $M_k \in LC_{g_k}^{|A_k|}$ such that $\alpha \leq M_{k,1} \leq n + 1 - \sum_{i=1}^{k}(g_k \times G(g_k))$ **do**
5: $C_k \leftarrow \{A_{k,i} \mid i \in M_k\}$ {extract C_k given M_k and A_k.}
6: **if** $k = |G|$ and $V(CS') < V(\overrightarrow{CS})$ **then**
7: $CS' \leftarrow \overrightarrow{CS}$ {update the current best.}
8: **else if** $V(CS') < \sum_{s\in\{g_1,\cdots,g_k\}} v(C_s) + \sum_{s\in\{g_{k+1},\cdots,g_n\}} max_s$ {branch only if there is potential of finding a coalition structure better than CS'.} **then**
9: $CS' \leftarrow$ `searchList`$(G, k + 1, M_{k,1}, A \setminus C, CS', \overrightarrow{CS})$ {branch to next coalition.}
10: **end if**
11: **if** $\frac{UB^*}{V(CS')} \leq \beta^*$ or $V(CS') = MAX_G$ {stop if the required solution has been found or if the current best is equal to the upper bound of this sub-space.} **then**
12: **return** CS'
13: **end if**
14: **end for**
15: **return** CS'

Εικόνα 3.9: Αλγόριθμος searchList για τον IP. Κάνει μια αναζήτηση μέσα στον υποχώρο. [23]

3.3 SlyCE Algorithm

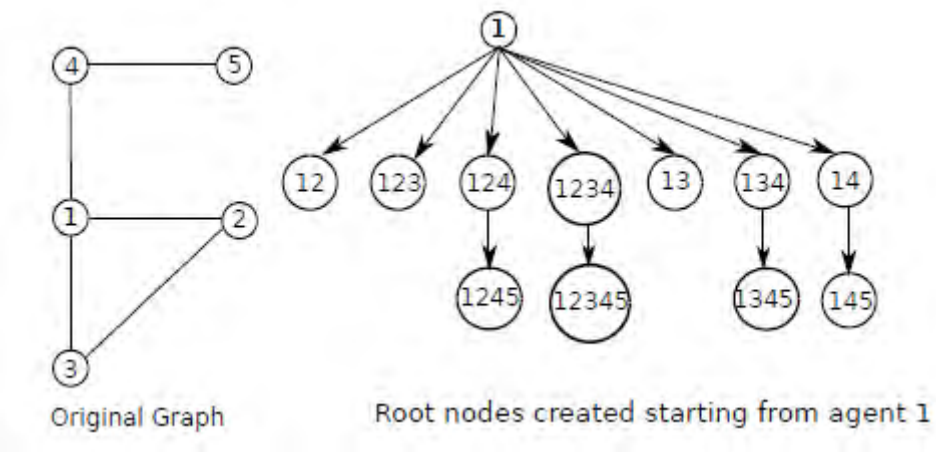
Σε αντίθεση με τους προηγούμενους δύο αλγορίθμους που δουλεύανε απλά γνωρίζοντας το πλήθος των πρακτόρων που θα συμμετέχουν στην συμμαχία ο αλγόριθμος SlyCE όπως και ο D-SlyCE (Distributed Sequentially connected Coalition Enumeration) χρειάζεται και ένα γράφημα σχέσεων με τους υπόλοιπους πράκτορες στην συμμαχία, όπου οι ακμές μεταξύ τους συμβολίζουν συνδέσμους όπως για παράδειγμα επικοινωνία, εμπιστοσύνη ή ακόμα και φυσικοί περιορισμοί. Στο [25] αναφέρεται ότι σε περίπτωση που δεν δημιουργηθεί ένα συνδεδεμένο υπό-γράφημα η δημιουργία συμμαχίας δεν είναι εφικτή. Σε κάθε περίπτωση θεωρείται ότι όλες οι συμμαχίες είναι εφικτό να πραγματοποιηθούν, πράγμα που σημαίνει ότι σε αντίθετη περίπτωση εκτελεί περιττούς υπολογισμούς.

Ο αλγόριθμος SlyCE λειτουργεί ως εξής. Το πρώτο του βήμα είναι για κάθε πράκτορα που συμμετέχει στην διαδικασία δημιουργίας συμμαχίας, εκτελεί depth-first search στο δένδρο αναπαράστασης του συνόλου των εφαρμόσιμων συμμαχιών. Κάθε στιγμή ο αλγόριθμος κρατάει δύο τιμές. Η πρώτη απ' αυτές είναι η ρίζα (R) του αλγορίθμου δηλαδή τον πράκτορα βάσει του οποίου εκτελείται ο depth-first search όπως φαίνεται π.χ. στην Εικόνα 3.10 για το αρχικό γράφημα με R = 1. Η δεύτερη τιμή είναι το σύνορο (F), το οποίο περιέχει πράκτορες που δεν είναι η ρίζα της συμμαχίας αλλά μπορεί να προστεθούν στην ρίζα με σκοπό την δημιουργία μιας έγκυρης συμμαχίας. Επίσης έχουμε την συνάρτηση N(· , ·) η οποία ορίζεται ως:

N(F , R) = {j|j > min(R ∪ F)} ∩ N(F) \ (N(R) ∪ R ∪ F)

Το N(.) ορίζει το σύνολο των γειτονικών κόμβων ενός υποσυνόλου του αρχικού μας συνόλου G, το οποίο ισούται με N(R) = {j : ∃i ∈ R, (i, j) ∈ E}. [25]

Στις σειρές 2 και 3 του αλγορίθμου στην Εικόνα 3.11 βλέπουμε τα βήματα παραγωγής νέας ρίζας χρησιμοποιώντας τιμές από ένα υποσύνολο του F το F* και την ένωση του F* με το R δηλαδή την τωρινή μας ρίζα, στην σειρά 4 την αξιολόγηση της επιλογής μας και τέλος στην σειρά 5 τον υπολογισμό νέου συνόρου χρησιμοποιώντας γειτονικούς πράκτορες με τους οποίους θα επεκτείνουμε την ρίζα. Τέλος αναδρομική κλήση του νέου αλγορίθμου με τα νέα δεδομένα ώστε να επαναληφθεί η διαδικασία με νέα ρίζα R' και νέο σύνορο F'. Στην Εικόνα 3.10 βλέπουμε την διαδικασία επέκτασης ρίζας για τον SlyCE με R = 1. Προφανώς η εικόνα θα ήταν τελείως διαφορετική σε περίπτωση που διαλέγαμε οποιαδήποτε άλλη ρίζα R ως την αρχική μας αφού αν διαλέγαμε ως ρίζα την R = 4 θα μας επέστρεφε τα {4}, {4, 5} ενώ αν διαλέγαμε για ρίζα την R = 5 το αποτέλεσμα θα ήταν {5}. Το παραπάνω συμβαίνει διότι ο SlyCE διαλέγει τους πράκτορες για την φάση της δημιουργίας καθώς και επέκτασης της ρίζας έτσι ώστε να μην υπολογίζει καμία συμμαχία δεύτερη φορά, επίσης φαίνεται ότι δεν υπάρχει καθόλου καλός καταμερισμός εργασίας στην περίπτωση του SlyCE πράγμα που διορθώνει αισθητά ο D-SlyCE. Ακολουθούν σε εικόνες ένα παράδειγμα εκτέλεσης επέκτασης ρίζα σε μια συμμαχία με γράφημα για R = 1 καθώς και ενδεικτικός αλγόριθμος σε ψευδοκώδικα του SlyCE απ' τους δημιουργούς του.



Εικόνα 3.10: Γράφημα συνέργειας για τον αλγόριθμος SlyCE. [25]

Algorithm 1 $slyce(R, F, m)$

1: **if** $F \neq \emptyset$ and $m > 0$ **then**
2: **for all** $F^* \subseteq F$ with $1 \leq |F^*| \leq m$ **do**
3: $R' \leftarrow F^* \cup R$ {Generate new root node.}
4: compute and store $v(R')$ {Evaluate new root node.}
5: $F' \leftarrow \overline{N}(F^*, R)$ {Generate new frontier set}
6: $slyce(R', F', m - |F^*|)$ {Recursive call.}
7: **end for**
8: **end if**

Εικόνα 3.11: Αλγόριθμος SlyCE. Με σκοπό την απαρίθμηση κάθε συμμαχίας

μεγέθους έως και m, ο SlyCE πρέπει να ψάξει κάθε οντότητα πράκτορα ξεχωριστά πράγμα το οποίο επιτυγχάνει καλώντας, slyce(∅, {i},m), for all i ∈ I. [25]

3.4 D-SlyCE Algorithm

Ο D-SlyCE είναι ο πρώτος αλγόριθμος που υπολόγισε έγκυρες τιμές για συμμαχία αποτελούμενη από πενήντα πράκτορες. Ο πρώτος αλγόριθμος με γράφημα για δημιουργία δομής συμμαχίας ήταν ο SlyCE ο οποίος όπως είδαμε στο παραπάνω παράδειγμα δεν κατανέμει την δουλειά μεταξύ πρακτόρων και έτσι πράκτορες με μεγάλα IDs έχουν λιγότερους υπολογισμούς απ' ότι οι πράκτορες με μικρότερο ID. Έτσι ο πράκτορας με το μεγαλύτερο ID θα υπολογίσει την τιμή μόνο μίας συμμαχίας, της συμμαχίας που περιλαμβάνει σαν μέλος μόνο τον ίδιο (βλέπε R = 5 στο παραπάνω παράδειγμα: R = 5 → {5}), ενώ αυτήν με το μικρότερο ID θα υπολογίσει κάθε δυνατή συμμαχία (βλέπε Εικόνα 3.10, R = 1). Για αυτόν τον λόγο ο SlyCE βελτιστοποιήθηκε και δημιουργήθηκε ο D-SlyCE.

Ο αλγόριθμος D-SlyCE αποτελεί μια βελτιστοποίηση του SlyCE καθώς κατανέμει μεταξύ όλων των πρακτόρων τον φόρτο εργασίας που αναλαμβάνει κάθε πράκτορας ώστε να μην υπολογίζονται έμμεσες δαπάνες στο κόστος επικοινωνίας. Επιπροσθέτως δεν υπολογίζει διπλότυπες συμμαχίες και μοιράζει τον φόρτο εργασίας αρκετά δίκαια μεταξύ των πρακτόρων.

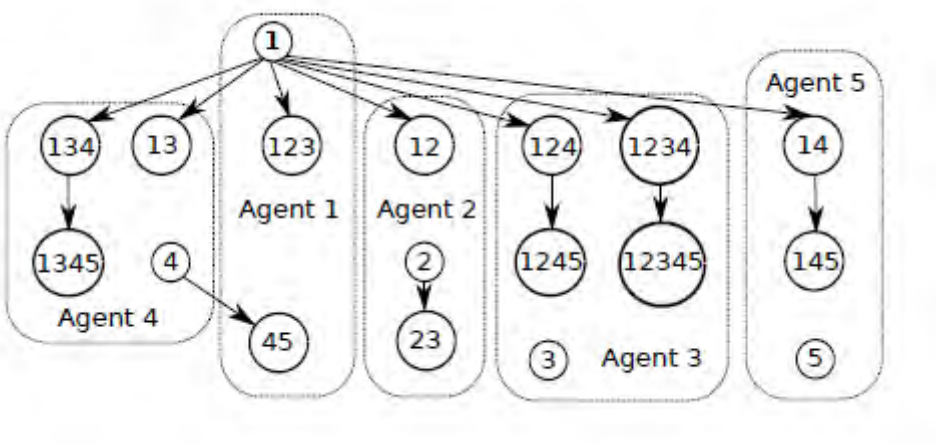
Algorithm 2 $dslyce(i, m)$

1: compute and store $v(\{i\})$
2: $x \leftarrow i - 1$
3: **for all** agents $a \in I$ **do**
4: $F \leftarrow \overline{N}(\{a\}, \emptyset)$ {Assign frontier set.}
5: **for all** $r = 1 \ldots, \min(|F|, m)$ **do**
6: $j \leftarrow \lfloor \binom{|F|}{r} \times x/n \rfloor$ {Index of beginning of share.}
7: **while** $j < \lfloor \binom{|F|}{r} \times (x+1)/n \rfloor$ **do**
8: $F' \leftarrow L(F, j+1, r)$ {Set next addition to root.}
9: $j \leftarrow j + 1$
10: compute and store $v(F' \cup \{a\})$ {As in SlyCE.}
11: $slyce(F' \cup \{a\}, \overline{N}(F', \{a\}), m - (r+1))$
12: **end while**
13: $x \leftarrow (x+1) \bmod n$ {Rotate share allocation.}
14: **end for**
15: **end for**

Εικόνα 3.12: Αλγόριθμος D-SlyCE. [25]

|F|: Το μέγεθος του συνόλου F. π.χ. για F = {A1, A2, A3} → |F| = 3 και $\binom{|F|}{r} = \frac{|F|!}{r! * (|F| - r)!}$

Ο D-SlyCE καλείται με είσοδο το ID του πράκτορα καθώς και το μέγιστο επιτρεπόμενο μέγεθος συμμαχίας που θα πρέπει να μπορεί να δημιουργήσει, m τέτοιο ώστε όλες οι συμμαχίες να πρέπει να δημιουργηθούν, m = n. Ο κάθε πράκτορας a, πρέπει αρχικά να υπολογίσει την v{a}. Στην συνέχεια κάθε πράκτορας a ∈ I αναλαμβάνει να υπολογίσει ένα περίπου ίσο κομμάτι των υποσυνόλων N({i}, ∅) για κάθε i ∈ I, τα οποία πρέπει αν αξιολογήσει ο a. Επειδή μεγαλύτερα σύνολα συνόρων πιθανώς συνεπάγονται και μεγαλύτερο κόστος υπολογισμού, και με την λογική του ότι το φορτίο πρέπει να μοιραστεί όσο το δυνατόν πιο δίκαια μεταξύ των πρακτόρων, ο a θα αναλάβει το 1/n του αρχικού υποσυνόλου N({i}, ∅), μεγέθους r για r = 1, … , | N({i}, ∅) |. Κάθε φορά που γίνεται αυτός ο υπολογισμός, ο αλγόριθμος αιτιοκρατικά αλλάζει την σειρά με την οποία γίνεται η μοιρασιά των υπολογισμών μεταξύ των πρακτόρων. Αυτό γίνεται επειδή λεξικογραφικά νωρίτερα σύνολα συνόρων είναι πιθανώς να συνεπάγονται μεγαλύτερο βάρος υπολογισμών. Όπως αναφέρεται στο [25] τόσο ο SlyCE όσο και ο D-SlyCE καλύπτουν ακριβώς τις κλήσεις των v(⋅), slyce(⋅ , ⋅ , ⋅) και επομένως αυτό σημαίνει ότι ο D-SlyCE καλύπτει ακριβώς τις ίδιες κλήσεις v(⋅) με τον SlyCE Απ' τα παραπάνω προκύπτει ότι, ο D-SlyCE είναι σωστός, ολοκληρωμένος και μη-περιττός.



Εικόνα 3.13: Δίκαιος καταμερισμός φόρτου εργασίας σε ένα σύνολο πέντε πρακτόρων, με χρήση του αλγορίθμου D-SlyCE. Κάθε πράκτορας έχει να υπολογίσει περίπου τον ίδιο αριθμό τιμών με τους υπόλοιπους. [25]

3.4 DyCE Algorithm

Ο DyCE λειτουργικά μοιάζει με τον DP αν και είναι γρηγορότερος στην εύρεση της βέλτιστης συμμαχίας, διότι χρησιμοποιεί τον SlyCE, ώστε να αγνοεί τις συμμαχίες που είναι ανέφικτες. Ουσιαστικά δηλαδή ο DyCE χρησιμοποιεί την μέθοδο nextslyce του SlyCE ώστε να παίρνει την νέα συμμαχία απ' το αρχικό γράφημα. Για κάθε γράφημα G = (I, E) υπάρχει ένα σύνολο κόμβων F(G) καθώς και ένα σύνολο ακμών E(G) όπως και στον D-SlyCE. Τέλος ορίζουμε και ένα σύνολο E*(G) το οποίο είναι ένα μειωμένο σύνολο ακμών υποσύνολο του αρχικού μας E(G). Όπως και στον DP έτσι και στον DyCE μπορούμε να βρούμε μια παρόμοια δομή συμμαχίας, η διαφορά βρίσκεται στο ότι η δομή του DP μπορεί να περνάει και από μη εφικτές συμμαχίες που δεν περιέχονται στο F(G), πράγμα το οποίο αλλάζει στην δομή του DyCE. Αυτό αποδεικνύεται με το παρακάτω θεώρημα

«Θεώρημα

Για κάθε εφικτή δομή συμμαχίας CS ∈ F(G) υπάρχει ένα μονοπάτι απ' το E*(G) το οποίο οδηγεί κατευθείαν απ' το {I} στο CS.» [25]

Ακολουθεί ο αλγόριθμος του DyCE σε ψευδοκώδικα.

Algorithm 3 *dyce*()

```
1: for all C ⊂ I do
2:    W(C) ← −∞ {Initialise memory.}
3: end for
4: for all a ∈ I do
5:    C ← {a}
6:    while C ≠ ∅ do
7:       W(C) ← v(C) {Store coalition value.}
8:       B(C) ← ∅ {Initialise best subset.}
9:       C ← nextslyce(C, I, n) {Get the next coalition gen-
         erated by SlyCE.}
10:   end while
11: end for
12: for all s = 1 . . . n do
13:   m ← ⌊s/2⌋
14:   if s < n then
15:      m ← min(m, n − s)
16:   end if{Maximum subset size}
17:   for i = 1, . . . (n s) do
18:      C ← L(I, i, s) {Go through sets of size s.}
19:      if W(C) > −∞ then {Ignore C if infeasible}
20:         for all a ∈ C do
21:            C′ ← {a}
22:            while C′ ≠ ∅ do
23:               if W(C′) + W(C \ C′) > W(C) then
24:                  W(C) ← W(C′) + W(C \ C′)
25:                  B(C) ← C′
26:               end if{Evaluate subset}
27:               C′ ← nextslyce(C′, C, m)
28:            end while{Calculate w(C)}
29:         end for
30:      end if
31:   end for
32: end for
33: return bestcs(I)
```

Algorithm 4 *bestcs*(C)

```
if B(C) = ∅ then
   return {C}
else
   return bestcs(B(C)) ∪ bestcs(C \ B(C))
end if
```

Εικόνα 3.14: Ψευδοκώδικας για τον αλγόριθμο DyCE καθώς και για την μέθοδο bestCS [25]

Ο DyCE ξεκινάει θέτοντας όλες τις τιμές συμμαχίας W(C) → -∞. Συμμαχίες των οποίων οι τιμές παραμένουν -∞ θεωρούνται μη εφικτές και αγνοούνται απ' τον αλγόριθμο (όπως φαίνεται στην σειρά 19 του ψευδοκώδικα). Σε συνέχεια των αρχικοποιήσεων θέτει όλες τις τιμές συμμαχίας καθώς επίσης αρχικοποιεί και το B(C)

το οποίο είναι μεταβλητή για το βέλτιστο υποσύνολο και παίρνει την επόμενη συμμαχία που του επιστρέφει ο SlyCE μέσω της nextslyce(▪ , ▪ , ▪). Στην συνέχεια για κάθε εφικτή συμμαχία μεγέθους s = 1, 2, …, n υπολογίζεται το w(C) και αντικαθίσταται το v(C). Για τον υπολογισμό του w(C) μιας συμμαχίας C != {I} ο DyCE χρησιμοποιεί τον SlyCE για κάθε υποσύνολο C' του C που είναι μικρότερο του n-|C| και |C|/2 και υπολογίζει το w(C') + w(C\C') και θέτει το w(C) = max{ w(C') + w(C\C'), v(C)} και επίσης ενημερώνει και το B(C) όπου χρειάζεται. Τέλος υπολογίζεται και το w(I) για όλα τα C υποσύνολα του I μεγιστοποιώντας το W(C) + W(I\C). Μέσω της μεθόδου bestCS υπολογίζεται και η βέλτιστη δομή συμμαχίας ξεκινώντας από CS = {I} και αναδρομικά θέτουμε C = C' και w(C) = w(C') + w(C\C') ή v(C) αντίστοιχα. Επιπλέον το συνολικό W(C) για κάθε C στο CS δεν αλλάζει για κάθε αντικατάσταση και έτσι για την τελική δομή συμμαχίας θα ισχύει w(C) = v(C) αφού ο αλγόριθμος σταματάει όταν δεν μπορεί να βρει παραπάνω υποδιαιρέσεις όπου ισχύει V(CS) = W(CS) = W(I) και επομένως η δομή μας είναι η βέλτιστη δυνατή. Μπορούμε να μειώσουμε λίγο τις απαιτήσεις μνήμης με το να σταματήσουμε να αποθηκεύουμε τα συγκεκριμένα υποσύνολα που χρησιμοποιούνται για τον υπολογισμό της βέλτιστης δομής και να τα βρίσκουμε αφού υπολογιστεί το αντίστοιχο w(C) στα υποσύνολα του.

ΚΕΦΑΛΑΙΟ 4: ΣΥΓΚΡΙΣΗ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

Για να συγκρίνουμε τους αλγορίθμους και να εξάγουμε τα αποτελέσματά μας θα πρέπει να εξετάσουμε δύο περιπτώσεις εισόδου. Μία όταν σαν είσοδο έχουμε ένα σύνολο πρακτόρων μόνο π.χ. I = {a1, a2, a3} και μία όταν λαμβάνουμε σαν είσοδο το σύνολο των πρακτόρων και ένα γράφημα συσχετίσεων G = {I, E} όπου οι κόμβοι του γραφήματος είναι οι πράκτορες ai ∈ I και οι ακμές E που συνδέουν τους πράκτορες είναι οι συσχετίσεις μεταξύ τους, το αν δηλαδή είναι εφικτή συμμαχία μεταξύ των δύο πρακτόρων ή όχι. Αυτό διότι δύο απ' τους τέσσερις αλγορίθμους που παρουσιάστηκαν στα πλαίσια της διπλωματικής εργασίας και συγκεκριμένα ο D-SlyCE και ο DyCE για να μπορέσουν να εκτελεστούν βασικό προαπαιτούμενο είναι το γράφημα συσχετίσεων των πρακτόρων G. Επειδή αυτό το γράφημα δεν δίνεται πάντα σαν είσοδος οι δύο αυτοί αλγόριθμοι δεν είναι δυνατόν να εκτελεστούν σε κάθε περίπτωση και επομένως σε αυτές τις περιπτώσεις θα πρέπει η εύρεση δομής συμμαχίας να γίνει υποχρεωτικά με τον DP ή τον IP. Γι' αυτόν τον λόγο θα πρέπει να γίνει επομένως έλεγχος και των δύο περιπτώσεων, τι μας συμφέρει να χρησιμοποιούμε δηλαδή ανάλογα με την διαθέσιμη είσοδο. Σημαντικό είναι να αναφερθεί ότι οι αλγόριθμοι DP και IP δεν μπορούν να εκτελεστούν για οποιοδήποτε αριθμό πρακτόρων. Από 30 πράκτορες και πάνω δεν επιστρέφουν αποτέλεσμα, επομένως σε τέτοια περίπτωση θα πρέπει να δοθεί και ένα γράφημα και να εκτελεστεί με τον SlyCE ή τον D-SlyCE οι οποίοι έχουν επιστρέψει λύση μέχρι και για προβλήματα 50 πρακτόρων.

Απ' τους αλγορίθμους που παρουσιάστηκαν στο προηγούμενο κεφάλαιο μόνο για τον DP έχω πειραματικά αποτελέσματα [26]. Ο κώδικας της πηγής ήταν γραμμένος σε γλώσσα C, έγινε compile με τον "gcc" και εκτελέστηκε σε δύο περιβάλλοντα Linux (Ubuntu 16.04 και ElementaryOS 5.1). Τα συστήματα στα οποία έτρεξε ήταν ένα VM (3.5GHz Dual Core, 2GB RAM) καθώς και ένα Laptop (Intel Core i3 M380 @ 2.53GHz, 4GB RAM).

Το όρισμα n4 δηλώνει πόσους πράκτορες θέλουμε να συμπεριλάβουμε στην δημιουργία της συμμαχίας μας και αποτελούν το CS μας. Το a1 υποδηλώνει την μέθοδο εκτέλεσης, στην συγκεκριμένη υλοποίηση επέτρεπε το τρέξιμο του DP με banker's sequence ή fast split που ήταν πολύ-νηματικός. Ο αλγόριθμος που περιγράφω στα πλαίσια της διπλωματικής μου εργασίας δεν είναι πολύ-νηματικός και έτσι αποφάσισα να μην συμπεριλάβω runs απ' την fast split εκδοχή του αλγορίθμου παρόλο που είναι αρκετά γρηγορότερος.

Ακολουθούν δύο screenshots με runs από διάφορες εκτελέσεις στα δύο περιβάλλοντα που περιγράφονται παραπάνω. (Το Execution Time μετριέται σε second).

```
ilias@Virtual:~/Downloads/DPIDP-master$ ./DPIDP -n4 -a1
Size 4
Running DP using banker's sequence
4 1709054488  [ 1 2 ] [ 3 4 ]
Execution time : 0.000007
ilias@Virtual:~/Downloads/DPIDP-master$ ./DPIDP -n6 -a1
Size 6
Running DP using banker's sequence
6 2062083787  [ 2 5 ] [ 1 3 4 6 ]
Execution time : 0.000011
ilias@Virtual:~/Downloads/DPIDP-master$ ./DPIDP -n8 -a1
Size 8
Running DP using banker's sequence
8 2038789283  [ 1 2 6 7 ] [ 3 4 5 8 ]
Execution time : 0.000043
ilias@Virtual:~/Downloads/DPIDP-master$ ./DPIDP -n10 -a1
Size 10
Running DP using banker's sequence
10 2124223002  [ 2 3 4 5 8 9 ] [ 1 6 7 10 ]
Execution time : 0.000313
ilias@Virtual:~/Downloads/DPIDP-master$ ./DPIDP -n15 -a1
Size 15
Running DP using banker's sequence
15 2134110068  [ 2 3 5 6 8 10 ] [ 1 4 7 9 11 12 13 14 15 ]
Execution time : 0.165540
ilias@Virtual:~/Downloads/DPIDP-master$ ./DPIDP -n20 -a1
Size 20
Running DP using banker's sequence
20 2145699819  [ 1 9 10 11 12 17 19 ] [ 2 3 4 5 6 7 8 13 14 15 16 18 20 ]
Execution time : 25.861925
ilias@Virtual:~/Downloads/DPIDP-master$
```

Εικόνα 4.1: Εκτελέσεις αλγορίθμου DP στο VM μου για μεταβλητό αριθμό πρακτόρων με τυχαίες τιμές v(C).

```
ilias@elementaryOS:~/Documents/DPIDP$ ./DPIDP -n4 -a1
Size 4
Running DP using banker's sequence
4 1701047019  [ 1 ] [ 2 ] [ 3 4 ]
Execution time : 0.000030
ilias@elementaryOS:~/Documents/DPIDP$ ./DPIDP -n6 -a1
Size 6
Running DP using banker's sequence
6 1950536204  [ 2 ] [ 4 ] [ 1 3 5 6 ]
Execution time : 0.000017
ilias@elementaryOS:~/Documents/DPIDP$ ./DPIDP -n6 -a1
Size 6
Running DP using banker's sequence
6 1945908390  [ 2 4 5 ] [ 1 3 6 ]
Execution time : 0.000050
ilias@elementaryOS:~/Documents/DPIDP$ ./DPIDP -n8 -a1
Size 8
Running DP using banker's sequence
8 1994260107  [ 2 3 5 6 ] [ 4 7 ] [ 1 8 ]
Execution time : 0.000144
ilias@elementaryOS:~/Documents/DPIDP$ ./DPIDP -n10 -a1
Size 10
Running DP using banker's sequence
10 2079885311  [ 2 7 ] [ 1 3 9 ] [ 4 5 6 8 10 ]
Execution time : 0.000593
ilias@elementaryOS:~/Documents/DPIDP$ ./DPIDP -n15 -a1
Size 15
Running DP using banker's sequence
15 2143619265  [ 6 12 13 ] [ 7 8 9 11 14 ] [ 1 2 3 4 5 10 15 ]
Execution time : 0.130987
ilias@elementaryOS:~/Documents/DPIDP$ ./DPIDP -n20 -a1
Size 20
Running DP using banker's sequence
20 2147292397  [ 2 6 10 13 15 16 ] [ 1 4 5 11 12 14 17 19 ] [ 3 7 8 9 18 20 ]
Execution time : 67.102251
```

Εικόνα 4.2: Εκτελέσεις αλγορίθμου DP στο Laptop μου για μεταβλητό αριθμό πρακτόρων με τυχαίες τιμές v(C).

Όπως φαίνεται στις δύο παραπάνω εικόνες για μικρό αριθμό πρακτόρων (4-10 πράκτορες)

- Ο χρόνος εκτέλεσης είναι πολύ μικρότερος του second συγκεκριμένα για το Laptop (με 10 πράκτορες) ισούται με 0.593 ms ενώ για το VM ισούται με 0.313ms
- Για 6 πράκτορες στο laptop παρατηρούμε ότι ανάλογα τα νούμερα μπορεί όταν η διαφορά είναι μικρή να έχουμε περισσότερους πράκτορες και μικρότερο χρόνο εκτέλεσης (Laptop n=4 → 0.000030s vs Laptop n=6 → 0.000017s)
- Παρατηρούμε επίσης ότι όσο μεγαλώνει το νούμερο τόσο μεγαλώνει και η διαφορά χρόνων εκτελέσεων που είναι άλλωστε και λογικό αφού ο DP έχει πολυπλοκότητα $O(3^n)$.

Πράκτορες	VM Execution Time (sec)	Laptop Execution Time (sec)
6	0.000011	0.000017
8	0.000043	0.000144
10	0.000313	0.000593
15	0.165540	0.130987
20	25.861925	67.102251

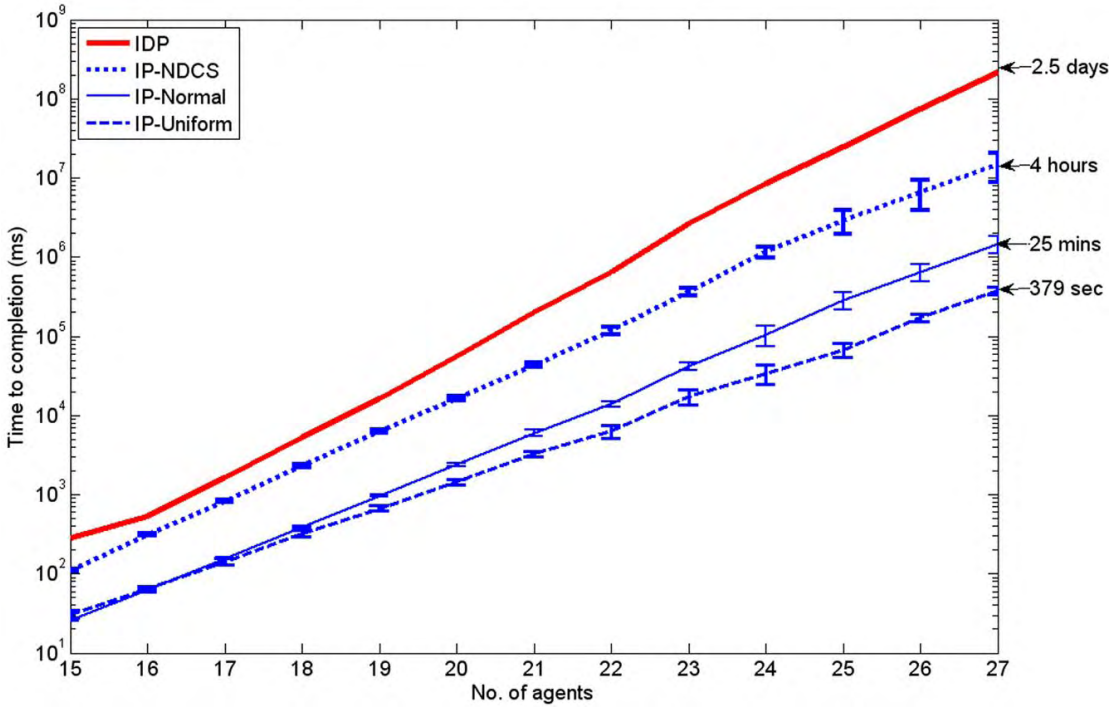
Όπως αναφέρθηκε ήδη και στο Κεφάλαιο 3, ο αλγόριθμος IP είναι γρηγορότερος του DP στις περισσότερες περιπτώσεις λόγω του ότι δεν εκτελεί περιττές πράξεις. Αναφέρεται απ' τους δημιουργούς του IP στο [23] ότι υπάρχουν δύο εκδοχές του αλγορίθμου IP:

- Normal: $v(C) \sim |C| * N(\mu, \sigma^2)$, για $\mu = 1$ & $\sigma = 0.1$.
- Uniform: $v(C) \sim |C| * U(a, b)$, για $a = 0$ & $b = 1$

Τέλος αναφέρεται ότι οι Normal & Uniform εκδοχές του IP μπορεί για κάποια μεγέθη συμμαχιών (για μεγάλο αριθμό) μπορεί να επιστρέψουν αποτελέσματα που αδικούν είτε τον ένα αλγόριθμο ή τον άλλον. Για αυτόν τον λόγο προτείνεται ένας άλλος αλγόριθμος ο NDCS (Normally Distributed Coalition Structure) για τον οποίο ισχύει:

- NDCS: $v(C) \sim N(\mu, \sigma^2)$, για $\mu = |C|$ και $\sigma = \sqrt[2]{|C|}$

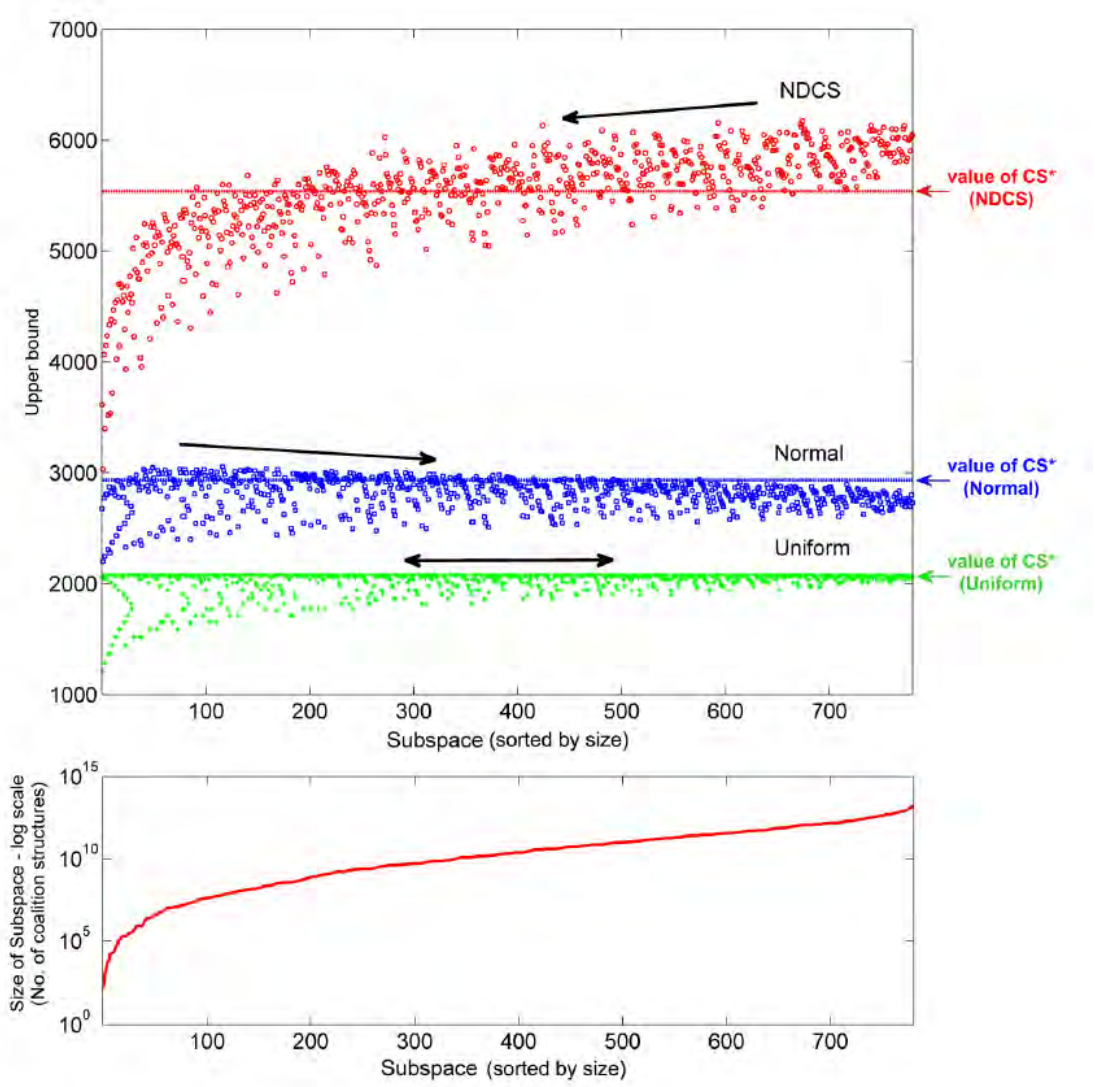
Ακολουθεί διάγραμμα σύγκρισης των αλγορίθμων (I)DP & IP όπου ο IDP (Improved Dynamic Programming) και είναι γρηγορότερος απ' τον απλό DP (Dynamic Programming).



Εικόνα 4.3: IDP vs. IP-NDCS vs. IP-Normal vs. IP-Uniform [23].

Απ' το παραπάνω γράφημα φαίνεται ότι ο IP είναι αρκετά πιο γρήγορος του IDP και κατ' επέκταση και του μη-βελτιστοποιημένου DP κατά πολύ. Για 27 πράκτορες που ο IDP θέλει 2.5 μέρες για να ολοκληρωθεί, ο απλός IP θέλει 25 λεπτά ενώ ο Uniform θέλει μόλις 379 δευτερόλεπτα, ενώ η εκδοχή του IP με τα πιο «δίκαια» αποτελέσματα ο NDCS θέλει περίπου 4 ώρες που σε κάθε περίπτωση είναι πολύ λιγότερο των 2.5 ημερών, το οποίο θεωρείται απαγορευτικό.

Ακολουθούν και δύο διαγράμματα που απεικονίζουν την κατάσταση του CS* σε κάθε περίπτωση εκτέλεσης του αλγορίθμου IP δηλαδή για τις NDCS, Normal & Uniform εκδοχές του αλγορίθμου. Στο πρώτο διάγραμμα στον κατακόρυφο άξονα φαίνεται το άνω όριο (Upper Bound) του υποχώρου ενώ στο δεύτερο φαίνεται μια λογαριθμική κλίμακα των δομών συμμαχίας που ελέγχθηκαν και στον οριζόντιο άξονα και στις δύο περιπτώσεις είναι ο υποχώρος ταξινομημένος κατά μέγεθος.

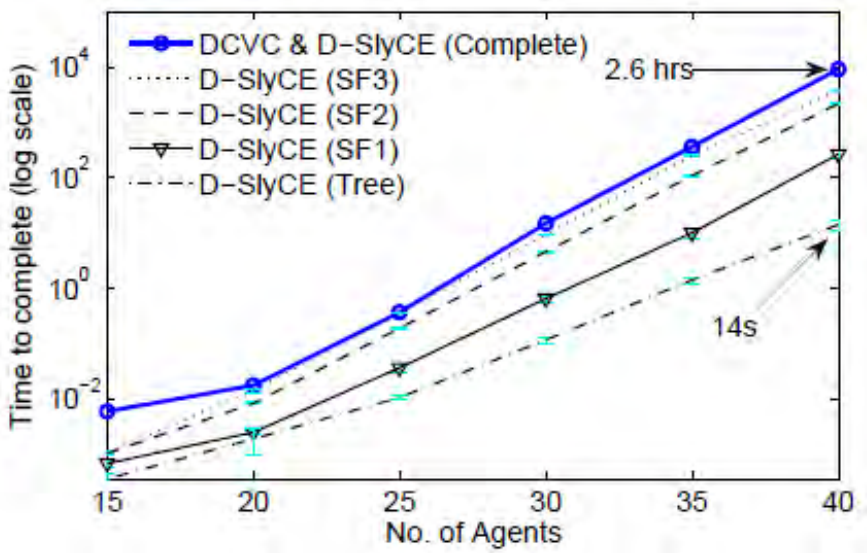


Εικόνα 4.4: Διαγράμματα υποχώρων [23].

Σε περίπτωση λοιπόν που σαν είσοδος για τον αλγόριθμο έχουμε απλά ένα σύνολο πρακτόρων π.χ. I = {a1, a2, a3} ο γρηγορότερος αλγόριθμος απ' αυτούς που παρουσιάστηκαν είναι ο IP αλγόριθμος.

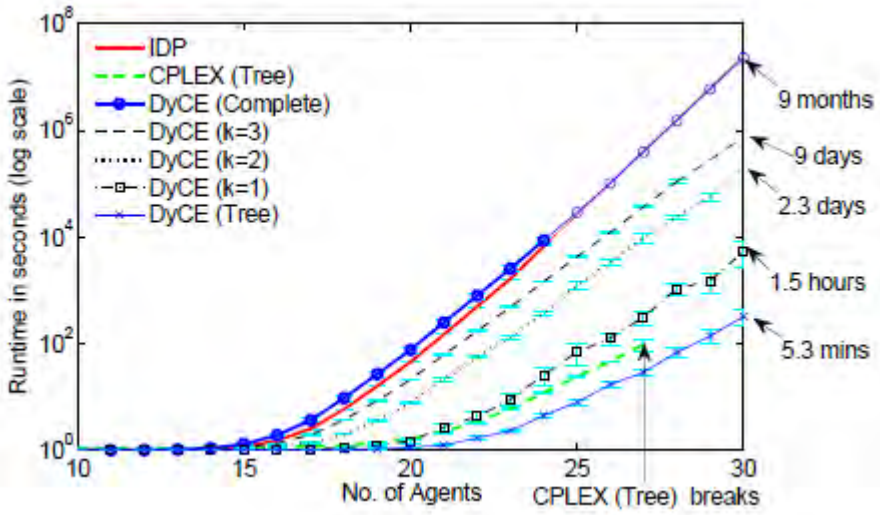
Στην περίπτωση όμως που δοθεί και γράφημα συσχετίσεων των πρακτόρων μπορούμε να τρέξουμε και τους άλλους δύο αλγορίθμους D-SlyCE και DyCE. Το βασικό μας πλεονέκτημα σε αυτήν την περίπτωση είναι ότι αυξάνεται όπως είδαμε και παραπάνω το όριο πρακτόρων για τους οποίους μπορούμε να εκτελέσουμε τον αλγόριθμο από 30 σε περίπου 50. Αυτό βοηθάει ιδιαίτερα σε ρεαλιστικά προβλήματα που ο αριθμός των διαθέσιμων πρακτόρων μπορεί να ξεπερνάει τους 30.

Ακολουθεί διάγραμμα σύγκρισης του D-SlyCE με τον DCVC και του DyCE με τον IDP και τον CPLEX ενός άλλου αλγορίθμου δημιουργίας δομής συμμαχίας.



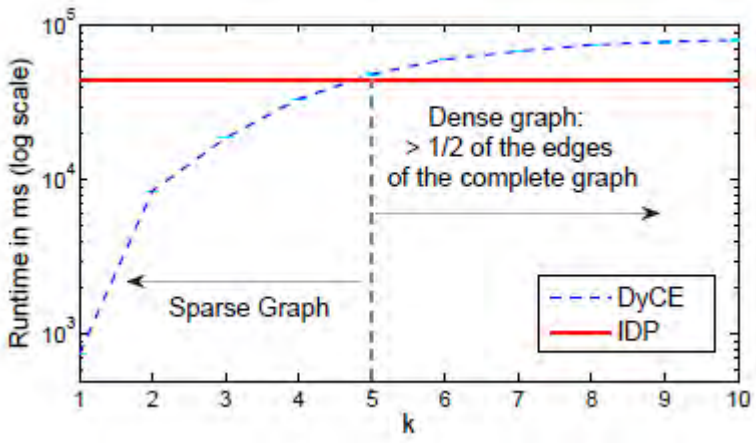
Εικόνα 4.5: D-SlyCE vs. DCVC [25].

Στο παραπάνω γράφημα φαίνεται ότι ο D-SlyCE είναι το ίδιο γρήγορος με τον DCVC όταν μιλάμε για ολόκληρο/πυκνό γράφημα. Σε περίπτωση όμως που έχουμε αραιό γράφημα είναι αρκετά γρηγορότερος για κάθε n (αριθμό πρακτόρων). Όσο πιο αραιό το γράφημα τόσο γρηγορότερος είναι ο αλγόριθμος. «Για δέντρα και scale-free γραφήματα είναι τουλάχιστον 2.5 φορές γρηγορότερος (3700s αντί για 9300s που θέλει με scale-free γράφημα, k = 3 και 40 πράκτορες), ενώ είναι καλύτερος κατά περίπου 660 φορές στην καλύτερη περίπτωση (14s αντί για 9300s σε δένδρο με 40 πράκτορες).» [25] Για 40 πράκτορες ο D-SlyCE σε δένδρο με τυχαίες τιμές χρειάζεται περίπου 14s για να εντοπίσει όλες τις εφικτές συμμαχίες ενώ για 50 πράκτορες επιστρέφει λύση σε περίπου 14 λεπτά. Τέλος για τυχαία δένδρα ακόμα και στην περίπτωση που έχουμε πυκνό δένδρο και από κάποιο σημείο και μετά ο DCVC γίνεται γρηγορότερος, ο D-SlyCE έχει και πολύ χαμηλό συνολικό κόστος εκτέλεσης επομένως σε κάθε περίπτωση συμφέρει έναντι του αρκετά πιο αργού και βαρύ υπολογιστικά DCVC.



Εικόνα 4.6: DyCE vs. IDP vs. CPLEX [25].

Στο γράφημα 4.6 γίνεται σύγκριση του IDP με τον CPLEX και τον DyCE με πλήρες γράφημα, σε scale-free γράφημα για k=3, 2, 1 καθώς και για δένδρο. Φαίνεται ότι μόνο στην περίπτωση του πλήρους γραφήματος ο IDP είναι γρηγορότερος από τον DyCE ενώ όταν μιλάμε για scale-free γράφημα ο DyCE αρχίζει να ξεπερνάει τον IDP δραστικά σε χρόνο εκτέλεσης. Για 30 πράκτορες ο χρόνος εκτέλεσης του IDP φαίνεται να είναι 9 μήνες ενώ του DyCE για scale-free γράφημα με k = 3 είναι 9 μέρες και στην καλύτερη περίπτωση που έχουμε DyCE με δένδρο ο χρόνος εκτέλεσης πέφτει στα 5.3 λεπτά! Στην βέλτιστη περίπτωση θα ξεπερνούσε κατά πολύ και τον IP-Normal ενώ πιθανώς να εκτελούνταν λίγο ταχύτερα ο IP-Uniform. Το βασικό πλεονέκτημα του DyCE είναι ότι συνεχίζει να κρατάει σχετικά χαμηλούς χρόνους και για ρεαλιστικά προβλήματα με παραπάνω από 30 πράκτορες, στην οποία περίπτωση ο IP δεν μπορεί καν να εκτελεστεί. Επομένως για ρεαλιστικά προβλήματα με n > 30 ο DyCE είναι η ταχύτερη επιλογή. Τέλος στο γράφημα που ακολουθεί φαίνεται ότι υπάρχει ένα κατώφλι μετά το οποίο ο IDP καθώς και ο IP (ο οποίος στις περισσότερες περιπτώσεις είναι γρηγορότερος απ' τον IDP) εκτελείται γρηγορότερα απ' τον DyCE. Αυτό εξαρτάται απ' το k δηλαδή το πόσο πυκνό είναι το γράφημα. Αυτήν η τιμή είναι όταν μιλάμε για γράφημα με πάνω απ' τις μισές ακμές απ' το πλήρες γράφημα.



Εικόνα 4.7: DyCE vs. IDP χρόνος εκτέλεσης σε λογαριθμική κλίμακα [25].

Συνολικά λοιπόν παρατηρούμε ότι σε περίπτωση που έχουμε ένα σύνολο σαν είσοδο η καλύτερη επιλογή μας είναι ο IP αλγόριθμος, ενώ ο DP και κατ' επέκταση ο IDP τον ξεπερνάει σε χρόνο εκτέλεσης μόνο όταν δεν μπορούμε να κάνουμε πολλές απαλοιφές. Σε περίπτωση που δοθεί και γράφημα συσχετίσεων των πρακτόρων και το γράφημα είναι αραιό (<= ½ των ακμών σε σχέση με το πλήρες γράφημα) ο DyCE είναι η καλύτερη επιλογή. Σε περίπτωση που δοθεί πυκνό γράφημα η καλύτερη επιλογή είναι ο IP και πάλι. Γενικά ο D-SlyCE ήταν 660 φορές ταχύτερος απ' τον DCVC ενώ ο DyCE ήταν $7*10^4$ φορές ταχύτερος απ' τον IDP και είναι επίσης οι πρώτοι αλγόριθμοι που επιστρέφουν λύση μέχρι και για γραφήματα 50 πρακτόρων και βοηθάνε στην επίλυση προβλημάτων όπου εμφανίζονται αραιές συσχετίσεις όπως π.χ. αποκεντροποιημένος συγχρονισμός αισθητήρων ή πομπούς έκτακτης ανάγκης, καθώς και σε άλλες υλοποιήσεις.

ΚΕΦΑΛΑΙΟ 5: ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ

Η δημιουργία δομής συμμαχίας είναι μια διαδικασία που μπορεί να βελτιωθεί περαιτέρω είτε με εύρεση νέων λύσεων η ακόμα και με συνδυασμό των λύσεων που ήδη έχουμε. Θα ήταν σημαντικό λοιπόν σε μελλοντικές έρευνες να δοθεί ιδιαίτερη βάση στην βελτίωση του χρόνου εκτέλεσης αυτής της διαδικασίας μια και είναι αρκετά σημαντική στην επίλυση πολλών ρεαλιστικών προβλημάτων.

Παρόλο που κάποιες απ' τις λύσεις που δόθηκαν στα πλαίσια αυτής της διπλωματικής μπορεί να εκτελούνται σε αποδεκτό έως και πολύ μικρό χρόνο υπάρχει πάντα δυνατότητα για βελτίωση αυτών των χρόνων. Έχουν ήδη γίνει προτάσεις οι οποίες φαίνεται να λειτουργούν καλύτερα απ' τον IP και τον IDP, για κάθε n, όπως για παράδειγμα ο IDP-IP που παρουσιάζεται στο [22] και στο [27]. Αντίστοιχα για τον DyCE, εύρεση τρόπου για τον συνδυασμό των D-SlyCE και DyCE με τεχνικές "branch-and-bound" για επιπλέον βελτίωση του χρόνου αναζήτησης και «κλαδέματος» του γραφήματος κρατώντας την εγγύηση επιστροφής λύσης, όπως επίσης και τεχνικές αυτόματης εύρεσης αραιών συσχετίσεων σε ένα γράφημα για ευκολότερες αποφάσεις σχετικά με το ποιος αλγόριθμος πρέπει να χρησιμοποιηθεί.

Μία άλλη πρόταση θα μπορούσε να είναι η αυτοματοποίηση της παραπάνω διαδικασίας δημιουργίας της δομής μιας συμμαχίας με χρήση νευρωνικών δικτύων, όπου οι πράκτορες μετά από κάθε εκτέλεση θα «μαθαίνουν» όλο και περισσότερο πως να χειρίζονται διάφορες καταστάσεις και να προσφέρουν ακόμα πιο άμεσα την βέλτιστη δυνατή λύση είτε αυτό είναι η κατάλληλη επιλογή αλγορίθμου για κάθε περίπτωση ή ταχύτερος χρόνος εκτέλεσης παρόμοιων προβλημάτων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Κ. Γεωργούλη, *Τεχνητή Νοημοσύνη Μια εισαγωγική προσέγγιση*, 2015

[2] S. Russell and P. Norvig, *Artificial intelligence A Modern Approach*, 1994

[3] M. Wooldridge, *An introduction to multiagent systems*. Chichester: John Wiley, 2009.

[4] M. Burgin and G. Dodig-Crnkovic, *A Systematic Approach to Artificial Agents*, 2009.

[5] M. Glavic, *Agents and Multi-Agent Systems: A Short Introduction for Power Engineers*, 2006.

[6] K. P. Sycara, *Multiagent Systems*, 1998

[7] W. Gruszczyk and H. Kwasnicka, *Coalition Formation in multi-agent systems-an evolutionary approach*, 2008

[8] O. Shehory and S. Kraus, *Methods for task allocation via agent coalition formation*, 1998

[9] P. Caillou, S. Aknine, S. Pinson, *How to Form and Restructure Multi-agent Coalitions*. National Conference on Artificial Intelligence (AAAI 02) Workshop on Coalition Formation, Edmonton, Canada, AAAI Press, 2002

[10] D. Dobrev, *The definition of AI in terms of multi agent systems*, 2012

[11] X. Li and L.-K. Soh, *Investigating Reinforcement Learning in Multiagent Coalition Formation*, 2004

[12] T. Sandholm, *Distributed Rational Decision Making*, 1999

[13] J. B. O'Day, *Joining Forces: A Guide for Forming, Joining and Building Political Coalitions*, 2004

[14] Business Advocacy Network, *Building Coalitions*

[15] National Democratic Institute, *Coalitions: A Guide for Political Parties*, 2015

[16] M. Husakova, *The Immunity-Based Multi-Agent Coalition Formation*, 2011

[17] L. Cohen, N. Baer and P. Satterwhite, *Developing Effective Coalitions: An Eight Step Guide*, 2003

[18] T. Rahwan, *Algorithms for Coalition Formation in Multi-Agent Systems*, 2007

[19] T. Rahwan and N. R. Jennings, *An algorithm for distributing coalitional value calculations among cooperating agents*, 2007

[20] T. Rahwan, and N. J. Jennings, *An Improved Dynamic Programming Algorithm for Coalition Structure Generation*, 2008

[21] F. Cruz et al, *Coalition structure generation problems: Optimization and parallelization of the IDP algorithm*, 2017

[22] T. Rahwan and N. R. Jennings, *Coalition Structure Generation: Dynamic Programming Meets Anytime Optimization*, 2008

[23] T. Rahwan, S. D. Ramchurn, N. R. Jennings, *An Anytime Algorithm for Optimal Coalition Structure Generation*, 2009

[24] T. Voice, M Polukarov and N. R. Jennings, *Coalition Structure Generation over Graphs*, 2011

[25] T. Voice, S. D. Ramchurn and N. R. Jennings, *On Coalition Formation with Sparse Synergies*, 2012

[26] F. Cruz et al, *Coalition structure generation problems: Optimization and parallelization of the IDP algorithm in multicore systems*, 2016,

https://doi.org/10.1002/cpe.3969

[27] T. Rahwan et al, *Anytime Optimal Coalition Structure Generation. Proceedings of the National Conference on Artificial Intelligence*, 2007.