ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Γεωργίου Π. Μαυρομμάτη

ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΕΚΘΕΤΙΚΟΥ ΧΡΟΝΟΥ ΜΕ ΣΥΓΧΡΟΝΙΣΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

ΠΕΙΡΑΙΑΣ 2002

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διατριβή
για την απόκτηση διδακτορικού διπλώματος
του Τμήματος Πληροφορικής

Γεωργίου Π. Μαυρομμάτη

ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΕΚΘΕΤΙΚΟΥ ΧΡΟΝΟΥ ΜΕ ΣΥΓΧΡΟΝΙΣΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Συμβουλευτική Επιτροπή

Επιβλέπων

Ιωάννης-Χρήστος Παναγιωτόπουλος
Καθηγητής
Πανεπιστημίου Πειραιώς

Μέλη

Νικήτας Ασημακόπουλος
Καθηγητής
Πανεπιστημίου Πειραιώς

Γεώργιος Βασιλακόπουλος
Καθηγητής
Πανεπιστημίου Πειραιώς

Εξεταστική Επιτροπή

Πρόεδρος

Ιωάννης-Χρήστος Παναγιωτόπουλος
Καθηγητής
Πανεπιστημίου Πειραιώς

Μέλη

Νικήτας Ασημακόπουλος
Καθηγητής
Πανεπιστημίου Πειραιώς

Γεώργιος Βασιλακόπουλος
Καθηγητής
Πανεπιστημίου Πειραιώς

Βασίλειος Αγγελής
Καθηγητής
Πανεπιστημίου Αιγαίου

Θεόδωρος Αρτίκης
Καθηγητής
Πανεπιστημίου Πειραιώς

Ιωάννης Σίσκος
Καθηγητής
Πανεπιστημίου Πειραιώς

Νικήτας – Μαρίνος Σγούρος
Λέκτορας
Πανεπιστημίου Πειραιώς

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Η εκμετάλλευση της ισχύος περισσοτέρων του ενός επεξεργαστών προκειμένου να επιλυθούν μεγάλα και απαιτητικά σε υπολογισμούς προβλήματα υπήρξε στόχος της Πληροφορικής σχεδόν από τις πρώτες μέρες εμφάνισης της Επιστήμης αυτής.

Από πλευράς Υλικού η κατάσταση σήμερα απέχει αρκετά από αυτή που ήταν πριν από δέκα ή είκοσι χρόνια. Η εξέλιξη του μικροεπεξεργαστή αφενός και των δικτύων αφετέρου έχει οδηγήσει στις σημερινές συνθήκες όπου χιλιάδες, χαμηλού κόστους και υπολογίσιμης ισχύος μικροϋπολογιστές που επικοινωνούν μέσα από τοπικά ή και ευρύτερα δίκτυα, βρίσκονται πάνω στα γραφεία στη δουλειά και στα σπίτια των ανθρώπων.

Από την άλλη πλευρά υπάρχουν μεγάλες κατηγορίες σημαντικών προβλημάτων όπως για παράδειγμα τα NP-Complete που ακόμα και σήμερα, αρκετές δεκαετίες από τότε που αναγνωρίστηκαν και πρωτοαντιμετωπίστηκαν, εξακολουθούν να αντιστέκονται στις προσπάθειες της Επιστήμης να βρει πολυωνυμικού χρόνου αλγορίθμους και να τα επιλύσει αποτελεσματικά.

Ο κεντρικός σκοπός της παρούσης Διδακτορικής Διατριβής είναι η διερεύνηση της δυνατότητας χρησιμοποίησης των δικτύων από κοινούς, οικονομικούς μικροϋπολογιστές του εμπορίου χωρίς κανένα ιδιαίτερο εξοπλισμό σε Υλικό ή Λογισμικό προκειμένου να επιλυθούν δύσκολα (σκληρά) προβλήματα για τα οποία η υπάρχουσα απλή υπολογιστική ισχύς δεν επαρκεί.

Στην παρούσα διατριβή περιλαμβάνονται συνολικά πέντε (5) Κεφάλαια, οι Βιβλιογραφικές αναφορές καθώς και τρία Παραρτήματα.

- Το 1° Κεφάλαιο «*Διανεμημένη Υπολογιστική Ισχύς και Πολυπλοκότητα*», αναπτύσσεται σε δύο κύριους άξονες: α) Γίνεται παρουσίαση στοιχείων Θεωρίας Αλγορίθμων και Πολυπλοκότητας. Παρουσιάζονται οι βασικές κατηγορίες Προβλημάτων, οι μέθοδοι επίλυσης και αξιολόγησης των αλγορίθμων τους, με ιδιαίτερη βαρύτητα στα NP-Complete προβλήματα. β) Παρουσιάζονται στοιχεία Θεωρίας παραλλήλων & κατανεμημένων υπολογιστικών συστημάτων, η εξέλιξη, και κυρίως η κατάσταση που επικρατεί σήμερα από πλευράς Υλικού, Λογισμικού όσο και επίλυσης προβλημάτων με κατανεμημένα συστήματα.

- Στο 2° Κεφάλαιο «*Κατανεμημένο Σύστημα Επίλυσης*», αρχικά τίθενται οι αρχές πάνω στις οποίες σχεδιάστηκε το προτεινόμενο Σύστημα, που προκύπτουν άμεσα από την έρευνα που έγινε στο 1° Κεφάλαιο. Στη συνέχεια μαθηματικοποιείται η διαδικασία διάσπασης προβλήματος σε υποπροβλήματα με στόχο την ισοδυναμία των λύσεών τους και διατυπώνονται σχετικοί ορισμοί και θεωρήματα. Στο υπόλοιπο του Κεφαλαίου αναλύεται από Συστημικής πλευράς το Κατανεμημένο Σύστημα, χωρίζεται σε Υποσυστήματα που περιγράφονται λεπτομερώς μαζί με τους σχετικούς Αλγορίθμους αλληλοεπίδρασης.

- Στο 3° Κεφάλαιο «*Εφαρμογή στο CLIQUE-NUMBER*», δίνονται ορισμοί του προβλήματος εύρεσης της μέγιστης κλίκας και παρουσιάζονται οι κυριότερες γνωστές μέθοδοι επίλυσής του. Στη συνέχεια περιγράφεται μία μέθοδος διάσπασης του αρχικού προβλήματος (γραφήματος) σε υποπροβλήματα (υπογραφήματα) στηριγμένη στην θεωρία του Κεφαλαίου 2, καθώς και μία νέα

υλοποίηση παλαιότερης μεθόδου ακριβούς επίλυσης του maximum clique problem. Τέλος γίνεται λεπτομερής περιγραφή της προσαρμογής του Κατανεμημένου Συστήματος Επίλυσης στην εφαρμογή της εύρεσης της μέγιστης κλίκας.

- Στο 4ο Κεφάλαιο «*Μετρήσεις*», μετά από σύντομη περιγραφή του υπολογιστικού περιβάλλοντος παρουσιάζονται αναλυτικές μετρήσεις των χαρακτηριστικών του Κατανεμημένου Συστήματος Επίλυσης πάνω σε τυχαία γραφήματα αλλά και σε benchmarks. Μετρώνται διάφορες πτυχές και υποσυστήματα και εξάγονται συμπεράσματα. Τέλος γίνεται μέτρηση της απόδοσης (ταχύτητας) του Συστήματος αλλά και σύγκριση με άλλες πρόσφατες εργασίες.

- Στο 5ο Κεφάλαιο «*Γενικεύσεις*», παρουσιάζονται σκέψεις για περαιτέρω έρευνα και γενικεύσεις του Συστήματος που προέκυψαν από την μελέτη των χαρακτηριστικών του καθώς και τα πειραματικά δεδομένα. Περιγράφεται ένα Σύστημα όπου εκτός της διάσπασης σε τάξεις υποπροβλημάτων υπάρχει και ιεράρχηση του σε τάξεις Διαχειριστών. Επίσης περιγράφονται μέθοδοι κωδικοποιημένης συμπίεσης για ελάττωση του επικοινωνιακού κόστους, μέθοδοι λήψης αποφάσεων από τον Διαχειριστή για βελτίωση της απόδοσης καθώς και πρόληψη αστοχιών – λαθών του Συστήματος. Τέλος αναπτύσσονται γενικότερες σκέψεις και προβληματισμοί.

- Η «*Βιβλιογραφία*» περιλαμβάνει πλήρη αναφορά όλων των δημοσιεύσεων και εκδόσεων που χρησίμευσαν σαν βάση για την παρούσα διατριβή.

- Τέλος στα *«Παραρτήματα»* περιλαμβάνονται οι κώδικες του Διαχειριστή και Επιλυτή Προβλήματος καθώς και βοηθητικά ευρετήρια.

Από την συνολική θεώρηση της παρούσης εργασίας, συνάγεται ότι η συμβολή της στην ανάπτυξη του γνωστικού πεδίου εντοπίζεται στα εξής κύρια σημεία:

- Μελετάται η δυνατότητα απλών, φθηνών συστημάτων μικροϋπολογιστών να παράσχουν υπολογιστική ισχύ που κανονικά συναντάται σε πολύ ακριβότερους υπολογιστές με ειδικό εξοπλισμό υλικό & λογισμικό.

- Σχεδιάζεται και υλοποιείται ένα **Σύστημα – Πλαίσιο** με το οποίο μπορούν να επιλυθούν κατανεμημένα «σκληρά» προβλήματα βελτιστοποίησης, σε απλό δίκτυο μικροϋπολογιστών. Το εν λόγω Σύστημα έχει αναλυθεί σε υποσυστήματα και είναι αρθρωτό, ενώ μπορεί για ερευνητικούς σκοπούς να αφαιρεθούν ή να προστεθούν νέα υποσυστήματα.

- Αναπτύσσεται νέα μέθοδος ακριβούς επίλυσης της μέγιστης κλίκας, στηριγμένη σε παλαιότερη εργασία με Boolean Άλγεβρα, εντυπωσιακά ικανοποιητικές ταχύτητες και προφανή προοπτική με την μετάβαση των απλών υπολογιστών στα 64 bits.

- Αναπτύσσεται θεωρητικό υποβάθρο διάσπασης προβλήματος σε υποπροβλήματα. Η εφαρμογή του υποβάθρου στην περίπτωση της μέγιστης κλίκας είχε σαν αποτέλεσμα την υλοποίηση και μαθηματική περιγραφή μεθόδου διάσπασης γραφήματος σε υπογραφήματα προκειμένου να επιλυθούν κατανεμημένα σε εντυπωσιακούς υπολογιστικούς χρόνους.

- Δίνονται γενικεύσεις του προτεινομένου Συστήματος και προτείνεται μία μέθοδος κωδικοποίησης των δεδομένων μεταξύ των μικροεπεξεργαστών με κύριο στόχο την ελαχιστοποίηση του χρόνου προσπέλασης των δεδομένων σε επίπεδο Δικτύου και Μνήμης.

Κλείνοντας, από την θέση αυτή θέλω να εκφράσω τις θερμές μου ευχαριστίες προς το Πανεπιστήμιο Πειραιώς και ιδιαίτερα το Τμήμα Πληροφορικής για την ευκαιρία που μου έδωσαν να εκπονήσω την παρούσα διατριβή. Ειδικότερα θέλω να ευχαριστήσω την Τριμελή Συμβουλευτική Επιτροπή και συγκεκριμμένα:

Τον καθηγητή κ. Ι-Χ. Παναγιωτόπουλο, επιβλέποντα της παρούσης διατριβής. Η τιμή της συνεργασίας μαζί του, η εμπνευσμένη καθοδήγηση της έρευνας και η συνεχής υποστήριξη που μου παρείχε, επηρέασαν αποφασιστικά το αποτέλεσμα.

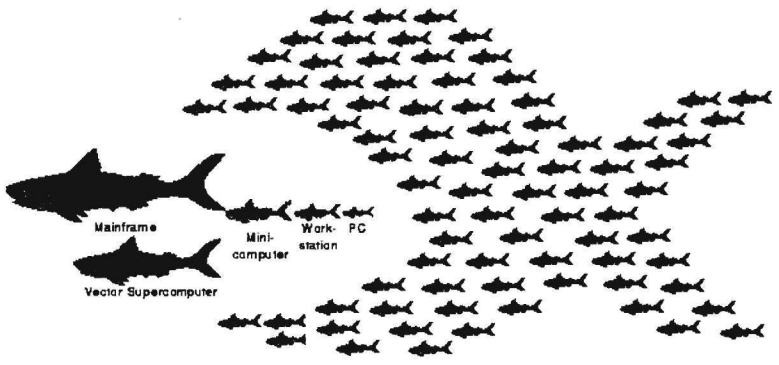
Τους καθηγητές κκ. Νικήτα Ασημακόπουλο και Γεώργιο Βασιλακόπουλο, μέλη της επιτροπής, για τις πολύτιμες συμβουλές και υποδείξεις που μου παρείχαν κατά την εκπόνηση της διατριβής.

Ευχαριστώ επίσης τους Καθηγητές κκ. Βασίλειο Αγγελή, Θεόδωρο Αρτίκη, Ιωάννη Σίσκο και Λέκτορα κ. Νικήτα Σγούρο, μέλη της Εξεταστικής μου Επιτροπής για τα εποικοδομητικά σχόλια τις παρατηρήσεις και διορθώσεις που υπέδειξαν, βελτιώνοντας την ποιότητα της διατριβής μου.

Τέλος, θέλω να εκφράσω ευχαριστίες στους γονείς μου και την σύζυγό μου Κωνσταντίνα για την, με κάθε τρόπο εκφρασμένη, βοήθεια και συμπαράσταση που μου παρείχαν καθόλη την διάρκεια της εκπόνησης της παρούσης διατριβής, την οποία και αφιερώνω στο γιο μου Παναγιώτη.

Γιώργος Π. Μαυρομμάτης

Πειραιάς, Μάϊος 2002



NOW

Εικόνα από το Site του NOW project στο Πανεπιστήμιο του Berkeley

Κεφάλαιο 1

ΔΙΑΝΕΜΗΜΕΝΗ ΥΠΟΛΟΓΙΣΤΙΚΗ ΙΣΧΥΣ ΚΑΙ ΠΟΛΥΠΛΟΚΟΤΗΤΑ

ΔΙΑΝΕΜΗΜΕΝΗ ΥΠΟΛΟΓΙΣΤΙΚΗ ΙΣΧΥΣ ΚΑΙ ΠΟΛΥΠΛΟΚΟΤΗΤΑ

1.1 Εισαγωγή

Προκειμένου να επιλυθεί ένα πρόβλημα είναι απαραίτητη η ύπαρξη ενός **Αλγορίθμου** που θα περιγράφει την διαδικασία επίλυσής του. Ωστόσο υπάρχουν προβλήματα που οι γνωστοί γι' αυτά αλγόριθμοι απαιτούν χρόνο ο οποίος αυξάνει εκθετικά με το μέγεθος τους. Ο αντίστοιχος κώδικας (πρόγραμμα) μπορεί να λειτουργεί σε έναν επεξεργαστή ή σε περισσότερους, μέσα στα πλαίσια τοπικού ή και ευρύτερου Δικτύου (Network), οπότε έχει νόημα η έννοια της **Διανεμημένης Υπολογιστικής Ισχύος** (**Distributed Processing**) [Xavier, Iyengar, 1998].

Οπωσδήποτε ο Αλγόριθμος επίλυσης του προβλήματος θα πρέπει να ικανοποιεί, μεταξύ άλλων, τις ακόλουθες Νόρμες:

- Ελαχιστοποίηση του αναμενόμενου υπολογιστικού χρόνου.
- Ελαχιστοποίηση της αναγκαίας κύριας μνήμης που απαιτεί το πρόβλημα.
- Ελαχιστοποίηση του χρόνου προσπέλασης.

Στην παρούσα διατριβή ιδιαίτερη σημασία θα δοθεί στην Νόρμα του αναμενόμενου υπολογιστικού χρόνου αλλά και στον χρόνο προσπέλασης με την γενικευμένη του έννοια που δεν περιορίζεται μόνον σε αναζήτηση και μεταφορά δεδομένων από επεξεργαστή σε περιφερειακά και αντίστροφα, αλλά και στην μεταφορά δεδομένων συνεννόησης και συντονισμού μεταξύ των επεξεργαστών στην περίπτωση της διανεμημένης επεξεργασίας.

1.2 Πολυπλοκότητα & Αλγόριθμοι

1.2.1 Προβλήματα

Ο όρος **Πρόβλημα** είναι γενικός και πολυμορφικός. Στο [Δημητράκος, 1964] αναφέρονται έντεκα κύριες επεξηγήσεις του όρου, ενώ από πλευράς φιλοσοφίας «Πρόβλημα είναι κάθε ζήτημα ή ερώτημα που αναμένει την λύση του». Κάτω από τον τελευταίο ορισμό μπορεί να περιέχει μέλη όπως, «Το Ενεργειακό Πρόβλημα» ή «Επίλυση Δευτεροβάθμιας εξίσωσης».

Στην παρούσα Διατριβή δεν ασχολούμαστε με κάθε πρόβλημα, αλλά μόνον με εκείνα που μπορούν να μαθηματικοποιηθούν και η επίλυσή τους να περιγραφεί μέσω ενός Αλγορίθμου που θα υλοποιηθεί σε υπολογιστή [Papadimitriou, Steiglitz, 1998].

Συνήθως, ένα Πρόβλημα περιέχει διάφορες παραμέτρους ή ελεύθερες μεταβλητές οι τιμές των οποίων είναι αδιευκρίνιστες. Το Πρόβλημα καθορίζεται δίνοντας

α) μία γενική περιγραφή όλων των παραμέτρων του, και

β) μία δήλωση των ιδιοτήτων που πρέπει να πληροί η απάντηση (**Λύση**).

Ένα **Στιγμιότυπο** του Προβλήματος προκύπτει όταν καθορίζονται συγκεκριμένες τιμές για όλες τις παραμέτρους του.

Προβλήματα Απόφασης (Decision Problems) λέγονται αυτά που η διαδικασία επίλυσής τους εκφράζεται με μια ακολουθία διατεταγμένων 0 - 1 στοιχείων (ΟΧΙ- ΝΑΙ).

Ορισμός 1.2.1.1 Στιγμιότυπο ενός προβλήματος βελτιστοποίησης είναι ένα ζεύγος (F,c), όπου F είναι το σύνολο - χώρος όλων των δυνατών (feasible) λύσεων, c είναι η συνάρτηση κόστους, μία απεικόνιση:

$$c : F \rightarrow R$$

Το πρόβλημα είναι να βρεθεί ένα σημείο $f \in F$ για το οποίο ισχύει:

$$c(f) \leq c(y) \ \forall \ y \in F$$

που ονομάζεται **καθολικά** (σφαιρικά) **βέλτιστη** (globally optimal) λύση του δεδομένου στιγμιότυπου, ή απλά αν δεν υπάρχει περίπτωση σύγχυσης, βέλτιστη λύση (optimal solution). Το σύνολο όλων των στιγμιότυπων I ονομάζεται **Πρόβλημα Βελτιστοποίησης (Optimization Problem)** [Papadimitriou, Steiglitz, 1998].

Τα προβλήματα που μπορούν να αντιμετωπιστούν σαν την επιλογή μίας βέλτιστης ανάθεσης τιμών σε μία ομάδα παραμέτρων προκειμένου να μεγιστοποιηθεί κάποια άλλη ενώ το σύνολο των δυνατών λύσεων αποτελείται από πεπερασμένες το πλήθος διακριτές τιμές, αναφέρονται σαν Προβλήματα **Συνδυαστικής Βελτιστοποίησης** (Combinatorial Optimization) [Papadimitriou, Steiglitz, 1998].

Ένα πρόβλημα βελτιστοποίησης μπορεί να μετασχηματιστεί σε πρόβλημα απόφασης, θεωρώντας μια τιμή - στόχο (φράγμα) για την μεταβλητή που θέλουμε να βελτιστοποιήσουμε και το πρόβλημα τεθεί στο αν η τιμή αυτή στόχος μπορεί να επιτευχθεί ή όχι.

Το **Μέγεθος** του **προβλήματος** (problem size, ισοδύναμα input size) εξαρτάται από το ίδιο το πρόβλημα, και είναι για παράδειγμα το μέγεθος του πίνακα των αριθμών που πρόκειται να ταξινομηθούν, ή προκειμένου περί γραφήματος εκφράζεται από το πλήθος των κόμβων ή / και των ακμών του [Papadimitriou, Steiglitz, 1998] [Cormen, Leiserson, Rivest, 2000].

Ένας **Αλγόριθμος** είναι μία καλά ορισμένη υπολογιστική διαδικασία που δέχεται κάποιες τιμές σαν είσοδο και παράγει κάποιες τιμές σαν έξοδο. Είναι δηλαδή μία σειρά από υπολογιστικά βήματα που μετατρέπει την είσοδο (input) σε έξοδο (output), ή διαφορετικά, ένα εργαλείο που λύνει ένα καλά ορισμένο υπολογιστικό πρόβλημα [Cormen, Leiserson, Rivest, 2000].

Ένας Αλγόριθμος λέγεται ότι επιλύει ένα Πρόβλημα όταν μπορεί να εφαρμοστεί σε κάθε στιγμιότυπο του Προβλήματος και είναι εξασφαλισμένο ότι θα δώσει λύση για κάθε τέτοιο στιγμιότυπο [Garey, Johnson, 1979].

Προκειμένου να επιλύσουμε ένα Πρόβλημα, ψάχνουμε να βρούμε τον πλέον **Αποτελεσματικό** (Efficient) αλγόριθμο. Ο όρος Αποτελεσματικός αλγόριθμος γενικά αφορά όλους τους πόρους που αυτός απαιτεί, όμως αυτό που ενδιαφέρει συνήθως είναι οι απαιτήσεις του σε υπολογιστικό χρόνο [Garey, Johnson, 1979]. "Time is the scarcest resource, and unless it is managed nothing else can be managed" – *Peter Drucker* [Gray, Larson, 2000].

Ο **Υπολογιστικός χρόνος** (time complexity) μεταξύ άλλων εξαρτάται και από το μέγεθος του προβλήματος και είναι το πλήθος των απλών βημάτων που χρειάζονται να εκτελεστούν. Θεωρούμε ότι κάθε βήμα i του κώδικα που εκτελείται, απαιτεί χρόνο έστω c_i.

Για παράδειγμα, έστω το απλό πρόβλημα ταξινόμησης των $a_i,\ i=1,2,...,n$

Αναλυτικός Αλγόριθμος	Κόστος	Επαναλήψεις
`Read(n);`	c_1	1
`for i=1 to n do read(a[i]);`	c_2	n
`for i= 1 to n-1 do`		(n-1)(n-1)
`for j=1 to n-1 do`		
`if a[j] > a[j+1] then`		
`swap a[j], a[j+1]`	c_3	
`for i=1 to n do write(a[i]);`	c_4	n

Ο υπολογιστικός χρόνος που απαιτεί αυτός ο αλγόριθμος είναι το άθροισμα των χρόνων των επιμέρους βημάτων:

$$T(n) = c_1 1 + c_2 n + c_3 (n-1)(n-1) + c_4 n =$$

$$c_3 n^2 + (c_2 + c_4) n + (c_1 - c_3) =$$

$an^2 + bn + c$, a,b,c σταθερές.

Η οριακή συμπεριφορά του υπολογιστικού χρόνου όταν το n αυξάνει, είναι αυτή που καθορίζει το μέγεθος των προβλημάτων που μπορεί να λύσει ο αλγόριθμος. Έτσι, για μεγάλες τιμές του n, ο όρος an^2 και μάλιστα ο n^2 είναι αυτός που καθορίζει το μέγεθος, τον ρυθμό αύξησης (growth rate) της παραπάνω παράστασης διότι το a είναι σταθερό.

Γράφουμε λοιπόν ότι ο αλγόριθμος τρέχει σε χρόνο $\Theta(n^2)$ που λέγεται πολυωνυμικός ενώ ένας αλγόριθμος π.χ. $\Theta(2^n)$ λέμε ότι τρέχει σε εκθετικό χρόνο. Οι **ασυμπτωτικοί συμβολισμοί** χρησιμοποιούνται για να εκφράσουν τον υπολογιστικό χρόνο που απαιτεί ένας αλγόριθμος [Miller, Boxer, 2000] [Cormen, Leiserson, Rivest, 2000]. Οι κυριότεροι από αυτούς, καθώς και βασικές ιδιότητές τους φαίνονται στους Πίνακες 1 και 2.

Τα υπολογιστικά βήματα που εκτελεί ο αλγόριθμος του παραδείγματος, είναι ανεξάρτητα από τα δεδομένα για σταθερό n (oblivious algorithm) που όμως δεν συμβαίνει σε κάθε περίπτωση. Για παράδειγμα, στη μέθοδο ταξινόμησης insertion sort η καλύτερη περίπτωση (**best case**) είναι όταν τα δεδομένα είναι ήδη ταξινομημένα. Αποδεικνύεται ότι τότε ο αλγόριθμος insertion sort τρέχει σε χρόνο $T(n) = an + b = \Theta(n)$ που είναι γραμμικός (**linear**).

Η χειρότερη περίπτωση (**worst case)** είναι όταν ο πίνακας των δεδομένων είναι ταξινομημένος αντίστροφα από την επιθυμητή διάταξη, οπότε αποδεικνύεται ότι ο αλγόριθμος τρέχει σε χρόνο $T(n) = an^2 + bn + c = \Theta(n^2)$ που είναι τετραγωνικός ως προς n, ενώ η μέση αντιπροσωπευτική περίπτωση είναι μία κατάσταση ανάμεσα στις δύο παραπάνω. Αποδεικνύεται ότι και σ' αυτήν ο αλγόριθμος insertion sort τρέχει σε τετραγωνικό χρόνο που λέγεται αναμενόμενος υπολογιστικός χρόνος (**average-case, expected running time**).

Πίνακας 1 - Βασικοί ασυμπτωτικοί συμβολισμοί

1. Η $g(n)$ είναι ένα **ασυμπτωτικά στενό φράγμα** για την $f(n)$:

$$f(n) = \Theta(g(n)) \text{ αν και μόνον αν}$$

$$\exists\, c_1, c_2, n_0 > 0 : \quad c_1 g(n) \le f(n) \le c_2 g(n) \quad \forall n \ge n_0$$

2. Η $g(n)$ είναι ένα **ασυμπτωτικά άνω φράγμα** για την $f(n)$:

$f(n) = O(g(n))$ αν και μόνον αν $\exists c, n_0 > 0 : \; f(n) \le cg(n) \quad \forall n \ge n_0$

3. Η $g(n)$ είναι ένα **ασυμπτωτικά κάτω φράγμα** για την $f(n)$:

$f(n) = \Omega(g(n))$ αν και μόνον αν $\exists c, n_0 > 0 : \quad cg(n) \le f(n) \quad \forall n \ge n_0$

όπου f, g θετικές συναρτήσεις του n.

Πίνακας 2 - Βασικές ιδιότητες ασυμπτωτικών συμβολισμών	
1.	$f(n) = \Theta(g(n))$ αν και μόνον αν $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$
2.	$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$
3.	$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$
4.	$f(n) = \Delta(g(n)) \wedge g(n) = \Delta(h(n)) \Rightarrow f(n) = \Delta(h(n))$ όπου $\Delta \in \{"\Theta","O","\Omega"\}$
5.	$f(n) = \Delta(f(n))$, όπου $\Delta \in \{"\Theta","O","\Omega"\}$
7.	Ένας αλγόριθμος τρέχει σε χρόνο $\Theta(g(n))$ αν και μόνο αν τρέχει στην χειρότερη περίπτωση σε χρόνο $O(g(n))$ και στην καλύτερη σε $\Omega(g(n))$

Για να γίνει κατανοητή η διαφορά στην απόδοση δύο αλγορίθμων $\Theta(n^2)$ και $\Theta(2^n)$, σημειώνεται ότι για μέγεθος προβλήματος $n = 100$ και υπολογιστή που απαιτεί 10^{-7} min/υπολογιστικό βήμα, ο μεν πολυωνυμικός αλγόριθμος χρειάζεται χρόνο 0,06 sec ενώ ο εκθετικός περισσότερα από 10^{11} έτη! Ο αριθμός των πρωτονίων στο γνωστό Σύμπαν έχει 79 ψηφία, ενώ για $n = 300$ η τιμή του 2^n έχει 91 ψηφία [Hromkovic, 2001].

Γενικά, αποτελεσματικοί (feasible, tractable) θεωρούνται οι πολυωνυμικοί αλγόριθμοι, αυτοί που ο υπολογιστικός τους χρόνος είναι στη χειρότερη περίπτωση $O(n^k)$, πολυώνυμο δηλαδή του μεγέθους της εισόδου n, με k σταθερό. Βέβαια, τείνει να γίνει παγκόσμια σιωπηρή συμφωνία να ισχύει $k \leq 5$. Η θέση αυτή προέρχεται κυρίως από την εμπειρία σε συνδυασμό με σειρά από χαρακτηριστικές ιδιότητες που έχουν οι πολυωνυμικού χρόνου αλγόριθμοι [Papadimitriou, Steiglitz, 1998] [Cormen, Leiserson, Rivest, 2000]. Προφανώς αποτελεσματικός είναι και ένας αλγόριθμος που ο χρόνος του φράσσεται άνω από ένα πολυώνυμο.

Σύμφωνα με την **Feasibility Thesis** στην [Cook, 1991], ένα φυσικό πρόβλημα έχει έναν αποτελεσματικό (feasible) αλγόριθμο αν και μόνον αν ο αλγόριθμος αυτός είναι πολυωνυμικού χρόνου.

Σ' αυτήν τη Θέση υπάρχουν αντιπαραδείγματα [Cook, 2000] όπως προβλήματα [Robertson, Seymour, 1995] που ενώ επιλύονται σε πολυωνυμικό χρόνο $O(n^3)$ οι αλγόριθμοι αυτοί δεν είναι αποτελεσματικοί λόγω των **πολύ μεγάλων συντελεστών** που κρύβονται.

Από την άλλη πλευρά, οι αλγόριθμοι που τρέχουν σε χρόνο που δεν έχει κανένα πολυώνυμο σαν άνω φράγμα γενικά θεωρούνται **δύσχρηστοι** (intractable) και ονομάζονται εκθετικού χρόνου (exponential), διότι μία αντιπροσωπευτική περίπτωση είναι αυτοί που τρέχουν σε χρόνο $O(2^n)$.

Ήδη από το 1953 ο Von Neumann [Neumann, 1953] [Papadimitriou, Steiglitz, 1998] έκανε διαχωρισμό ανάμεσα σε πολυωνυμικούς και εκθετικούς αλγόριθμους, η αυστηρή έννοια όμως του πολυωνυμικού χρόνου δόθηκε στην δεκαετία του '60 από τους [Cobham, 1964] [Edmonds, 1965].

Παρόλα αυτά, υπάρχουν περιπτώσεις αλγορίθμων που ενώ έχει αποδειχτεί ότι δεν είναι πολυωνυμικοί, στην πράξη έχουν μεγάλη **πιθανότητα πολυωνυμικής συμπεριφοράς**. Κλασικό παράδειγμα αποτελεί ο **Αλγόριθμος Simplex** ο οποίος ενώ βρίσκει βέλτιστη λύση σε προβλήματα Γραμμικού Προγραμματισμού με εκατοντάδες μεταβλητές και χιλιάδες περιορισμών είναι αποδεδειγμένα εκθετικός και μη κλειστός στην worst-case [Papadimitriou, Steiglitz, 1998]. Δεν πρέπει να διαφύγει της προσοχής ότι σχετικά πρόσφατα έχουν ανακαλυφθεί αλγόριθμοι πολυωνυμικού χρόνου για τα προβλήματα γραμμικού προγραμματισμού [Khachiyan, 1979] [Karmarkar, 1984] [Papadimitriou, 1995], οι οποίοι όμως μειονεκτούν στην πράξη σε άλλες παραμέτρους.

1.2.2 Πολυπλοκότητα

Η **Θεωρία Πολυπλοκότητας** (Computational Complexity) έχει σαν κύριο στόχο την ταξινόμηση των προβλημάτων ανάλογα με τον βαθμό «σκληρότητάς» τους, ανάλογα δηλαδή με το «πόσο δύσκολα είναι» να επιλυθούν.

Μη φορμαλιστικά, **Τάξη Πολυπλοκότητας** είναι ένα σύνολο από προβλήματα που κατατάσσονται σ' αυτό με βάση κάποιο **Κριτήριο Πολυπλοκότητας** κάποιο δηλαδή φράγμα σε έναν από τους πόρους που απαιτούνται προκειμένου να επιλυθούν από κάποιον αλγόριθμο [Cormen, Leiserson, Rivest, 2000] [Papadimitriou, 1995].

Τα προβλήματα σήμερα, σύμφωνα με τη διεθνή βιβλιογραφία, χωρίζονται στις ακόλουθες κατηγορίες:

- **P**: Προβλήματα επιλύσιμα σε πολυωνυμικό χρόνο.
- **NP**: Προβλήματα που η ορθότητα της λύσης τους (εφόσον την γνωρίζουμε) μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο. Τα **NP-Complete** προβλήματα περιέχονται σ' αυτήν την κλάση.
- **PSPACE**: Προβλήματα πολυωνυμικά ως προς το μέγεθος της εισόδου, που χρησιμοποιούν δηλαδή σχετικά μικρό μέγεθος μνήμης προκειμένου να επιλυθούν, ανεξάρτητα από το χρόνο που απαιτείται.
- **EXPTIME**: Προβλήματα που μπορούν να λυθούν σε εκθετικό χρόνο.
- **Undecidable**: Προβλήματα που αποδεδειγμένα δεν υπάρχει αλγόριθμος που να τα λύνει σε κάθε περίπτωση, ανεξάρτητα από το χρόνο ή το χώρο που διατίθεται.

Ανάμεσα στις πολλές προσπάθειες που έχουν γίνει για να μαθηματικοποιηθεί η έννοια του Αλγορίθμου – Μοντέλου επεξεργασίας η **Μηχανή Turing** (DTM) αποτελεί την πλέον προβεβλημένη [Manna, 1974] [Papadimitriou, 1995].

Ορισμός 1.2.2.1 Η κλάση **P** περιλαμβάνει όλα εκείνα τα προβλήματα απόφασης που λύνονται σε πολυωνυμικό χρόνο από μια DTM (δηλαδή από έναν υπαρκτό υπολογιστή).

Ωστόσο, πιο κοντά στο μοντέλο των σύγχρονων σειριακών υπολογιστών βρίσκεται το random access machine (RAM model) [Aho, Hopcroft, Ullman, 1974] ενώ πολλές προσεγγίσεις τείνουν να αποφύγουν την ιδέα του nondeterminism άρα και τη μηχανή Turing, και χρησιμοποιούν την ιδέα του **certificate verification** προκειμένου να ορίσουν την κλάση NP [Papadimitriou, Steiglitz, 1998] [Cormen, Leiserson, Rivest, 2000] [Hromkovic, 2001].

Ορισμός 1.2.2.2 Η τάξη πολυπλοκότητας **NP** περιέχει όλα εκείνα τα προβλήματα των οποίων η λύση μπορεί να **επαληθευτεί** από έναν πολυωνυμικό αλγόριθμο.

Ο συμβολισμός NP προέρχεται από τον όρο "nodeterministic polynomial time" λόγω του ορισμού της τάξης σαν το σύνολο των προβλημάτων που επιλύονται σε πολυωνυμικό χρόνο από ένα μη ντετερμινιστικού τύπου υπολογιστικό μοντέλο [Aho, Hopcroft, Ullman, 1974].

Προφανώς ισχύει $P \subseteq NP$. Το ερώτημα αν ισχύει και $NP \subseteq P$ (οπότε $P = NP$) παραμένει ανοικτό μέχρι σήμερα, με τους περισσότερους επιστήμονες να θεωρούν ότι δεν ισχύει για διάφορους λόγους [Cormen, Leiserson, Rivest, 2000] [Hromkovic, 2001].

Η θεωρία **NP-Completeness** αφορά στην διάκριση ανάμεσα στα προβλήματα P και NP: Η κατηγορία **NP-Complete** περιέχει τα προβλήματα για τα οποία οι **υπάρχοντες** αλγόριθμοι επίλυσης είναι εκθετικού χρόνου, δεν έχει δηλαδή βρεθεί μέχρι σήμερα πολυωνυμικός αλγόριθμος που να τα επιλύει, ούτε όμως έχει **αποδειχτεί** ότι είναι εκθετικού χρόνου, ότι δηλαδή ο χρόνος επίλυσής τους φράσσεται κάτω από ένα υπέρ-πολυωνυμικό φράγμα.

Τα πρώτα προβλήματα που αποδείχτηκε [Cook, 1971] ότι ανήκουν στην κατηγορία αυτή είναι:

- **SATISFIABILITY problem**: «Δοθέντων m όρων $C_1, C_2, ...C_m$ που περιέχουν τις Boolean μεταβλητές $x_1, x_2, x_3, ...x_n$, είναι η παράσταση $C_1 \bullet C_2 \bullet ...C_m$ επιλύσιμη (satisfiable);» Η παράσταση λέγεται επιλύσιμη όταν υπάρχει μία ανάθεση τιμών (*true* ή *false*) στις μεταβλητές $x_1, x_2, x_3, ...x_n$ ώστε να παίρνει την τιμή *true* [Papadimitriou, Steiglitz, 1998].
- **3-SATISFIABILITY problem**: πρόκειται για το πρόβλημα SATISFIABILITY όπου σε κάθε όρο $C_1, C_2, ...C_m$ υπάρχουν τρεις μεταβλητές (το καλύτερο άνω φράγμα που έχει αποδειχτεί σε αλγόριθμο για αυτό είναι περίπου 1.5^m) [Cook, 2000].

Να σημειωθεί ότι το πρόβλημα **2-SATISFIABILITY** δεν ανήκει στα NP-Complete Προβλήματα, λύνεται σε γραμμικό χρόνο άρα ανήκει στο σύνολο P.

Στα [Cook, 1971] [Karp, 1972] τέθηκαν τα θεμέλια της θεωρίας NP-Completeness. Δόθηκε έμφαση στην έννοια του **μετασχηματισμού πολυωνυμικού χρόνου** (polynomial time reducibility), εστιάστηκε το ενδιαφέρον στα Προβλήματα Απόφασης και δόθηκε καθαρά η σχέση των NP-Complete προβλημάτων με το πεδίο της Συνδυαστικής Βελτιστοποίησης.

1.2.3 Μετασχηματισμοί

Ένα **Αλφάβητο** Σ είναι ένα πεπερασμένο σύνολο από σύμβολα. Μια **Γλώσσα** L πάνω από το Σ είναι οποιοδήποτε σύνολο από σειρές στοιχείων του Σ (strings). Η Γλώσσα που περιέχει όλα τα strings πάνω από το Σ συμβολίζεται με Σ^*.

Στην περίπτωση που $\Sigma = \{0,1\}$ (δυαδικό αλφάβητο), τότε $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$, ε το κενό string.

Για κάθε γλώσσα L πάνω από το Σ ισχύει ότι $L \subseteq \Sigma^*$. Μία γλώσσα μπορεί να χρησιμοποιηθεί για να περιγράψει την είσοδο (input) ενός αλγοριθμικού προβλήματος.

Ορισμός 1.2.3.1 Έστω $L1 \subseteq \Sigma_1^*$ και $L_2 \subseteq \Sigma_2^*$ δύο γλώσσες. Λέμε ότι η L_1 **μετασχηματίζεται σε πολυωνυμικό χρόνο** στην L_2 (polynomial-time reducible), $L_1 \leq_p L_2$ αν υπάρχει ένας αλγόριθμος Α πολυωνυμικού χρόνου που υπολογίζει μία απεικόνιση από το Σ_1^* στο Σ_2^* έτσι ώστε $\forall x \in \Sigma_1^*$ να ισχύει

$$x \in L_1 \Leftrightarrow A(x) \in L_2$$

Ο Α λέγεται μετασχηματισμός πολυωνυμικού χρόνου από την L_1 στην L_2 (polynomial-time reduction).

Μία γλώσσα L λέγεται **NP-Hard** αν $\forall U \in NP, U \leq_p L$

Μια γλώσσα L λέγεται **NP-Complete** αν

i) $L \in NP$ και

ii) η L είναι NP-Hard.

Ισχύουν τα παρακάτω Θεωρήματα [Cormen, Leiserson, Rivest, 2000] [Papadimitriou, Steiglitz, 1998] [Hromkovic, 2001]:

- Έστω $L_1, L_2 \subseteq \{0,1\}^*$ γλώσσες τέτοιες ώστε $L_1 \leq_p L_2$. Τότε, $L_2 \in P \Rightarrow L_1 \in P$.

- Έστω $L_1, L_2 \subseteq \{0,1\}^*$ γλώσσες τέτοιες ώστε $L_1 \leq_p L_2$. Τότε, εάν L_1 είναι NP-Hard, θα είναι και L_2 NP-Hard.

- Εάν L είναι NP-Hard και $L \in P$ τότε $P = NP$. Εάν οποιοδήποτε πρόβλημα του NP δεν επιλύεται σε πολυωνυμικό χρόνο, τότε κανένα πρόβλημα NP-Complete δεν λύνεται σε πολυωνυμικό χρόνο.

- Αν L είναι γλώσσα τέτοια ώστε $L' \leq_p L$ για κάποιο $L' \in$ NP-Complete τότε η L είναι NP-Hard. Αν $L \in NP$ τότε $L \in$ NP-Complete.

Με βάση τα παραπάνω και αρχικό πρόβλημα – ρίζα το SATISFIABILITY, έχει αποδειχτεί η ύπαρξη χιλιάδων προβλημάτων στην κλάση NP-Complete (περισσότερα από 3000 προβλήματα, κατά άλλους $\simeq 1000$ [Cook, 2000]). Βέβαια οποιοδήποτε αποδεδειγμένα NP-Complete πρόβλημα μπορεί να χρησιμοποιηθεί για να αποδειχτεί ότι ένα νέο πρόβλημα είναι NP-Complete, στην πράξη όμως υπάρχει ένας πυρήνας προβλημάτων που φαίνεται να τους ταιριάζει καλύτερα αυτός ο ρόλος [Garey, Johnson, 1979].

Πρόκειται για τα :

- **3-SATISFIABILITY (3-SAT)**. Ήδη ορίστηκε στην παράγραφο 1.2.2.

- **CLIQUE-NUMBER**. Δοθέντος ενός γραφήματος $G = (V, E)$ και ενός θετικού ακεραίου k, ζητείται να βρεθεί αν το G περιέχει κλίκα μεγέθους k, ένα υποσύνολο δηλαδή του G με πληθάριθμο k, του οποίου κάθε ζεύγος κόμβων συνδέεται με ακμή του E [Cormen, Leiserson, Rivest, 2000] (3-SAT $\leq_p$ CLIQUE-NUMBER).

- **3-DIMENSIONAL MATCHING (3DM)**. Δοθέντος ενός συνόλου $T \subseteq U \times V \times W$, όπου U, V, W σύνολα με ίδιο πληθάριθμο έστω q, ζητείται να βρεθεί αν το T περιέχει ένα matching, δηλαδή αν υπάρχει $M \subseteq T$ ώστε $|M| = q$ και κάθε 2 στοιχεία του M να είναι διαφορετικά ως προς όλες τις συντεταγμένες τους [Papadimitriou, Steiglitz, 1998] (SAT $\leq_p$ 3DM).

- **VERTEX COVER (VC)**. Δοθέντος ενός γραφήματος $G = (V, E)$ και ενός ακεραίου $k \leq |V|$, ζητείται να βρεθεί αν υπάρχει ένα υποσύνολο $V' \subseteq V$, τέτοιο ώστε $|V'| \leq k$ και για κάθε ακμή $\{u, v\} \in E$ τουλάχιστον ένα από τα u, v να ανήκει στο V' [Garey, Johnson, 1979] (3-SAT $\leq_p$ VC).

- **HAMILTONIAN CIRCUIT (HC)**. Δοθέντος ενός γραφήματος $G = (V, E)$, ζητείται να βρεθεί αν υπάρχει μια διάταξη $< v_1, v_2, ..., v_n >$ των κόμβων του G, όπου $n = |V|$ τέτοια ώστε $\{v_n, v_1\} \in E$ και $\{v_i, v_{i+1}\} \in E \ \forall i = 1, 2, ..., n-1$.

[Garey, Johnson, 1979] (VC $\leq_p$ HC).

- **PARTITION**. Δοθέντος ενός πεπερασμένου συνόλου A και ενός μεγέθους $s(a) \in \mathbb{Z}^+ \ \forall a \in A$, ζητείται να βρεθεί αν υπάρχει υποσύνολο $A' \subseteq A$ τέτοιο ώστε

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$$

[Garey, Johnson, 1979] (PARTITION $\leq_p$ 3DM).

Άλλα γνωστά, χαρακτηριστικά NP-Complete ή NP-Hard προβλήματα που τελικά ανάγονται στον πυρήνα προβλημάτων που δόθηκε, είναι τα ακόλουθα:

- **GRAPH K-COLORABILITY**. Δοθέντος ενός γραφήματος $G = (V, E)$ και ενός ακεραίου $0 \leq k \leq |V|$, ζητείται να βρεθεί αν το γράφημα μπορεί να χρωματιστεί με k χρώματα, δηλαδή αν υπάρχει συνάρτηση $f : V \rightarrow \{1, 2, \ldots, k\}$ τέτοια ώστε $f(u) \neq f(v), \forall \{u, v\} \in E$ [Garey, Johnson, 1979].

- **INDEPENDENT SET**. Δοθέντος ενός γραφήματος $G = (V, E)$ και ενός ακεραίου $k \leq |V|$, ζητείται να βρεθεί αν το γράφημα περιέχει ένα ανεξάρτητο σύνολο μεγέθους k ή μεγαλύτερου, δηλαδή ένα σύνολο $V' \subseteq V$ τέτοιο ώστε $|V'| \geq k$ και $\forall u, v \in V', \{u, v\} \notin E$ [Garey, Johnson, 1979].

- **TRAVELING SALESMAN PROBLEM (TSP)**. Δίνεται ένας ακέραιος n > 0 και η απόσταση ανάμεσα σε κάθε ζεύγος από n πόλεις (π.χ. στη μορφή ενός $n \times n$ πίνακα $[d_{ij}]$, όπου $d_{ij} \in \mathbb{Z}^+$). Ζητείται να βρεθεί μία **κλειστή διαδρομή** (ίδια αρχή και τέλος) που να επισκέπτεται όλες τις πόλεις μία ακριβώς φορά (tour) (Hamiltonian Circuit) και η διαδρομή να έχει το ελάχιστο μήκος [Papadimitriou, Steiglitz, 1998].

- **KNAPSACK PROBLEM (KP)**. Δίνονται οι ακέραιοι $c_j, j = 1,2,...,n$ και ο ακέραιος k και ζητείται να βρεθεί αν υπάρχουν ακέραιοι $x_j \geq 0,\ j = 1,2,...,n$ τέτοιοι ώστε $\sum_{j=1}^{n} c_j x_j = k$ [Papadimitriou, Steiglitz, 1998].

- **BIN PACKING**. Δίνονται N θετικοί ακέραιοι $a_1, a_2, ..., a_N$ (items) και δύο ακέραιοι C (capacity) και B (number of bins). Ζητείται να βρεθεί αν οι αριθμοί μπορούν να διαμεριστούν σε B το πλήθος υποσύνολα που τα στοιχεία του καθενός να έχουν άθροισμα το πολύ C [Papadimitriou, 1995].

Να σημειωθεί ότι τα **CLIQUE-NUMBER, VERTEX COVER, INDEPENDENT SET** συνδέονται με το παρακάτω [Papadimitriou, Steiglitz, 1998]

Θεώρημα 1.2.3.1 Έστω $G = (V, E)$ ένα γράφημα και $S \subseteq V$.Τα ακόλουθα είναι ισοδύναμα:

(α) το S είναι κλίκα του G.

(β) Το S είναι ένα ανεξάρτητο σύνολο του $\overline{G}$, $\overline{G}$ το συμπλήρωμα του G.

(γ) Το $V - S$ είναι ένα κάλυμμα των κόμβων του $\overline{G}$.

Είναι φανερό από το παραπάνω θεώρημα ότι καθένα από τα τρία αναφερόμενα προβλήματα μπορεί να θεωρηθεί σαν διαφορετική έκδοση των άλλων δύο [Garey, Johnson, 1979].

1.2.4 Επίλυση

Όπως ήδη αναφέρθηκε, κάθε πρόβλημα για το οποίο δεν υπάρχει γνωστός πολυωνυμικός αλγόριθμος επίλυσης, ονομάζεται **Σκληρό** (Hard Problem). Οπωσδήποτε, η κατάταξη ενός προβλήματος σαν NP-Complete ή NP-Hard, δεν σημαίνει ότι ένα **οποιοδήποτε** στιγμιότυπό του θα είναι το ίδιο δύσκολο να επιλυθεί. Σημαίνει μάλλον ότι κάποια από τα στιγμιότυπά του θα είναι σκληρά, μία ανάλυση της χειρότερης περίπτωσης (worst-case).

- Πολλά NP-Hard προβλήματα σε ειδικές τους περιπτώσεις λύνονται σε πολυωνυμικό χρόνο, για παράδειγμα το CLIQUE-NUMBER λύνεται σε πολυωνυμικό χρόνο για μία σειρά από ειδικές κατηγορίες γραφημάτων (planar graphs κ.ά.).

- Υπάρχουν προβλήματα με αλγόριθμους **ψεύδο-πολυωνυμικού χρόνου**, για παράδειγμα το MULTIPROCESSOR SCHEDULING λύνεται σε πολυωνυμικό χρόνο για 2 επεξεργαστές [Papadimitriou, Steiglitz, 1998] και σε ψεύδο-πολυωνυμικό χρόνο για οποιονδήποτε σταθερό αριθμό επεξεργαστών [Garey, Johnson, 1979], ενώ το SUBSET-SUM λύνεται πάντα σε ψεύδο-πολυωνυμικό χρόνο.

- Έχει παρατηρηθεί (ακόμα και για strong NP-Hard προβλήματα, όπως είναι το HAMILTON CIRCUIT κ.ά.) ότι καθώς μεταβάλλονται οι παράμετροί τους, εμφανίζουν μία «**μετάπτωση φάσης**» (fase transition) που ξεχωρίζει τα εύκολα από τα δύσκολα στιγμιότυπα [Cheeseman, Kanefsky, Taylor, 1991] [Pennock, Stout, 1996] [Mertens, 1998].

Η Τεχνολογία των Αλγορίθμων παρέχει μία σειρά από **τεχνικές σχεδίασης** που έχουν ξεπηδήσει μέσα από χρόνια ανάπτυξης και έρευνας. Τέτοιες κλασσικές τεχνικές σχεδιασμού αλγορίθμων είναι [Hromkovic, 2001] [Cormen, Leiserson, Rivest, 2000]:

- **Διαίρει-και-Βασίλευε** (Divide-and-Conquer).
- **Δυναμικός Προγραμματισμός** (Dynamic Programming).
- **Backtracking**.
- **Τοπική έρευνα** (Local Search).
- **Άπληστοι Αλγόριθμοι** (Greedy algorithms).

Ειδικότερα, στην περίπτωση των Σκληρών Προβλημάτων η επίλυση των περισσότερων μπορεί να εκφραστεί σαν έρευνα της βέλτιστης λύσης ανάμεσα σε ένα πεπερασμένο (αλλά πολύ μεγάλου πληθαρίθμου) σύνολο δυνατών λύσεων. Πολυωνυμικός αλγόριθμος δεν υπάρχει για αυτά, ενώ η μέθοδος της **εξαντλητικής απαρίθμησης** (exhaustive enumeration) και ελέγχου όλων των δυνατών λύσεων προκειμένου να βρεθεί η βέλτιστη, είναι αναποτελεσματική διότι απαιτεί εκθετικό χρόνο ακόμα και για μεσαία μεγέθη προβλημάτων [Grotschel, Lovasz, 1993]. Διακρίνονται τέσσερις βασικές κατηγορίες αντιμετώπισης αυτών των προβλημάτων:

- **Ντετερμινιστικές Προσεγγίσεις** (Deterministic Approaches). Περιλαμβάνει Αλγόριθμους ψεύδο-πολυωνυμικού χρόνου, Αλγόριθμους Παραμετροποιημένης Πολυπλοκότητας με βάση κάποιο μέγεθος της εισόδου, Αλγόριθμους Branch-and-Bound, Αλγόριθμους ελάττωσης της worst-case πολυπλοκότητας, Τοπική Έρευνα, Μετάπτωση σε πρόβλημα Γραμμικού Προγραμματισμού [Hromkovic, 2001].

- **Προσεγγιστικοί Αλγόριθμοι** (Approximation Algorithms). Πρόκειται για αλγόριθμους που δεν παράγουν βέλτιστη λύση (optimal solution) αλλά λύσεις που εγγυημένα απέχουν το πολύ κάποιο καθορισμένο ποσοστό $\varepsilon\%, \varepsilon > 0$ από την βέλτιστη. Με κατάλληλη ρύθμιση του ε, υπάρχουν NP-Complete προβλήματα που λύνονται σε πολυωνυμικό χρόνο [Papadimitriou, Steiglitz, 1998] [Hromkovic, 2001].

- **Τυχαιοποιημένοι Αλγόριθμοι** (Randomized Algorithms). Είναι μη ντετερμινιστικοί αλγόριθμοι που κάποιες φορές αποφασίζουν το επόμενο βήμα τους τυχαία με βάση την τιμή 0 ή 1 κάποιου bit. Προφανώς κάθε τρέξιμο αυτού του τύπου του αλγορίθμου μπορεί να δίνει διαφορετικό αποτέλεσμα, μία και εξαρτάται από την τιμή του επόμενου bit [Hromkovic, 2001].

- **Ευρετικές Μέθοδοι** (Heuristics). Πρόκειται για κάθε είδους αλγοριθμική προσέγγιση η οποία δεν εγγυάται την απόδοσή της [Papadimitriou, Steiglitz, 1998]. Μία ευρετική μέθοδος βασίζεται σε κάποια, συνήθως απλή ιδέα, έρευνας του χώρου των δυνατών λύσεων χωρίς κανείς να μπορεί να αποδείξει πάντοτε ότι η λύση που παράγει έχει κάποια ποιότητα [Hromkovic, 2001]. Τα πλέον γνωστά τέτοια παραδείγματα είναι η GRASP [Aiex, Resende, Ribeiro, 2000] [Festa, Resende, 2001], Simulated Annealing [Hromkovic, 2001], Tabu Search [Glover, 1993], Genetic Algorithms [Goldberg, 1989]. Ευρετική μέθοδος που σε όλες τις περιπτώσεις στιγμιότυπων του προβλήματος βρίσκει λύση ίση ή καλύτερη από τα $2/3$ της βέλτιστης θεωρείται ότι είναι επαρκής και δίνει «καλές λύσεις» (good solutions).

1.3 Υπολογιστικά Συστήματα

"Life was simple before World War II. After that, we had systems." Grace Hopper

1.3.1 Απλή Υπολογιστική Ισχύς

Η αρχιτεκτονική των σύγχρονων, σειριακών υπολογιστικών μηχανών (RAM – Random Access Machine), που σήμερα βρίσκονται διαδεδομένες σε όλο τον πλανήτη, αρχικά προτάθηκε από τον Von Neumann και περιλαμβάνει [Miller R., Boxer, 2000] **Μνήμη, Επεξεργαστή, Μονάδα προσπέλασης μνήμης.** Η **Εκτέλεση** των εντολών γίνεται σειριακά, η μία κατόπιν της άλλης από έναν επεξεργαστή ο οποίος έχει αποκλειστική πρόσβαση στη μνήμη (RAM) όπου βρίσκονται αποθηκευμένα τόσο τα δεδομένα όσο και ο κώδικας που εκτελείται. Στα 50 περίπου χρόνια που μεσολάβησαν, έγιναν τεράστιες βελτιώσεις στην υπολογιστική ταχύτητα, που όμως κατά μεγάλο μέρος οφείλονταν στην εξέλιξη της τεχνολογίας των ηλεκτρονικών, ενώ το βασικό **μοντέλο του Von Neumann** έχει παραμείνει το ίδιο με πολύ μικρές αλλαγές και βελτιώσεις.

Όμως παρόλη την εξέλιξη, πολλοί τομείς της επιστήμης έχουν τεράστια (στην ουσία απεριόριστη) ανάγκη για υπολογιστική ισχύ. Τέτοιοι τομείς είναι:

- Πρόγνωση καιρού
- Μόλυνση περιβάλλοντος
- Προσομοίωση
- Σχεδίαση αυτοκινήτων κλπ
- Πρόγνωση σεισμών-μελέτη τεκτονικών πλακών
- Αεροδιαστημική τεχνολογία
- Κβαντομηχανική
- Μοριακή χημεία - σχεδιασμός φαρμάκων
- Κοσμογονία
- Κρυπτογραφία και γενικά συνδυαστικά προβλήματα

Η ανάγκη αυτή οδήγησε στην δημιουργία υπολογιστικών συστημάτων και αλγορίθμων που επεκτείνουν ή και ξεφεύγουν από το μοντέλο του Von Neumann.

1.3.2 Κατανεμημένη Ισχύς

Πρόκειται για τα **παράλληλα/κατανεμημένα συστήματα.** Μέσα σε δεκαετίες εξέλιξης έχουν δημιουργηθεί, εξελιχθεί (και σε πολλές περιπτώσεις περάσει στην ιστορία) διάφοροι τύποι συστημάτων που καλύπτονται κάτω από γενικούς όρους όπως **Parallel/Distributed Processing, Supercomputing, High-Performance Computing (HPC)**. Είναι ένα διεπιστημονικό πεδίο έρευνας που ασχολείται με όλα τα θέματα που αφορούν στην επίλυση μεγάλης κλίμακας επιστημονικών προβλημάτων τα οποία απαιτούν σημαντική υπολογιστική ισχύ [Marksteiner, 1996].

Μοντέλο Παράλληλης Επεξεργασίας (Parallel Model of Computation) είναι η αφηρημένη περιγραφή ενός παράλληλου υπολογιστή, ενώ **Παράλληλος Αλγόριθμος** είναι μία μέθοδος επίλυσης ενός προβλήματος σε ένα μοντέλο παράλληλης επεξεργασίας, εκτελώντας δηλαδή περισσότερες από μία λειτουργίες κάθε χρονική στιγμή [Akl, 2000].

Εδώ κύριος στόχος είναι η επιτάχυνση της επίλυσης ενός προβλήματος με εκμετάλλευση της έννοιας του **παραλληλισμού** (parallelism). Η εκμετάλλευση του παραλληλισμού μπορεί να γίνει σε διάφορα επίπεδα που ξεκινάνε από το Υλικό (αρχιτεκτονικές) και φθάνουν σε Αλγορίθμους και Λογισμικό. Μπορούμε να περιγράψουμε ένα σύστημα παράλληλης επεξεργασίας με την παρακάτω σχέση:

Σύστημα Παράλληλης Επεξεργασίας = Περιβάλλον + Πρόγραμμα

με το Περιβάλλον και το Πρόγραμμα να χαρακτηρίζονται από τις παρακάτω παραμέτρους:

Περιβάλλον	Πρόγραμμα
Επεξεργαστής (κόμβος) Αρχιτεκτονική Μνήμης Σύνδεση μονάδων επεξεργασίας Λειτουργικό σύστημα	Μέθοδος (Γλώσσα) προγραμματισμού Αλγόριθμος

Τα κατανεμημένα συστήματα κατηγοριοποιούνται με βάση την αρχιτεκτονική και τη διαχείριση επεξεργαστών και μνήμης.

Από πλευράς επεξεργαστή σε χαμηλό επίπεδο υλικού διακρίνονται διάφορες κατηγορίες αυτών [Marksteiner, 1996] [Xavier, Iyengar, 1998] [Litaize, Mzoughi, Rochange, Sainrat, 2000] ενώ σε υψηλότερο επίπεδο αφαίρεσης δεσπόζει η παλαιά αλλά ευρέως χρησιμοποιούμενη **Ταξινόμηση κατά Flynn** (Flynn's Taxonomy) που διαχωρίζει σε κατηγορίες ανάλογα με τον **αριθμό εντολών** και των **δεδομένων** που εκτελούνται και επεξεργάζονται ταυτόχρονα (παράλληλα) [Xavier, Iyengar, 1998]:

1. **S**ingle **I**nstruction **S**ingle **D**ata (**SISD**) που είναι ο κλασικός σειριακός υπολογιστής Von Neumann.

2. **S**ingle **I**nstruction **M**ultiple **D**ata (**SIMD**) όπου η ίδια εντολή εφαρμόζεται από πολλούς επεξεργαστές πάνω σε διαφορετικά για τον καθένα σύνολα δεδομένων.

3. **M**ultiple **I**nstruction **S**ingle **D**ata (**MISD**) ένα θεωρητικό μοντέλο που δεν έχει υλοποιηθεί μέχρι σήμερα, όπου μια μηχανή κάνει διάφορες ενέργειες πάνω σε ένα σύνολο δεδομένων.

4. **M**ultiple **I**nstruction **M**ultiple **D**ata (**MIMD**) που αναφέρεται σε ένα σύστημα με πολλούς επεξεργαστές που εργάζονται ανεξάρτητα πάνω σε διαφορετικά σετ δεδομένων παράγοντας αποτελέσματα για ένα γενικότερο σύστημα.

Στην αρχιτεκτονική μνήμης και στην μέθοδο πρόσβασης σ' αυτήν [Garcia, Ferreira, Guedes, 2000] στηρίζεται η ταξινόμηση που προτάθηκε από την [Johnson, 1988]:

- **UMA (Uniform Memory Access)** Εδώ ο παραλληλισμός περιορίζεται στους επεξεργαστές οι οποίοι έχουν πρόσβαση σε μία κοινή μνήμη με ίδιες επιδόσεις.
- **NUMA (Non-Uniform Memory Access)** Ο κάθε επεξεργαστής έχει την δική του τοπική μνήμη, αλλά μέσω του δικτύου επικοινωνίας έχει πρόσβαση στην μνήμη των άλλων. Ο προγραμματιστής έχει σφαιρική εικόνα της μνήμης («κοινή»).
- **NORMA (No Remote Memory Access)** Οι επεξεργαστές δεν έχουν δυνατότητα πρόσβασης σε απομακρυσμένη μνήμη, παρά μόνον στην τοπική. Η επικοινωνία μεταξύ τους γίνεται αποκλειστικά με ανταλλαγή μηνυμάτων (message passing). Οι υλοποιήσεις αυτής της κατηγορίας είναι οι οικονομικότερες.

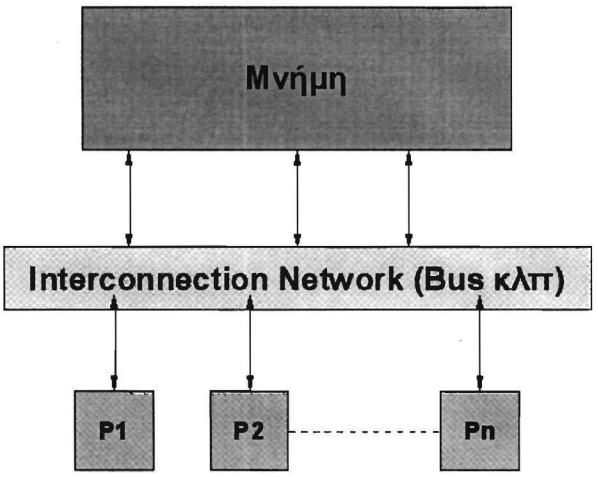
Έτσι, οι οικογένειες **Κοινής Μνήμης (Shared Memory)** και πλήρως **Κατανεμημένης Μνήμης (Distributed Memory)** βρίσκονται στα δύο άκρα. Στην πρώτη περίπτωση η μνήμη είναι κοινή για όλους τους επεξεργαστές (multiprocessors), ενώ στη δεύτερη είναι πλήρως κατανεμημένη τοπικά σε κάθε επεξεργαστή (multicomputers – distributed processing).

Από θεωρητικής πλευράς, το κλασικό μοντέλο **Κοινής Μνήμης** που χρησιμοποιείται για την ανάλυση της χειρότερης απόδοσης των παράλληλων αλγορίθμων, είναι το **PRAM** (Parallel Random Access Machine) [Miller R., Boxer, 2000], γενίκευση του RAM-model που αγνοεί την μέθοδο και το κόστος της επικοινωνίας. Κατά την **Εκτέλεση,** στις φάσεις της ανάγνωσης και εγγραφής από/προς τη μνήμη, υπάρχει η δυνατότητα προσπέλασης όλων των επεξεργαστών

στην ίδια διεύθυνση. Αυτό το χαρακτηριστικό δίνει τη δυνατότητα επικοινωνίας των επεξεργαστών (blackboard), αυξάνει όμως την πολυπλοκότητα του, λόγω των διενέξεων (conflicts) που δημιουργούνται όταν περισσότεροι του ενός επεξεργαστές προσπελαύνουν την ίδια θέση στη μνήμη και οι οποίες πρέπει να αντιμετωπιστούν.

Ένας αλγόριθμος PRAM λέγεται ότι είναι **ταυτόχρονης ανάγνωσης (concurrent-read)** όταν επιτρέπει σε περισσότερους του ενός επεξεργαστές να διαβάσουν ταυτόχρονα από την ίδια θέση μνήμης, αλλιώς λέγεται **αποκλειστικής ανάγνωσης (exclusive-read)**. Ανάλογα ορίζονται και οι έννοιες **concurrent-write** και **exclusive-write**. Έτσι, ανάλογα με τη **μορφή της πρόσβασης** που έχουν οι **επεξεργαστές** στη **μνήμη**, διακρίνονται οι υποκατηγορίες **EREW, CREW, ERCW, CRCW** [Cormen, Leiserson, Rivest, 2000].

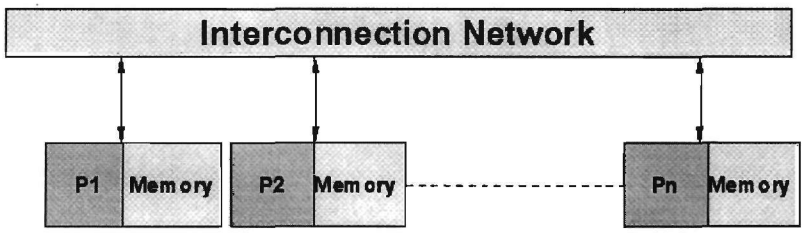
Εικόνα 1 – Μοντέλο Κοινής Μνήμης



Τα μοντέλα **Κατανεμημένης Μνήμης** [Akl, 2000] [Miller R., Boxer, 2000] αποτελούνται από n επεξεργαστές $P_1, P_2, \ldots P_n$ που ο καθένας ελέγχει τη δική του τοπική μνήμη. Η επικοινωνία γίνεται με **ανταλλαγή μηνυμάτων** (message passing)

μέσα από το **Δίκτυο Εσωτερικής Επικοινωνίας** (Interconnection Network) που είναι ειδικές συνδέσεις διπλής κατεύθυνσης. Σε κάποιες περιπτώσεις ορισμένα μόνον ζευγάρια επεξεργαστών (γείτονες) είναι άμεσα συνδεδεμένοι. Σε περίπτωση που δεν είναι άμεσα συνδεδεμένοι τα μηνύματα ακολουθούν κάποια διαδρομή περνώντας από άλλους επεξεργαστές μέχρι να φτάσουν στον προορισμό τους.

Εικόνα 2 – Μοντέλο Κατανεμημένης Μνήμης



Οι μηχανές **Κατανεμημένης Κοινής Μνήμης** (Distributed Shared Memory, DSM) είναι συστήματα φυσικά κατανεμημένης μνήμης που όμως είναι κοινή σε λογικό επίπεδο. Υπάρχουν υλοποιήσεις του DSM περιβάλλοντος σε διάφορα επίπεδα [Garcia, Ferreira, Guedes, 2000], ενώ αναφέρονται προσπάθειες για ομογενοποίηση και αξιοποίηση των θετικών στοιχείων των μοντέλων φυσικής, λογικής κοινής μνήμης και μηχανισμού ανταλλαγής μηνυμάτων [Hill, Larus, Wood, 1996] [Nieplocha, Harrison, Littlefield, 1996].

Η άποψη της κοινής μνήμης στα DSM συστήματα κάνει ευκολότερο τον προγραμματισμό της μηχανής, από αλγοριθμικής όμως απόψεως η συμπεριφορά τους είναι παρόμοια με αυτή των μηχανών κατανεμημένης μνήμης [Miller R., Boxer, 2000], ενώ μεταφέρεται δύσκολα σε διαφορετικά περιβάλλοντα, έχει μικρές δυνατότητες ελέγχου στις επικοινωνίες και μεταφορές δεδομένων μεταξύ των επεξεργαστών και βέβαια η πρόσβαση σε τοπική μνήμη είναι πολύ ταχύτερη από την πρόσβαση σε κάποια μακρινή μνήμη ενός άλλου επεξεργαστή.

1.3.3 Λογισμικό Κατανεμημένων Συστημάτων

Η διαχείριση των κατανεμημένων συστημάτων και η μετάδοση της πληροφορίας γίνεται σε διάφορα επίπεδα, με ειδικό λογισμικό που μπορεί να είναι:

- **Λειτουργικό Σύστημα**

Ένα **Κατανεμημένο Λειτουργικό Σύστημα** αποτελεί τη διασύνδεση ανάμεσα σε χρήστη και μηχανή ώστε διαχειριζόμενο κεντρικά τους κατανεμημένους σε φυσικό επίπεδο πόρους να φαίνεται στον χρήστη σαν μια ενιαία μηχανή [Garcia, Ferreira, Guedes, 2000] [Freisleben, Kielmann, 1995].

Σε ερευνητικό επίπεδο την προηγούμενη δεκαετία, έχουν δημιουργηθεί κατανεμημένα Λ.Σ., χαρακτηριστικό παράδειγμα το Amoeba [Tanenbaum, Renesse, Staveren +, 1990] τύπου DSM, όπου κάθε πόρος χαρακτηρίζεται λογικά από το σύστημα, άσχετα από την φυσική του θέση. Άλλα παλαιότερα ερευνητικά προγράμματα [Douglis, Kaashoek +, 1991] ήταν τα System V, Chorus, Locus, Mach, Plan 9, Sprite που υποστήριζε **μετανάστευση διαδικασιών** (process migration).

Σήμερα, με δεδομένη την στροφή προς τα χαλαρά συνδεδεμένα κατανεμημένα συστήματα, αυτά συνήθως τρέχουν τοπικά κάποια παραλλαγή ή κάποιο υποσύνολο – πυρήνα του UNIX ή Microsoft Windows. Ο παραλληλισμός υποστηρίζεται με **πρόσθετο Λογισμικό** που τρέχει πάνω από το τοπικό Λειτουργικό Σύστημα (Κατανεμημένο σύστημα αρχείων, υποστήριξη κατανεμημένης κοινής μνήμης – DSM, βιβλιοθήκες που υποστηρίζουν παράλληλες γλώσσες προγραμματισμού, διαχείριση πόρων κλπ).

Συστήματα όπως τα Beowulf, Condor, GLUnix του NOW, Castle [Freisleben, Kielmann, 1995] έχουν ιδιότητες που αποτελούν υποσύνολο ενός λειτουργικού συστήματος, ενώ το PODOS του Πανεπιστημίου του Mississippi [Vazhkudai, Syed, Maginnis, 2002] που τρέχει σαν κέλυφος πάνω από UNIX είναι μία γενικότερη προσέγγιση δημιουργίας ενός περιβάλλοντος διαμοιρασμού πόρων με άποψη από μεριάς χρήστη ενός ενιαίου συνόλου, δίνοντας όμως βάρος και στην απόδοση.

Το HARNESS [Dongarra, Geist, Kohl +, 1998] [Beck, Dongarra, +, 1999] έχει σαν αρχή την έννοια της Εικονικής Κατανεμημένης Μηχανής (Distributed Virtual Machine – DVM), για να παρέχει υπηρεσίες στις εφαρμογές παρόμοιες με έναν παράλληλο υπολογιστή κατανεμημένης μνήμης ενώ προορίζεται για μικρής σχετικά έκτασης κατανεμημένα ετερογενή περιβάλλοντα.

Σε επίπεδο WAN, το **Legion** [Grimshaw, Wulf, 1996] [Natrajan, Humphrey, Grimshaw, 2001] δίνει στον χρήστη την εντύπωση μίας απλής εικονικής μηχανής που στην πράξη αποτελείται από ετερογενείς, γεωγραφικά απομακρυσμένους πόρους ενώ κάθε αντικείμενο (μηχανή, χρήστης, κατάλογος, εφαρμογή κλπ) χαρακτηρίζεται μοναδικά από το Σύστημα.

Ενώ το Legion παρέχει ένα ομοιόμορφο, αντικειμενοστραφές προγραμματιστικό μοντέλο, το **Globus** [Foster, Kesselman, 1999] ακολουθεί την προσέγγιση της παροχής μίας σειράς **Υπηρεσιών** (Services) που χρησιμοποιούν οι εφαρμογές ανάλογα με τις ανάγκες τους. Για παράδειγμα, η διαχείριση των πόρων γίνεται από τα υποσυστήματα GRAM που τρέχουν σε κάθε site που μετέχει στο Globus system και συνεργάζονται με τα τοπικά αντίστοιχα συστήματα. Η εφαρμογή

εκφράζει απαιτήσεις σε πόρους μέσα από APIs χωρίς να επηρεάζει την πολιτική διάθεσης πόρων κάθε Site, το υποσύστημα βρίσκει τους απαιτούμενους πόρους και αναθέτει τις εργασίες.

Το Legion όπως και το Globus αποτελούν υλοποιήσεις της ιδέας του **Computational Grid** (Metasystems, Metacomputing) [Foster, Kesselman, 1998]. Ένα τέτοιο περιβάλλον ξεφεύγει από την απλή σύνδεση υπολογιστών. Μέρη του μπορεί να είναι και άλλα μη-παραδοσιακά περιφερειακά όπως συσκευές τηλεόρασης, μικροσκόπια, τηλεσκόπια κλπ. Οι Υπηρεσίες του Globus μπορούν να χρησιμοποιηθούν για να υλοποιήσουν το μοντέλο Legion και τα δύο προγράμματα θεωρούνται συμπληρωματικά.

Στην Ευρώπη το **DAS** (Distributed ASCI Supercomputer, όπου ASCI = Advanced School for Computing and Imaging, στην Ολλανδία) [Bal, Bhoedjang +, 2000] αποτελείτο από τέσσερις γεωγραφικά κατανεμημένες πλήρως ομογενείς συστοιχίες μικροϋπολογιστών και χρησιμοποιείται για διάφορα ερευνητικά προγράμματα [Bal, Plaat +, 1999] [Iskra, Belleman, Albada +, 2001]. Χρησιμοποιεί ιεραρχική δομή των Συστοιχιών προκειμένου αλγοριθμικά να επιτύχει μείωση των καθυστερήσεων που οφείλονται στην επικοινωνία.

Παρόμοιας λογικής με το Legion είναι το **Millennium system** [Bolosky, Draves +, 1997] της Microsoft, ενώ το **SNIPE** [Fagg, Moore, Dongarra, 1999] προσφέρει ένα ολοκληρωμένο περιβάλλον για κατανεμημένη υπολογιστική ισχύ, συλλογή και αποθήκευση δεδομένων, διαχείριση πόρων και διεπαφή με χρήστη.

❑ Παράλληλη Γλώσσα

Οι παράλληλες γλώσσες προγραμματισμού αποτελούν περιβάλλοντα ανάπτυξης εφαρμογών για παράλληλα / κατανεμημένα συστήματα υπολογιστών, περιέχοντας ειδικές λειτουργίες επεξεργασίας δεδομένων, καθώς και υποστήριξη επικοινωνίας και συντονισμού ανάμεσα στις εργασίες της εφαρμογής.

Οι παράλληλες γλώσσες προγραμματισμού μπορούν να κατηγοριοποιηθούν με βάση τους έξι παρακάτω άξονες [Foster, 2000]:

- Επεκτάσεις σε υπάρχουσες σειριακές – ανάπτυξη νέων γλωσσών.
- Παραλληλισμός σε επίπεδο υποπρογράμματος (task parallelism) – παραλληλισμός σε επίπεδο δεδομένων (data parallelism).
- Ρητός παραλληλισμός (explicit parallelism) – κρυφός παραλληλισμός (implicit parallelism).
- Ντετερμινιστική - μη ντετερμινιστική εκτέλεση (Determinism – nondeterminism).
- Πλήρης γλώσσα προγ/μού – Γλώσσα συντονισμού (coordination language).
- Γλώσσα προσανατολισμένη σε συγκεκριμένη αρχιτεκτονική – γλώσσα ανεξάρτητη της αρχιτεκτονικής.

Οι [Skillicorn, Talia, 1998] ακολουθούν μία περισσότερο δομημένη ταξινόμηση των γλωσσών προγραμματισμού σε δύο επίπεδα με βάση τα κριτήρια:

Α. Τον **βαθμό αφαίρεσης** που έχει η γλώσσα, δηλαδή αν είναι υψηλού ή χαμηλού επιπέδου ως προς τις αρμοδιότητες του προγραμματιστή σε σχέση με αυτές του συστήματος μεταγλώττισης / εκτέλεσης όπως φαίνεται στον Πίνακα 3.

ΑΒ. Τον **βαθμό ελέγχου** που έχει η γλώσσα στη **δομή** της εφαρμογής (αριθμός threads ή processes) και την **επικοινωνία**.

Πίνακας 3 - Κατηγορίες Γλωσσών Α' Επίπεδο

Αρμοδιότητες Προγραμματιστή	**Κατηγορία Γλώσσας**					
	A_1	**A_2**	**A_3**	**A_4**	**A_5**	**A_6**
Αναφορά Παραλληλισμού		**X**	**X**	**X**	**X**	**X**
Διάσπαση του προγράμματος σε νήματα (threads)			**X**	**X**	**X**	**X**
Αντιστοίχηση threads και επεξεργαστών				**X**	**X**	**X**
Επικοινωνία των threads					**X**	**X**
Συγχρονισμός						**X**

Μέσα σε κάθε μια από τις παραπάνω κατηγορίες $\mathbf{A_i}$ διακρίνονται οι ακόλουθες υποκατηγορίες:

A_iB_1. Μοντέλα που η δομή των threads είναι δυναμικά μεταβαλλόμενη. Σε αυτά η επικοινωνία δεν μπορεί να έχει περιορισμούς διότι κάθε νέο thread που δημιουργείται έχει και απαιτήσεις επικοινωνίας.

A_iB_2. Μοντέλα που τα threads είναι στατικής δομής αλλά η επικοινωνία δεν έχει περιορισμούς.

A_iB_3. Μοντέλα όπου η δομή των threads είναι στατική και η επικοινωνία περιορισμένη.

Η **Fortran M** [Foster, Chandy, 1995] είναι μία task-parallel επέκταση της Fortran 77 με δομές για τον καθορισμό τμημάτων προγράμματος που θα εκτελεστούν ταυτόχρονα, για την δυναμική δημιουργία καναλιών επικοινωνίας και την αποστολή και λήψη μηνυμάτων μέσα από τα κανάλια.

Η **F90** (Fortran 90) περιέχει στοιχεία παραλληλισμού στις εντολές επεξεργασίας πινάκων που διαθέτει χωρίς να έχει δομές ελέγχου της κατανομής των δεδομένων.

Η **HPF** (High Performance Fortran) [Loveman, 1993] έχει αναπτυχθεί από το HPF-Forum. Πρόκειται για data-parallel επέκταση της F90, που έχει δομές κατανομής δεδομένων σε επεξεργαστές. Ο compiler με βάση τις οδηγίες που δίνει ο προγραμματιστής δημιουργεί τον κώδικα επικοινωνίας. Υπάρχει τάση ενοποίησης των FM και HPF [Foster, Kohr +, 1996].

Η **HPC++** (High Performance C++) [HPC++, 1995] είναι αποτέλεσμα έρευνας στην Αμερική από ομάδα εργασίας σε αναλογία με το MPI-Forum. Υποστηρίζει τόσο το data όσο και το task parallel μοντέλο, ενώ μπορεί να επικοινωνεί με Java εφαρμογές. Διαφορετικές προσεγγίσεις [Gannon, Diwan, Johnson, 1996] ακολουθούνται στην Ευρώπη (**Europa parallel C++**) και στην Ιαπωνία (**MPC++**).

Η **CC++** (Compositional C++) [Chandy, Kesselman, 1993] [Foster, 2000] αποτελεί ένα αυστηρά υπερσύνολο της C++. Ο παραλληλισμός υλοποιείται ορίζοντας παράλληλα τμήματα κώδικα (blocks), παράλληλες επαναληπτικές διαδικασίες (loops), αλλά και μη δομημένος σε επίπεδο νημάτων (threads) ενώ διαχωρίζει τοπικά από απομακρυσμένα αντικείμενα. Τα νήματα επικοινωνούν μεταβάλλοντας κοινές (μοιραζόμενες) δομές π.χ. μία λίστα, ενώ για συγχρονισμό έχει μεταβλητές τύπου sync και συναρτήσεις τύπου atomic.

Η **NESL** [Blelloch, 1995] είναι μία συμπαγής γλώσσα ανωτέρου επιπέδου (δημιουργήθηκε μέσα από το ScandAL project του Carnegie Mellon University) με στόχο να κάνει την συγγραφή παράλληλου κώδικα παρόμοια με αυτήν του σειριακού. Υποστηρίζει φωλιασμένο παραλληλισμό πάνω στα δεδομένα (nested data parallelism) ενώ κρύβει την πολυπλοκότητα αφήνοντας στον Compiler και το σύστημα εκτέλεσης την ευθύνη της παράλληλης αποδοτικής εκτέλεσης.

Η **Linda** [Bjornson, Carriero +, 1988] είναι γλώσσα συντονισμού (coordination language) που συνδυάζεται με άλλες παραδοσιακές γλώσσες (C-Linda, Fortran-Linda). Βασίζεται σε ένα λογικό αντικείμενο κοινής μνήμης που ονομάζεται tuple space (Virtual Shared Memory) μέσα από το οποίο οι επεξεργασίες επικοινωνούν και συγχρονίζονται. Το **Piranha-Linda** [Freisleben, Kielmann, 1995]

είναι ένα από τα πρώτα συστήματα που εκμεταλλεύεται τους νεκρούς χρόνους των CPU σε δίκτυα υπολογιστών για να εκτελέσει εφαρμογές που περιμένουν. Οι αρχές του tuple space χρησιμοποιούνται σήμερα σε νέα περιβάλλοντα όπως είναι το **Jini/Javaspaces** [Kumaran, 2001] και **TSpaces** [TSpaces, 1998].

Η **Ada** [RM, 1995] του Υπουργείου Εθνικής Άμυνας των Η.Π.Α. αντιπροσωπεύει μαζί με την Concurrent C τις γλώσσες μοντέλου κατανεμημένης μνήμης που εμπεριέχουν την έννοια του Λογισμικού ραντεβού (Rendezvous based models) προκειμένου να επιτύχει την επικοινωνία μεταξύ των μονάδων (units) μίας εφαρμογής.

Η **Java** [De Pietro, 2000] είναι αντικειμενοστραφής γλώσσα πλήρως φορητή σε κάθε υπολογιστικό περιβάλλον λόγω του ενδιάμεσου κώδικα (bytecode) που παράγει. Ο κώδικας εκτελείται μέσω της Java Virtual Machine που είναι real-time διερμηνευτής. Η τεχνική του Just-In-Time Compiler (JIT) προσπαθεί να μειώσει τις καθυστερήσεις που οφείλονται στην έλλειψη κλασικού compiler και βρίσκεται στη φάση της εξέλιξης [Suganuma, Ogasawara +, 2000].

Είναι ήδη γνωστή η τεράστια εφαρμογή των Java Applets στο WWW, και βέβαια προφανείς οι δυνατότητες που ανοίγονται για εκμετάλλευση του Διαδικτύου σαν ένα καθολικό σύστημα Client/Server. Παρόλο που η Java απέχει ακόμα από το να είναι γλώσσα υψηλής απόδοσης, σε αυτό τον τομέα διεξάγεται εκτεταμένη έρευνα. Παράδειγμα, η Manta του ASCI DAS supercomputer, το Javelin [Neary, Christiansen +, 1999] μέρη του IceT project, [Gray P., Sunderam, 1999] το ATLAS, [Baldeschwieler, Blumofe +, 1996] το Popcorn, [Camiel, London +, 1997] το Bayanihan [Sarmenta, 1998], το Ninflet [Takagi, Matsuoka +, 1998].

- **APIs**

Πάνω από τα πρωτόκολλα των δικτύων συνήθως τρέχει κάποιο λογισμικό **σύστημα μετάδοσης μηνυμάτων (message passing system)**. Σήμερα, δύο είναι τα κύρια περιβάλλοντα APIs που έχουν επικρατήσει μετά από δεκαετίες εξέλιξης:

Το **PVM (Parallel Virtual Machine)** [Sunderam, 1990] [Geist, Beguelin, Dongarra, +, 1994] επιτρέπει σε ένα δυναμικά μεταβαλλόμενο, ετερογενές δίκτυο από υπολογιστές να αντιμετωπίζεται σαν μία υπολογιστική μηχανή κατανεμημένης μνήμης. Κάθε εργασία παίρνει από το σύστημα έναν ακέραιο (task identifier, TID) που είναι μοναδικός και αποτελεί την ταυτότητά της.

Το **MPI (Message Passing Interface)** [MPI-forum, 1994] [MPI-forum, 1995] [MPI-forum, 1997] αναπτύχθηκε από το Mpi-Forum, μία ομάδα από ερευνητές, χρήστες, επιστημονικά και κυβερνητικά εργαστήρια και πανεπιστήμια και αποτελεί την τυποποίηση (standard) ανταλλαγής μηνυμάτων για την δημιουργία παράλληλων εφαρμογών. Δεν είναι συγκεκριμένο Λογισμικό, ωστόσο υπάρχουν αρκετές υλοποιήσεις του όπως CHIMP [Alasdair, Bruce +, 1994], MPICH [Gropp, Lusk, 1996], LAM [Burns, Daoud, 1995], OOMPI [Squyres, McCandless, Lumsdaine, 1996], UNIFY [Vaughan, Skjellum, Reese, Cheng, 1995], FT-MPI [Fagg, Bukovsy, Dongarra, 2001], MPI για Grid περιβάλλοντα [Foster, Geisler, Gropp +, 1998].

Το MPI στηρίχτηκε σε άλλα συστήματα μετάδοσης μηνυμάτων που προϋπήρχαν, όπως τα p4 [Butler, Lusk, 1994], TCGMSG [Harrison, 1991], PARMACS [Hempel, Steppe, Supakov, 1993] που χρησιμοποιείται ιδιαίτερα στην Ευρώπη [Schattler, Krenzien, 1997], Chameleon [Gropp, Smith, 1993], Zipcode [Skjellum, Smith, Doss +, 1994] αλλά και PVM.

Η σύγκριση ανάμεσα στα PVM και MPI είναι δύσκολη, με συχνά αντίθετες απόψεις [Geist, Kohl, Papadopoulos, 1996] [Squyres, 1997] [Gropp, Lusk, 1997] [Geist, Kohl, Papadopoulos, Scott, 1997] ενώ δεν λείπουν και οι προσπάθειες ενοποίησης των δύο συστημάτων είτε με μεταφορά χαρακτηριστικών του ενός στο άλλο, είτε ανεξάρτητα [Lehman, 1994] [Vaughan, Skjellum, Reese, Cheng, 1995] [Fagg, Dongarra, Geist, 1997].

Το **Nexus** [Foster, Kesselman, Tuecke, 1994] είναι ένα ειδικό σύστημα που συνδυάζει νήματα (threads) και επικοινωνία, ενώ υποστηρίζει ετερογενή συστήματα σε διάφορα επίπεδα. Σε κάθε κόμβο δημιουργούνται συνδέσεις ανάμεσα σε εκτελέσιμο κώδικα και τμήματα δεδομένων (Contexts) και μέσα σε κάθε context ένα ή περισσότερα νήματα εκτελούν την υπολογιστική διαδικασία. Τα νήματα μπορούν να δημιουργηθούν είτε μέσα στο ίδιο context ή σε άλλο μέσω διαδικασίας που λέγεται **Αίτηση Απομακρυσμένης Υπηρεσίας** (Remote Service Request) και χρησιμοποιεί **Καθολικούς Δείκτες** (Global Pointers).

Η **Remote Service Request** [Foster, Kesselman, Tuecke, 1996] που είναι παρόμοια με τα **Active Messages** [Eicken, Culler, +, 1992] [Culler, Arpaci-Dusseau, Chun +, 1997] έχει σαν αποτέλεσμα την εκτέλεση μίας ειδικής συνάρτησης (handler) στο context που υποδεικνύεται από τον Καθολικό Δείκτη. Διαφέρει από την **Remote Procedure Request** (π.χ. CC++) [Foster, 2000] στο ότι δεν υπάρχει ανταλλαγή μηνυμάτων που να γνωρίζουν την λήψη της αίτησης ούτε επιστρεφόμενες τιμές, ενώ το νήμα που την κάνει δεν μπλοκάρει αναμένοντας απάντηση, αλλά συνεχίζει την εκτέλεση.

1.3.4 Συστήματα Κατανεμημένης Μνήμης

Οι μικροϋπολογιστές αποτελούν μηχανές κατανεμημένης μνήμης (NORMA) οι οποίες γενικότερα περιλαμβάνουν:

- Ειδικής αρχιτεκτονικής Μαζικά Παράλληλοι Επεξεργαστές (Massively Parallel Processors, MPP).
- Συστοιχίες από ομοιογενείς υπολογιστές (Workstation Farms, Clusters of Workstations - COW, Networks of Workstations – NOW, Pile of PCs – PoPCs).
- Δίκτυα ετερογενή, αποτελούμενα από διαφόρων ειδών υπολογιστές (Supercomputers, Workstations, PCs).

Στην πρώτη περίπτωση των MPPs έχουμε να κάνουμε με ομογενείς μηχανές, που περιέχουν μέχρι χιλιάδες επεξεργαστών ο καθένας με τοπική μνήμη, ενώ η επικοινωνία γίνεται με ανταλλαγή μηνυμάτων μέσω ειδικού για την κάθε μηχανή δικτύου εσωτερικής επικοινωνίας, π.χ. Intel NX message passing system [Pierce, Regnier, 1994].

Ωστόσο, είναι το **κόστος** αυτό που κατά κύριο λόγο προσδιορίζει την εξέλιξη [Anderson, Culler, Patterson, 1995]. Το 1995 η σχέση τιμής-απόδοσης στους Workstations αυξανόταν 80% κάθε έτος, ενώ η αντίστοιχη τιμή για τους supercomputers ήταν της τάξης του 20-30%. Οι παράλληλοι (με την ευρεία έννοια του όρου) υπολογιστές, δίνουν μεγάλες υποσχέσεις, όπως κάνουν διαρκώς εδώ και 30 χρόνια [Hill, Larus, Wood, 1996] μέχρι το 1993 στη μορφή των στενά συνδεδεμένων, ειδικού σχεδιασμού, ομογενών ομάδων επεξεργαστών, σήμερα όμως στη μορφή **χαλαρά συνδεδεμένων** ή και κατανεμημένων (κατανεμημένης μνήμης) **συνόλων από κοινές μηχανές του εμπορίου** που μπορεί να είναι και ετερογενείς [Pancake, 2001]. Στις μηχανές αυτού του τύπου μεγάλες προκλήσεις είναι η απόδοση (performance) και η μεταφερσιμότητα σε διάφορες πλατφόρμες (portability).

Ο IBM RS/6000 ήταν ένα κλασικό παράδειγμα αυτής της τάσης ενώ το μεγαλύτερο ίσως πρόγραμμα δημιουργίας τέτοιων μηχανών είναι το υλοποιούμενο από την Αμερικανική Κυβέρνηση Accelerated Strategic Computing Initiative (DoE ASCI) [ASCI, 2001] που ξεκίνησε από την ανάγκη για εξέλιξη της πυρηνικής τεχνολογίας των Η.Π.Α. μέσω προσομοιώσεων.

Οι **Clusters of Workstations** έχουν σαν αρχή την παρατήρηση ότι οι περισσότεροι χρήστες χρειάζονται σχεδόν πάντοτε λιγότερη υπολογιστική ισχύ από αυτήν που τους παρέχει η μηχανή τους. Έτσι, οι μηχανές περνούν τον περισσότερο χρόνο παραμένοντας αδρανείς, οπότε θα μπορούσαν να χρησιμοποιηθούν από εφαρμογές που σε άλλες μηχανές περιμένουν να εκτελεστούν [Hayes, 1998].

Χαρακτηριστικές τέτοιες περιπτώσεις το **NOW project** του Berkeley [Anderson, Culler, Patterson, 1995] [Culler, Arpaci-Dusseau, Chun +, 1997], το **Condor Project** [Basney, Livny, 1999] [Livny, Basney, Raman +, 1997] στο Πανεπιστήμιο του Wisconsin με κεντρικό σύστημα που έκανε το ταίριασμα ανάμεσα στις απαιτήσεις των εφαρμογών και τους διατιθέμενους ελεύθερους πόρους, ενώ είχε τη δυνατότητα σύνδεσης διαφορετικών Condor machines σε επίπεδο WAN [Epema, Livny +, 1996].

Περιβάλλοντα όπως το Condor, αναφέρονται και σαν **High Throughput Computing (HTC)** όπου η μονάδα μέτρησης της απόδοσής τους είναι όχι τα FLOPS, αλλά η υπολογιστική ισχύς που αποδίδουν σε χρονικά διαστήματα εβδομάδας, μηνός ή και έτους.

Το **Beowulf project** [Ridge, Becker, 1997] ξεκίνησε από τη NASA με σκοπό την εκμετάλλευση της ραγδαία αυξανόμενης υπολογιστικής ισχύος των μικροεπεξεργαστών του εμπορίου σε συνδυασμό με το χαμηλό κόστος. Οι προδιαγραφές του χαρακτηρίζονται από

- Κοινό Hardware μαζικής παραγωγής του εμπορίου.
- Αφιερωμένους επεξεργαστές (σε αντίθεση με το NOW).
- Αφιερωμένο δίκτυο διασύνδεσης (System Area Network– SAN).
- Καμία χρήση ειδικών περιφερειακών.
- Ευκολία υλοποίησης από πολλαπλούς κατασκευαστές.
- Χρήση δωρεάν διατιθέμενου Software ανοικτής αρχιτεκτονικής.

Οι κόμβοι έχουν τοπικούς σκληρούς δίσκους για αποθήκευση δεδομένων (προσανατολισμός προς SPMD εφαρμογές), ανταλλάσσουν μηνύματα μέσω δικτύου. Δεν γίνονται διακρίσεις σε κόμβους υπολογισμού, I/O κλπ. Το περιβάλλον του συστήματος Grendel (έλεγχος πόρων, άλλα εργαλεία) στέκεται πάνω στο UNIX.

Μηχανές αυτού του τύπου που είναι γνωστές και σαν **Piles of PCs**, έχουν πειραματικά κατασκευαστεί για επιστημονικούς, ερευνητικούς και διδακτικούς σκοπούς σε πολλά Πανεπιστήμια και Ιδρύματα. Η απόδοσή τους σε σχέση με το κόστος είναι από τις καλύτερες γνωστές.

Παράδειγμα ο **ABC** (AFIT Bimodal Cluster) [Bohn, Lamont, 2002] όπου μεταξύ άλλων ερευνήθηκε η παράλληλη επεξεργασία σε περιβάλλον Windows NT.

Στο **PMS Project** [Csikor, Fodor +, 2001] του Πανεπιστημίου Eotvos της Βουδαπέστης οι επεξεργαστές συνδέονταν με ειδική κάρτα επικοινωνίας.

Στο **Galaxy** [Deng, Korobka, 2001] του State University of New York οι κόμβοι είναι διαφοροποιημένης λειτουργικότητας ενώ η κατανομή των πόρων στις εργασίες που τρέχουν ρυθμίζεται από το **Portable Batch System (PBS)** που έχει αναπτυχθεί για την NASA [PBS, 2000]. Πυρήνας του PBS είναι ο προγραμματιστής εργασιών (job scheduler) που σε συνεργασία με τις διαδικασίες ελέγχου εργασιών (job monitor processes) που τρέχουν σε κάθε κόμβο αναθέτει τις εργασίες μόλις βρει επαρκείς πόρους στο σύστημα.

Το **Cplant** [Riesen, Brightwell +, 1999] χωρίζει τους κόμβους που δεν έχουν τοπικούς χώρους αποθήκευσης σε λογικές διαμερίσεις (service, compute, I/O, graphics partition κλπ). Περισσότερα του ενός Cplant μπορούν να ενωθούν, ενώ αποτελεί μέρος ενός ευρύτερου Wide-Area Computing Network.

Όλα βέβαια τα παραπάνω συστήματα έχουν κεντρικό στόχο τους την επίλυση προβλημάτων, για λόγους και με τρόπους που περιγράφονται παρακάτω.

1.3.5 Παραλληλισμός & Επίλυση Προβλημάτων

Οι στόχοι είναι προφανείς. Αύξηση της **υπολογιστικής ισχύος** και της **μνήμης** που διατίθεται για την επεξεργασία. Το **κόστος** είναι κυρίως ο **χρόνος** που απαιτείται για την επικοινωνία ανάμεσα στους υπολογιστικούς κόμβους αλλά και η μεγαλύτερη πολυπλοκότητα από αλγοριθμικής πλευράς.

Ο παραλληλισμός **βέβαια**, δεν μπορεί να ξεπεράσει το γεγονός ότι τα προβλήματα είναι NP-Hard επομένως εκθετικού χρόνου, εκτός και αν χρησιμοποιηθεί εκθετικός αριθμός επεξεργαστών [Cormen, Leiserson, Rivest, 2000], ωστόσο η επιτάχυνση της επίλυσης που προσφέρει είναι επιθυμητή σε πολλές πραγματικές εφαρμογές [Correa, Ferreira, 1995]. Η ιδανική βέβαια περίπτωση προβλήματος για να υποστεί παράλληλη επεξεργασία είναι όταν αυτό απαιτεί μεγάλο όγκο υπολογισμών και μικρή συχνότητα ή / και όγκο επικοινωνίας [Hromkovic, 2001].

Πολλές από τις **τεχνικές** που περιγράφτηκαν στην παράγραφο 1.2.4 μπορούν να υλοποιηθούν σε παράλληλες εφαρμογές [Ananth, Kumar, Pardalos, 1993] [Correa, Ferreira, 1995] με κάποιες να ταιριάζουν περισσότερο (Local Search, Branch-and-Bound, Backtracking, Genetic Algorithms) και άλλες λιγότερο (Simulated Annealing) [Hromkovic, 2001], ενώ ιδιαίτερη έμφαση δίνεται στην παράλληλη υλοποίηση Ευρετικών Αλγορίθμων [Resende, Pardalos, Eksioglu, 2001] [Linderoth, Lee, Savelsbergh, 2001].

Επίσης, παλαιότερες εργασίες [Clearwater, Hogg +, 1992] [Hogg, Williams, 1993] [Hogg, Huberman, 1993] έχουν ήδη δείξει ότι μέθοδοι επίλυσης βασισμένες σε **κατανεμημένη επεξεργασία** (distributed processing), όπου οι εργασίες ανταλλάσσουν μηνύματα, δεν είναι μόνον ταχύτερες από τις κλασσικές μεθόδους (μόνο μία κεντρική εργασία - job), αλλά μπορούν και να οδηγήσουν σε λύση.

Ήδη αναφέρονται περιπτώσεις όπου χιλιάδες (μικρο)υπολογιστές συνδεδεμένοι μέσω του INTERNET χρησιμοποιήθηκαν για εργασία πάνω σε ένα συγκεκριμένο σκοπό [Sheldon, 1998], για παράδειγμα σπάσιμο των κωδίκων κρυπτογράφησης RSA και DES [RSA, 1999].

Το σύστημα DAS και μία παραλλαγή της Java έχει χρησιμοποιηθεί για επίλυση συνδυαστικών προβλημάτων όπως είναι το Traveling Salesman Problem [Nieuwpoort, Maassen, Bal +, 1999], ενώ το NEOS (Network-Enabled Optimization System) [Czyzyk, Mesnier, More, 1996] είναι ένα περιβάλλον για την επίλυση προβλημάτων βελτιστοποίησης όπου ο χρήστης απλά υποβάλλει το πρόβλημα (e-mail, TCP/IP, Web Site). Σε περιπτώσεις μεγάλου μεγέθους προβλημάτων το NEOS χρησιμοποιεί το Condor για να αντλήσει πρόσθετη υπολογιστική ισχύ [Ferris, Mesnier, More, 1998]. Παρόμοιας αρχής με το NEOS (διαμοιρασμός υπολογιστικών και άλλων πόρων μέσα από το Διαδίκτυο) είναι τα NetSolve και Ninf [Petitet, Casanova, Dongarra +, 2000].

1.4 Συμπεράσματα

Ήδη στην παράγραφο 1.2 αναλύθηκαν οι δυσκολίες που παρουσιάζουν αρκετά προβλήματα στην διαδικασία επίλυσης τους, με βασικό στοιχείο την πολυπλοκότητα αλλά και την αναγκαιότητα σε μέγεθος κύριας μνήμης που απαιτούν. Δυσκολίες που είναι απαραίτητο να υπερνικηθούν ή έστω να αμβλυνθούν. Και αυτό λόγω των τεραστίων εφαρμογών που έχουν τα εν λόγω προβλήματα σε ένα μεγάλο φάσμα τομέων της Επιστήμης. Παράλληλα στην παράγραφο 1.3 δόθηκαν οι γνωστοί σχεδιασμοί υπολογιστικών συστημάτων τα οποία έχουν όλα σαν κύριο στόχο την αρχιτεκτονική του κώδικα έτσι ώστε να επιλύεται ταχύτερα ένα πρόβλημα αυξημένης πολυπλοκότητας.

Ωστόσο στην δεκαετία του 2000 αυτό που έχει σημασία είναι η συνδυαστική της μεθόδου επίλυσης και του υπολογιστικού συστήματος στο οποίο θα εφαρμοστεί. Μπορεί έτσι να έχουμε μία κακού χρόνου επίλυση σε ένα άριστο υπολογιστικό σύστημα ή αντίθετα ένα «φθηνό» υπολογιστικό σύστημα όπου «πιέζεται» με προγραμματιστική σκληρότητα για αναγκαστική εφαρμογή ενός άριστου αλγορίθμου. Αλλά ακόμα και στην περίπτωση που και οι δύο διαστάσεις είναι ικανοποιητικές αν όχι άριστες, δεν σημαίνει αυτό αναγκαστικά πως θα έχουμε και άριστες επιδόσεις. Αυτό σημαίνει ότι είναι πιθανόν ένας μέτριος αλγόριθμος να καταξιωθεί σε ένα συγκεκριμένο υπολογιστικό σύστημα και αυτόματα να προβιβαστεί σε άριστο, δηλαδή να προσφέρει άριστες επιδόσεις ενώ στο ίδιο υπολογιστικό σύστημα ένας άριστος αλγόριθμος υψηλής απόδοσης να κατασπαταληθεί σε ενδοεπικοινωνιακούς χρόνους γραφειοκρατίας του συντονισμού των μερών του υπολογιστικού συστήματος.

Είναι λοιπόν φανερό ότι αυτό που μετρά σαν **στρατηγικός σχεδιασμός επίλυσης προβλήματος** είναι η επιλογή αλγορίθμου και υπολογιστικού συστήματος, έτσι ώστε να συγκροτούν ένα ολόνιο ικανοποιητικών αποδόσεων.

Στην συνέχεια της παρούσης διατριβής ο στρατηγικός σχεδιασμός είναι η δημιουργία μεθόδου επίλυσης ικανής σε συνεργασία, με συνηθισμένο φθηνό μικροϋπολογιστικό υλικό (απλά PCs) με έναν οικονομικό τρόπο δικτύωσης μεταξύ τους, χωρίς ιδιαίτερο ειδικό λογισμικό ενδοσυνεννόησης. Ο στρατηγικός αυτός σχεδιασμός είναι ασφαλώς δύσκολο να γίνει. Είναι όμως αναγκαίος, διότι σχεδόν όλα τα υπολογιστικά συστήματα που αναλύθηκαν είναι **έξτρα κόστους** και διαφέρουν από αυτά που κάθε μέρα χρησιμοποιούνται στην Αγορά όσο το μοντέλο αυτοκινήτου που τρέχει σε αγώνες με το ίδιο μοντέλο που απλώς κινείται στους δρόμους μιας πόλης.

Γι' αυτό λοιπόν στην συνέχεια το υπολογιστικό σύστημα που θα αναπτυχθεί θα είναι απαλλαγμένο από κάθε πρόσθετο κόστος και θα μπορεί να συγκροτείται από οποιουσδήποτε διαθέσιμους υπολογιστές χωρίς αυτό να επηρεάζει αποφασιστικά την συνισταμένη επίδοση του όλου συστήματος.

Κεφάλαιο 2

ΚΑΤΑΝΕΜΗΜΕΝΟ ΣΥΣΤΗΜΑ ΕΝΙΑΥΣΗΣ

ΚΑΤΑΝΕΜΗΜΕΝΟ ΣΥΣΤΗΜΑ ΕΠΙΛΥΣΗΣ

"In pioneer days, they used oxen for heavy pulling, and when one ox couldn't budge a log they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers." Grace Hopper

2.1 Περιγραφή Προδιαγραφών

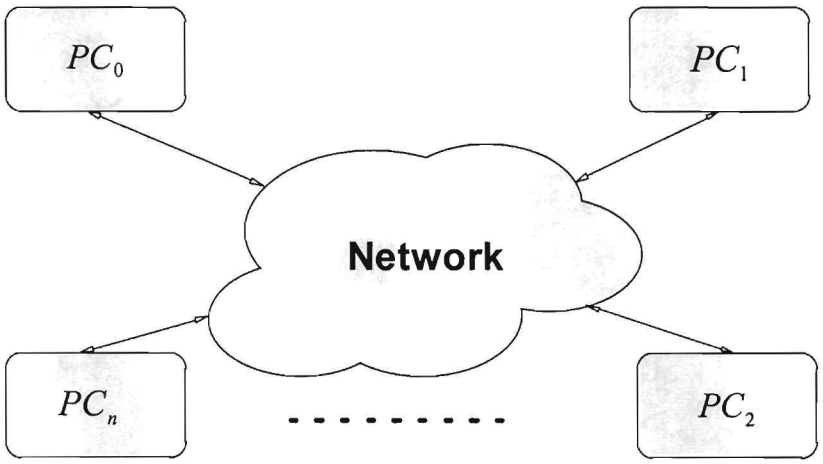
Προκειμένου να λειτουργήσει το προτεινόμενο **Κ**ατανεμημένο **Σ**ύστημα **Ε**πίλυσης (**Κ.Σ.Ε.**), απαιτείται από πλευράς Υλικού να υπάρχει διαθέσιμος ένας αριθμός μικροϋπολογιστών συνδεδεμένων μεταξύ τους μέσω δικτύου (τοπικού, LAN ή ακόμα και του Διαδικτύου, Internet), μοντέλο που σήμερα συναντάται κατά κανόνα (Εικόνα 3). Θεωρούμε $PCS = \{PC_0, PC_1, ..., PC_n\}$ το σύνολο των διαθέσιμων υπολογιστών με $n \geq 1$.

Με την εκκίνηση του Κατανεμημένου Συστήματος Επίλυσης υποθέτουμε αναγκαστικά ότι υπάρχει μία μηχανή έστω η PC_n, η οποία θα αναλάβει το ρόλο του **Διαχειριστή Προβλήματος (Problem Manager, PM)**, όπου δηλαδή, θα φροντίζει για το συντονισμό των υπολοίπων n μηχανών $PC_0, PC_1, ..., PC_{n-1}$, καθώς και το διαμοιρασμό των υποπροβλημάτων στις μηχανές αυτές, που θα ονομάζονται στο εξής **Επιλυτές Προβλήματος (Problem Solvers, PS)**. Οι γενικές προδιαγραφές του Συστήματος που προτείνεται έχουν ως εξής:

- Κάθε υπολογιστής διαθέτει τουλάχιστον έναν επεξεργαστή, τοπική μνήμη και προαιρετικά σκληρό δίσκο.
- Κάθε υπολογιστής PC_i, $i = 0,1,...,n$ του συνόλου PCS μπορεί να επικοινωνεί με όλους τους άλλους. Η τοπολογία και ο τύπος του δικτύου δεν ενδιαφέρει άμεσα, μολονότι είναι προφανές ότι αυτή θα επηρεάζει την ταχύτητα επικοινωνίας των μικροϋπολογιστών άρα και τον χρόνο επίλυσης.

- Όλοι οι υπολογιστές του PCS τρέχουν κάποιο απλό, κοινό Λειτουργικό Σύστημα όπως είναι το Unix ή τα Windows.
- Κανένας επεξεργαστής δεν έχει πρόσβαση σε μνήμη άλλου (NORMA system).
- Δεν χρησιμοποιούν κανένα ειδικό λογισμικό DSM ή Message Passing.
- Οι Επιλυτές λειτουργούν στο μοντέλο SPMD (Single Process Multiple Data).
- Οι υπολογιστές χρησιμοποιούν έναν Data Server (που μπορεί και να ανήκει στο PCS) μέσω του οποίου επικοινωνούν.
- Το Λογισμικό του Συστήματος είναι γραμμένο σε κάποια απλή Γλώσσα Προγραμματισμού, π.χ. C++, Delphi κ.ά.

Εικόνα 3 - Φυσική μορφή Κ.Σ.Ε.



Όπως κάθε σύστημα παράλληλης επεξεργασίας, έτσι και το προτεινόμενο Κ.Σ.Ε. αποτελείται όπως αναφέρθηκε ήδη στην παράγραφο 1.3.2 από το Περιβάλλον (Επεξεργαστή, Μνήμη, Σύνδεση μονάδων, Λειτουργικό σύστημα) και το Πρόγραμμα (Γλώσσα προγραμματισμού, Αλγόριθμοι). Στην παρούσα Διατριβή, οι απαιτήσεις σε όλες τις παραπάνω παραμέτρους είναι όσο το δυνατόν μικρότερες, με το βάρος να πέφτει στην παράμετρο των Αλγορίθμων, κυρίως Διάσπασης, Διοίκησης, και Επικοινωνίας.

2.2 Διάσπαση Προβλήματος

Έστω ένα πρόβλημα Π που αναλύεται ακριβώς σε ένα σύνολο υποπροβλημάτων

$$\Pi \overset{ορσ}{\equiv} \Pi_1^0 = \{\Pi_1^1, \Pi_2^1, ..., \Pi_k^1, ..., \Pi_\theta^1\}$$

με την ιδιότητα ότι η **ακριβής ανάλυση** κάθε υποπροβλήματος περιέχει αναγκαστικά τη λύση αυτού. Σταδιακά κάθε ένα υποπρόβλημα επίσης αναλύεται ακριβώς σε σύνολα υποπροβλημάτων της μορφής:

$$\Pi_k^1 = \{\Pi_1^2, \Pi_2^2, ..., \Pi_t^2, ..., \Pi_r^2\}$$

Γενικά, για κάθε υποπρόβλημα – πρόγονο θεωρούμε το σύνολο των υποπροβλημάτων – απογόνων του που θα είναι της παρακάτω μορφής:

$$\Pi_m^{w-1} = \{\Pi_1^w, \Pi_2^w, ..., \Pi_p^w, ..., \Pi_n^w\}$$

Ορισμός 2.2.1 Κάθε υποπρόβλημα Π_p^w που έχει προκύψει σαν απόγονος του Π_m^{w-1} με την ιδιότητα της ακριβούς ανάλυσης καλείται **υποπρόβλημα τάξης w** του αρχικού προβλήματος Π.

Ειδικά τα υποπροβλήματα 1ης τάξης

$$\Pi_1^1, \Pi_2^1, ..., \Pi_k^1, ..., \Pi_\theta^1$$

ονομάζονται **ρίζες (roots)** του αρχικού προβλήματος.

Με βάση τα παραπάνω, ένα αρχικό πρόβλημα θεωρείται τάξεως 0 (order 0) ενώ η πρώτη του διάσπαση γίνεται σε υποπροβλήματα τάξεως 1. Γενικά, η w-στη διάσπαση ενός προβλήματος, γίνεται σε σύνολα υποπροβλημάτων τάξεως w.

Κάθε μικροϋπολογιστής μπορεί να αναλάβει την επίλυση κάποιου υποπροβλήματος Π σε δεδομένη προγραμματιστική στιγμή μέσα στο δίκτυο, αρκεί να υπάρχει ένας **βασικός αλγόριθμος διαχείρισης** που να εκτελείται σε κάποιον από αυτούς, ο οποίος θα αποφασίζει με ορισμένα κριτήρια τον τρόπο διανομής προς επίλυση των υποπροβλημάτων αυτών. Εδώ, ο κοινός στόχος είναι η ικανότητα ανταλλαγής μηνυμάτων μεταξύ των μικροϋπολογιστών με βασικό επιδιωκόμενο αποτέλεσμα την κατανεμημένη επίλυση του αρχικού προβλήματος.

Ο κύριος στόχος κάθε μεθόδου επίλυσης υποπροβλήματος είναι να μην χαθεί στην διάρκεια της διαδικασίας καμία δυνατή λύση διότι είναι φανερό ότι όλες οι λύσεις έχουν πιθανότητα να ανήκουν στο μηχανισμό δημιουργίας της άριστης λύσης του αρχικού προβλήματος.

Έστω Ω το σύνολο όλων των δυνατών λύσεων ενός υποπροβλήματος Π_m^{w-1}, τάξεως w-1. Το σύνολο των λύσεών του, έστω Ω', αποτελεί προφανώς υποσύνολο του Ω. Το υποπρόβλημα Π_m^{w-1} αναλύεται στα υποπροβλήματα - απογόνους επομένης τάξεως

$$\Pi_1^w, \Pi_2^w, ..., \Pi_p^w, ..., \Pi_n^w$$

και έστω Ω_p, $p=1,2,...,n$ τα σύνολα των δυνατών λύσεων, Ω'_p, $p=1,2,...,n$ οι λύσεις καθενός από τα υποπροβλήματα - απογόνους.

Λόγω του Ορισμού 2.2.1 θα ισχύει

$$\bigcup_{p=1}^{n} \Omega'_p = \Omega' .$$

Ορισμός 2.2.2 Μία ακριβής ανάλυση ενός υποπροβλήματος Π_m^{w-1} σε υποπροβλήματα επόμενης τάξης καλείται **κλειστή ως προς την πολυπλοκότητα**, εάν ισχύει

$$\bigcup_{p=1}^{n} \Omega_p = \Omega \quad \forall p \in \{1, 2, ..., n\} .$$

Από τον παραπάνω Ορισμό 2.2.2 είναι προφανή τα ακόλουθα:

Θεώρημα 2.2.1 Τα σύνολα δυνατών λύσεων των υποπροβλημάτων-απογόνων μιας ακριβούς, κλειστής ως προς την πολυπλοκότητα ανάλυσης υποπροβλήματος, αποτελούν κάλυμμα (cover) του συνόλου δυνατών λύσεων του πατρικού υποπροβλήματος.

Θεώρημα 2.2.2 Κάθε υποπρόβλημα μιας ακριβούς, κλειστής ως προς την πολυπλοκότητα ανάλυσης προβλήματος, έχει σύνολο δυνατών λύσεων με πληθάριθμο μικρότερο από αυτόν του πατρικού υποπροβλήματος.

Παράλληλα, βασική προϋπόθεση είναι η ύπαρξη της κλειστότητας στην διάσπαση των υποπροβλημάτων.

Ορισμός 2.2.3 **Κλειστή ανάλυση** ενός υποπροβλήματος Π_m^{w-1} καλείται κάθε ακριβής, κλειστή ως προς την πολυπλοκότητα ανάλυση αυτού σε υποπροβλήματα επομένης τάξεως, με την επιπρόσθετη ιδιότητα ότι κάθε υποπρόβλημα έχει μικρότερο πληθάριθμο από το πρόγονό του - υποπρόβλημα της προηγούμενης τάξεως.

Δηλαδή, εάν $\Pi_x^{w_1}$ υποπρόβλημα τάξης w_1 και $\Pi_y^{w_2}$ ένα απόγονο υποπρόβλημά του τάξης w_2 ($w_1 < w_2$) που έχει προκύψει μέσα από διαδοχικές κλειστές αναλύσεις, τότε $card(\Pi_x^{w_1}) > card(\Pi_y^{w_2})$.

Εδώ στην ουσία είναι φανερό ότι κάθε υποπρόβλημα αναλύεται ακριβώς σε ένα σύνολο υποπροβλημάτων - απόγονοι οι οποίοι ορίζονται σε μία αμέσως επόμενη τάξη, που και ο καθένας από αυτούς με τη σειρά του δημιουργεί μια νέα πλήρη ανάλυση απογόνων σε μια επόμενη από αυτόν τάξη.

Η κλειστή ανάλυση προϋποθέτει δύο πράγματα:

1. Ότι κάθε ένα από τα παραγόμενα αυτά υποσύνολα προβλημάτων επομένων τάξεων έχει πληθάριθμο μικρότερο από το υποπρόβλημα μιας προηγούμενης τάξεως απ' όπου προέρχεται. Άρα η σχέση προγόνων και απογόνων μιας διάσπασης δημιουργούν μία γνήσια φθίνουσα διάταξη των πληθαρίθμων τους.

2. Ότι κάθε ένα από τα παραγόμενα αυτά υποσύνολα προβλημάτων επομένων τάξεων έχει χώρο δυνατών λύσεων με πληθάριθμο μικρότερο από τον χώρο δυνατών λύσεων του υποπροβλήματος μιας προηγούμενης τάξεως απ' όπου προέρχεται. Άρα και η σχέση προγόνων και απογόνων μιας διάσπασης δημιουργούν επίσης μία γνήσια φθίνουσα διάταξη των πληθαρίθμων όσον αφορά και τα σύνολα των δυνατών λύσεων.

Αυτό σημαίνει ότι σε κάθε υποπρόβλημα από μία τάξη ανάλυσης και μετά, δεν έχει νόημα η διάσπαση σε μία επόμενη τάξη διότι οδηγεί σε τετριμμένη περίπτωση φανερής λύσης.

Από τον ορισμό 2.2.1 συμπεραίνεται άμεσα, ότι ένα υποπρόβλημα Π_m^{w-1} είναι λυμένο αν και μόνο αν έχει επιλυθεί κάθε υποπρόβλημά του – απόγονος $\Pi_p^w, \mathrm{p} = 1,2,...n$.

Έστω $T(\Pi_m^{w-1})$ ο υπολογιστικός χρόνος που απαιτείται για την επίλυση του υποπροβλήματος Π_m^{w-1}. Είναι προφανές ότι ο χρόνος που απαιτείται για την επίλυση ενός υποπροβλήματος είναι συνάρτηση του χώρου των δυνατών λύσεών του. Επομένως ο χρόνος που απαιτείται για την επίλυση του υποπροβλήματος Π_p^w είναι μικρότερος από τον χρόνο που απαιτείται για την επίλυση του Π_m^{w-1}.

Από το Θεώρημα 2.2.1 συνάγεται ότι ο χρόνος επίλυσης ενός προβλήματος είναι μικρότερος ή ίσος από τον χρόνο επίλυσης όλων των υποπροβλημάτων του της αμέσως επόμενης τάξης και άρα ισχύει το ακόλουθο

Θεώρημα 2.2.3 Η διάσπαση ενός υποπροβλήματος σε υποπροβλήματα - απογόνους της αμέσως επόμενης τάξης είναι βέλτιστη όταν οι χώροι των δυνατών τους λύσεων αποτελούν διαμέριση του χώρου δυνατών λύσεων του πατρικού προβλήματος.

Προκειμένου ο Διαχειριστής προβλήματος να αναθέτει εργασίες στους Επιλυτές, κάθε υποπρόβλημα χαρακτηρίζεται με μοναδικό τρόπο σε συνάρτηση με το αρχικό πρόβλημα.

Έστω ένα πρόβλημα Π_m^{w-1} το οποίο αναλύεται κλειστά σε φ το πλήθος υποπροβλήματα επόμενης τάξεως. Ο αλγόριθμος διάσπασης A παράγει για κάθε υποπρόβλημα – απόγονο Π_i^w έναν ακέραιο κωδικό φ_i, $i \in \{1,2,...,\varphi\}$, $\varphi_i \neq \varphi_j$ ώστε

$$A(\Pi_m^{w-1}, \varphi_i) = \Pi_i^w, \forall i = \{1,2,...,\varphi\}$$

Έτσι, αν $\Pi_y^{w_2}$ είναι υποπρόβλημα που έχει προκύψει ως απόγονος του $\Pi_x^{w_1}$ μέσα από διαδοχικές κλειστές αναλύσεις, θα υπάρχει μία πεπερασμένη ακολουθία ακεραίων φ^i με $i = 1,2,...,(w_2 - w_1)$ η οποία το προσδιορίζει μοναδικά σε σχέση με το προγονικό πρόβλημα $\Pi_x^{w_1}$. Η ακολουθία αυτή ονομάζεται **χαρακτηριστική ακολουθία** του υποπροβλήματος - απόγονος.

Κάθε υποπρόβλημα μπορεί μέσω της χαρακτηριστικής αυτής ακολουθίας να παρασταθεί μοναδικά και να αναπαραχθεί από οποιοδήποτε μέλος του Κ.Σ.Ε. (Διαχειριστή ή Επιλυτή) ξεκινώντας με ρίζα το αρχικό πρόβλημα.

Έχοντας πλέον καθορίσει το πλαίσιο διάσπασης προβλήματος σε υποπροβλήματα, στις επόμενες παραγράφους περιγράφεται το **Κ**ατανεμημένο **Σ**ύστημα **Ε**πίλυσης σε όλες τις παραμέτρους του, με κύριους άξονες αυτούς της Επικοινωνίας και της Διαχείρισης.

2.3 Κατανεμημένο Σύστημα Επίλυσης

Το προτεινόμενο Κατανεμημένο Σύστημα Επίλυσης (Κ.Σ.Ε.) λειτουργεί σύμφωνα με τους παρακάτω άξονες:

- **Διάσπαση Προβλήματος.**

Η διάσπαση με βάση τα προαναφερθέντα στην παράγραφο 2.2, γίνεται με αλγόριθμο ο οποίος παράγει υποπροβλήματα με τις παρακάτω προδιαγραφές:

- Κάθε ακολουθία απογόνων περιέχει όλες τις δυνατές λύσεις του πατρικού προβλήματος.
- Κάθε ένας απόγονος έχει πληθάριθμο μικρότερο από το πατρικό πρόβλημα.
- Κάθε απόγονο υποπρόβλημα είναι **ανιχνεύσιμο**, μπορεί δηλαδή να αναπαραχθεί από το αρχικό πρόβλημα με βάση την χαρακτηριστική του ακολουθία.

- **Επικοινωνία.**

Οι εφαρμογές επικοινωνούν μέσω ενός Data Server μοιραζόμενοι αρχεία και εγγραφές. Ο Διαχειριστής και οι Επιλυτές μοιράζονται ένα κοινό αρχείο μέσα από το οποίο γίνονται όλες οι λειτουργίες αρχικοποίησης ενώ ο Διαχειριστής μοιράζεται με κάθε Επιλυτή μία εγγραφή σε δυαδικό αρχείο εν είδει Ταχυδρομικού κουτιού (mailbox).

Η μέθοδος επικοινωνίας είναι

- για τις λειτουργίες αρχικοποίησης, αυτή του μαυροπίνακα (blackboard) όπου ο Διαχειριστής γράφει τα δεδομένα στο κοινό αρχείο απ' όπου βρίσκονται στη διάθεση όλων των Επιλυτών.
- για την επικοινωνία κατά τη διάρκεια της Επίλυσης, ένας συνδυασμός μαυροπίνακα, tuple space και call-by-record, όπου κάθε Επιλυτής έχει μοναδική πρόσβαση σε μία εγγραφή αρχείου. Στην εγγραφή αυτή ο Διαχειριστής αφήνει την εντολή επεξεργασίας, που ο Επιλυτής μπορεί σε κάθε στιγμή να προσπελάσει, ενώ στην ίδια εγγραφή ο Επιλυτής καταχωρεί το αποτέλεσμα της Επίλυσης του υποπροβλήματος που του έχει ανατεθεί. Διαχειριστής και Επιλυτές κλείνουν «ραντεβού» στο οποίο είναι γνωστός μόνον ο χώρος αλλά όχι ο ακριβής χρόνος.

Η παραπάνω μέθοδος επικοινωνίας επιλέχθηκε διότι:

- Δεν απαιτεί κανενός είδους εξειδικευμένο Λογισμικό (π.χ. MPI, παράλληλη γλώσσα προγραμματισμού κλπ).
- Το σύστημα προορίζεται για επίλυση μεγάλου μεγέθους σκληρών προβλημάτων (NP-Hard) όπου το μέγεθος των παραγομένων υποπροβλημάτων είναι επίσης μεγάλο και ο αναμενόμενος χρόνος επίλυσης δύσκολο να προσδιοριστεί, ενώ οι ανάγκες επικοινωνίας είναι μικρές σε όγκο δεδομένων.
- Το Κ.Σ.Ε. μπορεί να χρησιμοποιηθεί και για μηχανές που επικοινωνούν μέσω Διαδικτύου οπότε είναι προφανές ότι πρέπει να λειτουργεί και ασύγχρονα, με τον κάθε Επιλυτή όποτε είναι διαθέσιμος να βρίσκει την επόμενη ανάθεση υποπροβλήματος να τον αναμένει στο ταχυδρομικό του κουτί.

- **Διαχείριση Συστήματος.**

Η μέθοδος επικοινωνίας που περιγράφηκε παραπάνω υπάγεται στο μοντέλο ανταλλαγής μηνυμάτων που χρησιμοποιείται στα συστήματα κατανεμημένης μνήμης όπως είναι και το Κ.Σ.Ε. Το μοντέλο ανταλλαγής μηνυμάτων είναι φορητό σε διαφορετικές πλατφόρμες αλλά παρουσιάζει δυσκολίες κωδικοποίησης σε θέματα πλεονασμού υπολογισμών όπως και στάθμισης του φόρτου εργασίας των επεξεργαστών ιδίως σε ετερογενή περιβάλλοντα.

Η απλούστερη μέθοδος ανάθεσης εργασιών στους Επιλυτές είναι η **στατική κατανομή** (ostrich approach) [Morin, 1998] σύμφωνα με την οποία ανατίθεται ίσος όγκος δουλειάς σε κάθε επεξεργαστή. Πρόσφατες μελέτες [Bohn, Lamont, 2002] δείχνουν βελτίωση της απόδοσης του συστήματος μέχρι και 92% όταν γίνεται **δυναμική ανάθεση** των εργασιών όπου ο ισχυρότερος επεξεργαστής αναλαμβάνει μεγαλύτερου χρόνου εργασίες από τον λιγότερο ισχυρό.

Όμως, ακόμα και αν το σύστημα αποτελείται από ομογενείς επεξεργαστές, δεν υπάρχει καμία εγγύηση ότι αυτοί θα έχουν ισοδύναμη απόδοση, ούτε ότι τα υποπροβλήματα θα είναι ομοιόμορφα και προβλέψιμα σε μέγεθος. Αυτό που μπορεί κάποιος να θεωρήσει είναι ότι ο αριθμός τους είναι μεγάλος και οπωσδήποτε πολύ μεγαλύτερος του πλήθους των διατιθεμένων επεξεργαστών.

Έτσι επιλέχθηκε σαν μέθοδος ανάθεσης υποπροβλημάτων μία παραλλαγή του overpartitioning, σύμφωνα με την οποία ο Διαχειριστής αναθέτει το επόμενο υποπρόβλημα στον Επιλυτή μόλις αυτός καταστεί διαθέσιμος (λύσει δηλαδή το προηγούμενο και απαντήσει τα αποτελέσματα).

- **Διανομή Υποπροβλημάτων.**

Το μέγεθος του κάθε υποπροβλήματος από το οποίο εξαρτάται σχεδόν αποκλειστικά ο αναμενόμενος υπολογιστικός χρόνος επίλυσης, είναι ο κρίσιμος παράγων που αποφασίζει το που αυτό θα επιλυθεί. Ένας δεύτερος παράγων που επίσης αποφασίζει την πολιτική διανομής είναι το κόστος (χρόνος) επικοινωνίας.

Το αρχικό πρόβλημα διασπάται σε υποπροβλήματα μικρότερου μεγέθους μέσω κλειστής ανάλυσης (παράγραφος 2.2) με αντικειμενικό σκοπό την «μετάπτωση φάσης» που θα το μεταφέρει από την κατηγορία των «δύσκολα» σε αυτή των «εύκολα» επιλύσιμων στιγμιότυπων.

Τα υποπροβλήματα δίνονται σε Επιλυτές εκτός από αυτά που λόγω του κόστους επικοινωνίας και των νεκρών χρόνων αναμονής του Διαχειριστή, κρίνεται ότι είναι συμφερότερο να επιλυθούν τοπικά.

- **Μέθοδος Επίλυσης.**

Είναι προφανές ότι εξαρτάται από το ποιο είναι το πρόβλημα που πρόκειται να επιλυθεί από το Κ.Σ.Ε., είναι δηλαδή άμεσα εξαρτημένη από το προς επίλυση πρόβλημα (problem dependent).

Υπάρχουν τρία γενικά Υποσυστήματα (πακέτα), τα οποία περιλαμβάνει ο Διαχειριστής Προβλήματος (PM) τα 2.4, 2.5, 2.6 που αναφέρονται στη συνέχεια (Εικόνα 4), ενώ ο Επιλυτής Προβλήματος (PS) λειτουργεί στα πακέτα 2.4 και 2.5.

Να σημειωθεί, ότι όπως θα δούμε στο τέλος της παρούσης διατριβής είναι δυνατόν να έχουμε nested φαινόμενο του όλου Συστήματος.

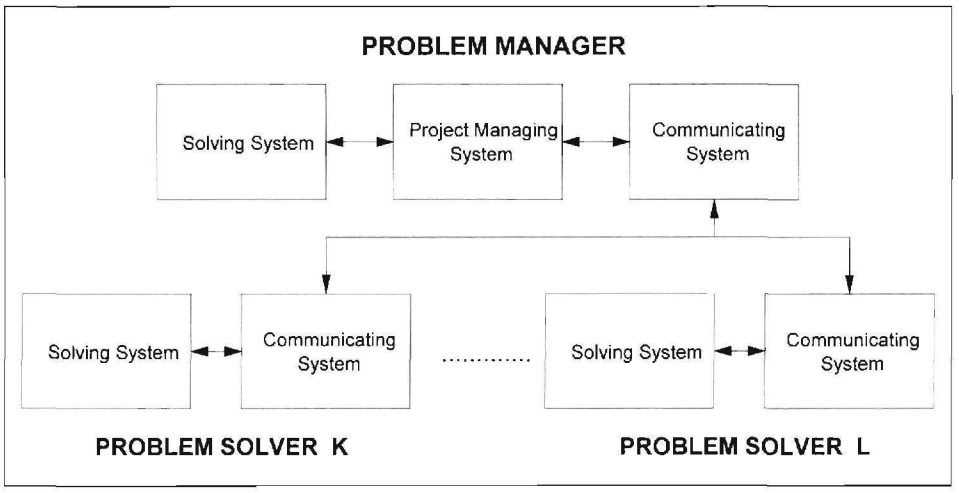
Από τα παρακάτω υποσυστήματα τα δύο πρώτα αναλύονται και περιγράφονται διεξοδικά στις αντίστοιχες παραγράφους ενώ μικρή μόνον αναφορά γίνεται στο Σύστημα Επίλυσης Υποπροβλήματος που εξαρτάται από την ίδια τη φύση του προβλήματος. Στην εφαρμογή του Κ.Σ.Ε. που περιγράφεται σε επόμενο Κεφάλαιο γίνεται αναλυτική περιγραφή του τρίτου υποσυστήματος ειδικά για το Πρόβλημα της μέγιστης κλίκας.

2.4 Σύστημα Διαχείρισης Έργου (Project Managing System) Διαχειρίζεται τον αλγόριθμο που διοικεί και συντονίζει όλα τα συστήματα.

2.5 Σύστημα Επικοινωνίας (Communicating system) Διεκπεραιώνει κάθε είδους επικοινωνία ανάμεσα στον Problem Manager και τους Solvers.

2.6 Σύστημα Επίλυσης Υποπροβλήματος (Solving system) Διαχειρίζεται την επίλυση των υποπροβλημάτων.

Εικόνα 4 – Κατανεμημένο Σύστημα Επίλυσης



Είναι προφανές ότι ένας κύριος στόχος είναι η ελαχιστοποίηση του ολικού χρόνου προσδιορισμού της άριστης λύσης του αρχικού προβλήματος Π. Επειδή το σύστημά μας αναλύεται στα υποσυστήματα που αναφέρθηκαν, η νόρμα του μέγιστου αναμενόμενου υπολογιστικού χρόνου αναλύεται στα επιμέρους αθροίσματα:

$$T_{max} = T_C + T_S + T_{PM}$$

T_C συνολικός χρόνος επικοινωνίας των μικροϋπολογιστών (Communicating time).

T_S συνολικός χρόνος επίλυσης των υποπροβλημάτων (Solving time).

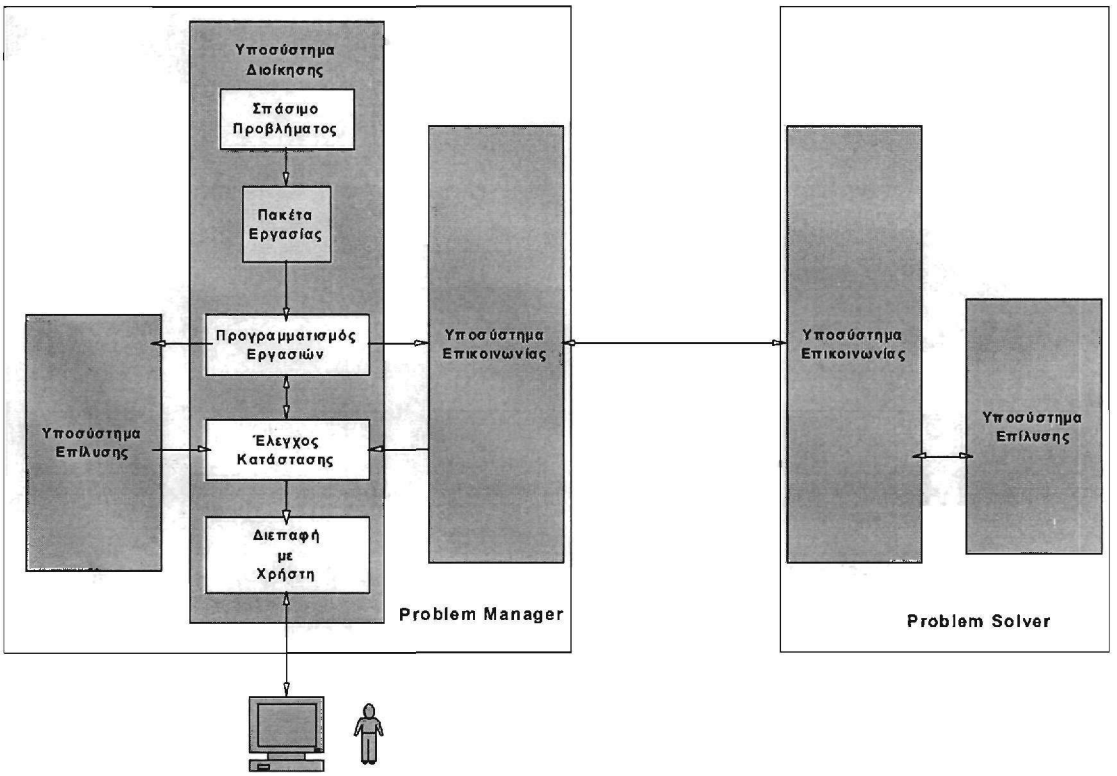
T_{PM} χρόνος διαχείρισης Έργου (Project Managing time).

Επομένως είναι αναγκαίο να γίνει αφενός μεν ελαχιστοποίηση όχι μόνον του χρόνου του βασικού αλγορίθμου επίλυσης κάθε συγκεκριμένου προβλήματος που αφορά το T_S, αλλά και του χρόνου επικοινωνίας των Επιλυτών με τον Διαχειριστή Προβλήματος καθώς και του χρόνου που απαιτεί το βασικό πακέτο διαχείρισης του όλου προβλήματος, αφετέρου δε εκμετάλλευση του παραλληλισμού μεταξύ αυτών.

2.4 Σύστημα Διαχείρισης Έργου

Το **Σ**ύστημα **Δ**ιαχείρισης **Έ**ργου (**Σ.Δ.Ε.**) διαχειρίζεται **πακέτα εργασίας** (υποπροβλήματα) και **υπολογιστικούς πόρους** (Επιλυτές) με αρμοδιότητες στις οποίες περιλαμβάνονται:

Εικόνα 5 - Διαχείριση Έργου



- Καθορισμός των υπολογιστικών πόρων (resources).
- Αρχικοποίηση (initialization) του όλου συστήματος.
- Χωρισμός (διάσπαση) του αρχικού προβλήματος Π (project) σε υποπροβλήματα (packets).

- Κατανομή (resource allocation) των πακέτων εργασίας στους διαθέσιμους υπολογιστικούς πόρους.
- Έλεγχο της κατάστασης (status monitoring) του συστήματος.
- Λειτουργίες τερματισμού του Συστήματος.

Ο PM διατηρεί στην τοπική του μνήμη τις ακόλουθες μεταβλητές:

1. **Πλήθος Επιλυτών.** Έστω d το πλήθος των Επιλυτών (PS) που τρέχουν στις μηχανές του Συστήματος. Να σημειωθεί ότι είναι δυνατόν να τρέχουν περισσότεροι του ενός Επιλυτές ανά υπολογιστή.

2. **Αρχείο Προβλήματος.** Αλφαριθμητικό Prob_file που κρατά την τοποθεσία όπου βρίσκεται το αρχείο το οποίο περιέχει την κωδικοποίηση του προς επίλυση προβλήματος.

3. **Μητρώο Επιλυτών.** Πίνακας R_i, $i = 0,..,d-1$ που κάθε στοιχείο αφορά στον Επιλυτή με τον αντίστοιχο ID αριθμό. Η κάθε θέση του πίνακα δείχνει την κατάσταση στην οποία βρίσκεται ο αντίστοιχος PS κάθε χρονική στιγμή. Ο R_i χρησιμοποιείται από τον PM προκειμένου κατά την διάρκεια της επίλυσης να παρακολουθεί τους PS. Ένας PS κατατάσσεται από τον Διαχειριστή σε μία από τις παρακάτω τρεις καταστάσεις.

Πίνακας 4 - Δυνατές Καταστάσεις Επιλυτών (PS)	
0	Διαθέσιμος, σε αναμονή της επόμενης ανάθεσης πακέτου (waiting).
1	Απασχολημένος με την επίλυση υποπροβλήματος (busy).
2	Μη διαθέσιμος (unavailable).

4. Τοπική Λίστα Υποπροβλημάτων. Πίνακας ακεραίων S_{ij}, δύο διαστάσεων όπου κάθε γραμμή αποθηκεύει την χαρακτηριστική ακολουθία ενός υποπροβλήματος σε σχέση με το αρχικό πρόβλημα. Για ένα υποπρόβλημα τάξης w τα πρώτα ισάριθμα στοιχεία της αντίστοιχης γραμμής του πίνακα θα έχουν τιμές σημαντικές ενώ τα υπόλοιπα θα είναι μηδέν.

Καθορισμός υπολογιστικών πόρων. Πρόκειται για την ταυτοποίηση και αναγνώριση κάθε επεξεργασίας που τρέχει σαν μέρος του όλου Συστήματος. Με την εκκίνηση του Συστήματος ο PM δέχεται σαν είσοδο το πλήθος των Επιλυτών που πρόκειται να διοικήσει. Ο Διαχειριστής Προβλήματος και οι Επιλυτές χαρακτηρίζονται με μοναδικό τρόπο μέσα στο Σύστημα. Σε κάθε επεξεργασία που τρέχει δίνεται ένας χαρακτηριστικός ακέραιος (ID) που αποτελεί την ταυτότητά της και την χαρακτηρίζει μοναδικά σε σχέση με τις υπόλοιπες. Έτσι, εφόσον διατίθενται d το πλήθος Επιλυτές,

- ο Διαχειριστής παίρνει ID = d.
- οι Επιλυτές τους διαδοχικούς διαφορετικούς ακέραιους 0,1, ..., $d-1$.

Αρχικοποίηση συστήματος. Ο PM δέχεται σαν είσοδο το όνομα του αρχείου στο οποίο βρίσκεται αποθηκευμένο το στιγμιότυπο του προβλήματος που πρόκειται να επιλυθεί και το φορτώνει στην τοπική του μνήμη. Θέτει στην κατάσταση waiting (0) όλους τους PS, ενώ με αποστολή μηνυμάτων ενημερώνει τους Επιλυτές για

- την τοποθεσία του αρχείου-προβλήματος.
- το πλήθος αυτών d που συμμετέχουν στην επίλυση.

Χωρισμός αρχικού προβλήματος σε υποπροβλήματα. Το Σ.Δ.Ε. προκειμένου να παράγει τα πακέτα εργασίας (work packages) χρησιμοποιεί ένα άνω φράγμα μεγέθους *Critical_High* και ένα φράγμα τάξης *Critical_Level* (≥ 1), σαν κριτήρια για τη διάσπαση των υποπροβλημάτων. Κάθε υποπρόβλημα διασπάται σε υποπροβλήματα του επομένου επιπέδου μέχρι είτε το μέγεθός του να γίνει μικρότερο από *Critical_High* ή διαφορετικά η τάξη διάσπασης να φθάσει στην τιμή *Critical_Level*. Σαν αποτέλεσμα της διάσπασης προκύπτει μία χαρακτηριστική ακολουθία για κάθε υποπρόβλημα.

Κατανομή πακέτων εργασίας στους υπολογιστικούς πόρους. Παράλληλα με τη διαδικασία διάσπασης ξεκινά η διανομή των υποπροβλημάτων. Προκειμένου να ελαχιστοποιηθεί το κόστος της επικοινωνίας με τους Επιλυτές καθώς και οι νεκροί χρόνοι αναμονής, ο Διαχειριστής χρησιμοποιεί ένα κάτω φράγμα, το *Critical_Low.*

Ο PM επιλύει τοπικά εκείνα τα υποπροβλήματα για τα οποία ισχύει

$$\mathrm{T}(\Pi_p^w) \leq Limit$$

όπου

$$Limit = \begin{Bmatrix} Critical\_High, \alpha\nu\ w = 0 \\ Critical\_Low, \alpha\nu\ w > 0 \end{Bmatrix}$$

Το υποσύστημα διάσπασης παράγει το επόμενο υποπρόβλημα το οποίο εφόσον έχει μέγεθος μικρότερο από *Critical_Low* αποθηκεύεται στην Τοπική Λίστα Υποπροβλημάτων S_{ij}. Σε διαφορετική περίπτωση, το Σ.Δ.Ε. ελέγχει στο μητρώο των επιλυτών, βρίσκει τον πρώτο ελεύθερο Επιλυτή και το αναθέτει.

Εφόσον δεν υπάρχει ελεύθερος Επιλυτής, ο Διαχειριστής μπαίνει σε κατάσταση αναμονής κατά την οποία ανασύρει το παλαιότερο υποπρόβλημα από την S_{ij} και το επιλύει τοπικά, ενώ καλεί τον έλεγχο κατάστασης του συστήματος μέχρι να βρεθεί ελεύθερος Επιλυτής. Να σημειωθεί ότι το παραπάνω σχήμα υλοποιήθηκε σε πρώτη φάση αν και όπως θα δούμε στο Κεφάλαιο 4 δεν είναι πάντα το καλύτερο δυνατό. Τέλος στο Κεφάλαιο 5 θα δούμε ένα εναλλακτικό σχήμα για τις περιπτώσεις που ο PM ελέγχει σχετικά μεγάλο αριθμό PS.

Έλεγχος κατάστασης συστήματος. Χρησιμοποιεί το υποσύστημα επικοινωνίας προκειμένου να ενημερώνει το Μητρώο. Συνδυάζει τις απαντήσεις των Επιλυτών καθώς και τις απαντήσεις του τοπικού υποσυστήματος επίλυσης προκειμένου να παράγει την βέλτιστη μέχρις στιγμής λύση.

Τερματισμός συστήματος. Μετά την ανάθεση όλων των υποπροβλημάτων για επίλυση και την είσοδο του συστήματος σε τελική κατάσταση αναμονής, φροντίζει για την συγκέντρωση όλων των απαντήσεων από τους Επιλυτές αποφασίζοντας το πότε τερματίζει η διαδικασία επίλυσης με βασικό στόχο να μην χαθεί κανένα μήνυμα απάντησης από Επιλυτή (transient message). Τέλος, στέλνει μηνύματα τερματισμού (kill signals) στους Επιλυτές που δεν χρειάζονται πλέον.

Στους Πίνακες 5 έως και 8 περιγράφονται οι βασικοί αλγόριθμοι του **υποσυστήματος διαχείρισης έργου**.

Πίνακας 5 - PM_Initialization

```
Εισαγωγή d, Prob_file
w=0
Εαν T(Π^w) ≤ Critical_High τότε
{
        Τοπική επίλυση προβλήματος
        Εξαγωγή αποτελεσμάτων
        Τέλος
}
Για i=0 έως d-1 θέσε
        Ri=0                    /*Θέτει σε wait status τους PS*/
Δημιουργία καναλιών επικοινωνίας με PS
Αποστολή Prob_file, d στους PS
```

Πίνακας 6 - PM_Create_SubProblems

```
w=w+1
Για κάθε υποπρόβλημα Π_p^w που δημιουργείται
{
        Εαν Π_p^w υπακούει στους κανόνες απόρριψης, επόμενο υποπρόβλημα
        Εαν T(Π_p^w) ≤ Critical_High τότε
        {
                Εαν T(Π_p^w) ≤ Critical_Low τότε
                        Πρόσθεση Π_p^w στην τοπική λίστα Sij
                διαφορετικά
                        Π_p^w διαθέσιμο για ανάθεση σε PS
        }
        διαφορετικά
        {
                Εαν w < Critical_Level τότε
                        PM_Create_SubProblems
                διαφορετικά
                        Π_p^w διαθέσιμο για ανάθεση σε PS
        }
}
```

Πίνακας 7 - PM_Assign_Subproblems

Όσο υπάρχουν υποπροβλήματα Π_p^w που δεν έχουν επιλυθεί
{
    Εαν υπάρχει i ώστε Ri=0 τότε
    {
        Ανάθεση του Π_p^w στον PS με ID=i
        Θέσε Ri=1
    }
    διαφορετικά
    {
        Όσο δεν υπάρχει i ώστε Ri=0 /*Wait Status*/
        {
            Ανάσυρε, αν υπάρχει, υποπρόβλημα $\Pi_p^{w'}$ από Sij
            Τοπική επίλυση προβλήματος $\Pi_p^{w'}$
            Έλεγχος όλων των PS για απάντηση
            Ενημέρωση μητρώου Ri
        }
        Ανάθεση του Π_p^w στον PS με Ri=0
        Θέσε Ri=1
    }
}

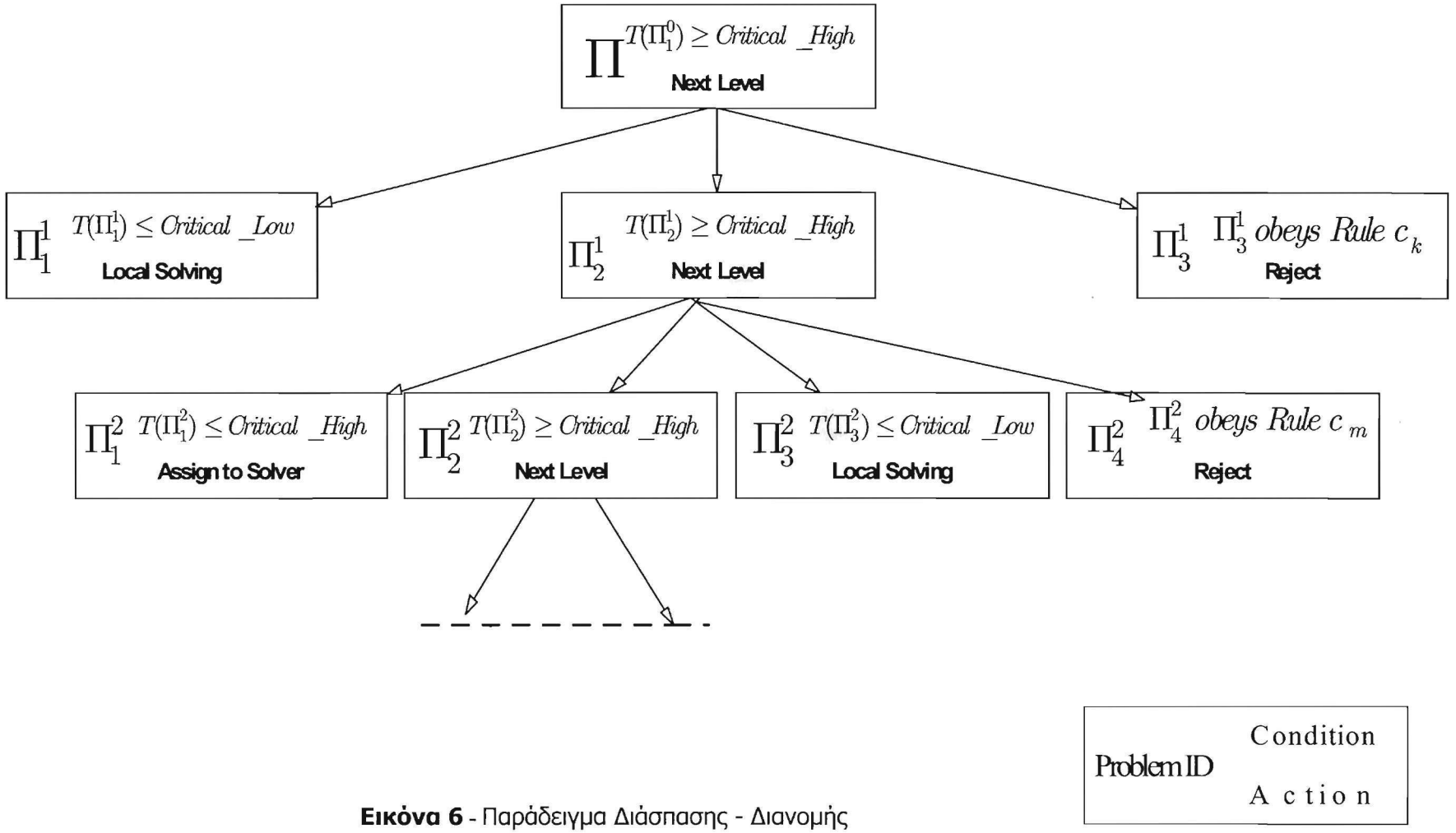
Σε παρακάτω Κεφάλαιο όπως έχει ήδη αναφερθεί θα αναπτυχθεί μια διαφορετική συμπεριφορά του PM κατά τη διάρκεια του Wait Status.

Πίνακας 8 - PM_Termination

```
Count=0
Όσο count < d
{
        Για i=0 έως d-1 εκτέλεσε
        {
                Εαν Ri=0 τότε
                {
                        count=count+1
                        Αποστολή μηνύματος τερματισμού στον i-PS
                        Ri=2
                }
                διαφορετικά
                {
                        Εαν Ri=1 τότε
                        {
                                Εαν ο i-Solver απάντησε
                                {
                                  count=count+1
                                  Αποστολή μηνύματος τερματισμού στον i-PS
                                  Ri=2
                                }
                        }
                }
        }
}
Εξαγωγή Αποτελεσμάτων
Τέλος
```

Στην Εικόνα 6 δίνεται ένα παράδειγμα διάσπασης προβλήματος από τον Διαχειριστή σε υποπροβλήματα όπου ανάλογα με την περίπτωση είτε

- προχωρά σε **επόμενο επίπεδο** διάσπασης, ή
- τοποθετεί το πρόβλημα στην λίστα για **τοπική επίλυση**, ή
- **στέλνει** το πρόβλημα σε κάποιον από τους διαθέσιμους Επιλυτές, ή
- **απορρίπτει** το πρόβλημα εφόσον αυτό υπακούει σε κάποιους κανόνες. Πρόκειται για ένα σύνολο $C = \{c_1, c_2, \ldots, c_z\}$ από **κανόνες (Rules) απόρριψης**, που όταν ένα υποπρόβλημα υπακούει σε έναν τουλάχιστον, δεν υπάρχει ελπίδα να δώσει καλύτερη λύση από την ήδη γνωστή.



Εικόνα 6 - Παράδειγμα Διάσπασης - Διανομής

2.5 Σύστημα Επικοινωνίας (Communicating system)

Η ανταλλαγή πληροφορίας ανάμεσα στα μέλη του Κ.Σ.Ε. γίνεται από το υποσύστημα επικοινωνίας το οποίο προκειμένου να μεταφέρει κάθε είδους μηνύματα από τον PM στους PS και αντίστροφα, χρησιμοποιεί τρία αρχεία:

Α) **Start_file** με περιεχόμενο:

- Το πλήθος των Επιλυτών d .
- Το όνομα του αρχείου Έργου Prob_file.

Β) **Comm_file** δυαδικό αρχείο (binary file) με d το πλήθος εγγραφές mailbox[addr], addr$= 0,1,...,d-1$ μία για κάθε Επιλυτή. Κάθε εγγραφή mailbox[addr] περιέχει μέχρι τρία τμήματα που το καθένα χρησιμοποιείται από ένα από τα τρία υποσυστήματα που απαρτίζουν τον Διαχειριστή Προβλήματος:

- **Ταυτότητα Αποστολέα** (SenderID). Πρόκειται για δεδομένα του υποσυστήματος επικοινωνίας.
  - $SenderID = d$ αν πρόκειται για τον Διαχειριστή ή
  - $SenderID \in \{0,1,...,d-1\}$ αν είναι κάποιος από τους Επιλυτές

- **Στοιχεία Υποπροβλήματος** (SpData). Περιέχει δεδομένα του υποσυστήματος επίλυσης που είναι:
  - κωδικός έργου (χαρακτηριστική ακολουθία υποπροβλήματος) φ^i, και
  - προκειμένου περί προβλήματος βελτιστοποίησης η καλύτερη μέχρις στιγμής λύση, BSF.

$$SpData = (\varphi^1, \varphi^2, ..., \varphi^n, BSF).$$

Χωρίς βλάβη της γενικότητας των αλγορίθμων και για λόγους απλοποίησης, δεχόμαστε ότι η τιμή BSF είναι μονοδιάστατη, π.χ. ένας ακέραιος με $BSF > 0$ αλλά και $\varphi^i > 0 \ \forall \ i = 1,..,n > 0$.

- **Εντολή** (Cmd). Είναι τα δεδομένα του υποσυστήματος διοίκησης που αποστέλλει ο PM στους PS. Στην περίπτωση του SPMD μοντέλου τα σήματα μπορεί να είναι

$$Cmd \in \{S, E, W\}, \text{ όπου}$$

S = Λύσε E = Τέλος W = Περίμενε

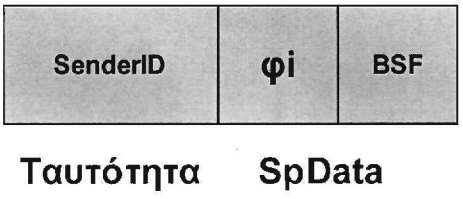
ενώ στην περίπτωση MPMD επεξεργασίας είναι

$$Cmd \in \{P_1, P_2, ...P_k, E, W\}$$

όπου P_i είναι ο κωδικοποιημένος αριθμός διαδικασίας (procedure) που θα εκτελεστεί από τον Επιλυτή.

Ωστόσο (Εικόνα 7) στην περίπτωση της επεξεργασίας SPMD που είναι και αυτή που ενδιαφέρει στην παρούσα, το πεδίο Cmd μπορεί να αντικατασταθεί από συνδυαστικές τιμών των SenderID και SpData σε σχέση με τις τιμές των τοπικών μεταβλητών κάθε διαδικασίας όπως θα δούμε παρακάτω.

Εικόνα 7 - Δομή Εγγραφής Mailbox



Γ) **Res_file$_i$** όπου $i = 0,1,...,d-1$ αρχεία κειμένου, ένα για κάθε Επιλυτή όπου καταχωρεί την καλύτερη λύση κάθε φορά που βρίσκει (εφόσον βρει).

Το αρχείο Start_file χρησιμοποιείται για την αρχικοποίηση και τον αρχικό συντονισμό (barrier synchronization) των PS από τον PM. Με την εκκίνηση του Κ.Σ.Ε. (Πίνακας 9) οι Επιλυτές προσπαθούν να ανοίξουν το εν λόγω αρχείο. Η επιτυχία της ανάγνωσης σηματοδοτεί ότι ο Διαχειριστής έχει προετοιμάσει (δημιουργήσει) τα κανάλια επικοινωνίας (Comm_file) και είναι έτοιμος να μοιράσει πακέτα εργασίας, ενώ οι Επιλυτές πληροφορούνται

- το ID του Διαχειριστή,
- την ακριβή τοποθεσία του Prob_file που περιέχει το στιγμιότυπο του προβλήματος, οπότε ξεκινά το φόρτωμα στην τοπική τους μνήμη.

Με την εκκίνηση της διαδικασίας επίλυσης η επικοινωνία γίνεται μέσα από σταθερής μορφής μηνύματα μέσω των mailbox. Σε κάθε μία από τις εγγραφές αυτές, που αποτελούν το κανάλι επικοινωνίας Επιλυτή και Διαχειριστών, έχει πρόσβαση μόνον ο αντίστοιχος Επιλυτής, ενώ ο Διαχειριστής έχει δυνατότητα πρόσβασης σε όλες.

Η αποστολή μηνύματος από PS προς τον PM έχει κύριους στόχους

1. Την ενημέρωση του PM ότι ο PS έχει περατώσει την επεξεργασία του υποπροβλήματος που του ανατέθηκε, οπότε είναι διαθέσιμος.

2. Την ενημέρωση του PM για τα αποτελέσματα που προέκυψαν από την επίλυση του υποπροβλήματος.

ενώ η αποστολή μηνύματος από τον PM προς τον PS έχει στόχους

1. Την ανάθεση σ' αυτόν του επομένου υποπροβλήματος για επίλυση.

2. Την ενημέρωση του Επιλυτή για την μέχρις στιγμής καλύτερη λύση που έχει προκύψει από ολόκληρο το σύστημα επίλυσης.

Τόσο ο Διαχειριστής Προβλήματος όσο και οι Επιλυτές χρησιμοποιούν την συνάρτηση **Communicate** για να **στείλουν** και την **Receive** για να **λάβουν** μέσω του mailbox μήνυμα:

```
Communicate(addr, iden, pdata)
{
        mailbox[addr].SenderID=iden
        mailbox[addr].SpData=pdata
        seek(comm_file,addr)
        writefile(comm_file,mailbox[addr])
}


Receive(addr, mailbox)
{
        seek(comm_file, addr)
        readfile(comm_file, mailbox[addr])
}
```

Πίνακας 9 - Συγχρονισμός & Αρχικοποίηση Επικοινωνίας PM

Διαχειριστής Προβλήματος (PM)	**Επιλυτής Προβλήματος (PS)**

Εισαγωγή Προβλήματος
Άνοιγμα Comm_file χωρίς περιορισμό για τρίτους
Για i=0 έως d-1 εκτέλεσε
Communicate(i, d+1, 0, 0)
Δημιουργία Start_file με αποκλειστικά δικαιώματα
Writefile(Start_file, d, Prob_file)
Κλείσιμο Start_file
	Επανέλαβε
	{
	Άνοιγμα Start_file για ανάγνωση χωρίς περιορισμό για τρίτους
	}
	Μέχρι άνοιγμα Start_file επιτυχές

	Readfile(Start_file, d, Prob_file)

Μετά τις λειτουργίες αρχικοποίησης, ο Επιλυτής με κωδικό myid προσπελαύνει στην εγγραφή του Comm_file με αριθμό myid και διαβάζει το μήνυμα που περιέχεται στο mailbox. Όπως φαίνεται στον Πίνακα 9, ο PM έχει δώσει τιμή στα $SenderID = d + 1$. Ο PS διαβάζει στο mailbox του και προχωρά στην δημιουργία και τοπική επίλυση του υποπροβλήματος **μόνον από τη στιγμή που θα βρει:**

- $SenderID = d$ (μήνυμα από PM)
- να υπάρχει νεότερο υποπρόβλημα

Στον Πίνακα 10 φαίνεται ο βασικός αλγόριθμος επικοινωνίας του PS.

Πίνακας 10 – Επικοινωνία Επιλυτή Προβλήματος

```
Συνεχώς εκτέλεσε
{
 mailbox[myid].SenderID=myid
Όσο (mailbox[myid].SenderID <> d)
      OR (mailbox[myid].φ^i = τοπικές τιμές φ^i) εκτέλεσε
        Receive(myid, mailbox)                    // Wait Status
 Εαν mailbox[myid].BSF = -1 τότε                  // Kill Signal Received
 {
        Closefile(comm_file)
        Exit
 }
                              αλλοιώς
 {
        Ενημέρωση τοπικών δεδομένων
        Επίλυση Υποπροβλήματος φ^i
        Εαν βρέθηκε καλύτερη λύση
        {
           Ενημέρωση BSF
           Δημιουργία και ενημέρωση Res_file_myid
        }
        Communicate(myid,myid,Spdata)
  }
}
```

Ο PM αναθέτει στον i-Solver (για τον οποίο γνωρίζει μέσα από το μητρώο R_i ότι βρίσκεται σε κατάσταση αναμονής) το επόμενο υποπρόβλημα με απλή κλήση της συνάρτησης

Communicate(i, d, SpData)

αφού πρώτα ενημερώσει τις μεταβλητές SpData με τις τιμές του τρέχοντος υποπροβλήματος.

Στην περίπτωση που σύμφωνα με το μητρώο δεν υπάρχει διαθέσιμος Επιλυτής, ο PM πέφτει σε Κατάσταση Αναμονής (Wait Status, Πίνακας 7). Η επικοινωνία ελέγχου απαντήσεων περιγράφεται στον Πίνακα 11.

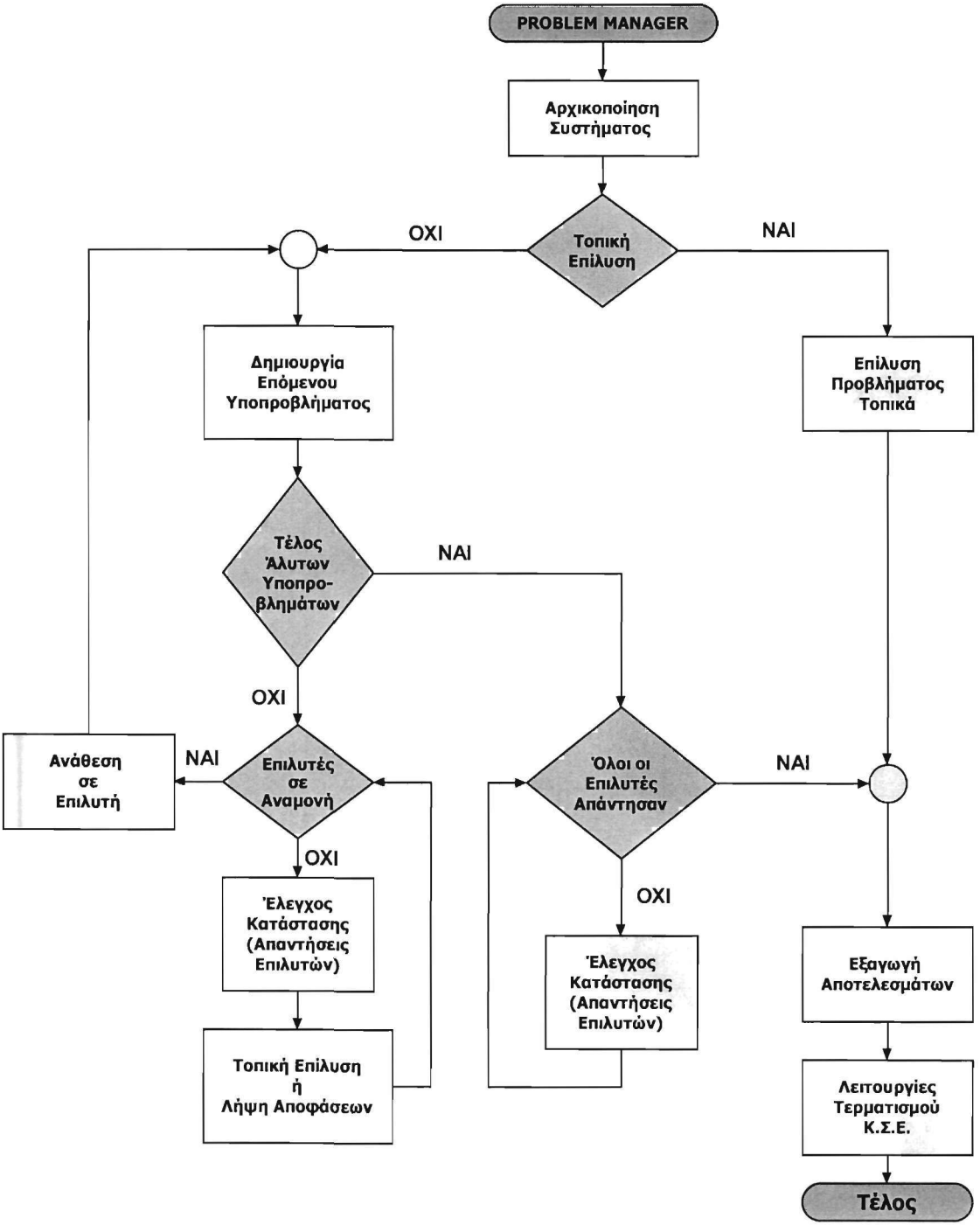
Όταν έχει ολοκληρωθεί η ανάθεση των πακέτων, ο PM κατά τη διάρκεια της διαδικασίας τερματισμού (Πίνακας 8), στέλνει εντολές τερματισμού στους Επιλυτές που είτε είναι αδρανείς, είτε αναφέρουν κατά την διαδικασία τερματισμού του Κ.Σ.Ε. ότι τελείωσαν:

```
pdata.BSF = -1
Communicate(i, d, pdata)
```

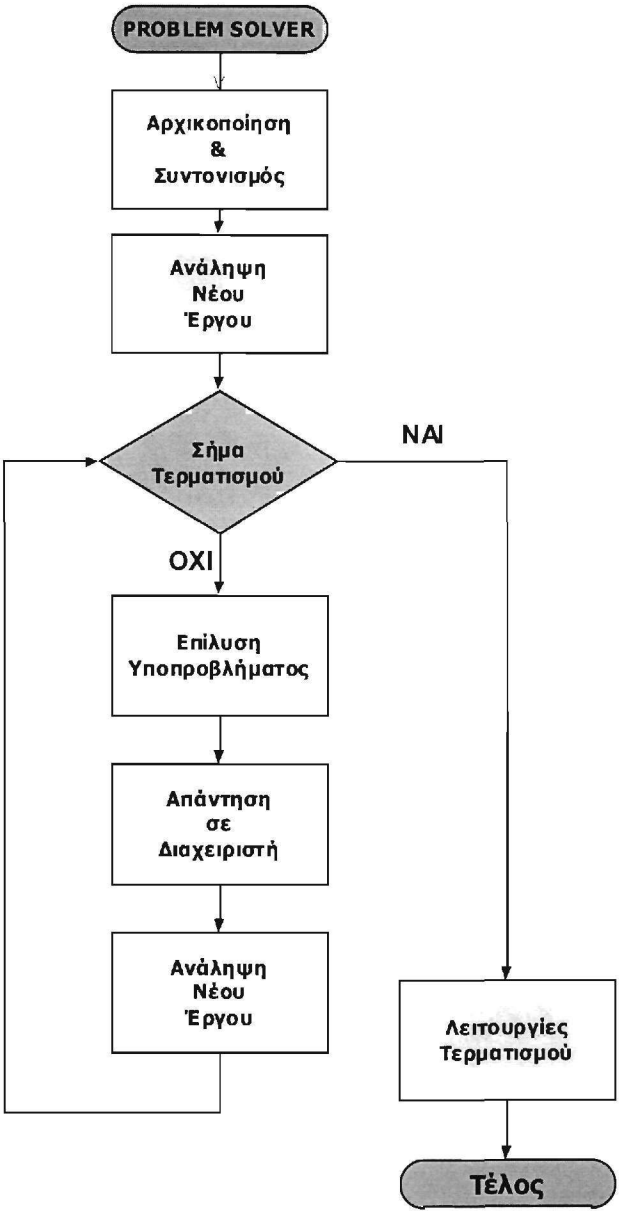
Πίνακας 11 – Έλεγχος Επιλυτών

```
Για addr=0 έως d-1 εκτέλεσε
{
    Receive(addr, mailbox)
    Εαν mailbox[addr].SenderID = addr
        {
            Ενημέρωση Μητρώου R_addr    // κατά την επίλυση, 2 τερματισμό
            Διάβασμα Αρχείου Res_file_addr
            Ενημέρωση καλύτερης Λύσης
        }
}
```

Εικόνα 8 – Διαχειριστής Προβλήματος



Εικόνα 9 – Επιλυτής Προβλήματος



2.6 Σύστημα Επίλυσης Υποπροβλήματος

Εδώ περιλαμβάνεται η μέθοδος επίλυσης του Προβλήματος. Είναι προφανές και έχει ήδη αναφερθεί ότι εξαρτάται άμεσα από το ίδιο το Πρόβλημα και επομένως το **Σ**ύστημα **Ε**πίλυσης **Υ**ποπροβλήματος (**Σ.Ε.Υ.**) μπορεί να θεωρηθεί σαν μαύρο κουτί (black box).

Εικόνα 10 – Σύστημα Επίλυσης Υποπροβλήματος

Το **Σ.Ε.Υ.** δέχεται σαν είσοδο:

- ένα υποπρόβλημα Π_p^w
- τις όποιες παραμέτρους επίλυσης όπως για παράδειγμα και προκειμένου περί προβλήματος βελτιστοποίησης την καλύτερη μέχρις στιγμής λύση BSF κλπ.

ενώ δίνει σαν έξοδο τα αποτελέσματα της επίλυσης του υποπροβλήματος Π_p^w έστω $\mathrm{Result}(\Pi_p^w)$.

Προκειμένου το Κ.Σ.Ε. να παρέχει ευελιξία, το Υποσύστημα Επίλυσης μπορεί να ενσωματώνεται στο υπόλοιπο σύστημα εν είδει plug-in [Beck, Dongarra, +, 1999], ενώ **είναι δυνατόν να επιλεγεί οποιαδήποτε μέθοδος επίλυσης, είτε ίδια για όλους τους Επιλυτές, αλλά ακόμα και διαφορετική ανά ομάδες Επιλυτών, δεδομένου ότι παλαιότερες εργασίες έχουν δείξει ότι σε αυτήν την περίπτωση αυξάνεται η ταχύτητα επίλυσης** [Clearwater, Hogg +, 1992] [Hogg, Huberman, 1993] .

Στα επόμενα χρησιμοποιείται το Κατανεμημένο Σύστημα Επίλυσης προκειμένου να επιλυθούν μεγάλου μεγέθους στιγμιότυπα του NP-Complete προβλήματος της εύρεσης της μέγιστης κλίκας σε προσανατολισμένο γράφημα.

Κεφάλαιο 3

ΕΦΑΡΜΟΓΗ ΣΤΟ CLIQUE-NUMBER

ΕΦΑΡΜΟΓΗ ΣΤΟ CLIQUE-NUMBER

3.1 Μέγιστη Κλίκα

3.1.1 Ορισμοί & Εφαρμογές

Έστω G ένα απροσανατόλιστο γράφημα με V και E τα σύνολα των κόμβων και των ακμών του, αντίστοιχα:

$$G = (V, E)$$
$$V = \{v_1, v_2, ..., v_n\}$$
$$E = \{(v_i, v_j) \in V \times V, i \neq j \in \{1, 2, ..., n\}\}$$

Ένα γράφημα G ανάμεσα σε άλλες, χαρακτηρίζεται από τις παρακάτω παραμέτρους:

- το μέγεθος που είναι το πλήθος των κόμβων του, $size = |V|$.
- το πλήθος των ακμών του, $edges = |E|$.
- Για κάθε κόμβο $u \in V$, ο βαθμός $\deg(u)$ που είναι το πλήθος των ακμών που καταλήγουν σ'αυτόν (vertex degree).
- Η μέση πυκνότητα των κόμβων

$$density = \frac{2edges}{size(size-1)}$$

Ένα σύνολο $C \subseteq V$ του οποίου οι κόμβοι συνδέονται ανά δύο με ακμές από το E είναι μία **κλίκα** (clique) του V. **Maximal κλίκα** του V είναι μία κλίκα που δεν περιέχεται σε καμία άλλη κλίκα του.

Το πρόβλημα της εύρεσης της μέγιστης κλίκας σε ένα απροσανατόλιστο γράφημα (**maximum clique problem**) είναι το παρακάτω πρόβλημα βελτιστοποίησης:

MAX-CLIQUE: Δοθέντος ενός γραφήματος $G = (V, E)$ να βρεθεί το **μεγαλύτερο** υποσύνολο $C \subseteq V$ τέτοιο ώστε για κάθε διαφορετικά $u, v \in C$ να ισχύει $(u, v) \in E$.

Το ισοδύναμο πρόβλημα απόφασης είναι αυτό της εύρεσης του πληθάριθμου της μέγιστης κλίκας (**clique number problem**) που ορίζεται ως εξής:

CLIQUE-NUMBER : Δοθέντος ενός γραφήματος $G = (V, E)$ και ενός θετικού ακεραίου k, έχει το γράφημα κλίκα μεγέθους k;

Ισοδύναμα δίνονται οι παρακάτω ορισμοί [Hromkovic, 2001]:

MAX-CLIQUE.

Είσοδος	Ένα γράφημα $G = (V, E)$.
Περιορισμοί	$M(G) = \{S \subseteq V \mid \{(u, v) \mid u, v \in S, u \neq v\} \subseteq E\}$. το $M(G)$ περιέχει όλα τα πλήρη υπογραφήματα (κλίκες) του G, δηλαδή είναι το σύνολο όλων των δυνατών (feasible) λύσεων.
Συνάρτηση κόστους	$\forall S \in M(G), c(S, G) = card(S)$.
Στόχος	Μεγιστοποίηση της c.

CLIQUE-NUMBER.

Είσοδος	Ένα γράφημα $G = (V, E)$ και ένας θετικός ακέραιος k.
Έξοδος	«ΝΑΙ» εάν το G περιέχει κλίκα μεγέθους k. «ΟΧΙ» διαφορετικά.

Το MAX-CLIQUE επελέγη σαν Πρόβλημα - πιλότος προκειμένου να δοκιμαστεί το προτεινόμενο Κ.Σ.Ε., και μάλιστα κυρίως αυτό της εύρεσης της μέγιστης κλίκας σε απροσανατόλιστο τυχαίο γράφημα. Οι λόγοι που οδήγησαν σ' αυτή την επιλογή μπορούν να συνοψιστούν στα παρακάτω:

1. Το CLIQUE-NUMBER και το ισοδύναμό του MAX-CLIQUE λόγω της γενικότητάς τους βρίσκουν εφαρμογή σε ένα μεγάλο φάσμα από τομείς της επιστήμης [DIMACS, 1992] [Bomze, Budinich +, 1999] ορισμένοι από τους οποίους παρατίθενται στον Πίνακα 12 που ακολουθεί. Έτσι, η επίλυσή του έχει συγκεντρώσει τις ερευνητικές προσπάθειες, ενώ δεν λείπουν ακόμα και οι προτάσεις αξιοποίησης της ιδιαίτερης δυσκολίας του σε τομείς όπως η Κρυπτογραφία [Juels, Peinado, 1998].

Πίνακας 12 – Εφαρμογές MAX-CLIQUE
Κατηγοριοποίηση – Ταξινόμηση σε κλάσεις ομοιότητας
Ανάκτηση ομοίων δεδομένων
Θεωρία Κωδικοποίησης
Αναγνώριση – Ταύτιση Προτύπων
Διάγνωση Σφαλμάτων
Σχεδιασμός Ηλεκτρονικών Κυκλωμάτων
Τεχνητή όραση
Οικονομία
Γεωμετρία
Βιολογία
Αρχαιολογία

2. Στην [Karp, 1972] ανάμεσα σε εικοσιένα συνολικά προβλήματα, αποδείχτηκε για πρώτη φορά ότι και το Πρόβλημα **CLIQUE-NUMBER** είναι **NP-Complete**, ενώ οι [Aho, Hopcroft, Ullman, 1974] αναφέρουν: «Μέχρι σήμερα δεν είναι γνωστός κανένας τρόπος που να λύνει το clique number problem σε πολυωνυμικό χρόνο». Η πρόταση αυτή εξακολουθεί να είναι ισχυρή για το CLIQUE-NUMBER που μάλιστα κατά τον Harary είναι το **πιο σκληρό μέλος της οικογένειας** των NP-Complete προβλημάτων [Harary, 1972].

3. Υπάρχει σχετικά πρόσφατη εργασία παράλληλης επίλυσης του MAX-CLIQUE [Pardalos, Rappe, Resende, 1999] με καταγεγραμμένες μετρήσεις που χρησιμοποιήθηκαν για σύγκριση σε επόμενο Κεφάλαιο της παρούσης.

3.1.2 Μέθοδοι επίλυσης

Στις μεθόδους εύρεσης **όλων των Maximal Cliques** ενός γραφήματος (άρα και την μέγιστη καθώς και τον πληθάριθμό της) περιλαμβάνεται η [Harary, Ross, 1957] που είναι από τις πρώτες στην βιβλιογραφία.

Επίσης, στηριζόμενοι στην μέθοδο της Αφαίρεσης Σημείου (vertex sequence, point removal) είναι οι αλγόριθμοι των Bonner αλλά και Bierstone [Auguston, Minker, 1970], όπου οι κλίκες του γραφήματος G παράγονται από αυτές του $G - \{u\}, u \in V$.

Στην μέθοδο backtracking στηρίζεται ο αλγόριθμος 457 της ACM [Bron, Kerbosch, 1973] ενώ στην [Loukakis, Tsouros, 1981] προτάθηκε ένας depth-first αλγόριθμος εύρεσης λεξικογραφικά, όλων των maximal ανεξαρτήτων υποσυνόλων ενός γραφήματος, πρόβλημα συγγενές με το MAX-CLIQUE (Θεώρημα 1.2.4.1).

Δύο διαφορετικοί αλγόριθμοι με παρόμοια συμπεράσματα όσον αφορά την πολυπλοκότητα τους, προτείνονται από τις [Tsukiyiama, Ide, +, 1977] και [Panayiotopoulos J-C., 1981] με την δεύτερη να στηρίζεται σε απλή boolean εξίσωση ενώ η πρώτη συνδυάζει προηγούμενες προσεγγίσεις των [Auguston, Minker, 1970] και [Bron, Kerbosch, 1973].

Εάν το πρόβλημα περιοριστεί στην εύρεση μίας μέγιστης κλίκας ή του μεγέθους της, τότε δύο είναι οι βασικοί άξονες στους οποίους στηρίζεται η προσπάθεια αντιμετώπισης του CLIQUE-NUMBER. Έξυπνες **παραλλαγές πλήρους απαρίθμησης** και **ευρετικές μέθοδοι**.

Οι πρώτες (**Implicit enumerative algorithms**) εγγυώνται την βέλτιστη ποιότητα της λύσης που παράγουν με εκθετικό χρόνο βέβαια στην χειρότερη περίπτωση. Στηρίζονται στην παρατήρηση ότι εφόσον γνωρίζει κάποιος μία κλίκα έστω μεγέθους k μπορεί να περιορίσει τον χώρο δυνατών λύσεων (search space) απορρίπτοντας όλες εκείνες που δεν μπορούν να παράγουν κλίκα με πληθάριθμο μεγαλύτερο του k.

Η συνηθέστερη μέθοδος εδώ είναι η branch and bound που στο CLIQUE-NUMBER έχει σημεία - κλειδιά το κριτήριο σπασίματος σε υποπροβλήματα (branch) και την εύρεση καλού κάτω ή / και άνω φράγματος για το μέγεθος της κλίκας (bound), που συνήθως γίνεται με κάποια ευρετική μέθοδο.

Από τους παλαιότερους τέτοιους αλγόριθμους αναφέρονται οι [Tarjan, 1972] και [Tarjan, Trojanowski, 1977] με πολυπλοκότητα $O(2^{n/3})$ ενώ βασισμένη σε έρευνα δένδρου (tree search) και μεγιστοποίηση boolean συνάρτησης είναι η [Loukakis, Tsouros, 1982].

Στην [Balas, Yu, 1986] βρίσκεται μία μέγιστη κλίκα ενός υπογραφήματος και στη συνέχεια ευρετικά το υπογράφημα αυτό επεκτείνεται έτσι ώστε να μην περιέχει άλλη μεγαλύτερη κλίκα. Με τον τρόπο αυτό μειώνεται ο αριθμός των υποπροβλημάτων που προκύπτουν άρα και ο χώρος των δυνατών λύσεων. Η [Robson, 1986] αποτελεί βελτίωση της [Tarjan, Trojanowski, 1977] με πολυπλοκότητα $O(2^{0.276n})$, το καλύτερο θεωρητικό φράγμα, χωρίς να είναι γνωστά αποτελέσματα δοκιμών της μεθόδου.

Στην [Carraghan, Pardalos, 1990] αρχικά ερευνάται ολόκληρο το γράφημα G σε σχέση με τον πρώτο κόμβο v_1 και βρίσκεται η μεγαλύτερη κλίκα που περιέχει τον κόμβο αυτό, ο οποίος στη συνέχεια διαγράφεται και ερευνάται το γράφημα

$G - \{u_1\}$ σε σχέση με τον δεύτερο κόμβο u_2. Η διαδικασία συνεχίζεται μέχρι να μην μπορεί να βρεθεί μεγαλύτερη κλίκα.

Στην [Babel, Tinhofer, 1990] χρησιμοποιείται η Brelaz heuristic για εύρεση φράγματος στο μέγεθος της κλίκας.

Οι **Ευρετικές μέθοδοι**, χωρίς να παρέχουν καμμία απόδειξη για την ποιότητα της λύσης έχουν πλεονέκτημα τον γενικά πολυωνυμικό χρόνο που απαιτούν.

Οι Άπληστες Σειριακές (Sequential Greedy Heuristics) [Pellilo, 2001] παράγουν κλίκες είτε προσθέτοντας κόμβους σε μία μικρότερη κλίκα ή αφαιρώντας κόμβους από ένα σύνολο που δεν είναι κλίκα μέχρι να γίνει.

Οι Ευρετικές τοπικής έρευνας (Local Search Heuristics) [Pellilo, 2001] βελτιώνουν τις προηγούμενες ψάχνοντας σε γειτονικούς (με διάφορους τρόπους οριζόμενους) κόμβους για να βελτιώσουν την κλίκα που έχει βρεθεί μέχρι στιγμής, ενώ πολλές από αυτές συνδυάζονται με τυχαιοποιημένους αλγορίθμους.

Οι Ευρετικές ανωτέρου επιπέδου (Advanced Search Heuristics) περιλαμβάνουν Simulated Annealing [Bomze, Budinich +, 1999], Νευρωνικά Δίκτυα (Neural Networks) [Smith, 1999], Genetic Algorithms [Bomze, Budinich +, 1999] , Tabu search [Soriano, Gendreau, 1994], GRASP [Festa, Resende, 2001].

Ένα άλλο σημείο που αξίζει αναφοράς είναι αυτό της έλλειψης χώρου (κύριας μνήμης) που παρουσιάζεται σε πολύ μεγάλα γραφήματα [Corno, Prinetto +, 1995] [Homer, Peinado, 1996]. Στην [Abello, Pardalos, Resende, 1999] αντιμετωπίζεται γράφημα με 54.000.000 κόμβους και πάνω από 170 εκατομμύρια ακμές που προέρχεται από πραγματικό πρόβλημα (Τηλεπικοινωνίες). Το θετικό σ' αυτήν την

περίπτωση είναι ότι η πυκνότητα τέτοιων γραφημάτων είναι κατά πολύ μικρότερη της μονάδος ($\simeq 1{,}166 \cdot 10^{-7}$).

Από πλευράς κατανεμημένης επίλυσης, στην [Panayiotopoulos J-C., 1989] προτείνεται μία μέθοδος υπολογισμού της μέγιστης κλίκας μέσω ενός δικτύου μικροϋπολογιστών, ενώ στηριγμένος στην [Carraghan, Pardalos, 1990] είναι ο παράλληλος αλγόριθμος ακριβούς επίλυσης του προβλήματος μέγιστης κλίκας της [Pardalos, Rappe, Resende, 1999].

Στην τελευταία, χρησιμοποιήθηκε ένα δίκτυο από μηχανές 64-bit Workstations που αντάλλασσαν μηνύματα μέσω MPICH (υλοποίηση του MPI) επιτυγχάνοντας έτσι τον παραλληλισμό, ενώ όλοι οι κώδικες ήταν γραμμένοι σε Fortran 77. Η εύρεση καλής αρχικής τιμής (LowBound) γινόταν με την μέθοδο GRASP [Feo, Resende, 1995] που επιτυγχάνει εξαιρετικές αποδόσεις. Μία κεντρική διαδικασία μοίραζε εργασίες στις υπόλοιπες (μία ή τρείς) που εκτελούσαν όλο το υπολογιστικό έργο.

3.2 Κατανεμημένη Επίλυση CLIQUE-NUMBER

3.2.1 Μέθοδος Διάσπασης & Επίλυσης

Θεωρούμε την πεπερασμένη ακολουθία συνόλων V^i, όπου

$$V^i = \{v_k \in V, \forall k > i, (v_i, v_k) \in E\} \subset V,\ i = 1,2,...,n-1 \qquad (1)$$

Κάθε στοιχείο της ακολουθίας ορίζει ένα παραγόμενο υπογράφημα του G, έστω $G(V^i)$. Είναι προφανές ότι κάθε τέτοιο παραγόμενο υπογράφημα $G(V^i)$ αποτελείται από τους κόμβους εκείνους που είναι λεξικογραφικά μεγαλύτεροι από τον κόμβο v_i και υπάρχει ακμή που τους συνδέει με αυτόν.

Έστω C μία μέγιστη κλίκα του G, όπου

$$|C| = CN(G) = \lambda$$

$$C = \{v_{i_1}, v_{i_2}, ..., v_{i_\lambda}\}.$$

Θεωρούμε, χωρίς βλάβη της γενικότητας ότι ισχύει $i_1 < i_2 < ... < i_\lambda$

Έστω ότι υπάρχει ένας κόμβος u έτσι ώστε $u \in C - \{v_{i_1}\}$ και $u \notin V^{i_1}$. Επειδή $u \in C$, και C είναι κλίκα του G που περιέχει και τον κόμβο v_{i_1}, θα ισχύει ότι οι κόμβοι u και v_{i_1} συνδέονται με ακμή του Ε, το οποίο λόγω της (1) σημαίνει $u \in V^{i_1}$ που είναι άτοπο. Επομένως

$$C - \{v_{i_1}\} \subseteq V^{i_1} \text{ (i)}$$

Αφού το C είναι κλίκα του G, λόγω της (i) το σύνολο $C - \{v_{i_1}\}$ που έχει πληθάριθμο $\lambda - 1$ θα είναι κλίκα του $G(V^i)$. Η κλίκα αυτή είναι μέγιστη, διότι εάν

υπήρχε μεγαλύτερη κλίκα C' πληθάριθμου έστω λ, τότε η μέγιστη κλίκα του γραφήματος G θα είναι το σύνολο $C' \cup \{v_{i_1}\}$ με πληθάριθμο $\lambda+1$, που είναι άτοπο διότι έχουμε υποθέσει ότι η μέγιστη κλίκα του G έχει πληθικό αριθμό λ. Επομένως

$C - \{v_{i_1}\}$ μέγιστη κλίκα του $G(V^{i_1})$ (ii)

Από τα συμπεράσματα (i) και (ii), εύκολα συνάγονται τα παρακάτω:

Θεώρημα 3.2.1.1 Το σύνολο $C - \{v_{i_1}\}$ αποτελεί μέγιστη κλίκα του παραγόμενου υπογραφήματος $G(V^{i_1})$.

Πόρισμα 3.2.1.1 Η μέγιστη κλίκα ενός γραφήματος G περιέχεται σε ένα τουλάχιστον σύνολο της μορφής $V^i \cup \{v_i\}$.

Ο κόμβος v_i ονομάζεται **pivot-node** και είναι προφανές ότι υπάρχει ένας τέτοιος κόμβος σε κάθε maximal κλίκα του G, και επομένως και στη μέγιστη.

Στην παραπάνω ανάλυση χρησιμοποιήθηκε η λεξικογραφική ταξινόμιση προκειμένου να διαφυλαχθεί η γενικότητα της μεθόδου διάσπασης. Το γράφημα που εξετάζεται μπορεί να είναι είτε το αρχικό, ή κάποιο ισοδύναμο που έχει προκύψει με οποιαδήποτε αντιμετάθεση των κόμβων με βάση ένα **οποιοδήποτε κριτήριο** που ορίζει μία διάταξη των κόμβων.

Είναι βέβαια προφανές ότι αν είναι γνωστό πως το γράφημα G περιέχει κλίκα μεγέθους $m \leq \lambda$, τότε κάθε υπογράφημα $G(V^i)$ με $|V^i| < m$ δεν μπορεί να περιέχει κλίκα μεγαλύτερου πληθαρίθμου και επομένως το εν λόγω γράφημα μπορεί να αγνοηθεί από κάθε περαιτέρω επεξεργασία (κανόνας απόρριψης).

Πόρισμα 3.2.1.2 Κάθε παραγόμενο υπογράφημα έχει μέγεθος μικρότερο από το πατρικό και είναι προγραμματιστικά ανεξάρτητο από τη μέθοδο υπολογισμού της μέγιστης κλίκας.

Από τα παραπάνω είναι φανερό ότι η μέθοδος διάσπασης που περιγράφηκε παράγει υποπροβλήματα που αφενός τηρούν τις προϋποθέσεις Κλειστής Ανάλυσης της παραγράφου 2.2, αφετέρου η λύση δεν εξαρτάται από παράγοντες όπως το Λειτουργικό Σύστημα αλλά και τη μέθοδο επίλυσης που χρησιμοποιεί κάθε Problem Solver για τον υπολογισμό της κλίκας.

Στην παρούσα εργασία, για την τελική, ακριβή επίλυση των υπογραφημάτων που παράγονται αναπτύχθηκε μία παραλλαγή της μεθόδου που περιγράφεται στην [Panayiotopoulos J-C., 1981] όπου είχαν αποδειχτεί τα ακόλουθα Λήμματα :

Λήμμα 3.2.1.1 Το $C \subseteq V$ είναι maximal κλίκα του G, αν και μόνον αν

$$\prod_r (\overline{v_r} + \prod_p \overline{v_p}) = 1,\ r \in B,\ p \in Z_r \qquad \text{(L1)}$$

$$I = \{1, 2, \ldots, n\}$$
$$B = \{r \in I : \sum_j \overline{a_{rj}} > 0, r < j, j \in I\}$$
$$Z_r = \{p \in I : p > r, a_{rp} = 0\}$$

(a_{ij}) είναι ο πίνακας συσχέτισης (adjacency matrix) του G.

Κάνοντας Boolean πράξεις και απορροφήσεις στην (L1) καταλήγουμε στην ισοδύναμη εξίσωση:

$$\sum \overline{v_{i_1}}\ \overline{v_{i_2}}\ \ldots \overline{v_{i_m}} = 1,\ i_1 < i_2 < \ldots < i_m \in I \quad (L2)$$

Λήμμα 3.2.1.2 Κάθε όρος στο αριστερό μέλος της εξίσωσης (L2) είναι μία maximal κλίκα του G ενώ ο όρος με την ελάχιστη τιμή για το m δίνει τη μέγιστη κλίκα $C^* = V - \{v_{i_1}, v_{i_2}, \ldots, v_{i_m}\}$ με πληθάριθμο $n-m$.

Για την εκτέλεση των Boolean πράξεων, δεν ακολουθείται η μέθοδος υλοποίησης του παραπάνω αλγορίθμου που είχε προταθεί, αλλά μία νέα μέθοδος που στηρίζεται στην ανάλυση που ακολουθεί:

Πόρισμα 3.2.1.3 Η Boolean μέθοδος του Λήμματος 3.2.1.2 βελτιστοιποιείται από πλευράς αναμενόμενου υπολογιστικού χρόνου, όταν το πλήθος των κόμβων n του G είναι μικρότερο ή ίσο από το πλήθος των bits της λέξης της μηχανής που χρησιμοποιείται.

Έτσι, κάθε όρος στις παρενθέσεις στο πρώτο μέλος της (L1) παριστάνεται σαν ένας 32-bit ακέραιος, ενώ οι πράξεις του πολλαπλασιασμού και οι απορροφήσεις γίνονται με χρήση των ψηφιακών τελεστών (bitwise operators) και ψηφιακών τελεστών ολίσθησης (bitwise shift operators) που διαθέτουν οι σύγχρονες γλώσσες προγραμματισμού (π.χ. Visual C++).

Για παράδειγμα, έστω ότι δύο από τις παρενθέσεις της (L1) είναι οι

$$\overline{v_4} + \overline{v_6 v_7} \quad \text{και} \quad \overline{v_5} + \overline{v_7}$$

Ο όρος $\overline{v_4}$ παριστάνεται με τον δεκαδικό αριθμό

$$d1=(00000000\ 00000000\ 00000000\ 00001000)_{(2)} = 1_{(10)} << 3$$

ενώ ο $\overline{v_5}$ με τον δεκαδικό

$$d2=(00000000\ 00000000\ 00000000\ 00010000)\ (2) = 1(10) << 4$$

(με << συμβολίζεται ο τελεστής αριστερής ολίσθησης).

Ο Boolean πολλαπλασιασμός $\overline{v_4 v_5}$ γίνεται με χρήση του ψηφιακού τελεστή OR | :

$$\overline{v_4 v_5} = d1 \mid d2 = (00000000\ 00000000\ 00000000\ 00011000)\ (2)$$

Με δεδομένο ότι οι τελεστές ολίσθησης είναι αρκετά ταχύτεροι από οποιονδήποτε άλλο αριθμητικό τελεστή πράξης, είναι προφανές το κέρδος των πράξεων αυτών. Στις μηχανές PC όπου το μήκος λέξης είναι 32 bit, οι boolean πράξεις για γραφήματα μεγέθους μικρότερου ή ίσου των 32 κόμβων γίνονται σε χρόνο πρακτικά ακαριαίο, που είναι και ο λόγος επιλογής της παραπάνω μεθόδου για το Σύστημα Επίλυσης Υποπροβλήματος.

Έχοντας ορίσει την μέθοδο διάσπασης του αρχικού προβλήματος σε υποπροβλήματα καθώς και την μέθοδο (ακριβούς) επίλυσης υποπροβλήματος, στην επόμενη παράγραφο περιγράφεται η προσαρμογή του Κ.Σ.Ε. στο πρόβλημα MAX-CLIQUE.

3.2.2 Προσαρμογή Κ.Σ.Ε.

Με την εκκίνηση του Συστήματος, ο Διαχειριστής Προβλήματος (PM) δέχεται σαν είσοδο εκτός από το πλήθος των Επιλυτών d, την τοποθεσία όπου βρίσκεται το αρχείο του προβλήματος Prob_file. Το αρχείο αυτό περιέχει τον πίνακα συσχέτισης A του γραφήματος (adjacency matrix) σε τριγωνική μορφή για λόγους οικονομίας χώρου με διάταξη που φαίνεται στον Πίνακα 13.

Πίνακας 13 – Αρχείο γραφήματος Prob_file

Αριθμός Γραμμής	Περιεχόμενο
1	Size
2	A(2,1)
3	A(3,1) A(3,2)
.	.
.	.
.	.
size	A(size,1) A(size, 2) . . . A(size -1, size - 1)

Size: ακέραιος, το μέγεθος του γραφήματος
A(i, j) = 0 ή 1, ο adjacency matrix
A(i, i) = 0 για κάθε i∈ {1,2,..,size}

Η παραπάνω παράσταση σε **αρχείο κειμένου** προτιμήθηκε από την αντίστοιχη, επίσης σε αρχείο κειμένου, του [DIMACS, 1993] διότι δημιουργεί αρχεία μικρότερου μεγέθους (Πίνακας 14).

Πίνακας 14 – Σύγκριση format Prob_file

Γράφημα	Μέγεθος αρχείου κειμένου σε Bytes	
	DIMACS	Adjacency Matrix
Keller4	92.001	15.053
Hamming6-2	16.168	2.212
Hamming8-4	212.167	33.413

Όπως αναφέρθηκε στην παράγραφο 3.2.1, το γράφημα που θα επιλυθεί από πλευράς Clique Number μπορεί να είναι είτε το αρχικό, ή κάποιο ισοδύναμο που θα προκύψει αντιμεταθέτοντας τους κόμβους με βάση ένα οποιοδήποτε κριτήριο που ορίζει μία διάταξη των κόμβων. Στην παρούσα διατριβή, το κριτήριο που επιλέχθηκε είναι αυτό της **διάταξης με αύξουσα σειρά ως προς το βαθμό των κόμβων**.

Έτσι, ο PM αφού φορτώσει το γράφημα στην τοπική του μνήμη, το ταξινομεί παράγοντας ένα ισοδύναμο του αρχικού με κόμβους έστω $V = \{v_1, v_2, ..., v_i, ..., v_n\}$ για τους οποίους ισχύει

$$\deg(v_1) \le \deg(v_2) \le ... \deg(v_i) \le ... \le \deg(v_n) .$$

Είναι προφανές ότι η μεθόδος διάσπασης ξεκινώντας από το αρχικό γράφημα μπορεί να εφαρμοστεί αναδρομικά στο εκάστοτε παραγόμενο υπογράφημα, με έναν pivot-node να προκύπτει σε κάθε μία τέτοια εφαρμογή. Οι κόμβοι αυτοί συνιστούν την χαρακτηριστική ακολουθία του τελικού υποπροβλήματος και αποτελούν την ταυτότητά του μέσω της οποίας μπορεί να αναπαραχθεί από τον PS.

Για παράδειγμα έστω το γράφημα G με μέγεθος $n = 10$ και πίνακα συσχέτισης A όπως παρακάτω:

$$A = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} \\ v_1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ v_2 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ v_3 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ v_4 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ v_5 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ v_6 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ v_7 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ v_8 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ v_9 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ v_{10} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Αναδιατάσσοντας τους κόμβους με αύξουσα σειρά βαθμού, προκύπτει το παρακάτω, ισοδύναμο με το αρχικό, γράφημα:

$$\Pi_1^0 = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} \\ v_1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ v_2 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ v_3 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ v_4 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ v_5 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ v_6 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ v_7 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ v_8 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ v_9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ v_{10} & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Έχοντας για παράδειγμα pivot-node τον κόμβο v_4 προκύπτει το υπογράφημα (υποπρόβλημα) με κόμβους τους $v_5, v_7, v_8, v_9, v_{10}$ που μετά από επαναρίθμηση αντιστοιχούν στον πίνακα:

$$\Pi_4^1 = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 & v_5 \\ v_1 & 0 & 0 & 1 & 1 & 0 \\ v_2 & 0 & 0 & 1 & 1 & 1 \\ v_3 & 1 & 1 & 0 & 0 & 1 \\ v_4 & 1 & 1 & 0 & 0 & 1 \\ v_5 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Παρόμοια, στο νέο (υπο)γράφημα, με pivot-node π.χ. τον v_2 προκύπτει το υποπρόβλημα 2ου επιπέδου που αποτελείται από τους κόμβους v_3, v_4, v_5 :

$$\Pi_2^2 = \begin{pmatrix} & v_1 & v_2 & v_3 \\ v_1 & 0 & 0 & 1 \\ v_2 & 0 & 0 & 1 \\ v_3 & 1 & 1 & 0 \end{pmatrix}$$

που έχει χαρακτηριστική ακολουθία την $\varphi^1 = 4, \varphi^2 = 2$. Ένας Επιλυτής που γνωρίζει το αρχικό πρόβλημα καθώς και την παραπάνω χαρακτηριστική ακολουθία, μπορεί να κατασκευάσει το υποπρόβλημα.

Αν τώρα υποθέσουμε ότι το υποπρόβλημα Π_2^2 περιέχει κλίκα μεγέθους k έστω C', τότε το αρχικό πρόβλημα Π θα περιέχει κλίκα μεγέθους $k+2$ που θα αποτελείται από τους κόμβους του C' και τους δύο κόμβους που προσδιορίζονται από τις τιμές της χαρακτηριστικής ακολουθίας $\varphi^i,\ i = 1,2$.

Η επιλογή της Boolean μεθόδου για ακριβή επίλυση των τελικών υποπροβλημάτων και η υλοποίησή της που περιγράφηκαν στην παράγραφο 3.2, αποδίδει μεγάλη ταχύτητα επεξεργασίας σε μικρά γραφήματα ενώ κάνει αναγκαστική την εμφάνιση μιάς ακόμα παραμέτρου του $Word\_Length$. Πρόκειται για το μήκος λέξης της μηχανής όπου εκτελείται το πρόγραμμα, για παράδειγμα σε μηχανές PC των 32 bits θα είναι $Word\_Length = 32$.

Με βάση το παραπάνω, αλλά κυρίως προκειμένου να μελετηθεί η απόδοση του Κ.Σ.Ε. σε

- σύγκριση με παλαιότερες εργασίες
- διαφορετικές συνθέσεις από πλευράς πλήθους Επιλυτών
- διαφορετικές κατανομές φόρτων εργασίας Επιλυτών – Διαχειριστή

το σύστημα διάσπασης Προβλήματος και διανομής των Υποπροβλημάτων, διαμορφώθηκε ως εξής:

- $Critical\_Level = 1$. Ο PM διασπά το αρχικό Πρόβλημα σε υποπροβλήματα πρώτου επιπέδου.
- Σταθερό κάτω φράγμα η παράμετρος $Word\_Length$. Όλα τα υποπροβλήματα μεγέθους μικρότερου του $Word\_Length$ καταχωρούνται στην τοπική λίστα S_{ij} του PM για τοπική επίλυση.
- Το φράγμα $Critical\_Low$ σε επόμενο επίπεδο κάτω φράγματος μεταβλητής τιμής προκειμένου να δοκιμαστεί η στάθμιση του φόρτου εργασίας.
- Σε περίπτωση που το μέγεθος ενός προβλήματος ξεπερνά το $Critical\_High$ η επίλυση παύει να είναι ακριβής και γίνεται ευρετική, όπως θα δούμε στη συνέχεια της παρούσης Διατριβής.

Η εγγραφή του Comm_file, κεντρικού αρχείου του υποσυστήματος επικοινωνίας έχει μέλη:

> SenderID: ακέραιος $\in [0, d]$.
>
> Node: ακέραιος $\in [1, size)$ ο pivot-node του υποπροβλήματος.
>
> Bound: μέγεθος της γνωστής μέγιστης κλίκας.

ενώ τα αρχεία Res_file_i είναι κειμένου και περιέχουν τους κόμβους που συνιστούν την μέγιστη κλίκα σε free-format.

Κάθε γράφημα διασπάται από τον PM σε υπογραφήματα πρώτου επιπέδου. Στη συνέχεια κάθε υπογράφημα ανάλογα με τις παραμέτρους του υφίστανται την κατάλληλη αντιμετώπιση που μπορεί να είναι

- τοπική επίλυση
- αποστολή σε Επιλυτή
- απόρριψη

Στις δύο πρώτες περιπτώσεις, η διάσπαση συνεχίζεται από το σύστημα επίλυσης υποπροβλήματος μέχρις ότου τα προκύπτοντα υπογραφήματα να έχουν μέγεθος μικρότερο από $Word\_Length$ οπότε παραδίδονται στη Boolean function για να επιλυθούν ακριβώς από πλευράς μέγιστης κλίκας:

1. Δημιουργούνται οι παράγοντες, έστω $TotalFactors$ το πλήθος, του πρώτου μέλους της σχέσης

$$\prod_r (\overline{v_r} + \prod_p \overline{v_p}) = 1,\ r \in B,\ p \in Z_r$$

(Λήμμα 3.2.1.1) και αποθηκεύονται σε κατάλληλο πίνακα ακεραίων $Factors[first][\sec ond]$ όπου $first = 1,..,TotalFactors$ και $\sec ond = 1,2$, ενώ η μέγιστη τιμή που μπορεί να πάρει ο $first$ είναι βέβαια 32.

2. Οι παράγοντες (ο πίνακας $Factors$) ταξινομούνται με φθίνουσα σειρά ως προς το πλήθος των όρων που περιέχει ο δεύτερος προσθετέος $\prod_p \overline{v_p}$. Το πλήθος αυτό, για την τυχαία i-στή παρένθεση (παράγοντα) ισούται με το πλήθος των bits του ακεραίου $Factors[i][2]$ τα οποία βρίσκονται στην κατάσταση 1.

3. Η συνάρτηση χρησιμοποιεί έναν καταλλήλου μεγέθους πίνακα ακεραίων $BoolTable[terms]$ όπου κρατά τους όρους που προκύπτουν από τις boolean πράξεις. Οι πράξεις γίνονται με την μέθοδο που έχει περιγραφεί στην παράγραφο 3.2 και η διαχείριση του $BoolTable[terms]$ με τον τρόπο που φαίνεται στον Πίνακα 15.

4. Μετά το τέλος των boolean πράξεων χρησιμοποιείται το Λήμμα 3.2.1.2 για να βρεθεί η μέγιστη κλίκα του υπογραφήματος αυτού από τις απομένουσες γραμμές του πίνακα $BoolTable[terms]$.

Πίνακας 15 – Εκτέλεση Boolean Πράξεων

```
LastRow = 2
BoolTable[1] = factors[1][1]
BoolTable[2] = factors[1][2]
Απορρίψεις & απορροφήσεις γραμμών*
Για i = 1 έως TotalFactors εκτέλεσε
{
        HalfLines = LastRow
        LastRow = LastRow * 2
        Για Lines = HalfLines + 1 έως LastRow επανέλαβε
                BoolTable[Lines] = BoolTable[Lines – HalfLines]
        Πολλαπλασίασε τις πρώτες μισές γραμμές του boolean πίνακα με factors[i][1]
        Πολλαπλασίασε τις δεύτερες μισές γραμμές του boolean πίνακα με factors[i][2]
        Απορρίψεις & απορροφήσεις γραμμών*
}
Εύρεση Μέγιστης Κλίκας από τις γραμμές του BoolTable
```

(*) Οι γραμμές που απορρίπτονται είναι εκείνες που δεν έχουν ελπίδα να δώσουν μεγαλύτερο clique number από το ήδη γνωστό, ενώ η απορρόφηση στηρίζεται στην γνωστή ιδιότητα της Άλγεβρας Bool, $a + a \bullet b = a$.

Το αρχικό κάτω φράγμα εκκίνησης για την μέγιστη κλίκα LowBound (LB) υπολογίζεται ευρετικά. Είναι προφανές ότι για τον σκοπό αυτό μπορεί να χρησιμοποιηθεί οποιαδήποτε ευρετική μέθοδος. Ο αλγόριθμος της απλής μεθόδου που χρησιμοποιείται στην παρούσα παρατίθεται στον Πίνακα 16.

Πίνακας 16 – Ευρετική Μέθοδος LB

```
CN = 1
CliqueSet = {v_n}
Για i = n-1 έως 1 εκτέλεσε
{
        member = 1
        Για j = 1 έως CN εκτέλεσε
        {
                Εαν (v_j, v_i) ∉ E τότε
                {
                        member = 0
                        exit (j)
                }
        }
        Εαν member = 1 τότε
        {
                CN = CN + 1
                CliqueSet = CliqueSet U {v_i}
        }
}
```

Όπως θα δούμε μεταξύ άλλων σε επόμενο κεφάλαιο μετρήσεων, η τιμή LB επηρεάζει πολύ λιγότερο τον συνολικό χρόνο επίλυσης του Κ.Σ.Ε. απ' ότι για παράδειγμα σε μία μέθοδο όπως η Boolean. Το γεγονός αυτό οφείλεται στην διάσπαση σε υποπροβλήματα και κατανεμημένη επεξεργασία που κάνει την τιμή του LB να μεταβάλλεται (βελτιώνεται) **δυναμικά** αλλά και με **μεγάλη ταχύτητα**.

Κεφάλαιο 4

ΜΕΤΡΗΣΕΙΣ

ΜΕΤΡΗΣΕΙΣ

4.1 Περιβάλλον

Όπως έχει ήδη τονιστεί, κεντρικός στόχος της παρούσης διατριβής είναι η υλοποίηση ενός Συστήματος επίλυσης με περιορισμό των απαιτήσεων τόσο Υλικού όσο και Λογισμικού σε καταστάσεις που σήμερα συναντάμε σχεδόν σε οποιοδήποτε χώρο εργασίας, γραφείο, αλλά ακόμα και στο οικιακό περιβάλλον.

Για τον παραπάνω λόγο, όλες οι μετρήσεις που παρατίθενται στο παρόν κεφάλαιο έγιναν σε απλούς, κοινούς μικροϋπολογιστές συνδεδεμένους μέσω τοπικού δικτύου, με διαμόρφωση φθηνού (χαμηλού κόστους) Υλικού:

- Επεξεργαστής Pentium III 450 MHz
- Τοπικό Δίκτυο Ethernet 100 Mbit

ενώ από πλευράς Λογισμικού:

- Λειτουργικό Σύστημα MS-Windows
- Εφαρμογές Κ.Σ.Ε. σε Visual C++ της Microsoft.

4.2 Στάθμιση Παραμέτρων

Στην παρούσα παράγραφο εξετάζονται ειδικά θέματα που αφορούν στην επίδραση των παρακάτω παραμέτρων πάνω στον χρόνο επίλυσης:

1. Της ταξινόμησης στα υπογραφήματα που προκύπτουν κατά την διαδοχική διάσπαση.

2. Της τιμής του $Word\_Length$ που καθορίζει το μέγιστο μέγεθος γραφήματος που επιλύει η Boolean Function.

3. Της εκτέλεσης περισσοτέρων του ενός PS ανά υπολογιστή (επεξεργαστή).

4. Της αρχικής τιμής εκκίνησης για την Μέγιστη Κλίκα (LowBound).

5. Της μετατροπής της επίλυσης από ακριβούς σε ευρετική όταν το μέγεθος των προβλημάτων γίνεται μεγαλύτερο από κάποια τιμή $Critical\_High$ που ξεπερνά τις δυνατότητες του Συστήματος να το επιλύσει ακριβώς σε «λογικό» χρόνο.

Είναι προφανές ότι οι κόμβοι που έχουν μεγαλύτερο βαθμό (οι πυκνότεροι) έχουν μεγαλύτερη πιθανότητα να είναι pivot-nodes σε μεγάλες κλίκες και φυσικά και στην μέγιστη επομένως είναι σκόπιμο η επίλυση να ξεκινά από αυτές. Για τον λόγο αυτό γίνεται ταξινόμηση του αρχικού γραφήματος πριν την εκκίνηση της επίλυσης η οποία επιταχύνει σε πολύ μεγάλο βαθμό την εξεύρεση της λύσης. Όμως από τις μετρήσεις του Πίνακα 17 φαίνεται ότι εάν κατά τη διαδικασία διάσπασης **όλα τα υπογραφήματα υφίστανται ταξινόμηση** όπως το αρχικό, τότε οι συνολικοί χρόνοι επίλυσης βελτιώνονται σε σχέση με αυτούς που μετρήθηκαν χωρίς περαιτέρω ταξινόμηση.

Πίνακας 17 – Απόδοση Ταξινόμησης Υπογραφημάτων

A

Ταυτότητα Γραφήματος		Με Ταξινόμηση		Χωρίς Ταξινόμηση	
Κόμβοι	200	Χρόνος Επίλυσης (sec)	91,28	Χρόνος Επίλυσης (sec)	145,44
Ακμές	13.985	Ταξινομηθέντα Υπογρ.	64.201	Ταξινομηθέντα Υπογρ.	0
Πυκνότητα	0,7028	Κλήσεις Boolean Function	1.292.743	Κλήσεις Boolean Function	1.896.622
Κάτω Φράγμα	14	Μέση Πυκνότητα Γραφημάτων Boolean	0,81	Μέση Πυκνότητα Γραφημάτων Boolean	0,76
Μέγιστη Κλίκα	18				

Αναλυτική κατάσταση γραφημάτων που υποβλήθηκαν στην Boolean Function

Αριθμός Κόμβων	Αριθμός Γραφημάτων		Αριθμός Κόμβων	Αριθμός Γραφημάτων		Αριθμός Κόμβων	Αριθμός Γραφημάτων	
	Με Ταξινόμηση	Χωρίς Ταξινόμηση		Με Ταξινόμηση	Χωρίς Ταξινόμηση		Με Ταξινόμηση	Χωρίς Ταξινόμηση
12		57	19	106.572	152.976	26	54.177	80.442
13	500	3.015	20	116.657	150.906	27	36.620	65.915
14	4.839	55.692	21	127.828	146.345	28	23.108	53.674
15	83.884	137.177	22	127.113	137.053	29	14.538	43.496
16	93.476	147.056	23	113.439	126.050	30	9.848	34.601
17	96.102	151.717	24	94.008	111.558	31	8.166	27.818
18	100.754	152.734	25	73.822	96.271	32	7.292	22.069

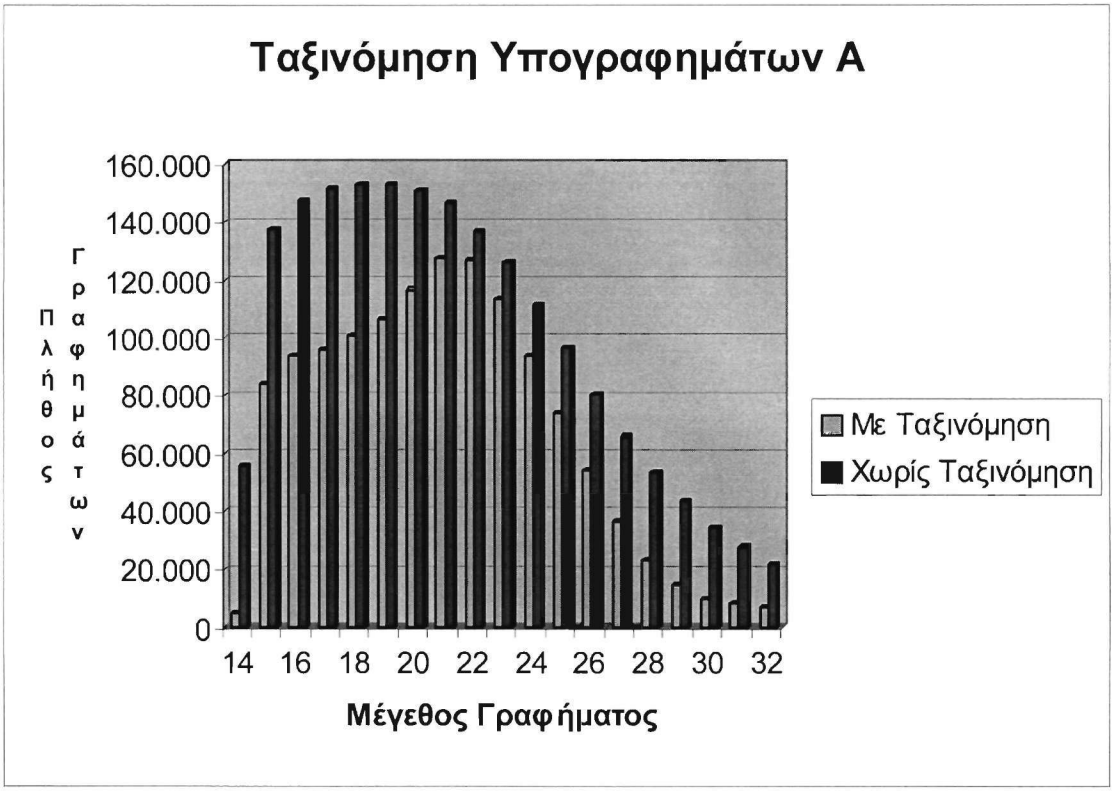
B

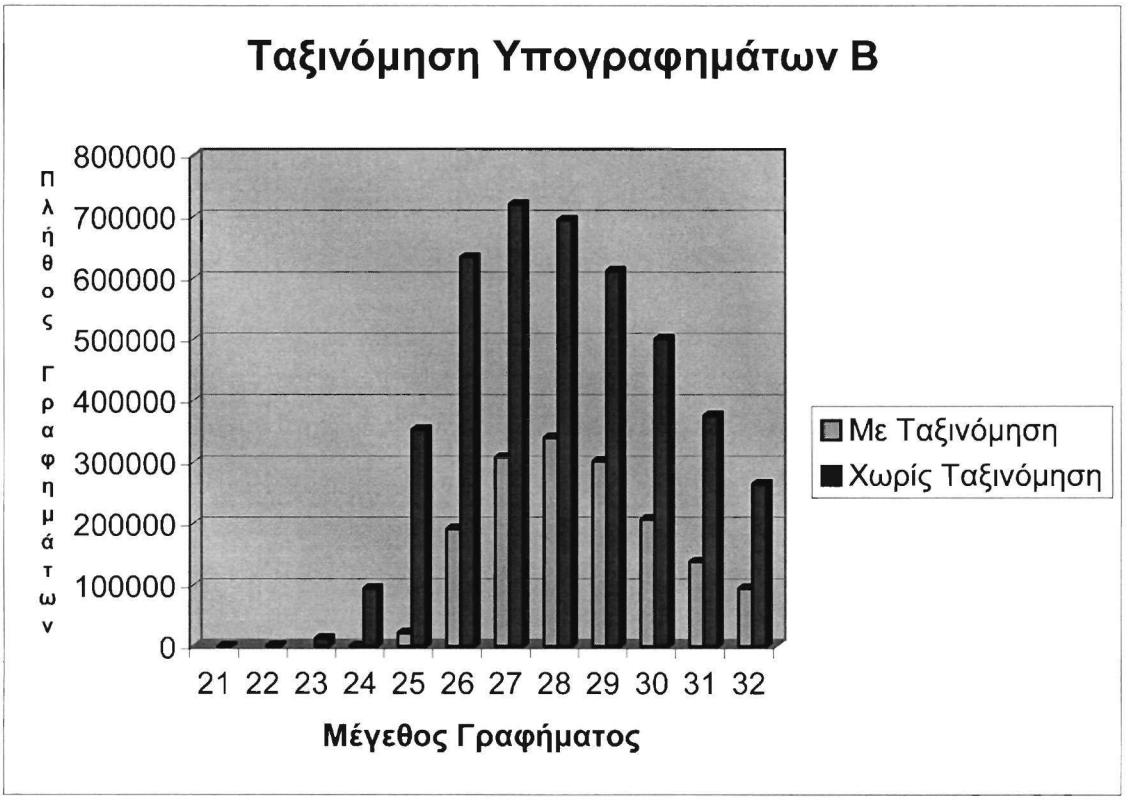
Ταυτότητα Γραφήματος		Με Ταξινόμηση		Χωρίς Ταξινόμηση	
Κόμβοι	100	Χρόνος Επίλυσης (sec)	93,98	Χρόνος Επίλυσης (sec)	227,06
Ακμές	4.465	Ταξινομηθέντα Υπογρ.	242.617	Ταξινομηθέντα Υπογρ.	0
Πυκνότητα	0,902	Κλήσεις Boolean Function	1.603.576	Κλήσεις Boolean Function	4.265.737
Κάτω Φράγμα	28	Μέση Πυκνότητα Γραφημάτων Boolean	0,96	Μέση Πυκνότητα Γραφημάτων Boolean	0,948
Μέγιστη Κλίκα	31				

Αναλυτική κατάσταση γραφημάτων που υποβλήθηκαν στην Boolean Function

Αριθμός Κόμβων	Αριθμός Γραφημάτων		Αριθμός Κόμβων	Αριθμός Γραφημάτων		Αριθμός Κόμβων	Αριθμός Γραφημάτων	
	Με Ταξινόμηση	Χωρίς Ταξινόμηση		Με Ταξινόμηση	Χωρίς Ταξινόμηση		Με Ταξινόμηση	Χωρίς Ταξινόμηση
21		13	25	21.882	352.940	29	301.434	612.215
22		1.015	26	191.448	634.226	30	207.311	500.937
23		13.531	27	308.229	720.651	31	137.861	376.168
24	252	94.783	28	340.377	695.190	32	94.782	264.068

Εικόνα 11 – Επίδραση Ταξινόμησης Στην Απόδοση*





(*) Πλήθος Γραφημάτων που στέλνονται στην Boolean function για επεξεργασία.

Η βέλτιστη τιμή για το $Word\_Length$ όπως φαίνεται στον Πίνακα 18 είναι 32 σε κάθε περίπτωση. Μάλιστα ενώ στις σχετικά μικρές πυκνότητες (Γραφήματα II και IV του Πίνακα) η διαφορά στην απόδοση είναι μικρή, στις περιπτώσεις όμως των γραφημάτων με μεγάλες πυκνότητες (I, III) παρατηρείται μέχρι και υποδιπλασιαμός του χρόνου μόλις γίνει $Word\_Length = 32$.

Πίνακας 18 – Επίδραση Word_Length

	Ταυτότητα Γραφήματος			
	I	II	III	IV
Word_Length	n = 100 e = 4.465 d = 0,902 LB = 28 CN = 31	n = 200 e = 11.843 d = 0,5951 LB = 12 CN = 14	n = 200 e = 13.900 d = 0,6985 LB = 13 CN = 18	n = 400 e = 39.717 d = 0,4977 LB = 10 CN = 12
	Χρόνος (sec)			
10	182,08	7,19	116,11	59,75
20	181,97	7,47	114,35	62,84
28	150,78	6,59	84,37	58,45
30	121,88	6,59	82,34	58,17
32	94,47	6,59	81,73	58,11

Η απόδοση του Κ.Σ.Ε. εξαρτάται αποκλειστικά από το **πλήθος** των **μικροϋπολογιστών** (επεξεργαστών) που το απαρτίζουν και σε καμμία περίπτωση από το πλήθος των Επιλυτών που τρέχουν εφόσον δεν αντιστοιχούν σε διαφορετική μηχανή. Στον Πίνακα 19 βλέπουμε πως όταν αυξάνεται το πλήθος Επιλυτών ανά υπολογιστή σε 2 PS ανά PC, η απόδοση υφίσταται μικρή μείωση.

Πίνακας 19 – Κ.Σ.Ε. & Ισχύς Επεξεργαστή					
Κόμβοι 200 Ακμές 13.940 Πυκνότητα 0,7005 Κάτω Φράγμα 14 Μέγιστη Κλίκα 18		Κόμβοι 400 Ακμές 47.817 Πυκνότητα 0,5992 Κάτω Φράγμα 12 Μέγιστη Κλίκα 16		Κόμβοι 1000 Ακμές 200.014 Πυκνότητα 0,4004 Κάτω Φράγμα 8 Μέγιστη Κλίκα 12	
1 PS / PC	2 PS / PC	1 PS / PC	2 PS / PC	1 PS / PC	2 PS / PC
9.28 sec	9.94 sec	97.82 sec	98.7 sec	97.72 sec	99.75 sec
Σε όλες τις παραπάνω μετρήσεις χρησιμοποιήθηκαν εννέα μικροϋπολογιστές (επεξεργαστές) στους οποίους έτρεχε ένας ή δύο Επιλυτές καθώς και ένας ακόμη μικροϋπολογιστής όπου έτρεχε ο Problem Manager.					

Όπως έχει ήδη αναφερθεί, στο πρόβλημα εύρεσης της μέγιστης κλίκας το κριτήριο απόρριψης τμημάτων του χώρου λύσεων που δεν έχουν πιθανότητα να δώσουν καλύτερη λύση παρέχεται από το αρχικό κάτω φράγμα. Ωστόσο, ενώ στην περίπτωση της Boolean function το LowBound παραμένει σταθερό καθόλη τη διάρκεια της εκτέλεσης των πράξεων, στο Κ.Σ.Ε. μεταβάλλεται βελτιούμενο δυναμικά όσο προχωρεί η διαδικασία επίλυσης. Για τον λόγο αυτό η εύρεση ενός ιδιαίτερα καλού LowBound πριν αρχίσει η επίλυση δεν επηρεάζει ιδιαίτερα τον χρόνο που συνολικά χρειάζεται το Σύστημα όπως άλλωστε επιβεβαιώνεται και από τις μετρήσεις του Πίνακα 20.

Πίνακας 20 – Κ.Σ.Ε. & LowBound Εκκίνησης					
Κόμβοι 200 Ακμές 13.940 Πυκνότητα 0,7005 Μέγιστη Κλίκα 18		Κόμβοι 400 Ακμές 47.817 Πυκνότητα 0,5992 Μέγιστη Κλίκα 16		Κόμβοι 1.000 Ακμές 149.690 Πυκνότητα 0,2997 Μέγιστη Κλίκα 9	
LB = 14	LB = 2	LB = 12	LB = 2	LB = 7	LB = 2
25.04 sec	25.1 sec	282.04 sec	283.19 sec	22.08 sec	22.19 sec
Οι παραπάνω μετρήσεις έγιναν σε Κ.Σ.Ε. αποτελούμενο από 3 PS και έναν PM.					

Είναι προφανές ότι κάθε μέθοδος ακριβούς επίλυσης του Clique Number (που είναι NP-Complete πρόβλημα) φθάνει κάποια στιγμή στα όριά της. Έτσι για περιπτώσεις που πρέπει να βρεθεί λύση σε προβλήματα με μέγεθος που απαγορεύει την ακριβή επίλυση, είμαστε αναγκασμένοι να καταλήξουμε σε κάποια ευρετικής μορφής προσπάθεια επίλυσης από αυτές που έχουν αναφερθεί.

Το Κ.Σ.Ε., όπως φαίνεται από τους Πίνακες 21 & 22 βρίσκει την βέλτιστη λύση σε χρόνο που είναι κατά πολύ μικρότερος του συνολικού, ενώ η υπόλοιπη επεξεργασία καταναλώνεται σε ψάξιμο του χώρου λύσεων που απομένει χωρίς να δίνει καλύτερη λύση.

Πίνακας 21 – Κ.Σ.Ε. & Χρονική Εξέλιξη Εύρεσης Μέγιστης Κλίκας

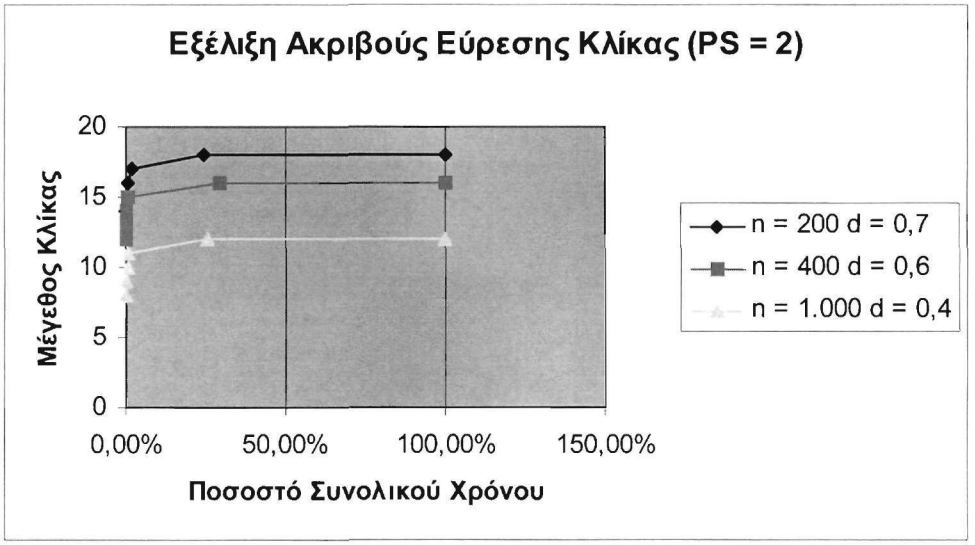
Κόμβοι	200	Κόμβοι	400	Κόμβοι	1.000
Ακμές	13.940	Ακμές	47.817	Ακμές	200.014
Πυκνότητα	0,7005	Πυκνότητα	0,5992	Πυκνότητα	0,4004
T (sec)	**CN**	**T (sec)**	**CN**	**T (sec)**	**CN**
0	14	0	12	0	8
0,21	15	0,55	13	1,16	9
0,32	16	0,93	14	1,98	10
0,71	17	2,03	15	3,85	11
9,06	18	123,25	16	111,06	12
37,18	18	420,34	16	429,19	12

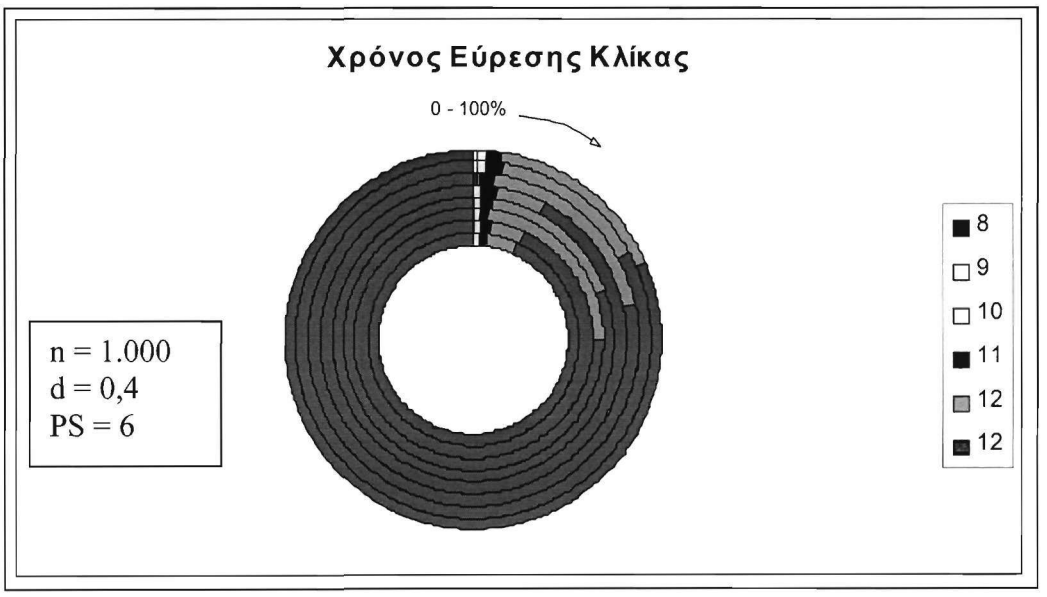
Οι παραπάνω μετρήσεις έγιναν σε Κ.Σ.Ε. αποτελούμενο από 2 PS και έναν PM.

Στον Πίνακα 22 μετρήθηκε ο χρόνος σε διαφορετικά, τυχαία γραφήματα με σταθερό μέγεθος 1.000 κόμβων και περίπου σταθερή πυκνότητα 0,4. Παρατηρούμε ότι σε όλες τις περιπτώσεις χρειάζονται 3 sec περίπου για να βρεθεί λύση μόλις κατά ένα υπολειπόμενη της βέλτιστης που και αυτή βρίσκεται από το Κ.Σ.Ε. σε χρόνο κατά πολύ μικρότερο του συνολικά απιτούμενου προκειμένου να διερευνηθεί ολόκληρος ο χώρος των λύσεων.

Πίνακας 22 – Κ.Σ.Ε. & Χρονική Εξέλιξη Εύρεσης Μέγιστης Κλίκας								
Κόμβοι n = 1.000	e = 199.683 d = 0,3998	e = 200.507 d = 0,4014	e = 199.867 d = 0,4001	e = 199.728 d = 0,3999	e = 199.373 d = 0,3991	e = 199.724 d = 0,3998	e = 199.932 d = 0,4003	e = 199.788 d = 0,4
CN	**Χρόνος T (sec)**							
7						0		
8		0	0	0		0,33	0	0
9	0	0,6	0,49	0,55	0	0,55	0,55	0,55
10	1,21	1,09	0,93	0,99	1,15	0,55	1,21	1,26
11	2,3	2,14	2,41	2,63	2,42	2,91	3,19	2,41
12	6,59	7,3	45,91	30,59	8,18	35,87	24,5	28,34
12	140,33	149,01	148,46	144,07	137,48	145,5	145,0	143,9
Οι παραπάνω μετρήσεις έγιναν σε Κ.Σ.Ε. αποτελούμενο από 6 PS και έναν PM.								

Εικόνα 12 – Χρονική Εξέλιξη Εύρεσης Μέγιστης Κλίκας





4.3 Στάθμιση Φόρτου Εργασίας

Ο PM αποθηκεύει υποπροβλήματα προκειμένου να τα επιλύει κατά τη διάρκεια των Wait Status. Όλα τα υποπροβλήματα με μέγεθος μικρότερο του $Word\_Length$ δεν αποστέλλονται σε καμμία περίπτωση σε Επιλυτή διότι δεν χρειάζονται διάσπαση και λύνονται απευθείας από την Boolean function στον PM σε χρόνο που υπολείπεται του κόστους επικοινωνίας.

Στον Πίνακα 23 παρατίθενται οι μέσοι χρόνοι που μετρήθηκαν για διάφορες τιμές του $Critical\_Low$ σε τυχαία γραφήματα με κόμβους n = 500, πυκνότητα d = 0,5 και ακμές e = 62.414, κάτω φράγμα LB = 10, μέγιστη κλίκα CN = 14. Κάθε υπογράφημα με μέγεθος μεγαλύτερο από $Critical\_Low$ στέλνεται σε PS ενώ διαφορετικά η επίλυσή του γίνεται από τον PM.

Ο PM μόνος του χρειάζεται κατά μέσο όρο 193.67 sec για να επιλύσει αυτού του μεγέθους προβλήματα ενώ όταν ο PM στέλνει όλα τα υποπροβλήματα σε έναν PS χρειάζονται 199.21 sec. Έτσι στην περίπτωση αυτή το overhead που απαιτεί το Κ.Σ.Ε. για επικοινωνία και λοιπές λειτουργίες πλην επίλυσης σε ένα κοινό τοπικό δίκτυο είναι της τάξης του 3%.

Παρατηρούμε επίσης ότι όταν το Σύστημα περιέχει μέχρι 1 - 2 Επιλυτές είναι εμφανής η βοήθεια που προσφέρει ο PM όταν επιλύει ικανό αριθμό υποπροβλημάτων, σε ποσότητα που βέβαια πρέπει να σταθμιστεί προσεκτικά διότι αν ξεπεραστεί τότε το Σύστημα καθυστερεί. Μόλις όμως αυξηθεί ο αριθμός των PS τότε, όπως είναι αναμενόμενο, η συμμετοχή του PM στην επίλυση γίνεται αμελητέα.

Συμπερασματικά, όταν ο αριθμός των Επιλυτών είναι σχετικά μικρός, ο Διαχειριστής αναλαμβάνει να επιλύσει τοπικά όσα προβλήματα έχουν μέγεθος μικρότερο από Critical_Low, ενώ όταν οι PS αυξηθούν τότε ο PM αφενός δεν συνεισφέρει ιδιαίτερα στην επιτάχυνση με το να επιλύει τοπικά προβλήματα,

αφετέρου πρέπει να δαπανά περισσότερο χρόνο στον συντονισμό των Επιλυτών οπότε τίθεται $Critical\_Low = Word\_Length$. Στην περίπτωση αυτή είναι προτιμότερο, όπως θα αναπτυχθεί στο επόμενο Κεφάλαιο, ο PM να χρησιμοποιεί τους νεκρούς χρόνους αναμονής (wait status) σε λειτουργίες λήψης αποφάσεων με στόχο την βελτίωση της απόδοσης του Συστήματος αλλά και έλεγχο σφαλμάτων αυτού.

Ένα άλλο σημείο που πρέπει να σημειωθεί είναι ότι στον παρακάτω πίνακα η καλύτερη τιμή για την Επιτάχυνση (Speedup) εμφανίζεται για τις τρείς πρώτες τιμές του Critical_Low οπότε και ξεπερνά το 2,9 με άριστη τιμή το 3.

Πίνακας 23 – Στάθμιση Φόρτου Εργασίας

Critical_Low (Μέγεθος Υπογραφήματος)	**Χρόνος (sec)**		
	M = 1 S = 9	**M = 1 S = 3**	**M = 1 S = 1**
0	23.46	68.28	199.21
100	23.39	67.78	197.79
110	23.18	67.29	196.52
120	23.34	67.34	195.20
130	23.29	66.85	193.62
140	23.67	66.57	191.26
150	23.72	66.85	189.66
160	24.00	66.41	185.05
170	32.46	65.47	177.74
180	43.50	64.43	172.36
190		64.49	161.81
200		98.27	158.73
210			147.81
220			199.32

Κόμβοι n = 500, Πυκνότητα d = 0,5, Ακμές e = 62.414, Κάτω φράγμα LB = 10, Μέγιστη κλίκα CN = 14

4.4 Boolean Μέθοδος

Η Boolean function που βρίσκει όλες τις maximal κλίκες ενός γραφήματος άρα και την μέγιστη κλίκα όπως και το clique number είναι στηριγμένη στην [Panayiotopoulos J-C., 1981] και υλοποιήθηκε με την μεθοδολογία που έχει περιγραφεί νωρίτερα.

Γραφήματα μέχρι 32 κόμβων και οποιασδήποτε πυκνότητας επιλύονται σε χρόνο μερικών χιλιοστών του δευτερολέπτου, ενώ όταν η πυκνότητα του γραφήματος γίνεται της τάξης του 0.98, η απόδοση είναι ακόμα καλύτερη. Για μεγαλύτερη ακρίβεια, η επίλυση κάθε γραφήματος (n = 32) έγινε 1.000 ή 10.000 φορές, μετρήθηκε ο συνολικός χρόνος και η απόδοση της Boolean function φαίνεται στον Πίνακα 24.

Το κρίσιμο μέγεθος εδώ είναι η μέγιστη τιμή των γραμμών του πίνακα $BoolTable$. Είναι προφανές ότι θεωρητικά και εφόσον η αρχική boolean παράσταση περιέχει $TotalFactors$ παρενθέσεις, ο πίνακας μπορεί να έχει μέχρι και $2^{TotalFactors}$ γραμμές. Στην πράξη το πλήθος γραμμών διατηρείται σε πολύ μικρότερες τιμές λόγω των απορρίψεων και απορροφήσεων που πραγματοποιούνται από ειδική συνάρτηση που υλοποιήθηκε και η οποία καλείται από την Boolean function κάθε φορά που γίνεται πολλαπλασιαμός με την επόμενη παρένθεση. Το πλήθος των γραμμών που περιέχει ο πίνακας BoolTable στον οποίο εκτελούνται οι boolean πράξεις καθώς και η εξάρτησή τους από το φράγμα εκκίνησης για την τιμή της μέγιστης κλίκας LowBound (LB) φαίνονται στους Πίνακες 25 και 26 για διαφορετικές πυκνότητες γραφημάτων.

Πίνακας 24 – Απόδοση Boolean Function

	Πυκνότητα	Χρόνος (x 10^{-3}) sec	Αρχικό Φράγμα	Μέγιστη Κλίκα
	0,6996	5,82	9	9
	0,7218	3,74	10	10
	0,6956	5,5	9	9
	0,6935	18,73	8	9
	0,6754	28,45	7	8
	0,6956	4,61	9	9
	0,7137	11,81	9	9
	0,7258	52,01	7	9
	0,6754	7,3	8	9
	0,7097	9,4	9	10
Μέση Τιμή	0,70061	14,737		

	Πυκνότητα	Χρόνος (x 10^{-3}) sec	Αρχικό Φράγμα	Μέγιστη Κλίκα
	0,8931	7,14	16	17
	0,9032	2,03	18	18
	0,8931	20,22	15	15
	0,873	6,26	15	16
	0,9032	37,57	15	15
	0,9012	17,53	15	16
	0,9032	79,31	15	15
	0,9073	2,14	18	18
	0,9113	6,48	17	17
	0,9073	99,42	14	17
Μέση Τιμή	0,89959	27,81		

	Πυκνότητα	Χρόνος (x 10^{-4}) sec	Αρχικό Φράγμα	Μέγιστη Κλίκα
	0,9758	4,56	24	24
	0,9698	8,57	23	23
	0,9617	2,26	24	24
	0,9556	64,15	20	21
	0,9556	20,54	21	21
	0,9839	3,9	25	25
	0,9718	2,36	25	25
	0,9758	2,47	25	25
	0,9597	8,3	22	22
	0,9655	9,66	21	21
Μέση Τιμή	0,96752	12,677		

Πίνακας 25 – Πλήθος Γραμμών BoolTable				
Αριθμός παρένθεσης για πολλαπλασιασμό	**Πλήθος γραμμών πριν τον πολ/μο**	**Πλήθος γραμμών μετά τον πολ/μο και τις απορρίψεις και απορροφήσεις**	**Πλήθος γραμμών πριν τον πολ/μο**	**Πλήθος γραμμών μετά τον πολ/μο και τις απορρίψεις και απορροφήσεις**
	LB = 14 Χρόνος T = $8{,}4 \cdot 10^{-2}$ sec		LB = 16 Χρόνος T = $1{,}92 \cdot 10^{-2}$ sec	
2	2	4	2	4
3	4	6	4	6
4	6	10	6	10
5	10	16	10	16
6	16	32	16	32
7	32	48	32	48
8	48	82	48	81
9	82	110	81	109
10	110	155	109	154
11	155	303	154	257
12	303	369	257	263
13	369	450	263	232
14	450	446	232	196
15	446	417	196	171
16	417	427	171	153
17	427	565	153	124
18	565	601	124	71
19	601	484	71	49
20	484	495	49	40
21	495	398	40	16
Κόμβοι Ακμές Πυκνότητα	n = 32 e = 447 d = 0,9012		Μέγιστη Κλίκα CN = 17 Παράγοντες TotalFactors = 21	

Πίνακας 26 – Πλήθος Γραμμών BoolTable				
Αριθμός παρένθεσης για πολλαπλασιασμό	**Πλήθος γραμμών πριν τον πολ/μο**	**Πλήθος γραμμών μετά τον πολ/μο και τις απορρίψεις και απορροφήσεις**	**Πλήθος γραμμών πριν τον πολ/μο**	**Πλήθος γραμμών μετά τον πολ/μο και τις απορρίψεις και απορροφήσεις**
	LB = 8 Χρόνος T = $3{,}73 \cdot 10^{-2}$ sec		LB = 9 Χρόνος T = $1{,}92 \cdot 10^{-2}$ sec	
2	2	3	2	3
3	3	4	3	4
4	4	6	4	6
5	6	10	6	10
6	10	13	10	13
7	13	21	13	21
8	21	32	21	28
9	32	47	28	41
10	47	56	41	46
11	56	66	46	55
12	66	88	55	70
13	88	101	70	79
14	101	122	79	96
15	122	137	96	106
16	137	148	106	109
17	148	186	109	134
18	186	219	134	153
19	219	236	153	170
20	236	273	170	196
21	273	316	196	209
22	316	335	209	215
23	335	326	215	172
24	326	272	172	129
25	272	287	129	99
26	287	233	99	88
27	233	186	88	34
Κόμβοι	n = 32		Μέγιστη Κλίκα	CN = 10
Ακμές	e = 354		Παράγοντες TotalFactors = 27	
Πυκνότητα	d = 0,7137			

4.5 Συγκριτική Απόδοση Κ.Σ.Ε.

Τα τυχαία γραφήματα που χρησιμοποιούνται στις μετρήσεις δημιουργήθηκαν με κριτήριο την μέση πυκνότητα των κόμβων και παραγωγή ψευδοτυχαίων αριθμών μέσω της συνάρτησης βιβλιοθήκης `rand()` που διαθέτει η C++. Τα γραφήματα Hamming και Keller προέρχονται από τον χώρο της Θεωρίας Κωδικοποίησης και της Γεωμετρίας αντίστοιχα και αποτελούν μέλη της ομάδας των benchmarks του «Second DIMACS Implementation Challenge: 1992-1993» που οργανώθηκε από το Πανεπιστήμιο του Rutgers [DIMACS, 1992]. Τα δύο συγκεκριμμένα γραφήματα επελέγησαν προκειμένου να γίνει η σύγκριση με τις μετρήσεις της [Pardalos, Rappe, Resende, 1999].

Επιτάχυνση (Speedup) ορίζεται το πηλίκον $S = \frac{T_n}{T_m}$ [Xavier, Iyengar, 1998] όπου T_n , T_m οι χρόνοι που απαιτούνται για να επιλυθεί ένα πρόβλημα με n και m επεξεργαστές ($n < m$) αντίστοιχα να εκτελούν όλους τους υπολογισμούς (δεν υπολογίζεται δηλαδή ο PM). Στην ιδανική περίπτωση θα είναι $S = \frac{m}{n}$. Στους πίνακες που ακολουθούν, με Tn συμβολίζεται ο χρόνος που δίνει ένα Κ.Σ.Ε. αποτελούμενο από έναν PM και n-1 PS, για παράδειγμα T2 είναι ο χρόνος Κ.Σ.Ε. που αποτελείται από έναν Διαχειριστή και έναν Επιλυτή.

Προκειμένου να είναι δυνατή η σύγκριση με τις μετρήσεις της προγενέστερης εργασίας ρυθμίστηκε $Critical\_Low = Word\_Length = 32$ οπότε όλα τα μεγαλύτερου μεγέθους γραφήματα αποστέλλονται σε PS για επίλυση.

Συγκρινόμενα με τα αποτελέσματα της [Pardalos, Rappe, Resende, 1999], φαίνεται (Πίνακες 27 & 28) πως αυτά που δίνει το παρόν Κ.Σ.Ε. είναι καλύτερα κατά μία τάξη μεγέθους δέκα, παρόλο που το Κ.Σ.Ε. δοκιμάστηκε σε φθηνούς μικροϋπολογιστές των 32-bit, ενώ τα αποτελέσματα που αναφέρονται στην [Pardalos, Rappe, Resende, 1999] λήφθηκαν σε υπολογιστές 64-bit. Επιπρόσθετα, καθώς η πυκνότητα των γραφημάτων μεγαλώνει και μάλιστα ξεπερνά το 85%, φαίνεται πως μόνον η παρούσα μέθοδος μπορεί να λύσει ακριβώς το πρόβλημα της μέγιστης κλίκας σε απλούς μικροϋπολογιστές. Ακόμα περισσότερο, σε «φυσιολογικές» πυκνότητες μπορεί να αντιμετωπίσει γραφήματα με χιλιάδες κόμβους και εκατομμύρια ακμές σε λίγα δευτερόλεπτα (π.χ. 6.000 κόμβοι, 1.798.238 ακμές σε περίπου 55 sec).

Συμπερασματικά, η προτεινόμενη μέθοδος μπορεί να αντιμετωπίσει οποιοδήποτε γράφημα που προέρχεται από πραγματικό πρόβλημα εκτός βέβαια από πολύ μεγάλα γραφήματα (όπως αυτό της [Abello, Pardalos, Resende, 1999] με 54.000.000 κόμβους και 107.000.000 ακμές), το οποίο δεν χωρά στη μνήμη ενός φθηνού μικροϋπολογιστή.

Πίνακας 27 – Συγκριτική Απόδοση του Κ.Σ.Ε. σε Τυχαία Γραφήματα

Κόμβοι n	Πυκνότητα d	Ακμές	Low Bound	Clique Number	Χρόνος (sec)			Speedup T4/T10	Speedup T2/T4	Αποτελέσματα της [Pardalos, Rappe, Resende, 1999]		
					S = 9 M = 1	S = 3 M = 1	S = 1 M = 1			2 επεξεργ.	4 επεξεργ.	Speedup T2/T4
100	0,8	3.952	15	19	1,43	2,42	5,49	1,69	2,27	35,78	16,48	2,17
100	0,9	4.450	26	31	12,90	29,77	81,95	2,31	2,75	1731,35	666,27	2,60
200	0,6	11.974	13	14	1,75	2,86	7,14	1,63	2,50	64,16	24,42	2,63
200	0,7	13.899	13	18	9,17	24,23	70,14	2,64	2,89	929,04	353,01	2,63
200	0,8	15.879	20	25	366,73							
300	0,4	17.879	6	10	2,00	1,65	3,29	0,83	1,99	14,79	8,14	1,82
300	0,5	22.481	9	12	2,52	4,40	10,93	1,75	2,48	66,72	28,44	2,35
300	0,6	26.809	11	15	12,90	35,32	102,79	2,74	2,91	1033,63	388,80	2,66
300	0,7	31.362	16	20	318,02							
400	0,4	31.954	7	10	2,64	3,46	8,02	1,31	2,32	48,56	21,00	2,31
400	0,5	39.763	8	13	6,54	15,93	45,2	2,44	2,84	384,79	151,10	2,55
400	0,6	48.000	13	16	101,11	289,5	855,1	2,86	2,95	9213,88	3466,43	2,66
400	0,7	55.725	17	21	4661,9							
500	0,3	37.204	6	8	3,02	2,31	5,11	0,76	2,21	18,56	10,13	1,83
500	0,4	49.604	7	11	4,01	7,36	19,44	1,84	2,64	121,07	21,12	2,37
500	0,5	62.414	10	14	23,46	68,28	199,21	2,91	2,91	1452,71	584,03	2,48
500	0,6	74.858	12	17	535,74							
1000	0,3	150.284	6	10	13,84							
1000	0,4	199.521	8	12	92,77							
2000	0,3	600.260	6	11	190,36							
4000	0,2	1.598.868	6	9	202,18							
5000	0,2	2.499.856	6	10	555,84							
6000	0,1	1.798.238	5	7	55,48							
6000	0,2	3.598.382	6	10	1424,3							

M είναι το πλήθος των PM και **S** των PS που χρησιμοποιεί το Κ.Σ.Ε. Τ2 ο χρόνος που απαιτεί ένας PS, Τ4 τρείς και Τ10 εννέα (δεν υπολογίζεται ο PM).

Πίνακας 28 – Συγκριτική Απόδοση του Κ.Σ.Ε. σε Benchmarks

Hamming 8 – 4 Graph

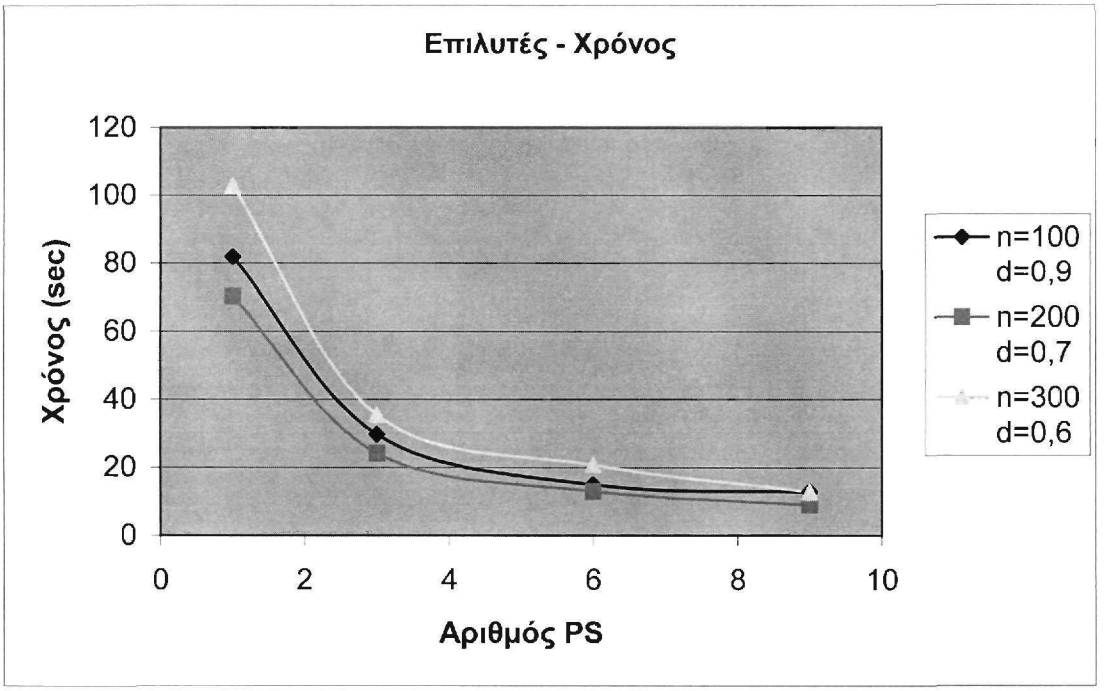
n	d	Κόμβοι	Πυκνότητα	Ακμές	LowBound	Clique Number	Χρόνος (sec)			Speedup T4/T10	Speedup T2/T4	Αποτελέσματα της [Pardalos, Rappe, Resende, 1999]		
							S = 9 M = 1	S = 3 M = 1	S = 1 M = 1			2 επεξεργ.	4 επεξεργ.	Speedup T2/T4
8	4	256	0,64	20.864	16	16	7,31	17,91	50,86	2,45	2,84	421,12	166,85	2,52

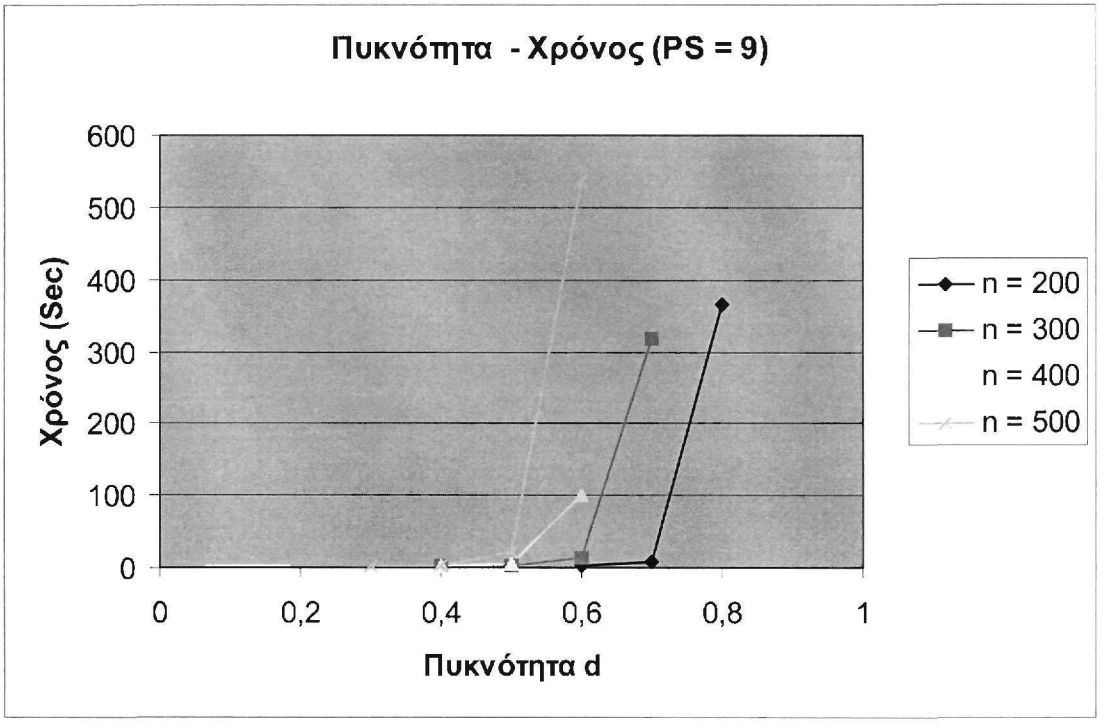
Keller 4 Graph

n	Κόμβοι	Πυκνότητα	Ακμές	LowBound	Clique Number	Χρόνος (sec)			Speedup T4/T10	Speedup T2/T4	Αποτελέσματα της [Pardalos, Rappe, Resende, 1999]		
						S = 9 M = 1	S = 3 M = 1	S = 1 M = 1			2 επεξεργ	4 επεξεργ.	Speedup T2/T4
4	171	0,65	9.435	7	11	3,57	7,36	20,65	2,06	2,80	80,289	33,362	2,41

M είναι το πλήθος των PM και **S** των PS που χρησιμοποιεί το Κ.Σ.Ε. Τ2 ο χρόνος που απαιτεί ένας PS, Τ4 τρείς και Τ10 εννέα (δεν υπολογίζεται ο PM). Τα παραπάνω γραφήματα είναι διαθέσιμα από το ftp site του Rutgers University (ftp://dimacs.rutgers.edu/pub/challenge/)

Εικόνα 13 – Επίδραση Πυκνότητας & Επιλυτών Στον Χρόνο





Κεφάλαιο 5

ΓΕΝΙΚΕΥΣΕΙΣ

ΓΕΝΙΚΕΥΣΕΙΣ

5.1 Πολυεπίπεδα

Στα όσα έχουμε δεί μέχρι τώρα, το Κ.Σ.Ε. αποτελείται από έναν Διαχειριστή Προβλήματος PM συντονιστή ενός ή περισσοτέρων Επιλυτών Προβλήματος PS στους οποίους διανέμει τα υποπροβλήματα. Ωστόσο, έχει ήδη αναφερθεί ότι είναι δυνατόν να έχουμε nested φαινόμενο του όλου Συστήματος.

Η δόμηση αυτή είναι δυνατόν να εφαρμοστεί σε περιπτώσεις πολύ μεγάλου μεγέθους προβλημάτων που μάλιστα προορίζονται για επίλυση με όχι αναγκαστικά σύγχρονη επικοινωνία μεταξύ των μελών του (π.χ. Internet).

Στην περίπτωση αυτή δεν μιλάμε μόνο για τάξεις (επίπεδα) στη διάσπαση του αρχικού προβλήματος, αλλά και για τάξεις στο επίπεδο των Διαχειριστών, όπου ένας Επιλυτής είναι εν δυνάμει ένας νέος μικρότερος Διαχειριστής στο δικό του περιβάλλον. Στο Κ.Σ.Ε. που περιγράφηκε στην παρούσα διατριβή ο Διαχειριστής Προβλήματος PM θεωρείται μηδενικής τάξης και διαχειρίζεται τους Επιλυτές.

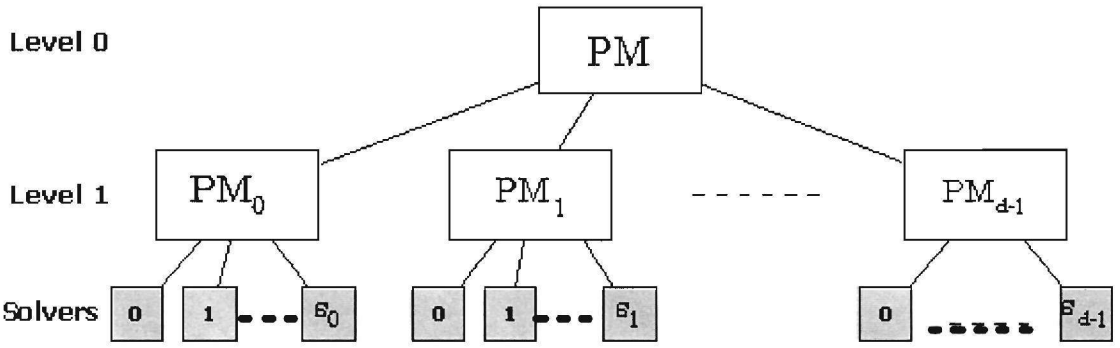
Αν θεωρήσουμε ότι ο Διαχειριστής Προβλήματος δεν απευθύνεται σε τελικούς Επιλυτές αλλά σε μία επόμενη τάξη Διαχειριστών οι οποίοι ελέγχουν τις ομάδες των τελικών Επιλυτών, τότε αναφερόμαστε σε Διαχείριση δύο επιπέδων. Με τον ίδιο τρόπο μπορεί βέβαια να οριστεί Κ.Σ.Ε. τριών ή και **k-επιπέδων**.

Είναι προφανές ότι το Πολυεπίπεδο Κ.Σ.Ε. που περιγράφεται εδώ, αποτελείται από πολλαπλά, ιεραρχικά συνενωμένα σε δομή δένδρου, Κ.Σ.Ε. ενός επιπέδου. Κάθε επεξεργασία του Πολυεπίπεδου Κ.Σ.Ε. ανήκει σε κάποιο επίπεδο, και μαζί με τις υπόλοιπες του ιδίου επιπέδου και τον κόμβο - πατέρα αποτελούν Κ.Σ.Ε. ενός επιπέδου.

Στην Εικόνα 14 όπου σχηματοποιείται Κ.Σ.Ε. δύο επιπέδων:

- Ο κεντρικός Διαχειριστής μηδενικής τάξης PM που έχει ταυτότητα d, ελέγχει στο αμέσως επόμενο επίπεδο d το πλήθος Διαχειριστές 1ης τάξης.
- Οι Διαχειριστές πρώτης τάξης είναι οι PM_i, $i = 0,1,...,d-1$ με ταυτότητα τον αντίστοιχο δείκτη $0,1,...,d-1$.
- Κάθε Διαχειριστής 1ης τάξης PM_i ελέγχει s_i το πλήθος Επιλυτές με ταυτότητες τους διαφορετικούς ανά δύο ακεραίους $0,1,...,s_i$.

Εικόνα 14 – Διαχείριση 2-επίπεδων Εξηρτημένων Επιλυτών



Οι ακέραιοι που περιγράφτηκαν παραπάνω αρκούν για τον κατά μοναδικό τρόπο χαρακτηρισμό των Επιλυτών εντός του συγκεκριμένου υποσυστήματος, ενώ σε καθολικό επίπεδο, κάθε Διαχειριστής αλλά και Επιλυτής παίρνουν ταυτότητα μέσω της χαρακτηριστικής ακολουθίας των κωδικών τους ξεκινώντας από την ρίζα.

Για παράδειγμα οι ελεγχόμενοι από τον PM_0 Επιλυτές με ταυτότητα 0 και s_0 έχουν καθολικό και μοναδικό για όλο το Πολυεπίπεδο Κ.Σ.Ε. κωδικό - ταυτότητα 0.0 και $0.s_0$ αντίστοιχα.

Σε κάθε υποσύστημα - Κ.Σ.Ε. που περιέχεται στο Πολυεπίπεδο Κ.Σ.Ε. η επικοινωνία γίνεται με τα γνωστής μορφής αρχεία που έχουν ήδη περιγραφεί. Σε περίπτωση που χρησιμοποιείται κοινός κατάλογος για όλα τα αρχεία στον ίδιο data server, σαν τμήμα του ονόματος κάθε αρχείου μπαίνει και το καθολικό id του ιδιοκτήτη προκειμένου να ξεχωρίσει από τα υπόλοιπα. Για παράδειγμα ο PM_0 επικοινωνεί με τους Επιλυτές που διοικεί μέσω του αρχείου Comm_file_0 ενώ ο Επιλυτής $0.s_0$ αποστέλει το καλύτερο αποτέλεσμα που βρίσκει μέσω του αρχείου Res_file_0.s0.

Κάθε Διαχειριστής εκτός από αυτόν της μηδενικής τάξης επικοινωνεί μέσω των αντιστοίχων αρχείων τόσο με τον πατρικό Διαχειριστή της προηγούμενης τάξης, όσο και με τους Διαχειριστές επόμενης τάξης ή τους Επιλυτές που ελέγχει.

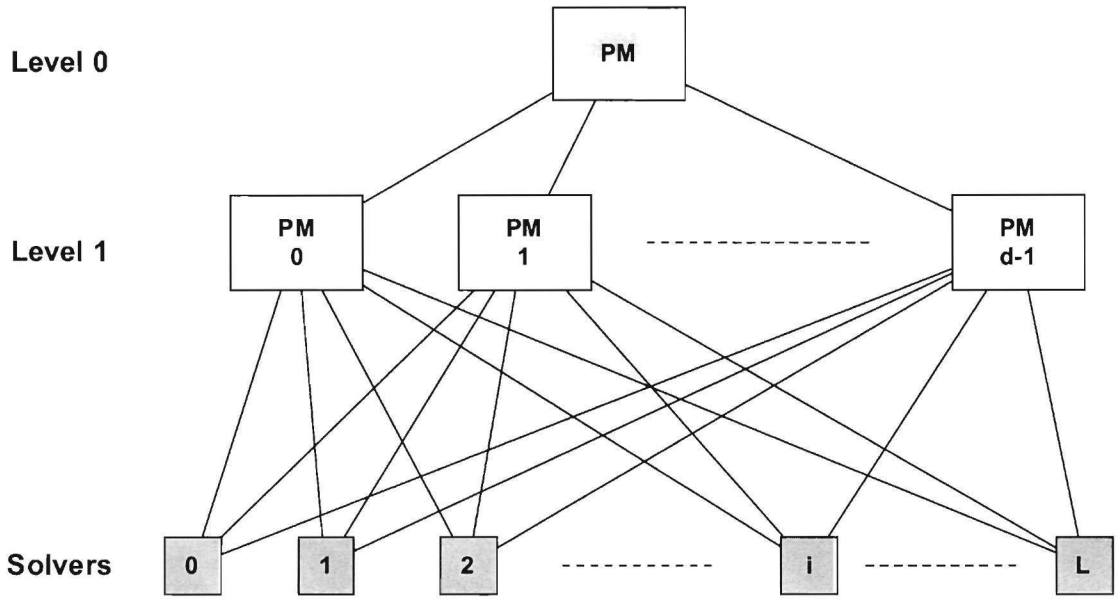
Το Σύστημα Διαχείρισης Έργου σε κάθε Διαχειριστή μπορεί να διαφέρει από επίπεδο σε επίπεδο με τους Διαχειριστές που βρίσκονται σε χαμηλότερο επίπεδο να συμμετέχουν περισσότερο στην Επίλυση ενώ οι υψηλοτέρου επιπέδου να απασχολούνται περισσότερο ή καθολικά με την Διοίκηση του Συστήματος.

Αποτελεί αντικείμενο έρευνας και βρίσκεται στο στάδιο της εξέλιξης η δημιουργία ενός Κ.Σ.Ε. πολλαπλών επιπέδων που θα επικοινωνεί σε τοπικό δίκτυο αλλά και ασύγχρονου Κ.Σ.Ε. που θα επικοινωνεί μέσω του Internet.

Η κατηγορία διαχείρισης k-επιπέδων που αναφέρθηκε έχει το χαρακτήρα ότι ο κάθε διαχειριστής οποιασδήποτε τάξης έχει δικούς του εξαρτημένους Επιλυτές ή δικούς του εξαρτημένους Διαχειριστές της επόμενης τάξης, δηλαδή πρόκειται περί διαχείρισης **k-επιπέδων εξαρτημένων επιλυτών**.

Ωστόσο η έρευνά μας σήμερα βρίσκεται στο σημείο αντιμετώπισης του όλου προβλήματος με σχεδιασμό Συστήματος διαχείρισης **k-επιπέδων ανεξαρτήτων επιλυτών**, όπου και οι διαχειριστές του επομένου επιπέδου αλλά και οι επιλυτές είναι σε κοινή διάθεση του ανωτέρου επιπέδου. Τα μέχρι στιγμής αποτελέσματα συγκλίνουν σε πολύ ικανοποιητικές αποδόσεις.

Εικόνα 15 – Διαχείριση 2-επίπεδων Ανεξαρτήτων Επιλυτών



5.2 Κωδικοποίηση

Στις πειραματικές μετρήσεις που έγιναν, το υποσύστημα επικοινωνίας χρησιμοποιήθηκε στη σύγχρονη μορφή του προκειμένου να μετρηθούν οι χρόνοι απόδοσης του Κ.Σ.Ε. του οποίου τα μέλη, Διαχειριστής και Επιλυτές, επικοινωνούσαν μέσω τοπικού δικτύου. Το Σύστημα που προτείνεται όμως μπορεί να χρησιμοποιηθεί και με ασύγχρονο τρόπο κυρίως στην περίπτωση όπου οι υπολογιστές που το απαρτίζουν επικοινωνούν όχι σε τοπικό αλλά σε ευρύτερο Δίκτυο, ή Internet.

Αν αγνοηθεί το μέσον επικοινωνίας (δίκτυο κλπ) απομένει το **περιεχόμενο** αυτής, δηλαδή «το τι μεταδίδουν». Λόγω του χρόνου και του κόστους επικοινωνίας, είναι ζωτικής σημασίας τα δεδομένα που μεταφέρονται να περιέχουν την μεγαλύτερη δυνατή **πυκνότητα** πληροφορίας. Ήδη στην παράγραφο 2.5 έχουν αναφερθεί τρόποι με τους οποίους το σύστημα επικοινωνίας μπορεί να σηματοδοτήσει γεγονότα ή εντολές χωρίς να αναφέρεται αποκλειστικά (ευθέως) σε αυτά.

Για παράδειγμα το Κ.Σ.Ε. χρησιμοποιεί συνδυασμό των δεδομένων Στοιχεία Αποστολέα και Στοιχεία Υποπροβλήματος προκειμένου να μεταδώσει την πληροφορία Εντολή. Έτσι τα δεδομένα περιέχουν πληροφορία πολλαπλών επιπέδων που αξιοποιείται από τα αντίστοιχα υποσυστήματα στα οποία απευθύνεται.

Τα δεδομένα που ανταλλάσσουν οι επεξεργασίες είναι, ειδωμένα από μεριάς Πληροφορίας, δύο ειδών:

- Δεδομένα **συγχρονισμού & συνεννόησης,** και
- Δεδομένα **προβλήματος**.

Συνήθως τα δεδομένα συγχρονισμού και συνεννόησης που ανταλλάσσουν οι διαδικασίες είναι σχετικά μικρού όγκου αν και όχι αμελητέου. Ο κύριος όγκος δεδομένων αφορά βέβαια αυτά του προβλήματος, που σε περιπτώσεις μεγάλων γραφημάτων είναι ιδιαίτερα σημαντικός. Σε κάθε περίπτωση επομένως γίνεται από επιθυμητή μέχρι και απολύτως αναγκαία η **κωδικοποίηση** των δεδομένων επικοινωνίας.

Ο προφανής στόχος της κωδικοποίησης αυτής είναι η ελαχιστοποίηση του χρόνου προσπέλασης μεταξύ των επεξεργαστών, όμως όσον αφορά τα δεδομένα του προβλήματος, όπως θα δούμε παρακάτω υπάρχουν και άλλοι σημαντικότεροι ίσως λόγοι που κάνουν το συγκεκριμένο θέμα να χρήζει ιδιαίτερης μελέτης.

Έστω ότι το Κ.Σ.Ε. αντιμετωπίζει n διαφορετικά προβλήματα (CLIQUE-NUMBER, TRAVELING SALESMAN κλπ):

$$SP = \{\Pi_1^0, \Pi_2^0, ..., \Pi_n^0\}$$

Η κωδικοποίηση / τυποποίηση των υποσυνόλων δεδομένων επικοινωνίας από επεξεργαστή σε επεξεργαστή προτείνεται να γίνει με τους εξής τρόπους:

- **Μεταγλώσσα**

Ένα σύνολο Διαταγών

$$\Omega = \{\Delta_1, \Delta_2, ...\Delta_i, ..., \Delta_\lambda\}$$

όπου Δ_i είναι διαταγή με λέξη και κώδικα συνάρτηση σε βιβλιοθήκη

Μία διαταγή μπορεί να είναι για παράδειγμα

- Μέθοδος επίλυσης υποπροβλήματος. Ο PM ανάλογα με τις παραμέτρους του υποπροβλήματος ή και από γενικότερες παραμέτρους του προβλήματος επιλέγει την μέθοδο επίλυσης.
- Μέθοδος εξαγωγής – δημιουργίας του υποπροβλήματος που ανατίθεται στον PS.

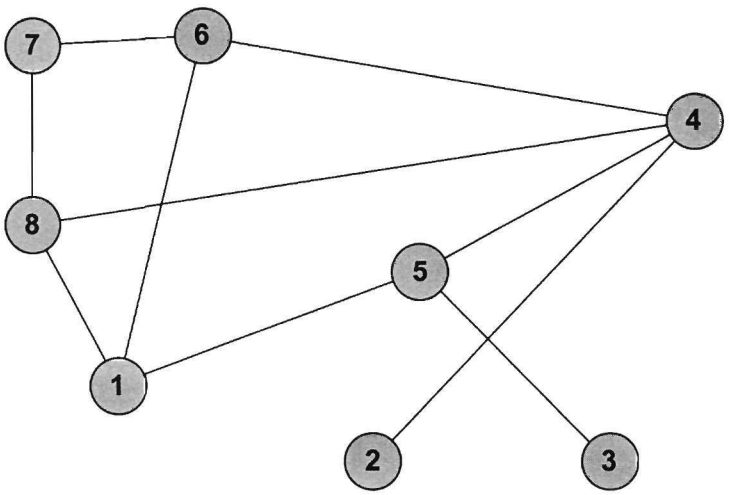
Για κάθε ένα είδος προβλήματος Π_i^0, το Κ.Σ.Ε. διαθέτει σε ειδική βάση δεδομένων την αντίστοιχη μεταγλώσσα, μέρος της οποίας μπορεί να είνα κοινό και ανεξάρτητο του $i = 1, 2, ..., n$.

- **Μαθηματικό Σχέδιο (Mathematical Design)**

Το κάθε μέλος του Κ.Σ.Ε. διαθέτει σε μορφή πίνακα μιάς ή περισσοτέρων διαστάσεων την κωδικοποίηση ενεργειών – υποπροβλημάτων και με χωρικές συντεταγμένες επιλέγεται είτε το μήνυμα ενέργειας, είτε το υποπρόβλημα ή συνδυασμός.

Η κωδικοποίηση των υποπροβλημάτων στην περίπτωση του case study CLIQUE-NUMBER έγινε με την μέθοδο χαρακτηριστικής ακολουθίας, η οποία εξάγεται από την παράθεση των διαδοχικών pivot-nodes που προκύπτουν κατά τη φάση της διάσπασης σε υπογραφήματα. Η μέθοδος αυτή ενώ είναι οικονομική από πλευράς όγκου δεδομένων, παρουσιάζει το πρόβλημα της επανάληψης των υπολογισμών, πρόβλημα ούτως ή άλλως εγγενές στην κατανεμημένη επεξεργασία. Συγκεκριμμένα είναι δυνατόν κατά την διάρκεια της επίλυσης να ανατεθεί τελικά πολλές φορές το ίδιο (υπο)γράφημα προς επίλυση στον ίδιο ή σε διαφορετικούς Επιλυτές.

Εικόνα 16 - Πλεονασμός Υπολογισμών σε Γράφημα



Παρατηρούμε ότι με την μέθοδο διάσπασης σε υποπροβλήματα που αναλυθηκε στο Κεφάλαιο 2, αν στο γράφημα της εικόνας 12 επιλεγεί σαν pivot-node ο κόμβος v_1 προκύπτει το υποπρόβλημα $\{v_5, v_6, v_8\}$ το ίδιο δηλαδή που προκύπτει με pivot-node τον v_4.

Έτσι η μέθοδος της χαρακτηριστικής ακολουθίας σηματοδοτεί με μοναδικό τρόπο ένα συγκεκριμένο γράφημα που ανατίθεται προς επίλυση σε PS, όμως σε καμμία περίπτωση δεν εξασφαλίζει υποχρεωτικά την μοναδικότητα του υποπροβλήματος, το ότι δηλαδή το συγκεκριμένο υποπρόβλημα δεν έχει λυθεί ξανά και θα λυθεί για μία και μοναδική φορά. Και είναι γνωστό (Θεώρημα 2.2.3) ότι η διάσπαση ενός προβλήματος σε υποπροβλήματα είναι βέλτιστη όταν οι χώροι των δυνατών λύσεων αποτελούν διαμέριση του χώρου των δυνατών λύσεων του πατρικού προβλήματος.

Αποτελεί αντικείμενο έρευνας σήμερα

- Σε ποιό επίπεδο διάσπασης εμφανίζεται σε μεγαλύτερο βαθμό το φαινόμενο της εμφάνισης του ίδιου υποπροβλήματος για επίλυση.
- Με ποιά μέθοδο κωδικοποίησης μπορεί το Σύστημα να γνωρίζει ότι το υποπρόβλημα που αντιμετωπίζει έχει ήδη επιλυθεί οπότε μπορεί να το απορρίψει. Για παράδειγμα, στο προηγούμενο γράφημα το υποπρόβλημα $\{v_5, v_6, v_8\}$ που προέκυψε θα μπορούσε να «ανάβει» τα αντίστοιχα bits ενός byte σχηματίζοντας έτσι τον δυαδικό αριθμό $10110000_{(2)}$ που ο αντίστοιχός του στο δεκαδικό είναι ο $176_{(10)}$ ο οποίος είναι ο μοναδικός αριθμός ταυτότητας του γραφήματος.

5.3 Διαχείριση Κινδύνου

Στο Κ.Σ.Ε. όπως σε κάθε Σύστημα, υπάρχουν σημεία ικανά να προκαλέσουν δυσλειτουργία ή ακόμα και την καθολική κατάρρευσή του. Για τον λόγο αυτό και προκειμένου να είναι ολοκληρωμένο το προτεινόμενο Σύστημα, κρίνεται απαραίτητο να παρέχει σε κάποιο βαθμό ανοχή στα λάθη και τις αστοχίες που μπορεί να προκύψουν κατά την περίοδο της λειτουργίας του.

Στο Κ.Σ.Ε. διακρίνονται δύο κατηγορίες **κατάρρευσης** αυτού. Η πρώτη κατηγορία αναφέρεται σε ολοκληρωτική κατάρρευση (Black Operational Hole [Panayiotopoulos J-C., 1992]), και η δεύτερη σε αστοχίες και δυσλειτουργίες (Gray Operational Holes). Η πρώτη κατηγορία μπορεί να συμβεί στις ακόλουθες περιπτώσεις:

1. Κατάρρευση Διαχειριστή Προβλήματος 0 – Επιπέδου (PM).
2. Ολοκληρωτική Κατάρρευση Δικτύου.

Και στις δύο περιπτώσεις η μετάλλαξη του Συστήματος που προκαλείται είναι η απλή ακινησία.

Στην δεύτερη κατηγορία υπάρχουν δύο βασικά είδη αστοχίας της λειτουργίας του Συστήματος:

1. Κατάρρευση Διαχειριστή Προβλήματος k – Επιπέδου (k>0).
2. Κατάρρευση Επιλυτή Προβλήματος.

Στην περίπτωση της κατάρρευσης του Διαχειριστή Προβλήματος k - Επιπέδου, το Σύστημα αντιδρά ως εξής:

1. Σταματά η διάσπαση, παραγωγή και διανομή των υποπροβλημάτων στα επόμενα επίπεδα.
2. Οι ήδη ευρισκόμενοι σε αναμονή Επιλυτές των επομένων επιπέδων παραμένουν διαρκώς σε αυτή την κατάσταση.
3. Οι Επιλυτές του επομένου επιπέδου που ολοκληρώνουν την Επίλυση του τελευταίου υποπροβλήματος που τους είχε ανατεθεί, μεταβαίνουν σε wait status.
4. Η κατάσταση ισορροπίας του Συστήματος επέρχεται όταν όλοι οι Επιλυτές του επομένου επιπέδου έχουν επιστρέψει το αποτέλεσμα του πακέτου που τους είχε ανατεθεί και βρίσκονται σε κατάσταση αναμονής ελέγχοντας το mailbox τους για την επόμενη εντολή του PM.

Είναι προφανές και βέβαια αναμενόμενο για μοντέλο Διαχειριστή – Εργάτη (Master/Slave), ότι η κατάρρευση είναι ολοκληρωτική αν δεν προβλεφθούν οι εξής δύο παράμετροι:

1. Αντικατάσταση του PM από άλλη επεξεργασία – υπολογιστή.
2. Εκκίνηση του Συστήματος στη νέα μορφή από σημείο όσο το δυνατόν πιό κοντά στην κατάσταση που βρισκόταν την στιγμή της κατάρρευσης, προκειμένου να μην χαθεί η επεξεργασία που έχει ήδη γίνει (recovering system).

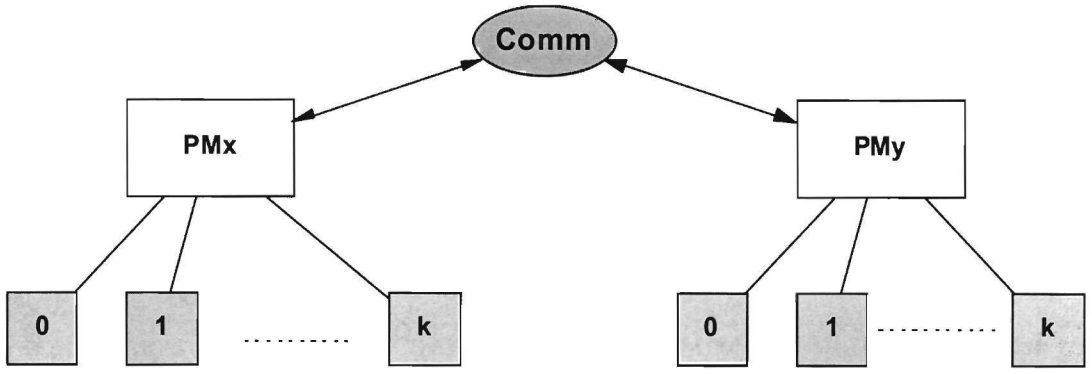
Η μέθοδος αντιμετώπισης στα πρότυπα που χρησιμοποιούνται στα κατανεμημένα συστήματα προσομοίωσης [Fujimoto, 2000], βασίζεται στην περιοδική αποθήκευση της τρέχουσας κατάστασης του Συστήματος προκειμένου να μπορεί να αναπαραχθεί από το τελευταίο αποθηκευμένο στιγμιότυπο του.

Στην περίπτωση του Κ.Σ.Ε. δεν απαιτείται να ληφθεί ειδική μέριμνα αποθήκευσης της κατάστασής του, δεδομένου ότι η χρήση των mailboxes για επικοινωνία ανάμεσα στον PM και τους PS «αφήνει ίχνη» που μπορούν αμέσως μετά την τελική κατάσταση ισορροπίας που περιγράφηκε να συλλεχθούν και έτσι να αναπαραχθεί η κατάσταση που βρισκόταν το Σύστημα την στιγμή προ της κατάρρευσης.

Επειδή η διαδικασία διάσπασης και παραγωγής των υποπροβλημάτων είναι γνωστή και γίνεται με ντετερμινιστικό τρόπο, η εφαρμογή που αναλαμβάνει να αντικαταστήσει τον PM, διαβάζοντας διαδοχικά τις εγγραφές του Comm_file γνωρίζει τα τελευταία d προβλήματα που ανατέθηκαν καθώς και τα αποτελέσματα που προέκυψαν.

Η ενδεχόμενη αντικατάσταση του 0 - PM μπορεί να αντιμετωπιστεί με διαχωρισμό του Συστήματος σε δύο ισότιμα Υποσυστήματα με έναν Διαχειριστή το καθένα.

Εικόνα 17 – Παράλληλα Κ.Σ.Ε.



Όταν το Κ.Σ.Ε. ξεκινά οι PM αναλαμβάνουν να διαχειριστούν ο καθένας το μισό του όγκου εργασίας, για παράδειγμα με κάποια $\mathrm{mod}\,2$ διαχείριση των παραγομένων υποπροβλημάτων. Οι δύο Διαχειριστές Έργου, έστω PMx και PMy επικοινωνούν και πάλι μέσω αρχείου με κατάλληλη δομή και όνομα έστω Comm. Οι στόχοι της επικοινωνίας είναι

- η ανταλλαγή πληροφοριών σχετικά με την πορεία της λύσης του Προβλήματος (καλύτερη λύση κλπ), αλλά και
- η ενημέρωση αλλήλων ότι όλα βαίνουν καλώς.

Ο PMx παρακολουθεί την κατάσταση του PMy (και αντίστροφα) μέσα από το αρχείο Comm και κάθε φορά που παίρνει νέα αναφορά, μηδενίζει ένα εσωτερικό ρολόϊ. Ανάλογα με το αν παίρνει και πότε αναφορά τίθεται σε μία από τις ακόλουθες καταστάσεις:

Green. Είναι η αρχική κατάσταση εκκίνησης. Ο PMx λαμβάνει κανονικά αναφορά από τον PMy σε χρονικά διαστήματα που δεν ξεπερνούν κάποιο μέγιστο Δt. Σε αυτή την περίπτωση η επίλυση συνεχίζεται κανονικά. Ο προσδιορισμός του Δt είναι βέβαια ένα ανοικτό πρόβλημα προς επίλυση.

Yellow. Ο PMx μπαίνει στην κατάσταση yellow κάθε φορά που το χρονικό διάστημα της τελευταίας λήψης αναφοράς από τον PMy ξεπερνά το Δt. Η επίλυση και ο συντονισμός των PS που ελέγχει συνεχίζεται κανονικά ενώ με την είσοδό του στη φάση αυτή ελέγχει πλέον εκτός από το αρχείο Comm και το Comm_file_y, το αρχείο δηλαδή επικοινωνίας του PMy με τους Επιλυτές του. Σε περίπτωση που για χρόνο Δt_1, το αρχείο Comm_file_y παραμείνει χωρίς ενημέρωση από τον PMy, χωρίς δηλαδή να γίνει ανάθεση νέου Έργου σε Επιλυτή, ο PMx εισέρχεται στην επόμενη κατάσταση.

Red. Με την είσοδό του σε κατάσταση red, ο PMx θεωρεί ότι ο PMy έχει σταματήσει να εργάζεται και αναλαμβάνει τον συντονισμό και διοίκηση των Επιλυτών του. Διαβάζει το Comm_file_y, αποτυπώνει την κατάσταση προ της κατάρρευσης του PMy και αναλαμβάνει την συνέχιση του έργου του, μαζί με αυτό που ήδη επιτελεί ο ίδιος.

Από μεριάς Επιλυτών, δεν απαιτείται καμμία επιπρόσθετη ενέργεια ούτε καν η ενημέρωσή τους για την νέα κατάσταση.

Στην περίπτωση της κατάρρευσης κάποιου Επιλυτή Προβλήματος, έστω PSz, το Σύστημα εξελίσσεται ως εξής:

1. Η διάσπαση, παραγωγή και διανομή των υποπροβλημάτων συνεχίζεται κανονικά από τον PM.
2. Η επίλυση του τελευταίου υποπροβλήματος που ανετέθη στον PSz έχει απροσδιόριστο αποτέλεσμα.
3. Ο PM που ελέγχει τον εν λόγω PSz κατά τον έλεγχο της κατάστασης του Κ.Σ.Ε., δεν παίρνει απάντηση οπότε τον διατηρεί στο μητρώο με τον χαρακτηρισμό 1 (απασχολημένος), με αποτέλεσμα να μην του αναθέτει νέο υποπρόβλημα.
4. Η διαδικασία συνεχίζει μέχρι να ολοκληρωθεί η αποστολή όλων των υποπροβλημάτων, όμως ο PM δεν προχωρεί στις διαδικασίες τερματισμού του συστήματος αν δεν πάρει απάντηση από όλους τους Επιλυτές, επομένως το Σύστημα «κρεμάει» αναμένοντας απάντηση από τον PSz.

Γιαυτό λοιπόν και προκειμένου να αποφευχθεί η εξ ορισμού αυτοματοποιημένη αντίδραση (by default) που αναφέρθηκε πρέπει να γίνουν οι παρακάτω ενέργειες:

1. Αφαίρεση του PS από την ομάδα Επιλυτών, με χαρακτηρισμό της κατάστασής του στο μητρώο σαν 2 (μη διαθέσιμος).
2. Αποστολή του τελευταίου υποπροβλήματος για επίλυση σε άλλο PS.

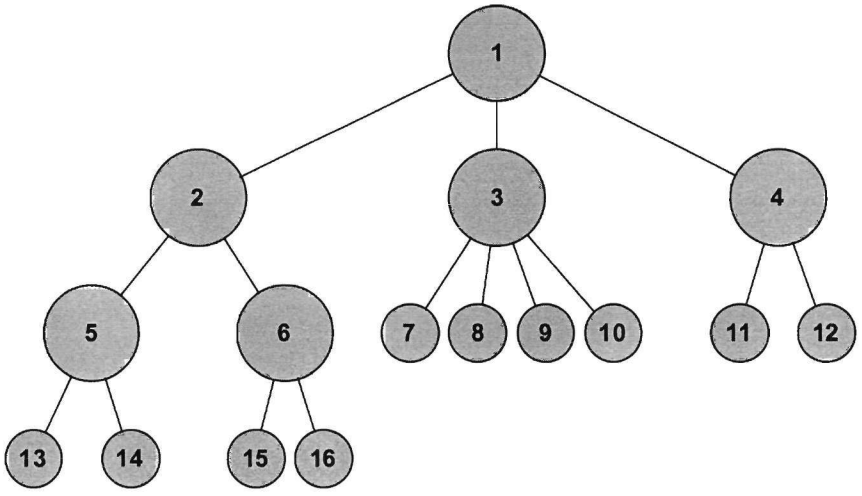
Για τον σκοπό αυτό ο PM διατηρεί στο μητρώο των Επιλυτών ένα πρόσθετο πεδίο που κρατάει την χρονική στιγμή παράδοσης – ανάθεσης του υποπροβλήματος.

Περιοδικά κατά τη διάρκεια του wait status ελέγχει αν υπάρχουν καθυστερήσεις από Επιλυτή, και σε θετική περίπτωση βάζει τον συγκεκριμένο PS σε yellow status. Μετά πάροδο ικανού χρόνου, ο PS παίρνει στο μητρώο τον χαρακτηρισμό 2 (μη διαθέσιμος), το δε υποπρόβλημα που του είχε ανατεθεί και δεν πρόλαβε να επιλύσει δίνεται σε άλλον PM.

Ένα παράδειγμα της γενικής μορφής του πολυεπιπέδου Κ.Σ.Ε. είναι αυτό που φαίνεται στην εικόνα 18, όπου κάθε διαδικασία είναι αριθμημένη με διαδοχικούς ακέραιους αριθμούς. Ενώ διακρίνονται οι ακόλουθες κατηγορίες μελών του πολυεπιπέδου συστήματος:

Διαχειριστής 0 – Επιπέδου	Επεξεργαστής 1
Διαχειριστές k – Επιπέδου (k > 0)	Επεξεργαστής 2 - 6
Επιλυτές	Επεξεργαστής 7 – 16

Εικόνα 18 – Γενική μορφή πολυεπίπεδου Κ.Σ.Ε.

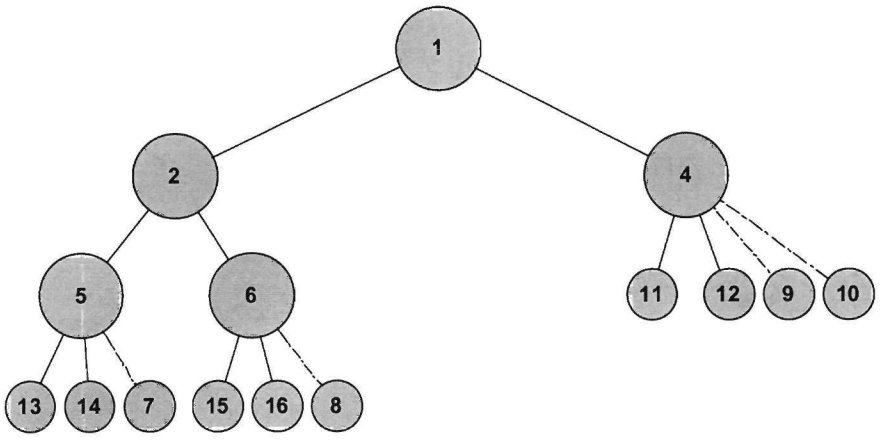


Έστω για παράδειγμα ότι ο PM3 (με αριθμό 3) που είναι Διαχειριστής 1 – Επιπέδου καταρρέει. Ο πατρικός Διαχειριστής PM1, με τρόπους που περιγράφηκαν παραπάνω (καθυστέρηση επικοινωνίας) αναγνωρίζει την κατάρρευση του PM3, οπότε, φυσικά παύει να του αναθέτει νέες εργασίες ενώ τη τρέχουσα την επαναναθέτει σε άλλον Διαχειριστή. Το πρόβλημα που παραμένει στην περίπτωση αυτή είναι το ότι οι Επιλυτές 7 – 10, χάνονται από το Σύστημα και πρέπει να αξιοποιηθούν και να επανενταχθούν σ' αυτό.

Γίνεται φανερό, ότι το Σύστημα θα πρέπει να διατηρεί κάποια εσωτερική επικοινωνία μεταξύ των μελών του, για παράδειγμα κάποιο αρχείο εγγραφών που θάχουν πρόσβαση όλοι, ενώ θα μπορεί να μεταβάλλει μόνον ο PM1, έτσι ώστε να μπορεί να ανασυντάσσεται η δομή του ανάλογα με τις εξελίξεις. Στο παράδειγμα που αναφέρθηκε ο PM1 θα μπορούσε να διαθέσει π.χ. τον επιλυτή 7 στον PM5, τον Επιλυτή 8 στον PM6 και τους απομένοντες 9 και 10 στον PM4, οπότε θα προκύψει μετάλλαξη του αρχικού Συστήματος (Εικόνα 18) όπως φαίνεται στην Εικόνα 19.

Η έρευνά μας σήμερα προσανατολίζεται στην πλήρη μαθηματικοποίηση / κωδικοποίηση της αυτόματης αντιμετώπισης μεταλλάξεων κατανεμημένων συστημάτων.

Εικόνα 19 – Μεταλλακτική μορφή πολυεπίπεδου Κ.Σ.Ε.



Εκτός από τις περιπτώσεις κατάρρευσης του Κ.Σ.Ε. που αναφέρθηκαν παραπάνω, ο PM και κατ' επέκταση το Σύστημα, μπορεί να παίρνει αποφάσεις και να προβαίνει σε ενέργειες που θα έχουν στόχο την βελτίωση της αποδοτικότητας και ευελιξίας του. Οι κατηγορίες των αποφάσεων που βρίσκονται στο στάδιο της έρευνας είναι

- Δυνατότητα Δυναμικής πρόσθεσης ή και αφαίρεσης Επιλυτών ή γενικότερα μελών.
- Δρομολόγηση των πακέτων στους Επιλυτές με βάση αφενός τα χαρακτηριστικά του υποπροβλήματος, αφετέρου την επίδοση του κάθε Επιλυτή που ο PM υπολογίζει μέσω δεδομένων που κρατά σε ειδικά διαμορφωμένο Μητρώο Επιλυτών.
- Επιλογή της μεθόδου επίλυσης για κάθε υποπρόβλημα ανάλογα με τα χαρακτηριστικά του ή τις ανάγκες της πορείας επίλυσης, δεδομένου ότι όπως αναφέρθηκε στην παράγραφο 2.6, οι χρήση πολλαπλών μεθόδων επίλυσης επιταχύνει την τελική επίλυση του προβλήματος.
- Δυνατότητα εκμάθησης μέσα από διατήρηση αρχείου με περιεχόμενο στοιχεία προβλημάτων που το Σύστημα επιλύει και χρήσης της γνώσης σε επόμενη επίλυση.

Συμπεράσματα & Προβληματισμοί

Συμπεράσματα & Προβληματισμοί

Στην παρούσα Διατριβή, παρουσιάστηκε ένα Σύστημα Κατανεμημένης Επίλυσης σκληρών προβλημάτων εκθετικού χρόνου, αποτελούμενο από κοινούς, οικονομικούς μικροϋπολογιστές που επικοινωνούν μέσω δικτύου.

Από την βιβλιογραφική έρευνα που έγινε και αναλύθηκε στο Κεφάλαιο 1, προέκυψαν συμπεράσματα που μπορούν να συνοψιστούν ως εξής:

- Τα NP-Complete προβλήματα με τα οποία ασχολήθηκε η παρούσα Διατριβή θεωρούνται «σκληρά», και αυτό λόγω της μη ύπαρξης πολυωνυμικού αλγορίθμου επίλυσης γι' αυτά.
- Ακόμα περισσότερο, παρόλο που το όλο ζήτημα παραμένει ανοικτό, η σύγχρονη βιβλιογραφία κινείται προς μάλλον απαισιόδοξες εκτιμήσεις με τις περισσότερες αναφορές να προσπαθούν να στηρίξουν την εικασία ότι $P \neq NP$ πράγμα που σημαίνει ότι τα προβλήματα αυτής της κατηγορίας θα είναι πάντα σκληρά.
- Στην περίπτωση που η παραπάνω εικασία είτε αποδειχτεί σωστή, ή έστω όσο δεν αποδεικνύεται λανθασμένη, αποτελεί γεγονός κάθε άλλο παρά αμελητέο, ότι αντιμετωπίζουμε μία ομάδα πραγματικά σκληρών προβλημάτων.
- Αρκετά προβλήματα στην διαδικασία επίλυσης τους παρουσιάζουν δυσκολίες όχι μόνο ως προς την πολυπλοκότητα αλλά και την αναγκαιότητα σε μέγεθος κύριας μνήμης που απαιτούν.
- Το MAX-CLIQUE αποτελεί αντιπροσωπευτική περίπτωση αυτού του είδους των προβλημάτων και μάλιστα από τις «σκληρότερες».
- Το MAX-CLIQUE όπως και πολλά άλλα προβλήματα NP-Complete βρίσκουν εφαρμογή σε πολλούς τομείς της επιστήμης.

- Πολλοί τομείς της επιστήμης έχουν τεράστια, στην ουσία απεριόριστη, ανάγκη για υπολογιστική ισχύ που συχνά παρέχεται από παράλληλα / κατανεμημένα συστήματα.
- Μέσα σε δεκαετίες εξέλιξης έχουν δημιουργηθεί, εξελιχθεί (και πολλές φορές εκλείψει) διάφοροι τύποι παράλληλων και κατανεμημένων συστημάτων με προφανή στόχο την αύξηση της διατιθέμενης υπολογιστικής ισχύος.
- Η εκμετάλλευση του παραλληλισμού μπορεί να γίνει σε διάφορα επίπεδα που ξεκινάνε από το Υλικό και φθάνουν σε Αλγορίθμους και Λογισμικό.
- Τα παράλληλα / κατανεμημένα υπολογιστικά συστήματα από τη μορφή των στενά συνδεδεμένων, ειδικού σχεδιασμού, ομογενών ομάδων επεξεργαστών τείνουν να περάσουν σήμερα στη μορφή χαλαρά συνδεδεμένων συνόλων από μηχανές που μπορεί να είναι και ετερογενείς.
- Σχεδόν όλα τα υπολογιστικά συστήματα που αναλύθηκαν είναι έξτρα κόστους και διαφέρουν από αυτά που κάθε μέρα χρησιμοποιούνται.
- Σχεδόν όλα τα υπολογιστικά συστήματα που αναλύθηκαν είναι έξτρα προδιαγραφών είτε σε επίπεδο υλικού ή / και σε επίπεδο λογισμικού.
- Η ισχύς των μικροϋπολογιστών αυξάνεται με εντυπωσιακούς ρυθμούς εδώ και δεκαετίες.
- Παράλληλα, λόγω της μείωσης του κόστους αυξάνεται και ο πληθυσμός των μηχανών αυτών.
- Η μεγαλύτερη ίσως εξέλιξη είναι οι δυνατότητες επικοινωνίας που έχουν σήμερα και που βέβαια προβλέπεται να έχουν σε πολλαπλάσιο βαθμό αύριο.

Με βάση τους παραπάνω άξονες παρουσιάστηκε Σύστημα Κατανεμημένης Επίλυσης με κεντρικά σημεία:

- Η μορφή του Συστήματος και οι σχέσεις μεταξύ των μελών του.
- Η εκκίνηση των μελών του Συστήματος.
- Ο συγχρονισμός των μηχανών και των διαδικασιών.
- Ο συντονισμός και η καθοδήγησή τους για την επίτευξη του στόχου.
- Πρωτόκολλα επικοινωνίας και ανταλλαγής πληροφορίας.

Έχοντας σαν ανάλογο τις ομάδες των ανθρώπων που εργάζονται για έναν κοινό στόχο όπου υπάρχει πάντα σχεδόν κάποιος συντονιστής / εργοδηγός, με παρόμοια λογική ορίστηκε η δομή του Συστήματος με κεντρικό / αποκεντρωμένο έλεγχο και διοίκηση.

Το Σύστημα δοκιμάστηκε στην επίλυση του Προβλήματος εύρεσης της μέγιστης κλίκας σε γράφημα. Για το σκοπό αυτό παρουσιάστηκε:

- Μέθοδος διάσπασης γραφήματος σε υπογραφήματα με σκοπό την κατανεμημένη επίλυση.
- Νέα μέθοδος ακριβούς επίλυσης γραφήματος από πλευράς μέγιστης κλίκας στηριγμένη σε παλαιότερη εργασία.

Οι πρωτοτυπίες της παρούσης διατριβής συνίστανται στα παρακάτω σημεία:

- Μελετάται η δυνατότητα απλών, «φθηνών» συστημάτων μικροϋπολογιστών να παράσχουν υπολογιστική ισχύ που κανονικά συναντάται σε πολύ ακριβότερους υπολογιστές με ειδικό εξοπλισμό υλικού & λογισμικού.
- Σχεδιάζεται και υλοποιείται ένα **Σύστημα – Πλαίσιο** με το οποίο μπορούν να επιλυθούν κατανεμημένα «σκληρά» προβλήματα βελτιστοποίησης, σε απλό δίκτυο μικροϋπολογιστών Το εν λόγω Σύστημα είναι αντικειμενοστραφές και έχει αναλυθεί σε υποσυστήματα με αρθρώσεις, ενώ μπορεί για ερευνητικούς σκοπούς να αφαιρεθούν ή να προστεθούν νέα υποσυστήματα.
- Αναπτύσσεται νέα μέθοδος ακριβούς επίλυσης της μέγιστης κλίκας, στηριγμένη σε παλαιότερη εργασία με Boolean Άλγεβρα, εντυπωσιακά ικανοποιητικές ταχύτητες και προφανή προοπτική ισχυρή με την μετάβαση των απλών υπολογιστών στα 64 bits.
- Αναπτύσσεται θεωρητικό υπόβαθρο διάσπασης προβλήματος σε υποπροβλήματα. Η εφαρμογή του υποβάθρου στην περίπτωση της μέγιστης κλίκας είχε σαν αποτέλεσμα την υλοποίηση και μαθηματική περιγραφή μεθόδου διάσπασης γραφήματος σε υπογραφήματα προκειμένου να επιλυθούν κατανεμημένα σε εντυπωσιακούς υπολογιστικούς χρόνους.

- Δίνονται γενικεύσεις του προτεινομένου Συστήματος και προτείνεται μία μέθοδος κωδικοποίησης των δεδομένων μεταξύ των μικροεπεξεργαστών με κύριο στόχο την ελαχιστοποίηση του χρόνου προσπέλασης των δεδομένων σε επίπεδο Δικτύου και Μνήμης.

Κλείνοντας, σημειώνουμε τα παρακάτω σημεία

- Ένα σύστημα πρέπει να έχει και αυτό όπως και τα ανθρώπινα συστήματα την δυνατότητα μεταβολής των παραμέτρων του, δυναμικής εξέλιξης και αντίδρασης σε εξωτερικές δυνάμεις.
- Η παρούσα διατριβή δείχνει ότι χρησιμοποιώντας απλούς μικροϋπολογιστές, είναι δυνατόν να λύσουμε από πλευράς clique number μεγάλα προβλήματα σε μερικά μόνον δευτερόλεπτα. Είναι επίσης γνωστό ότι υπάρχουν προβλήματα του πραγματικού κόσμου στα οποία τα γραφήματα που προκύπτουν είναι της τάξης των εκατομμυρίων κόμβων με αποτέλεσμα να μην χωράνε στη μνήμη ενός μικροϋπολογιστή. Και είναι επίσης βεβαιο ότι θα μπορούσε να βρεθεί ένα πρόβλημα – απλά αυξάνοντας τις απαιτήσεις – ακόμα μεγαλύτερο σε κάθε περίπτωση.
- Η αίσθηση που βγαίνει από την μελέτη του θέματος είναι πάνω απ' όλα, ότι με ένα κατάλληλα οργανωμένο Σύστημα δεν θα υπήρχε «λογικού» μεγέθους πρόβλημα που να μην μπορεί να αντιμετωπιστεί, πολύ δε περισσότερο από την μελλοντική υπολογιστική ισχύ.
- Από πλευράς μεθόδου διάσπασης γραφήματος, στην παρούσα διατριβή επελέγη για λόγους γενικότητας η λεξικογραφική διάταξη των κόμβων ενώ σε δεύτερο επίπεδο η διάταξη σε αύξουσα σειρά της πυκνότητας των κόμβων που αποδεδειγμένα μειώνει τον χρόνο επίλυσης. Ωστόσο

παραμένει ανοικτό το πρόβλημα της βέλτιστης ταξινόμησης των κόμβων ίσως με κάποιο κριτήριο που θα στηρίζεται στην εσωτερική σχέση ανάμεσα στους κόμβους.

- Η Boolean μέθοδος όπως φαίνεται και από τις μετρήσεις σε συγκεκριμένο μέγεθος γραφήματος, ενώ αυξανομένης της πυκνότητας μεγαλώνει και ο αναμενόμενος χρόνος της, από ένα σημείο πυκνότητας και πάνω παρουσιάζει ραγδαία μείωση του χρόνου απόκρισης. Το φαινόμενο αυτό δεν συμβαίνει εξίσου έντονα στο κατανεμημένο Σύστημα, γεγονός που αποδίδεται στην μέθοδο διάσπασης η οποία παράγει μεγάλο αριθμό πυκνών γραφημάτων που αποτελούνται από τους ίδιους κάθε φορά κόμβους και ακμές (το ίδιο δηλαδή γράφημα).
- Για μιά ακόμη φορά, το μεγαλύτερο πρόβλημα που χρήζει άμεσης επίλυσης είναι η ήδη αναφερθείσα κωδικοποίηση των υποπροβλημάτων και ο τρόπος καταχώρησης αυτών έτσι ώστε η επίλυση οποιουδήποτε από αυτά να καταλήγει είτε σε απόρριψη από τους κανόνες του συστήματος ή σε επίλυση μία μόνο φορά στη διαδικασία. Ίσως το πρόβλημα αυτό να είναι αρκετά σημαντικό στην διανεμημένη υπολογιστική ισχύ και ιδιαίτερα στην αντιμετώπιση σκληρών προβλημάτων.

Βιβλιογραφία

ΒΙΒΛΙΟΓΡΑΦΙΑ

[Abello, Pardalos, Resende, 1998] J. Abello, P. M. Pardalos, M. G. C. Resende, On Very Large Maximum Clique Problems (Extended Abstract), Proceedings of "Algorithms and Experiments" (ALEX98), R. Battiti and A. A. Bertossi eds., pp. 175-183, Trento, Italy, 1998.

[Abello, Pardalos, Resende, 1999] J. Abello, P. M. Pardalos, M. G. C. Resende, On Maximum Clique Problems in Very Large Graphs, DIMACS series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1999.

[Acacio, Canovas +,2002] M. Acacio, O. Canovas, J.M. Garcia, P.E. Lopez-de-Teruel, MPI-Delphi: an MPI implementation for visual programming environments and heterogeneous computing, Future Generation Computer Systems 18 (2002), 317-333, Elsevier Science, 2002.

[Adleman, McCurley, 1994] Leonard M. Adleman and Kevin S. McCurley, Open Problems in Number Theoretic Complexity II, Proceedings of ANTS-I, LNCS 877, 1994, pp.291-322, 1994.

[Aho, Hopcroft, Ullman, 1974] Aho A., Hopcroft J., Ullman J., The Design and analysis of computer algorithms, Addison-Wesley Publ. Company, 1974.

[Ahuja, Dozzi, Abourizk, 1994] Hira N. Ahuja, S.P. Dozzi, S.M. Abourizk, PROJECT MANAGEMENT Techniques in Planning and Controlling Construction Projects, John Wiley & Sons, 1994.

[Ahmad, 1997] Ishfaq Ahmad, Express versus PVM: A performance comparison, Parallel Computing 23(1997) 783-812, 1997.

[AKL, 1999] Selim G. AKL, Parallel Computation, Models and Methods, Prentice-Hall Inc. 1999.

[Akl, 2000] S. Akl, The Design of Efficient Parallel Algorithms, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Alasdair, Bruce +, 1994] R. Alasdair, A. Bruce, J. Mills, A.G. Smith, Chimp version 2.0 interface, Technical report EPCC-KTP-CHIMP-V2,IFACE, Edinburgh Parallel Computing Centre, Univ. of Edinburgh, Feb. 1994

[Alasdair, Bruce +, 1994b] R. Alasdair, A. Bruce, J. Mills, A.G. Smith, CHIMP/MPI User Guide, EPCC-KTP-CHIMP-V2-USER 1.2, Edinburgh Parallel Computing Centre, June 1994.

[Allan, 1992] R.J.Allan, Fortnet V4.0: The parallel Programming Software, Technical Report, Daresbury Laboratory, 1992.

[Ananth, Kumar, Pardalos, 1993] Grama Y. Ananth, Vipin Kumar, Panos Pardalos, Parallel Processing of Discrete Optimization Problems, In Encyclopedia of Microcomputers, Vol.13 (1993), pp. 129-147, Marcel Dekker Inc., N.Y., 1993.

[Anderson, Culler, Patterson, 1995] Thomas E. Anderson, David E. Culler, David E. Patterson, and the NOW team, A Case for NOW (Networks of Workstations, IEE Micro, Feb. '95.

[Arbab, 1996] F. Arbab, The IWIM Model for Coordination of Concurrent Activities, Coordination Languages and Models Vol. 1061 of Lecture Notes in Comp. Science, pp.34-56, Springer-Verlag, April 1996.

[Arthur, Nelson, 1993] Trey Arthur, Michael Nelson, Intel NX to PVM3.2 Message Passing Conversion Library, NASA Technical Memorandum 109038, Oct. 1993.

[ASCI, 2001] Accelerated Strategic Computing Initiative, http://www.llnl.gov/asci/, http://www.sandia.gov/ASCI/, 2001.

[Auguston, Minker, 1970] J. Auguston, J. Minker, An analysis of some graph theoretical cluster techniques, Journal of ACM Vol. 17:517-588, 1970.

[Baase, Gelder, 2000] Sara Baase, Allen Van Gelder, Computer Algorithms, Introduction to Design & Analysis, Addison Wesley, 2000.

[Babel, Tinhofer, 1990] L. Babel, G. Tinhofer, A branch and bound algorithm for the maximum clique problem, ZOR-Methods and Models of Operations Research, Vol. 34:207-217, 1990.

[Bal, Bhoedjang +, 2000] H.Bal, R.Bhoedjang, R.Hofman, C.Jacobs, T.Kielmann, J.Maassen, R.Nieuwpoort, J.Romein, L.Renambot, T.Ruhl, R.Veldema, K.Verstoep, A.Baggio, G.Ballintijn, I.Kuz, G.Pierre, M.Steen, A.Tanenbaum, G.Doornbos, D.Germans, H.Spoelder, E.Baerends, S.Gisbergen, H.Afsermanesh, D.Albada, A.Belloum, D.Dubbeldam, Z.Hendrikse, B.Hertzberger, A.Hoekstra, Kamil Iskra, Drona Kandhai, Dennis Koelma, F.Linden, B.Overeinder, P.Sloot, P.Spinnato, D.Epema, A.Gemund, P.Jonker, A.Radulescu, C.Reeuwijk, H.Sips, P.Knijnenburg, M.Lew, F.Sluiter, L.Wolters, H.Blom, C.Laat, A.Steen, The Distributed ASCI Supercomputer Project, ACM Special Interest Group, Operations Systems Review, Vol. 34, No 4, Oct. 2000.

[Bal, Kaashoek, Tanenbaum, 1992] H. Bal, M. Kaashoek, A. Tanenbaum, Orca : a language for parallel programming of distributed systems, IEEE Transactions on Software engineering, 18(3):190-205, 1992.

[Bal, Plaat +, 1999] H. Bal, A. Plaat, T. Kielmann, J. Maassen, R. Nieuwpoort, R. Veldema, Parallel Computing on Wide-Area Clusters: the Albatross Project, Extreme Linux Workshop, p. 20-24, Monterey, CA, June 1999.

[Balas, Yu, 1986] E. Balas, C. Yu, Finding a maximum clique in an arbitrary graph, SIAM J. Comp., Vol.14:1054-1068, 1986.

[Baldeschwieler, Blumofe +, 1996] J. Baldeschwieler, R. Blumofe, E. Brewer, ATLAS: An Infrastructure for Global Computing, Proceedings of the Seventh

ACM SIGOPS, European Workshop on System Support for Worldwide Applications, 1996.

[Basney, Livny, 1999] Jim Basney, Miron Livny, Deploying a High Throughput Computing Cluster, In High Performance Cluster Computing, Rajkumar Buyya, Editor, Vol. 1, Chapter 5, Prentice Hall PTR, May 1999.

[Battiti, Protasi, 1995] R. Battiti, M. Protasi Reactive Local Search for the Maximum Clique Problem, Algorithmica 2000, Preliminary version: Technical Report TR-95-052, ICSI, 1947, Berkeley, Calif., 1995.

[Beck, Dongarra, +, 1999] M. Beck, J. Dongarra, G. Fagg, G. Geist, P. Gray, J. Kohl, M. Migliardi, K. Moore, T. Moore, P. Papadopoulos, S. Scott, V. Sunderam, HARNESS: a next generation distributed virtual machine, Future Generation Computer Systems 15 Issues 5-6 p.571-582, Oct. 1999.

[Ben-Ari, 1990] M. Ben-Ari, Principles of Concurrent and Distributed Programming, Prentice Hall International Series in Com. Science, 1990.

[Bjornson, Carriero +, 1988] Robert Bjornson, Nicholas Carriero, David Gelernter, Jerrold Leichter, Linda, the Portable Parallel, Technical Report 520, Yale University Department of Computer Science, Jan. 1988.

[Blazewicz, Drozdowski, Ecker, 2000] J. Blazewicz, M. Drozdowski, K. Ecker, Management of Resources in Parallel Systems, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Blelloch, 1995] Guy E. Blelloch, NESL: A Nested Data-Parallel Language (Version 3.1) September 19, 1995 CMU-CS-95-170 (Updated version of CMU-CS-92-103, January 1992, and CMU-CS-93-129, April 1993), School of Computer Science Carnegie Mellon University, 1995.

[Bohn, Lamont, 2002] Christopher A. Bohn, Gary B. Lamont, Load Balancing for heterogeneous clusters of PCs, Future Generation Computer Systems 18 (2002), 389-400, Elsevier Science, 2002.

[Bolosky, Draves +, 1997] J. Bolosky, R. Draves, R. Fitzgerald, C. Fraser, M. Jones, T. Knoblock, R. Rashid, Operating System Directions for the Next Millennium, Workshop on Hot Topics in Operating Systems 1997: 106-110, 1997.

[Bomze, Budinich +, 1999] Immanuel M. Bonze, Marco Budinich, Panos M. Pardalos, Marcello Pellilo, The Maximum Clique Problem, In D.-Z. Du and P. M. Pardalos, editors, Handbook of Combinatorial Optimization, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.

[Bron, Kerbosch, 1973] Coen Bron and Joep Kerbosch, Algorithm 457, Finding All Cliques of an Undirected Graph, Comm. ACM, Vol. 16 (9), pp. 575 – 577, 1973.

[Burns, Daoud, Vaigl, 1994] Greg Burns, Raja Daoud, James Vaigl, LAM: An Open Cluster Environment for MPI, Ohio Supercomputer Center, 1994.

[Burns, Daoud, 1995] Greg Burns, Raja Daoud, Robust MPI Message Delivery with Guaranteed Resources, MPI Developers Conference, University of Notre Dame, 1995.

[Butler, Lusk, 1994] Ralph M. Butler, Ewing L. Lusk, Monitors, Messages and Clusters: The p4 Parallel Programming System, (Argonne National Laboratory of Mathematics and Computer Science Division, p362-0493), Parallel Computing 20(4): 547-564, April 1994.

[Camiel, London +, 1997] N. Camiel, S. London, N. Nisan, O. Regev, The POPCORN Project: Distributed Computation over the Internet in Java, 6th International World Wide Web Conference, April 1997.

[Carraghan, Pardalos, 1990] R. Carraghan, P. Pardalos, An exact algorithm for the maximum clique problem, Oper. Res. Lett., Vol. 9:375-382, 1990.

[Carver, Lesser, 1992] Norman Carver and Victor Lesser, The Evolution of Blackboard Control Architectures, Expert Systems with Applications, Special issue on The Blackboard Paradigm and its Applications, 1992.

[Casavant, Braun, Kaliannan +, 2001] Thomas L. Casavant, Terry A. Brown, Sureshkumar Kaliannan, Todd E. Scheetz, Kyle J. Munn, Clayton L. Birkett, A Parallel/Distributed architecture for hierarchically heterogeneous web-based cooperative applications, Future Generation Computer Systems 17 (2001), 783-793, Elsevier Science, 2001.

[Cencek, Komasa, Rychlewski, 2000] W. Cencek, J. Komasa, J. Rychlewski, High-performance Computing in Molecular Sciences, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Chandy, Kesselman, 1993] K. Chandy, C. Kesselman, CC++: A declarative concurrent object-oriented programming notation, In Research Directions in Concurrent Object-Oriented Programming, MIT Press, 1993.

[Cheeseman, Kanefsky, Taylor, 1991] Peter Cheeseman, Bob Kanefsky, William M. Taylor, Where the *Really* Hard Problems Are, In J. Mylopoulos and R. Reiter, editors, Proceedings of IJCAI-91, pages 331-337, San Mateo, CA, Morgan Kaufmann, 1991.

[Clearwater, Hogg +, 1992] Scott H. Clearwater, Tad Hogg, Bernardo A. Hubberman, Cooperative Problem Solving, Computation: The Micro and the Macro View, B.A. Hubberman ed., World Scientific, pp. 33-70, 1992.

[Cobham, 1964] A.Cobham, The intrinsic computational difficulty of functions, In Yehoshua Bar-Hillel editor, Proceedings of the 1964 International Congress

for Logic, Methodology, and Philosophy of Science, p. 24 – 30, Elsevier/North-Holland, 1964.

[Cook, 1971] S.A. Cook, The Complexity of Theorem Proving Procedures, Proc. 3rd ACM Symp. On the theory of computing , ACM (1971), 151-158, 1971.

[Cook, 1991] Stephen Cook, Computational complexity of higher type functions, Proceedings of the International Congress of Mathematicians Ichiro Satake editor, Japan, pages 55 – 69, Springer Verlag, 1991.

[Cook, 2000] Stephen Cook University of Toronto, The P versus NP Problem, Official Problem Description for the millennium prize problems by Clay Mathematics Institute, http://www.claymath.org/prize_problems/, 2000

[Correa, Ferreira, 1995] Ricardo Correa, Afonso Ferreira, Parallel Best-First Branch-and-Bound in Discrete Optimization: a Framework, DIMACS Techical Report 95-03 (in Solving Combinatorial Problems in Parallel I: Methods, Springer-Verlag), 1995.

[Cormen, Leiserson, Rivest, 2000] Cormen T., Leiserson C., Rivest R., Introduction to Algorithms, The MIT Press, 2000.

[Corno, Prinetto +, 1995] Fulvio Corno, Paolo Prinetto, Matteo Sonza Reondra, Using Symbolic Techniques to find the Maximum Clique in Very Large Sparse Graphs, ED&TC'95: IEEE European Design and Test Conference, Paris, March 1995.

[Crescenzi, Kann, 1998] Pierluigi Crescenzi, Viggo Kann, How to find the best approximation results – a follow-up to Garey and Johnson, in ACM SIGACT news, December 1998.

[Csikor, Fodor +, 2001] F. Csikor, Z. Fodor, P. Hegedus, V.K. Horvath, S.D. Katz, A. Piroth, The PMS Project: Poor man's supercomputer, Computer Physics Communications 134 (2001) 139-149, 2001.

[Culler, Arpaci-Dusseau, Chun +, 1997] David E. Culler, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Brent Chun, Steven Lumetta, Alan Mainwaring, Richard Martin, Chad Yoshikawa, Frederick Wong, Parallel Computing on the Berkeley NOW, JSPP '97 (9th Joint Symposium on Parallel Processing), Kobe, Japan, 1997.

[Czyzyk, Mesnier, More, 1996] J. Czyzyk, M. Mesnier, J. More, The Network-Enabled Optimization System (NEOS) Server, Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996.

[Deng, Korobka, 2001] Yuefan Deng, Alex Korobka, The performance of a supercomputer built with commodity components, Parallel Computing 27 Issues 1-2, Jan. 2001.

[De Pietro, 2000] G. De Pietro, Multimedia Applications for Parallel and Distributed Systems, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[DIMACS, 1992] Clique and Coloring Problems, A Brief Introduction, with Project Ideas, NP Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability The Second DIMACS Implementation Challenge: 1992-1993, Last Revision December 1992, ftp://dimacs.rutgers.edu/pub/challenge, 1992.

[DIMACS, 1993] Clique and Coloring Problems Graph Format, NP Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability The Second DIMACS Implementation Challenge: 1992-1993, Last Revision May, 1993 ftp://dimacs.rutgers.edu/pub/challenge, 1993.

[Δημητράκος, 1964] Δ. Δημητράκος, Μέγα Λεξικόν της Ελληνικής Γλώσσης, Εκδόσεις ΔΟΜΗ, Αθήνα 1964.

[Dongarra, Geist, Kohl +, 1998] Jack Dongarra, Al Geist, James Arthur Kohl, Philip M. Papadopoulos, Vaidy Sunderam, HARNESS: Heterogeneous Adaptable Reconfigurable NEtworked SystemS, Conference on High Performance Distributed Computing, Chicago, IL, July 1998.

[Douglis, Kaashoek +, 1991] F. Douglis, M. Kaashoek, J. Ousterhout, Tanenbaum, A Comparison of Two Distributed systems: Amoeba and Sprite, Computing Systems vol. 4, no 3, pp. 353 – 384, Dec. 1991.

[Durfee, 1992] Edmund H. Durfee, What your Computer Really Needs to Know, You Learned in Kindergarten, Proceedings of the tenth National Conference on Artificial Intelligence, pp. 858-864, July 1992.

[Durfee, Lesser, Corkill, 1995] Edmund H. Durfee, Victor R. Lesser, Daniel D. Corkill, Trends in Cooperative Distributed Problem Solving, IEEE Transactions on Knowledge and Data Engineering, KDE-1(1):63-83, March 1989, Reprint 1995.

[Edmonds, 1965] Jack Edmonds, Minimum partition of a matroid into independent subsets, J. Res. Nat. Bur. Standards Sect. B, 69:67 - 72, 1965.

[Eicken, Culler, +, 1992] T. von Eicken, D. Culler, S. Goldstein, K. Schauser, Active Messages: a Mechanism for Integrated Communication and Computation, Proceedings of the 19th Int'l Symp. on Computer Architecture, Gold Coast, Australia May 1992.

[Epema, Livny +, 1996] D. H. J Epema, Miron Livny, R. van Dantzig, X. Evers, Jim Pruyne, A Worldwide Flock of Condors : Load Sharing among Workstation Clusters, Journal on Future Generations of Computer Systems, Volume 12, 1996.

[Fagg, Dongarra, Geist, 1997] G. Fagg, J. Dongarra, A. Geist, Heterogeneous MPI Application Interoperation and Process Management under PVMPI,

University of Tennessee Computer Science Technical Report, CS-97, June 1997, Submitted to the Euro PVM-MPI Conference, Cracow, Poland, November 1997.

[Fagg, Moore, Dongarra, 1999] G. Fagg, K. Moore, J. Dongarra, Scalable networked information processing environment (SNIPE), International Journal on Future Generation Computer Systems, 15:595--605, 1999.

[Fagg, Bukovsy, Dongarra, 2001] Graham Fagg, Antonin Bukovsy, Jack Dongarra, HARNESS and fault tolerant MPI, Parallel Computing 27 Issue 11, pp.1479-1495, Oct. 2001.

[Feige, Krauthgamer, 2000] Uriel Feige and Robert Krauthgamer, Finding and cerifying a large hidden clique in a semi-random graph, Weizmann Institute Technical Report#MCS99-04, January 1999, Random Structures and Algorithms 16(2): 195-208, 2000.

[Feo, Resende, 1995] Tomas A. Feo, Mauricio G.C. Resende, Greedy Randomized Adaptive Search Procedures, J. of Global Optimization, 6:109 - 133, 1995.

[Ferris, Mesnier, More, 1998] Michael C. Ferris, Michael P. Mesnier,Jorge J. More, NEOS and CONDOR: Solving Optimization Problems Over the Internet, Argonne National Laboratory Technical Report Preprint ANL/MCS-P708-0398, 1998.

[Festa, Resende, 2001] Paola Festa and Mauricio G.C. Resende, GRASP: An annotated bibliography, AT&T Labs Research Technical Report, 2001.

[Flower, Kolawa, Bharadwaj, 1991] J. Flower, A. Kolawa, S. Bharadwaj, The Express way to distributed processing, Supercomputing review, p.54-55, 1991.

[Flower, Kolawa, 1994] Jon Flower, Adam Kolawa, Express is not just a Message Passing System Current and Future Directions in Express, Parallel Computing 20(4): 597-614, 1994.

[Foster, Kesselman, Tuecke, 1994] Ian Foster, Carl Kesselman, Steven Tuecke, The Nexus Task-parallel Runtime System, Proceedings of the First International Workshop on Parallel Processing, 1994.

[Foster, Chandy, 1995] I. Foster, K. Chandy, Fortran M: a language for modular parallel programming, Journal of Parallel and Distributed Computing, 25(1), 1995.

[Foster, Kesselman, Tuecke, 1996] Ian Foster, Carl Kesselman, Steven Tuecke, The Nexus Approach to Integrating Multithreading and Communication, Journal of Parallel and Distributed Computing 37, 70-82, 1996.

[Foster, Kohr +, 1996] I. Foster, D. Kohr, R. Krishnaiyer, A. Choudhary, Double Standards: Bringing Task Parallelism to HPF via the Message Passing Interface, Proceedings of Supercomputing '96, 1996.

[Foster, Kesselman, 1998] Ian Foster and Carl Kesselman, eds, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufman, 1998.

[Foster, Geisler, Gropp +, 1998] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvanthukal, S. Tuecke, A wide-area implementation of the message passing interface, Parallel Computing, 24(11), 1998.

[Foster, Kesselman, 1999] Ian Foster, Carl Kesselman, The Globus Project: a status report, Future Generation Computer Systems 15 (1999), 607-621, 1999.

[Foster, 2000] Ian Foster, Languages for Parallel Processing, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Freisleben, Kielmann, 1995] Bernd Freisleben and Thilo Kielmann, Approaches to Support Parallel Programming on Workstation Clusters: A Survey, Informatik-Bericht Nr. 95-01, 1995.

[Fujimoto, 2000] Richard M. Fujimoto, Parallel and Distributed Simulation Systems, John Wiley & Sons, 2000.

[Gannon, Diwan, Johnson, 1996] D. Gannon, S. Diwan, E. Johnson, HPC++ and the Europa Call Reification Model, ACM Applied Computing Review vol. 4(1), Spring 1996.

[Garcia, Ferreira, Guedes, 2000] J. Garcia, P. Ferreira, P. Guedes, Parallel Operating Systems, Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Garey, Johnson, 1979] Garey M., Johnson D., Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences), W H Freeman & Co., 1979, reprint 1997.

[Geist, Heath +, 1990] G.A.Geist, M.T.Heath, B.W.Peyton, P.H.Worley, PICL: A portable instrumented communications library, Technical report TM-11130, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1990.

[Geist, Heath +, 1992] G.A.Geist, M.T.Heath, B.W.Peyton, P.H.Worley, a user's guide to PICL: A portable instrumented communications library, Technical report ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1992.

[Geist, Beguelin, Dongarra, +, 1994] A. Geist, A. Beguelin,J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, PVM:Parallel Virtual Machine,A User's Guide and Tutorial for Networked Parallel Computing, The MIT Press, 1994.

[Geist, Kohl, Papadopoulos, 1996] G.A. Geist, J. A. Kohl, P. M. Papadopoulos, PVM and MPI: a Comparison of Features, Calculateurs Paralleles 8(2), pp. 137-150, June 1996.

[Geist, Kohl +, 1997] G.A. Geist, J. A. Kohl, P.M. Papadopoulos, S.L. Scott, Beyond PVM 3.4: What We've Learned, What's Next, and Why, Invited talk to European PVM and MPI user's Group Meeting, Krakow, Polland, Nov. 1997.

[George, Hagedorn, Devaney, 2000] W. George, J. Hagedorn, J. Devaney, "IMPI: Making MPI Interoperable", with appendix I by IMPI Steering Committee, "IMPI: Interoperable MessagePassing Interface", Protocol Version 0.0, January, 2000, http://impi.nist.gov/IMPI/, Journal of Research of the National Institute of Standards and Technology, 2000.

[Glover, 1993] F. Glover, A User's Guide to Tabu Search, Annals of Operations Research 41, p. 3-28, 1993.

[Goldberg, 1989] D. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[Golubchik, Muntz, 2000] L. Golubchik, R. Muntz, Parallel Database Systems and Multimedia Servers , In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Gray C., Larson, 2000] Clifford F. Gray, Eric W. Larson, PROJECT MANAGEMENT

The Managerial Process, McGraw-Hill Higher Education, 2000.

[Gray P., Sunderam, 1999] Developing technologies for broad-network concurrent programming, Journal of Systems Architecture 45, pp.1279-1291, 1999.

[Grimshaw, Wulf, 1996] A. Grimshaw, W. Wulf, Legion. Flexible Support for Wide-Area Computing, Proceedings of the 1996 7th SIGOPS European WorkShop, Ireland, 1996.

[Gropp, Smith, 1993] W. Gropp and B. Smith, Users manual for the Chameleon parallel programming tools, Report ANL-93/23, Argonne National Laboratory, Argonne, Illinois, 1993.

[Gropp, Lusk, Doss, Skjellum, 1996] W. Gropp and E. Lusk and N. Doss and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing 22(6):789-828, 1996.

[Gropp, Lusk, 1996] User's Guide for mpich, a portable Implementation of MPI, Ver. 1.2.2 by W. Gropp and E. Lusk, ANL/MCS-TM-ANL-96/6 Rev D., 1996.

[Gropp, Lusk, 1997] W. Gropp,E. Lusk, Why are PVM and MPI so different?, in Recent advances in parallel virtual machine and message passing interface, Proceedings of the 4th European PVM/MPI Users' Group Meeting , Lecture notes in Computer Science, Springer, November 1997.

[Grotschel, Lovasz, 1993] Martin Grotschel, Laszlo Lovasz, Combinatorial Optimization: A Survey, DIMACS Technical Report 93-29, 1993.

[Harary, Ross, 1957] F. Harary, I. Ross, A procedure for clique detection using the group matrix, Sociometry Vol. 20:205-215, 1957.

[Harary, 1972] Frank Harary, Graph Theory, Addison-Wesley Publishing Company, Reading, Massachusetts, 1972.

[Harrison, 1991] R. Harrison, Portable tools and applications for parallel computers, International Journal of Quantum Chemistry, 40(847), 1991.

[Hayes, 1998] Brian Heyes, Collective Wisdom, The American Scientist Vol 86 No 2 p118-122, 1998.

[Hempel, Steppe, Supakov, 1993] R. Hempel, H. Steppe, A. Supakov, PARMACS 6.0 library interface specifications, GMD, St. Augustin, 1993.

[Hill, Larus, Wood, 1996] Mark Hill, James Larus, David Wood, ParallelComputer Reasearch in the Winsconsin Wind Tunnel Project, NSF Conference on Experimental Research in Computer Systems, June 1996.

[Hogg, Huberman, 1993] Tad Hogg, Bernardo A. Huberman, Better than the Best: The Power of Cooperation, L. Nadel and D. Stein eds., SFI 1992, Lectures in Complex Systems, 163-184, Addison-Wesley 1993.

[Hogg, Williams, 1993] Tad Hogg, Colin P. Williams, Solving the Really Hard Problems with Cooperative Search, in Proc. Of AAAI93, pp. 231-236, AAAI Press, 1993.

[Homer, Peinado, 1996] Steven Homer, Marcus Peinado, On the Performance of Polynomial-time CLIQUE Approximation Algorithms on Very Large Graphs, In D.S. Johnson and M.Trick eds., Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, American Mathematical Society, 1996, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1996.

[HPC++, 1995] The HPC++ Working Group, HPC++ White papers, Technical Report 95633, Center for research on parallel computation, 1995.

[Hromkovic, 2001] Juraj Hromkovic, Algorithmics for Hard Problems, Springer-Verlag, 2001.

[Hromkovic, 1997] J. Hromkovic, Computational Complexity and Parallel Computing : The Application of Communication Complexity in Parallel Computing Springer Verlag, 1997.

[Iskra, Belleman, Albada +, 2001] K. Iskra, R. Belleman, G. Albada, J. Santoso, P. Sloot, H. Bal, H. Spoelder, M. Bubak, The Polder Computing Environment,

a system for interactive distributed simulation, Concurrency – Practice and Experience, 2001;0:1-40, 2001.

[Johnson, 1988] E. Johnson, Completing a MIMD multiprocessor Taxonomy, Computer Architecture News 16 p 44-47, 1988.

[Jensen, 1997] Kurt Jensen, Coloured Petri Nets, Vol.1,2,3, Springer, 1997.

[Juels, Peinado, 1998] A. Juels, M. Peinado, Hiding Cliques for Cryptographic Security, Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 1998) pp.678-684, ACM, New York, 1998.

[Karmarkar, 1984] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica, 4, pp. 373-395, 1984.

[Karp, 1972] R. M. Karp, Reducibility among Cobinatorial Problems, pp. 85-103 in Complexity of Computer Computations, ed. R.E. Miller and J.W. Thatcher, NY: Plenum Press, 1972.

[Kaye, 2000] Richard Kaye, Minesweeper is NP-complete, Mathematical Intelligencer, 2000.

[Keleher, Dwarkadas, Cox +, 1994] P. Keleher, S. Dwarkadas, A. Cox, W. Zwaenepoel, TreadMarks: Distributed Shared Memory on standard workstations and operating systems, Proc. Of the Winter '94 Usenix Conference, p. 115-131, 1994.

[Khachiyan, 1979] L. G. Khachiyan, A polynomial algorithm for linear programming, Dokl. Akad. Nauk SSSR, 224, pp.1093-1096, 1979, English translation Soviet Math. Doklad 20, pp. 191-194, 1979.

[Knuth, 1998] Donald E. Knuth, The Art of computer programming: Vol.1 Fundamental Algorithms, Addison Wesley Longman, 1998.

[Knuth, 1998] Donald E. Knuth, The Art of computer programming: Vol. 2 Seminumerical Algorithms, Addison Wesley Longman, 1998.

[Knuth, 1998] Donald E. Knuth, The Art of computer programming: Vol. 3 Sorting and Searching, Addison Wesley Longman, 1998.

[Kumaran, 2001] Ilango Kumaran, Jini Technology: An Overview, Prentice Hall PTR, 2001.

[Ladner, 1975] R.E. Ladner, On the structure of polynomial time reducibility, J.ACM, 22, pp. 155-171, 1975.

[Lehman, 1994] Li-wei H. Lehman, Integrating Zipcode and PVM: Towards a Higher-Level Message-Passing Environment, Mississippi State University Technical Report MSSU-EIRS-ERC-94-2, 1994.

[Lewis, Papadimitriou, 1981] H. R. Lewis and C. H. Papadimitriou, Elements of the Theory of Computation, Prentice-Hall Publishing Co., 1981.

[Litaize, Mzoughi, Rochange, Sainrat, 2000] D. Litaize, A. Mzoughi, C. Rochange, P. Sainrat, Architecture of Parallel and Distributed Systems, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Livny, Basney, Raman +, 1997] Miron Livny, Jim Basney, Rajesh Raman, Todd Tannenbaum, Mechanisms for High Throughput Computing, SPEEDUP Journal, Vol. 11, No. 1, June 1997.

[Loukakis, Tsouros, 1981] E. Loukakis, C. Tsouros, A depth first search algorithm to generate the family of maximal independent sets of a graph lexicographically, Computing Vol. 27: 249-266, 1981.

[Loukakis, Tsouros, 1982] E. Loukakis, C. Tsouros, Determining the number of internal stability of a graph, Int. Journal Comp. Math., Vol.11: 207-220, 1982.

[Loveman, 1993] D. Loveman, High Performance Fortran, IEEE Parallel and Distributed Technology 1 p. 25-42, 1993.

[Malony, 2000] A. Malony, Tools for Parallel Computing: A Performance Evaluation Perspective, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Manke, 2001] J.W. Manke, Guest Editorial: Parallel computing in aerospace, Parallel Computing 27(2001) 329-336, 2001.

[Marksteiner, 1996] P. Marksteiner, High-performance computing – an overview, Computer Physics Communications 97(1996) 16-35, 1996.

[Mavrommatis, Panayiotopoulos J-C., 2002] G. Mavrommatis, J-C. Panayiotopoulos, Risk Transportation Via a Clique Number Problem Formulation, European Journal of Operational Research, accepted 2002.

[Miller G., 1976] Gary L. Miller, Riemann's hypothesis and tests for primality, Journal of Computer and System Science, 13:300-317, 1976.

[Miller R., Boxer, 2000] Miller R., Boxer L., Algorithms Sequential & Parallel, Prentice Hall, 2000.

[Manna, 1974] Zohar Manna, Mathematical Theory of Computation, McGraw-Hill, 1974.

[Manoharan, Shanmuganathan, 1999] S. Manoharan, S. Shanmuganathan, A comparison of search mechanisms for structural optimization, Computers and Structures 73 (1999) 363-372,1999.

[Mertens, 1998] Stephan Mertens, Phase Transition in the Number Partitioning Problem, Physical Review Letters Vol 81 No 20,The American Physical Society, 1998.

[Morin, 1998] P. Morin, Coarse-Grained Parallel Computing on Heterogeneous Systems, SAC'98, the 1998 ACM Symposium on Applied Computing, Atlanta, Georgia, U. S. A., 1998.

[MPI-forum, 1994] Message Passing Interface Forum, MPI: A Message-Passing Interface standard, The International Journal of Supercomputer Applications and High Performance Computing, 8, 1994.

[MPI-forum, 1995] Message Passing Interface Forum, MPI: A Message-Passing Interface standard (Version 1.1), Technical report, 1995, **http://www.mpi-forum.org**, 1995.

[MPI-forum, 1997] MPI-2: Extensions to the Message-Passing Interface, Technical report, **http://www.mpi-forum.org**, 1997.

[Mohring ed., 1997], Editor Rolf H. Mohring, Graph-Theoretic Consepts in Computer Science 23rd International Workshop, WG'97, Berlin, Germany, June 18-20, 1997 Proceedings, Springer-Verlag, 1997.

[Nabrzyski, Stroinski, Weglarz, 2000] J. Nabrzyski, M. Stroinski, J. Weglarz, Networking Aspects of Distributed and Parallel Computing, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Natrajan, Humphrey, Grimshaw, 2001] A. Natrajan, M. Humphrey, A. Grimshaw, Grids: Harnessing Geographically-Separated Resources in a Multi-Organisational Context, Presented at High Performance Computing Systems, June 2001.

[Neary, Christiansen +, 1999] Michael O. Neary, Bernd O. Christiansen, Peter Cappello, Klaus E. Schauser, Javelin: Parallel computing on the internet, Future Generation Computer Systems 15 (1999), 659-674, Elsevier Science, 1999.

[Neumann, 1953] J.von Neumann, Acertain zero-sum two-person game equivalent to the optimal assignment problem, In H.W.Kahn and A.W. Tucker editors, Contributions to the Theory of Games II, Princeton Univ. Press, 1953.

[Nevin, 1996] Nick Nevin, The performace of LAM 6.0 and MPICH 1.0.12 on a Workstation Cluster, Ohio Supercomputer Center, Technical Report OSC-TR-1996-4, 1996.

[Nieplocha, Harrison, Littlefield, 1996] Global Arrays: A nonuniform memory access programming model for high-performance computers, The Journal of Supercomputing, 10:197-220, 1996.

[Nieuwpoort, Maassen, Bal +, 1999] R. Nieuwpoort, J. Maassen, H. Bal, T. Kielmann, R. Veldema, Wide-Area Parallel Computing in Java. In ACM 1999 Java Grande Conference, Palo Alto, CA, June 1999.

[O'hEigeartaigh, Lenstra, Rinnooy Kan Editors, 1985] Edited by O'hEigeartaigh M., Lenstra J.K., Rinnooy Kan A.H.G., Combinatorial Optimization, Annotated Bibliographies, John Wiley & Sons, 1985.

[Ousterhout, Cherenson, Douglis +, 1988] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, B. Welch, The Sprite network Operating System, IEEE Computer, 21(2):23-36, Feb. 1988.

[Panayiotopoulos J-C., 1979] J-C. Panayiotopoulos, Stock Exchange and Black Economical Holes, presented at the International Meeting of ORSA/TIMS, Milwaukee, Session: Portofolio Theory and Applications, 1979.

[Panayiotopoulos J-C., 1981] J-C. Panayiotopoulos, Boolean Programming and Clique Number, The Journal of the Industrial Mathematics Society, Volume 31, Part 1, 1981.

[Panayiotopoulos J-C., 1989] J-C. Panayiotopoulos, Using Fifth Generation Tools for Solving the Clique Number Problem, Journal of Information and Optimization Sciences, Vol. 10 (1989), No. 3, pp. 535-555, Analytic Publishing Co., 1989.

[Panayiotopoulos J-C., 1992] J-C. Panayiotopoulos, White, Gray and Black Operational Holes: An Artificial Intelligence Approach, Journal of Information and Optimization Sciences, Vol. 13 (1992), No. 3, pp. 407-425, Analytic Publishing Co., 1992.

[Παναγιωτόπουλος Ι-Χ., 1990] Παναγιωτόπουλος Ι-Χ., Αρχές Προγραμματισμού - Basic, Εκδόσεις Σταμούλη, 1990.

[Παναγιωτόπουλος Ι-Χ., 1999] Παναγιωτόπουλος Ι-Χ., Προγραμματισμός σε περιβάλλον παραθύρων Windows 98/NT, Εκδόσεις Σταμούλη, 1999.

[Pancake, 2001] C. Pancake, Performance Tools for today's HPC: Are we addressing the right issues?, Parallel Computing 27(2001) 1403-1415, 2001.

[Papadimitriou, Steiglitz, 1998] Papadimitriou C., Steiglitz K., Combinatorial Optimization: Algorithms and Complexity, Dover Pubns, 1998.

[Papadimitriou, 1995] Papadimitriou C., Computational Complexity, Addison-Wesley Pub Co, 1995.

[Papadimitriou, 1997] Christos Papadimitriou, NP-Completeness: A Retrospective, ICALP 97 published by Springer LNCS, 1997.

[Pardalos, Rappe, Resende, 1999] Panos M. Pardalos, Jonas Rappe, Mauricio G.C. Resende, An Exact Parallel Algorithm for the Maximum Clique Problem, in "High Performance Algorithms and Software in nonlinear Optimization", R. De Leone et al. (eds.), Kluwer Academic Press, pp. 279-300, 1999.

[PBS, 2000] Portable Batch System, **http://pbs.mrj.com**, 2000.

[Pellilo, 2001] M. Pellilo, Heuristics for maximum clique and independent set, Encyclopedia of Optimization, C.A. Floudas and P.M. Pardalos eds., Kluwer Academic Publishers, Boston, MA, (in Press) 2001.

[Pennock, Stout, 1996] David M. Pennock, Quentin F. Stout, Exploiting a Theory of Phase Transitions in Three-Satisfiability Problems, Proceedings of the 13th

National Conference in Artificial Intelligence (AAAI-96), pp. 253-258, Portland, OR, USA, Aug 1996.

[Petitet, Casanova, Dongarra +, 2000] A. Petitet, H. Casanova, J. Dongarra, Y. Robert, R.C. Whaley, Parallel and Distributed Scientific Computing, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Pierce, Regnier, 1994] Paul Pierce and Greg Regnier, The Paragon implementation of the NX message passing interface, In *SHPCC 94*, 1994.

[Plateau, Trystram, 2000] B. Plateau, D. Trystram, Parallel and Distributed Computind: State-of-the-Art and Emerging Trends, In Blazewicz, Ecker, Plateau, Trystram Eds., Handbook on Parallel and Distributed Processing, Springer-Verlag, 2000.

[Plotnikov, 1996] Anatoly D. Plotnikov, Polynomial-Time Partition of a Graph into Cliques, Electronic Journal: Southwest Journal of Pure and Applied Mathematics, Vol. No. 1, pp. 16-29, 1996.

[Pratt, 1975] Vaughn Pratt, Every prime has a succinct certificate, SIAM Journal of Computing, 4:214-220,1975.

[Proud, 1999] Proud F. John, Master Scheduling, John Wiley & Sons, 1999.

[Resende, Pardalos, Eksioglu, 2001] Mauricio G.C. Resende, Panos M. Pardalos and Sandra Duni Eksioglu, Advanced Algorithmic Techniques for Parallel Computations with Applications, Correa R. et al. (eds.), Kluwer Academic Publishers, 2001.

[Riesen, Brightwell +, 1999] R. Riesen, R. Brightwell L. A..Fisk, T. Hudson, J. Otto, A. Maccabe, Cplant, Proceedings of the Second Extreme Linux WorkShop, 1999 USENIX Annual Technical Conference, Monterey, CA, 1999.

[Ridge, Becker, 1997] D. Ridge, D. Becker, P. Merkey, T. Sterling, Beowulf:Harnessing the Power of Parallelism in a Pile-of-PCs, In Proc. Of the 1997 IEEE Aerospace Conference, Vol.2.IEEE, p. 79 – 91, 1997.

[RM, 1995] Ada 95 Reference Manual (RM), Version 6.0, Intermetrics, Inc., January 1995, Revised international standard ISO/IEC 8652:1995, (http://lglwww.epfl.ch/Ada/LRM/9X/rm9x/rm9x-toc.html), 1995.

[Robertson, Seymour, 1995] N. Robertson and P. D. Seymour, Graph minors i – xiii, Journal of Combinatorial Theory B ,1983 – 1995, 1995.

[Robson, 1986] J. Robson, Algorithms for maximum independent sets, J. Algorithms, Vol. 7:425-440, 1986.

[RSA, 1999] http://www.rsasecurity.com/rsalabs/challenges/, 1999.

[Sarmenta, 1998] L.F.G. Sarmenta, Bayanihan: Web-Based Volunteer Computing Using Java, 2nd International Conference on World-Wide Computing and its Applications, Mar. 1998.

[Scales, Gharachorloo, 1997] D.J. Scales, K. Gharachorloo, Towards transparent and efficient software distributed shared memory, Proc. Of the 16th ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review 31, p.157-169, 1997.

[Schattler, Krenzien, 1997] U. Schattler, E. Krenzien, The parallel 'Deutschland-Model' – A message-passing version for distributed memory computers, Parallel Computing 23(1997) 2215-2276, 1997.

[Sheldon, 1998] Sheldon Tom, Encyclopaedia of Networking, Osborne/McGraw-Hill, 1998.

[Skillicorn, Talia, 1998] D. Skillicorn, D. Talia, Models and Languages for parallel Computation, ACM Computing Surveys Vol.30, No.2, p123-169, 1998.

[Skjellum, Smith, Doss +, 1994] A. Skjellum, S. Smith, N. Doss, A. Leung, M. Morari, The Design and Evolution of Zipcode, Parallel Computing 20(4): 565-596, 1994.

[Soriano, Gendreau, 1994] P. Soriano, M. Gendreau, Tabu Search Algorithms for the Maximum Clique Problem, Technical Report CRT-968, University of Montreal, 1994.

[Smith, 1999] Kate A. Smith, Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research, INFORMS Journal on Computing, Vol.11 No.1, Winter 1999.

[Squyres, McCandless, Lumsdaine, 1996] J. Squyres, B. McCandless, A. Lumsdaine, Object Oriented MPI: A Class Library for the Message Passing Interface, POOMA '96, THE 1996 PARALLEL OBJECT-ORIENTED METHODS AND APPLICATIONS CONFERENCE, Santa Fe, New Mexico, February 28 - March 1, 1996.

[Squyres, 1997] Jeffrey M. Squyres, PVM? Are you joking?" A paper for CSE 643: Principles of Parallel Computing Fall, 1997 Department of Computer Science and Engineering University of Notre Dame Notre Dame, 1997.

[Stroustrup, 1997] Bjarne Stroustrup, The C++ Programming Language, Addison-Wesley, 1997.

[Suganuma, Ogasawara +, 2000] T. Suganuma, T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, T. Nakatani, Overview of the IBM Java Just-in-Time Compiler, IBM SYSTEMS JOURNAL, VOL 39, NO 1, 2000.

[Sunderam, 1990] V. Sunderam, PVM:A framework for parallel distributed computing, Concurrency: Practice and Experience, 2(4):315-339, 1990.

[Taft, 1996] S. Tucker Taft, Intermetrics, Inc., Cambridge, MA USA , Programming the Internet in Ada 95, submitted to Ada Europe '96, 1996.

[Takagi, Matsuoka +, 1998] H. Takagi, S. Matsuoka, H. Nakada, S. Sekiguchi, M. Satoh, U. Nagashima, Ninflet: a Migratable Parallel Objects Framework using Java, Proceedings of the ACM 1998 Workshop on Java for High-Performance Network Computing, Palo Alto, CA, Feb. 1998.

[Tanenbaum, Renesse, Staveren +, 1990] A. Tanenbaum, R. van Renesse, H. van Staveren, G. Sharp, S. Mullender, J. Jansen, G. van Rossum, Experiences with the Amoeba Distributed Operating System, Communications of the ACM, 33(2):46--63, December 1990.

[Tarjan, 1972] R. Tarjan, Finding a maximum clique, Technical report 72-123 Comp. Science Dept., Cornell University, 1972.

[Tarjan, Trojanowski, 1977] R. Tarjan, A. Trojanowski, Finding a maximum independent set, SIAM J. Comput., Vol. 6:537-546, 1977.

[Tspaces, 1998] IBM Systems Journal, August 1998.

[Tsukiyiama, Ide, +, 1977] S. Tsukiyiama, M. Ide, H. Ariyoshi, I. Shirakawa, A New Algorithm for generating All the Maximal Independent Sets, SIAM J. Comput. Vol. 6:505-517, 1977.

[Turner, 1993] Roy M. Turner, The tragedy of the commons and distributed AI systems, Proceedings of the 12th International Workshop on Distributed Artificial Intelligence, pp. 379-390, PA, May 1993.

[Vaughan, Skjellum, Reese, Cheng, 1995] P. Vaughan, A. Skjellum, D. Reese, F-C. Cheng, Migrating from PVM to MPI, part I: The Unify System, Frontiers'95, The Fifth Symposium on the Frontiers of Massively Parallel Computation, McLean, Virginia, 1995.

[Vazhkudai, Syed, Maginnis, 2002] S. Vazhkudai, J. Syed, T. Maginnis, PODOS – The design and implementation of a performance oriented Linux cluster, Future generation computer systems 18 Issue 3, pp.335-352, Jan. 2002.

[Xavier, Iyengar, 1998] Xavier C., Iyengar S.S., Introduction to parallel Algorithms, Wiley Interscience, 1998.

Παραρτήματα

ΠΑΡΑΡΤΗΜΑΤΑ

1 Κώδικες C++

1.1 Διαχειριστής Προβλήματος(PM)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <share.h>

static char path[40]="h:";

static int word_length=32;
int send;
struct MITROO
{
int identity;
int node;
int bound;
};


int sortgraph(int a[][6001],int size,int order[6001],int degree[6001])
{
int i,j,minindex,mindegree,temp,dedges;

dedges=0;
for(i=1;i<=size;i++)
{
        degree[i]=0;order[i]=i;
        for (j=1;j<=size;j++)
                if (a[i][j] == 1) degree[i]++;
        dedges+=degree[i];
}
for(i=1;i<=size-1;i++)
{
        minindex=i;mindegree=degree[i];
        for (j=i+1;j<=size;j++)
        {
                if (mindegree > degree[j])
                {
                        mindegree=degree[j];minindex=j;
                }
        }
        temp=order[minindex];order[minindex]=order[i];order[i]=temp;
        temp=degree[minindex];degree[minindex]=degree[i];degree[i]=temp;
}
for (i=1;i<=size;i++)
        for (j=1;j<=i-1;j++)
                if (order[i] < order[j])
                        a[i][j]=a[order[i]][order[j]];
                else a[i][j]=a[order[j]][order[i]];
for (i=1;i<=size-1;i++) for (j=i+1;j<=size;j++) a[i][j]=a[j][i];
return dedges;

} //sortgraph


void sortsbgraph(int a[][6001],int sb[][6001],int level,int size)
{
```

```
int i,j,minindex,mindegree,temp, order[6001],degree[6001];

for(i=1;i<=size;i++)
{
        degree[i]=0;order[i]=i;
        for (j=1;j<=size;j++)
                if (a[sb[level][i]][sb[level][j]] == 1) degree[i]++;
}
for(i=1;i<=size-1;i++)
{
        minindex=i;mindegree=degree[i];
        for (j=i+1;j<=size;j++)
        {
                if ( degree[j] < mindegree )
                {
                        mindegree=degree[j];minindex=j;
                }
        }
        temp=sb[level][minindex];
        sb[level][minindex]=sb[level][i];
        sb[level][i]=temp;
        degree[minindex]=degree[i];
}

} //sortsbgraph


void Evangelos(unsigned int boo[2201],int *lastrow,int size,int LowBound)
{
int last,i,j,assoi;unsigned int a,help,norma,helper[2201];
int aporipteos[2201];

last=0;
for (i=1;i<=*lastrow;i++)
{
        aporipteos[i]=0;
        assoi=0;
        help=boo[i];
        for (j=1;j<=size;j++)
        {
                a= help & 01;
                if (a == 1) assoi++;
                help = help >> 1;
        }
        if (size - assoi <LowBound) aporipteos[i] = 1;
}
for (i=1;i<=*lastrow-1;i++)
        for (j=i+1;j<=*lastrow;j++)
        {
                norma=boo[i] & boo[j];
                if (norma==boo[i]) aporipteos[j]=1; else
                        if (norma == boo[j]) aporipteos[i]= 1;
        }
for (i=1;i<=*lastrow;i++)
        if (aporipteos[i]==0)
{
        last++;helper[last]=boo[i];
}
*lastrow=last;for(i=1;i<=last;i++) boo[i]=helper[i];

} //Evangelos
```

```
int heurlow(int graph[][6001],int size,int better[])
{
int i,j,bestcliq,kalo;

bestcliq=1;
better[bestcliq]=size;
for (i=size-1;i>=1;i--)
{
        kalo=1;
        for (j=1;j<=bestcliq;j++)
                if (graph[better[j]][i] ==0) { kalo=0;break; }
        if (kalo==1)
        {
                bestcliq++;better[bestcliq]=i;
        }
}
return bestcliq;
} //heurlow


int prakseis(int a[][33],int size,int LowBound,int better[453])
{
        int numterm,found,index,lastrow,mises,counter,tsou,seira[33],megal,idxmegal;
        unsigned int boo[2201],paragontes[33][2],pilot,temp;
        register i,j;

        numterm=0;
for (i=1;i<=size-1;i++)
{
        seira[i]=0;
        found=0;
        index=i+1;
        while ((found ==0) && (index <= size))
                if (a[i][index]==0) found=1; else index++;
                if (found==1)
                {
                        numterm++;
                        paragontes[numterm][1]=1;
                        paragontes[numterm][1]= paragontes[numterm][1] << (i -1);
                        paragontes[numterm][2]=0;
                        for (j=i+1;j<=size;j++)
                                if (a[i][j] ==0)
                                {
                                pilot=1;seira[i]++;
                                pilot= pilot << (j-1);
                                paragontes[numterm][2]=paragontes[numterm][2] | pilot;
                                }
                }
}
for (i=1;i<=numterm-1;i++)
        for(j=i+1;j<=numterm;j++)
                if (seira[i] < seira[j])
                {
                        temp=paragontes[i][1];paragontes[i][1]=paragontes[j][1];paragontes[j][1]=temp;
                        temp=paragontes[i][2];paragontes[i][2]=paragontes[j][2];paragontes[j][2]=temp;
                        tsou=seira[i];seira[i]=seira[j];seira[j]=tsou;
                }

if (numterm >0)
{
lastrow=2;boo[1]=0;boo[2]=0;
boo[1] = boo[1] | paragontes[1][1];
boo[2] = boo[2] | paragontes[1][2];
Evangelos(boo,&lastrow,size,LowBound);
```

```
for (counter=2;counter<=numterm;counter++)
{

        mises=lastrow;
        lastrow=lastrow*2;
        for (i=mises+1;i<=lastrow;i++)
                boo[i]=boo[i-mises];
        for (i=1;i<=mises;i++) boo[i]= boo[i] | paragontes[counter][1];
        for (i=mises+1;i<=lastrow;i++)          boo[i] = boo[i] | paragontes[counter][2];
        Evangelos(boo,&lastrow,size,LowBound);
}
}

else
{
        lastrow=1;boo[1]=0;
}


megal=-1;
for(i=1;i<=lastrow;i++)
{
        temp=boo[i];
        counter=0;
        for(j=1;j<=size;j++)
        {
                index=temp & 01;
                if (index==0)
                        counter++;
                temp = temp >> 1;
        }
        if (counter > megal)
        {
                megal=counter;idxmegal=i;
        }
}
if (megal >-1)
{
        temp=boo[idxmegal];counter=0;
        for(j=1;j<=size;j++)
        {
                index=temp & 01;
                if (index==0)
                {
                        counter++;
                        better[counter]=j;

                }
        temp = temp >> 1;
        }
}
return megal;

} //of prakseis
```

```
int menu(int graph[][6001],int *size)
{
        int tmp,choice,i,j;   char ch,fname[35],tp[4];float dens,s;
        FILE *fp;
        int solvers;
        MITROO rec[31];


printf("1...Load Graph from File\n2...Random Graph Generator\nOther...Exit\n\n  Your Choice:");
scanf("%d",&choice);while (ch=getchar()!='\n');

if (choice == 1)
{
        printf("Enter Full Pathname (Enter---> %s\\graph.txt):",path);ch=getchar();
        if (ch != '\n')
        {
        ungetc(ch,stdin);
        scanf("%s",fname);
        }
        else strcpy(fname, "H:\\graph.txt");

}
else if (choice==2)
{
        printf("Enter Full Pathname (Enter---> %s\\graph.txt):",path);ch=getchar();
        if (ch != '\n')
        {
        ungetc(ch,stdin);
        scanf("%s",fname);
        }
        else strcpy(fname, "H:\\graph.txt");

        printf("\nEnter Graph Size:");
        scanf("%d",size);
        printf("Enter density (0,0 .. 1,0):");
        scanf("%f",&dens);
        while (ch=getchar()!='\n');
        if(( fp = fopen(fname,"w")) ==NULL)
        {
                printf("Can't open %s for output\n",fname);
                exit(1);
        }
        srand( (unsigned)time( NULL ) );
        _itoa(*size,tp,10);
        fprintf(fp,tp);fprintf(fp,"\n");
        for (i=1;i<=*size;i++)
        {
                for (j=1;j<i;j++)
                {
                        tmp=rand();
                        s= (float) tmp / (float) RAND_MAX;
                        if (s<=dens)
                                fprintf(fp,"1");
                        else fprintf(fp,"0");
                }
                fprintf(fp,"0\n");
        }
        fclose(fp);
}
else
{
        printf("Bye!\n");exit(1);
```

```
} //if choice

        if(( fp = _fsopen(fname,"rt",_SH_DENYNO)) ==NULL)
{
        printf("Can't open %s for input\n",fname);
        exit(1);
}
fscanf(fp,"%s",tp);ch=getc(fp);
*size=atoi(tp);
for (i=1;i<=*size;i++)
{
for (j=1;j<=i;j++)
{
        ch= getc(fp);
        if (ch=='1')
        {
                graph[i][j]=1;
        }
        else if (ch=='0')
        {
                graph[i][j]=0;
        }
        else
        {
        printf("ERROR in Matrix! %d %d %d",ch,i,j);
        exit(2);
        }
}
ch=getc(fp);
}
fclose(fp);
for (i=1;i<=*size;i++)
for(j=1;j<=i;j++)
        graph[j][i]=graph[i][j];
for (i=1;i<=*size;i++)
for(j=1;j<=*size;j++)
        if (i==j) graph[j][i]=0;
//---------------------------------------------------------------------
printf("Enter number of solvers:");
scanf("%d",&solvers);
if(( fp = _fsopen("H:\\comm.dat","w+b",_SH_DENYNO)) ==NULL)
{
        printf("Can't open comm.dat for output!!\n");
        exit(1);
}
for(i=0;i<solvers;i++)
{
        rec[i].bound=0;
        rec[i].identity=solvers+1;
        rec[i].node=0;
}
j=fwrite(rec,sizeof(rec),solvers,fp);
fclose(fp);
_itoa(solvers,tp,10);
if(( fp = _fsopen("H:\\fname.dat","wt",_SH_DENYRW)) ==NULL)
{
        printf("Can't open fname.dat for output!!\n");
        exit(1);
}
fprintf(fp,"%s",fname);fprintf(fp,"\n");
fprintf(fp,"%s",tp);
fclose(fp);


return solvers;
```

```
}//of menu

int breaknextlevel(int level,int graph[][6001],int sb[][6001],int size,int LowBound,int best[])
{

int nlevel,i,j,kaliteros,k,lnode,sbsize,maxcliq,a[33][33],better[453];

maxcliq=LowBound;
sortsbgraph(graph,sb,level-1,size);
for (lnode=size-1;lnode>=1;lnode--)
{
        sbsize=0;
        for(k=lnode+1;k<=size;k++)
                if (graph[sb[level-1][lnode]][sb[level-1][k]]==1)
                {
                        sbsize++;
                        sb[level][sbsize]=sb[level-1][k];
                }
                if (sbsize > maxcliq - level)
                {
                        if (sbsize <= word_length)
                        {
                                for(i=1;i<=sbsize;i++)
                                for(j=1;j<=sbsize;j++)
                                        a[i][j]=graph[sb[level][i]][sb[level][j]];
                                kaliteros=prakseis(a,sbsize,maxcliq-level+1,better);
                                if (kaliteros >= maxcliq -level +1)
                                {
                                        maxcliq= kaliteros + level;
                                        for (j=1;j<=maxcliq - level; j++)
                                                best[j]= sb[level][better[j]];
                                        best[maxcliq-level+1]=sb[level-1][lnode];

                                }
                        }
                        else
                        {
                                        nlevel=level+1;
                                        kaliteros=breaknextlevel(nlevel,graph,sb,sbsize,maxcliq,better);
                                        if (kaliteros > maxcliq )
                                        {
                                        maxcliq=kaliteros;
                                        for (j=1;j<=maxcliq-level;j++)
                                                best[j]=better[j];
                                        best[maxcliq-level+1]=sb[level-1][lnode];

                                        }//if kaliteros > maxcliq
                        } // sbsize <= word_length
                }// sbsize >maxcliq -level
} //for
return maxcliq;
}
```

```
void main()
{
int size;
int LowBound;

double  duration;clock_t start,finish;
char ch;
int
solvers,level,nlevel,maxcliq,dedges,a[33][33],graph[6001][6001],order[6001],degree[6001],better[453],best[453];
float edges,dens;
int lnode,xnode,i,j,k,sbsize,sb[7][6001],kaliteros;
int lbl,count,id[31],free;char clq[30],tp[3];FILE *fp,*fp1;
int fifo[6001],last,spointer;
MITROO rec[31];


solvers=menu(graph,&size);

for (i=0;i<solvers;i++) id[i]=0;
printf("enter Critical_Low:");scanf("%d",&send);
if(( fp = _fsopen("H:\\comm.dat","r+b",_SH_DENYNO)) ==NULL)
{
        printf("Can't open comm.dat for I/O!!\n");
        exit(1);
}
printf("Starting....\n\n");
start = clock();
dedges=sortgraph(graph,size,order,degree);
LowBound=heurlow(graph,size,best);lbl=LowBound;
maxcliq=LowBound;printf("Low Bound: [%d] \n\n",LowBound);
level=1;last=0;spointer=0;
for (lnode=size-1;lnode>=1;lnode--)
{
        sbsize=0;
        for (k=lnode+1;k<=size;k++)
                if (graph[lnode][k]==1)
                        {
                                sbsize++;
                                sb[1][sbsize]=k;
                        }
                printf("\nlnode=%d sbsize=%d",lnode,sbsize);
                if (sbsize >maxcliq - level )
                {
                        if (sbsize <=word_length)
                        {       printf("Local Processing");
                                for(i=1;i<=sbsize;i++)
                                for(j=1;j<=sbsize;j++)
                                        a[i][j]=graph[sb[1][i]][sb[1][j]];
                        kaliteros=prakseis(a,sbsize,maxcliq,better);
                        if (kaliteros > maxcliq -level )
                        {
                                maxcliq=kaliteros +1;
                                for (j=1;j<=maxcliq-1;j++)
                                        best[j]=sb[1][better[j]];
                                best[maxcliq]=lnode;
                                printf("[%d %d] ",maxcliq,lnode);for (k=1;k<=maxcliq;k++) printf("%d "
,best[k]);
                        }
                        }
                        else // edo sbsize> word_length
                        {
                                if (sbsize <=send)
                                {       printf("put to FIFO");
```

```
                                        last++;
                                        fifo[last]=lnode;
                                }// if sbsize <=send
                                else //sbsize> send
                                {printf(" for transmition to solver \n");
                                        free=-1;
                                        for (i=0;i<solvers;i++)
                                                if (id[i] == 0)
                                                {
                                                  free=i;break;
                                                }//if id[i]==0
                                                while (free==-1)
                                        {
                                                 spointer++;
                                                 if (spointer <=last)
                                                 {
                                                sbsize=0;xnode=fifo[spointer];printf("working localy,
node=%d",xnode);
                                                                for (k=xnode+1;k<=size;k++)
                                                                if (graph[xnode][k]==1)
                                                                {
                                                                        sbsize++;
                                                                        sb[1][sbsize]=k;
                                                                }
                                                nlevel=level+1;

        kaliteros=breaknextlevel(nlevel,graph,sb,sbsize,maxcliq,better);
                                                if (kaliteros > maxcliq )
                                                {
                                                        maxcliq=kaliteros;
                                                        for (j=1;j<=maxcliq-1;j++)
                                                        best[j]=better[j];
                                                        best[maxcliq]=xnode;
                                                        printf("Local from FIFO:[%d %d]
",maxcliq,xnode);
                                                        for (k=1;k<=maxcliq;k++) printf("%d "
,best[k]);
                                                }
                                                printf("\nend local process\n");
                                                 }//spointer <=last
                                                for (i=0;i<solvers;i++)
                                                {
                                                        fseek(fp,i*sizeof(rec[i]),SEEK_SET);
                                                        j=fread(&rec[i],sizeof(rec[i]),1,fp);
                                                        if (rec[i].identity == i)
                                                        {
                                                        printf(" solver no [%d] answered ",i);
                                                        id[i]=0;free=i;
                                                        if (rec[i].bound> maxcliq)
                                                                {
                                                                maxcliq=rec[i].bound;
                                                                _itoa(i,tp,10);printf("Solver:[%d]
",maxcliq);

        strcpy(clq,"H:\\");strcat(clq,tp);strcat(clq,".clq");
                                                        while (( fp1 =
_fsopen(clq,"rt",_SH_DENYRW)) ==NULL);
                                                                for (j=1;j<=maxcliq;j++)
                                                                fscanf(fp1,"%d ",&best[j]);
                                                                fclose(fp1);
                                                                }
                                                        }
                                                }
                                }
```

```
        rec[free].identity=solvers;rec[free].bound=maxcliq;rec[free].node=lnode;
                                        fseek(fp,free*sizeof(rec[free]),SEEK_SET);
                                        j=fwrite(&rec[free],sizeof(rec[free]),1,fp);id[free]=1;
                                        printf("sending to Solver lnode= %d",lnode);

                                }

                        }
                }
}
count=0;
while (count <solvers)
{
        for (i=0;i<solvers;i++)
        {
                if (id[i] ==1)
                {
                fseek(fp,i*sizeof(rec[i]),SEEK_SET);
                j=fread(&rec[i],sizeof(rec[i]),1,fp);
                if (rec[i].identity==i)
                {
                        id[i]=2;
                        count++;
                                if (rec[i].bound> maxcliq)
                                {
                                        maxcliq=rec[i].bound;
                                        _itoa(i,tp,10);
                                        strcpy(clq,"H:\\");strcat(clq,tp);strcat(clq,".clq");
                                        while (( fp1 = _fsopen(clq,"rt",_SH_DENYRW)) ==NULL);
                                        for (j=1;j<=maxcliq;j++)
                                        fscanf(fp1,"%d ",&best[j]);
                                        fclose(fp1);
                                }
                        rec[i].bound=-1;rec[i].identity=solvers;rec[i].node=-1;
                        fseek(fp,i*sizeof(rec[i]),SEEK_SET);
                        j=fwrite(&rec[i],sizeof(rec[i]),1,fp);

                }
                } // id[i] ==1
                else if (id[i]==0)
                {
                        rec[i].bound=-1;rec[i].identity=solvers;rec[i].node=-1;count++;id[i]=2;
                        fseek(fp,i*sizeof(rec[i]),SEEK_SET);
                        j=fwrite(&rec[i],sizeof(rec[i]),1,fp);


                }//id[i]==0
        }//for i=0 to solvers
} //while count <solvers


do{
spointer++;
 if (spointer <=last)
        {
        sbsize=0;xnode=fifo[spointer];printf("clearing FIFO, working localy , lnode=%d",xnode);
        for (k=xnode+1;k<=size;k++)
        if (graph[xnode][k]==1)
                {
                        sbsize++;
                        sb[1][sbsize]=k;
                }
                nlevel=level+1;
                kaliteros=breaknextlevel(nlevel,graph,sb,sbsize,maxcliq,better);
                if (kaliteros > maxcliq )
```

```
                {
                        maxcliq=kaliteros;
                        for (j=1;j<=maxcliq-1;j++)
                        best[j]=better[j];
                        best[maxcliq]=xnode;
                        printf("Local from FIFO:[%d %d] ",maxcliq,xnode);
                        for (k=1;k<=maxcliq;k++) printf("%d " ,best[k]);
                }
                printf("\nend local process\n");
        }//spointer <=last
}while (spointer <last);
printf("\n\nDone in ");
 finish = clock();fclose(fp);
 duration = (double)(finish - start) / CLOCKS_PER_SEC;
 printf( "%8.5f seconds\n\n", duration );
dens= (float)dedges /((float) size*((float)size-(float)1.0));
edges=(float) dedges/(float)2.0;
printf("Nodes: %d\nEdges: %5.0f\nDensity: %9.4f\nLOWBOUND: %d\n\n",size,edges,dens,lbl);
printf("Maximum clique number= [%d]\n\n",maxcliq);
for (i=1;i<=maxcliq;i++)
printf("%d ",order[best[i]]);
for(i=1;i<=maxcliq-1;i++)
        for(j=i+1;j<=maxcliq;j++)
                if (graph[best[i]][best[j]]==0) printf("ERROR!!, nodes [%d %d] ",best[i],best[j]);
 remove("H:\\fname.dat");remove("H:\\comm.dat");
ch=getchar();printf("\nPress enter");ch=getchar();

} // of main
```

1.2 Επιλυτής Προβλήματος (PS)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <share.h>
#include <process.h>

static int krit=32;

int sortgraph(int a[][6001],int size,int order[6001],int degree[6001])
{
int i,j,minindex,mindegree,temp,dedges;

dedges=0;
for(i=1;i<=size;i++)
{
        degree[i]=0;order[i]=i;
        for (j=1;j<=size;j++)
                if (a[i][j] == 1) degree[i]++;
        dedges+=degree[i];
}
for(i=1;i<=size-1;i++)
{
        minindex=i;mindegree=degree[i];
        for (j=i+1;j<=size;j++)
        {
                if (mindegree > degree[j])
                {
                        mindegree=degree[j];minindex=j;
                }
        }
        temp=order[minindex];order[minindex]=order[i];order[i]=temp;
        temp=degree[minindex];degree[minindex]=degree[i];degree[i]=temp;
}
for (i=1;i<=size;i++)
        for (j=1;j<=i-1;j++)
                if (order[i] < order[j])
                        a[i][j]=a[order[i]][order[j]];
                else a[i][j]=a[order[j]][order[i]];
for (i=1;i<=size-1;i++) for (j=i+1;j<=size;j++) a[i][j]=a[j][i];
return dedges;

} //sortgraph

void sortsbgraph(int a[][6001],int sb[][6001],int level,int size)
{
int i,j,minindex,mindegree,temp, order[6001],degree[6001];

for(i=1;i<=size;i++)
{
        degree[i]=0;order[i]=i;
        for (j=1;j<=size;j++)
                if (a[sb[level][i]][sb[level][j]] == 1) degree[i]++;
}
for(i=1;i<=size-1;i++)
{
        minindex=i;mindegree=degree[i];
        for (j=i+1;j<=size;j++)
        {
                if ( degree[j] < mindegree )
```

```
                {
                        mindegree=degree[j];minindex=j;
                }
        }
        temp=sb[level][minindex];
        sb[level][minindex]=sb[level][i];
        sb[level][i]=temp;
        degree[minindex]=degree[i];
}

} //sortsbgraph

void Evangelos(unsigned int boo[2201],int *lastrow,int size,int LowBound)
{
int last,i,j,assoi;unsigned int a,help,norma,helper[2201];
int aporipteos[2201];

last=0;
for (i=1;i<=*lastrow;i++)
{
        aporipteos[i]=0;
        assoi=0;
        help=boo[i];
        for (j=1;j<=size;j++)
        {
                a= help & 01;
                if (a == 1) assoi++;
                help = help >> 1;
        }
        if (size - assoi <LowBound) aporipteos[i] = 1;
}
for (i=1;i<=*lastrow-1;i++)
        for (j=i+1;j<=*lastrow;j++)
        {
                norma=boo[i] & boo[j];
                if (norma==boo[i]) aporipteos[j]=1; else
                        if (norma == boo[j]) aporipteos[i]= 1;
        }
for (i=1;i<=*lastrow;i++)
        if (aporipteos[i]==0)
{
        last++;helper[last]=boo[i];
}
*lastrow=last;for(i=1;i<=last;i++) boo[i]=helper[i];

} //Evangelos

int heurlow(int graph[][6001],int size,int better[])
{
int i,j,bestcliq,kalo;

bestcliq=1;
better[bestcliq]=size;
for (i=size-1;i>=1;i--)
{
        kalo=1;
        for (j=1;j<=bestcliq;j++)
                if (graph[better[j]][i] ==0) { kalo=0;break; }
        if (kalo==1)
        {
                bestcliq++;better[bestcliq]=i;
        }
}
return bestcliq;
} //heurlow
```

```
int prakseis(int a[][33],int size,int LowBound,int better[453])
{
        int numterm,found,index,lastrow,mises,counter,tsou,seira[33],megal,idxmegal;
        unsigned int boo[2201],paragontes[33][2],pilot,temp;
        register i,j;

        numterm=0;
for (i=1;i<=size-1;i++)
{
        seira[i]=0;
        found=0;
        index=i+1;
        while ((found ==0) && (index <= size))
                if (a[i][index]==0) found=1; else index++;
                if (found==1)
                {
                        numterm++;
                        paragontes[numterm][1]=1;
                        paragontes[numterm][1]= paragontes[numterm][1] << (i -1);
                        paragontes[numterm][2]=0;
                        for (j=i+1;j<=size;j++)
                                if (a[i][j] ==0)
                                {
                                pilot=1;seira[i]++;
                                pilot= pilot << (j-1);
                                paragontes[numterm][2]=paragontes[numterm][2] | pilot;
                                }
                }
}
for (i=1;i<=numterm-1;i++)
        for(j=i+1;j<=numterm;j++)
                if (seira[i] < seira[j])
                {
                        temp=paragontes[i][1];paragontes[i][1]=paragontes[j][1];paragontes[j][1]=temp;
                        temp=paragontes[i][2];paragontes[i][2]=paragontes[j][2];paragontes[j][2]=temp;
                        tsou=seira[i];seira[i]=seira[j];seira[j]=tsou;
                }

if (numterm >0)
{
lastrow=2;boo[1]=0;boo[2]=0;
boo[1] = boo[1] | paragontes[1][1];
boo[2] = boo[2] | paragontes[1][2];

Evangelos(boo,&lastrow,size,LowBound);

for (counter=2;counter<=numterm;counter++)
{

        mises=lastrow;
        lastrow=lastrow*2;
        for (i=mises+1;i<=lastrow;i++)
                boo[i]=boo[i-mises];
        for (i=1;i<=mises;i++) boo[i]= boo[i] | paragontes[counter][1];
        for (i=mises+1;i<=lastrow;i++)          boo[i] = boo[i] | paragontes[counter][2];

        Evangelos(boo,&lastrow,size,LowBound);
}
}

else
{
        lastrow=1;boo[1]=0;
```

```
}

megal=-1;
for(i=1;i<=lastrow;i++)
{
        temp=boo[i];
        counter=0;
        for(j=1;j<=size;j++)
        {
                index=temp & 01;
                if (index==0)
                        counter++;
                temp = temp >> 1;
        }
        if (counter > megal)
        {
                megal=counter;idxmegal=i;
        }
}
if (megal >-1)
{
        temp=boo[idxmegal];counter=0;
        for(j=1;j<=size;j++)
        {
                index=temp & 01;
                if (index==0)
                {
                        counter++;
                        better[counter]=j;

                }
        temp = temp >> 1;
        }
}
return megal;

} //of prakseis


int breaknextlevel(int level,int graph[][6001],int sb[][6001],int size,int LowBound,int best[])
{

int nlevel,i,j,kaliteros,k,lnode,sbsize,maxcliq,a[33][33],better[453];

maxcliq=LowBound;// if (level > l) { l=level; printf("l=%d ",l);}
sortsbgraph(graph,sb,level-1,size);
for (lnode=size-1;lnode>=1;lnode--)
{
        sbsize=0;
        for(k=lnode+1;k<=size;k++)
                if (graph[sb[level-1][lnode]][sb[level-1][k]]==1)
                {
                        sbsize++;
                        sb[level][sbsize]=sb[level-1][k];
                }
                if (sbsize > maxcliq - level)
                {
                        if (sbsize <= krit)
                        {
                                for(i=1;i<=sbsize;i++)
                                for(j=1;j<=sbsize;j++)
                                        a[i][j]=graph[sb[level][i]][sb[level][j]];
                                kaliteros=prakseis(a,sbsize,maxcliq-level+1,better);
                                if (kaliteros >= maxcliq -level +1)
                                {
```

```
                                        maxcliq= kaliteros + level;
                                        for (j=1;j<=maxcliq - level; j++)
                                                best[j]= sb[level][better[j]];
                                        best[maxcliq-level+1]=sb[level-1][lnode];

                                }
                        }
                        else
                        {
                                nlevel=level+1;
                                        kaliteros=breaknextlevel(nlevel,graph,sb,sbsize,maxcliq,better);
                                        if (kaliteros > maxcliq )
                                        {
                                        maxcliq=kaliteros;
                                        for (j=1;j<=maxcliq-level;j++)
                                                best[j]=better[j];
                                        best[maxcliq-level+1]=sb[level-1][lnode];

                                        }
                        }
                }
}
return maxcliq;
}


void main(int argc,char *argv[])
{
int size;
struct MITROO
{
int identity;
int node;
int bound;
} ;
//double  duration;clock_t start,finish;
char ch;
int level,nlevel,maxcliq,dedges,graph[6001][6001],order[6001],degree[6001],better[453],best[453];
//float edges,dens;
int lnode,i,j,k,sbsize,sb[7][6001],kaliteros;
FILE *fp,*fp1;char clq[30],fname[35],tp[3];int solvers,myid;
MITROO rec[31];


if (argc <= 1 ) { printf("\nsyntax: Program_name [Id]\n\npress any key...");ch=getchar();exit(9);}
myid=atoi(argv[1]);             printf("My ID is [%d] \n",myid);
printf("Waiting to receive Graph Filename..... ");
while (( fp = _fsopen("H:\\fname.dat","rt",_SH_DENYNO)) ==NULL);
fscanf(fp,"%s",fname);
fscanf(fp,"%s",tp);fclose(fp);
printf("   Received!\nReading Graph from file [%s]..... ",fname);
solvers=atoi(tp);

if(( fp = _fsopen(fname,"rt",_SH_DENYNO)) ==NULL)
{
        printf("Can't open %s for input\n",fname);
        exit(1);
}

fscanf(fp,"%s",tp);ch=getc(fp);
size=atoi(tp);
for (i=1;i<=size;i++)
{
for (j=1;j<=i;j++)
{
```

```
            ch= getc(fp);
            if (ch=='1')
            {
                        graph[i][j]=1;
            }
            else if (ch=='0')
            {
                        graph[i][j]=0;
            }
            else
            {
            printf("ERROR in Matrix! %d %d %d",ch,i,j);
            exit(2);
            }
}
ch=getc(fp);
}
fclose(fp);

for (i=1;i<=size;i++)
for(j=1;j<=i;j++)
            graph[j][i]=graph[i][j];
for (i=1;i<=size;i++)
for(j=1;j<=size;j++)
            if (i==j) graph[j][i]=0;
printf("Done!");


strcpy(clq,"H:\\");
_itoa(myid,tp,10);
strcat(clq,tp);
strcat(clq,".clq");

            if(( fp = _fsopen("H:\\comm.dat","r+b",_SH_DENYNO)) ==NULL)
{
            printf("Can't open comm.dat for I/O!!\n");
            exit(1);
}
lnode=-2;rec[myid].node=-2;
dedges=sortgraph(graph,size,order,degree);maxcliq=0;
do
{
            for(i=1;i<=10000;i++) for(j=1;j<=1000;j++);
            printf("Waiting to receive Job.....\n ");
            rec[myid].identity=myid;
            while ((rec[myid].identity != solvers) || (rec[myid].node == lnode))
            {
            fseek(fp,myid*sizeof(rec[myid]),SEEK_SET);j=fread(&rec[myid],sizeof(rec[myid]),1,fp);
            }

if (rec[myid].bound  == -1) {printf("\nProject finished.\n");exit(3);}
printf("Job Received. Starting....");
level=1;lnode=rec[myid].node;
sbsize=0;if (rec[myid].bound > maxcliq) maxcliq=rec[myid].bound;
for (k=lnode+1;k<=size;k++)
            if (graph[lnode][k]==1)
                        {
                                    sbsize++;
                                    sb[1][sbsize]=k;
                        }
            printf("LB=%d sbsize=%d lnode=%d\n",maxcliq,sbsize,lnode);
                        nlevel=level+1;
                        kaliteros=breaknextlevel(nlevel,graph,sb,sbsize,maxcliq,better);

                        if (kaliteros > maxcliq )
```

```
                {
                        maxcliq=kaliteros;
                        for (j=1;j<=maxcliq-1;j++)
                        best[j]=better[j];
                        best[maxcliq]=lnode;
                        printf("[%d] ",maxcliq);
                        while (( fp1 = _fsopen(clq,"wt",_SH_DENYRW)) ==NULL);
                        for (j=1;j<=maxcliq;j++)
                                fprintf(fp1,"%d ",best[j]);
                        fclose(fp1);
                        rec[myid].identity=myid;rec[myid].bound=maxcliq;
                        fseek(fp,myid*sizeof(rec[myid]),SEEK_SET);
                        j=fwrite(&rec[myid],sizeof(rec[myid]),1,fp);

                }else
                {rec[myid].identity=myid;rec[myid].bound=maxcliq;
                        fseek(fp,myid*sizeof(rec[myid]),SEEK_SET);
                        j=fwrite(&rec[myid],sizeof(rec[myid]),1,fp);}

                printf("\nFinished Job\n");
}while (1==1);


ch=getchar();


} // of main
```

2 Κατάλογος Πινάκων

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

3 Κατάλογος Εικόνων

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ