2Η ΕΡΓΑΣΙΑ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ
ΒΛΑΧΟΓΙΑΝΝΗΣ ΜΑΡΙΟΣ

Επισημάνσεις

Α) Στο δοθέν πρόγραμμα bcsp.c,ο χρόνος στον οποίο ολοκληρωνόταν η εκτέλεση του προγράμματος ανεξαρτήτως επιλογής αλγορίθμου από τον χρήστη (hill climbing ή depth first) ήταν στην πλειοψηφία του μηδενικός . Συνεπώς, δύσκολα θα μπορούσε να αναχθεί συμπέρασμα σχετικά με την χρονική συμπεριφορά των αλγορίθμων πάνω στα προβλήματα που θα δοκιμαστούν στην εν λόγω εργασία . Μετά από σχετική έρευνα παρατηρήθηκε ότι ο χρόνος εκτέλεσης προερχόταν από την συνάρτηση clock() της βιβλιοθήκης time.h που ουσιαστικά αναπαριστά τον χρόνο εκτέλεσης συνολικά του προγράμματος βασισμένο στους κτύπους της CPU , κάτι που συγχρόνως εξαρτάται και εξετάζει περισσότερο δυναμική συστήματος παρά χρόνο . Συνεπώς έγινε μια μικρή μετατροπή στο σημείο του κώδικα που γίνεται από τον χρήστη η επιλογή αλγορίθμου , κομμάτι κώδικα το οποίο βρίσκεται εντός της main .Πιο συγκεκριμένα , χρησιμοποιήθηκε ξανά η συνάρτηση clock της βιβλιοθήκης time ώστε να μετρηθεί ο χρόνος αμέσως πριν και αμέσως μετά την επιλογή και εκτέλεση των διαθέσιμων αλγορίθμων (start και end time) και η διαφορά και συνεπώς ο ακριβής χρόνος εκτέλεσης του επιλεγμένου αλγορίθμου πάνω στο εκάστοτε πρόβλημα ,να προκύψει μέσω της προσθήκης μιας συνάρτησης elapsed time που όπως προαναφέραμε να μετράει την διαφορά και συνεπώς τον ακριβή χρόνο.

Ακολουθεί στιγμιότυπο κώδικα:

```
clock_t start_time, end_time;


  if (strcmp(argv[1], "hill") == 0) {
     start_time = clock();
     hill_climbing(argv);
     end_time = clock();
  } else if (strcmp(argv[1], "depth") == 0) {
     start_time = clock();
     depth_first(argv);
     end_time = clock();
  }


double elapsed_time = get_elapsed_time(start_time, end_time);
  display_elapsed_time(elapsed_time);
```

Β) Με σταθερό αριθμό διαθέσιμων λογικών προτάσεων Ν =10 δοκιμάζω για αριθμό λογικών προτάσεων ανά διάζευξη Κ=4 και Κ=5 και για αριθμό διαζεύξεων Μ={10,30,60,90,120,160,200,250} για κάθε Κ.

Για κάθε Μ του κάθε Κ και για κάθε αλγόριθμο που μας δίνεται δοκιμάστηκαν 15 προβλήματα τα οποίο παρήχθησαν μέσω του δοθέντος προγράμματος bcsp_generate.c , παραμετροποιώντας τα αντίστοιχα Μ και Κ μιας και το Ν=10 σταθερά.

Η ονοματολογία των αρχείων που περιέχουν τα προβλήματα ακολουθεί το παρακάτω μοτίβο . test αριθμος1Χαριθμός2_αριθμός3.txt .Το σύμβολο Χ αναπαριστά τον αριθμό 10 στα λατινικά οπότε πολλαπλασιασμένο με τον αριθμό1 σηματοδοτεί τον εκάστοτε αριθμό διαζεύξεων Μ πχ(1Χ,3Χ,6Χ,9Χ,12Χ,16Χ,20Χ,25Χ αντιστοιχεί στο Μ ={10,30,60,90,120,160,200,250}). Ο αριθμός2 αναπαριστά το σύνολο των λογικών προτάσεων ανά διάζευξη Μ και στην ουσία για αριθμος2 =4 είναι το Κ=4 και αντίστοιχα και για το 5 . Τέλος , ο αριθμός3 αναπαριστά το εκάστοτε test από τα συνολικά 15 που θα εξεταστούν ανα περίπτωση και είναι αύξων 1,2,3 ...15.

Ενδεικτικά παραδείγματα test1Χ4_1.txt και test25Χ5_15.txt όπου είναι το 1ο πρόβλημα με 10 συνολικά διαζεύξεις και 4 λογικές προτάσεις ανά διάζευξη και 15ο πρόβλημα με 250 συνολικά διαζεύξεις και 5 λογικές προτάσεις ανά διάζευξη αντιστοίχως.

Τα output αρχεία στα οποία αποθηκεύεται η λύση του κάθε προβλήματος ,εάν υπάρχει , ακολουθούν την ίδια ονοματολογία με τα αρχεία προβλημάτων με την διαφορά ότι πριν τον αύξων αριθμό προβλήματος μπαίνουν κάποια γράμματα που συμβολίζουν τον αλγόριθμο που επεξεργάστηκε το κάθε πρόβλημα . Η για τον hill climbing, DD για τον depth first ,W για τον walksat και DP για τον dpll. Οι δύο τελευταίοι αλγόριθμοι αφορούν το 2ο ερώτημα της εργασίας .

Πχ out1Χ4Η_1.txt ή out25Χ5DD_15.txt

Τα output αρχεία δημιουργήθηκαν μέσω του προγράμματος generateBlank.c όπου στην ουσία είναι το bcsp_generate.c με την διαφορά ότι η συνάρτηση write to file δεν γράφει κάτι και στην ουσία δημιουργείται ένα κενό αρχείο.

Δημιουργήθηκε και ένα πρόγραμμα generate_Inputs_Tries.c όπου δημιουργώ όλους τους πιθανούς συνδυασμούς test και out αρχείων προσαρμοσμένα και στο path που θα χρησιμοποιηθεί στην command line για να γίνουν οι δοκιμές. Όλοι οι παραχθέντες συνδυασμοί βρίσκονται στο text αρχείο ALL TRIES WITH INPUTS_OUTPUTS οι οποίοι είναι 960 συνολικά.

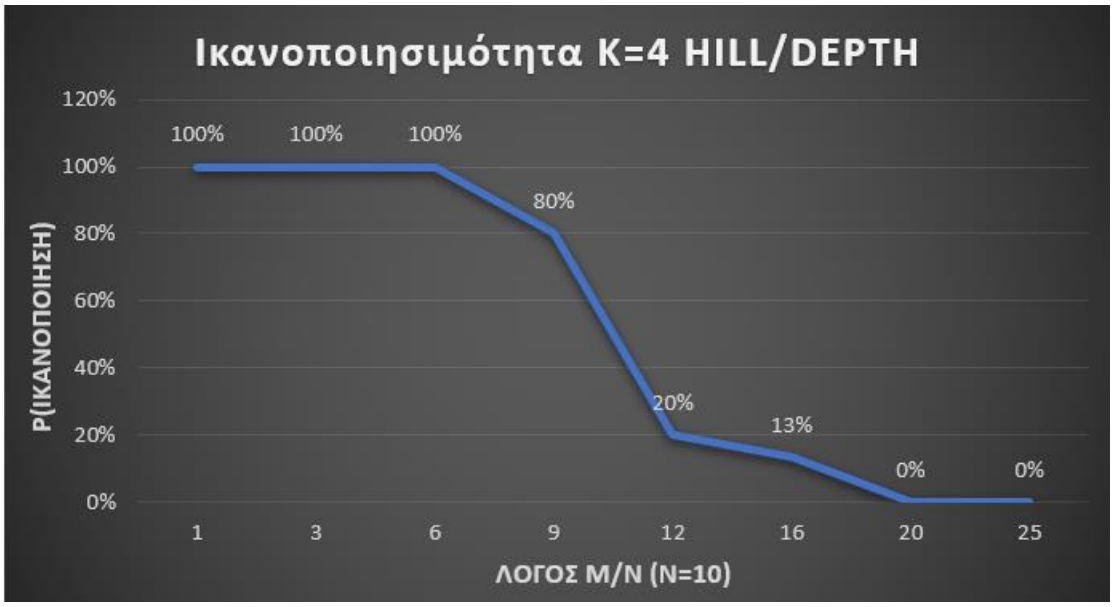
Επιπλέον υπάρχει και ένα text αρχείο ALL VALIDATES οπού εκεί βρίσκονται όλα τα path όλων των συνδυασμών test και out όπου ουσιαστικά ελέγχεται αν η λύση που αποθηκεύεται στο αρχείο out επαληθεύει το πρόβλημα του αντίστοιχου test αρχείου.

Όλες οι μετρήσεις χρόνου και ικανοποιησιμότητας των διαζεύξεων Μ μέσω depth και hill για Κ=4 και Κ=5 , αναπαριστώνται στο Book του Excel με όνομα Analysis και συγκεκριμένα στο sheet Default καθώς στο sheet Plus αναπαριστώνται οι μετρήσεις χρόνου και ικανοποιησιμότητας των διαζεύξεων Μ μέσω walksat και dpll για Κ=4 και Κ=5 .

Α ΕΡΩΤΗΜΑ

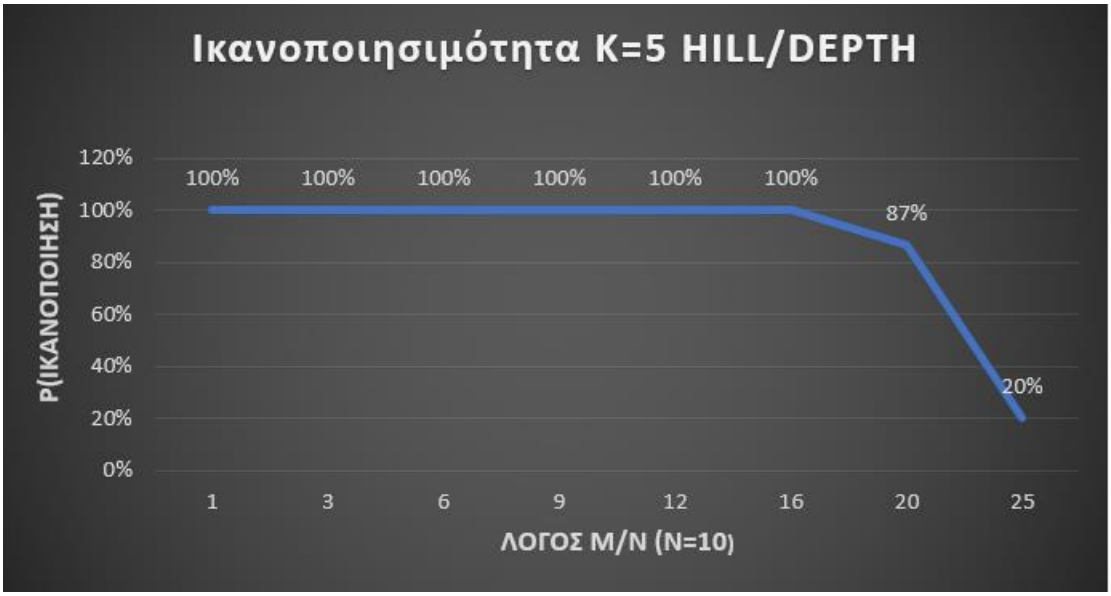
Στο ζήτημα της **ικανοποιησιμότητας** παρατηρώ κοινή συμπεριφορά στο Hill και στο Depth συνεπώς παραθέτω μόνο ένα διάγραμμα για το Κ=4 και ένα για το Κ=5 .

Για Κ=4



Παρατηρώ ότι η πρώτη ποσοστιαία πτώση στην ικανοποιησιμότητα για 4 λογικές προτάσεις ανά διάζευξη με 10 λογικές προτάσεις διαθέσιμες , επέρχεται στις 90 συνολικά διαζεύξεις με ποσοστό 80% ,στις 120 και 160 διαζεύξεις παρατηρούμε ικανοποιησιμότητα περί το 20% και τελός μηδενική ικανοποιησιμότητα στις 200 και 250 διαζεύξεις.

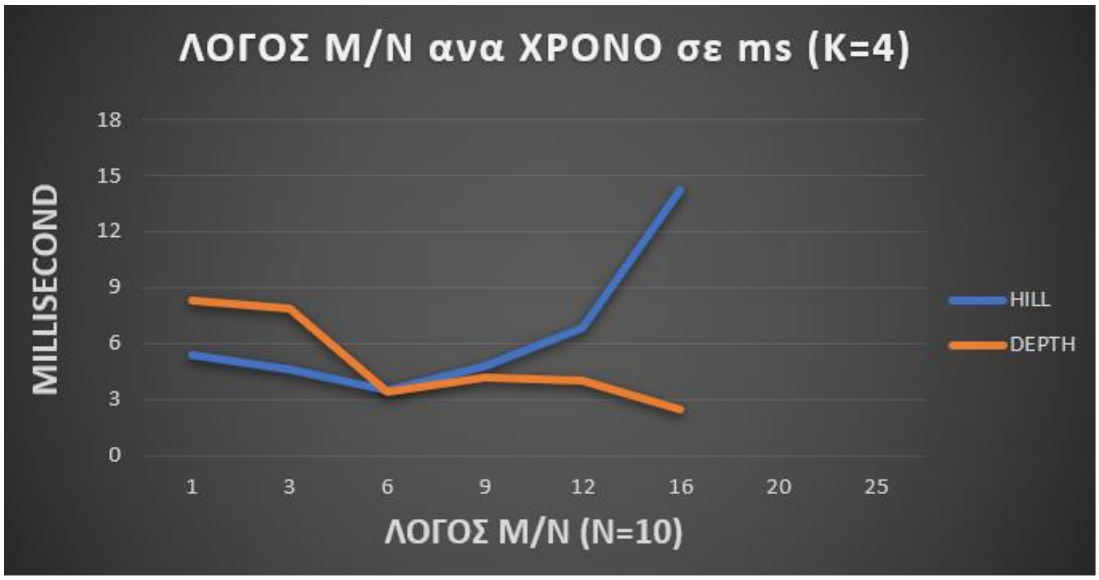
Για Κ=5



Παρατηρώ ότι η πρώτη ποσοστιαία πτώση στην ικανοποιησιμότητα για 5 λογικές προτάσεις ανά διάζευξη με 10 λογικές προτάσεις διαθέσιμες , επέρχεται στις 200 συνολικά διαζεύξεις με ποσοστό 87% ενώ στις 250 διαζεύξεις μετρά ακόμα ποσοστό ικανοποιησιμότητας περί το 20%.

Στο ζήτημα του **χρόνου** αντιπαραβάλουμε σε κοινό διάγραμμα την χρονική συμπεριφορά του hill και depth για K=4 και K=5 ξεχωριστά .

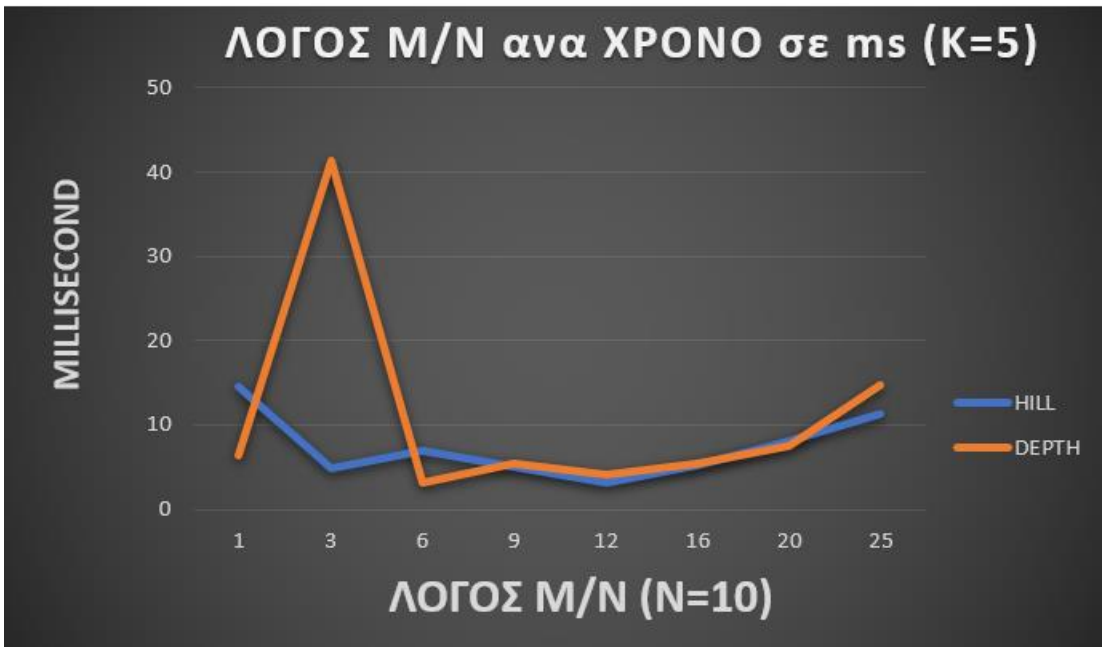
Για K=4



Στο παραπάνω διάγραμμα παρατηρούμε ότι τόσο η depth όσο και η hill έχουν πτωτική χρονική πορεία μέχρι τις πρώτες 60 διαζεύξεις , με την depth να βρίσκεται

υψηλότερα χρονικά από την hill και να ακολουθεί μια πιο απότομη πτώση συγκριτικά με την hill .Έπειτα των 60 διαζεύξεων όπου χρονικά συναντιούνται σε χαμηλούς χρόνους η hill αυξάνεται όλο και περισσότερο χρονικά για όλες τις εναπομείνασες διαζεύξεις όπου ικανοποιείται ,με την depth έπειτα από εκείνο το σημείο να ακολουθεί σταθερή -ελαφρώς αυξημένη χρονική πορεία ,σε κλίμακα χρόνου κάτω από την hill όμως και με ελαφρώς πτωτική πορεία στις τελευταίες ικανοποιήσιμες διαζεύξεις .

Για K=5



Μέχρι τις 30 πρώτες διαζεύξεις ακολουθούν αντίθετες πορείες με τον depth να αυξάνεται ραγδαία και τον hill να μειώνεται ομαλά. Στις επόμενες 30 διαζεύξεις παρατηρούμε ραγδαία μείωση του depth και ομαλή αύξηση του hill. Στις επόμενες 30 διαζεύξεις υπάρχει αύξηση του depth και μείωση του hill σε ομαλά επίπεδα, ενώ για τις εναπομείνασες διαζεύξεις ακολουθούν περίπου σταθερή πορεία με τον depth να υπερβαίνει ελαφρώς χρονικά.

Β ΕΡΩΤΗΜΑ

Εξηγώ συνοπτικά την **hill climbing**

→Αρχικοποιώ τυχαία ένα διάνυσμα

→ Ελέγχω πόσες διαζεύξεις μένουν ανικανοποίητες εάν εφάρμοζα στις λογικές προτάσεις κάθε διάζευξης τις τιμές του παραπάνω διανύσματος και τις καταγράφω.

→ Όσο οι συνολικές ανικανοποίητες διαζεύξεις δεν μηδενίζονται (συνεπώς δεν έχω βρει λύση) και είμαι και μέσα σε συγκεκριμένα χρονικά πλαίσια αλλάζω την τιμή του πρώτου στοιχείου του διανύσματος, έχοντας επί της ουσίας ένα νέο διάνυσμα .

→ Αφού μετρήσω το σύνολο των ανικανοποίητων διαζεύξεων με αυτή την αλλαγή , ελέγχω αν είναι μικρότερο ή μεγαλύτερο του αμέσως προηγούμενου . Εάν είναι μικρότερο σημειώνω την θέση του διανύσματος που άλλαξα ως καλύτερη προς στιγμήν και σημειώνω και το σύνολο των ανικανοποίητων διαζεύξεων. Αν είναι μεγαλύτερο απλά επαναφέρω το τρέχων στοιχείο του διανύσματος στην αρχική τιμή κάτι που κάνω ακόμα και όταν το σύνολο των ανικανοποίητων διαζεύξεων είναι μικρότερο του προηγούμενου αφού το έχω σημειώσει ως καλύτερη λύση.

→Επαναλαμβάνω την διαδικασία αλλαγής τιμής και ελέγχου για κάθε στοιχείο του διανύσματος . Αν στην συνολική επεξεργασία του διανύσματος βρούμε καλύτερη αλλαγή τότε στο σημείο που βρήκαμε την καλύτερη αλλαγή αλλάζουμε και την τιμή και επανερχόμαστε στο 3ο βήμα . Αν όμως η συνολική επεξεργασία δεν καταλήξει σε καλύτερη αλλαγή επανερχόμαστε στο πρώτο βήμα.

Εξηγώ συνοπτικά τον **walksat**

Ουσιαστικά hill climbing εφαρμόζοντας τις παρακάτω αλλαγές.

***Ορίζω μια υψηλή τιμή σε μια μεταβλητή που αναπαριστά το συνολικό χώρο των πιθανών αλλαγών στις οποίες μπορώ να προβώ ώστε να φτάσω στην λύση και μια πολύ μικρή τιμή σε μια άλλη μεταβλητή η οποία είναι υπεύθυνη ώστε να μειώνει σταδιακά τον συνολικό χώρο των αλλαγών. Αυτό μας εξασφαλίζει ότι εξερευνώ τον συνολικό χώρο σχολαστικά αλλά μη εξαντλητικά ώστε να κινδυνεύω να μην βρώ λύση από τον περιορισμό του χρόνου , κάτι που θα λέγαμε ότι είναι μια πρώτη διαφορά ανάμεσα στον hill climbing και τον walksat. Μια δεύτερη αλλά η πιο βασική διαφορά είναι ότι κατά τον έλεγχο του διανύσματος σχετικά με το αν κάθε αλλαγή είναι καλύτερη ή μη από την προηγούμενη, αποδεκτές δεν γίνονται μόνο οι πραγματικά καλύτερες λύσεις αλλά και κάποιες που τυπικά δεν είναι καλύτερες. Η αποδοχή αυτή εξαρτάται από το πόσο μεγάλος είναι ο χώρος αναζήτησης την στιγμή του ελέγχου, όπου σε μεγαλύτερο χώρο είναι πολύ πιο πιθανό να γίνουν αποδεκτές τυπικά μη καλύτερες λύσεις ενώ σε μικρότερο χώρο είναι λιγότερο πιθανό να γίνουν αποδεκτές. Η προσθήκη αυτή στην συνθήκη ελέγχου και συνεπώς η αποδοχή μη βέλτιστων λύσεων μας δίνει την δυνατότητα να δοκιμάσουμε όλο και περισσότερα διανύσματα ,εξασφαλίζοντας έτσι ότι θα φτάσουμε στην πραγματική βέλτιστη λύση και όχι σε κάποιο τοπικό ακρότατο , δηλαδή μια λύση που την ανακηρύσσουμε ως την γενική βέλτιστη λύση αλλά στην πραγματικότητα είναι βέλτιστη για ένα πολύ μικρότερο εύρος .

Τα επιπλέον χαρακτηριστικά επί του κώδικα .

Παρακάτω βλέπουμε την υλοποίηση του αλγόριθμου **hill climbing** που η προσθήκη των **κόκκινων** στοιχείων τον μετατρέπει σε **walksat.**

```
void walkSat(char **argv) {
   FILE *outfile;
   int *vector;
   int i, h, h1, h2;
   clock_t t, t1, t2;
   long restarts=0, steps=0;
   int best_change;
   double temperature = 1000.0; // Ο χώρος αναζήτησης
   double cooling_rate = 0.95; // Η μείωση του χώρου αναζήτησης
   vector=(int*) malloc(N*sizeof(int));

   t1=clock();

   // Initialization steps
   initialize(vector);
   h=count(vector);

   while (h>0 && temperature > 0.01) { // Ο χώρος αναζήτησης δεν έχει εξερευνηθεί ακόμη πλήρως
      // Check for time limit
      t=clock();
      if (t-t1>CLOCKS_PER_SEC*TIMEOUT) {
         t2=clock();
         printf("\n\nNO SOLUTION found with walkSAT...\n");
         printf("Time spent: %f secs\n",((float) t2-t1)/CLOCKS_PER_SEC);
         printf("Number of restarts: %ld\n",restarts);
         printf("Number of steps: %ld\n",steps);
         return ;
```

```
}

steps++;
h2=h;
best_change=-1;
for(i=0;i<N;i++) {     // For each proposition...
   vector[i]=-vector[i];   // ...if we flip its value...
   h1=count(vector);   // ...how many sentences are not satisfied?
   if (h1<h2 || exp((h2 - h1) / temperature) > (double)rand() / RAND_MAX) { //
```

Αν η συνθήκη είναι αληθής γίνεται αποδεκτή ή μη βέλτιστη λύση.

Το πρώτο μέρος της συνθήκης είναι επί της ουσίας το e^((h2-h1)/temperature) και το δεύτερο είναι ένας τυχαία παραγόμενος αριθμός στο [0,1].

Όσο μεγαλύτερη temperature τόσο μεγαλύτερη η πιθανότητα ο αριθμός e^((h2-h1)/temperature) να είναι μεγαλύτερος από τον όποιο ή τους περισσότερους παραγόμενους αριθμούς σε αυτο το εύρος [0,1] . Όσο όμως μικρότερη τόσο μικρότερη και η πιθανότητα ο αριθμός e^((h2-h1)/temperature) να είναι μεγαλύτερος από τον όποιο ή τους περισσότερους παραγόμενους αριθμούς σε αυτό το εύρος [0,1]

```
      h2=h1;        // ...remember it!
      best_change=i;
   }
   vector[i]=-vector[i];   // Cancel the last flip of value.
}

if (best_change>=0)
   vector[best_change]=-vector[best_change];
else          // ...else if we had no improvement
{
   initialize(vector);     // Random restart
   restarts++;
}
h=count(vector);
```

```
      // Μείωση της θερμοκρασίας
      temperature *= cooling_rate; Μείωση του χώρου αναζήτησης, μετά από κάθε προσπάθεια εύρεσης λύσης συνολικά στο διάνυσμα ,κατά cooling rate.
   }

   t2=clock();
   printf("\n\nSolution found with walkSAT!\n"); display(vector); printf("\n");  Αντί hill climbing
   printf("Time spent: %f secs\n",((float) t2-t1)/CLOCKS_PER_SEC);
   printf("Number of restarts: %ld\n",restarts);
   printf("Number of steps: %ld\n",steps);

   outfile=fopen(argv[3],"w");
   if (outfile==NULL) {
      printf("Cannot open output file. Now exiting...\n");
      return;
   }
   for(i=0;i<N;i++)
      fprintf(outfile,"%d ",vector[i]);
   fclose(outfile);
}
```

Εξηγώ συνοπτικά την **depth first**

→ Αρχικοποιώ ένα διάνυσμα

→ Το προσθέτω σε μια στοίβα

→ Όσο η στοίβα δεν είναι άδεια και είμαι εντός ενός ορισμένου χρονικού πλαισίου αφαιρώ το ανώτερο στοιχείο της στοίβας και ελέγχω αν είναι λύση . Αν είναι έχω βρει την λύση και τερματίζω , διαφορετικά παράγω τα παιδιά του διανύσματος που έλεγξα και τα προσθέτω και αυτά στην στοίβα .

→ Επαναλαμβάνω το 3ο βήμα.

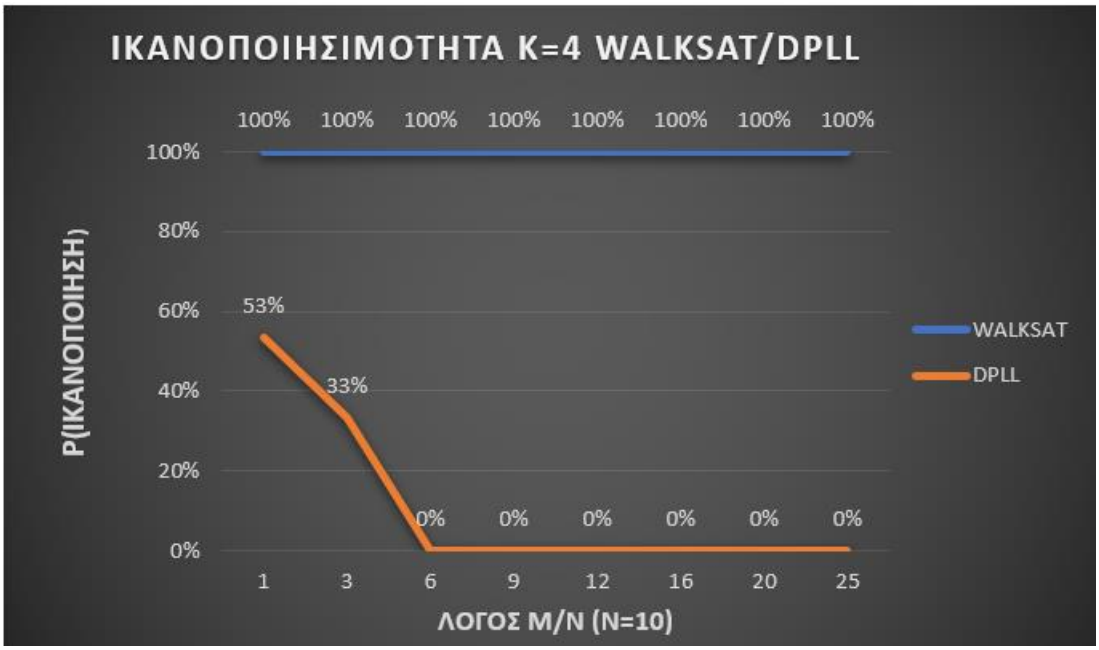
Εξηγώ συνοπτικά την **dpll**

Ουσιαστικά η depth first αλλά όχι εφαρμόζοντας προσθήκες όπως η hill climbing με τον walksat , αλλά εφαρμόζοντας μια διαφορετική προσέγγιση. Στην depth first όλα κινούνται στο πλαίσιο μια δομής δεδομένων και συστηματικά ενώ στην dpll δυναμικά μέσω της αναδρομής. Πιο συγκεκριμένα , στην depth first μετά από κάθε έλεγχο διανύσματος που βγάζω από την στοίβα και δεν είναι λύση, παράγονται αμέσως τα παιδιά του τα οποία και εισάγονται στην στοίβα. Αντιθέτως , στην dpll χρησιμοποιώ αναδρομική προσέγγιση και αφού ελέγξω ότι ένα διάνυσμα δεν αποτελεί λύση ,παράγω το παιδί του μέσω της αλλαγής του κατάλληλου στοιχείου του διανύσματος σε 1 οπού επαναλαμβάνω την διαδικασία για όλους τους απογόνους με την αλλαγή σε 1 μέχρι και το βάθος που έχει τεθεί ως όριο. Έπειτα ,επιστρέφω στο προηγούμενο επίπεδο ή διαφορετικά τον πατέρα και παράγω και ένα ακόμα παιδί μέσω της αλλαγής του κατάλληλου στοιχείου του διανύσματος σε -1 για όλους τους απογόνους ως και το ορισμένο βάθος. Ακόμη, στην depth first ελέγχω τα διανύσματα που βγάζω από την στοίβα ένα προς ένα αντιμετωπίζοντας τα ως αυτοτελή διανύσματα χωρίς να ορίζεται κάποια σχέση μεταξύ τους. Στην dpll όμως λόγω της σχέσης πατέρα και παιδιών μπορώ να ανεβαίνω από κάτω προς τα πάνω (αναδρομικά) επίπεδα και μιας και έχει προηγηθεί ο σχετικός έλεγχος για λύση , να εκχωρώ μηδενικές τιμές στους προγόνους ώστε φτάνοντας στο αρχικό επίπεδο να είναι απόλυτα ξεκάθαρο μέσω αυτού του μηδενισμού ότι όλοι οι απόγονοι μέχρι και το βάθος που μας περιορίζει , έχουν ελεγχθεί και δεν αποτελούν λύση.

Δεν παρατίθεται τμήμα κώδικα της depth και της dpll ώστε να συγκριθούν όπως παραπάνω με τον hill climbing και τον walksat δίοτι προηγουμένως ήταν προσθήκες στον ήδη υπάρχοντα κώδικα και ήταν απόλυτα ευδιάκριτο συγκριτικά με την περίπτωση της depth και της dpll που μπορεί η λογική να είναι ίδια όμως η υλοποίηση διαφέρει στην ολότητα της .

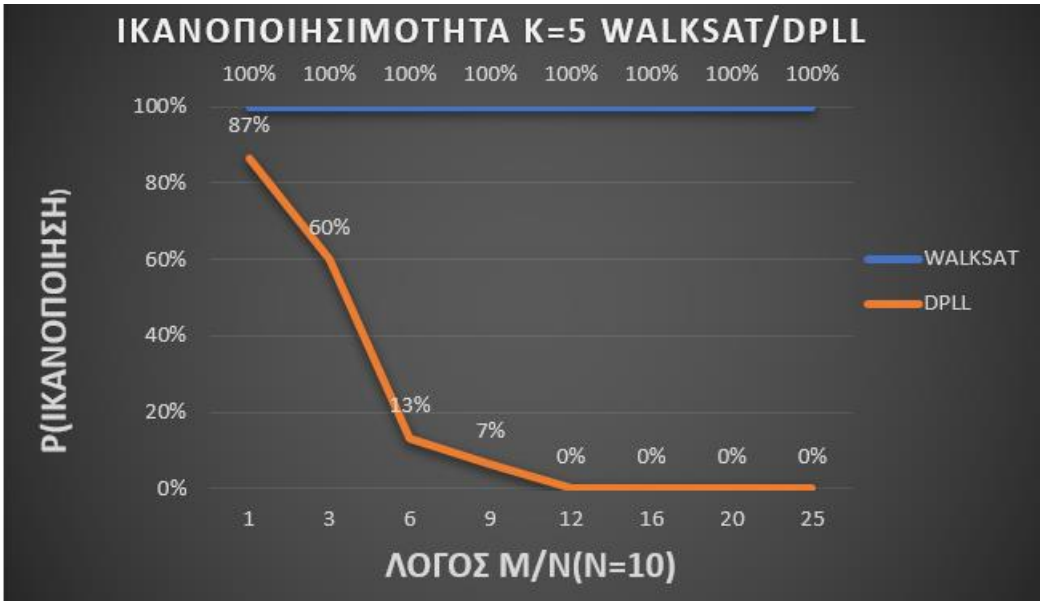
Δοκιμάζοντας τα προβλήματα του ερωτήματος Α στους εξελιγμένους αλγορίθμους του walksat και του dpll έχω τα παρακάτω συμπεράσματα.

Για την **ικανοποιησιμότητα για K=4** έχω :



Ικανοποιούνται απόλυτα όλες οι διαζεύξεις για K=4 και εφαρμογή του WalkSat σε αντίθεση με την εφαρμογή του DPLL που μέχρι τις πρώτες 30 διαζεύξεις η ικανοποιησιμότητα δεν ξεπερνά το 50% ενώ έπειτα αυτού η ικανοποιησιμότητα δεν υφίσταται.

Για την **ικανοποιησιμότητα για K=5** έχω :



Ικανοποιούνται απόλυτα όλες οι διαζεύξεις για K=5 και εφαρμογή του WalkSat σε αντίθεση με την εφαρμογή του DPLL που παρατηρείται η πρώτη αισθητή μείωση στην ικανοποιησιμότητα στις πρώτες 30 διαζεύξεις της τάξης του 30% , την οποία διαδέχεται μια ακομή μεγαλύτερη μείωση της ικανοποιησιμότητας στις επόμενες 30 διαζεύξεις της τάξης του 50% . Στις επόμενες 30 διαζεύξεις παρατηρείται μια ομαλή μείωση της ικανοποιησιμότητας και έπειτα δεν υπάρχει λύση για κανένα από τα προβλήματα σε όλες τις επόμενες διαζεύξεις.

Όσον αφορά τον **χρόνο** :

Για K=4

Για την DPLL παρατηρείται μια ελαφρώς αυξητική χρονική πορεία για το περιορισμένο αριθμό προβλημάτων που ικανοποιεί , σε κατώτερη όμως χρονικά κλίμακα από τον walksat. Αντιθέτως, ο walksat ακολουθεί μια μικρή αυξητική χρονικά πορεία στις πρώτες 30 διαζεύξεις την οποία διαδέχεται μια εξίσου μικρή μείωση στις επόμενες 30 διαζεύξεις . Έπειτα, ακολουθεί μια ομαλή ανοδική χρονικά πορεία για τις επόμενες 100 διαζεύξεις την οποία διαδέχεται μια απότομα μεγάλη χρονική αύξηση για τις επόμενες 40 διαζεύξεις και μια απότομη αλλά μικρή μείωση στις τελευταίες 50 διαζεύξεις.

Για K=5

Για την DPLL παρατηρείται μια ελαφρώς αυξητική πορεία χρονικά στις πρώτες 30 διαζεύξεις ,την οποία ακολουθεί μια απότομη και σχετικά μεγάλη μείωση στις επόμενες 30 διαζεύξεις και μια σταθερή χρονικά πορεία στις τελευταίες 30 διαζεύξεις μέχρι τις οποίες υφίσταται μια μικρή ικανοποιησιμότητα. Αντιθέτως, ο WalkSat ακολουθεί μια ομαλότατη αυξητική χρονικά πορεία σε όλο το εύρος των διαζεύξεων που υπάρχει ικανοποιησιμότητα.

Συνεπώς σε ένα πρόγραμμα που οι μέχρι τώρα επιλογές του χρήστη ήταν ο hill climbing και ο depth first ώστε να επιλέξει από την command line, προσθέτω και τον walksat και τον dpll . Το προσθέτω σε συγκεκριμένο σημείο της main υποδεικνύοντας ότι το 2ο όρισμα στο input του χρήστη από την command line μπορεί εκτός από τον hill και τον depth , να είναι και ο walksat ή ο dpll.

Το συγκεκριμένο μέρος του κώδικα

```
if (strcmp(argv[1], "hill") == 0) {
        start_time = clock();
        hill_climbing(argv);
        end_time = clock();
    } else if (strcmp(argv[1], "depth") == 0) {
        start_time = clock();
        depth_first(argv);
        end_time = clock();
```

```
}
else if (strcmp(argv[1], "walksat") == 0)
{
    start_time = clock();
   walkSat(argv);
   end_time = clock();
}

else if (strcmp(argv[1], "dpll") == 0)
{
   start_time = clock();
   dpll(argv);
   end_time = clock();
}
else {
   syntax_error(argv);
   return -1;
}
```

Και αλλάζω και το μήνυμα προς το χρήστη σχετικά με το τι θέλω να μου δώσει ως input

```
void syntax_error(char **argv) {
        printf("Use the following syntax:\n\n");
        printf("%s <method> <inputfile> <outputfile>\n\n",argv[0]);
        printf("where:\n");
        printf("<method> is either 'hill' or 'depth' or 'walksat' or 'dpll' (without the quotes)\n");
        printf("<inputfile> is the name of the file with the problem description\n");
        printf("<outputfile> is the name of the output file with the solution\n");
}
```

Ο επιπλέον κώδικας σχετικά με τον WalkSat και τον DPLL καθώς και ο default κώδικας σχετικά με την hill climbing και την depth first, δεν έχει προστεθεί στο bcsp.c ,το δοθέν δηλαδή πρόγραμμα, αλλά σε ένα αντίγραφο αυτού , το bcspPlus.c