ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ 2020-PROJECT 3

By Βάιος Λύτρας

Σημειώσεις για τον Κώδικα:

1. Όλος ο απαραίτητος κώδικας για την άσκηση είναι γραμμένος μέσα στο τέλος του αρχείου csp.py με πολύ λίγες αλλαγές στις συναρτήσεις που είχε από πριν και αυτές είναι κυρίως για τις μετρήσεις.
2. Για την αρχικοποίηση του κάθε προβλήματος έφτιαξα μια κλάση rlfa η οποία είναι υποκλάση της CSP παίρνει ως ορίσματα 3 ανοιγμένα αρχεία (var,dom,ctr) και: σετάρει τις μεταβλητές, σετάρει το λεξικό των domain της κάθε μεταβλητής, φτιάχνει το λεξικό-γράφο γειτόνων ανάλογα με το ποιες μεταβλητές έχουν constraint μεταξύ τους και τέλος εφόσον οι μεταβλητές δεν έχουν κάποιο γενικό και σταθερό constraint, φτιάχνω άλλο ένα λεξικό που κρατάει τα constraints (πχ. '0 1': ''> 1'').Όλα αυτά γίνονται διαβάζοντας σειρά-σειρά τα 3 files και παίζοντας με strings.
3. Οι αλγόριθμοι fc και mac ήταν ήδη υλοποιημένοι στο αρχείο κώδικα που μας δώθηκε μέσω της backtracking_search οπότε δεν έκανα κάτι πάνω σε αυτό. Πρόσθεσα λίγες γραμμές μόνο στην forward_checking (αναφέρεται παρακάτω τι) και κάποιες άλλες γραμμές που αυξάνουν τον global μετρητή που χρησιμοποιείται για τις μετρήσεις
4. Η ευριστική dom/wdeg υλοποιήθηκε βάζοντας το weight του κάθε constraint δίπλα από το constraint στο λεξικό με τα constraints (πχ . '0 1': (> 1,1)) και γίναν τα 2 μαζί ένα tuple. Η λειτουργία του είναι όπως ακριβώς λέει στην ιστοσελίδα δηλαδή μετράει τα weight από κάθε constraint που έχει μια μεταβλητή (δηλαδή από κάθε γείτονα) και στο τέλος διαιρούνται με το current domain size. Η μεταβλητή που έχει το μικρότερο αυτό πηλίκο επιλέγεται ως η επόμενη και επιστρέφεται.
5. Η συνάρτηση constraints που επιστρέφει true ή false ανάλογα με το αν ικανοποιείται ένα constraint είναι υλοποιημένη μέσα στην κλάση και με βάση την δομή του λεξικού που προαναφέρθηκε στην παρατήρηση 3, παίζει με strings integers και μετατροπές και επιστρέφει True αν το constraint ικανοποιείται. Επίσης αυξάνει το weight που χρησιμοποιείται από την ευριστική dom/wdeg κάθε φορά που είναι να επιστρέψει false και το current domain size της $2^{ης}$ μεταβλητής είναι 1 (επειδή αφού επιστρέψει false τότε θα αδειάσει).Επίσης να σημειωθεί ότι αυτή η λειτουργία υλοποιήθηκε μέσα στην συνάρτηση των constraints έτσι ώστε να μην την γράφω σε κάθε μέθοδο ξεχωριστά.
6. Ο FC-CBJ έχει πάρα πολλά ίδια χαρακτηριστικά με την συνάρτηση backtracking search που χρησιμοποιείται για τις άλλες μεθόδους με την μόνη διαφορά ότι έχει μέσα μια ειδική λειτουργία ώστε να κάνει backtrack σε πάνω από 1 κόμβους ανάλογα με το conflict set της κάθε μεταβλητής. Επίσης τα conflict set αρχικοποιούνται και αυτά μέσα

στην init της rlfa και προστίθενται μεταβλητές μέσα σε αυτά μέσω της συνάρτησης forward_checking. Για το πως ο αλγόριθμος κάνει backjump είναι σχολαστικώς σχολιασμένο με σχόλια μέσα στον κώδικα.

7. Η κλάση rlfa και οι συναρτήσεις που πρόσθεσα είναι γραμμένες κάτω κάτω στο αρχείο csp.py που πήρα από το AIMA μαζί με μία main .
8. Επειδή τα αρχεία τα πήρα έτσι όπως είναι από το github, έχω σβήσει τα παραδείγματα που είχε και έκοψα όσα κομμάτια δεν χρησιμοποίησα αλλά μπορεί μερικές συναρτήσεις που δεν χρησιμοποιούνται καν να υπάρχουν ακόμα μέσα. Επίσης πολλές συναρτήσεις που είχε από πριν δεν τις έχω πειράξει όπως επίσης τα αρχεία search.py και utils.py είναι απείραχτα απλά χρειαζόμουν κάποιες λειτουργίες μέσα από αυτά.
9. Στο κάτω κάτω μέρος του αρχείου υπάρχει μια “main” στην οποία επιλέγεις τα 3 files του στιγμιότυπου που θες να λύσεις και μετά αποσχολιάζεις όποιο κομμάτι κώδικα ανάλογα με ποια μέθοδο θέλεις να το λύσεις. Αν θέλουμε να χρησιμοποιήσουμε την ευριστική dom/wdeg βάζουμε όπου: select_unassigned_variable=dom_wdeg (αντίστοιχα για mrv).
10. Στο αρχείο dom7-w1-f4.txt λείπει μια αλλαγή γραμμής στην τελευταία γραμμή οπότε η init που έχω φτιάξει στην rlfa δεν μπορεί να σετάρει σωστά τα domains γιατί δεν μπορεί να καταλάβει που τελειώνει το file. Για να δουλέψει στο πρόγραμμα το πρόβλημα 7-w1-f4 πρέπει να προστεθεί αυτή η αλλαγή γραμμής στο τέλος του dom7-w1-f4.txt. Το ίδιο συμβαίνει και στο αρχείο dom11.
11. Υπάρχει μέσα στο πρόγραμμα μία global μεταβλητή constr_counter που αυξάνεται κάθε φορά που ελέγχουμε ένα constraint σε μία μέθοδο και χρησιμοποιείται για τις μετρήσεις του παρακάτω πίνακα. Υπάρχει κώδικας σε κάθε μέθοδο που την σταματάει αν η constr_counter περάσει ένα όριο και υπάρχει κώδικα σε κατάλληλα σημεία για κάθε μέθοδο που την αυξάνει.Το ίδιο συμβαίνει με την μεταβλητή assignment_counter.
12. Όλες οι μέθοδοι αν υπάρχει λύση επιστρέφουν και εκτυπώνουν ένα λεξικό με το assignment της κάθε μεταβλητής και μετά εκτυπώνουν τον constr_counter,cpu time,assignment_counter. Αν δεν υπάρχει λύση εκτυπώνουν “There is no solution!!” και ξανά τον constr_counter.
13. Αν θέλετε να τσεκάρετε κατά πόσο οι μέθοδοι που έφτιαξα επιστρέψουν σωστή λύση μπορείτε να τις δοκιμάσετε για το πρόβλημα 2-f24 που όλες εκτός από την MAC-DOM/WDEG το λύνουν σχεδόν αμέσως.

Μετρήσεις προβλήματος 1:

Σημειώσεις: Στον παρακάτω πίνακα οι γραμμές είναι τα στιγμιότυπα. Επίσης μία από τις μετρικές είναι **ο μέσος όρος των constraint checks σε 3 runtimes εκτός αν η πρώτη κάνει πάνω από ένα όριο constraint checks**. Το όριο για κάθε μέθοδο έχει οριστεί περίπου στα πόσα constraint checks κάνει σε 20 λεπτά. Πχ. η FC σε 20 λεπτά κάνει περίπου 40000K constraint checks ενώ η MAC-MRV περίπου 1000K λόγω του AC3b. Όπου τα checks είναι >ορίου θεωρήστε ότι cpu_time > 20min. **Τα assignments μετρήθηκαν και αυτά ως τον μέσο όρο από 3 runtimes, το ίδιο και για τον χρόνο της CPU**. Οι μετρήσεις για τους χρόνους είναι γραμμένες σε δευτερόλεπτα και έγιναν στο

δικό μου μηχάνημα καθώς από τα λινουξ της σχολής έλειπαν ορισμένες βιβλιοθήκες όπως η SortedContainers. Παρόλα αυτά δε νομίζω να υπάρχει μεγάλη διαφορά στους χρόνους.

Παρατηρήσεις: Γενικότερα μπορούμε να παρατηρήσουμε ότι ο FC-CBJ με την ευριστική dom/wdeg είναι ο καλύτερος αλγόριθμος γιατί λύνει όλα τα προβλήματα και στα περισσότερα δύσκολα στιγμιότυπα έχει τον μικρότερο αριθμό constraint checks και cpu_time (εκτός από το 11 που το έχει ο FC-dom/wdeg και μάλλον η φύση του προβλήματος το επιτρέπει αυτό). Σε εύκολα στιγμιότυπα τυχαίνει μερικές φορές αλγόριθμοι ''χειρότεροι'' από τον FC-CBJ- dom/wdeg να φέρνουν καλύτερα αποτελέσματα αλλά αυτό συμβαίνει επειδή στα εύκολα στιγμιότυπα όλοι οι αλγόριθμοι καλοί είναι. Επίσης μπορούμε να παρατηρήσουμε ότι ανεξαρτήτως των ευριστικών συναρτήσεων ότι ο αλγόριθμος MAC είναι ο χειρότερος όλων (εκτός του min-con). Άλλη μία παρατήρηση είναι ότι η ευριστική dom/wdeg σε σχέση με την mrv σε γενικές γραμμές αποδίδει καλύτερα όμως το πιο σημαντικό είναι ότι σε προβλήματα που δεν έχουν λύση (UNSAT) τα εντοπίζει με υπερβολικά πολύ πάρα πολύ λιγότερα constraint checks και χρόνο από την mrv.

Σημειώσεις για την Min-Con: Η min-con ήταν ήδη υλοποιημένη μέσα στον κώδικα με αριθμό max_steps=100000 ο οποίος είναι ένας καλός αριθμός. Επίσης η min-con σε προβλήματα που δεν έχουν λύση (UNSAT) δεν έχει τρόπο να διακρίνει ότι το πρόβλημα είναι UNSAT οπότε απλά θα τρέξει όλα τα βήματα και δεν θα βρει λύση μέσα σε αυτά.

Παρατηρήσεις για τη Min-Con: Η Min-Con δεν κατάφερε να λύσει ούτε ένα SAT στιγμιότυπο μέσα σε 100.000 βήματα και ο λόγος μάλλον είναι ο μεγάλος αριθμός περιορισμών και μεταβλητών στα προβλήματα RLFA.

Παρακάτω είναι ο πίνακας:

	FC-MRV	FC-dom/wdeg	MAC-MRV	MAC-dom/wdeg	FC-CBJ-MRV	FC-CBJ-dom/wdeg	MIN-CON
2-f24	Checks: ~20K Cpu: 0.18 Assigns: 612 (SAT)	Checks: ~43,5K Cpu: 0.34 Assigns: 410 (SAT)	Checks: ~25K Cpu: 27.185 Assigns:207520 (SAT)	Checks: >100K (>40minutes)	Checks: ~35K Cpu:0.13 Assigns: 401.6 (SAT)	Checks: ~22K Cpu: 0.19 Assigns: 261 (SAT)	Max steps exceeded
2-f25	Checks: >40000K	Checks: >40000K	Checks: >1000K	Checks: >100K	Checks: >40000K	Checks: ~19606K Cpu: 116.08 Assigns:107266 (UNSAT)	Max steps exceeded
3-f10	Checks: ~115K Cpu: 0.57 Assigns: 1437 (SAT)	Checks: ~8560K Cpu: 47.6 Assigns: 88563 (SAT)	Checks: ~27K Cpu: 2.11 Assigns: 454 (SAT)	Checks: >100K	Checks: ~341K Cpu: 1.31 Assigns: 2287 (SAT)	Checks: ~441K Cpu: 3.16 Assigns: 2230 (SAT)	Max steps exceeded

3-f11	Checks: >40000K	Checks: >40000K	Checks: >1000K	Checks: >100K	Checks: >40000K	Checks: ~1133K Cpu: 7.09 Assigns: 4387 (UNSAT)	Max steps exceeded
6-w2	Checks: >40000K	Checks: ~48K Cpu: 0.17 Assigns: 213 (UNSAT)	Checks: >1000K	(>40minutes) Checks: >50K	Checks: ~18363K Cpu: 61.5 Assigns: 164300 (UNSAT)	Checks: ~48K Cpu: 0.17 Assigns: 263 (UNSAT)	Max steps exceeded
7-w1-f4	Checks: >40000K	Checks: ~738K Cpu: 4,7 Assigns: 11885 (SAT)	Checks: >1000K	(>40minutes) Checks: >45K	Checks: ~2396K Cpu: 25.29 Assigns: 50974 (SAT)	Checks: ~125K Cpu: 1.2 Assigns: 2070 (SAT)	Max steps exceeded
	FC-MRV	FC-dom/wdeg	MAC-MRV	MAC-dom/wdeg	FC-CBJ-MRV	FC-CBJ-dom/wdeg	MIN-CON
7-w1-f5	Checks: >40000K	Checks: >40000K	Checks: >1000K	(>40minutes) Checks: >10K	Checks: >40000K	Checks: ~38000K Cpu: 305.2 Assigns: 262178 (UNSAT)	Max steps exceeded
8-f10	Checks: >40000K	Checks: >40000K	Checks: >1000K	Checks: >100K	Checks: ~1393K Cpu: 243.2 Assigns: 23500 (SAT)	Checks: ~100K Cpu: 115 Assigns: 21000 (SAT)	Max steps exceeded
8-f11	Checks: >40000K	Checks: >40000K	Checks: >1000K	Checks: >100K	Checks: >40000K (UNSAT)	Checks: ~73K Cpu: 0.75 Assigns: 322 (UNSAT)	Max steps exceeded
11	Checks: >40000K	Checks: ~716K Cpu: 9.5 Assigns: 3647 (SAT)	Checks: >1000K	Checks: >100K	Checks: ~7377K Cpu: 27.91 Assigns: 36613 (SAT)	~25035K Cpu: 250.5 Assigns: 132178 (SAT)	Max steps exceeded
14-f27	Checks: >40000K	Checks: ~18542K Cpu: 408.86 Assigns:470048 (SAT)	Checks: >1000K	Checks: >100K	Checks: ~1738K Cpu: 29.54 Assigns: 29382 (SAT)	Checks: ~386K Cpu: 12.08 Assigns: 2448 (SAT)	Max steps exceeded
14-f28	Checks: >40000K	~7001K Cpu: 217.96 Assigns: 132577 (UNSAT)	Checks: >1000K	Checks: >100K	Checks: ~1112K Cpu: 21.75 Assigns: 24068 (UNSAT)	Checks: ~163K Cpu: 3.97 Assigns: 945 (UNSAT)	Max steps exceeded