ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΧΩΡΗΜΕΝΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ & ΑΛΓΟΡΙΘΜΟΙ

ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ
για το πρόβλημα KNAPSACK 0-1

Ηλίας Λάμπρου
Α.Μ.: 56

31/12/2019

Επιβλέπων καθηγητής: ***Χρήστος Γκόγκος***

Περιεχόμενα

---

1. Περίληψη

Η αναφορά περιλαμβάνει πληροφορίες σχετικά με τους τρόπους λύσης του προβλήματος Knapsack 0-1 καθώς και ανάλυση των αποτελεσμάτων της επίλυσης 320 προβλημάτων με έξι διαφορετικούς αλγόριθμους.
Παρουσιάζονται πίνακες και διαγράμματα με τα αποτελέσματα των λύσεων που εξάγουν έξι διαφορετικού αλγόριθμοι για το πρόβλημα Knapsack 0-1.

2. Εισαγωγή

Το Knapsack 0-1 πρόβλημα είναι γνωστό και ως πρόβλημα του σακιδίου. Έχουμε ένα ορισμένο σύνολο αντικειμένων με διαφορετική αξία και βάρος το κάθε ένα και θέλουμε να εισάγουμε σε ένα σακίδιο, ένα ορισμένο αριθμό από αυτά ώστε να μην ξεπεράσουμε το βάρος που μας επιτρέπει το σακίδιο να εισάγουμε, αλλά και να πετύχουμε την μεγαλύτερη δυνατή αξία στα αντικείμενα του σακιδίου που θα εισαχθούν.

Για την επίλυση του προβλήματος θα έχουμε έξι διαφορετικούς τρόπους αναλόγως με τους αλγοριθμους που θα χρησιμοποιηθούν.
Θα γίνει επίλυση 320 προβλημάτων knapsack 0-1 με κάθε αλγόριθμο ξεχωριστά και τα αποτελέσματα των λύσεων θα εισαχθούν σε ένα αρχείο CSV για την δημιουργία στατιστικών.
Ο χρόνος επίλυσης του κάθε προβλήματος θα περιορίζεται σε ένα χρονικό όριο (time limit).

3. Αλγόριθμοι:

Οι αλγόριθμοι που θα χρησιμοποιηθούν είναι:

1. **Άπληστη μέθοδος** (Greedy Approach)
2. **Εξαντλητική απαρίθμηση συνδυασμών** (Brute Force Approach)
3. **Διακλάδωση και φραγή** (Branch and Bound)
4. **Δυναμικός προγραμματισμός** (Dynamic Programming)
5. **Δυναμικός προγραμματισμός -OR-Tools** (Dynamic Programming)
6. **Ακέραιος προγραμματισμός -OR-Tools** (Integer Programming)

Ακολουθεί μια ανάλυση σχετικά με τις δυνατότητες και τον τρόπο επίλυσης του προβλήματος για τους αλγορίθμους που χρησιμοποιήθηκαν.

1 Άπληστη μέθοδος (Greedy Approach)

Ο αλγόριθμος αυτός αρχικά ταξινομεί τα αντικείμενα του προβλήματος σύμφωνα με την μέγιστη αξία/βάρος και στη συνέχεια επιλέγει με την σειρά τα αντικείμενα με το μεγαλύτερο λόγο μέχρι να φτάσει στο όριο του βάρους.
Η λύση είναι γρήγορη αλλά δεν είναι η βέλτιστη γιατί μπορεί κάποιος άλλος συνδυασμός να είναι καλύτερος.

2 Εξαντλητική απαρίθμηση συνδυασμών (Brute Force Approach)

Ο αλγόριθμος αυτός δημιουργεί όλους τους δυνατούς συνδυασμούς αντικειμένων και επιλέγει τον συνδυασμό αντικειμένων που χωράνε στο σακίδιο αλλά έχουν και την μέγιστη αξία.
Το αποτέλεσμα είναι η βέλτιστη λύση αλλά αναλόγως το πρόβλημα η εύρεση της λύσης απαιτεί πάρα πολύ χρόνο με βαθμό πολυπλοκότητας O($n2^n$).

3 Διακλάδωση και φραγή (Branch and Bound)

Στη μέθοδο αυτή γίνεται προσπάθεια να μην ελεγχθούν από τον αλγόριθμο κάποιες ομάδες συνδυασμών οι οποίες δεν υπάρχει περίπτωση να δώσουν το βέλτιστο αποτέλεσμα. Με αυτόν τον τρόπο ο αλγόριθμος καταφέρνει να μειώσει το χρόνο επίλυσης αρκετά σε σχέση με τον αλγόριθμο Brute Force ενώ μας δίνει την βέλτιστη λύση με βαθμό πολυπλοκότητας O(2n-1 -1).

4 Δυναμικός προγραμματισμός (Dynamic Programming)

Ο δυναμικός προγραμματισμός (dynamic programming), αποτελεί ένα ισχυρό εργαλείο, για την επίλυση συνδυαστικών προβλημάτων βελτιστοποίησης όπως το πρόβλημα knapsack 0-1. Η επίλυση ενός προβλήματος, προκύπτει από την λύση μικρότερων υποπροβλημάτων τα οποία αλληλοεπικαλύπτονται (overlapping subproblems).
Κάθε υποπρόβλημα λύνεται μια μόνο φορά και η βέλτιστη λύση αποθηκεύεται σ΄ ένα πίνακα. Αυτό απαιτεί περισσότερη μνήμη ενώ ελαττώνεται σημαντικά ο χρόνος επίλυσης.
Με βαθμό πολυπλοκότητας O(n*W) μίας από τις καλύτερες επιλογές για την επίλυση του προβλήματος knapsack 0-1.

5. Dynamic Programming by OR-Tools

Ο πέμπτος αλγόριθμος που χρησιμοποιήθηκε περιλαμβάνεται στα εργαλεία OR-Tools και χρησιμοποιεί την μέθοδο του δυναμικού προγραμματισμού.

6. Ακέραιος προγραμματισμός (Integer Programming)

Ο ακέραιος προγραμματισμός (Integer Programming) αποτελεί έναν γενικό τρόπο επίλυσης προβλημάτων που είναι δυνατόν να μοντελοποιηθούν με συγκεκριμένο τρόπο. Στηρίζεται στον αλγόριθμο επαναληπτικής βελτίωσης simplex και είναι σε θέση να εντοπίζει λύσεις ακόμα και εάν υπάρχουν περιορισμοί ακεραιότητας για τις τιμές των μεταβλητών απόφασης.
Χρησιμοποιήθηκε επιλυτής ακέραιου προγραμματισμού από την εργαλειοθήκη του OR-Tools.

4. Επίλυση προβλημάτων

Αρχικά δημιουργήσαμε 320 στιγμιότυπα με τον generator του Pisinger.
Οι παράμετροι που χρησιμοποιήθηκαν για την δημιουργία του κάθε συνδυασμού ήταν: n={10,50,100,500}, r={50,100,500,1000} και type={1,2,3,4}. Δημιουργήθηκαν 5 προβλήματα για κάθε συνδυασμό (5 X 4 X 4 X 5 = 320 στιγμιότυπα).

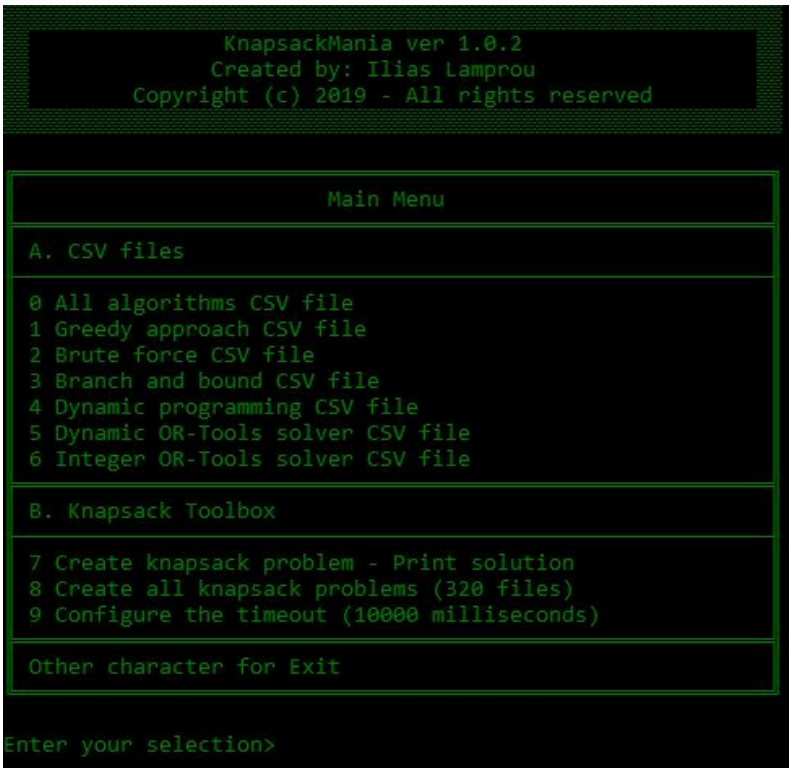
```
10
    1    46     4
    2    19    48
    3    20     4
    4    50    13
    5     6    28
    6    41    44
    7    20     1
    8    10     8
    9    45    19
   10    33    20
63
```

Εικόνα 1: Στην εικόνα βλέπουμε την μορφή ενός αρχείου προβλήματος knapsack. Στο επάνω μέρος ο αριθμός των αντικειμένων. Στη συνέχεια ακολουθούν οι αντίστοιχες γραμμές που περιλαμβάνουν τον αριθμό, αξία και βάρος των αντικειμένων και στην τελευταία γραμμή η χωρητικότητα του σακιδίου.

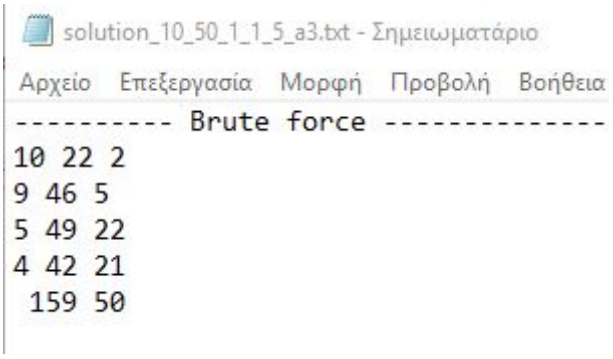
Στην συνέχεια μέσα από κώδικα σε γλώσσα C++ διαβάζαμε κάθε ένα αρχείο προβλήματος ξεχωριστά και βρίσκαμε την λύση του καλώντας τους 6 αλγορίθμους με την σειρά που είδαμε παραπάνω. Οι αλγόριθμοι του OR-Tools καλούνταν μέσω εξωτερικών αρχείων τα οποία είχαν ήδη μεταγλωτιστεί από πριν και ήταν σε μορφή εκτελέσιμων αρχείων.

Κάθε αλγόριθμος υπολόγιζε την αξία και το βάρος του καλύτερου δυνατού συνδυασμου ενώ γινόταν υπολογισμός και του χρόνου εκτέλεσης του αλγορίθμου σε milliseconds.
Τα στοιχεία αυτά αποθηκεύονταν σε αρχείο μορφής CSV ώστε στη συνέχεια να μπορέσουμε να βγάλουμε συμπεράσματα μέσω διαγραμμάτων.

Για να μπορεί να γίνει με ευκολία η όλη διαδικασία δημιουργίας και επίλυσης των προβλημάτων προστέθηκαν στην βασική εφαρμογή πρόσθετα εργαλεία και δυνατότητες όπως κεντρικό μενού, δυνατότητα αλλαγής του time limit, δυνατότητα δημιουργίας με ένα κλικ όλων των προβλημάτων και το σημαντικότερο από όλα ήταν η προσθήκη ενός εργαλείου δημιουργίας οποιουδήποτε προβλήματος knapsack και επίλυσής του με όποιον αλγόριθμο επιλέξουμε, με ταυτόχρονη αποθήκευση των αποτελεσμάτων αλλά και των επιλεγμένων αντικειμένων σε ξεχωριστά αρχεία.



Εικόνα 2: Το κεντρικό μενου της εφαρμογής - εργαλείου που κατασκευάστηκε για την επίλυση και διαχείριση της εργασίας



Εικόνα 3: Στην εικόνα βλέπουμε ένα αρχείο επίλυσης του προβλήματος με παραμέτρους 10,50,1,1,5. Οι γραμμές περιέχουν τα επιλεγμένα αντικείμενα και στην τελευταία σειρά βλέπουμε την αξία που πέτυχε ο αλγόριθμος Brute force και το αντίστοιχο βάρος των αντικειμένων

5. Αποτελέσματα

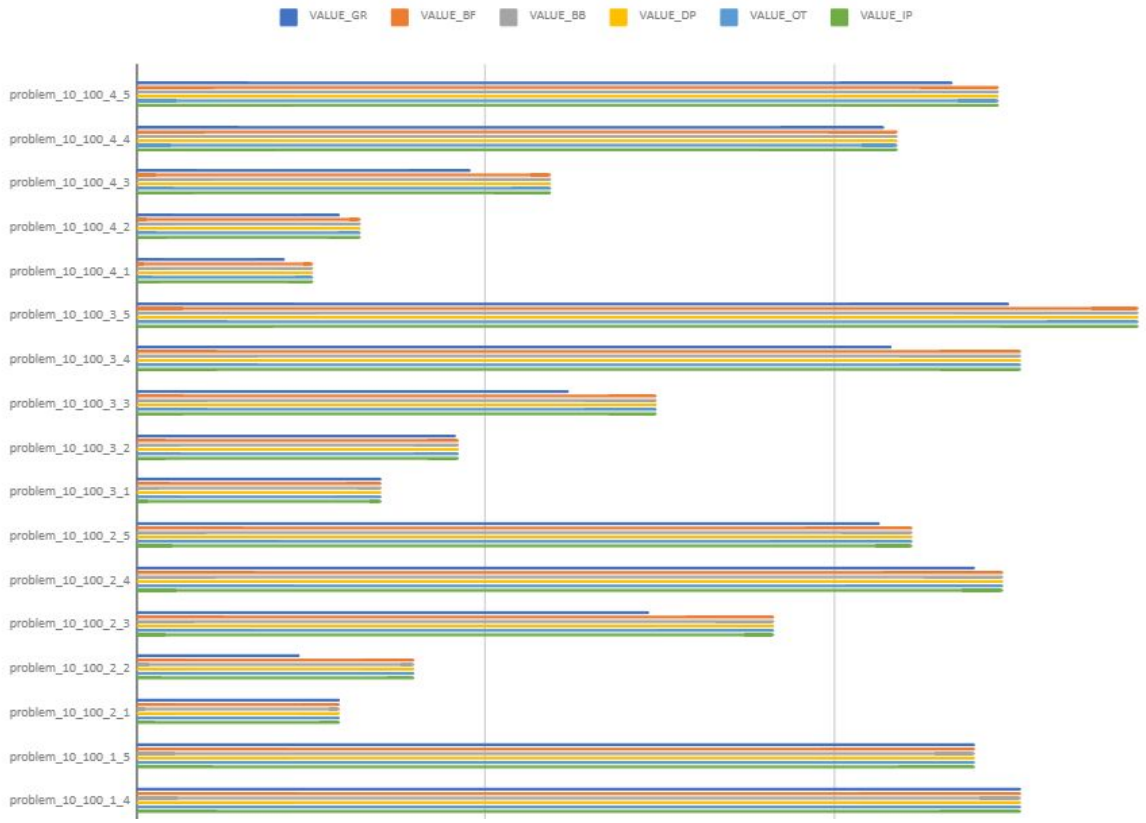
Παρουσιάζονται παρακάτω ορισμένες εικόνες από το αρχείο αποτελεσμάτων CSV καθώς και ορισμένα ενδεικτικά διαγράμματα.

INSTANCE	WEIGH	VALUE_	EXECU	WEIGH	VALUE_	EXECU	WEIGH	VALUE_	EXECU	WEIGH	VALUE_	EXECU	WEIGH	VALUE_	EXECU	WEIGH	VALUE_	EXECUT
problem_10_50_1_1_5.txt	50	159	0	50	159	0	50	159	0	50	159	0	50	159	0	50	159	7
problem_10_50_1_2_5.txt	61	214	0	61	214	0	61	214	0	61	214	0	61	214	0	61	214	7
problem_10_50_1_3_5.txt	100	184	0	121	199	0	121	199	0	121	199	0	121	199	0	121	199	11
problem_10_50_1_4_5.txt	221	246	0	221	246	0	221	246	0	221	246	0	221	246	0	221	246	8
problem_10_50_1_5_5.txt	242	213	0	242	213	0	242	213	0	242	213	0	242	213	0	242	213	6
problem_10_50_2_1_5.txt	48	63	1	50	64	0	50	64	0	50	64	0	50	64	0	50	64	9
problem_10_50_2_2_5.txt	30	45	0	61	79	0	61	79	0	61	79	0	61	79	0	61	79	11
problem_10_50_2_3_5.txt	102	113	0	122	130	0	122	130	0	122	130	0	122	130	0	122	130	11
problem_10_50_2_4_5.txt	185	195	2	228	233	0	226	233	0	228	233	0	228	233	0	226	233	11
problem_10_50_2_5_5.txt	222	226	0	237	238	0	240	238	0	237	238	0	237	238	0	240	238	16
problem_10_50_3_1_5.txt	43	73	1	49	79	0	49	79	0	49	79	0	49	79	0	49	79	13
problem_10_50_3_2_5.txt	66	126	0	75	135	0	75	135	0	75	135	0	75	135	0	75	135	12
problem_10_50_3_3_5.txt	112	182	0	112	182	0	112	182	0	112	182	0	112	182	0	112	182	7
problem_10_50_3_4_5.txt	213	283	0	238	308	0	238	308	0	238	308	0	238	308	0	238	308	16
problem_10_50_3_5_5.txt	197	287	0	200	290	0	200	290	0	200	290	0	200	290	0	200	290	9
problem_10_50_4_1_5.txt	35	35	0	51	51	0	51	51	0	51	51	0	51	51	0	51	51	9
problem_10_50_4_2_5.txt	50	50	1	79	79	0	79	79	0	79	79	0	79	79	0	79	79	11
problem_10_50_4_3_5.txt	92	92	0	113	113	0	113	113	0	113	113	0	113	113	0	113	113	6
problem_10_50_4_4_5.txt	229	229	1	238	238	0	238	238	0	238	238	0	238	238	0	238	238	12
problem_10_50_4_5_5.txt	168	168	0	204	204	0	204	204	0	204	204	0	204	204	0	204	204	5
problem_10_100_1_1_5.txt	86	198	0	86	198	0	86	198	0	86	198	0	86	198	0	86	198	9
problem_10_100_1_2_5.txt	145	371	0	145	371	0	145	371	0	145	371	0	145	371	0	145	371	7
problem_10_100_1_3_5.txt	321	323	1	342	324	0	342	324	0	342	324	0	342	324	0	342	324	11
problem_10_100_1_4_5.txt	462	507	1	462	507	0	462	507	0	462	507	0	462	507	0	462	507	6
problem_10_100_1_5_5.txt	433	481	0	433	481	0	433	481	0	433	481	0	433	481	0	433	481	7
problem_10_100_2_1_5.txt	100	117	0	100	117	0	100	117	0	100	117	0	100	117	0	100	117	6
problem_10_100_2_2_5.txt	82	94	0	146	159	0	146	159	0	146	159	0	146	159	0	146	159	8

Εικόνα 4: Στην εικόνα βλέπουμε την μορφή του αρχείου CSV με τις λύσεις όλων των αλγορίθμων για ένα μέρος μιας ομάδας προβλημάτων. Παρατηρούμε ότι επειδή τα αντικείμενα είναι 10 κανένας αλγόριθμος δεν βγήκε εκτός χρόνου.
Όλοι οι αλγόριθμοι εκτός τον Greedy Approach μας δίνουν το ιδιο (βέλτιστο) αποτέλεσμα

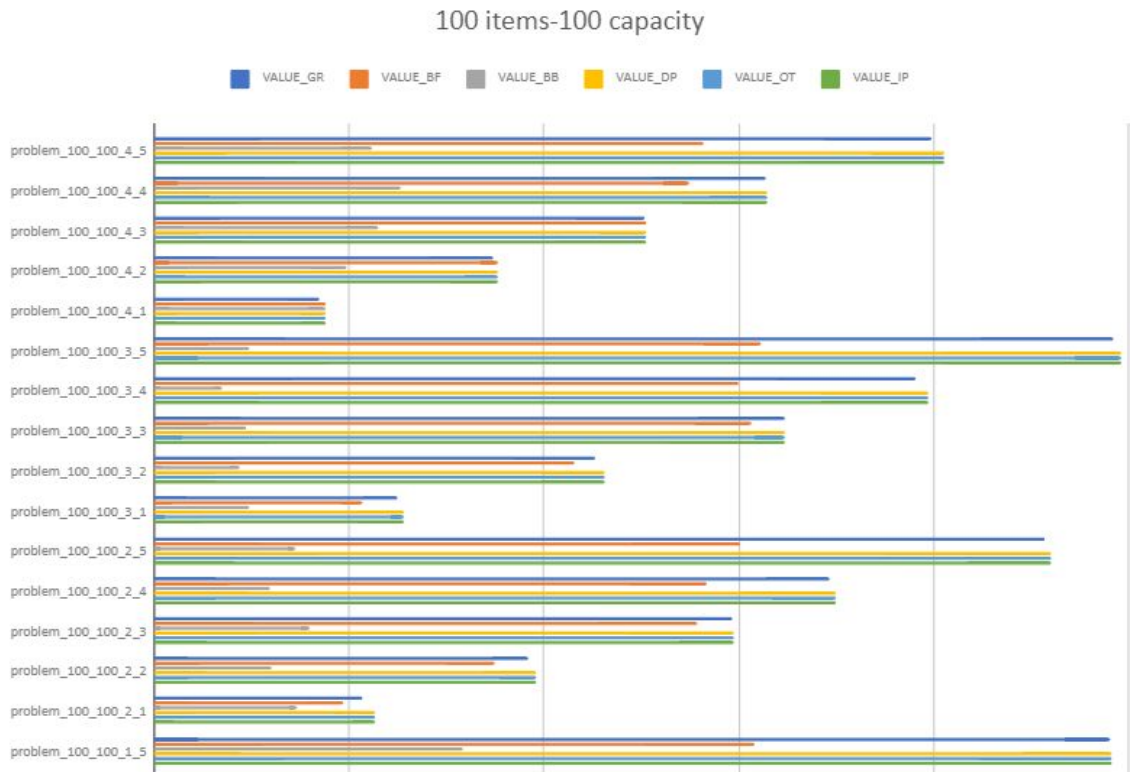
problem_50_50_3_1_5.txt	224	404	0	239	389	10001	239	419	106	239	419	0	239	419	0	239	419	17
problem_50_50_3_2_5.txt	390	700	0	398	638	10001	219	459	10001	398	708	0	398	708	0	398	708	15
problem_50_50_3_3_5.txt	658	998	0	658	968	10001	350	590	10001	660	1000	0	660	1000	0	660	1000	11
problem_50_50_3_4_5.txt	802	1212	2	824	1184	10001	209	449	10001	824	1234	0	824	1234	1	824	1234	21
problem_50_50_3_5_5.txt	981	1431	0	1001	1411	10001	309	549	10001	1005	1455	0	1005	1455	1	1005	1455	22
problem_50_50_4_1_5.txt	238	238	0	239	239	10001	239	239	0	239	239	0	239	239	0	239	239	7
problem_50_50_4_2_5.txt	381	381	1	398	398	10001	398	398	110	398	398	0	398	398	0	398	398	8
problem_50_50_4_3_5.txt	650	650	0	660	660	10001	650	650	10001	660	660	0	660	660	0	660	660	11
problem_50_50_4_4_5.txt	783	783	0	824	824	10001	688	688	10001	824	824	0	824	824	0	824	824	9
problem_50_50_4_5_5.txt	1000	1000	0	1005	1005	10001	570	570	10001	1005	1005	0	1005	1005	0	1005	1005	11
problem_50_100_1_1_5.txt	425	1067	1	455	889	10001	458	1098	8	458	1098	0	458	1098	0	458	1098	26
problem_50_100_1_2_5.txt	798	1730	0	802	1334	10001	798	1730	621	798	1730	0	798	1730	0	798	1730	18
problem_50_100_1_3_5.txt	1330	1866	2	1356	1586	10001	1371	1888	642	1371	1888	0	1371	1888	1	1371	1888	12
problem_50_100_1_4_5.txt	1594	2130	0	1616	1728	10001	1612	2137	6618	1616	2137	0	1616	2137	1	1612	2137	13
problem_50_100_1_5_5.txt	2004	2235	1	2024	1970	10001	1062	1608	10001	2080	2267	0	2080	2267	2	2080	2267	17
problem_50_100_2_1_5.txt	457	565	1	464	528	10001	464	570	13	464	570	0	464	570	0	464	570	12
problem_50_100_2_2_5.txt	749	873	0	807	882	10001	807	930	1433	807	930	0	807	930	0	807	930	27
problem_50_100_2_3_5.txt	1367	1501	0	1372	1466	10001	1270	1402	10001	1373	1504	0	1373	1504	0	1373	1504	20
problem_50_100_2_4_5.txt	1555	1695	0	1618	1699	10001	1112	1260	10001	1618	1752	0	1618	1752	1	1618	1752	30
problem_50_100_2_5_5.txt	2076	2162	0	2024	2046	10001	1120	1253	10001	2076	2162	0	2076	2162	2	2076	2162	14
problem_50_100_3_1_5.txt	433	633	1	439	579	10001	439	639	343	439	639	0	439	639	0	439	639	11
problem_50_100_3_2_5.txt	912	1182	0	931	1151	10001	791	1041	10001	931	1201	0	931	1201	0	931	1201	18

Εικόνα 5: Στην εικόνα παρατηρούμε ότι time out κάνουν μόνο οι αλγόριθμοι Brute Force και Branch and Bound. Σε αυτές τις περιπτώσεις επιστρέφουν την καλύτερη λύση που είχαν βρει μέχρι τότε. Παρατηρούμε ακόμη ότι ορισμένες φορές παρά το time out ο αλγόριθμος είχε ήδη βρει την καλύτερη λύση.
Ο καλύτερος χρόνος επίλυσης παρουσιάζεται στους δύο δυναμικούς αλγορίθμους, ενώ πολύ καλούς χρόνους παρατηρούμε και στον IP



Εικόνα 6: Στην εικόνα βλέπουμε ορισμένα αποτελέσματα για ομάδα προβλημάτων με 10 αντικείμενα. Μόνο ο αλγόριθμος Greedy Approach δεν δίνει το βέλτιστο αποτέλεσμα

Εικόνα 7: Με την αύξηση των αντικειμένων βλέπουμε ότι ο αλγόριθμοι Brute Force και Branch and Bound λόγω time out δεν δίνουν το βέλτιστο αποτέλεσμα και διαφοροποιούνται από τους υπόλοιπους.



Εικόνα 8: Για προβλήματα με 100 αντικείμενα έχουμε την ίδια ακριβώς εικόνα με την προηγούμενη περίπτωση με τους αλγόριθμους Brute Force και Branch and Bound λόγω time out να μην δίνουν το βέλτιστο αποτέλεσμα και διαφοροποιούνται από τους υπόλοιπους.

Εικόνα 8: Για προβλήματα με 500 αντικείμενα έχουμε τα ίδια ακριβώς αποτελέσματα

6. Συμπεράσματα

Συγκρίνοντας του έξι αλγορίθμους παρατηρούμε ότι οι αλγόριθμοι Dynamic Programming, Dynamic -OR-Tools και Integer Programming -OR-Tools, δίνουν πάντα το ίδιο αποτέλεσμα χωρίς να ξεπερνούν τον χρόνο time out.

Το συντομότερο χρόνο τον επιτυγχάνουν οι δύο δυναμικοί αλγόριθμοι.

Ο αλγόριθμος Greedy Approach δίνει πάντα αποτέλεσμα εντός του χρόνου αλλά το αποτέλεσμα όπως δείξαμε παραπάνω δεν είναι η καλύτερη λύση που μπορούμε να έχουμε.

Ο αλγόριθμος Brute force στις περιπτώσεις προβλημάτων με περισσότερα από 10 αντικείμενα ξεπέρασε το όριο του χρόνου με αποτέλεσμα να είναι ο χειρότερος από άποψη χρόνου μη μπορώντας σε αυτές τις περιπτώσεις να δώσει την καλύτερη λύση.

Ο αλγόριθμος Branch and Bound στις περιπτώσεις προβλημάτων με περισσότερα από 10 αντικείμενα στις περισσότερες φορές και αυτός ξεπέρασε το όριο του χρόνου μη δίνοντας σωστή λύση. Σε σχέση με τον Brute force σε ελάχιστες μόνο περιπτώσεις έδωσε αποτέλεσμα εντός του χρόνου για προβλήματα με 50, 100 ή 500 αντικείμενα. Οι επιδόσεις του θεωρούνται καλύτερες σε σχέση με τις επιδόσεις του Brute force αλλά η διαφορά δεν φάνηκε αισθητά στα γραφήματα που πήραμε.

Στις περιπτώσεις προβλημάτων με 10 αντικείμενα και οι δύο αλγόριθμοι εκτελούνταν σε χρόνο μικρότερο του 1ms.

Ίσως μια επιπλέον έρευνα με μεγαλύτερο time out θα μπορούσε να δείξει καλύτερα την διαφορά στην απόδοση του Branch and Bound.

Η επιλογή του ορίου χρόνου στα 10 sec κρίνεται εντελώς λανθασμένη αφού για τους υπόλοιπους αλγορίθμους δεν είχε καμιά σημασία αλλά και για τους δύο με τα μεγάλα ποσοστά time out δεν μπορούσε να μας δείξει τα πλεονεκτήματα του του Branch and Bound. Θα έπρεπε να επιλεγεί ένας χρόνος όπου ο Branch and Bound να μπορεί να λύσει αρκετά προβλήματα.

31-12-2019