ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Δικριτηριακός προγραμματισμός μιας μηχανής με χρόνους εξάρμωσης

Κοσμάς Χαριτωνίδης

Μεταπτυχιακή Εργασία

Ηράκλειο, Κρήτη
Μάρτιος 1993

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Δικριτηριακός προγραμματισμός μιας μηχανής με χρόνους εξάρμωσης

Εργασία που υποβλήθηκε από τον
ΚΟΣΜΑ Π. ΧΑΡΙΤΩΝΙΔΗ
ως μερική απαίτηση για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΞΕΙΔΙΚΕΥΣΗΣ

Ηράκλειο, Μάρτιος 1993

Συγγραφέας : . . . .
Τμήμα Επιστ. Υπολογιστών, 16 Μαρτίου 1993

Εισηγητική Επιτροπή

. . . .
Αναπληρωτής Καθηγητής Πάνος Κωνσταντόπουλος, Επόπτης

. . . .
Καθηγητής Στέλιος Ορφανουδάκης, Μέλος

. . . .
Αναπληρωτής Καθηγητής Κώστας Κουρκουμπέτης, Μέλος

Δεκτή : . . . .
Αναπληρωτής Καθηγητής Πάνος Κωνσταντόπουλος,
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Δικριτηριακός προγραμματισμός μιας μηχανής
με χρόνους εξάρμωσης

Μεταπτυχιακή Εργασία του
Κοσμά Χαριτωνίδη

Περίληψη

Η κατανομή των διαθέσιμων πόρων (πρώτες ύλες, μηχανές κ.α.) μίας βιομηχανικής επιχείρησης, με στόχο την ελαχιστοποίηση του κόστους παραγωγής των αγαθών, λέγεται Προγραμματισμός Παραγωγής. Κάθε πρόβλημα βέλτιστης διάταξης εργασιών σε μία ή περισσότερες μηχανές, που εμφανίζεται στον Προγραμματισμό Παραγωγής, ονομάζεται πρόβλημα Χρονικού Προγραμματισμού. Η παρούσα εργασία ασχολείται με την επίλυση ενός τέτοιου προβλήματος. Συγκεκριμένα, του προβλήματος εκείνου που, δοθέντων N εργασιών, που πρέπει να εκτελεστούν σε μία μηχανή, χωρισμένων σε B ομάδες, έτσι ώστε μεταξύ των εκτελέσεων δύο εργασιών που ανήκουν σε διαφορετικές ομάδες να μεσολαβεί ένα χρονικό διάστημα που η μηχανή μένει ανενεργή (χρόνος εξάρμωσης), ζητείται η ελαχιστοποίηση δύο κριτηρίων απόδοσης: του αθροίσματος των χρόνων αποπεράτωσης των εργασιών και της μέγιστης καθυστέρησης.

Τα πολυκριτηριακά προβλήματα Χρονικού Προγραμματισμού αποτελούν ρεαλιστικότερη απεικόνιση της πραγματικότητας από τα προβλήματα ενός κριτηρίου, γιατί στην πράξη ζητούνται συνήθως προγράμματα (διατάξεις εργασιών) με μια γενικά καλή συμπεριφορά, ως προς διάφορα κριτήρια. Η γενικότερη λύση ενός πολυκριτηριακού προβλήματος είναι ο υπολογισμός του συνόλου των αποδοτικών λύσεων, δηλαδή εκείνων για τις οποίες δεν υπάρχει κάποια άλλη με καλύτερη τιμή σε ένα κριτήριο και όχι χειρότερη στα υπόλοιπα. Αυτό είναι το ζητούμενο και στο πρόβλημα που πραγματεύεται η παρούσα εργασία.

Η εύρεση μιας αποδοτικής λύσης ανάγεται στην επίλυση του προβλήματος ελαχιστοποίησης του αθροίσματος των χρόνων αποπεράτωσης δοθέντος ότι, ως προς κατάλληλα μετατοπισμένες δεξιότερα στον άξονα του χρόνου (από τις αρχικές τους τιμές) προθεσμίες, δε θα καθυστερήσει καμία εργασία. Αντίθετα από άλλα προβλήματα Χρονικού Προγραμματισμού με χρόνους εξάρμωσης, το τελευταίο πρόβλημα δεν είναι διατεταγμένων ομάδων, δηλαδή δεν είναι γνωστή η διάταξη των εργασιών σε κάθε ομάδα από την αρχή, με αποτέλεσμα η πλοκή

του να μην είναι εκθετική ως προς τον αριθμό των ομάδων, Β, αλλά ως προς τον αριθμό των εργασιών, Ν.

Αφού παρουσιάσουμε αναλυτικά τα όσα είπαμε παραπάνω, στη συνέχεια της εργασίας παρουσιάζουμε μεθόδους επίλυσης προβλημάτων Χρονικού Προγραμματισμού που έχουν χρησιμοποιηθεί σε άλλες εργασίες (κυρίως μεθόδους υπολογισμού κάτω φράγματος για ένα κοινό σχήμα διαμερισμού και φραγής) και εξηγούμε γιατί καθεμία από τις μεθόδους αυτές δεν μπορεί να εφαρμοστεί στο πρόβλημά μας. Οι βαθύτεροι λόγοι γι αυτό είναι πάντα η ύπαρξη των χρόνων εξάρμωσης και το γεγονός ότι το πρόβλημα δεν είναι διατεταγμένων ομάδων.

Στη συνέχεια επειχηρούμε μία συγκριτική παρουσίαση των κλασσικότερων αλγορίθμων Δυναμικού Προγραμματισμού που χρησιμοποιούνται για την επίλυση προβλημάτων Χρονικού Προγραμματισμού, και αναλύουμε τα προτερήματα και τα μειονεκτήματα τους έναντι του γενικού αλγορίθμου διαμερισμού και φραγής που συνήθως χρησιμοποίειται. Αμέσως μετά αναλύουμε τους λόγους που μας όθησαν στην επιλογή του Δυναμικού Προγραμματισμού για την επίλυση του προβλήματός μας και παρουσιάζουμε τους δύο αλγορίθμους Δυναμικού Προγραμματισμού που υλοποίησαμε.

Έπειτα παρουσιάζουμε τα αποτελέσματα που πήραμε από την εκτέλεση εκτεταμένων πειραμάτων και εξηγούμε την παρατηρούμενη επίδραση του πλήθους των εργασιών, Ν, του πλήθους των ομάδων, Β, της θέσης που κατέχει κάποια αποδοτική λύση ανάμεσα στις υπόλοιπες και διάφορων παραμέτρων του προβλήματος, στον απαιτούμενο χρόνο επίλυσής του. Το μέγεθος των προβλημάτων κάθε αποδοτική λύση των οποίων υπολογίζεται σε χρόνο ενός λεπτού (μέχρι λίγο περισσότερες από 20 εργασίες) είναι μικρότερο του μεγέθους των προβλημάτων που σε άλλες εργασίες, που ασχολούνται με προβλήματα Χρονικού Προγραμματισμού χωρίς χρόνους εξάρμωσης, λύνονται στον ίδιο χρόνο (συνήθως γύρω στις 30 εργασίες, χωρίς να λείπουν και λίγες περιπτώσεις που φτάνουν τις 50), αλλά κρίνεται ικανοποιητικό αν συγκριθεί με τις επιδόσεις αλγορίθμου, που παρουσιάζεται σε πρόσφατα δημοσιευμένη εργασία, για την επίλυση του προβλήματος που προκύπτει εάν από το δικό μας αφαιρέσουμε το κριτήριο της μέγιστης καθυστέρησης και που η λύση του αποτελεί μία από τις αποδοτικές λύσεις του δικού μας προβλήματος.

Τελειώνοντας, παρουσιάζουμε μία γενική ιδέα που μπορεί να ακολουθεί ένας ευρηματικός αλγόριθμος, ο οποίος πολύ ταχύτερα από τον ακριβή θα λύνει προσεγγιστικά το δικριτηριακό μας πρόβλημα. Παρουσιάζουμε, επίσης, μία γενίκευση του αλγορίθμου Δυναμικού Προγραμματισμού που χρησιμοποιήσαμε για τον υπολογισμό μιας αποδοτικής λύσης, που λύνει απευθείας, με μεγαλύτερη όμως υπολογιστική πλοκή, το αρχικό δικριτηριακό πρόβλημα.

Επόπτης εργασίας: Πάνος Κωνσταντόπουλος
Αναπληρωτής Καθηγητής Επιστήμης Υπολογιστών,
Πανεπιστημίου Κρήτης

One machine bicriterion scheduling
with set-up times

Master's Thesis by
Kosmas Haritonides

Abstract

Allocation of resources of an industrial enterprise, aiming to the minimization of the production cost, is called Production Scheduling. Each problem of optimal job sequencing (order) in one or more machines, is called Scheduling Problem. In the present work we deal with the solution of one such bicriterion problem: the minimization of the total finishing time and the maximum tardiness of N jobs in one machine, given that the jobs are distributed to B different groups, and a setup time is interposed between the executions of two jobs belonging to different groups.

Multicriteria Scheduling Problems are in fact a more realistic view of reality compared to one-criterion problems. The most general solution of a multicriteria problem is the computation of the set of efficient solutions, that is, the solutions for which there is no other with better value in one criterion and not worse in the rest. This is the issue to the problem which we deal with in this work.

The finding of an efficient solution can be achieved by solving the problem of minimization of the total completion time, given that, according to modified due dates in a right direction adjustment mode, no job will be tardy. This problem does not belong to the class of ordered batch scheduling problems, and so its complexity is exponential to the number of jobs N - not the number of groups B.

After the analytical presentation of what we have said until now, we present solution methods of Scheduling Problems which were used in other works (basically, methods to derive lower bounds for a common branch and bound scheme), and we explain the reasons for which none of these methods can be applied to our problem. The main reason for this is the existence of setup times as well as the fact that our problem is not an ordered batch problem.

Then we present a comparative study of the most common Dynamic Programming algorithms used for the solution of Scheduling Problems, and we analyze their advantages and disadvantages towards the generally used branch and bound algorithm. The reasons for which we used Dynamic Programming as the solution technique to our problem, as well as the two algorithms implemented, are then presented.

We next give the results taken from extended experimentation, and discuss the influence of the number of jobs N, the number of groups B, the position of an efficient solution inside a set of other solutions, and other problem parameters to the solution time. The size of problems (a few more than 20 jobs) for which each efficient solution is found within 1 minute CPU time, is smaller than the size of problems solved, in the same computational time, in other works dealing with Scheduling Problems without setup times (usually around 30 jobs but in a few cases they reach 50). On the other hand, it can be characterized satisfactory when compared to the effectiveness of a recently presented algorithm which is used for the solution of the problem without the tardiness criterion, and whose solution is one of the efficient solutions of our problem.

Finally, we give a general idea for a heuristic algorithm that can solve approximately our bicriterion problem in much less time. We also present a generalization of the Dynamic Programming algorithm that we used to find one efficient solution. This generalized algorithm can solve directly, but with greater complexity, the initial bicriterion problem.

Thesis supervisor : Panos Constantopoulos
Associate Professor of Computer Science,
University of Crete

Ευχαριστίες

Θα ήθελα, καταρχήν, να ευχαριστήσω τον επόπτη καθηγητή της εργασίας μου, κύριο Πάνο Κωνσταντόπουλο, για την πολύτιμη καθοδήγηση, τις συμβουλές και τις υποδείξεις του σε όλη την εξέλιξη της παρούσας εργασίας.

Θα ήθελα, έπειτα, να ευχαριστήσω τον συμφοιτητή Νίκο Τσατσάκη για το χρόνο που αφιέρωσε, στο ξεκίνημα κυρίως της ενασχόλησής μου με την περιοχή, σε συζητήσεις και παροχή συμβουλών, που ήταν για μένα χρήσιμες και κατατοπιστικές, για τη βοήθειά του στη μετάφραση της περίληψης της παρούσας εργασίας, καθώς και για την παραχώρηση πολλών χρήσιμων εργασιών, κάτι για το οποίο θα ήθελα να ευχαριστήσω και τον Θαλή Γεωργίου. Επίσης θα ήθελα να ευχαριστήσω τον φίλο Μιχάλη Ανδρουλάκη που μου έστειλε, από αθηναϊκή βιβλιοθήκη, μερικές πολύ χρήσιμες εργασίες.

Θα ήθελα, επίσης, να ευχαριστήσω το Ινστιτούτο Πληροφορικής για την υλικοτεχνική και οικονομική υποστήριξη που μου παρείχε κατά τη διάρκεια των μεταπτυχιακών σπουδών μου.

Επιθυμώ, επίσης, να ευχαριστήσω πολύ θερμά τους συμφοιτητές και φίλους Μαρία Καραβασίλη, Γιώργο Ξουρή, Πόπη Χαλκιά και Δημήτρη Κοτζίνο που αφιέρωσαν μεγάλο και πολύτιμο μέρος από τον χρόνο τους για να πληκτρολογήσουν το μεγαλύτερο μέρος της παρούσας εργασίας, όταν έμοιαζε μάλλον αδύνατον να προλάβαινα εγώ να το κάνω αυτό. Χωρίς τη βοήθειά τους, είναι σχεδόν σίγουρο ότι η παρούσα εργασία δεν θα είχε ολοκληρωθεί. Θα ήθελα να ευχαριστήσω επιπλέον τον Γιώργο Ξουρή και τον Γιάννη Κοπιδάκη για τη βοήθεια που μου προσέφεραν στην προετοιμασία της παρουσίασης της παρούσας εργασίας.

Θα ήθελα, τελειώνοντας, να ευχαριστήσω όλους εκείνους (μερικοί από τους οποίους είναι όσοι έχουν αναφερθεί, αλλά υπάρχουν και πολλοί άλλοι, φίλοι και συγγενείς, που δεν είναι πρακτικά δυνατό να κατονομαστούν) που, ο καθένας με τον τρόπο του, μου συμπαραστάθηκαν ηθικά στη δύσκολη περίοδο της συγγραφής της παρούσας εργασίας.

Περιεχόμενα

1. Εισαγωγή

1.1. Η θέση του χρονικού προγραμματισμού στον προγραμματισμό παραγωγής

Το ανταγωνιστικό περιβάλλον των βιομηχανικών επιχειρήσεων και η επιδίωξη της μεγιστοποίησης του κέρδους προβάλλουν έντονη την απαίτηση της όσο το δυνατόν αποδοτικότερης αξιοποίησης των διαθέσιμων πόρων (πρώτες ύλες, μηχανές, ανθρώπινο δυναμικό κ.α.) κάθε τέτοιας επιχείρησης. Η κατανομή των πόρων αυτών, για ένα συγκεκριμένο χρονικό ορίζοντα, που στοχεύει στην καλύτερη δυνατή ικανοποίηση κάποιων κριτηρίων απόδοσης, που ανάγονται πάντα σε μεγιστοποίηση κέρδους (ή ελαχιστοποίηση κόστους), λέγεται Προγραμματισμός Παραγωγής (Production Scheduling) ([Ters-85], [RoWh-88]).

Λόγω της σύνθετης φύσης του ο Προγραμματισμός Παραγωγής διαχωρίζεται σε επίπεδα που αντιστοιχούν σε διαφορετικούς χρονικούς ορίζοντες, που ποικίλουν κατά περίπτωση. Ο προγραμματισμός των υψηλότερων επιπέδων (μεγαλύτεροι χρονικοί ορίζοντες) αποτελεί περιορισμό για τα χαμηλότερα επίπεδα (μικρότεροι χρονικοί ορίζοντες). Δεν υπάρχει ομοφωνία μεταξύ των ειδικών για το ποιός είναι ο σωστότερος διαχωρισμός. Ένας κάπως λεπτομερής διαχωρισμός είναι σε 5 επίπεδα ([Ters-85]): Στρατηγικός Προγραμματισμός, Αθροιστικός Προγραμματισμός Παραγωγής, Βασικός Προγραμματισμός Παραγωγής, Προγραμματισμός των αναγκών σε υλικά και Λεπτομερειακός Προγραμματισμός. Στον Προγραμματισμό των αναγκών σε υλικά καθορίζεται ποιές ακριβώς εργασίες πρέπει να εκτελεστούν στις διαθέσιμες μηχανές και μέχρι πότε, σε μία σχετικά μικρή χρονική περίοδο, και στον Λεπτομερειακό Προγραμματισμό αποφασίζεται το πότε ακριβώς θα εκτελεστεί η κάθε εργασία. Κάθε πρόβλημα βέλτιστης διάταξης εργασιών σε μία ή περισσότερες μηχανές, που εμφανίζεται στον Λεπτομερειακό Προγραμματισμό, ονομάζεται πρόβλημα Χρονικού Προγραμματισμού. Όταν η στιγμή έναρξης της εκτέλεσης κάποιας διάταξης εργασιών είναι καθορισμένη, η διάταξη λέγεται πρόγραμμα. Οι όροι "μηχανές" και "εργασίες" μπορεί να αναφέρονται σε άλλα μη βιομηχανικά περιβάλλοντα, όπως σ' ένα υπολογιστικό σύστημα όπου "μηχανές" είναι οι επεξεργαστές και "εργασίες" οι προς εκτέλεση διεργασίες. Η μεγάλη πλειοψηφία όμως των προβλημάτων Χρονικού Προγραμματισμού προέρχεται από το περιβάλλον της βιομηχανίας.

Παριστάνοντας τα προβλήματα Χρονικού Προγραμματισμού ως συνδυαστικά προβλήματα εμφανίζεται η ανάγκη χρησιμοποίησης ηλεκτρονικού υπολογιστή (που είναι έτσι κι αλλιώς απαραίτητος στα σύγχρονα παραγωγικά συστήματα) για την επίλυσή τους. Η παράσταση αυτή προϋποθέτει συνήθως κάποια αφαίρεση ([RoWh-88]) γιατί η πραγματικότητα είναι συχνά ιδιαίτερα πολύπλοκη, υπάρχουν παράγοντες που είναι δύσκολο να παρασταθούν μαθηματικά (π.χ. ιδιαιτερότητες κάποιων πελατών), ενδέχεται να υπάρχουν ατέλειες και ασάφειες στη διατύπωση του προβλήματος και κυρίως στο καθορισμό των στόχων οι οποίοι μπορεί να είναι πολλοί και αντικρουόμενοι και δεν αποκλείεται, τέλος, η εμφάνιση απρόβλεπτων καταστάσεων (βλάβη μηχανών, απεργία προσωπικού κ.α.). Το πρόβλημα Χρονικού Προγραμματισμού που διαμορφώνεται ακόμα και μετά την αφαίρεση αυτή είναι σχεδόν πάντα NP-hard. Παρ’ όλα αυτά θεωρείται ([Ters-85]) ότι το σύνθετο πρόβλημα του Προγραμματισμού της Παραγωγής είναι αδύνατο να αντιμετωπισθεί αποτελεσματικά χωρίς τη βοήθεια του ηλεκτρονικού υπολογιστή, ο οποίος προτείνει απλώς κάποιες λύσεις που είναι συχνά κοντά σ’ αυτήν που αναζητεί ο χρήστης.

Αντιμετωπίζοντας κανείς ένα πρόβλημα Χρονικού Προγραμματισμου θέτει έναν από τους εξής δύο στόχους: είτε να βρει τη βέλτιστη λύση του προβλήματος, κάτι που σχεδόν πάντα ξέρει εκ των προτέρων ότι θα είναι χρονοβόρο για μεσαίου και μεγάλου μεγέθους προβλήματα αφού αυτά θα είναι NP-hard, είτε να βρει γρήγορα μία "σχετικά καλή" προσεγγιστική λύση του προβλήματος. Το δεύτερο επιτυγχάνεται με την επινόηση ευρηματικών αλγορίθμων. Το πρώτο επιτυγχάνεται συνήθως με μεθόδους συνδυαστικής βελτιστοποίησης, κυρίως απαριθμητικές μεθόδους (αλγόριθμοι διαμερισμού και φραγής, δυναμικός προγραμματισμός) και λιγότερο 0-1 γραμμικό προγραμματισμό, αλλά και μεθόδους τεχνητής νοημοσήνης (έμπειρα συστήματα). Η όλη διαδικασία μπορεί να υποβοηθείται σε πολύπλοκα συστήματα από κάποιο μοντέλο προσομοίωσης ([Ters-85]).

1.2. Κατηγορίες προβλημάτων Χρονικού Προγραμματισμού

Τα προβλήματα Χρονικού Προγραμματισμού μπορούν να χωριστούν σε κατηγορίες με διάφορους τρόπους, όπως περιγράφεται αναλυτικά στην εργασία [Grav-81]. Ένας σημαντικός διαχωρισμός είναι σε προβλήματα που αναφέρονται σε open shops και προβλήματα που αναφέρονται σε closed shops. Όταν οι εντολές παραγωγής (οι εργασίες, δηλαδή) προέρχονται άμεσα από παραγγελίες πελατών έχουμε open shop. Όταν υπάρχει απόθεμα από το οποίο ικανοποιούνται οι παραγγελίες και οι εντολές παραγωγής εκδίδονται από τον μηχανικό παραγωγής όταν αυτός το κρίνει κατάλληλο για συμπλήρωση του αποθέματος ώστε αυτό να είναι πάντα αρκετό έχουμε closed shop. Η διαφορά από τη σκοπιά αυτού που καλείται να λύσει το πρόβλημα είναι ότι στο open shop πρέπει να αποφασίσει μονάχα το πότε θα εκτελεστούν οι απαιτούμενες εργασίες παραγωγής, ενώ στο closed shop εκτός του πότε πρέπει να αποφασίσει και το τι θα παραχθεί και σε τι ποσότητα. Συνήθως σ’ ένα πρόβλημα που αναφέρεται σε closed shop αποφασίζεται πρώτα το τι και πόσο θα παραχθεί (σε επίπεδο Προγραμματισμού αναγκών σε υλικά) και έτσι ανάγεται σε πρόβλημα open shop που αυτό που μένει να αποφασιστεί είναι το πότε θα εκτελεστούν οι εντολές παραγωγής (επίπεδο Λεπτομερειακού Προγραμματισμού).

Ένας άλλος διαχωρισμός είναι ανάλογα με την πολυπλοκότητα της απαιτούμενης εργασίας. Εδώ έχουμε προβλήματα ενός σταδίου (ή επιπέδου) και προβλήματα πολλών σταδίων. Στο πρόβλημα ενός σταδίου έχουμε μία ή περισσότερες πανομοιότυπες μηχανές και κάθε εργασία πρέπει να εκτελεσθεί ακριβώς μία φορά σε μία οποιαδήποτε από τις μηχανές (ή στη μία και μοναδική στην περίπτωση της μίας μηχανής). Στα προβλήματα πολλών σταδίων έχουμε πολλές, διαφορετικές εν γένει, μηχανές και κάθε εργασία πρέπει να εκτελεσθεί σε ένα υποσύνολο αυτών (ίσως σε όλες), πιθανώς με κάποια προκαθορισμένη σειρά. Εάν υπάρχει τέτοια σειρά και είναι ίδια για όλες τις εργασίες το περιβάλλον παραγωγής λέγεται flow shop. Αλλιώς job shop.

Ένας άλλος διαχωρισμός των προβλημάτων είναι σε αιτιοκρατικά και στοχαστικά. Όταν τα δεδομένα που φτάνουν στα χέρια του μηχανικού παραγωγής είναι συγκεκριμένοι αριθμοί μιλάμε για αιτιοκρατικό πρόβλημα, ενώ αν κάποια από αυτά είναι τυχαίες μεταβλητές με συγκεκριμένη κατανομή μιλάμε για στοχαστικά. Ένας άλλος διαχωρισμός είναι σε στατικά και δυναμικά. Όταν όλα τα δεδομένα του προβλήματος είναι γνωστά (ακόμα κι αν περιέχονται σ' αυτά κάποιες τυχαίες μεταβλητές) τη στιγμή που λύνεται το πρόβλημα, έχουμε στατικό πρόβλημα, ενώ όταν δεν αποκλείονται αλλαγές ή προσθήκες στα δεδομένα (συνηθέστερη περίπτωση είναι η εμφάνιση νέων εργασιών) έχουμε δυναμικό πρόβλημα. Τα περισσότερα προβλήματα που εμφανίζονται στην πράξη είναι στοχαστικά και δυναμικά, αλλά τα περισσότερα υλοποιημένα μοντέλα Χρονικού Προγραμματισμού καθώς και οι περισσότερες εργασίες που εμφανίζονται στη βιβλιογραφία αναφέρονται σε αιτιοκρατικά και στατικά προβλήματα.

Κάποια ακόμα στοιχεία των προβλημάτων Χρονικού Προγραμματισμού που μπορούν να χρησιμοποιηθούν για το διαχωρισμό τους σε κατηγορίες είναι η ύπαρξη ή όχι ομάδων εργασιών, έτσι ώστε όταν τελειώνει η εκτέλεση μιας εργασίας που ανήκει σε κάποια ομάδα κι αρχίζει η εκτέλεση εργασίας που ανήκει σε άλλη ομάδα να απαιτείται κάποιος χρόνος (ή κόστος) εξάρμωσης (set-up time ή set-up cost), που λέγεται και χρόνος (κόστος) αλλαγής κατεργασίας, κατά τον οποίον η μηχανή δεν εκτελεί καμία εργασία, το εάν επιτρέπεται ή όχι η διακοπή της εκτέλεσης μιας εργασίας για να εκτελεστεί κάποια ή κάποιες άλλες (οπότε έχουμε preemptive και non-preemptive προβλήματα, αντίστοιχα) και το εάν έχουμε ένα ή περισσότερα κριτήρια αξιολόγησης των προγραμμάτων και ποια είναι αυτά. Τα συνηθέστερα κριτήρια απόδοσης είναι ο μέσος ή μέγιστος χρόνος αποπεράτωσης των εργασιών, ο αριθμός των καθυστερημένων εργασιών, όταν υπάρχουν προθεσμίες, η μέση ή μέγιστη καθυστέρηση, ο βαθμός χρισιμοποίησης των μηχανών, καθώς και σταθμισμένα τα παραπάνω ή συνδυασμένα.

Για τους χρόνους εξάρμωσης που αναφέραμε παραπάνω και που έχουν, όπως θα δούμε αργότερα, βασικό ρόλο στην εργασία αυτή, μπορεί να ισχύει ένα από τα εξής τρία ενδεχόμενα: είτε να εξαρτώνται και από την ομάδα στην οποία ανήκε η τελευταία εργασία που εκτελέστηκε πριν την παρέμβαση του χρόνου εξάρμωσης και από την ομάδα που ανήκει η εργασία που θα εκτελεστεί αμέσως μετά, οπότε λέγονται εξαρτημένοι ακολουθίας και συμβολίζονται με s_{ij}, είτε να εξαρτώνται μόνο από την ομάδα που θα ακολουθήσει (αλλά όχι από αυτήν από την οποία μόλις εκτελέστηκε κάποια εργασία) οπότε λέγονται ανεξάρτητοι ακολουθίας και συμβολίζονται με s_j, είτε να είναι ανεξάρτητοι και από τις δύο ομάδες οπότε λέγονται (όπως και είναι) σταθεροί και συμβολίζονται με s.

1.3. Το αντικείμενο της παρούσας εργασίας

Το πρόβλημα που θα μας απασχολήσει στην εργασία αυτή αναφέρεται σε open shop, ενός σταδίου με μία μηχανή. Αυτή η κατηγορία προβλημάτων, τα οποία λέγονται και προβλήματα μιας μηχανής, έχει ερευνηθεί περισσότερο από κάθε άλλη γιατί παρότι είναι η απλούστερη περιέχει πολλά ενδιαφέροντα NP-hard προβλήματα που είναι καλό να αντιμετωπίζονται μόνα τους προτού αποτελέσουν μέρος άλλων γενικότερων προβλημάτων. Το πρόβλημα που θα μας απασχολήσει είναι επίσης αιτιοκρατικό και στατικό, υποθέτει ύπαρξη ομάδων εργασιών άρα και χρόνων εξάρμωσης και δεν επιτρέπει preemption. Δύο είναι τα κριτήρια που μας ενδιαφέρουν: το άθροισμα των χρόνων αποπεράτωσης και η μέγιστη καθυστέρηση. Στόχος είναι ο υπολογισμός του συνόλου των αποδοτικών λύσεων. Μία λύση ενός πολυκριτηριακού προβλήματος λέγεται αποδοτική όταν δεν υπάρχει άλλη καλύτερη ως προς ένα κριτήριο που να μην είναι χειρότερη ως προς κάποιο άλλο.

Ένας συνηθισμένος συνοπτικός τρόπος συμβολισμού των προβλημάτων Χρονικού Προγραμματισμού (που δεν παρέχει πάντα όλες τις λεπτομέρειες του προβλήματος αλλά μόνο τα βασικά χαρακτηριστικά του) έχει τη μορφή **α/β/γ**, όπου το κάθε γράμμα δηλώνει: **α**: τον αριθμό των μηχανών, **β**: κάποια ιδιαιτερότητα του προβλήματος (χρόνοι εξάρμωσης, σχέσεις προτεραιότητας μεταξύ των εργασιών, preemption κ.λ.π.) και **γ**: το κριτήριο ή τα κριτήρια βελτιστοποίησης. (Συχνά σημειώνεται και ο αριθμός των εργασιών, που είναι πάντα n και άρα η αναφορά του δεν προσθέτει κάποια πληροφορία.) Σύμφωνα με το συμβολισμό αυτόν το πρόβλημα με το οποίο θα ασχοληθούμε στην εργασία αυτή είναι το $1/S_{ij}/\sum C_i, T_{\max}$, όπου με C_i εννοούμε, όπως θα πούμε κι αργότερα, τον χρόνο αποπεράτωσης της εργασίας i και με $T_{\max}$ τη μέγιστη εμφανιζόμενη καθυστέρηση. Η καθυστέρηση T_i της εργασίας i ορίζεται σαν max $\{0, C_i - d_i\}$, όπου d_i είναι η προθεσμία της, ενώ η βραδύτητά της, L_i, σαν $C_i - d_i$ και η "νωρίτητα" της E_i σαν $d_i - C_i = -L_i$.

1.4. Προβλήματα Χρονικού Προγραμματισμού με χρόνους εξάρμωσης

Συγκριτικά με τον μεγάλο αριθμό εργασιών που έχουν δημοσιευτεί σχετικά με προβλήματα Χρονικού Προγραμματισμού γενικά, πολύ λίγες είναι αυτές που ασχολούνται με προβλήματα Χρονικού Προγραμματισμού με χρόνους εξάρμωσης. Το 1972 ο Sahney παρουσιάζει μια εργασία [Sahn-72] που πραγματεύεται το εξής πρόβλημα: Έχουμε δύο μηχανές και N εργασίες που καθεμιά πρέπει να εκτελεσθεί σε μία συγκεκριμένη από τις δύο μηχανές. Ο ένας και μοναδικός χειριστής των μηχανών, που έχουμε στη διάθεσή μας, πρέπει να βρίσκεται κοντά στη μηχανή που εκτελεί κάποια εργασία (και άρα μακριά από την άλλη) και χρειάζεται κάποιος χρόνος για τη μετακίνησή του από τη μία μηχανή στην άλλη. Το ζητούμενο είναι να βρεθούν οι διατάξεις των εργασιών στις δύο μηχανές ώστε να ελαχιστοποιείται ο μέσος χρόνος αποπεράτωσης (πρόβλημα που μπορεί να εκφραστεί συνοπτικά: 2/ ένα επίπεδο, χρόνος μετάβασης /$\sum C_i$). Το πρόβλημα λύνεται με έναν αλγόριθμο διαμερισμού και φραγής πολυπλοκότητας O(2^n) (κάθε κόμβος του δένδρου έρευνας διακλαδίζεται σε δύο, που αντιστοιχούν σε μετακίνηση ή όχι του χειριστή) και δεν παρουσιάζονται χρόνοι εκτέλεσης κάποιου προγράμματος που υλοποιεί τον αλγόριθμο.

Το 1975 οι Uskup και Smith παρουσιάζουν μία εργασία [UsSm-75] για το εξής πρόβλημα: N εργασίες πρέπει να εκτελεστούν σε δύο μηχανές στη σειρά έτσι ώστε η εκτέλεση κάθε εργασίας στη δεύτερη μηχανή να έχει τελειώσει πριν από κάποια προθεσμία συγκεκριμένη για κάθε εργασία. Ανάμεσα στις εκτελέσεις των εργασιών παρεμβάλλονται χρόνοι εξάρμωσης εξαρτημένοι ακολουθίας. Στόχος είναι η ελαχιστοποίηση του αθροίσματος των εμφανιζόμενων χρόνων εξάρμωσης. Το πρόβλημα αυτό μπορεί να συμβολιστεί 2/ 2 στάδια, $s_{ij}/\sum s_{ij}$. Λύνεται επαναληπτικά από έναν αλγόριθμο που βασίζεται σε ένα μηχανισμό διαμερισμού και φραγής (που αντιστοιχεί στη δεύτερη μηχανή) που θέτει επαναληπτικά σε λειτουργία έναν άλλο μηχανισμό διαμερισμού και φραγής (που αντιστοιχεί στην πρώτη μηχανή). Η υπολογιστική πολυπλοκότητα είναι O($n!^2$). Υπάρχουν βέβαια, όπως γίνεται συνήθως σ’ αυτές τις περιπτώσεις, μηχανισμοί περιορισμού του χώρου έρευνας. Αναφέρεται ότι προβλήματα 10 εργασιών χρειάζονται, σε έναν IBM 360/75, από ένα κλάσμα του δευτερολέπτου μέχρι μερικά δευτερόλεπτα, 20 εργασιών από 45 δευτερόλεπτα μέχρι 3 λεπτά και δύο προβλήματα 30 εργασιών που εκτελέστηκαν χρειάστηκαν 4 και 6 λεπτά.

Το 1978 οι Bruno και Downey [BrDo-78] αποδεικνύουν κάτι πολύ σημαντικό για το πρόβλημα μιας μηχανής με χρόνους εξάρμωσης και προθεσμίες. Ότι το πρόβλημα "δοθεισών N εργασιών που ανήκουν σε B ομάδες και χρόνου εξάρμωσης S που παρεμβάλλεται μεταξύ των εκτελέσεων δύο εργασιών διαφορετικών ομάδων, υπάρχει διάταξη των εργασιών τέτοια ώστε κάθε εργασία να τελειώνει πριν την προθεσμία της;" είναι NP-complete. Η απόδειξη έγινε ανάγοντας το πρόβλημα του εκδρομικού σάκου (knapsack problem) σε μια πολύ ειδική περίπτωση του παραπάνω (συγκεκριμένα για 3 εργασίες ανά ομάδα και 3 διαφορετικές προθεσμίες). Η παραπάνω πρόταση είναι πολύ σημαντική γιατί συνεπάγεται άμεσα ότι το πρόβλημα ελαχιστοποίησης της μέγιστης καθυστέρησης (1/S/$T_{\max}$) και το πρόβλημα ελαχιστοποίησης του αριθμού των καθυστερημένων εργασιών (1/S/n_T - όπου με n_T συμβολίζεται το πλήθος των καθυστερημένων εργασιών και με wn_T το σταθμισμένο πλήθος) σε μία μηχανή με χρόνους εξάρμωσης είναι NP-hard. Και φυσικά τα παραπάνω επεκτείνονται για χρόνους εξάρμωσης ανεξάρτητους και εξαρτημένους ακολουθίας.

Το 1980 ο Ψαραύτης [Psar-80] δημοσιεύει μια εργασία για το εξής γενικό πρόβλημα: Εχουμε πάλι N εργασίες που ανήκουν σε B ομάδες. Οι εργασίες της ίδιας ομάδας έχουν τον ίδιο χρόνο επεξεργασίας. Χρόνοι εξάρμωσης εξαρτημένοι ακολουθίας παρέμβαλονται μεταξύ διαδοχικών εκτελέσεων εργασιών από διαφορετικές ομάδες. Οταν εκτελείται μια εργασία της ομάδας n αμέσως μετά από μια εργασία της ομάδας m (δεν αποκλείεται m = n) και απομένουν ακόμη k_i εργασίες προς εκτέλεση από την ομάδα i (i = 1, ..., B) προκύπτει ένα κόστος f(m, n, k_1, k_2, ..., k_B). Ζητείται διάταξη των εργασιών τέτοια ώστε να ελαχιστοποιήται το άθροισμα των κοστών. Για το πρόβλημα αυτο αναπτύσσει έναν αλγόριθμο Δυναμικού Προγραμματισμού με πολυπλοκότητα $O(B^2N^B)$. Επίσης αποδεικνύει ότι το πρόβλημα ελαχιστοποίησης του μέγιστου χρόνου αποπεράτωσης ($1/S_{ij}/C_{\max}$) (που είναι ουσιαστικά το TSP) και το πρόβλημα ελαχιστοποίησης του σταθμισμένου αθροίσματος των χρόνων αποπεράτωσης ($1/S_{ij}/\sum w_i C_i$), όταν βέβαια οι εργασίες της ίδιας ομάδας έχουν ίσο χρόνο επεξεργασίας ανάγονται στο γενικό πρόβλημα που απασχολεί αυτόν και άρα μπορούν να λυθούν σε $O(B^2N^B)$, που είναι πολύ λιγότερο από O(n!) που θα απαιτούσε η απαρίθμηση όλων των δυνατών διατάξεων και, το βασικό, δεν είναι εκθετικό ως προς τον αριθμό των εργασιών N αλλά ως προς τον αριθμό των ομάδων B.

Το 1989 οι Monma και Potts [MoPo-89] δημοσιεύουν μια εργασία στην οποία αποδεικνύουν κάτι πολύ σημαντικό: υπάρχει βέλτιστη λύση του προβλήματος της μέγιστης καθυστέρησης (σε μια μηχανή με χρόνους εξάρμωσης εξαρτημένους ακολουθίας) ($1/S_{ij}/T_{\max}$) όπου οι εργασίες σε κάθε ομάδα είναι διατεταγμένες κατά αύξουσα σειρά των προθεσμιών (earliest due date - EDD), υπάρχει βέλτιστη λύση του προβλήματος του σταθμισμένου αθροίσματος των χρόνων αποπεράτωσης (n/$1/S_{ij}/\sum w_i C_i$) που έχει τις εργασίες διατεταγμένες κατά αύξουσα σειρά των λόγων P_i/w_i (όπου w_i το βάρος της εργασίας i) και υπάρχει βέλτιστη λύση του προβλήματος του σταθμισμένου αθροίσματος των καθυστερημένων εργασιών ($1/S_{ij}/wn_T$) στην οποία οι μη καθυστερημένες εργασίες κάθε ομάδας είναι διατεταγμένες κατά EDD. Ορίζουν σαν πρόβλημα διατεταγμένων ομάδων κάθε πρόβλημα με χρόνους εξάρμωσης στον οποίο η διάταξη σε κάθε ομάδα είναι γνωστή. Τέτοια είναι τα δύο πρώτα από τα τρία που προαναφέραμε. Για την επίλυση τού γενικού προβλήματος διαταγμένων ομάδων παρουσιάζει έναν αλγόριθμο Δυναμικού Προγραμματισμού με πολυπλοκότητα O($B^2N^B\min\{N^s,T\}$), όπου s είναι ο αριθμός των διαφορετικών τιμών που μπορούν να πάρουν οι χρόνοι εξάρμωσης και T είναι η μέγιστη δυνατή τιμή που μπορεί να πάρει το $C_{\max}$ (δηλ. $T=\sum_{k=1}^{B}\sum_{i=1}^{N_k}(\max_{0\le a\le B}\{S_{ak}\}+P_i)$, όπου N_k είναι ο αριθμός των εργασιών της ομάδας k). Δίνουν επίσης έναν αλγόριθμο Δυναμικού Προγραμματισμού για το πρόβλημα του σταθμισμένου αριθμού των καθυστερημένων εργασιών με πολυπλοκότητα O($B^2N^B\min\{W,D,T\}$), όπου $W=\sum_{i=1}^{N}w_i$, $D=\max_{1\le i\le N}d_i$ και T όπως πριν. Οι παραπάνω εκφράσεις είναι σημαντικές γιατί, όπως και ο αλγόριθμος του Ψαραύτη, δίνουν πολυπλοκότητα εκθετική ως προς τον αριθμό των ομάδων B και όχι ως προς τον αριθμό των εργασιών N. Τέλος αποδεικνύουν ότι τέσσερα προβλήματα που αναφέρονται σε δύο ή περισσότερες πανομοιότυπες παράλληλες μηχανές, οταν το preemption επιτρέπεται, είναι NP-hard. Πρόκειται για τα προβλήματα με κριτήρια το μεγιστο χρόνο αποπεράτωσης, τη μέγιστη καθυστέρηση, το σταθμισμένο άθροισμα των χρόνων αποπεράτωσης και τον αριθμό των καθυστερημένων εργασιών. Η απόδειξη βασίζεται κυρίως σε αναγωγή του προβλήματος της διαμέρισης σε κάποιο από αυτά. Οταν το preemption δεν επιτρέπεται, έχει αποδειχτεί από άλλους ότι τα προβλήματα αυτά είναι NP-hard ακόμα και χωρίς χρόνους εξάρμωσης.

Λίγο αργότερα, το 1990, οι Ahn και Hyun [AhHy-90] δημοσιεύουν μια εργασία που πραγματεύεται το $1/s_{ij}/\sum C_i$. Λύνουν το πρόβλημα με έναν αλγόριθμο Δυναμικού Προγραμματισμού με πολυποκότητα O(B^2N^B), γενικεύοντας τον αλγόριθμο του Ψαραύτη που αναφέρεται σε περίπτωση που οι εργασίες κάθε ομάδας έχουν ίσους χρόνους επεξεργασίας και βελτιώνοντας την πολυπλοκότητα του αλγορίθμου των Monma και Potts. Παρουσιάζουν και έναν ευρηματικό αλγόριθμο για γρήγορη ανεύρεση προσεγγιστικής λύσης. Τα μεγέθη των προβλημάτων που λύθηκαν σε λιγότερο από 2 λεπτά σε σταθμούς εργασίας HP3000 και HP9000 από προγράμματα που υλοποιούν τον αλγόριθμο Δυναμικού Προγραμματισμού, είναι: 60 εργασίες χωρισμένες σε 3 ομάδες (23.5 δευτερόλεπτα κατά μέσο όρο), 40 εργασίες σε 4 ομάδες (66.5 δευτερόλεπτα.), 25 εργασίες σε 5 ομάδες (53.8 δευτ.), 18 εργασίες σε 6 ομάδες (39.6 δευτ.) και 14 εργασίες σε 7 ομάδες (27.7 δευτ.). Προβλήματα μνήμης (η διαθέσιμη είναι 12Mbytes) εμφανίστηκαν σε προβλήματα που αν λύνονταν θα απαιτούσαν πολύ περισσότερο χρόνο από τους παραπάνω. Συγκεκριμένα τα μεγαλύτερα προβλήματα για τα οποία παρουσιάζεται χρόνος επίλυσης, λύθηκαν σε χρόνο από 6.5 έως 22 λεπτά (ανάλογα με τις τιμές των παραμέτρων N και B). Οπότε εκείνα τα προβλήματα που απαιτούν περισσότερη μνήμη από τη διαθέσιμη, θα λύνονταν εάν υπήρχε η απαιτούμενη μνήμη σε περισσότερο χρόνο από 6.5 έως 22 λεπτά ανάλογα με την περίπτωση.

Το ίδιο ακριβώς πρόβλημα απασχόλησε τον Τσατσάκη στην μεταπτυχιακή του εργασία [Τσατ-93]. Ο Τσατσάκης έλυσε το πρόβλημα με έναν αλγόριθμο διαμερισμού και φραγής με πολυπλοκότητα O(B^N) και πρότεινε έναν ευρηματικό αλγόριθμο που προσεγγίζει τη βέλτιστη λύση καλύτερα από τον ευρηματικό αλγόριθμο των Ahn και Hyun.

Λίγο πρίν, το 1991, ο Γεωργίου [Γεωρ-91] είχε ασχοληθεί με το $1/s_{ij}/T_{\max}$ προτείνοντας για την επίλυσή του έναν αλγόριθμο διαμερισμού και φραγής με πολυπλοκότητα O(2^n) καθώς και έναν ευρηματικό αλγόριθμο για γρήγορες προσεγγιστικές λύσεις. Ο χρόνος εκτέλεσης του προγράμματος που υλοποιεί τον ακριβή αλγόριθμο επηρεάζεται έντονα από το εύρος των προθεσμιών r. Οσο αυτό αυξάνεται τόσο αυξάνεται και ο απαιτούμενος χρόνος εκτέλεσης. Το r κυμαίνεται στο διάστημα (0, 1). Για r = 0.4 λύνονται μέσα σε ένα λεπτό, σε ένα SPARCstation 2, προβλήματα 45 εργασιών χωρισμένες σε 10, 15 και 20 ομάδες, 50 εργασιών χωρισμένες σε 7 ομάδες και 65 εργασιών χωρισμένες σε 5 ομάδες. Για r = 0.6 λύνονται προβλήματα 45 εργασιών χωρισμένες σε 5 ομάδες. Ο προσεγγιστικός αλγόριθμος έχει πολύ καλύτερες επιδόσεις. Εκτελείται 30-35 φορές ταχύτερα από τον ακριβή, λύνοντας σε χρόνο τάξης ενός λεπτού για r = 0.4 προβλήματα 55 εργασιών χωρισμένες σε 10, 15 και 20 ομάδες, 60 εργασιών χωρισμένες σε 7 ομάδες και 75 εργασιών χωρισμένες σε 5 ομάδες και για r = 0.6 προβλήματα 50 εργασιών χωρισμένες σε 10, 15 και 20 ομάδες. Η πιθανότητα επίτευξης βελτίστου είναι 70-80% και το μέσο σχετικό σφάλμα, μετρημένο μόνο ως προς τα προβλήματα που δε λύνονται βέλτιστα, 1.5-2%.

Ενα άλλο πρόβλημα που περιέχει χρόνους εξάρμωσης και για το οποίο έχουν γραφτεί κάποιες εργασίες έχει την εξής γενική μορφή: Μια μηχανή παράγει προιόντα που τοποθετούνται σε ένα δοχείο άπειρης χωρητικότητας. Χρόνος κατασκευής των προιόντων θεωρείται η στιγμή που το δοχείο απομακρύνεται από τη μηχανή για να αντικατασταθεί από ένα νέο, διαδικασία που απαιτεί ένα χρόνο εξάρμωσης (κατά τον οποίο η μηχανή δεν δουλεύει). Ζητούνται η σειρά παραγωγής των προιόντων και οι χρόνοι αλλαγής των δοχείων, με στόχο την ελαχιστοποίηση του μέσου χρόνου αποπεράτωσης. Για την περίπτωση που οι χρόνοι κατασκευής των προιόντων είναι ίσοι, έχουν βρεθεί πολυωνυμικοί (και μάλιστα γραμμικοί) αλγόριθμοι ([NaSa-88], [SaMa-85] και [CNY-89] σε μία διαφορετική γενικότερη μορφή).

Τέτοια προβλήματα όμως (που λέγονται batch size) καθώς και flowshop προβλήματα με χρόνους εξάρμωσης για τα οποία έχουν επίσης δημοσιευθεί μερικές εργασίες απέχουν πολύ από αυτά που θα απασχολήσουν εμάς.

1.5. Πολυκριτηριακά προβλήματα Χρονικού Προγραμματισμού

Υπάρχουν περιπτώσεις που εκφράζοντας κανείς ένα πρόβλημά που εμφανίζεται στην πράξη ως συνδυαστικό πρόβλημα, διαπιστώνει ότι ενδιαφέρεται για περισσότερα από ένα κριτήρια απόδοσης και έτσι κατασκευάζει ένα πολυκριτηριακό συνδυαστικό πρόβλημα. Αντιμετωπίζοντας κανείς ένα πολυκριτηριακό πρόβλημα θα ήθελε να βρεί μία εφικτή λύση που να βελτιστοποιεί όλα τα κριτήρια. Επειδή τέτοια λύση (που λέγεται ιδανική) γενικά δεν υπάρχει, καλείται σχεδόν πάντα να κάνει ένα από τα εξής:

α) να αναπτύξει ένα διαλογικό αλγόριθμο ο οποίος θα έχει σαν στόχο να διερευνήσει τις προτιμήσεις του χρήστη (αυτού που θα πάρει τη τελική απόφαση) και να του προτείνει, μετά από μια σειρά ερωταποκρίσεων, την πιο κατάλληλη για τις προτιμήσεις του λύση,

β) να ιεραρχήσει τα κριτήρια και από όσες λύσεις βελτιστοποιούν το πρωτεύον να διαλέξει αυτές που βελτιστοποιούν το δευτερεύον και από αυτές όσες βελτιστοποιούν το τρίτο κατά σειρά κ.ο.κ,

γ) να υπολογίσει μια συνάρτηση από το $R^m \to R$ (συνάρτηση χρησιμότητας), όταν υπαρχουν m κριτήρια, που σε κάθε διάνυσμα τιμών των m αντικειμενικών συναρτήσεων θα αντιστοιχεί έναν πραγματικό αριθμό, ενδεικτικό της προτίμησης του χρήστη για το διάνυσμα αυτό (και άρα για την αντίστοιχη λύση) και να λύσει το μονοκριτηριακό πρόβλημα που προκύπτει με αντικειμενική συνάρτηση την συνάρτηση χρησιμότητας,

δ) να υπολογίσει το σύνολο των αποδοτικών λύσεων.

Στα προβλήματα Χρονικού Προγραμματισμού που αποτελούν, έτσι κι αλλιώς, αφαιρετικό μοντέλο του πραγματικού βιομηχανικού προβλήματος, μια από τις απλουστεύσεις που συνήθως γίνονται είναι να θεωρείται ένα μόνο κριτήριο απόδοσης. Στα περισσότερα όμως πραγματικά προβλήματα ενδιαφέρεται κανείς να βρεί ένα πρόγραμμα που θα έχει μια "γενικά καλή" συμπεριφορά ως προς διαφορετικά κριτήρια. Βελτιστοποιώντας κανείς ένα μόνο κριτήριο λαμβάνει ένα πρόγραμμα για τον οποίο δεν υπάρχει καμία ένδειξη ότι θα έχει όχι μόνο "καλή" αλλά ούτε καν "ανεκτή" απόδοση σε άλλα κριτήρια. Η παρατήρηση αυτή οδήγησε πολλούς ερευνητές της περιοχής να ασχοληθούν με πολυκριτηριακά και κυρίως δικριτηριακά προβλήματα Χρονικού Προγραμματισμού.

Η πρώτη από τις τέσσερεις προσεγγίσεις για την επίλυση πολυκριτηριακών προβλημάτων που αναφέραμε παραπάνω (διαλογικοί αλγόριθμοι που διερευνούν τις προτιμήσεις του χρήστη) αφενός δεν έχει σχεδόν καθόλου χρησιμοποιηθεί σε προβλήματα Χρονικού Προγραμματισμού ([HRRW-80]), κι αφετέρου χρησιμοποιεί κυρίως διαφορετικές μεθόδους (μέχρι και έμπειρα συστήματα) ([Evan-84]) από αυτές που χρησιμοποιούν οι άλλες 3 προσεγγίσεις και από αυτές που θα προβληματίσουν εμάς στην εργασία αυτή και δεν θα αναφερθούμε περισσότερο σ’ αυτήν.

Τα δικριτηριακά προβλήματα που αντιστοιχούν στην δεύτερη από τις τέσσερεις παραπάνω προσεγγίσεις (ιεράρχιση κριτηρίων) συμβολίζεται γενικά με α/β/C2|C1, όπου C1 είναι το πρωτεύον κριτήριο και C2 το δευτερεύον. Αυτά λύνονται ελαχιστοποιώντας αρχικά το C1 (αγνοώντας το C2) και μετά θέτοντας σαν περιορισμό να έχει αυτό την τιμή που βρέθηκε ελαχιστοποιώντας το C2. Επομένως η λύση κάθε τέτοιου προβλήματος ανάγεται στη λύση δύο μονοκριτηριακών. Μερικές μάλιστα φορές η βέλτιστη τιμή του C1 είναι γνωστή οπότε το πρόβλημα μπορεί να θεωρηθεί μονοκριτηριακό. Στην κατηγορία αυτή το πρόβλημα που αρχικά μελετήθηκε ήταν το $1//\sum C_i|T_{\max}$ ([Smit-56], [HeRo-72], [Emmo-75b] που γενικεύει το πρώτο κριτήριο σε κάθε αύξουσα με το χρόνο συνάρτηση) που λύνεται πολυωνυμικά. Αργότερα οι ερευνητές στράφηκαν στην NP-hard [LRB-77] γενίκευση: $1//\sum w_i C_i|T_{\max}$ για την οποία έχουν δημοσιευθεί πολλές εργασίες ([Burn-76], [Bans-80], [Miya-81], [ShBu-82], [PoWa-83], [Posn-85], [BaAh-87], [Posn-88]) όπου όλοι σχεδόν αναπτύσσουν παρόμοιους αλγόριθμους διαμερισμού και φραγής με μόνη και σημαντική διαφορά το κάτω φράγμα που χρησιμποιείται, ενώ κάποιοι προτείνουν και ευρηματικούς αλγορίθμους. Δύο ακόμη προβλήματα που ανήκουν στην κατηγορία αυτή και έχουν επίσης λυθεί με αλγόριθμους διαμερισμού και φραγής είναι τα $1//\sum C_i \mid n_T$ [Emmo-75a] και $1//T_{\max} \mid n_T$ [Shan-83]. Οι Chen και Bulfin [ChBu-90] λύνουν πολυωνυμικά το πρόβλημα μιας μηχανής με ίσους χρόνους εκτέλεσης των εργασιών με δύο ιεραρχημένα κριτήρια όταν αυτά είναι οποιαδήποτε δύο από τα $\sum w_i C_i$, $\sum w_i T_i$, wn_T και $T_{\max}$.

Εχουν δημοσιευτεί πολλές εργασίες που αναφέρονται σε δικριτηριακά προβλήματα της τρίτης κατηγορίας (διατύπωση συνάρτησης χρησιμότητας). Σχεδόν σε όλες αυτές τις εργασίες η συνάρτηση χρησιμότητας είναι ένας γραμμικός συνδυασμός των δύο κριτηρίων, το ένα από τα οποία είναι σχεδόν πάντα το άθροισμα (σταθμισμένο ή όχι) των χρόνων αποπεράτωσης και η επίλυση του προβλήματος γίνεται, σχεδόν πάντα επίσης, με αλγόριθμους διαμερισμού και φραγής ενώ μερικές φορές προτείνονται και ευρηματικοί αλγόριθμοι. Προβλήματα αυτής της κατηγορίας που έχουν μελετηθεί έχουν τις εξής αντικειμενικές συναρτήσεις:

α) $\sum w_i C_i + \sum u_i T_i$ [GeKl-74], [GeKl-75] (w_i, u_i: κόστη ανά μονάδα χρόνου ροής και καθυστέρησης αντίστοιχα της εργασίας i),

β) $\sum w_i C_i + \sum S_{i,i+1}$ [BaVa-81] ($S_{i,i+1}$: κόστος αλλαγής διεργασίας από την εργασία που βρίσκεται στην θέση i στην εργασία που βρίσκεται στην θέση $i+1$)

γ) $\sum w_i C_i + \sum u_i z_i$ [Vick-80] (z_i: συμπίεση του χρόνου επεξεργασίας της εργασίας i)

δ) $p\sum C_i + qT_{\max}$ (p και q είναι συντελεστές βάρους των δύο κριτηρίων οι οποίοι στα προηγούμενα ήταν ίσοι [SeGu-83])

ε) $p\sum C_i + q(L_{\max} - L_{\min})$, που επειδή για κάθε εργασία i $E_i = -L_i$, γράφεται και $p\sum C_i + q(L_{\max} + E_{\max})$ [SRD-88] και για p = q [HoVe-92]

στ) $p\sum C_i + q\sum E_i$ [FrLe-87], όπου ειδικά εδώ ορίζουν το E_i σαν το θετικό τμήμα του E_i όπως το έχουμε ορίσει εμείς και διαμορφώνουν ένα 0-1 γραμμικό πρόβλημα, αντίθετα από τα άλλα προβλήματα που λύνονται με αλγόριθμους διαμερισμού και φραγής

ζ) Έναν γραμμικό συνδυασμό υπερωριακού κόστους και ποινών καθυστέρησης [Mats-88]

η) $L_{\max} + E_{\max}$ (ή $L_{\max} - L_{\min}$) [GuSe-84], [TeVl-88], [HoVe-92] για το οποίο ακόμη και με σταθμισμένα κριτήρια, οι τελευταίοι βρίσκουν πολυωνυμικό αλγόριθμο.

Οι Chen και Bulfin [ChBu-90] λύνουν πολυωνυμικά το πρόβλημα μιας μηχανής με ίσους χρόνους εκτέλεσης των εργασιών για οποιαδήποτε συνάρτηση χρησιμότητας δύο οποιωνδήποτε κριτηρίων από τα $\sum w_i C_i$, $\sum w_i T_i$, wn_T και $T_{\max}$, ανάγοντας το σε πρόβλημα ανάθεσης. Ο Baghi [Bagh-89] ασχολείται με το πρόβλημα στο οποίο τα δύο κριτήρια των οποίων ο γραμμικός συνδυασμός αποτελεί την αντικειμενική συνάρτηση είναι το $\sum C_i$ και ένα μέτρο της διασποράς των χρόνων αποπεράτωσης και το λύνει σε πολυωνυμικό χρόνο. Τέλος, ο Kao [Kao-80] αναπτύσει έναν αλγόριθμο διαμερισμού και φραγής για το γενικό πρόβλημα μιας μηχανής με αντικειμενική συνάρτηση $\sum_{i=1}^{m} u_i(x_i)$ όπου x_1, x_2, ..., x_m οι τιμές των m κριτηρίων και u_1, u_2, ..., u_m m συναρτήσεις χρησιμότητας.

Πολλές φορές η ιεράρχηση των κριτηρίων ή η διατύπωση της κατάλληλης συνάρτησης χρησιμότητας δεν είναι κάτι εύκολο ή και εφικτό. Η γενικότερη λύση ενός πολυκριτηριακού προβλήματος είναι ο υπολογισμός του συνόλου των αποδοτικών λύσεων. Η λύση που παίρνει κανείς όταν ιεραρχήσει τα κριτήρια είναι μία μόνο (ακραία) από τις αποδοτικές λύσεις, όπως μια από αυτές είναι και η λύση που παίρνει διατυπώνοντας μια συνάρτηση χρησιμότητας (αύξουσα ως προς κάθε κριτήριο). Πολυκριτηριακά (σχεδόν πάντα δικριτηριακά) προβλήματα Χρονικού Προγραμματισμού μιας μηχανής που έχουν αντιμετωπιστεί με τον υπολογισμό του συνόλου των αποδοτικών λύσεων (και συμβολίζονται 1/β/C1,C2) περιλαμβάνουν τα κριτήρια: $\sum C_i, T_{\max}$, [VaGe-80] η κάθε αποδοτική λύση του οποίου υπολογίζεται σε πολυωνυμικό χρόνο βασιζόμενη στην επίλυση του $1//\sum C_i | T_{\max}=0$ [Smit-56]. $\sum C_i, n_T$, $n_T, T_{\max}$ και $\sum C_i, n_T, T_{\max}$ [NSD-86] η επίλυση των οποίων βασίζεται σε μεθόδους έμμεσης απαρίθμησης. Στα δικριτηριακά από τα προβλήματα που αναφέραμε οι Liao, Huang, Tseng [LHT-92] εφαρμόζουν μια τεχνική που τους επιτρέπει να εντοπίζουν ποιό απόκομμα μιας αποδοτικής λύσης θα τροποποιηθεί για να προκύψει η επόμενη, επιταχυνοντας έτσι τους υπάρχοντες αλγορίθμους. Επίσης για το γενικό πρόβλημα με κριτήρια $\sum C_i$, $\max g_i(C_i)$ (g_i αύξουσα συνάρτηση κόστους) ο John [John-89] γενικεύοντας την εργασία των Van Wassenhove και Gelders [WaGe-80] παρουσιάζει πολυωνυμικό για κάθε αποδοτική λύση αλγόριθμο. Οι Chen και Bulfin λύνουν όπως και στις προηγούμενες κατηγορίες, το πρόβλημα μιας μηχανής με ίσους χρόνους επεξεργασίας για δύο κριτήρια από τα $\sum C_i$, $\sum w_i T_i$, wn_T, $T_{\max}$ με πολυωνυμικό αλγόριθμο για τα περισσότερα και πολυωνυμικό για κάθε αποδοτική λύση για τα υπόλοιπα. Ο Baghi [Bagh-89] που παίρνει σαν κριτήρια το άθροισμα των χρόνων αποπεράτωσης και τη διασπορά τους υπολογίζει σε πολυωνυμικό χρόνο ορισμένες από τις αποδοτικές λύσεις: αυτές για τις οποίες υπάρχει a $\in$ [0, 1] τέτοιο ώστε να αποτελούν λύση του $a\,C_1 + (1-a)\,C_2$ (όπου C_1 και C_2 τα δύο κριτήρια).

Να σημειώσουμε, τέλος, την αξιόλογη κριτική ανασκόπηση της έρευνας που έχει γίνει σε δικριτηριακά προβλήματα Χρονικού Προγραμματισμού από τους Dileepan και Sen [DiSe-88].

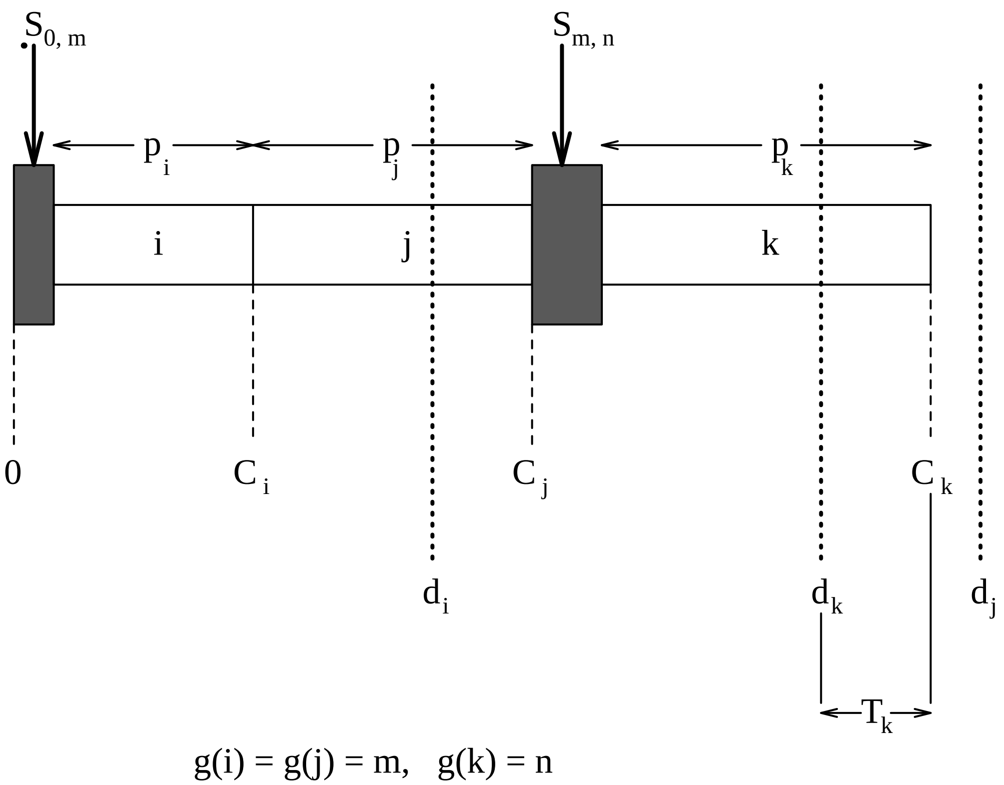
1.6. Ορολογία και συμβολισμοί

Στην παρούσα εργασία χρησιμοποιούνται οι εξής συμβολισμοί, που παριστάνονται γραφικά στο σχήμα 1.1:

Θεωρούμε ότι υπάρχουν **N** (ή **n**) εργασίες που συμβολίζονται με τους ακεραίους από 1 έως και Ν. Ο χρόνος επεξεργασίας, η προθεσμία, το βάρος, η release date (η χρονική στιγμή δηλαδή από την οποία και έπειτα μπορεί να ξεκινήσει η επεξεργασία) και ο χρόνος αποπεράτωσης της εργασίας i, συμβολίζονται με $\mathbf{p_i}$ (ή $\mathbf{P_i}$), $\mathbf{d_i}$, $\mathbf{w_i}$, $\mathbf{r_i}$ και $\mathbf{C_i}$ (ή $\mathbf{c_i}$) αντίστοιχα. Οπως γίνεται συνήθως ορίζουμε ως βραδύτητα, $\mathbf{L_i}$, της εργασίας i τη διαφορά $C_i - d_i$, ως καθυστέρησή της, $\mathbf{T_i}$, το $\max\{0, L_i\}$ και ως "νωρίτητά" της, $\mathbf{E_i}$, το $-L_i$, δηλαδή το $d_i - C_i$. Επίσης, με $\mathbf{T_{max}}$, $\mathbf{L_{max}}$ και $\mathbf{E_{max}}$ ενός προγράμματος συμβολίζουμε τη μέγιστη εμφανιζόμενη καθυστέρηση, βραδύτητα και "νωρίτητα" αντίστοιχα και με $\mathbf{n_T}$ τον αριθμό των καθυστερημένων εργασιών του. Ορίζουμε ακόμα $\overline{P} = \sum_{i=1}^{n} P_i/n$, τον μέσο χρόνο επεξεργασίας των εργασιών.

Οι εργασίες είναι χωρισμένες σε **B** (ή **b**) ομάδες. Η τυχούσα ομάδα k έχει $\mathbf{n_k}$ εργασίες ($\sum_{k=1}^{B} n_k = N$). Ορίζουμε $\mathbf{n_{max}} = \max_{1\le k\le b}\{ n_k \}$. Η εργασία i ανήκει στην ομάδα **g(i)**. Ανάμεσα στις εκτελέσεις δύο εργασιών, i και j με $g(i) = k$ και $g(j) = l$ μεσολαβεί ένας χρόνος εξάρμωσης $\mathbf{s_{kl}}$, που συχνά όμως, όταν δεν υπάρχει πρόβλημα σαφήνειας, θα συμβολίζεται με $\mathbf{s_{ij}}$, δηλαδή θα έχει ως δείκτες τις εργασίες και όχι τις αντίστοιχες ομάδες. Για κάθε ομάδα k ισχύει ότι, $s_{kk} = 0$. Επίσης, όπως γίνεται συνήθως, θα θεωρούμε ότι για κάθε τριάδα ομάδων k, l, m ισχύει $s_{kl} + s_{lm} \le s_{km}$, δηλαδή η τριγωνική ανισότητα. Οταν οι χρόνοι εξάρμωσης είναι ανεξάρτητοι ακολουθίας τότε για δύο ομάδες k και l το s_{kl} θα σημαίνει s_l εάν $k \neq l$ και 0 διαφορετικά. Αντίστοιχη σύμβαση ισχύει και όταν οι δείκτες είναι εργασίες. Για την περίπτωση αυτή (χρόνοι εξάρμωσης ανεξάρτητοι ακολουθίας) ορίζουμε $\overline{\mathbf{P+S}} = \sum_{i=1}^{n}(P_i + S_i)/n$.

Εάν π είναι μια διάταξη των εργασιών τότε με $\pi(i)$ εννοούμε την εργασία που βρίσκεται στην i θέση της διάταξης η οποία τώρα μπορεί να γραφεί ως η διατεταγμένη n-άδα (π(1), π(2),...,π(n)). Με $\pi^{-1}(i)$ εννοούμε τη θέση που βρίσκεται



Σχήμα 1.1: Ορισμός των βασικών μεγεθών του προβλήματος

η εργασία i. Τώρα μπορούμε να δώσουμε και τον μαθηματικό ορισμό του χρόνου αποπεράτωσης της εργασίας i σε μια διάταξη π, που είναι βέβαια διαισθητικά φανερός: $C_i = \sum_{j=1}^{\pi^{-1}(i)} (S_{\pi(j-1)\pi(j)} + P_{\pi(j)})$. Στη θέση 0 θεωρούμε ότι βρίσκεται μια νοητή εργασία, έστω η εργασία 0, που ανήκει στην ομάδα 0, που αντιστοιχεί στην ανενεργή κατάσταση της μηχανής, της οποίας η ύπαρξη αιτιολογεί την εμφάνιση των αρχικών χρόνων εξάρμωσης S_{0k} για κάθε ομάδα k. Ορίζουμε επίσης, $C_{max} = C_{\pi(n)}$. Οταν θέλομε να δηλώσουμε ότι ο χρόνος αποπεράτωσης της εργασίας i αναφέρεται ως προς μια διάταξη π_α, τον συμβολίζομε με C_i^α. Η τιμή του κριτηρίου απόδοσης C (π.χ. $\sum C_i$, T_{max} ή C_{max}) όταν οι εργασίες διαταχθούν σε μια διάταξη π, συμβολίζεται με $C(\pi)$ (π.χ. $\sum C_i(\pi)$, $T_{max}(\pi)$ ή $C_{max}(\pi)$).

Μερικές φορές είναι βολικότερο αντί να συμβολίζονται οι εργασίες με τους ακεραίους από 1 έως N να συμβολίζονται με κάποιον άλλο τρόπο που να δηλώνει άμεσα σε ποιά ομάδα ανήκει η κάθε εργασία. Αυτό γίνεται συμβολίζοντας κάθε εργασία με διατεταγμένο ζεύγος ακεραίων (k,j) ο πρώτος από τους οποίους δηλώνει σε ποιά ομάδα ανήκει η εργασία και ο δεύτερος ποιά εργασία αυτής της ομάδας είναι η συγκεκριμένη $(1 \leq j \leq n_k)$. Χρησιμοποιώντας αυτόν τον συμβολισμό των εργασιών τα διάφορα χαρακτηριστικά τους, (όπως χρόνοι επεξεργασίας και προθεσμίες ή χρόνοι αποπεράτωσης και καθυστερήσεις) χρησιμοποιούν δύο ακέραιους δείκτες (π.χ. P_{ki}, d_{ki}, C_{ki} και T_{ki}) αντί του ενός που χρησιμοποιούν με τον προηγούμενο συμβολισμό.

Οι όροι πρόγραμμα και διάταξη θα χρησιμοποιούνται με την ίδια έννοια, εφόσον η στιγμή έναρξης ενός προγράμματος θα είναι πάντα η στιγμή 0. Σαν υποπρόγραμμα, ή μερική διάταξη, θα χαρακτηρίζουμε ένα τμήμα (απόκομμα) μιας διάταξης.

Τέλος, αναφερόμενοι σε δικριτηριακά προβλήματα, με τους όρους **αποδοτική λύση**, **αποδοτικό πρόγραμμα** και **αποδοτικό σημείο** εννοούμε κάθε λύση, πρόγραμμα ή σημείο τέτοιο ώστε να μην υπάρχει άλλο που να έχει καλύτερη απόδοση στο ένα κριτήριο και να μην έχει χειρότερη στο άλλο. Αναζητώντας τα αποδοτικά προγράματα ενός δικριτηριακού προβλήματος ενδιαφερόμαστε για ένα μόνο από εκείνα που έχουν την ίδια τιμή και στα δύο κριτήρια, όπως συνηθίζεται σε τέτοιες περιπτώσεις (π.χ. [WaGe-80]).

2. Ανάλυση του $1/s_{ij}/\sum C_i, T_{max}$

2.1. Αναγωγή στο $1/s_{ij}/\sum C_i | T_{max}=0$

Οπως είπαμε και στην εισαγωγή, οι Van Wassenhove και Gelders [WaGe-80] πρότειναν έναν αλγόριθμο για το $1//\sum C_i$, T_{max} που υπολογίζει μία-μία τις αποδοτικές λύσεις αρχίζοντας από αυτές με το μικρότερο $\sum C_i$ (και μεγαλύτερο T_{max}) και καταλήγοντας σ'αυτήν με το μικρότερο T_{max} (και το μεγαλύτερο $\sum C_i$). Η τεχνική αυτή βασίζεται σε μία πρόταση που απέδειξαν ειδικά για το πρόβλημα που τους απασχολούσε. Μια γενικότερη μορφή της που καλύπτει και το πρόβλημα που απασχολεί εμάς ($1/s_{ij}/\sum C_i$, T_{max}) είναι η εξής:

Πρόταση 2.1: Έστω δικριτηριακό συνδυαστικό πρόβλημα με κριτήρια A, B, F το σύνολο των εφικτών λύσεων, A(x), B(x) οι τιμές των κριτηρίων A και B αντίστοιχα για την $x \in F$ και L πραγματικός αριθμός. Θέτοντας $S_1 = \{x \in F: A(x) \le L\}$ (έστω μή κενό), $S_2 = \{x \in S_1 : B(x) \le B(y)$ για κάθε $y \in S_1\}$ και $x_0 \in S_2: A(x_0) \le A(x)$ για κάθε $x \in S_2$, τότε η x_0 είναι αποδοτική λύση.

Απόδειξη: Για κάθε $x_1 \in F$ έχουμε: εάν $A(x_1) < A(x_0)$ τότε $x_1 \in S_1$ (γιατί $A(x_1) < A(x_0) \le L$) και $x_1 \notin S_2$ (αφού $A(x_0) \le A(x)$ για κάθε $x \in S_2$) οπότε $B(x_1) > B(x_0)$ (γιατί διαφορετικά θα ίσχυε $x_1 \in S_2$ αφού $x_0 \in S_2$). Εάν $B(x_1) < B(x_0)$ τότε $x_1 \notin S_1$ (γιατί αλλιώς $x_0 \notin S_2$) οπότε $A(x_1) > L \ge A(x_0)$. Επομένως η x_0 είναι αποδοτική λύση.

Ειδικές περιπτώσεις της παραπάνω πρότασης αποτελούν οι εφαρμογές της στα προβλήματα Χρονικού Προγραμματισμού. $1//\sum C_i$, T_{max} και $1/s_{ij}/\sum C_i$, T_{max}.

Ο αλγόριθμος των Van Wassenhove - Gelders για τον υπολογισμό των αποδοτικών λύσεων του $1//\sum C_i$, T_{max} ελαφρά παραλλαγμένος για το $1/s_{ij}/\sum C_i$, T_{max} είναι ο εξής:

Βήμα 1: Θέσε $\Delta = \sum_{i=1}^{n}(P_i + \max_{0 \le j \le B} s_{j,g(i)})$ και πήγαινε στο Βήμα 2.

Βήμα 2: Βρες το x_0 της πρότασης 2.1 (το οποίο ας το λέμε τώρα π^*) όταν το συνδυαστικό πρόβλημα είναι το $1/s_{ij}/\sum C_i$, T_{max}, $A=T_{max}$, $B=\sum C_i$ και $L = \Delta$. Εάν υπάρχει τέτοιο πρόγραμμα π^* (που είναι ισοδύναμο με το να υπάρχει πρόγραμμα π με $T_{max}(\pi) \le \Delta$), τότε σημείωσε αυτό ως αποδοτικό και πήγαινε στο Βήμα 3. Αλλιώς σταμάτα.

Βήμα 3: Υπολόγισε το $T_{max}(\pi^*)$. Εάν $T_{max}(\pi^*)=0$ σταμάτα. Αλλιώς θέσε $\Delta = T_{max}(\pi^*)-1$ (όταν τα δεδομένα είναι ακέραιοι - γενικότερα αφαίρεσε το μεγαλύτερο αριθμό που διαιρεί ακριβώς όλα τα δεδομένα) και πήγαινε στο Βήμα 2.

Πρόταση 2.2: Ο παραπάνω αλγόριθμος δημιουργεί όλα τα αποδοτικά σημεία και μόνο αυτά.

Απόδειξη: Σύμφωνα με την Πρόταση 2.1, κάθε πρόγραμμα που υπολογίζεται στο Βήμα 2 είναι αποδοτικό επομένως ο αλγόριθμος δημιουργεί μόνο αποδοτικά σημεία. Εστω ότι σε κάποια επανάληψη του αλγορίθμου υπολογίζεται το αποδοτικό σημείο π_1 και στην επόμενη το π_2. Επειδή $T_{max}(\pi_2) \le T_{max}(\pi_1)-1$, κι αφού είναι και τα δύο αποδοτικά σημεία, ισχύει $T_{max}(\pi_2) < T_{max}(\pi_1)$ και $\sum C_i(\pi_1) < \sum C_i(\pi_2)$. Εάν υπάρχει αποδοτική λύση π' με $T_{max}(\pi_2) < T_{max}(\pi') < T_{max}(\pi_1)$ θα ισχύει $\sum C_i(\pi_1) < \sum C_i(\pi') < \sum C_i(\pi_2)$ ώστε να είναι και τα τρία προγράμματα αποδοτικά. Επειδή όμως $T_{max}(\pi')<T_{max}(\pi_1)$ το π' ανήκει στο σύνολο S_1 όπως αυτό ορίστηκε στην Πρόταση 2.1 οπότε το π_2 δεν μπορεί να ανήκει στο S_2, αφού $\sum C_i(\pi_2)>\sum C_i(\pi')$ πράγμα άτοπο γιατί εξ ορισμού ανήκει. Επομένως ο αλγόριθμος μετά από κάποιο αποδοτικό σημείο υπολογίζει εκείνο το αποδοτικό σημείο (εάν υπάρχει) με το αμέσως μικρότερο T_{max} (και το μεγαλύτερο $\sum C_i$), άρα υπολογίζεται κάθε αποδοτικό σημείο.

Να παρατηρήσουμε ότι :

1) Το Βήμα 2 του αλγορίθμου απαιτεί ουσιαστικά την επίλυση του $1/S_{ij}/\sum C_i|T_{max}\leq\Delta$, και την επιλογή από τις βέλτιστες λύσεις αυτού εκείνης με το μικρότερο T_{max}. Αυξάνοντας τις προθεσμίες των εργασιών κατά Δ το παραπάνω πρόβλημα γίνεται $1/S_{ij}/\sum C_i|T_{max}=0$ (γιατί η καθυστέρηση καθε εργασίας i από T_i γίνεται $\max\{0,T_i-\Delta\}$ οπότε το T_{max} γίνεται $\max\{0,T_{max}-\Delta\}$, που είναι 0 όταν $T_{max}\leq\Delta$) και αρκεί μετά η επιλογή της βέλτιστης λύσης με το μικρότερο L_{max}. Επομένως το $1/S_{ij}/\sum C_i|T_{max}$ λύνεται με επαναληπτική λύση του $1/S_{ij}/\sum C_i|T_{max}=0$ και αυτο θα μας απασχολήσει από εδώ και πέρα.

2) Στην πρώτη επανάληψη του αλγορίθμου το Δ είναι αρκετά μεγάλο ώστε κάθε πρόγραμμα να έχει $T_{max}\leq\Delta$ (να ανήκει δηλαδή στο S_1) οπότε το πρώτο αποδοτικό σημείο είναι η λύση του $1/S_{ij}/\sum C_i$ με την μικρότερη μέγιστη καθυστέρηση.

3) Εάν υπάρχει πρόγραμμα με $T_{max}=0$ ο αλγόριθμος σταματάει στο Βήμα 3 διαφορετικά στο Βήμα 2. Οπου και να σταματήσει, το τελευταίο αποδοτικό σημείο που υπολογίζεται είναι η λύση του $1/S_{ij}/T_{max}$ με το μικρότερο αθροισμα του χρόνου αποπεράτωσης.

2.2. Προβλήματα διατεταγμένων ομάδων και υπολογιστική πολυπλοκότητα

Οπως είπαμε και στην εισαγωγή, οι Monma και Potts [MoPo-89] όρισαν σαν πρόβλημα διατεταγμένων ομάδων κάθε πρόβλημα Χρονικού Προγραμματισμού με χρόνους εξάρμωσης στο οποίο η βέλτιστη διάταξη σε κάθε ομάδα είναι από την αρχή γνωστή (δηλαδή μπορεί να βρεθεί σε πολυωνυμικό χρόνο). Οι ίδιοι παρουσίασαν έναν αλγόριθμο επίλυσης κάθε προβληματος διατεταγμένων ομάδων με πλοκή εκθετική ως προς τον αριθμό των ομάδων B και πολυωνυμική ως προς τον αριθμό των εργασιών N. Αυτό είναι πολύ σημαντικό γιατί το B είναι γενικά πολύ μικρότερο από το N οπότε το εκθετικό κομμάτι αυξάνει με πολύ αργότερο ρυθμό από ότι θα αύξανε αν η πολυπλοκότητα ήταν εκθετική ως προς τον αριθμό των εργασιών N. Απέδειξαν επίσης ότι προβλήματα διατεταγμένων ομάδων είναι το $1/S_{ij}/\sum w_iC_i$ και το $1/S_{ij}/T_{max}$. Η υπολογιστική πλοκή του πρώτου (και για ίσα βάρη) είναι ακόμα ένα ανοικτό ερώτημα, ενώ για το δεύτερο έχει αποδειχτεί [BrDo-78] ότι είναι NP-hard. Αρα και το $1/S_{ij}/\sum C_i|T_{max}=0$ (που όπως είπαμε είναι το πρόβλημα που θα μας απασχολήσει από δω και πέρα), είναι επίσης NP-hard αφού το $1/S_{ij}/T_{max}$ δεν αποτελεί παρά το πρόβλημα εφικτότητας του.

Εύλογα νομίζουμε τίθεται το ερώτημα μήπως είναι και το $1/S_{ij}/\sum C_i|T_{max}=0$ πρόβλημα διατεταγμένων ομάδων. Τρείς είναι οι κανόνες βάσει των οποίων μοιάζει λογικό, σε πρώτη τουλάχιστον ματιά, να είναι διατεταγμένες οι εργασίες σε κάθε ομάδα: SPT, EDD και η διάταξη που προκύπτει από τη βέλτιστη λύση του $1//\sum C_i|T_{max}=0$ σε κάθε ομάδα. Είναι πολύ απλό να δημιουργήσει κανείς αντιπαραδείγματα που να φανερώνουν ότι οι δυο πρώτοι κανόνες δεν δίνουν βέλτιστη λύση, ούτε καν στην περίπτωση της μιάς μόνο ομάδας (στην περίπτωση δηλαδή χωρίς χρόνους εξάρμωσης). Δυστυχώς, ούτε ο τρίτος κανόνας μπορεί να χρησιμοποιηθεί για μια εκ των προτέρων διάταξη των εργασιών σε κάθε ομάδα, όπως δείχνει το παρακάτω αντιπαράδειγμα: Εστω δύο ομάδες απο τις οποίες η μία περιέχει μια εργασία με $p_{1,1}=1$ και $d_{1,1}=5$ και η άλλη δύο με $p_{2,1}=1$, $d_{2,1}=7$, $p_{2,2}=2$, $d_{2,2}=5$ και ακόμα σταθερός χρόνος εξάρμωσης $S=1$. Η βέλτιστη διάταξη είναι (1,1)-(2,2)-(2,1) παρόλο που η βέλτιστη διάταξη του $1//\sum C_i|T_{max}=0$ στη δεύτερη ομάδα είναι (2,1)-(2,2). Αυτό που συμβαίνει είναι ότι όταν λύνεται το $1//\sum C_i|T_{max}=0$ σε κάθε ομάδα ξεχωριστά, οι χρόνοι αποπεράτωσης είναι μικρότεροι από ότι είναι όταν λύνεται το $1/S_{ij}/\sum C_i|T_{max}=0$ αφού το πρόγραμμα έχει λιγότερες εργασίες (και κανένα χρόνο εξάρμωσης) οπότε απορρίπτονται λιγότερες διατάξεις λόγω παραβίασης των προθεσμιών (μπορεί και καμιά) με αποτέλεσμα να μην υπάρχει, γενικά, κάποια εφικτή συγχώνευση των επιμέρους βέλτιστων διατάξεων οι οποίες σχηματίζονται επηρεαζόμενες ελάχιστα (ή καθόλου) από τις προθεσμίες. Για παράδειγμα στο παραπάνω πρόβλημα μόνο 2 από τις 6 διατάξεις ήταν εφικτές. Βλέπουμε λοιπόν ότι δεν μοιάζει, αν και δεν το έχουμε αποδείξει μαθηματικά, να υπάρχει κανόνας διάταξης των εργασιών σε κάθε ομάδα που μπορεί να εφαρμοστεί εξ αρχής και επομένως το πρόβλημα μοιάζει απίθανο να είναι πρόβλημα διατεταγμένων ομάδων και άρα μοιάζει απίθανο να μπορεί να λυθεί σε χρόνο εκθετικό μόνο ως προς το πλήθος των ομάδων και όχι ως προς το πλήθος των εργασιών.

2.3. Κανόνες προτεραιότητας

Λύνοντας κανείς ένα πρόβλημα Χρονικού Προγραμματισμού με μια μέθοδο έμμεσης απαρίθμησης, προσπαθεί με κάθε τρόπο να ελαττώσει την περιοχή έρευνας (δηλαδή το σύνολο υποψήφιων βέλτιστων διατάξεων) για να είναι ταχύτερη η εκτέλεση του αλγορίθμου του. Ενας τρόπος ελάττωσης του χώρου έρευνας είναι η απόδειξη σχέσεων της μορφής "υπάρχει βέλτιστη λύση όπου η εργασία i εκτελείται πρίν την εργασία j" ή της μορφής "σε κάθε εφικτή λύση η εργασία i προηγείται της εργασίας j". Αυτές οι σχέσεις ονομάζονται σχέσεις (ή κανόνες) προτεραιότητας και μας επιτρέπουν, όταν ενδιαφερόμαστε για μια μόνο βέλτιστη λύση να την αναζητήσουμε στα προγράμματα που τις ικανοποιούν. Επειδή εμείς ζητάμε τη βέλτιστη λύση του $1/S_{ij}/\sum C_i|T_{max}=0$ με το μικρότερο L_{max}, πρέπει οι σχέσεις προτεραιότητας της πρώτης μορφής που θα χρησιμοποιήσουμε να εγγυώνται ότι δεν υπάρχει βέλτιστη λύση στην οποία η j προηγείται της i με μικρότερο L_{max} από το L_{max} της βέλτιστης λύσης που η i προηγείται της j. Για το πρόβλημά μας μπορούν να διατυπωθούν δύο προτάσεις που οδηγούν στη δημιουργία σχέσεων προτεραιότητας. Και οι δύο είναι παραλλαγές προτάσεων για το $1//\sum w_iC_i|T_{max}=0$ που δίνονται στην [PoWa-83] (μια γενική μορφή της πρώτης διατυπώνεται στην [RLL-75]).

Πρόταση 2.3: Εάν για δύο εργασίες i και j της ίδιας ομάδας ισχύει $p_i \le p_j$ και $d_i \le d_j$, υπάρχει βέλτιστη λύση του $1/S_{ij}/\sum C_i | T_{max}=0$, με L_{max} όχι μεγαλύτερο από τις υπόλοιπες βέλτιστες λύσεις του, στην οποία η εργασία i προηγείται της εργασίας j.

Απόδειξη: Εστω πρόγραμμα π_1 στο οποίο η εργασία j προηγείται της εργασίας i κι έστω πρόγραμμα π_2 που προκύπτει αντιμεταθέτοντας τις i και j. Επειδή αυτές ανήκουν στην ίδια ομάδα δεν προστίθεται ούτε αφαιρείται κάποιος χρόνος εξάρμωσης από την αντιμετάθεσή τους. Για το λόγο αυτό και επειδή $p_i \le p_j$ έχουμε ότι $C_i^2 \le C_j^1$, $C_j^2 = C_i^1$, $C_k^2 \le C_k^1$ για κάθε εργασία που έχει διαταχθεί μεταξύ των i και j και $C_k^2 = C_k^1$ για κάθε άλλη εργασία. Επομένως $\sum C_i(\pi_2) \le \sum C_i(\pi_1)$. Επειδή $L_k^2 \le L_k^1$ αφού $C_k^2 \le C_k^1$ για κάθε $k \ne i,j$ και η L_i^1 που είναι μεγαλύτερη από την L_j^1 (αφού $C_j^1 < C_i^1$ και $d_i \le d_j$) δεν είναι μικρότερη από τις L_i^2 (αφού $C_i^2 \le C_i^1$) και L_j^2 (αφού $C_i^1 = C_j^2$ και $d_i \le d_j$), έχουμε ότι $L_{max}(\pi_2) \le L_{max}(\pi_1)$. Επομένως για κάθε εφικτό πρόγραμμα στο οποίο η j βρίσκεται πριν την i υπάρχει ένα άλλο εφικτό πρόγραμμα που έχει την i πριν την j με όχι μεγαλύτερο $\sum C_i$ και L_{max}, οπότε υπάρχει βέλτιστη λύση του $1/S_{ij}/\sum C_i | T_{max}=0$ με το μικρότερο μάλιστα L_{max} στο οποίο η i βρίσκεται πριν την j.

Πρόταση 2.4: Εστω ότι οι εργασίες έχουν αριθμηθεί κατά EDD. Εάν υπάρχουν ακέραιοι i, j με $1 \le i \le j \le n$ για τους οποίους ισχύει $\sum_{k=1}^{i} P_k + P_j + \sum_{b \in S} \min_a \{S_{ab}\} > d_i$, όπου S το σύνολο των ομάδων που ανήκουν οι εργασίες 1 έως i και j, τότε σε κάθε εφικτή λύση οι εργασίες 1 έως i βρίσκονται πριν την j.

Απόδειξη: Εάν η εργασία j βρισκόταν πριν από κάποια ή κάποιες από τις εργασίες 1 έως i τότε η τελευταία από αυτές θα τελείωνε σε μια χρονική στιγμή τουλάχιστον ίση με το αριστερό μέλος της ανισότητας της πρότασης (αφού ο τρίτος όρος του αθροίσματος είναι ένα κάτω φράγμα του αθροίσματος των χρόνων εξάρμωσης που θα έχουν παρεμβληθεί μέχρι τότε) κι επομένως μετά την προθεσμία της (αφού $d_k \le d_i$ για κάθε $k \le i$) άρα το πρόγραμμα αυτό δεν είναι εφικτό.

2.4. Σχέσεις κυριαρχίας μεταξύ μερικών διατάξεων

Ενας άλλος τρόπος ελάττωσης του χώρου έρευνας είναι αποδεικνύοντας ότι για κάθε πρόγραμμα που προκύπτει συμπληρώνοντας με κάποιο τρόπο μια μερική διάταξη των εργασιών, π, υπάρχει ένα άλλο πρόγραμμα με μικρότερη (ή ίση) τιμή στην αντικειμενική συνάρτηση, οπότε δεν υπάρχει λόγος να εξεταστούν τα προγράμματα που προκύπτουν συμπληρώνοντας την π. Όταν μάλιστα τα προγράμματα αυτά με τις μικρότερες (ή ίσες) τιμές στην αντικειμενική συνάρτηση, από τα προγράμματα που προκύπτουν συμπληρώνοντας την π, προκύπτουν συμπληρώνοντας κατάλληλα μια άλλη μερική διάταξη π_0 λέμε ότι η π_0 κυριαρχεί της π. Μια πρόταση που δημιουργεί τέτοιες σχέσεις κυριαρχίας και πηγάζει από τις ιδέες των εργασιών [Psar-80] και [AhHy-90], παρόλο που δεν διατυπώνεται ούτε χρησιμοποιείται εκεί, είναι η εξής:

Πρόταση 2.5: Εστω μερικές διατάξεις π_1 και π_2 των ίδιων n-k εργασιών. Εστω b_1 και b_2 οι ομάδες όπου ανήκουν οι τελευταίες εργασίες των π_1 και π_2 αντίστοιχα. Εάν ισχύει

$$\sum C_i(\pi_1) + k(C_{max}(\pi_1) + S_{b_1b_2}) \le \sum C_i(\pi_2) + k\cdot C_{max}(\pi_2) \qquad (1)$$

τότε για το πρόβλημα $1/S_{ij}/\sum C_i$ η π_1 κυριαρχεί της π_2, ενώ αν επιπλέον ισχύει και

$$C_{max}(\pi_1) + S_{b_1b_2} \le C_{max}(\pi_2) \qquad (2)$$

τότε η σχέση κυριαρχίας ισχύει και για το $1//S_{ij}/\sum C_i|T_{max}=0$.

Απόδειξη: Εστω οποιαδήποτε διάταξη π_0 των υπόλοιπων k εργασιών. Εστω ότι το άθροισμα των χρόνων αποπεράτωσης των εργασιών της π_0, όταν η πρώτη της εργασία αρχίζει να εκτελείται την στιγμή μηδέν (χωρίς αρχικό χρόνο εξάρμωσης) είναι A. Εάν η π_0 προστεθεί στις π_1 και π_2 η πρώτη της εργασία, που έστω ότι ανήκει στην ομάδα b_0, θα ξεκινήσει τις χρονικές στιγμές $C_{max}(\pi_1) + S_{b_1b_0}$ και $C_{max}(\pi_2) + S_{b_2b_0}$ αντίστοιχα. Αυτό έχει σαν αποτέλεσμα: $\sum C_i(\pi_1\pi_0) = \sum C_i(\pi_1) + A + k(C_{max}(\pi_1) + S_{b_1b_0})$ και $\sum C_i(\pi_2\pi_0) = \sum C_i(\pi_2) + A + k(C_{max}(\pi_2) + S_{b_2b_0})$. Οι τρίτοι όροι των αθροισμάτων εμφανίζονται επειδή η π_0 που έχει k εργασίες δεν ξεκινάει τη στιγμή μηδέν αλλά τις χρονικές στιγμές που προαναφέραμε. Επειδή οι χρόνοι εξάρμωσης ικανοποιούν την τριγωνική ανισότητα, έχουμε $S_{b_1b_0} \le S_{b_1b_2} + S_{b_2b_0} \Leftrightarrow A + k\cdot S_{b_1b_0} \le A + k(S_{b_1b_2} + S_{b_2b_0})$. Προσθέτωντας την προηγούμενη σχέση στη σχέση (1) της πρότασης παίρνουμε $\sum C_i(\pi_1) + A + k(C_{max}(\pi_1) + S_{b_1b_0}) \le \sum C_i(\pi_2) + A + k(C_{max}(\pi_2) + S_{b_2b_0}) \Leftrightarrow \sum C_i(\pi_1\pi_0) \le \sum C_i(\pi_2\pi_0)$. Επομένως, όταν δεν υπάρχουν προθεσμίες η π_1 κυριαρχεί της π_2.

Οταν υπάρχουν προθεσμίες όλα τα παραπάνω ισχύουν με την προυπόθεση ότι για κάθε διάταξη π_0 των αδιάτακτων εργασιών για την οποία είναι εφικτό το πρόγραμμα $\pi_2\pi_0$, είναι εφικτό και το $\pi_1\pi_0$. Μια ικανή συνθήκη για να ισχύει αυτό, είναι να μην αρχιζει αργότερα η π_0 στο πρόγραμμα $\pi_1\pi_0$ απ'ότι στο $\pi_2\pi_0$, δηλαδή $C_{max}(\pi_1) + S_{b_1b_0} \le C_{max}(\pi_2) + S_{b_2b_0}$. Προσθέτωντας τη συνθήκη της τριγωνικής ανισότητας, $S_{b_1b_0} \le S_{b_1b_2} + S_{b_2b_0}$, στη σχέση (2) της πρότασης παίρνουμε ακριβώς την παραπάνω συνθήκη.

Να παρατηρήσουμε ότι στην διατύπωση και στην απόδειξη της παραπάνω πρότασης δεν έγινε καμία υπόθεση για τα b_1, b_2, b_0, οπότε δεν αποκλείεται δύο από αυτά ή και τα τρία να είναι ίσα. Οταν $b_1=b_2$ οι σχέσεις (1) και (2) γίνονται $\sum C_i(\pi_1) + k\cdot C_{max}(\pi_1) \le \sum C_i(\pi_2) + k\cdot C_{max}(\pi_2)$ και $C_{max}(\pi_1) \le C_{max}(\pi_2)$ αντίστοιχα.

3. Μέθοδοι επίλυσης προβλημάτων χρονικού προγραμματισμού και το $1/S_{ij}/\sum C_i\,|\,T_{max}=0$

Στο κεφάλαιο αυτό θα κάνουμε μια ανασκόπηση μεθόδων που ακολουθήθηκαν από διάφορους ερευνητές για την επίλυση φαινομενικά παραπλήσιων με το δικό μας πρόβλημα Χρονικού Προγραμματισμού ενώ παράλληλα θα εντοπίζουμε τους λόγους που καθιστούν την κάθε μέθοδο αναποτελεσματική για το δικό μας πρόβλημα. Εκτός από μία περίπτωση που θα ασχοληθούμε με το $1/S_{ij}/\sum C_{i,}T_{max}$, στο υπόλοιπο κεφάλαιο (όπως και στην υπόλοιπη εργασία) θα ασχοληθούμε με το $1/S_{ij}/\sum C_i|T_{max}=0$ ή μερικές φορές για απλούστευση με το $1/S_j/\sum C_i|T_{max}=0$. Οι περισσότερες από τις μεθόδους που θα εξετάσουμε έχουν εφαρμοστεί σε κάποιο (ή κάποια) από τα $1//\sum w_iC_i|T_{max}=0$, $1/r_i/\sum w_iC_i$ ή $1//\sum w_iT_i$. Στόχος των μισών τουλάχιστον εργασιών που θα αναφερθούν είναι ο υπολογισμός ενός καλού κάτω φράγματος για μία διαδικασία διαμερισμού και φραγής. Στις υπόλοιπες γίνεται συνήθως προσπάθεια να λυθεί το αντίστοιχο πρόβλημα με κάποιον άλλο τρόπο (ή και με αλγόριθμο διαμερισμού και φραγής να λύνεται δεν βρίσκεται η ουσία της εργασίας στον υπολογισμό κάτω φράγματος) ενώ λίγες εργασίες παρουσιάζουν μεθόδους εύρεσης προσεγγιστικών λύσεων. Εμείς, παρουσιάζουμε πρώτα μεθόδους εύρεσης βέλτιστης λύσης που δεν ασχολούνται με υπολογισμό κάτω φράγματος για αλγορίθμους διαμερισμού και φραγής. Τέτοιες μεθόδους παρουσιάζουμε αμέσως μετά. Στο τέλος παρουσιάζουμε κάποιες μεθόδους εύρεσης προσεγγιστικής λύσης.

3.1. Μέθοδοι ακριβούς επίλυσης

Το 1956 ο Smith [Smit-56] παρουσίασε τον εξής πολυωνυμικό αλγόριθμο για την επίλυση του $1//\sum C_i|T_{max}=0$ (δηλαδή του προβλήματος που προκύπτει εάν δεν υπάρχουν χρόνοι εξάρμωσης σ'αυτό που απασχολεί εμάς) : στη βέλτιστη διάταξη k (=n, ..., 2) εργασιών (που έστω ότι είναι οι 1,2, ..., κ) τελευταία βρίσκεται η εργασία i τέτοια ώστε $P_i = \max\{P_j | \sum_{l=1}^{k} P_l \le d_j\}$. Σε περίπτωση που με τον παραπάνω κανόνα δεν καθορίζεται μοναδικά η εργασία i, μπορεί να μπει τελευταία μια οποιαδήποτε από αυτές με το μεγαλύτερο χρόνο επεξεργασίας, εκτός αν το πρόβλημα $1//\sum C_i|T_{max}=0$ εμφανίζεται στον υπολογισμό μιας αποδοτικής λύσης του $1//\sum C_i$, T_{max} (όπως είπαμε στο προηγούμενο κεφάλαιο οτι προτείνουν οι Van Wassenhove - Gelders [WaGe-80]), οπότε επειδή από όλες τις βέλτιστες λύσεις ψάχνουμε για εκείνη με το μικρότερο L_{max}, θα μπει τελευταία η εργασία με την μεγαλύτερη προθεσμία από αυτές με το μεγαλύτερο χρόνο επεξεργασίας. Ο παραπάνω κανόνας ισχύει γιατί αν σε μία διάταξη η εργασία i δεν βρίσκεται τελευταία, τότε ανταλλάσοντας την i με αυτήν που είναι τελευταία, οι μετά την i εργασίες (καθώς και αυτή που μπήκε στη θέση της) τελειώνουν νωρίτερα οπότε η νέα διάταξη είναι εφικτή και με μικρότερη τιμή του $\sum C_i$.

Αυτό δεν βρίσκει εφαρμογή στην περίπτωση που υπάρχουν χρόνοι εξάρμωσης γιατί πρώτον ο χρόνος αποπεράτωσης των n εργασιων είναι άγνωστος (ενώ πριν ήταν γνωστός, ίσος με $\sum_{k=1}^{n} P_k$) και δεύτερον ακόμα και αν ο χρόνος αυτός ήταν γνωστός, αυτό δεν θα σήμαινε ότι η εργασία i που θα ορίζοταν όπως προηγουμένως θα έπρεπε να μπει τελευταία, γιατί αν σε κάποια διάταξη δεν ήταν τελευταία και την ανταλλάσαμε με την τελευταία θα ήταν ενδεχόμενη η εισαγωγή κάποιων χρόνων εξάρμωσης, οπότε δεν θα ίσχυε πια ότι οι μετά την i εργασίες θα τελείωναν νωρίτερα ούτε κατ'επέκταση ότι η νέα προκύπτουσα διάταξη θα ήταν εφικτή και με μικρότερη τιμή του $\sum C_i$ από την αρχική.

Για το πρόβλημα $1/S_{ij}/\sum C_i$ (δηλαδή σαν το πρόβλημα που απασχολεί εμάς χωρίς όμως προθεσμίες) έχουν αναπτυχθεί αλγόριθμοι Δυναμικού Προγραμματισμού από τον Ψαραύτη [Psar-80] για την ειδική περίπτωση των ίσων χρόνων επεξεργασίας και από τους Monma και Potts [MoPo-89] και Ahn και Hyhn [AhHy-90] για την γενική περίπτωση. Από τους δύο τελευταίους αλγορίθμους την μικρότερη υπολογιστική πλοκή έχει αυτός των Ahn και Hyhn που εκμεταλλεύεται το γεγονός ότι το πρόβλημα που καλείται να λύσει είναι πρόβλημα διατεταγμένων ομάδων (κάτι που γίνεται και στις άλλες δύο εργασίες) ως εξής: Κάθε κατάσταση καθορίζεται από B μεταβλητές, n(1), n(2), ..., n(B), που δείχνουν πόσες και επομένως (επειδή το πρόβλημα είναι διατεταγμένων ομάδων), ποιες εργασίες έχουν διαταχθεί στις πρώτες t = n(1) + n(2) + ... + n(B) θέσεις και μια ακόμα v που δηλώνει σε ποια ομάδα ανήκει η τελευταία εργασία. Σε κάθε κατάσταση δεν αντιστοιχεί, όπως γίνεται συνήθως (όπως θα πούμε στο επόμενο κεφάλαιο), η μικρότερη δυνατή τιμή της αντικειμενικής συνάρτησης για τις πρώτες t εργασίες (δηλαδή το min $\sum_{i=1}^{t} C_i$), αλλά η μικρότερη δυνατή συνολική αύξηση της τιμής της αντικειμενικής συνάρτησης λόγω της ύπαρξης στις πρώτες t θέσεις των συγκεκριμένων t εργασιών, που δίνεται από τον αναδρομικό τύπο : $F(v,n(1),...,n(v),...,n(B))=\min_{u}\{F(u,n(1),...,n(v)-1,...,n(B))+(n-t+1)(S_{uv}+P_{v,n(v)})\}$ με αρχική τιμή F(0, 0, ...,0) = 0. Ο παραπάνω τύπος προκύπτει από τη διαπίστωση ότι η τοποθέτηση στη θέση t της εργασίας n(v) της ομάδας v αμέσως μετά από μία εργασία που ανήκει στην ομάδα u αυξάνει τους χρόνους αποπεράτωσης των n - t εργασιών που θα εκτελεστούν μετά από αυτήν, κατά $S_{uv}+P_{v,n(v)}$. Η λύση του προβλήματος δίνεται από το $\min_{v} F(v,n_1,...,n_B)$. Ο συνολικός αριθμός των καταστάσεων δεν είναι μεγαλύτερος από BN^B (κι ακόμα καλύτερα από Bn_{max}^B) κι επειδή για τον υπολογισμό της τιμής της καθεμιάς απαιτείται χρόνος O(B), η συνολική πλοκή του αλγορίθμου είναι $O(B^2N^B)$.

Κάποιος παρόμοιος αλγόριθμος δεν μπορεί να εφαρμοστεί στο δικό μας πρόβλημα ($1/S_{ij}/\sum C_i|T_{max}=0$) γιατί πρώτον, επειδή το πρόβλημα δεν είναι διατεταγμένων ομάδων δεν μπορεί να καθοριστεί το σύνολο των εργασιών που βρίσκονται στις πρώτες t θέσεις ορίζοντας μόνο το πλήθος των εργασιών κάθε ομάδας που ανήκουν σ'αυτό, αλλά πρέπει αυτό να οριστεί άμεσα, δεύτερον, για τον ίδιο λόγο δεν μπορεί να καθοριστεί η τελευταία εργασία λέγοντας μόνο σε ποια ομάδα ανήκει αλλά πρέπει πάλι να καθοριστεί άμεσα και τρίτον επειδή υπάρχουν προθεσμίες που δεν πρέπει να παραβαίνονται, είναι απαραίτητο να χρησιμοποιηθεί μια ακόμα μεταβλητή κατάστασης, ενδεικτική του χρόνου αποπεράτωσης της τελευταίας εργασίας. Οι τρεις αυτές παρατηρήσεις οδηγούν σε σχήμα Δυναμικού Προγραμματισμού με σαφώς μεγαλύτερη πλοκή από τον παραπάνω αλγόριθμο, που εξετάζεται αναλυτικά στο πέμπτο κεφάλαιο.

Ο John [John-89] και οι Liao-Huang-Tseng [LHT-92] ασχολούνται με τον καθορισμό του αποκόμματος ενός αποδοτικού προγράμματος που θα τροποποιηθεί για να δημιουργηθεί το επόμενο αποδοτικό πρόγραμμα ενός δικριτηριακού προβλήματος Χρονικού Προγραμματισμού (ενώ οι εργασίες που βρίσκονται εκτός του αποκόμματος μένουν στις θέσεις τους) όταν τα κριτήρια είναι το $\sum C_i$ και η μέγιστη τιμή μιας οποιαδήποτε αύξουσας συνάρτησης κόστους G ($1//\sum C_i,G_{max}$) για την πρώτη εργασία, ενώ στην δεύτερη εξετάζονται τα τρία δικριτηριακά προβλήματα που έχουν σαν κριτήρια δύο από τα $\sum C_i$, T_{max}, n_T. Οι Liao-Huang-Tseng αποδεικνύουν για το $1//\sum C_i$, T_{max} ότι κάθε αποδοτικό πρόγραμμα (εκτός από το πρώτο) προκύπτει αναδιατάσσοντας κατάλληλα το απόκομμα του προηγούμενου αποδοτικού προγράμματος που έχει για τελευταία εργασία αυτή με την μέγιστη καθυστέρηση, έστω v_2 και έστω ότι είναι μοναδική, και πρώτη την κοντινότερη εργασία (από αριστερά) στην v_2 με προθεσμία μεγαλύτερη από d_{v_2}, έστω v_1. Η απόδειξη βασίζεται στον τρόπο που δημιουργούμε τα αποδοτικά προγράμματα από το αλγόριθμο των Van Wassenhove και Gelders και τον χρησιμοποιούμενο απ'αυτόν αλγόριθμο του Smith. Συγκεκριμένα, ο πρώτος αλγόριθμος υπολογίζει κάθε αποδοτικό πρόγραμμα εκτός του πρώτου, αυξάνοντας τις αρχικές προθεσμίες κατά $T_{max}-1$, με T_{max} τη μέγιστη καθυστέρηση του προηγούμενου αποδοτικού προγράμματος, και λύνοντας με τις νέες προθεσμίες το $1//\sum C_i|T_{max}=0$ με τον δεύτερο αλγόριθμο. Εφόσον οι εργασίες που βρίσκονται δεξιότερα της v_2 έχουν μικρότερη καθυστέρηση από T_{max}, η τοποθέτησή τους στις νέες θέσεις είναι και πάλι εφικτή (όπως ήταν και στον υπολογισμό του προηγούμενου αποδοτικού προγράμματος) οπότε μένουν στις θέσεις τους. Οι εργασίες που βρίσκονται αριστερά της v_1 είτε δεν μπορούν λόγω προθεσμιών να τοποθετηθούν δεξιά από αυτήν, είτε έχουν μικρότερο χρόνο επεξεργασίας από την v_1, επομένως και από κάθε εργασία δεξιότερα της v_1 μέχρι και την v_2 (γιατί όπως εύκολα αποδεικνύεται η v_1 έχει μικρότερο χρόνο επεξεργασίας από την v_2 και όσες βρίσκονται ανάμεσά τους) και επομενως παραμένουν και στο νέο αποδοτικό πρόγραμμα αριστερά της v_1 και επειδή η καθυστέρηση της καθεμιάς από αυτές είναι μικρότερη από T_{max} η διάταξη τους στο προηγούμενο αποδοτικό πρόγραμμα παραμένει εφικτή και άρα ίδια και στο νέο. Οπότε το απόκομμα που μένει να αναδιαταχθεί είναι αυτό που περιέχεται μεταξύ των v_1 και v_2 (τις οποίες και συμπεριλαμβάνει), και έτσι για τον υπολογισμό του κάθε αποδοτικού προγράμματος απαιτείται η λύση ενός μικρότερου προβλήματος $1//\sum C_i|T_{max}=0$ απ'ότι απαιτεί ο αλγόριθμος των Van Wassenhove και Gelders.

Στο πρόβλημα $1/S_{ij}/\sum C_i,T_{max}$ δεν μπορεί να εφαρμοστεί κάτι παρόμοιο γιατί το $1/S_{ij}/\sum C_i|T_{max}=0$ δεν λύνεται με τον κανόνα του Smith (ή κάποιον παραπλήσιο). Πιο συγκεκριμένα, επειδή το C_{max} και γενικότερα ο χρόνος αποπεράτωσης της τελευταίας εργασίας κάθε αρχικού αποκόμματος εργασιών δεν είναι σταθερός αλλά μπορεί να μεταβάλλεται (μικραίνει ή μεγαλώνει) πηγαίνοντας από ένα αποδοτικό πρόγραμμα στο επόμενο, η συλλογιστική που οδήγησε πριν στο ότι ένα αρχικό και ένα τελικό απόκομμα παραμένουν σταθερά, δεν ισχύει πλέον.

Οι Shanthikumar-Buzacott [ShBu-82] επιλύουν το $1//\sum C_i|T_{max}=0$ διασπώντας το σε όμοια αλλά με λιγότερες εργασίες υποπροβλήματα. Συγκεκριμένα, παρουσιάζουν έναν (γραμμικό σε χρόνο) τρόπο διάσπασης της κατά EDD διάταξης των εργασιών σε m ($1 \le m \le n$) υποσύνολα έτσι ώστε κάθε εργασία του i υποσυνόλου να βρίσκεται πριν από κάθε εργασία του i + 1 υποσυνόλου σε κάθε εφικτή διάταξη, και λύνουν το πρόβλημα χωριστά για κάθε υποσύνολο, έχοντας έτσι μεγάλη χρονοβελτίωση ακόμα κι αν δεν καταφέρουν να κάνουν το m πολύ μεγάλο, αρκεί τα δημιουργηθέντα υποσύνολα να έχουν περίπου ίδιο αριθμό εργασιών. Τα υποσύνολα που προκύπτουν από τη διάσπαση της κατά EDD διάταξης έχουν την ιδιότητα ότι η νωρίτητα ($E_i=d_i-C_i$) της τελευταίας εργασίας κάθε υποσυνόλου είναι μικρότερη από τους χρόνους επεξεργασίας των εργασιών του επόμενου υποσυνόλου, οπότε όπως αποδεικνύεται, δεν είναι δυνατόν εργασία κάποιου υποσυνόλου i + 1, να προηγείται εργασίας του προηγούμενου υποσυνόλου i, γιατί τότε μία τουλάχιστον εργασία από το i υποσύνολο ή κάποιο προηγούμενο θα είναι εκπρόθεσμη.

Στο $1/S_{ij}/\sum C_i|T_{max}=0$ όμως η κατά EDD διάταξη δεν είναι σίγουρα εφικτή και η χρησιμοποίηση σαν αρχικής διάταξης, βάση της οποίας θα γίνει η διάσπαση, της λύσης του $1/S_{ij}/L_{max}$ απαιτεί την επίλυση ενός NP-complete προβλήματος. Κι αν ακόμα λυθεί το $1/S_{ij}/L_{max}$ και ορίσουμε υποσύνολα εργασιών όπως και πριν δεν θα ισχύει ότι σε κάθε εφικτή λύση κάθε εργασία του i υποσυνόλου θα βρίσκεται πριν από κάθε εργασία του i+1 υποσυνόλου. Και αυτό γιατί η μετακίνηση μιας εργασίας του i+1 υποσυνόλου πριν από κάποιες του i μπορεί να συνοδευτεί από κάποια αναδιάταξη των εργασιών ώστε να χαθεί κάποιος ή κάποιοι χρόνοι εξάρμωσης και η διάταξη να παραμείνει εφικτή.

Μια άλλη προσέγγιση στα προβλήματα Χρονικού Προγραμματισμού είναι η διαμόρφωσή τους σε προβλήματα 0-1 γραμμικού προγραμματισμού. Την ιδέα αυτή ακολουθούν οι Fry και Leong [FrLe-87] για την επίλυση ενός προβλήματος με αντικειμενική συνάρτηση έναν γραμμικό συνδυασμό του $\sum C_i$ και του $\sum E_i$ (εδώ παίρνω το E_i ίσο με $\max\{0,d_i-C_i\}$). Η μέθοδος αυτή είναι πολύ γενική και μπορεί να εφαρμοστεί σε κάθε πρόβλημα Χρονικού Προγραμματισμού άρα και στο προβλημά μας. Το βασικό της μειονέκτημα είναι ότι η μεγάλη υπολογιστική πλοκή (εκθετική ως προς τον αριθμό των μεταβλητών) των αλγορίθμων επίλυσης προβλημάτων 0-1 γραμμικού προγραμματισμού σε συνδυασμό με τον μεγάλο αριθμο μεταβλητών του διαμορφωμένου 0-1 γραμμικού προβλήματος ($O(N^2)$) έχουν σαν αποτέλεσμα να λύνονται μικρού μόνο μεγέθους προβλήματα. Στην εργασία που αναφέραμε προβλήματα με 10 εργασίες απαιτούν κατά μέσο όρο περισσότερο από 1 λεπτό χρόνο CPU, γεγονός που ωθεί τους συγγραφείς της στη σκέψη ότι είναι ίσως απαραίτητη η χρησιμοποίηση έμμεσης απαρίθμησης (Δυναμικού Προγραμματισμού ή αλγορίθμων διαμερισμού και φραγής) για την επίλυση μεγαλύτερων προβλημάτων.

Στην ειδική περίπτωση των ίσων χρόνων επεξεργασίας με αθροιστικό κριτήριο και χωρίς χρόνους εξάρμωσης, το κόστος μιας εργασίας i αν τοποθετηθεί στη θέση j είναι συγκεκριμένο έστω c_{ij} ανεξάρτητο δηλαδή των εργασιών που έχουν τοποθετηθεί στις άλλες θέσεις. Για την περίπτωση αυτή ο

Lawler [Lawl-64] διαμόρφωσε ένα πρόβλημα ανάθεσης στο οποίο το ένα σύνολο κόμβων αντιστοιχεί στο σύνολο των N εργασιών, το άλλο στο σύνολο των N θέσεων και το κόστος που συνεπάγεται η αντιστοίχιση μιας εργασίας i στη θέση j είναι c_{ij}. Οι Chen και Bulfin [ChBu-90] γενικεύουν τη σκέψη αυτή σε δικριτηριακά προβλήματα με ίσους χρόνους επεξεργασίας. Οταν η αντικειμενική συνάρτηση είναι ένας γραμμικος συνδυασμός των δύο κριτηρίων, βρίσκουν τα κατάλληλα κόστη c_{ij}, όταν τα κριτήρια είναι ιεραρχημένα αποδεικνύουν ότι αρκεί η επίλυση δύο προβλημάτων ανάθεσης κι όταν ζητούνται όλες οι αποδοτικές λύσεις προτείνουν την χρησιμοποίηση τεχνικών επίλυσης πολυκριτηριακών προβλημάτων γραμμικού προγραμματισμού και ειδικότερα έναν αλγόριθμο επίλυσης δικριτηριακού προβλήματος μεταφοράς ([AhNa-79]). Στη γενικότερη περίπτωση των όχι ίσων χρόνων επεξεργασίας η παραπάνω ιδέα δεν μπορεί να εφαρμοστεί γιατί ο χρόνος αποπεράτωσης οπότε και το κόστος μιας εργασίας i εάν τοποθετηθεί στη θέση j εξαρτάται από το ποιες εργασίες έχουν διαταχθεί πριν από αυτήν οπότε το κόστος αυτό δεν είναι εκ των προτέρων γνωστό. Μπορεί όμως να χρησιμοποιηθεί η ιδέα αυτή για τον υπολογισμό κάποιου κάτω φράγματος της αντικειμενικής συνάρτησης, το οποίο θα μπορεί να χρησιμοποιηθεί σε έναν αλγόριθμο διαμερισμού και φραγής, όπως θα πούμε παρακάτω.

3.2. Μέθοδοι υπολογισμού κάτω φράγματος

Η συνηθέστερη μέθοδος που χρησιμοποιείται για την επίλυση προβλημάτων Χρονικού Προγραμματισμού είναι ένας αλγόριθμος διαμερισμού και φραγής που ακολουθεί το εξής γενικό σχήμα: Κάθε κόμβος του δέντρου έρευνας αντιστοιχεί σε μία διάταξη ενός υποσυνόλου των εργασιών που αποτελεί αρχικό (ή τελικό, εάν η κατασκευή του προγράμματος γίνεται από το τέλος προς την αρχή) απόκομμα ενός σχηματιζόμενου προγράμματος. Στο k επίπεδο του δέντρου (η ρίζα είναι σε επίπεδο 0) βρίσκονται οι κόμβοι που αντιστοιχούν σε διατάξεις υποσυνόλων με k εργασίες. Ενας κόμβος του επιπέδου k, που αντιστοιχεί σε διάταξη π ενός συνόλου R, διακλαδίζεται σε n-k παιδιά που το καθένα αντιστοιχεί σε μία διάταξη που προκύπτει προσθέτοντας μια (διαφορετική για κάθε παιδί) από τις n-k εργασίες που δεν ανήκουν στο R στο τέλος (ή στην αρχή) της π. Σε κάθε κόμβο υπολογίζεται ένα κάτω φράγμα της τιμής που μπορεί να πάρει η αντικειμενική συνάρτηση δεδομένου ότι η διάταξη που αντιστοιχεί στον συγκεκριμένο κόμβο θα βρίσκεται στην αρχή (ή στο τέλος) του προγράμματος. Κάθε κόμβος που δεν έχει μικρότερο κάτω φράγμα από κάποιο άνω φράγμα της τιμής της βέλτιστης λύσης (άνω φράγμα που μπορεί να είναι ή τιμή μιάς εφικτής λύσης) "κλαδεύεται", δηλαδή δεν συνεχίζεται η έρευνα στους κόμβους-απογόνους του. Αυτός είναι ο μόνος, συνήθως, τρόπος να περιοριστεί ο μεγάλος (O(n!)) αριθμός κόμβων και κατ'επέκταση να "δαμαστεί" η αντίστοιχα μεγάλη πλοκή του αλγορίθμου. Γι'αυτό και είναι κρίσιμη για την ταχύτητα του αλγορίθμου η εύρεση ενός "σφιχτού" και γρήγορα υπολογιζόμενου κάτω φράγματος. Παρακάτω εξετάζουμε διάφορες μεθόδους υπολογισμού κάτω φραγμάτων ώστε να γίνει φανερό το γιατί καθεμία από αυτές δεν μπορεί να εφαρμοστεί στο $1/S_{ij}/\sum C_i|T_{max}=0$.

Το απλούστερο και συνήθως, χαλαρότερο κάτω φράγμα που χρησιμοποιήθηκε για το $1//\sum w_iC_i|T_{max}=0$ ([Bans-80]) προκύπτει αγνοώντας τους περιορισμούς προθεσμιών και λύνοντας το $1//\sum w_iC_i$ διατάσσοντας τις εργασίες κατά SWPT. Πέρα από το ότι το προκύπτον κάτω φράγμα είναι χαλαρό, η μέθοδος αυτή δεν μπορεί να εφαρμοστεί στη δική μας περίπτωση γιατί δεν έχει βρεθεί κάποιος πολυωνυμικός αλγόριθμος για την επίλυσή του $1/S_{ij}/\sum w_iC_i$ (για το οποίο, όπως έχουμε πει, μένει ανοιχτό το ερώτημα αν είναι NP-complete ή όχι) και δεν είναι δυνατόν να καλείται ένας εκθετικός αλγόριθμος σε κάθε κόμβο του δέντρου έρευνας.

Μια από τις πρώτες μεθόδους υπολογισμού κάτω φράγματος σε προβλήματα Χρονικού Προγραμματισμού, που προτάθηκε από τον Lawler [Lawl-64], είναι η επίλυση ενός προβλήματος μεταφοράς που αποτελεί χαλάρωση του αρχικού προβλήματος. Συγκεκριμένα, θεωρώντας τις χρονικές μονάδες σαν διαθέσιμα αγαθά και τις εργασίες σαν καταναλωτές ζητείται η αποδοτικότερη μεταφορά αγαθών σε καταναλωτές για κατάλληλους συντελεστές κόστους c_{it} για τη μεταφορά του αγαθού (χρονική μονάδα) t, στον καταναλωτή (εργασία) i, οπότε προκύπτει το πρόβλημα μεταφοράς :

$$\min \sum_{i=1}^{n}\sum_{t=1}^{P} c_{it}x_{it}$$

$$\text{δ.ο.} \quad \sum_{t=1}^{P} x_{it}=P_i \qquad i=1, ..., n \qquad (1)$$

$$\sum_{i=1}^{n} x_{it}=1 \qquad t=1, ..., P \qquad (2)$$

$$x_{it}\in\{0,1\} \qquad i=1, ..., n \;\; t=1, ..., P \qquad (3)$$

όπου η μεταβλητή x_{it} είναι 1 εάν η εργασία i εκτελείται στο χρονικό διάστημα [t-1, t] και 0 διαφορετικά. Αν το πρόβλημα είναι τέτοιο που η μηχανή δεν μένει καθόλου ανενεργή τότε $P=\sum_{i=1}^{n}P_i$. Διαφορετικά οι περιορισμοί (2) ισχύουν με ≤ και το P είναι ένα άνω φράγμα του C_{max}. Για κατάλληλους ανάλογα με το πρόβλημα, συντελεστές c_{it} και προσθέτοντας τον περιορισμό τα χρονικά διαστήματα που αντιστοιχούν σε κάθε εργασία να είναι συνεχόμενα το παραπάνω πρόβλημα δίνει είτε ένα κάτω φράγμα στο αρχικό πρόβλημα Χρονικού Προγραμματισμού ([GeKi-74], [GeKi-75] για το $1//\sum(w_iT_i+u_iC_i)$ και [APW-90] για το $1//\sum w_iT_i$) είτε τη βέλτιστη λύση του (προσθέτοντας ίσως κάποια σταθερά στην αντικειμενική συνάρτηση, όπως στην εργασία [DyWo-90] για το $1/r_i/\sum w_iC_i$) οπότε το πρόβλημα μεταφοράς (που δεν περιέχει τον περιορισμό να είναι συνεχόμενα τα διαστήματα που αντιστοιχούν σε κάθε εργασία) δίνει ένα κάτω φράγμα. Η μεγάλη υπολογιστική πλοκή ($O(n^2P)$) της μεθόδου αυτής την καθιστά αρκετά αργή για την ποιότητα του φράγματος που αποδίδει, όπως παρατηρούν και πειραματικά οι Van Wassenhove και Gelders [WaGe-78].

Στο δικό μας τώρα πρόβλημα ($1/S_{ij}/\sum C_i|T_{max}=0$) η λύση του προβλήματος μεταφοράς δίνει ένα κάτω φράγμα που είναι και κάτω φράγμα του ίδιου προβλήματος χωρίς χρόνους εξάρμωσης, κάτι που δεν έχει νόημα, αφού το ίδιο πρόβλημα χωρίς χρόνους εξάρμωσης μπορεί να λυθεί με ακρίβεια. Για να λάβουμε υπ'όψιν μας τους χρόνους εξάρμωσης πρέπει να προσθέσουμε περιορισμούς της μορφής $x_{it}+x_{js}\leq 1$ για κάθε i, j που ανήκουν σε διαφορετικές ομάδες και $t<s\leq t+S_{ij}$, οπότε δεν έχουμε πια πρόβλημα μεταφοράς αλλά κάποιο άλλο που δεν λύνεται με κάποιο γνωστό πολυωνυμικό αλγόριθμο. NP-complete προβλήματα Χρονικού Προγραμματισμού, όπως το $1//\sum C_i|T_{max}=0$, μπορούν να γραφτούν στην μορφή αυτή, σαν προβλήματα μεταφοράς, δηλαδή στα οποία έχουν προστεθεί περιορισμοί αμοιβαίου αποκλεισμού κάποιων "μεταφορών" (και συγκεκριμένα της μορφής $x_{it}+x_{is}\leq 1$ για $|s - t| \geq P_i$ για κάθε εργασία i). Οι περιορισμοί όμως, που προσθέτουμε εμείς παραπάνω έχουν μια συγκεκριμένη δομή και δεν ξέρουμε εάν το πρόβλημα που προκύπτει είναι NP-complete ή λύνεται πολυωνυμικά.

Εχουμε πει ότι προβλήματα Χρονικού Προγραμματισμού με ίσους χρόνους επεξεργασίας των εργασιών με αθροιστικά κριτήρια και χωρίς χρόνους εξάρμωσης μπορούν να λυθούν μετασχηματιζόμενα σε προβλήματα ανάθεσης, αντιστοιχίζοντας στην τοποθέτηση της εργασίας i στη θέση της διάταξης j ένα κόστος c_{ij} που είναι από την αρχή γνωστό, εφόσον ο χρόνος αποπεράτωσης της εργασίας που τοποθετείται σε οποιαδήποτε θέση είναι γνωστός αφού οι χρόνοι επεξεργασίας είναι ίσοι μεταξύ τους (και δεν υπάρχουν χρόνοι εξάρμωσης). Στη γενικοτερη περίπτωση των όχι ίσων χρόνων επεξεργασίας οι Rinnooy Kan, Lageweg και Lenstra [RLL-75] χρησιμοποιούν την παραπάνω ιδέα για να εξάγουν ένα κάτω φράγμα, ως εξής: υπολογίζουν τη νωρίτερη δυνατή στιγμή που μπορεί η εργασία i να τελειώσει εάν εκτελεστεί στη θέση j του προγράμματος (εάν δεν υπάρχουν - είτε εξ αρχής, είτε εξαγόμενοι από κάποιον αλγόριθμο - κανόνες προτεραιότητας μεταξύ των εργασιών, η νωρίτερη αυτή στιγμή βρίσκεται θεωρώντας ότι στις j-1 πρώτες θέσεις βρίσκονται ισάριθμες εργασίες με τους μικρότερους χρόνους επεξεργασίας) και το κόστος που θα έχει εάν τελειώσει τότε, έστω c'_{ij}, που αποτελεί κάτω φράγμα του κόστους που επιφέρει η τοποθέτηση της εργασίας i στη θέση j σε οποιοδήποτε πρόγραμμα, οπότε η λύση ενός όμοιου προβλήματος ανάθεσης με το προηγούμενο με κόστη τώρα τα c'_{ij} δίνει ένα κάτω φράγμα στη βέλτιστη λύση του αρχικού προβλήματος.

Επειδή, για το δικό μας τώρα πρόβλημα, δεν ξέρουμε κάποιον πολυωνυμικό τρόπο να βρίσκουμε τη νωρίτερη στιγμή που μπορεί να ξεκινήσει κάποια εργασία i εάν τοποθετηθεί στη θέση j, δε μπορούμε σε πολυωνυμικό χρόνο να βρούμε κάτω φράγματα c'_{ij} όπως πριν που να λαμβάνουν υπ'όψιν τους τους χρόνους εξάρμωσης, εκτός και αν κάνουμε το εξής : να τοποθετήσουμε στον άξονα του χρόνου μια φορά τον χρόνο εξάρμωσης της κάθε ομάδας με τρόπο τέτοιο που η συνεπαγόμενη αύξηση της τιμής της αντικειμενικής συνάρτησης να μην είναι μεγαλύτερη από την αντίστοιχη αύξηση που προκαλεί η ύπαρξη των χρόνων εξάρμωσης σε ένα οποιοδήποτε πρόγραμμα (ακολουθώντας την ιδέα για υπολογισμό κάτω φράγματος του Γεωργίου [Γεωρ-91]) και με δεδομένη την ύπαρξη των τοποθετημένων χρόνων εξάρμωσης και χωρίς να τοποθετήσουμε άλλους να υπολογίσουμε τα c'_{ij} βάζοντας στις πρώτες j-1 θέσεις ισάριθμες εργασίες με τους μικρότερους χρόνους επεξεργασίας. Ετσι όμως δεν παίρνουμε παρά ένα κάτω φράγμα (ή τη βέλτιστη λύση) στο $1//\sum C_i$ με κάποιες συγκεκριμένες χρονικές στιγμές που η μηχανή δεν λειτουργεί του οποίου η βέλτιστη λύση αποτελεί κάτω φράγμα της βέλτιστης λύσης του ίδιου προβλήματος με προθεσμίες, το οποίο εύκολα αποδεικνύεται ότι λύνεται πολυωνυμικά όπως το αντίστοιχο πρόβλημα χωρίς τις ανενεργές περιόδους της μηχανής, οπότε ούτε κι έτσι δεν παίρνουμε κάποιο χρήσιμο κάτω φράγμα και άρα γενικά η μέθοδος αυτή δεν μπορεί να χρησιμοποιηθεί στη περίπτωσή μας.

Μια παραλλαγή της μεθόδου αυτής [PiQu-78] είναι να θεωρήσουμε c_{hij} (ή πιο συγκεκριμένα κάτω φράγματα αυτών c'_{hij}) σαν το κόστος τοποθέτησης της εργασίας i στη θέση j αμέσως μετά την εργασία h και να βρεθεί η διάταξη (σ(1), σ(2), ..., σ(n)) των εργασιών που ελαχιστοποιεί την ποσότητα $\sum_{j=1}^{n} c_{\sigma(j-1),\sigma(j),j}$ (με σ(0) = 0, μια νοητή εργασία με $P_0=0$) και το κόστος αυτής της διάταξης που αποτελεί κάτω φράγμα για το αρχικό πρόβλημα. Στο δικό μας πρόβλημα όπως και πριν, δεν μπορούμε να υπολογίσουμε χρήσιμα κάτω φράγματα για τα c_{hij}, οπότε και η παραλλαγή αυτή δε μπορεί να χρησιμοποιηθεί.

Μια άλλη μέθοδος υπολογισμού του κάτω φράγματος για το $1//\sum C_i|T_{max}=0$ βασίζεται στη σκέψη ότι αν σπάσουμε κάθε εργασία i σε P_i κομμάτια με χρόνους επεξεργασίας 1 και βάρος w_i/P_i το καθένα [Posn-85] ή σε w_i κομμάτια με χρόνους επεξεργασίας P_i/w_i και βάρος 1 το καθένα [BaAh-87] (και προσθέσουμε στην αντικειμενική συνάρτηση μια σταθερή ποσότητα για κάθε εργασία που ισούται με τη μείωση της αντικειμενικής συνάρτησης που προκαλείται από το σπάσιμο αυτό) η λύση του προβλήματος που προκύπτει αποτελεί κάτω φράγμα για τη λύση του αρχικού προβλήματος, γιατί το νέο πρόβλημα είναι μια χαλάρωση του αρχικού αφού οι εργασίες μπορεί και να μην εκτελούνται αδιάσπαστες. Η λύση του νέου (χαλαρωμένου) προβλήματος υπολογίζεται σε πολυωνυμικό χρόνο από τον ευρηματικό κανόνα του Smith [Smit-56], που αποτελεί γενίκευση στην περίπτωση που υπάρχουν βάρη, του κανόνα που λύνει το $1//\sum C_i|T_{max}=0$, και που για ίσους χρόνους επεξεργασίας ή ίσα βάρη (που είναι ουσιαστικά το $1//\sum C_i|T_{max}=0$) δίνει βέλτιστη λύση και λέει ότι τελευταία στη βέλτιστη διάταξη k εργασιών μπαίνει εκείνη με το μεγαλύτερο P/w από όσες μπορούν να μπουν στη θέση εκείνη, δηλαδή από εκείνες τις εργασίες j για τις οποίες ισχύει $d_j \geq \sum_{i=1}^{k} P_i$.

Στο δικό μας πρόβλημα δεν μπορούμε να χρησιμοποιήσουμε ένα παρόμοια κατασκευασμένο κάτω φράγμα γιατί κάποιος παρόμοιος αλγόριθμος δεν μπορεί να λύσει το πρόβλημα με ίσους χρόνους επεξεργασίας ή ίσα βάρη (που είναι ουσιαστικά, το αρχικό πρόβλημα που είναι NP-complete), γιατί ούτε πότε τελειώνει η εκτέλεση κάποιων εργασιών οπότε και το ποιες μπορούν να τοποθετηθούν στην τελευταία θέση είναι γνωστό, ούτε κι αν αυτό ήταν γνωστό θα ίσχυε οτι θα τοποθετούνταν τελευταία αυτή με το μεγαλύτερο P/w απ'όλες αυτές.

Μια ιδέα που έχει χρησιμοποιηθεί μερικές φορές, με διαφορετικούς τρόπους ή σε κάποια παραλλαγή της για τον υπολογισμό κάτω φράγματος είναι η χαλάρωση Lagrange. Για πρώτη φορά χρησιμοποιείται αυτή η ιδέα από τον Fisher [Fish-76] στην επίλυση του $1//\sum T_i$ (με τρόπο που άμεσα γενικεύεται οπως παρατηρείται στην εργασία [APW-90], και στο $1//\sum w_i T_i$) : Αρχικά εξάγονται κάποιες συνθήκες προτεραιότητας της μορφής $i \rightarrow j$, με την έννοια ότι υπάρχει βέλτιστη διάταξη όπου η εργασία i προηγείται της j, οι οποίες χρησιμοποιούνται παρακάτω στον υπολογισμό του κάτω φράγματος. Ορίζοντας x_i τη στιγμή έναρξης της εργασίας i, a_i, b_i τη μικρότερη και μεγαλύτερη τιμή αντίστοιχα που μπορεί να πάρει η x_i δοθεισών των συνθηκών προτεραιότητας, A_i ένα (τυχαίο) υποσύνολο του συνόλου των άμεσων απογόνων της i με $|A_i| \leq 1$ (η αναγκαιότητα του περιορισμού αυτού, του να λαμβάνουμε υπόψιν μας, δηλαδή, ένα υποσύνολο του συνόλου των σχέσεων προτεραιότητας τέτοιο ώστε κάθε εργασία να έχει το πολύ έναν απόγονο, γίνεται φανερή στην επίλυση του προβλήματος Langrange), $B_i = \{j : i \in A_j\}$, $T = \sum_{i=1}^{n} P_i$ και $g_t(x)$ ($t \in \{1, 2, ..., T\}$) το πλήθος των εργασιών που εκτελούνται κατά τη διάρκεια του διαστήματος [t - 1, t], όπου x το διάνυσμα των χρόνων έναρξης των εργασιών, έχουμε ότι η λύση του προβλήματος δίνεται από τη λύση του : $\min_{x\in X}\sum_{i=1}^{n}\max(x_i+P_i-d_i,0)$ δοθέντος ότι $g_t(x) = 1$, $t = 1, 2, ..., T$ και $X = \{x : a_i \leq x_i \leq b_i,\ x_j \geq x_i+P_i$ για $j \in A_i,\ x_i \in \mathbf{N},\ i = 1, ..., n\}$. Η ύπαρξη του περιορισμού $x \in X$ δεν επηρεάζει τη λύση του προβλήματος, αφού έτσι κι αλλιώς το X είναι ένα υποσύνολο του R^n που έχει κατασκευαστεί έτσι ώστε να περιέχει τη βέλτιστη λύση του προβλήματος. Εμφανίζεται όμως αυτός ο περιορισμός εδώ για να εμφανιστεί και στο πρόβλημα Langrange, που χωρίς αυτόν θα έδινε ένα πολύ χαλαρότερο κάτω φράγμα. Χαλαρώνοντας τους περιορισμούς $g_t(x) = 1$ παίρνουμε το πρόβλημα Lagrange :

$$w(u)=\min_{x\in X}\left(\sum_{i=1}^{n}\max(x_i+P_i-d_i,0)+\sum_{t=1}^{T}u_t g_t(x)\right)-\sum_{t=1}^{T}u_t=\min_{x\in X}\left\{\sum_{i=1}^{n}\left(\max(x_i+P_i-d_i,0)+\sum_{t=x_i+1}^{x_i+P_i}u_t\right)\right\}-\sum_{t=1}^{T}u_t$$

και θέτοντας $L_i(x_i,u)=\max(x_i+P_i-d_i,0)+\sum_{t=x_i+1}^{x_i+P_i}u_t$ και $L(x,u)=\sum_{i=1}^{n}L_i(x_i,u)$, το πρόβλημα $w(u)=\min_{x\in X}L(x,u)-\sum_{t=1}^{T}u_t$, η λύση του οποίου για ένα συγκεκριμένο διάνυσμα δυϊκών μεταβλητων u ανάγεται στη λύση του $\min_{x\in X}L(x,u)$. Ορίζοντας $\overline{B_i}$ το σύνολο όλων των προγόνων της εργασίας i βάσει των συνθηκών προτεραιότητας που ορίζονται από τα A_j (και όχι μόνον των άμεσων, όπως το B_i), $\hat{B}_i=\overline{B_i} \cup \{i\}$, $c_i(t)=\min\sum_{j\in B_i}L_j(x_j,u)$ για κάθε $x \in X$ με $x_i+P_i\leq t$, μια αρίθμηση των εργασιών έτσι ώστε $B_i \subseteq \{1, 2, ..., i - 1\}$ και υποθέτοντας ότι $\hat{B}_n = \{1, 2, ..., n\}$ (κάτι που δεν ισχύει αν εισάγουμε μια νοητή εργασία με $B = \{i : A_i = \emptyset\}$, $p = 0$ και $d = \infty$, ώστε πάντα όλες οι εργασίες να είναι πρόγονοι της τελευταίας - έστω n-οστης - εργασίας) έχουμε ότι $\min_{x\in X}L(x,u)=c_n(T)$ και

$$c_i(t)=\begin{cases} +\infty, & t=0,\dots,a_i+P_i-1 \\ \min\{c_i(t-1),L_i(t-P_i,u)+\sum_{k\in B_i}c_k(t-P_i)\}, & t=a_i+P_i,\dots,b_i+P_i \\ c_i(b_i+P_i), & t=b_i+P_i+1,\dots,T \end{cases}$$

με i = 1, 2, ..., n. Επειδή οι σχέσεις προτεραιότητας που λαμβάνουμε υπόψιν είναι τέτοιες που κάθε εργασία να έχει το πολύ έναν άμεσο απόγονο, δεν υπάρχουν δύο άμεσοι πρόγονοι της ίδιας εργασίας με κάποιο κοινό πρόγονο, οπότε δεν υπάρχει εργασία η συνεισφορά της οποίας στο κόστος να έχει προστεθεί πάνω από μία φορά στο $\sum_{k\in B_i}c_k(t-P_i)$ για κάποια εργασία k. Από τον αναδρομικό αυτό τύπο υπολογίζουμε το $\min_{x\in X} L(x,u)$ (= $c_n(T)$) και άρα το w(u), η τιμή του οποίου βελτιώνεται με subgradient optimization.

Για να μπορέσει αυτή η μέθοδος να εφαρμοστεί στο $1/S_j/\sum C_i|T_{max}=0$ πρέπει να υπάρχουν αρκετές συνθήκες προτεραιότητας μεταξύ εργασιών διαφορετικών ομάδων, γιατί αν δεν υπάρχουν τέτοιες συνθήκες, το w(u) δεν αποτελεί παρά κάτω φράγμα του προβλήματος χωρίς χρόνους εξάρμωσης, αφού οι χρόνοι εξάρμωσης δεν αναμιγνύονται καθόλου με τον υπολογισμό των $c_i(t)$. Από το δεύτερο θεώρημα εξαγωγής κανόνων προτεραιοτήτας που αναφέραμε στο Κεφάλαιο 2, προκύπτει ότι αν υπάρχουν αρκετές εργασίες με σχετικά μικρές προθεσμίες, φαίνεται πιθανό να εξάγονται συνθήκες προτεραιότητας μεταξύ εργασιών διαφορετικών ομάδων και η μέθοδος αυτή έχει ελπίδες να δουλέψει αποτελεσματικά. Διαφορετικά δεν υπάρχει ελπίδα να δώσει σφιχτό κάτω φράγμα. Οι μετατροπές που θα γίνουν για την περίπτωσή μας στον αλγόριθμο υπολογισμού του κάτω φράγματος, w(u), είναι: το T τώρα θα είναι ένα άνω φράγμα στο C_{max} (π.χ. $T=\sum_{i=1}^{n}(P_i+S_i)$), τα a_i και b_i (οριακές τιμές της x_i ώστε να τηρούνται οι συνθήκες προτεραιότητας) θα υπολογίζονται λαμβάνοντας υπ'όψιν τους χρόνους εξάρμωσης προγόνων και απογόνων, δηλαδή $a_i=\sum_{j\in\beta_i}P_j+\sum_{l\in\{g(j):j\in\hat{\beta}_i\}}S_l$ και $b_i=T-\sum_{j\in\alpha_i}P_j-\sum_{l\in\{g(j):j\in\hat{\alpha}_i\}}S_l$ (α_i το σύνολο όλων των απογόνων της i, β_i το σύνολο όλων των προγόνων της, $\hat{\alpha}_i=\alpha_i\cup\{i\}$ και $\hat{\beta}_i=\beta_i\cup\{i\}$). $L_i(x_i,u)=x_i+\sum_{t=x_i+1}^{x_i+P_i}u_t$ και τέλος

$$c_i(t)=\begin{cases} +\infty, & t=0,\ldots,a_i+P_i-1\\ \min\{c_i(t-1),L_i(t-P_i,u)+\sum_{k\in B_i}C_k(t-P_i-S_{ki})\}, & t=a_i+P_i,\ldots,\min\{b_i+P_i,d_i\}\\ c_i(\min\{b_i+P_i,d_i\}), & t=\min\{b_i+P_i,d_i\},\ldots.T\end{cases}$$

Επειδή στην περίπτωσή μας η εισαγωγή μιας νοητής εργασίας, σαν n-οστής επηρεάζει τα $c_n(t)$ θα υπολογίσουμε το $\min_{x\in X}L(x,u)$ σαν $\sum_{k:A_k=\varnothing}c_k(T)$, όπως θα μπορούσε να είχε συμβεί και στο αρχικό πρόβλημα. Πρέπει πάντως να τονισθεί, ότι η χρησιμοποίηση της μεθόδου αυτής για το $1/S_{ij}/\sum C_i|T_{max}=0$ έχει νόημα μόνο όταν ισχύει η απαραίτητη προυπόθεση των στενών προθεσμιών, οπότε δεν μπορεί να χρησιμοποιηθεί σαν μία γενική μέθοδος επίλυσης του προβλήματός μας.

Μια παρόμοια μέθοδος εφαρμόζεται και στην εργασία [FLLR-83]. Το πρόβλημα που εξετάζεται εκεί είναι: n εργασίες πρέπει να εκτελεσθούν σε m μηχανές. Η εργασία j αποτελείται από μία σειρά λειτουργιών ($m_{j-1}+1,\ldots,m_j$) (j = 1, 2, ..., n), με $0=m_0<m_1<\cdots<m_n$. Η λειτουργία u εκτελείται στη μηχανή μ_u, έχει χρόνο επεξεργασίας P_u και πρέπει να έχει τελειώσει πριν ξεκινήσει η u + 1 ($u=m_{j-1}+1,\ldots,m_j-1$ j = 1, 2, ..., n). Στόχος είναι να ελαχιστοποιηθεί ο μέγιστος χρόνος αποπεράτωσης z (=C_{max}). Η εξαγωγή κάτω φράγματος γίνεται με τη χρήση μιας μεθόδου που ονομάζεται surrogate duality relaxation και σημαίνει την αντικατάσταση ενός συνόλου περιορισμών με ένα γραμμικό συνδυασμό τους. Η μέθοδος εφαρμόζεται στους περιορισμούς χωρητικότητας των μηχανών και στους περιορισμούς προτεραιότητας μεταξύ των λειτουργιών των εργασιών. Μια τυποποιημένη μορφή του προβλήματος σαν πρόβλημα μαθηματικού

προγραμματισμού είναι :

$\min z$

δεδομένου ότι

$$\left.\begin{array}{l}(1.1)\quad x_u+P_u-x_{u+1}\le 0 \quad (u=m_{j-1}+1,\ldots,m_j-1 \quad j=1,2,\ldots,n)\\ (1.2)\quad x_u+P_u-z\le 0 \quad (u=1,\ldots,m_n)\\ (1.3)\quad x_u\ge 0\end{array}\right\} \quad (1)$$

όπου x_u η στιγμή έναρξης της λειτουργίας u και λείπουν οι περιορισμοί χωρητικότητας που μπορούν να εκφραστούν είτε σαν

$$\left.\begin{array}{l}(2.1)\quad x_u+P_u-x_v-TY_{uv}\le 0\\ (2.2)\quad Y_{uv}+Y_{vu}=1\\ (2.3)\quad Y_{uv}\in\{0,1\}\end{array}\right\} \text{ για όλα τα } (u, v) \text{ με } \mu_u=\mu_v \quad (2)$$

όπου $Y_{uv}=0$ εάν η u προηγείται της v και 1 εάν η v προηγείται της u και T ένας "αρκετά μεγάλος" ακέραιος (μπορεί να είναι ένα άνω φράγμα του βέλτιστου z), είτε σαν

$n_{it}(x)-1\le 0$ (i=1, ..., m t=1, 2, ..., T) (3)

όπου $n_{it}(x)$ ο αριθμός των λειτουργιών που εκτελούνται στη μηχανή i τη στιγμή t για ένα διάστημα χρόνων έναρξης x.

Θεωρώντας ότι εκφράζουμε τους περιορισμούς χωρητικότητας με τους περιορισμούς (2) και αντικαθιστώντας τους περιορισμούς (2.1) με κάποιο γραμμικό συνδυασμό τους προκύπτει πρόβλημα που, όπως αποδεικνύεται εύκολα (αφού το T είναι μεγαλος ακέραιος), πληροί τους νέους περιορισμούς (2) για κάθε διάνυσμα x που ικανοποιεί τους περιορισμούς (1) οπότε η λύση του δίνει το προφανές και χαλαρό κάτω φράγμα $z=\max_j \sum_{u=m_{j-1}+1}^{m_j} P_u$.

Στην περίπτωση του δικού μας προβλήματος που δεν έχουμε τους περιορισμούς (1.1) και οι (1.2) γίνονται $x_u+P_u-d_u\le 0$ (τώρα το u συμβολίζει εργασία) και η αντικειμενική συνάρτηση $\sum_{u=1}^{n}(x_u+P_u)$ αφού, όπως και πριν, οι περιορισμοί (2) ικανοποιούνται (μετά την αντικατάσταση των (2.1) από τον γραμμικό συνδυασμό τους) τα x_u μπορούν να μικρύνουν μέχρι την τιμή 0, οπότε δεν παίρνουμε κάποιο ουσιαστικό κάτω φράγμα. Ας σημειώσουμε ότι στη δική μας περίπτωση οι περιορισμοί (2.1) γίνονται $x_u+P_u+S_{uv}-x_v-TY_{uv}\le 0$.

Εκφράζοντας τους περιορισμούς χωρητικότητας με τους περιορισμούς (3) και αντικαθιστώντας αυτούς με έναν γραμμικό συνδυασμό τους παίρνουμε τον περιορισμό $\sum_{i=1}^{m}\sum_{t=1}^{T}\beta_{it}(n_{it}(x)-1)\le 0$ (3'), όπου β_{it} είναι μη αρνητικά βάρη. Για ένα συγκεκριμένο διάνυσμα βαρών β και ένα συγκεκριμένο z μπορεί να βρει κανείς τη μικρότερη τιμή που μπορεί να πάρει το αριστερό μέλος της ανισότητας (3'), δοθέντος ότι τηρούνται οι περιορισμοί (1), με ένα τρόπο παρόμοιο με αυτόν που χρησιμοποιήθηκε στην εργασία που περιγραφηκε προηγουμένως ([Fish-74]) :

$$\min_X \sum_{i=1}^{m}\sum_{t=1}^{T}\beta_{it}(n_{it}(x)-1)=\min_X\left(\sum_{j=1}^{n}\sum_{u=m_{j-1}+1}^{m_j}\sum_{t=x_u+1}^{x_u+P_u}\beta_{\mu_u t}\right)-\sum_{i=1}^{m}\sum_{t=1}^{T}\beta_{it}$$

και θέτοντας

$$B_j(\beta,z)=\min_X \sum_{u=m_{j-1}+1}^{m_j} \sum_{t=x_u+1}^{x_u+P_u} \beta_{\mu_u t}$$

η μικρότερη τιμή του αριστερού μέλους της (3’) είναι

$$\sum_{j=1}^{n} B_j(\beta,z) - \sum_{i=1}^{m}\sum_{t=1}^{T} \beta_{it}.$$

Ορίζοντας $b_u(\tau)$ το μικρότερο κόστος για να εκτελεσθούν οι λειτουργίες $m_{j-1}+1$, ..., u στο διάστημα [0, τ] έχουμε ότι $B_j(\beta,z)=b_{m_j}(z)$, με $b_u(\tau)$ να υπολογίζεται από τον αναδρομικό τύπο

$$b_u(\tau)=\min\{b_u(\tau-1), b_{u-1}(\tau-p_u)+\sum_{t=\tau-P_u+1}^{\tau} \beta_{\mu_u t}\}$$

Ξεκινώντας λοιπόν από ένα κάτω φράγμα του z και αυξάνοντας την τιμή του κάθε φορά κατά 1, μπορούμε να υπολογίσουμε κάθε φορά τα $B_j(\beta,z)$ και κατ’επέκταση την μικροτερη τιμή του αριστερού μέλους της (3’). Η πρώτη (άρα μικροτερη) τιμή του z για την οποία η (3’) θα ισχύει είναι ένα κάτω φράγμα που μπορεί να βελτιωθεί χρησιμοποιώντας μια απλοποιημένη μορφή του subgradient optimization.

Γυρνώντας τώρα στο δικό μας πρόβλημα παρατηρούμε ότι οι αντίστοιχοι περιορισμοί των περιορισμών (1) και (3) δεν αρκούν για να εισάγουν τους χρόνους εξάρμωσης στο πρόβλημα, οπότε εάν εκφράσουμε τον περιορισμό χωρητικότητας της μηχανής με τους περιορισμούς (3) θα χρειαστούμε κάποιους ακόμα περιορισμούς που θα αντιπροσωπεύουν την ύπαρξη των χρόνων εξάρμωσης. Τέτοιοι περιορισμοί μπορεί να είναι οι περιορισμοί (2) (περιλαμβάνοντας και τους χρόνους εξάρμωσης όπως είπαμε παραπάνω) μόνο όμως για τα ζεύγη εργασιών (u, v) που οι εργασίες u και v ανήκουν σε διαφορετικές ομάδες. Ακολουθώντας τη μέθοδο υπολογισμού κάτω φράγματος που περιγράφηκε παραπάνω έχουμε ότι ένα κάτω φράγμα δίνεται από το μικρότερο c για το οποίο η λύση του $\min_X \left(\sum_{u=1}^{n} \sum_{t=x_u+1}^{x_u+P_u} \beta_t \right) - \sum_{t=1}^{T} \beta_t$ δεδομένου ότι :

(α) $\sum_{u=1}^{n}(x_u+P_u)=c$

(β) $x_u+P_u \le d_u$

$$(\gamma) \quad \left.\begin{array}{c} x_u+P_u+S_{uv} \le x_v+TY_{uv} \\ Y_{uv}+Y_{vu}=1 \\ Y_{uv} \in 0,1 \end{array}\right\} \text{ για κάθε } (u, v) \text{ με } g(u)\neq g(v)$$

είναι αριθμός ≤ 0. Το πρόβλημα αυτό όμως (που πιθανώς να είναι NP-complete) δε λύνεται με μία αντίστοιχη αναδρομική σχέση με το προηγούμενο γιατί πρώτον δεν υπάρχει εδώ μια διάταξη των εργασιών, όπως η διάταξη των λειτουργιών σε κάθε εργασία που υπήρχε πριν και δεύτερον μια αναδρομική σχέση, παρόμοια τουλάχιστον με την προηγούμενη, δε θα μπορούσε να λάβει υπ’όψιν της τους περιορισμούς (α) και (γ). Αρα με το τρόπο αυτό δεν μπορούμε να εξάγουμε κάτω φράγμα για το πρόβλημά μας.

Είχαμε τέλος, πει ότι στην εργασία αυτή χρησιμοποιούν αυτόν το μηχανισμό (surrogate duality relaxation) και για τους περιορισμούς προτεραιότητας μεταξύ των λειτουργιών των εργασιών (οπότε προκύπτει πρόβλημα που σπάει σε τόσα υποπροβλήματα όσα και οι μηχανές και σε καθένα από αυτά για να ελεγχθεί εάν για συγκεκριμένο z ισχύει ο γραμμικός συνδυασμός των περιορισμών που προστέθηκαν λύνεται ένα πρόβλημα παρόμοιο του $\sum w_iC_i$ για κατάλληλα w_i). Αυτό όμως δε βρίσκει εφαρμογή στο πρόβλημά μας, τουλάχιστον γιατί περιορισμοί προτεραιότητας δεν υπάρχουν. Κι αν εισάγουμε σαν τέτοιους περιορισμούς τους κανόνες προτεραιότητας που εξάγονται με τους δύο τρόπους που αναφέραμε στο κεφάλαιο 2, το προκύπτον πρόβλημα θα είναι τουλάχιστον τόσο δύσκολο όσο το αρχικό, δηλαδή NP-complete. Αρα τελικά, η μέθοδος που χρησιμοποιείται σ'αυτήν την εργασία, τουλάχιστον σε κάποια παρόμοια μορφή, δε μπορεί να χρησιμοποιηθεί στην περίπτωσή μας.

Χαλάρωση Lagrange, σε άλλους περιορισμούς, χρησιμοποιούν και οι Potts και Van Wassenhove [PoWa-85] για το $1//\sum w_iT_i$. Το πρόβλημα γράφεται:

$\min\sum_{i=1}^{n} w_iT_i$

δ.ο. $T_i\geq 0$ $\quad(i=1, ..., n)\quad$ (1)

$T_i\geq C_i-d_i$ $\quad(i=1, ..., n)\quad$ (2)

περιορισμοί χωρητικότητας μηχανής (3).

Χαλαρώνοντας τους περιορισμούς (2), με κάποιο διάνυσμα μη αρνητικών συντελεστών u, παίρνουμε το πρόβλημα Lagrange :

$$L(u) = \min \sum_{i=1}^{n}\left[(w_i-u_i)T_i+u_i(C_i-d_i)\right]$$

δ.ο. $T_i\geq 0$ $\quad(i=1, ..., n)$

περιορισμοί χωρητικότητας μηχανής

Για κάποια (τυχαία) διάταξη των εργασιών, με χρόνους αποπεράτωσης αυτών C_i^*, ψάχνουμε για το διάνυσμα πολλαπλασιαστών εκείνο που, από όσα u έχουν σα βέλτιστη λύση στο L(u) την προκειμένη διάταξη, έχει τη μέγιστη τιμή του L(u). Για να μην μικραίνει απεριόριστα το L(u) πρέπει $u_i\leq w_i$ για κάθε i=1, ..., n (διαφορετικά, εάν $u_j>w_j$ για κάποια εργασία j, τίποτα δεν εμποδίζει να έχουμε $T_j\to+\infty$ οπότε $L(u) \to -\infty$ και δεν παίρνουμε (ουσιαστικά) κάτω φράγμα) οπότε και για κάθε i έχουμε $T_i=0$, στην λύση του L(u). Για να είναι τώρα μια διάταξη βέλτιστη λύση του L(u) πρέπει και αρκεί να έχει τις εργασίες της διατεταγμένες κατά αύξουσα σειρά του P_i/u_i. Με βάση λοιπόν, τα παραπάνω έχουμε ότι το διάνυσμα που ψάχνουμε (και κατ'επέκταση το κάτω φράγμα που ψάχνουμε που δίνεται από την τιμή του L(u) για το συγκεκριμένο διάνυσμα) δίνεται από τη λύση του :

$$\max \sum_{i=1}^{n} u_i(C_i^*-d_i)$$

δ.ο. $P_i/u_i \leq P_{i+1}/u_{i+1}$ $\quad i=1, ..., n - 1$

$0\leq u_i\leq w_i$ $\quad i=1, ..., n$

που όπως αποδεικνύεται λύνεται σε γραμμικό χρόνο.

Στο δικό μας πρόβλημα η μέθοδος αυτή δε βρίσκει εφαρμογή γιατί χαλαρώνοντας τους περιορισμούς των προθεσμιών (που μπορούμε να τους θεωρήσουμε σαν τους αντίστοιχους των περιορισμών (2) του παραπάνω προβλήματος) προκύπτει το $\sum w_iC_i$ (με $w_i=1+u_i$, για κάθε $i=1,...,n$) με χρόνους εξάρμωσης για τους οποίους δεν έχουμε καμιά ικανή συνθήκη βελτίστου και άρα δεν μπορούμε να πάρουμε μια (τυχαία) διάταξη εργασιών και να βρούμε τιμές για τα w_i (και άρα για τα u_i) για τις οποίες η δοθείσα διάταξη να είναι βέλτιστη.

Προσπάθειες να βρεθεί ικανή συνθήκη βελτίστου για το $1/S_j/\sum w_iC_i$ δεν έδωσαν κάποιο θετικό αποτέλεσμα. Διατάσσοντας τις εργασίες σε κάθε ομάδα κατ'αύξοντα λόγο p/w και συμβολίζοντας p_{ki} και w_{ki} το χρόνο επεξεργασίας και το βάρος αντίστοιχα της i εργασίας (σύμφωνα με την παραπάνω διάταξη) της k ομάδας έχουμε ότι μια ικανή συνθήκη για να είναι βέλτιστη μια διάταξη στην οποία οι ομάδες θα εμφανίζονται αδιάσπαστες είναι $\frac{S_k+\sum_{i=a}^{b}P_{ki}}{\sum_{i=a}^{b}w_{ki}} \leq \frac{S_l+\sum_{i=c}^{d}P_{li}}{\sum_{i=c}^{d}w_{li}}$ για κάθε a, b, c, d, k, l με $1 \leq a \leq b \leq n_k$, $1 \leq c \leq d \leq n_l$ και $k<l$ (υποθέτουμε ότι οι ομάδες έχουν αριθμηθεί ανάλογα με τη θέση που κατέχουν στη διάταξη), κάτι που αποδεικνύεται ότι ισχύει εάν για κάθε a, c, k με $1 \leq a \leq n_k$, $1 \leq c \leq n_{k+1}$, και $k \leq b-1$, ισχύει : $\frac{S_k+P_{ka}}{w_{ka}} \leq \frac{S_{k+1}+\sum_{i=1}^{c}P_{k+1,i}}{\sum_{i=1}^{c}w_{k+1,i}}$ οπότε η διάταξη αυτή (με τις αδιάσπαστες ομάδες) είναι βέλτιστη εάν για τα βάρη ισχύει :

$$w_{ki} \leq 1 \qquad i=1, ..., n_k \quad k=1, ..., B \qquad (1) \quad (\text{γιατί } w_i=1+u_i \text{ με } u_i \geq 0)$$

$$w_{ki} \leq \frac{P_{ki}w_{k,i-1}}{P_{k,i-1}} \qquad i=2, ..., n_k \quad k=1, ..., B \qquad (2)$$

$$w_{ki} \leq \frac{S_k+\sum_{j=1}^{i}P_{kj}}{\max\limits_{1 \leq j \leq n_{k-1}} \frac{S_{k-1}+P_{k-1,j}}{w_{k-1,j}}} - \sum_{j=1}^{i-1}w_{kj} \qquad i=1, ..., n_k \quad k=2, ..., B \qquad (3)$$

και θα ήταν καλό να βρεθούν τα w_{ki} εκείνα που, για δοθέντα C_{ki} (δηλαδή δοθείσα διάταξη), μεγιστοποιούν το $\sum_{ki}w_{ki}(C_{ki}-d_{ki})$ γιατί αυτό συν κάποια σταθερά αποτελούν την τιμή της αντικειμενικής συνάρτησης του προβλήματος Lagrange και άρα το κάτω φράγμα. Οχι όμως μόνο δεν μπορούμε να μεγιστοποιήσουμε την παραπάνω ποσότητα αλλά ούτε καν ξέρουμε κάποιο τρόπο εύρεσης βαρών που να ικανοποιούν τις σχέσεις (1), (2) και (3) κι ακόμα περισσότερο ούτε καν ξέρουμε αν υπάρχουν τέτοια βάρη.

Μια άλλη σκέψη είναι να βρεθεί ικανή συνθήκη για να είναι βέλτιστη η διάταξη των εργασιών κατά αύξοντα λόγο P/w (κατά SWPT δηλαδή), ανεξάρτητα ομάδας. Αποδεικνύεται ότι μια ικανή συνθήκη για να ισχύει αυτό είναι για κάθε ζευγάρι εργασιών i, j με $P_i/w_i \leq P_j/w_j$ (ισοδύναμα $P_iw_j \leq P_jw_i$) να ισχύει επιπλέον $P_iw_j+\sum_{h=1, h \neq i}^{n}w_h \max\limits_{k,l=1,...,B,\ k \neq l}(S_k+S_l) \leq P_jw_i$ κάτι που μπορεί να ισχύει μόνο για πολύ μικρές τιμές των S_k συγκριτικά με τα P_i (φαίνεται ας πούμε, εύκολα ότι η παραπάνω σχέση δεν μπορεί να ισχύει για κάθε ζεύγος εργασιών i, j με $P_i/w_i \leq P_j/w_j$ και $P_i=P_j=\max(S_k+S_l)$).

Μια τελευταία μέθοδος υπολογισμού κάτω φράγματος που χρησιμοποιεί μια παραλλαγή της χαλάρωσης Lagrange εφαρμόζεται στις εργασίες [PoWa-83] και [HaPo-83] για τα προβλήματα $1//\sum w_iC_i | T_{max}=0$ και $1/r_i/\sum w_iC_i$ αντίστοιχα. Επειδή η ουσία της μεθόδου είναι κοινή στις δύο εργασίες θα αναφερθούμε μόνο στη δεύτερη έχοντας υπ'όψιν ότι μετατρέποντας κατάλληλα όσα γράφονται για περιορισμούς release dates σε περιορισμούς προθεσμιών τα παρακάτω ισχύουν για τη πρώτη εργασία. Κατ'αρχήν παρουσιάζονται ένας ευρηματικός κανόνας εύρεσης μιας εφικτής λύσης και μια ικανή συνθήκη για να είναι μια εφικτή λύση βέλτιστη. Με βάση τον ευρηματικό κανόνα υπολογίζεται μια εφικτή λύση. Μετά εκτελείται κάτι σαν χαλάρωση Lagrange : οι περιορισμοί των release dates, πολλαπλασιασμένοι επί κάποιους μη αρνητικούς πολλαπλασιαστές λ_i, προστίθενται στην αντικειμενική συνάρτηση, ενώ ταυτόχρονα παραμένουν και σαν περιορισμοί στο πρόβλημα, δίνοντας το κάτω φράγμα : $L(\lambda)=\min_C\left\{\sum_{i=1}^{n} w_i C_i + \sum_{i=1}^{n}\lambda_i(r_i+P_i-C_i)\right\}=\min_C\left\{\sum_{i=1}^{n}(w_i-\lambda_i)C_i\right\}+\sum_{i=1}^{n}\lambda_i(r_i+P_i)$ δεδομένου ότι $r_i+P_i\le C_i (i=1,2,...,n)$ και φυσικά περιορισμοί χωρητικότητας μηχανής. Επειτα υπολογίζεται, σε γραμμικό χρόνο, διάνυσμα πολλαπλασιαστών λ για το οποίο η λύση του παραπάνω προβλήματος δίνεται από την διάταξη που έχει υπολογίσει ο ευρηματικός κανόνας όταν εφαρμόστηκε στο αρχικό πρόβλημα. Το λ που υπολογίζεται, μάλιστα, δίνει την μεγαλύτερη τιμή στο κάτω φράγμα L(λ) από όσα διανύσματα πολλαπλασιαστών έχουν την παραπάνω ιδιότητα. Με τον τρόπο αυτό υπολογίζεται ένα κάτω φράγμα (το L(λ)).

Στο δικό μας πρόβλημα δεν μπορεί να εφαρμοστεί η μέθοδος αυτή, γιατί πρώτον δεν μπορούμε να βρούμε εφικτή λύση σε πολυωνυμικό χρόνο και δεύτερον ακόμα κι αν είχαμε μια εφικτή λύση δεν έχουμε κάποια ικανή συνθήκη βελτίστου. Η συγκεκριμένη εργασία ([HaPo-83]) παρουσιάζει και ένα βελτιωμένο κάτω φράγμα (που βασίζεται στο ότι ξαναγραφεται με διαφορετικό τρόπο το L(λ) όπου εμφανίζονται κάποιες ποσότητες $b_j^{(n)}$ που είναι κάτω φράγματα των $\sum_{i\in S_j^{(n)}} C_i$ (όπου $S_j^{(n)}$ κάποια υποσύνολα του συνόλου των εργασιών) και αντικαθίστανται τα $b_j^{(n)}$ με καλύτερα κάτω φράγματα των $\sum_{i\in S_j^{(n)}} C_i$, βελτιώνοντας έτσι το συνολικό κάτω φράγμα, που αποδεικύεται ότι παραμένει κάτω φράγμα), που όμως δε μπορεί να εφαρμοστεί στην περίπτωσή μας γιατί απαιτεί υπολογισμό του προηγούμενου κάτω φράγματος (συγκεκριμένα της εφικτής διάταξης και του διανύσματος πολλαπλασιαστών λ που χρησιμοποιήθηκαν εκεί).

Η τελευταία μέθοδος εξαγωγής κάτω φράγματος που θα εξετάσουμε εδώ λέγεται χαλάρωση του χώρου καταστάσεων του Δυναμικού Προγραμματισμού και εφαρμόζεται στην εργασία [AbPo-88] για το $1//\sum w_i T_i$. Η βασική ιδέα είναι ότι πραγματοποιείται μια ομαδοποίηση καταστάσεων, σε μία τυποποίηση του προβλήματος σε μορφή Δυναμικού Προγραμματισμού, δημιουργώντας ένα νέο χαλαρωμένο πρόβλημα Δυναμικού Προγραμματισμού με ψευδοπολυωνυμική πλοκή και απαίτηση μνήμης (αντί της εκθετικής που έχει το αρχικό) του οποίου η βέλτιστη λύση αποτελεί κάτω φράγμα της βέλτιστης λύσης του αρχικού. Ενα εξυπηρετικό, για τη πραγματοποίηση της παραπάνω σκέψης, σχήμα Δυναμικού Προγραμματισμού είναι εκείνο σε κάθε στάδιο του οποίου αντιστοιχούν ημιτελή προγράμματα με συγκεκριμένο αριθμό εργασιών, σε κάθε κατάσταση αντιστοιχεί το ημιτελές πρόγραμμα με το μικρότερο κόστος από όσα περιέχουν τις εργασίες ενός συγκεκριμένου υποσυνόλου R, και έχουν στην τελευταία θέση μια συγκεκριμένη εργασία j, το κόστος κάθε κατάστασης (η τιμή της αντικειμενικής συνάρτησης για το ημιτελές πρόγραμμα) δίνεται από την αναδρομική σχέση $f(R, j) = \min_{i\in R-\{j\}}\{f(R-\{j\},i)\}+w_j\max\{\sum_{k\in R}P_k-d_j,0\}$ και η βέλτιστη τιμή της αντικειμενικής συνάρτησης είναι $\min_{j\in S} f(S, j)$ (S είναι το σύνολο των εργασιών) ενώ οι αρχικές τιμές είναι $f(\varnothing, j) = 0$ για κάθε εργασία j. Αντιστοιχίζοντας τώρα σε κάθε R το $t=\sum_{i\in R} P_i$ παίρνουμε νέες καταστάσεις, αντικαθιστώντας στις αρχικές τη μεταβλητή κατάστασης R με την t οι οποίες είναι λιγότερες (ψευδοπολυωνυμικές στον αριθμό, αφού το t παίρνει τιμές μέχρι n $\overline{P}$) από τις αρχικές. Λύνοντας τώρα το αντίστοιχο πρόβλημα Δυναμικού Προγραμματισμού στον αντίστοιχο χώρο καταστάσεων παίρνουμε ένα κάτω φράγμα στο αρχικό πρόβλημα, γιατί κάθε μονοπάτι στον αρχικό χώρο καταστάσεων αντιστοιχίζεται σε κάποιο μονοπάτι στο νέο χώρο καταστάσεων χωρίς όμως να ισχύει και το αντίστροφο, γιατί τα μονοπάτια του νέου χώρου καταστάσεων αντιστοιχούν και σε μη εφικτές διατάξεις εργασιών που περιέχουν κάποιες εργασίες περισσότερες από μια φορά και δεν περιέχουν καθόλου άλλες. Συνδέοντας κάθε εργασία με ένα επιπρόσθετο κόστος λ_i, με στόχο την αποθάρρυνση χρησιμοποίησης των ίδιων εργασιών πολλές φορές, το κάτω φράγμα δίνεται από $\min_{j\in S}\{f'(P,j;\lambda)\}-\sum_{i=1}^{n}\lambda_i$ με $f'(t,j;\lambda)=\min_{i\in S-\{j\}}\{f'(t-P_j,i;\lambda)\}+w_j\max\{t-d_j,0\}+\lambda_j$ ($P = \sum_{i=1}^{n} P_i=n\overline{P}$). Το διάνυσμα λ υπολογίζεται με subgradient optimization, κάθε επανάληψη του οποίου έχει πλοκή O(P n) (γιατί για κάθε χρονική στιγμή t αρκεί να αποθηκεύονται οι δύο μικρότερες τιμές της f′), δηλαδή $O(n^2\overline{P})$.

Στο $1/S_j/\sum C_i|T_{max}=0$ τώρα, ας ορίσουμε κατ'αρχήν, $P_{min}=\sum_{i=1}^{n}P_i+\sum_{k=1}^{b}S_k$, $P_1=P_{min}+\max_{1\le i\le n}\{P_i+S_i\}$ και $P_{max}=\sum_{i=1}^{n}(P_i+S_i)$. Υπάρχουν δύο τρόποι να χρησιμοποιήσουμε την παραπάνω ιδέα. Ο ένας είναι να πάρουμε το κάτω φράγμα σαν

$$\min_{\substack{j\in S\\ P_{min}\le t\le \min\{P_1,P_{max}\}}}\{f'(t,j;\lambda)\}-\sum_{i=1}^{n}\lambda_i, \text{ με}$$

$$f'(t,j;\lambda)=\begin{cases}\min_{i\in S-\{j\}}\{f'(t-P_j-S_{ij},i;\lambda)\}+t+\lambda_i, & d_i\ge t\\ +\infty, & d_j<t\end{cases}$$

(κανονικά θα έπρεπε να πάρει κανείς το min για $P_{min}\le t\le P_{max}$, αλλά αυτό θα βρίσκεται σε κάποιο $t\le P_1$, οπότε μπορεί να σταματήσει εκεί όταν $P_1<P_{max}$).Ο αριθμός των διαφορετικών τιμών που παίρνει η μεταβλητή κατάστασης t είναι μικρότερος από P_{max}, δηλαδή από $n\cdot\overline{P+S}$, οπότε ο αριθμός των καταστάσεων είναι μικρότερος από $n\cdot\overline{P+S}\cdot n$ και η τιμή κάθε κατάστασης υπολογίζεται σε O(b) (για κάθε χρονική στιγμή αρκεί να αποθηκεύονται οι δύο μικρότερες τιμές της f′ για κάθε ομάδα), οπότε η πλοκή του αλγορίθμου αυτού είναι $O(bn^2\overline{P+S})$. Ο άλλος τρόπος προκύπτει από τον πρώτο αν προσθέσουμε σε κάθε κατάσταση μια τρίτη μεταβλητή k, που να δείχνει πόσες εργασίες έχουν τοποθετηθεί μέχρι εκείνη τη στιγμή και το κάτω φράγμα δίνεται από

$$\min_{\substack{j\in S\\ P_{min}\le t\le P_{max}}}\{f'(t,j,n;\lambda)\}-\sum_{i=1}^{n}\lambda_i .$$

Ετσι κερδίζουμε σε ποιότητα (δηλαδή το κάτω φράγμα αυτό είναι καλύτερο από το προηγούμενο) γιατί τώρα υπάρχει ο περιορισμός να περιέχει η διάταξη n εργασίες (κάτι που δεν ήταν απαραίτητο πριν) και χάνουμε σε ταχύτητα υπολογισμού του φράγματος αφού η υπολογιστική πλοκή γίνεται τώρα $O(bn^3\overline{P+S})$.

Το βασικό μειονέκτημα της μεθόδου αυτής είναι η μεγάλη της υπολογιστική πλοκή. Σε κάθε κόμβο του δέντρου έρευνας απαιτείται η εκτέλεση μερικών επαναλήψεων του subgradient optimization καθεμια από τις οποίες έχει πλοκή μια από τις δύο που αναφέραμε, ανάλογα με την παραλλαγή του επιλέγεται. Επίσης, η άγνοια του C_{max} της βέλτιστης λύσης, έχει σαν αποτέλεσμα να παίρνεται το κάτω φράγμα σαν η μικρότερη τιμή της f' σε ευρύτερη περιοχή απ'ότι στο πρόβλημα που αρχικά εφαρμόστηκε η μέθοδος αυτή και επομένως το κάτω φράγμα να είναι χαλαρώτερο από το αντίστοιχο φράγμα της προαναφερθείσας εργασίας. Ακόμα όμως και για το πρόβλημα που πραγματεύεται εκείνη, η παραπάνω μέθοδος έχει λιγότερο καλά αποτελέσματα από άλλες μεθόδους που περιγράφηκαν παραπάνω (και από τον Δυναμικό Προγραμματισμό), όπως αναφέρεται στην εργασία [APW-90], κάτι που αποδίδεται στην μεγάλη υπολογιστική πλοκή της.

3.3. Μέθοδοι προσεγγιστικής επίλυσης

Τελειώνοντας το κεφάλαιο αυτό θα αναφέρουμε δύο αλγόριθμους υπολογισμού προσεγγιστικών λύσεων, που εφαρμόστηκαν σε ισάριθμα προβλήματα Χρονικού Προγραμματισμού.

Ο πρώτος [Burn-76] είναι αρκετά απλός και χρησιμοποιήθηκε για την εύρεση τοπικού ελαχίστου (διάταξης, δηλαδή, καλύτερης από κάθε άλλη που προκύπτει από αντιμετάθεση δύο εργασιών της) στο $1//\sum w_iC_i|T_{max}=0$. Ο αλγόριθμος ξεκινάει από μια εφικτή διάταξη και για κάθε ζεύγος εργασιών ελέγχει εάν η αντιμετάθεση των δύο μελών του θα δημιουργήσει μια εφικτή διάταξη με μικρότερη τιμή του $\sum w_iC_i$ από την προηγούμενη, πράγμα που, όταν συμβαίνει, οδηγεί στην πραγματοποίηση της αντιμετάθεσης αυτής και στο ξεκίνημα του αλγορίθμου από την αρχή. Ο αλγόριθμος αυτός δεν μπορεί να εφαρμοστεί όταν υπάρχουν χρόνοι εξάρμωσης γιατί δεν μπορούμε (σε πολυωνυμικό χρόνο) να βρούμε εφικτή λύση για το ξεκίνημα.

Ο δεύτερος αλγόριθμος [Posn-86] είναι ένας ευρηματικός κανόνας για το $1/r_1/w_iC_i$. Αρχικά παρουσιάζεται ένας αλγόριθμος που λύνει την ειδική μορφή του προβλήματος όταν υπάρχουν clusters με την εξής έννοια : Οι εργασίες χωρίζονται σε m clusters K_k ($k=1, 2, ..., m$) και κάθε εργασία του K_k προηγείται κάθε εργασίας του K_{k+1}. Για κάθε k ισχύει $\max_{j\in K_k}\{r_j\}\leq\min_{j\in K_k}\{r_j+P_j\}$, που σημαίνει ότι σε κάθε διάταξη εργασιών ενός cluster δεν υπάρχει χρόνος που η μηχανή μένει ανενεργή. Για κάθε cluster, k, βρίσκονται πολυωνυμικά οι αποδοτικές λύσεις στο δικριτηριακό $1/r_i/C_{max},\sum w_iC_i$ (αυτές είναι της μορφής: πρώτη μια οποιαδήποτε εργασία και ακολουθούν οι υπόλοιπες κατά SWPT - το σύνολο των αποδοτικών λύσεων, L_k, είναι υποσύνολο του συνόλου που περιέχει αυτές τις διατάξεις).

Εστω ότι έχει βρεθεί το σύνολο των αποδοτικών λύσεων του ίδιου δικριτηριακού προβλήματος, αλλά τώρα για όλες τις εργασίες μαζί των clusters 1, 2, ..., k-1, έστω M_{k-1}. Αποδεικνύεται σχετικά εύκολα, ότι το $|M_k|$ δε φράσσεται απλώς από το $|M_{k-1}|\,|L_k|$, όπως είναι φυσικό (αφού δεν είναι δυνατόν η διάταξη των εργασιών των $k-1$ πρώτων clusters να μην ανήκει στο M_{k-1} και η διάταξη των εργασιών του clusrter k να μην ανήκει στο L_k), αλλά και από το $|M_{k-1}| + |L_k|$ και ότι το M_k υπολογίζεται πολυωνυμικά. Τελικά το M_m, άρα και η λύση στο πρόβλημα, υπολογίζεται σε $O(n^2)$. Επειτα παρουσιάζεται ο βασιζόμενος στον παραπάνω αλγόριθμο ευρηματικός κανόνας για τη γενική περίπτωση που δεν υπάρχουν clusters (που είναι NP-complete). Με κάποιο άλλο εύρημα δημιουργείται μια εφικτή λύση η οποία διαμερίζεται σε όσο το δυνατό μεγαλύτερα ψευδο-clusters (συνεχόμενων εργασιών) με κριτήριο σε κάθε τέτοιο να ισχύει το $\max\{r_j\}\leq\min\{r_j+P_j\}$ και λύνεται το πρόβλημα που προκύπτει εάν τα ψευδο-clusters ήταν πραγματικά clusters.

Αυτά δεν μπορούν να εφαρμοστούν στο δικό μας πρόβλημα γιατί 1) δεν μπορούμε να βρούμε σε πολυωνυμικό χρόνο, εφικτή λύση που είναι αναγκαία στον ευρηματικό αλγόριθμο, 2) ακόμα κι αν έχουμε μία εφικτή λύση δεν ξέρουμε πως να την χωρίσουμε σε ψευδο-clusters (όπου ένας αντίστοιχος ορισμός για το cluster στη δική μας περίπτωση θα ήταν ένα σύνολο εργασιών, διάφορων ομάδων, που πρέπει να εκτελεστεί αδιάσπαστο και κάθε διάταξη των εργασιών του, μετά τα προηγούμενα clusters και πριν τα επόμενα, είναι εφικτή), 3) κι αν ορίζαμε ψευδο-clusters δε θα μπορούσαμε να λύσουμε πολυωνυμικά το $1/S_{ij}/T_{max},\sum C_i$ και 4) κι αν ακόμα μπορούσαμε να λύσουμε το τελευταίο πολυωνυμικά για κάθε ψευδο-cluster, το $|M_k|$ δε θα φρασσόταν από το $|M_{k-1}| + |L_k|$, οποτε η υπολογιστική πλοκή για την εύρεση του M_m θα έπαυε να είναι πολυωνυμική.

4. Ο Δυναμικός Προγραμματισμός στη λύση προβλημάτων Χρονικού Προγραμματισμού

Στα NP-complete συνδυαστικά προβλήματα είναι συχνά απαραίτητη η απαρίθμηση όλων των στοιχείων του συνόλου των εφικτών (και όχι μόνο) λύσεων. Επειδή αυτό είναι εν γένει πολύ χρονοβόρο, είναι επιθυμητό να χρησιμοποιείται μια τεχνική έμμεσης (ή ελεγχόμενης) απαρίθμησης η οποία πετυχαίνει να απορρίπτει, ως μή περιέχουσα βέλτιστη λύση, μια περιοχή του χώρου έρευνας χωρίς να εξετάζει ένα-ένα τα στοιχεία της. Οι κλασσικότερες μέθοδοι απαρίθμησης είναι ο διαμερισμός και φραγή και ο Δυναμικός Προγραμματισμός.

Για πρώτη φορά προτάθηκε η χρήση Δυναμικού Προγραμματισμού για επίλυση προβλημάτων Χρονικού Προγραμματισμού από τους Held και Karp [HeKa-62] σε ένα γενικό πλαίσιο. Από τότε έχει χρησιμοποιηθεί σε αρκετές εργασίες που πραγματεύονται τέτοια προβλήματα. Μερικές από αυτές (π.χ. [Sahn-76], [Psar-80], [MoPo-89], [AhHy-90], [CCMM-92]) δίνουν απλώς ένα σχήμα Δυναμικού Προγραμματισμού που λύνει κάποιο πρόβλημα Χρονικού Προγραμματισμού χωρίς να αναφέρουν λεπτομέρειες για την υλοποίηση του αλγορίθμου τους. Στο κεφάλαιο αυτό θα ασχοληθούμε με τις εργασίες εκείνες που πρότειναν συγκεκριμένους τρόπους υλοποίησης αλγορίθμων Δυναμικού Προγραμματισμού για προβλήματα Χρονικού Προγραμματισμού. Μερικές από τις ιδέες που θα αναφέρουμε χρησιμοποιούνται και στη δική μας υλοποίηση. Ολες οι εργασίες που θα αναφέρουμε δεν ασχολούνται με το ίδιο πρόβλημα. Αλλες ασχολούνται με ένα πρόβλημα Χρονικού Προγραμματισμού που λέγεται εξισορρόπηση γραμμής παραγωγής (assembly line balancing) και άλλες με το $1//\sum G_i$ (όπου G μια οποιαδήποτε αύξουσα συνάρτηση κόστους) ή με την ειδική του μορφή $1//\sum w_iT_i$. Το πρώτο από τα προβλήματα αυτά δεν σχετίζεται με το πρόβλημα που απασχολεί εμάς, γι'αυτό και δεν θα ασχοληθούμε μαζί του. Θα ασχοληθούμε όμως με τις ιδέες για την υλοποίηση του Δυναμικού Προγραμματισμού που χρησιμοποιήθηκαν στην επίλυσή του, που χωρίς καμιά αλλαγή, μπορούν να χρησιμοποιηθούν και στο $1//\sum G_i$.

4.1. Το γενικό σχήμα

Το σχήμα Δυναμικού Προγραμματισμού που χρησιμοποιήθηκε για το τελευταίο κατάλληλα τροποποιημένο, όπως θα πούμε στο επόμενο κεφάλαιο, μπορεί να χρησιμοποιηθεί για το δικό μας πρόβλημα, γι'αυτό και το παρουσιάζουμε: υπάρχουν N+1 στάδια, καθένα από τα οποία αντιστοιχεί στα υποσύνολα του συνόλου των εργασιών με ένα συγκεκριμένο πληθάριθμο (από 0 έως N)· κάθε κατάσταση αντιστοιχεί σε ένα συγκεκριμένο υποσύνολο εργασιών για το οποίο αποθηκεύεται η μικρότερη τιμή που μπορεί να πάρει η αντικειμενική συνάρτηση εάν οι εργασίες του υποσυνόλου αυτού διαταχθούν κατάλληλα στις πρώτες θέσεις κάποιου προγράμματος, τιμή που δίνεται από την αναδρομική σχέση $f(\mathrm{R}) = \min\limits_{\mathrm{i}\in \mathrm{R}} \{ f(\mathrm{R} - \{\mathrm{i}\}) + g(\mathrm{i}, \sum\limits_{\mathrm{j}\in\mathrm{R}} \mathrm{P_j}) \}$, με $f(\varnothing) = 0$, όπου R είναι το υποσύνολο των εργασιών και $g(\mathrm{i}, \mathrm{t})$ το κόστος της εργασίας i εάν τελειώσει τη στιγμή t.

Το παραπάνω σχήμα έχει εκθετική υπολογιστική πλοκή, καθώς και εκθετική απαίτηση μνήμης, για την αποθήκευση των σχετικών με κάθε υποσύνολο R πληροφοριών (δηλαδή του *f*(R) και της εργασίας i για την οποία υπολογίστηκε η συγκεκριμένη τιμή της *f*). Ο αριθμός των απαιτουμένων πράξεων για την επίλυση του προβλήματος με το παραπάνω σχήμα καθώς και η απαιτούμενη μνήμη μειώνονται σημαντικά λαμβάνοντας υπ’όψιν σχέσεις προτεραιότητας μεταξύ των εργασιών που είτε δίνονται από την εκφώνηση του προβλήματος είτε προκύπτουν με κάποιους κανόνες (όπως π.χ. οι δύο κανόνες που αναφέραμε στο δεύτερο κεφάλαιο για το πρόβλημά μας). Βάσει των σχέσεων αυτών, κάποια υποσύνολα χαρακτηρίζονται εφικτά όταν υπάρχει διάταξη των εργασιών τους που μπορεί να τοποθετηθεί στην αρχή ενός προγράμματος χωρίς να παραβιάζονται αυτές οι σχέσεις, κάτι που είναι ισοδύναμο με το ότι όλοι οι πρόγονοι κάθε εργασίας του υποσυνόλου βρίσκονται στο υποσύνολο αυτό, ενώ τα υπόλοιπα υποσύνολα χαρακτηρίζονται μη εφικτά, που σημαίνει ότι ένας τουλάχιστον πρόγονος μιας τουλάχιστον εργασίας δεν ανήκει στο υποσύνολο αυτό. Χαρακτηρίζοντας κάποια υποσύνολα σαν μη εφικτά μπορούμε να μην ασχολούμαστε με αυτά κερδίζοντας έτσι και σε ταχύτητα αφού δεν είναι πλέον απαραίτητος ο υπολογισμός της συνάρτησης *f* γι’αυτά τα υποσύνολα (κάνοντας, βέβαια, τη ρεαλιστική υπόθεση ότι ο χρόνος που απαιτείται για την επιλογή των εφικτών συνόλων και την απόριψη των μη εφικτών είναι λιγότερος από αυτόν που απαιτείται για την επεξεργασία των μη εφικτών) αλλά και, κυρίως, σε μνήμη αφού δεν χρειάζεται να αποθηκεύουμε πλέον κάποια πληροφορία για τα μη εφικτά σύνολα. (Εαν χρησιμοποιούσε κανείς ένα μόνο byte για κάθε υποσύνολο - εφικτό ή μη - θα χρειαζόταν για ένα πρόβλημα 30 εργασιών 1 GByte μνήμης!). Τώρα, που ενδιαφερόμαστε μόνο για τα εφικτά σύνολα, το min στον αναδρομικό τύπο που δίνει την τιμή του *f*(R) δεν παίρνεται πάνω σε όλες τις εργασίες που ανήκουν στο R αλλά σε κείνες που επιπλέον δεν έχουν κάποιο απόγονό τους σ’ αυτό, οπότε μπορούν να τοποθετηθούν τελευταίες (μεταξύ των εργασιών του R) και το σύνολο που προκύπτει εάν αφαιρεθούν από το R είναι εφικτό. Το σύνολο των εργασιών αυτών (που είναι βέβαια υποσύνολο του R) θα το συμβολίζουμε με Q(R).

4.2. Οι αλγόριθμοι των Held-Karp-Shareshian

Οι Held, Karp και Shareshian είναι οι πρώτοι που μελέτησαν συγκεκριμένους τρόπους αξιοποίησης των σχέσεων προτεραιότητας και υλοποίησης του σχήματος Δυναμικού Προγραμματισμού προτείνοντας, στην εργασία [HKS-63], δύο αλγορίθμους. Ο πρώτος είναι αρκετά απλός με συγκρίσιμη όμως υπολογιστική πλοκή και απαίτηση μνήμης με επόμενους πιο πολύπλοκους: Τα εφικτά σύνολα εργασιών (καθένα από τα οποία αντιστοιχεί, όπως έχουμε πει, σε μία κατάσταση του σχήματος Δυναμικού Προγραμματισμού) δημιουργούνται με λεξικογραφική σειρά και οι πληροφορίες που συνδέονται με αυτά αποθηκεύονται σε διαδοχικές θέσεις μνήμης (σε κάποιο πίνακα δηλαδή). Η πληροφορία που αποθηκεύεται για κάθε σύνολο είναι το ποια είναι η τελευταία εργασία στη βέλτιστη διάταξη των εργασιών αυτού του συνόλου (κάτι που έστω ότι απαιτεί α_1 bytes), η τιμή της αντικειμενικής συνάρτησης στη βέλτιστη διάταξη (που έστω ότι απαιτεί α_2 bytes και θέτουμε $\alpha = \alpha_1 + \alpha_2$) και μια "ταυτότητα" του συνόλου (που έστω ότι απαιτεί β bytes).

Ο οικονομικότερος, από άποψη μνήμης και ταχύτητας τρόπος περιγραφής ενός συνόλου ("ταυτότητά" του δηλαδή), όπως επισημαίνεται σε εργασίες στις οποίες θα αναφερθούμε παρακάτω, είναι ένα διάνυσμα m με n στοιχεία τέτοιο ώστε $m(i) = 1$ εάν η εργασία i ανήκει στο σύνολο και 0 διαφορετικά. Το διάνυσμα αυτό λέγεται χαρακτηριστικό διάνυσμα του συνόλου. Για την αποθήκευση ενός τέτοιου διανύσματος χρειάζονται $\lceil n/w \rceil$ λέξεις (words) όπου w είναι ο αριθμός των bits ανά λέξη (και λέξη η μεγαλύτερη ποσότητα μνήμης που μπορεί να αποτελεί όρισμα ή αποτέλεσμα μιας πράξης και εξαρτάται κυρίως από τα κατασκευαστικά χαρακτηριστικά του υπολογιστή). Στο εξής θα θεωρούμε ότι η περιγραφή ενός συνόλου εργασιών γίνεται πάντα με τον παραπάνω τρόπο. Ξαναγυρνώντας στον αλγόριθμο που περιγράφουμε έχουμε ότι η απαιτούμενη μνήμη είναι M(α+β) bytes (για τον κύριο όγκο των δεδομένων) όπου M ο αριθμός των εφικτών συνόλων.

Στην εργασία αυτή δεν περιγράφεται κάποιος αλγόριθμος δημιουργίας των εφικτών συνόλων με λεξικογραφική σειρά. Ενας τέτοιος αλγόριθμος δίνεται στην εργασία [ScBa-78], στην οποία θα αναφερθούμε παρακάτω και είναι ο εξής : Εστω m το χαρακτηριστικό διάνυσμα του εφικτού συνόλου R που υπολογίστηκε τελευταίο. Για το μικρότερο i με $m(i) = 0$ θέσε $m(i) = 1$. Από $j = i - 1$ μειώνοντας το j κατά 1 μέχρι $j = 1$ εάν η εργασία j ανήκει στο Q(R) (δηλαδή δεν ανήκει στο R κάποιος απόγονός της) διαγράφεται από το R (θέτοντας δηλαδή $m(i) = 0$). Η πλοκή του αλγορίθμου αυτού είναι $O(n^2)$.

Αφού δημιουργηθεί ένα νέο εφικτό σύνολο R (υπολογισθεί δηλαδή το χαρακτηριστικό του διάνυσμα), για κάθε εργασία $i \in Q(R)$ γίνεται μία δυαδική αναζήτηση στον πίνακα που φυλάγονται οι πληροφορίες για τα εφικτά σύνολα που μέχρι τώρα έχουν δημιουργηθεί και βρίσκεται η θέση που αντιστοιχεί στο $R - \{i\}$. Η αναζήτηση μπορεί να γίνει δυαδικά γιατί τα σύνολα είναι αποθηκευμένα με λεξικογραφική σειρά (δηλαδή με αύξουσα σειρά των χαρακτηριστικών τους διανυσμάτων). Εχοντας βρει λοιπόν, όλα τα $R - \{i\}$ ($i \in Q(R)$) και επομένως τις τιμές $f(R - \{i\})$ μπορεί να υπολογιστεί η f(R) από τον αναδρομικό τύπο του Δυναμικού Προγραμματισμού. Η ανεύρεση μέσα στον πίνακα, ενός συνόλου $R - \{i\}$ έχει πλοκή $O(n^2)$ γιατί γίνεται μια δυαδική αναζήτηση σ'έναν πίνακα με $O(2^n)$ στοιχεία και κάθε έλεγχος εάν το χαρακτηριστικό διάνυσμα κάποιου συνόλου που αντιστοιχεί σε μια θέση του πίνακα είναι μικρότερο ή μεγαλύτερο από το χαρακτηριστικό διάνυσμα του συνόλου που ψάχνουμε γίνεται σε O(n). Επειδή λοιπόν για τον υπολογισμό της τιμής f(R) θα χρειαστούν O(n) τέτοιες αναζητήσεις, η πλοκή του υπολογισμού αυτού είναι $O(n^3)$. Επομένως ο αριθμός των πράξεων που σχετίζονται με ένα εφικτό σύνολο R είναι $O(n^2 + n^3) = O(n^3)$ (ο πρώτος όρος είναι για τη δημιουργία του και ο δεύτερος για τον υπολογισμό της τιμής του f(R)) και η πλοκή του αλγορίθμου είναι $O(n^3 \cdot M)$.

Στην εργασία αυτή προτείνουν πριν ξεκινήσει ο παραπάνω αλγόριθμος, να καλείται ένας άλλος με πλοκή $O(n^2 \cdot M)$ που θα μετράει τα εφικτα σύνολα, ώστε να δεσμεύεται όση ακριβώς μνήμη θα χρειαστεί για τον πίνακα. Αυτό όμως, που δεν αυξάνει βέβαια τη συνολική πλοκή, δεν είναι απαραίτητο αφού δεν χάνει κανείς τίποτα αν δεσμεύσει το μεγαλύτερο ποσό μνήμης που του επιτρέπει το λειτουργικό σύστημα.

Είπαμε ότι στην εργασία αυτή παρουσιάζονται δύο αλγόριθμοι Δυναμικού Προγραμματισμού. Ο δεύτερος έχει σα στόχο τη μείωση της απαιτούμενης μνήμης. Οι συγγραφείς παρουσιάζουν έναν εκθετικό αλγόριθμο που σε κάθε εφικτό σύνολο αντιστοιχεί έναν ακέραιο αριθμό από 1 έως M που χρησιμοποιείται ως διεύθυνση του συνόλου αυτού στον πίνακα αποθήκευσης του συνόλου. Με τον τρόπο αυτό δεν είναι απαραίτητη η αποθήκευση των β bytes για το χαρακτηριστικό διάνυσμα κάθε συνόλου, γιατί κάθε φορά που για κάποιο σύνολο R αναζητούμε τα $f(R - \{i\})$, για $i \in Q(R)$, μπορούμε να υπολογίσουμε με τον εκθετικό αλγόριθμο τις διευθύνσεις που αυτά είναι αποθηκευμένα. Ετσι η απαιτούμενη μνήμη είναι $M \cdot \alpha$ bytes. Η υπολογιστική πλοκή είναι όμως πολύ μεγαλύτερη από τον προηγούμενο αλγόριθμο γιατί τώρα για καθένα από τα M εφικτά σύνολα και καθεμιά από τις O(n) εργασίες του που μπορούν να τοποθετηθούν τελευταίες (που ανήκουν δηλαδή στο Q(R)) καλείται ένας εκθετικός αλγόριθμος, οπότε η συνολική πλοκή όλου του αλγορίθμου είναι $O(n \cdot M \cdot 2^n)$.

4.3. Οι αλγόριθμοι των Schrage-Baker

Μια διαφορετική τεχνική για τον υπολογισμό της διεύθυνσης ενός εφικτού συνόλου, που απαιτεί περισσότερη μνήμη αλλά έχει μικρότερη υπολογιστική πλοκή, χρησιμοποιείται από τους Schrage και Baker στις εργασίες [BaSc-78] και [ScBa-78]. Εδώ αντιστοιχίζεται ένα label (ένας ακέραιος αριθμός) σε κάθε εργασία έτσι ώστε το άθροισμα των labels των εργασιών ενός συνόλου R, που αποτελεί τη διεύθυνσή του, κι ας το συμβολίζουμε με A(R), να είναι μοναδικό. Τα εφικτά σύνολα δημιουργούνται, υπολογίζεται δηλαδή το χαρακτηριστικό τους διάνυσμα, πάλι με λεξικογραφική σειρά (κάτι που όπως έχουμε πει έχει πλοκή $O(n^2)$ για κάθε εφικτό σύνολο), μόνο που τώρα η διεύθυνση κάθε συνόλου της μορφής $(R - \{i\})$ για $i \in Q(R)$ βρίσκεται σε σταθερό χρόνο αφαιρώντας το label της εργασίας i από τη διεύθυνση του συνόλου R. Ετσι η πλοκή των αλγορίθμων αυτών είναι $O(n^2 \cdot M)$, αφού ο υπολογισμός των labels γίνεται όπως θα πούμε παρακάτω σε πολυωνυμικό χρόνο. Μπορεί ο καθορισμός των labels να γίνεται με τέτοιο τρόπο ώστε το κάθε εφικτό σύνολο να έχει ξεχωριστή διεύθυνση, όμως δεν αποκλείεται (όπως και πράγματι γίνεται), να υπάρχουν διευθύνσεις που δεν αντιστοιχούν σε κανένα εφικτό σύνολο. Επομένως ο πίνακας όπου θα φυλάγονται οι πληροφορίες για τα εφικτά σύνολα δε θα έχει μόνο M θέσεις αλλά LSUM όπου με LSUM θα συμβολίζουμε το άθροισμα των labels όλων των εργασιών, που είναι η διεύθυνση του συνόλου όλων των εργασιών. Για κάθε σύνολο δεν χρειάζεται να αποθηκεύεται το χαρακτηριστικό του διάνυσμα οπότε η μνήμη που απαιτούν αυτοί οι αλγόριθμοι είναι LSUM·α bytes. Να παρατηρήσουμε ότι η ποσότητα της απαιτούμενης μνήμης εξαρτάται από τα labels που αντιστοιχούν στις εργασίες.

Ο αλγόριθμος που χρησιμοποιείται στην πρώτη από τις δύο εργασίες ([BaSc-78]) για την τοποθέτηση labels στις εργασίες είναι ο εξής : έστω ο κατευθυνόμενος γράφος με κόμβους τις εργασίες και ακμές τις άμεσες σχέσεις προτεραιότητας μεταξύ τους (δηλαδή τις σχέσεις $i \rightarrow j$ εκείνες - συμβολίζοντας με $\alpha \rightarrow \beta$ το γεγονός ότι η α είναι πρόγονος της β - για τις οποίες δεν υπάρχει εργασία k με $i \rightarrow k$ και $k \rightarrow j$). Ο γράφος διαμερίζεται σε αλυσίδες εργασιών που παίρνουν το ίδιο label ίσο με 1 + {το άθροισμα των labels των εργασιών που έχουν ήδη πάρει label εκτός από αυτές που είναι πρόγονοι της πρώτης εργασίας της αλυσίδας ή απόγονοι της τελευταίας}. Αποδεικνύεται ότι τα labels που δίνονται στις εργασίες με τον πολυωνυμικό αυτό τρόπο τηρούν την απαραίτητη συνθήκη να είναι μοναδική η διεύθυνση κάθε εφικτού συνόλου. Τα labels που θα δοθούν, άρα και το άθροισμά τους - άρα και η απαιτούμενη μνήμη -, εξαρτώνται από τη διαμέριση του γράφου σε αλυσίδες και τη σειρά με την οποία αυτές (οι εργασίες τους δηλαδή) θα πάρουν labels.

Στη δεύτερη εργασία ([ScBa-78]) χρησιμοποιείται ένας γενικότερος αλγόριθμος μέ την έννοια ότι η ανάθεση ενός label σε μια εργασία επηρεάζεται από ένα υπερσύνολο των σχέσεων προτεραιότητας που επηρεάζουν την αντίστοιχη ανάθεση στον προηγούμενο αλγόριθμο: με μια οποιαδήποτε σειρά οι εργασίες παίρνουν μία-μία ως label το 1 + {άθροισμα των labels των εργασιών που έχουν ήδη πάρει label εκτός από αυτές που είναι πρόγονοι ή απόγονοί τους}. Ο αλγόριθμος αυτός έχει πλοκή $O(n^2)$. Αποδεικνύεται και γι' αυτόν ότι τα labels που αναθέτει στις εργασίες είναι τέτοια ώστε να μην υπάρχουν δύο εφικτά σύνολα με την ίδια διεύθυνση.

Το άθροισμα των labels, οπότε και η απαιτούμενη μνήμη, που είναι μικρότερο από το αντίστοιχο άθροισμα με τον προηγούμενο αλγόριθμο (όπως αποδεικνύεται και πειραματικά στην [ScBa-78]) εξαρτάται από την σειρά με την οποία ανατίθενται labels σε εργασίες. Η εύρεση της βέλτιστης σειράς, αυτή δηλαδή που ελαχιστοποιεί το LSUM, είναι από μόνη της ένα δύσκολο συνδυαστικό πρόβλημα όπως παρατηρείται στην εργασία [KaQu-82]. Οι Schrage-Baker προτείνουν κάθε φορά να παίρνει label η εργασία που αν πάρει εκείνη τη στιγμή θα πάρει το μικρότερο.

Οι Burns και Steiner [BuSt-81] παρουσιάζουν μια παραλλαγμένη μορφή του παραπάνω αλγορίθμου που όταν ο γράφος προτεραιοτήτων έχει μια ειδική μορφή πετυχαίνει LSUM = M δηλαδή δεν υπάρχει διεύθυνση στην οποία δεν αντιστοιχεί κάποιο εφικτό σύνολο. Η συνθήκη που πρέπει να πληρούν οι σχέσεις προτεραιότητας για να βρίσκει εφαρμογή ο αλγόριθμος των Burns-Steiner είναι ότι δύο οποιεσδήποτε εργασίες θα πρέπει ή να μην έχουν κανένα κοινό απόγονο ή να έχουν ακριβώς τους ίδιους. Αυτή η συνθήκη όμως δεν ισχύει ούτε στα δύο προβλήματα με τα οποία ασχολούνται οι εργασίες που αναφέρουμε, ούτε κυρίως στο δικό μας πρόβλημα, οπότε αυτή η παραλλαγή δε θα μας απασχολήσει περισσότερο.

4.4. Ο αλγόριθμος των Kao-Queyranne

Οι Kao και Queyranne [KaQu-82] προτείνουν έναν αλγόριθμο με μικρότερη απαίτηση μνήμης από τον προηγούμενο, των Schrage-Baker, του οποίου και αποτελεί ουσιαστικά μία παραλλαγή. Παρατηρούν ότι εάν τα εφικτά σύνολα δημιουργούνται με αύξουσα σειρά των διευθύνσεων τους και αν υπάρχει τρόπος, δοθείσης μιας διεύθυνσης, να βρίσκεται το εφικτό σύνολο (εάν υπάρχει κάποιο) που αντιστοιχεί σ'αυτήν, τότε αρκεί η αποθήκευση LMAX μόνο εφικτών συνόλων, όπου LMAX είναι το μέγιστο label που ανατίθεται σε κάποια εργασία. Αυτό ισχύει γιατί η διεύθυνση κάποιου συνόλου δε μπορεί να διαφέρει περισσότερο από LMAX από τις διευθύνσεις των συνόλων που προκύπτουν από αυτό εάν αφαιρεθεί κάποια εργασία. Επομένως, για την αποθήκευση της σχετικής με τα εφικτά σύνολα πληροφορίας μπορεί να χρησιμοποιηθεί κυκλικά ένας πίνακας με LMAX θέσεις (αντί των LSUM που απαιτεί ο αλγόριθμος των Schrage-Baker) αποθηκεύοντας την πληροφορία του κάθε συνόλου R στη θέση A(R) modulo LMAX.

Οι εργασίες παίρνουν labels με τον αλγόριθμο των Schrage-Baker, αλλά υπάρχει και πάλι το ερώτημα με ποια σειρά των εργασιών πρέπει να γίνεται αυτό ώστε να ελαχιστοποιηθεί τώρα το LMAX. Προτείνεται τώρα ευρηματικός κανόνας που θυμίζει τον αλγόριθμο τοποθέτησης labels της εργασίας [BaSc-78]: Στον γράφο των προτεραιοτήτων εντοπίζεται και απομακρύνεται η μεγαλύτερη αλυσίδα εργασιών, στον εναπομείναντα γράφο εντοπίζεται και απομακρύνεται η αλυσίδα που είναι τώρα η μεγαλύτερη κ.ο.κ μέχρι να απομακρυνθούν όλες οι εργασίες. Η σειρά με την οποία τροφοδοτούνται οι εργασίες στον αλγόριθμο τοποθέτησης labels των Schrage-Baker είναι η αντίστροφη από εκείνη με την οποία απομακρύνθηκαν από το γράφο. Αποδεικνύεται πειραματικά, και στην εργασία που αναφερόμαστε και στην [Stei-90], ότι η διάταξη των εργασιών που χρησιμοποιούν οι Schrage-Baker για να τους δώσουν labels δίνει μικροτερο LSUM και αυτή των Kao-Queyranne μικρότερο LMAX.

Επειδή η λεξικογραφική διάταξη των συνόλων δεν είναι, γενικά, η ίδια με την κατά αύξουσα σειρά των διευθύνσεων διάταξη, δεν μπορούν τα σύνολα να απαριθμούνται με λεξικογραφική σειρά, όπως γινόταν στην [ScBa-78]. Γι'αυτό και χρησιμοποιείται ένας άλλος αλγόριθμος δημιουργίας των εφικτών συνόλων: για κάθε διεύθυνση από 1 έως LSUM υπολογίζεται με κάποιο τρόπο, σε O(n), το εφικτό σύνολο που αντιστοιχεί σ'αυτήν, αν υπάρχει τέτοιο. Ετσι τα εφικτά σύνολα δημιουργούνται με αύξουσα σειρά των διευθύνσεών τους, κάτι που όπως είπαμε είναι απαραίτητο για να αρκεί η αποθήκευση μόνο LMAX συνόλων. Η πλοκή του αλγορίθμου αυτού για κάποια διεύθυνση από 1 έως LSUM είναι O(n) (για την εύρεση του συνόλου που αντιστοιχεί στην συγκεκριμένη διεύθυνση) + $O(n^2)$ (για τον έλεγχο εφικτότητας του συνόλου αυτού) + O(n) εάν το σύνολο αυτό είναι εφικτό (για τον υπολογισμό της συνάρτησης *f* σε αυτό), άρα η πλοκή όλου του αλγορίθμου είναι $O(n^2 \cdot LSUM)$.

Ενα μειονέκτημα της μεθόδου των Kao-Queyranne, στο οποίο δε δίνεται προσοχή στην εργασία τους, είναι το γεγονός ότι εφ'όσον χρησιμοποιούνται οι ίδιες θέσεις του πίνακα κυκλικά για διαφορετικά σύνολα, δεν υπάρχει τρόπος, αφού φτάσει στο τέρμα ο Δυναμικός Προγραμματισμός και βρεθεί η βέλτιστη τιμή της συνάρτησης *f*, να ανακατασκευαστεί η βέλτιστη λύση. Στην εργασία [APW-90] αναφέρεται ότι αυτό μπορεί να γίνει χρησιμοποιώντας δευτερεύουσα μνήμη. Συγκεκριμένα, μπορούμε να αποθηκεύσουμε σε ένα αρχείο για κάθε εφικτό σύνολο τη διεύθυνσή του και το ποια είναι η εργασία που θα εκτελεστεί τελευταία στη βέλτιστη διάταξη των εργασιών του. Αντί της διεύθυνσης ενός συνόλου μπορεί να αποθηκεύεται το χαρακτηριστικό του διάνυσμα, κάτι όμως που από άποψη πλοκής δεν συμφέρει γιατί πρώτον οι συγκρίσεις, που απαιτούνται σε μία αναζήτηση, μεταξύ διευθύνσεων γίνονται σε O(1) αφού αυτές καταλαμβάνουν σαν ακέραιοι που είναι, μία λέξη μνήμης ενώ μεταξύ χαρακτηριστικών διανυσμάτων σε O(n) αφού αυτά καταλαμβάνουν $\lceil n / w \rceil$ λέξεις και δεύτερον αφού η δημιουργία και αποθήκευση στο αρχείο των συνόλων γίνεται με αύξουσα σειρά των διευθύνσεων, συμφέρει να κρατάμε αυτές σαν αναγνωριστικά των συνόλων, γιατί έτσι είναι αυτόματα ταξινομημένα και αυτό, όπως θα δούμε παρακάτω, ανάλογα με την υλοποίηση, μπορεί να οδηγήσει σε ταχύτερη αναζήτηση των συνόλων. Αφού βρεθεί η βέλτιστη τιμή της *f*, θα πρέπει το αρχείο οπου βρίσκεται η παραπάνω πληροφορία, να διαβαστεί n φορές, βρίσκοντας κάθε φορά ποια είναι η τελευταία από τις εναπομείνασες προς διάταξη εργασίες και, ισοδύναμα ποιο το εφικτό σύνολο που μένει να διαταχθεί. Η υπολογιστική πλοκή της ανακατασκευής με τον παραπάνω τρόπο της βέλτιστης λύσης είναι $O(n{\cdot}M)$, αφού διαβάζεται n φορές ένα αρχείο με M στοιχεία, οπότε η συνολική πλοκή παραμένει $O(n^2{\cdot}LSUM)$. Στη πράξη όμως, ο πραγματικός χρόνος εκτέλεσης που υλοποιεί τον παραπάνω αλγόριθμο θα είναι πολύ μεγαλύτερος όταν συμπεριλαμβάνει την ανακατασκευή της βέλτιστης λύσης, εφ'όσον είναι γνωστή η πολύ μεγάλη (σχετικά με τους χρόνους εκτέλεσης των εντολών) καθυστέρηση που συνεπάγεται μία πρόσβαση στο δίσκο και επομένως το γράψιμο και διάβασμα ενός αρχείου.

Μια βελτιωμένη έκδοση του παραπάνω τρόπου ανακατασκευής της βέλτιστης λύσης είναι να διαβάζεται μία φορά το παραπάνω αρχείο και να αποθηκεύονται τα στοιχεία του σε ένα πίνακα μεγέθους M ταξινομημένα, όπως είναι, κατά τη διεύθυνσή τους. Με αυτό το τρόπο διαβάζεται μία μόνο φορά το αρχείο και η αναζήτηση των συνόλων μπορεί να γίνεται δυαδικά και όχι σειριακά όπως θα γινόταν στο αρχείο (εκτός κι αν είχαν αποθηκευτεί οργανωμένα σε μία κατάλληλη δομή, όπως ένα B-δένδρο, κάτι που όμως θα έκανε πιο πολύπλοκη και χρονοβόρα την εγγραφή τους), οπότε το πρόγραμμα θα είναι ταχύτερο, χωρίς όμως και πάλι να έχουμε αποφύγει το γράψιμο και το διάβασμα ενός αρχείου.

Από άποψη μνήμης τώρα, ο πίνακας που χρησιμοποιείται κυκλικά για την αποθήκευση των ενδιαμέσων αποτελεσμάτων της αναδρομής του Δυναμικού Προγραμματισμού καταλαμβάνει $LMAX{\cdot}\alpha_2$ bytes (τώρα δεν υπάρχει λόγος να αποθηκεύεται το ποια είναι η τελευταία εργασία στη διάταξη κάθε συνόλου). Οι απαιτήσεις σε μνήμη του πρώτου από τους δύο παραπάνω τρόπους της ανακατασκευής της βέλτιστης λύσης είναι, ουσιαστικά, μηδαμινές. Αντίθετα, ο δεύτερος τρόπος απαιτεί ένα πίνακα με M θέσεις, σε κάθε θέση του οποίου θα αποθηκεύονται η διεύθυνση του αντίστοιχου συνόλου και η τελευταία εργασία της βέλτιστης διάταξής του, δηλαδή, αν πούμε ότι μία διεύθυνση αποθηκεύεται σε γ bytes, θα καταλαμβάνει $M(\alpha_1 + \gamma)$ bytes. Σε ένα πρόβλημα που δεν υπάρχει καμία σχέση προτεραιότητας ισχύει $M = LSUM = 2^n$ και $LMAX = 2^{n-1}$, και επειδή $\alpha_1 + \gamma$ είναι (σε μία τυπική υλοποίηση σε C) 1.5 φορά το α_2, η μνήμη που απαιτείται για την ανακατασκευή της βέλτιστης λύσης είναι μεγαλύτερη από τη μνήμη που απαιτείται για την υλοποίηση της αναδρομής του Δυναμικού Προγραμματισμού. Για τη γενικότερη όμως περίπτωση που υπάρχουν σχέσεις προτεραιότητας, ο Steiner [Stei-90], μετά από εκτεταμένες πειραματικές μελέτες για διάφορες τιμές της "πυκνότητας" του γράφου των προτεραιοτήτων, κατέληξε στο συμπέρασμα ότι μία μέση τιμή για το $LSUM / M$ είναι γύρω στο 12 ενώ για το $LSUM / LMAX$ γύρω στο 3 που δείχνει ότι το LMAX είναι, κατά μέσο όρο, περίπου τετραπλάσιο από το M. Επομένως η μνήμη που απαιτείται για την ανακατασκευή της βέλτιστης λύσης με το δεύτερο τρόπο μπορεί να θεωρηθεί γενικά μικρότερη από αυτή που απαιτείται για την αναδρομή του Δυναμικού Προγραμματισμού, οπότε η απαιτούμενη μνήμη της μεθόδου αυτής, που με ακρίβεια είναι $\max\{ LMAX \cdot \alpha_2, M(\alpha_1 + \gamma) \}$, θα είναι, συνήθως τουλάχιστον, $LMAX \cdot \alpha_2$ (όση δηλαδή και με τη πρώτη μέθοδο).

Ενας άλλος τρόπος αντιμετώπισης του προβλήματος της ανακατασκευής της βέλτιστης λύσης είναι να χρησιμοποιηθεί κατάλληλα η ιδέα των Munro και Ramirez [MuRa-82] που δεν απαιτεί χρησιμοποίηση δευτερεύουσας μνήμης. Αυτοί ασχολούνται γενικά με την εύρεση ελάχιστου μονοπατιού σε ένα διαστρωματομένο γράφο. Προτείνουν, όταν δεν αρκεί η μνήμη για αποθήκευση όλου του γράφου, οπότε όταν βρεθεί το μήκος του βέλτιστου μονοπατιού δεν μπορεί κανείς να γυρίσει πίσω ανακατασκευάζοντας το βέλτιστο μονοπάτι, να αποθηκεύεται σε κάθε κόμβο του γράφου από το μεσαίο επίπεδο και έπειτα το ποιος κόμβος του μεσαίου επιπέδου ανήκει στο βέλτιστο μονοπάτι κι έπειτα να λύνεται το πρόβλημα αναδρομικά στο πρώτο και δεύτερο μισό του γράφου. Αποδεικνύουν ότι στο πρόβλημα αυτό, η χρησιμοποίηση της μεθόδου τους οδηγεί σε πολλαπλασιασμό της πλοκής (από ότι δηλαδή θα ήταν εάν μπορούσε να αποθηκεύεται όλος ο γράφος) με ένα παράγοντα $O(\log n)$ εάν είναι n τα επίπεδα του γράφου.

Κάτι αντίστοιχο μπορεί να εφαρμοστεί στον αλγόριθμο των Kao-Queyranne για τα προβλήματα Χρονικού Προγραμματισμού. Συγκεκριμένα σε κάθε σύνολο που θα περιέχει τουλάχιστον τις μισές εργασίες μπορούμε να αποθηκεύσουμε την πληροφορία ποιες από τις εργασίες του βρίσκονται στις πρώτες $\lfloor n/2 \rfloor$ θέσεις της βέλτιστης διάταξης των εργασιών του συνόλου αυτού. Ετσι με το πέρας του Δυναμικού Προγραμματισμού, εκτός της τιμής της βέλτιστης λύσης, θα ξέρουμε ποιες εργασίες βρίσκονται, στη βέλτιστη διάταξη, πριν τη μέση και ποιες μετά και το πρόβλημα μπορεί να λυθεί αναδρομικά για τα δύο μισά ξεχωριστά. Εχουμε πει ότι η πλοκή του αλγορίθμου των Kao-Queyranne για n εργασίες είναι $O(n^2 \cdot LSUM)$ που επειδή το LSUM είναι της τάξης του 2^n, μπορεί να γραφεί και $O(n^2 \cdot 2^n)$. Επομένως η αναδρομική κλήση του αλγορίθμου των Kao-Queyranne, που περιγράφηκε παραπάνω, θα έχει πλοκή $O(n^2 \cdot 2^n + 2 \cdot (n/2)^2 \cdot 2^{(n/2)} + 4 \cdot (n/4)^2 \cdot 2^{(n/4)} + \ldots)$ κι επειδή ο πρώτος όρος είναι κατά πολύ μεγαλύτερος από το άθροισμα των υπόλοιπων, η πλοκή θα είναι πάλι $O(n^2\, 2^n)$ και πιο συγκεκριμένα $O(n^2 \cdot LSUM)$. Επειδή τώρα, δε γίνονται προσβάσεις σε δευτερεύουσα μνήμη ο πραγματικός χρόνος εκτέλεσης του προγράμματος που υλοποιεί τον αλγόριθμο αυτό αναμένεται αισθητά μικρότερος από ότι όταν χρησιμοποιείται κάποιο αρχείο για ενδιάμεση αποθήκευση δεδομένων. Η απαιτούμενη μνήμη όμως, θα είναι μεγαλύτερη από ότι όταν χρησιμοποιείται αρχείο γιατί για κάθε σύνολο, εκτός από την τιμή της συνάρτησης *f* θα φυλάγεται και κάποιο αναγνωριστικό (το χαρακτηριστικό διάνυσμα, ή καλύτερα, η διεύθυνση) ενός συνόλου, οπότε ο πίνακας αποθήκευσης των συνόλων θα καταλαμβάνει LMAX$(\alpha_2 + \gamma)$ bytes.

Μια παρατήρηση που μπορεί να κάνει κανείς στη μέθοδο των Kao-Queyranne είναι ότι έχει πλοκή $O(n^2 \cdot LSUM)$ που είναι μεγαλύτερη από $O(n^2 \cdot M)$ που έχει εκείνη των Schrage-Baker. Η αντικατάσταση του M από LSUM οφείλεται στο ότι τώρα τα σύνολα, πρέπει να δημιουργούνται με αύξουσα σειρά των διευθύνσεών τους και όχι με λεξικογραφική σειρά, για να αρκούν οι LMAX θέσεις του πίνακα. Εάν όμως οι εργασίες πάρουν τέτοια labels ώστε να ταυτίζονται οι δύο παραπάνω διατάξεις των συνόλων τότε και ο αλγόριθμος των Kao-Queyranne θα έχει πλοκή $O(n^2 \cdot M)$, αφού τα σύνολα θα μπορούν να δημιουργούνται με λεξικογραφική σειρά, όπως στον αλγόριθμο των Schrage-Baker, η οποία θα είναι ταυτόχρονα και αύξουσα ως προς τις διευθύνσεις. Οι Kao-Queyranne αποδεικνύουν ότι μια ικανή συνθήκη για να συμβαίνει αυτό είναι η σειρά με την οποία οι εργασίες παίρνουν labels να είναι συμβατή με τις σχέσεις προτεραιότητας, δηλαδή κάθε εργασία να παίρνει label αφού έχουν πάρει όλοι οι προγονοί της (και με βάση την ίδια σειρά να ορίζεται η λεξικογραφική διάταξη των συνόλων). Χρησιμοποιώντας λοιπόν, μια τέτοια ανάθεση labels σε εργασίες συνδυασμένη με τον αλγόριθμο λεξικογραφικής απαρίθμησης των εργασιών, ο αλγόριθμος των Kao-Queyranne αποκτά πλοκή $O(n^2 \cdot M)$.

Η απαιτούμενη μνήμη όμως αναμένεται μεγαλύτερη γιατί δεν χρησιμοποιείται τώρα ο ευρηματικός κανόνας για τη σειρά με την οποία θα πάρουν labels οι εργασίες (εκείνος που διαμερίζει το γράφο των προτεραιοτήτων σε αλυσίδες και δίνει labels στις εργασίες από τη μικρότερη αλυσίδα προς τη μεγαλύτερη), κανόνας ο οποίος είχε προταθεί ακριβώς για την ελαχιστοποίηση του LMAX. Επομένως δίνοντας labels στις εργασίες με διαφορετική σειρά (που να είναι συμβατή με τους κανόνες προτεραιότητας) από αυτή που προκύπτει από τον παραπάνω ευρηματικό κανόνα, αναμένεται μεγαλύτερο LMAX, δηλαδή μεγαλύτερη απαίτηση σε μνήμη.

4.5. Ο αλγόριθμος του Lawler

Ενας εντελώς διαφορετικός αλγόριθμος που έχει σαν στόχο την ελαχιστοποίηση της απαιτούμενης μνήμης, προτείνεται από τον Lawler [Lawl-79]. Στον αλγόριθμο αυτό τα σύνολα-καταστάσεις δημιουργούνται με αύξοντα πληθάριθμο ή ισοδύναμα κατά στάδιο. Από κάθε σύνολο R ενός σταδίου k (δηλαδή με k στοιχεία) σχηματίζονται όλα τα σύνολα της μορφής $R \cup \{j\}$ για κάθε εργασία j που δεν ανήκει στο R και όλοι οι πρόγονοι της ανήκουν σ'αυτό. Καθένα από τα σύνολα αυτά, που ανήκει στο στάδιο k+1, αναζητείται σε κάποια κατάλληλη δομή που βρίσκονται αποθηκευμένα σύνολα (δηλαδή πληροφορίες που σχετίζονται μ'αυτά) του σταδίου k+1. Εάν δεν υπάρχει εκεί, προστίθεται. Εάν υπάρχει, ελέγχεται η τιμή της συνάρτησης *f* που έχει μέχρι τώρα υπολογιστεί για το σύνολο αυτό και αν χρειάζεται ενημερώνεται. Οταν δημιουργηθούν όλα τα σύνολα του σταδίου k+1 (όταν δηλαδή, για κάθε σύνολο του σταδίου k δημιουργηθούν τα σύνολα εκείνα που προκύπτουν από αυτό με την προσθήκη μιας εργασίας), τα σύνολα του σταδίου k διαγράφονται, δηλαδή η μνήμη που αυτά καταλαμβάνουν επιστρέφεται στο σύστημα ώστε να μπορεί αργότερα να ξαναχρησιμοποιηθεί.

Αν λοιπόν, συμβολίσουμε με k_i τα εφικτά σύνολα του σταδίου i ($\sum_{i=1}^{n} k_i = M$) ο μέγιστος αριθμός εφικτών συνόλων που είναι ταυτόχρονα αποθηκευμένα θα είναι $\max_{2 \le i \le n} \{k_{i-1} + k_i\}$, που συμβολιζεται με SMAX. Αντίθετα από τις μεθόδους των Schrage-Baker και Kao-Queyranne των οποίων η απαίτηση μνήμης είναι από την αρχή γνωστή (επειδή είναι γνωστά τα LSUM και LMAX), εδώ δεν συμβαίνει αυτό γιατί το LMAX δε μπορεί να υπολογιστεί με κάποιο τρόπο από την αρχή. Αυτό θεωρείται το βασικότερο μειονέκτημα της μεθόδου, γιατί υπάρχει περίπτωση να χαθεί χρόνος προχωρώντας ο Δυναμικός Προγραμματισμός μέχρι κάποιο σημείο και σταματώντας κάποια στιγμή λόγω έλλειψης μνήμης. Εφόσον ο αριθμός των εφικτών συνόλων κάθε επιπέδου δεν είναι εκ των προτέρων γνωστός είναι βολικότερη η χρησιμοποίηση κάποιας δυναμικής δομής για την αποθήκευσή τους (π.χ. ισορροπημένα δέντρα) από κάποια στατική (πίνακες). Για κάθε σύνολο (που θα αποτελεί ένα στοιχείο της δομής που αντιστοιχεί στο στάδιο που αυτό ανήκει) θα πρέπει να αποθηκεύονται η τιμή της *f* σ' αυτό, το χαρακτηριστικό του διάνυσμα και οι κατάλληλοι δείκτες σε επόμενα σύνολα, οι οποίοι είναι απαραίτητοι αφού μιλάμε για δυναμική δομή και έστω ότι αποθηκεύονται σε δ bytes. Οπότε η μνήμη που απαιτείται για τον κύριο όγκο δεδομένων αυτού του αλγορίθμου είναι $SMAX \cdot (\alpha_2 + \beta + \delta)$ bytes.

Σχετικά τώρα με την υπολογιστική πλοκή του αλγορίθμου αυτού: Για κάθε σύνολο R, από τα M εφικτά σύνολα, και κάθε εργασία j, από τις O(n) που δεν ανήκουν στο R αλλά οι πρόγονοι της (αν υπάρχουν) ανήκουν, γίνεται μιά αναζήτηση στη δομή που κρατάει τα σύνολα του επόμενου (έστω i) επιπέδου για να βρεθεί το σύνολο που προκύπτει από την προσθήκη της j στο R. Επειδή στο επίπεδο i βρίσκονται το πολύ $\binom{n}{i}$ σύνολα που είναι λιγότερα από 2^n, εάν η δομή στην οποία είναι αυτά τοποθετημένα επιτρέπει αναζήτηση σε λογαριθμικό χρόνο (όπως π.χ. τα ισορροπημένα δέντρα), η αναζήτηση κάθε συνόλου απαιτεί O(n) συγκρίσεις χαρακτηριστικών διανυσμάτων και επειδή η σύγκριση δύο τέτοιων απαιτεί χρόνο O(n) η όλη αναζήτηση απαιτεί χρόνο $O(n^2)$. Η συνολική λοιπόν, πλοκή του αλγορίθμου είναι $O(n^3 \cdot M)$.

Οπως στον αλγόριθμο των Kao-Queyranne έτσι και εδώ υπάρχει το πρόβλημα της ανακατασκευής της βέλτιστης λύσης εφόσον οι ενδιάμεσοι κόμβοι-σύνολα διαγράφονται. Το πρόβλημα μπορεί κι εδώ να αντιμετωπιστεί με τους ίδιους τρόπους :

α) αποθηκεύοντας σε ένα αρχείο το ποια είναι η τελευταία εργασία στη βέλτιστη διάταξη κάθε συνόλου και διαβάζοντας το αρχείο αυτό n φορές βρίσκοντας κάθε φορά και μια εργασία της βέλτιστης διάταξης από το τέλος προς την αρχή, β) αποθηκεύοντας πάλι την ίδια πληροφορία σ'ένα αρχείο αλλά διαβάζοντάς το τώρα μία φορά μεταφέροντας το περιεχόμενο του σ'ένα πίνακα - έτσι όμως η απαιτούμενη μνήμη αυξάνει σε $M(\alpha_1 + \beta)$ bytes και

γ) προσδιορίζοντας το σύνολο των εργασιών που βρίσκονται στις πρώτες μισές θέσεις της βέλτιστης διάταξης και λύνοντας αναδρομικά το πρόβλημα για τις πρώτες μισές και τις δεύτερες μισές εργασίες ξεχωριστά - έτσι για κάθε σύνολο αποθηκεύεται ένα ακόμη χαρακτηριστικό διάνυσμα που αναφέρεται στο σύνολο των εργασιών που βρίσκονται στις $\lfloor n/2 \rfloor$ πρώτες θέσεις. Οπως λέγαμε και στον αλγόριθμο των Kao-Queyranne, θεωρητικά η υπολογιστική πλοκή μένει η ίδια. Πρακτικά όμως, ο πραγματικός χρόνος αναμένεται αισθητά μεγαλύτερος στις περιπτώσεις που χρησιμοποιείται δευτερεύουσα μνήμη.

Μια παραλλαγή της μεθόδου αυτής είναι να μήν διαγράφονται τα στάδια που περνιούνται, ώστε τελειώνοντας ο Δυναμικός Προγραμματισμός να βρίσκονται στη κύρια μνήμη όλες οι καταστάσεις όλων των σταδίων. Εάν λοιπόν, αποθηκεύοντας σε κάθε σύνολο-κατάσταση το ποια είναι η τελευταία εργασία στη βέλτιστη διάταξή του μπορούμε σε πολυωνυμικό χρόνο ($O(n^2)$) να ανακατασκευάσουμε τη βέλτιστη διάταξη. Ετσι κερδίζουμε σε χρόνο αφού δεν είναι απαραίτητη η χρήση των προαναφερθεισών εκθετικών διαδικασιών ανακατασκευής της βέλτιστης λύσης, αλλά αυξάνεται η απαιτούμενη μνήμη που τώρα γίνεται $M(\alpha + \beta + \delta)$ bytes.

Να σημειώσουμε εδώ ότι όταν η λέξη του υπολογιστή έχει τουλάχιστον τόσα bits όσα οι προς διάταξη εργασίες, όπως υποθέτει ο Lawler, τότε ερωτήματα όπως "ανήκει μια εργασία σε ένα σύνολο;" ή "ανήκουν όλοι (ή ένας) οι πρόγονοι (ή απόγονοι) μιάς εργασίας σ'ένα σύνολο;" ή "ποιο από δύο σύνολα προηγείται λεξικογραφικά;" και γενικότερα ερωτήματα που απαιτούν κάποια πράξη πάνω σε χαρακτηριστικά διανύσματα απαντιούνται σε O(1) και όχι σε O(n) όπως έχουμε δεχθεί μέχρι τώρα. Αυτό έχει σάν αποτέλεσμα όλες (εκτός του δευτέρου αλγορίθμου της εργασίας [HKS-63]) οι εκφράσεις υπολογιστικής πλοκής που δόθηκαν στο κεφάλαιο αυτό να έχουν τον εκθέτη του n μειωμένο κατά ένα. Στην πράξη, σε καμία από τις εργασίες που αναφέραμε και παρουσιάζουν αποτελέσματα εκτελέσης προγραμμάτων, δε χρειάστηκαν περισσότερες από δύο λέξεις για την αποθήκευση των χαρακτηριστικών διανυσμάτων. Επομένως, στην πράξη πάντα, ο απαιτούμενος αριθμός πράξεων διαφέρει από O(n) γιατί αν υποθέσουμε έναν υπολογιστή με λέξεις των 32 bits, αυξάνοντας το n από 5 σε 30 θα περίμενε κανείς έναν εξαπλασιασμό του αριθμού των πράξεων, κάτι που όμως δεν συμβαίνει αφού ο αριθμός αυτός μένει σταθερός ίσος με 1. Από την άλλη όμως διαφέρει κι από O(1) γιατί, υποθέτοντας πάλι το ίδιο μήκος λέξης, αυξάνοντας το n από 32 σε 33 ο αριθμός των πράξεων δεν μένει σταθερός αλλά διπλασιάζεται (αυξάνοντας από 1 σε 2). Οπως όμως παρατηρούν και οι Kao-Queyranne μιλώντας για υπολογιστική πλοκή, που αποτελεί ασυμπτωτική έκφραση του αριθμού των πράξεων, πρέπει κάθε πράξη μεταξύ χαρακτηριστικών διανυσμάτων να θεωρείται ότι έχει πλοκή O(n).

4.6. Συνοπτική σύγκριση των αλγορίθμων

Συγκρίνοντας τους αλγορίθμους που αναφέραμε παρατηρούμε ότι τη μικρότερη υπολογιστική πλοκή, $O(n^2 \cdot M)$, έχουν οι δύο αλγόριθμοι των Baker και Schrage και ο αλγόριθμος των Kao-Queyranne όταν χρησιμοποιείται με την παραλλαγή στην οποία οι εργασίες παίρνουν labels με σειρά συμβατή με τις σχέσεις προτεραιότητας και τα εφικτά σύνολα δημιουργούνται με λεξικογραφική σειρά (που είναι ταυτοχρονα και αύξουσα σειρά των διευθύνσεών τους). Η διαδικασία ανάθεσης labels σε εργασίες που ακολουθείται στην εργασία [BaSc-78] είναι πιο πολύπλοκη και λιγότερο αποτελεσματική (δίνει μεγαλύτερο LSUM) από αυτή που ακολουθείται στην [ScBa-78] με αποτέλεσμα να έχει μεγαλύτερη απαίτηση και σε χρόνο και σε μνήμη, όπως παρατηρείται και πειραματικά στη δεύτερη. Ο αλγόριθμος των Kao-Queyranne, επίσης, ανεξάρτητα από το εάν θα υλοποιηθεί με παραλλαγή που έχει πλοκή $O(n^2 \cdot LSUM)$ ή $O(n^2 \cdot M)$, απαιτεί, αφού τελειώσει η αναδρομή του Δυναμικού Προγραμματισμού, την ενεργοποίηση κάποιου εκθετικού αλγορίθμου (που μπορεί και να απαιτεί χρήση δευτερεύουσας μνήμης) για την ανακατασκεύη της βέλτιστης λύσης. Κάτι τέτοιο δε γίνεται στους αλγορίθμους των Schrage-Baker. Από τα παραπάνω, λοιπόν, φαίνεται ότι ο μικρότερος αριθμός πράξεων, συγκρίνοντας τους αλγορίθμους με βάση τη χειρότερη παρουσιαζόμενη περίπτωση (αυτό δηλαδή που δηλώνει η υπολογιστική πλοκή), απαιτείται από τον αλγόριθμο των Schrage-Baker που παρουσιάζεται στην εργασία [ScBa-78].

Ίσως όμως επειδή η μέση συμπεριφορά ενός αλγορίθμου είναι συχνά πολύ διαφορετική από τη χειρότερη, οι μέχρι τώρα δημοσιευμένες εργασίες που έχουν συγκρίνει πειραματικά τους αλγορίθμους των Schrage-Baker και του Lawler (και κάποια των Kao-Queyranne) δεν πήραν τα ίδια αποτελέσματα σχετικά με την ταχύτητα. Συγκεκριμένα, από τις τρεις εργασίες που ασχολήθηκαν με μια τέτοια σύγκριση, στην [KaQu-82] καταλήγουν στο συμπέρασμα ότι ο αλγόριθμος του Lawler είναι ο ταχύτερος, ενώ στις [PoWa-87] και [APW-90] ότι ταχύτερος είναι των Schrage-Baker, όπως μας υπέδειξε η υπολογιστική τους πλοκή. Με τις δύο τελευταίες εργασίες συμφωνούν και τα δικά μας πειράματα όπως θα πούμε στο επόμενο κεφάλαιο. Ο Steiner [Stei-90] θεωρεί ότι είναι δύσκολο να γίνονται συγκρίσεις γιατί η αποδοτικότητα του αλγορίθμου του Lawler, λόγω του ότι είναι πιο περίπλοκος από τον άλλο αλγόριθμο, επηρεάζεται από την αποδοτικότητα του κώδικα που τον υλοποιεί και από την αποδοτικότητα της εκτέλεσης του κώδικα στον συγκεκριμένο υπολογιστή περισσότερο απ’ ότι η αποδοτικότητα του αλγορίθμου των Schrage-Baker.

Όσον αφορά στις απαιτήσεις σε μνήμη των αλγορίθμων που παρουσιάστηκαν, παρατηρούμε ότι τη μικρότερη μνήμη, $SMAX \cdot (\alpha_2+\beta+\delta)$ bytes για τον κύριο όγκο των δεδομένων, απαιτεί ο αλγόριθμος του Lawler όταν χρησιμοποιεί κάποιο αρχείο σε δευτερεύουσα μνήμη για την αποθήκευση πληροφοριών σχετικών με την ανακατασκευή της βέλτιστης λύσης (το οποίο διαβάζεται έπειτα n φορές) και λίγο περισσότερη όταν το πρόβλημα λύνεται αναδρομικά, με τον αλγόριθμο του Lawler πάλι, χρησιμοποιώντας την ιδέα των Munro και Ramirez. Το γεγονός ότι οι παραπάνω παραλλαγές του αλγορίθμου του Lawler χρησιμοποιούν περισσότερη (μέχρι περίπου τριπλάσια) μνήμη για κάθε εφικτό σύνολο από άλλες παραλλαγές του αλγορίθμου ή από τον αλγόριθμο των Held-Karp-Shareshian που κρατούν πληροφορίες για M σύνολα ταυτόχρονα δεν είναι αρκετό να αλλάξει τον ισχυρισμό ότι είναι οι μέθοδοι που απαιτούν τη λιγότερη μνήμη, γιατί σύμφωνα με τις μετρήσεις του Steiner το SMAX είναι, κατά μέσο όρο, 160-170 φορές μικρότερο από το LSUM δηλαδή (επειδή το M είναι, κατά μέσο όρο, 12 φορές μικρότερο από το LSUM) 13-14 φορές μικρότερο από το M. Αυτή άλλωστε (η μέθοδος του Lawler) είναι η μέθοδος που προτείνεται από τις σχετικές εργασίες σαν εκείνη που απαιτεί τη μικρότερη μνήμη από όσες, τουλάχιστον, εξετάζουν.

5. Αλγόριθμοι επίλυσης του 1/s_{ij}/$\sum C_i$,T_{max}

Εφόσον παρουσιάσαμε, στο δεύτερο κεφάλαιο, μία ανάλυση του 1/s_{ij}/$\sum C_i$,T_{max} και, κυρίως, του 1/s_{ij}/$\sum C_i$|T_{max}=0 στο οποίο ανάγεται το προηγούμενο, αναλύσαμε, στο τρίτο κεφάλαιο, το γιατί διάφοροι μέθοδοι επίλυσης προβλημάτων Χρονικού Προγραμματισμού που έχουν εφαρμοστεί σε άλλες περιπτώσεις δεν μπορούν να εφαρμοστούν στο δικό μας πρόβλημα, παρουσιάσαμε, στην παράγραφο 3.2, το γενικό σχήμα των αλγορίθμων διαμερισμού και φραγής, που συνήθως χρησιμοποιούνται για την επίλυση προβλημάτων Χρονικού Προγραμματισμού και περιγράψαμε, στο τέταρτο κεφάλαιο, τους κυριότερους αλγορίθμους Δυναμικού Προγραμματισμού που έχουν προταθεί για λύση προβλημάτων Χρονικού Προγραμματισμού έφτασε η στιγμή να παρουσιάσουμε τους αλγορίθμους που υλοποιήσαμε για το δικό μας πρόβλημα και να αιτιολογήσουμε αφενός την επιλογή των συγκεκριμένων αλγορίθμων κι αφετέρου μικρότερης σημασίας αποφάσεις που λήφθηκαν κατά την υλοποίηση.

5.1. Το ενδεχόμενο της χρησιμοποίησης αλγορίθμου διαμερισμού και φραγής

Όπως έχουμε ξαναπεί η συνηθέστερη μέθοδος που χρησιμοποιείται για την επίλυση προβλημάτων Χρονικού Προγραμματισμού είναι ένας αλγόριθμος διαμερισμού και φραγής που ακολουθεί το γενικό σχήμα που παρουσιάσαμε στην παράγραφο 3.2. Το σχήμα αυτό θα μπορούσε να ακολουθεί και κάποιος αλγόριθμος που θα είχε ως στόχο την επίλυση του 1/s_{ij}/$\sum C_i$|T_{max}=0. Η ταχύτητα του αλγορίθμου αυτού θα εξαρτιόταν κυρίως από τη μέθοδο υπολογισμού κάτω φράγματος σε κάθε κόμβο του δένδρου. Συγκεκριμένα για να ήταν παραδεκτή η συνολική ταχύτητα του αλγορίθμου θα έπρεπε τα κάτω φράγματα να υπολογίζονται σχετικά γρήγορα και να είναι αρκετά σφικτά ώστε να απορρίπτεται μια σημαντική περιοχή του χώρου έρευνας, που περιέχει O(n!) κόμβους. Ενδεικτικό της μεγάλης σημασίας που έχει η ποιότητα του κάτω φράγματος (και η ταχύτητα υπολογισμού του) είναι το γεγονός ότι όλες σχεδόν οι εργασίες που λύνουν προβλήματα Χρονικού Προγραμματισμού με αλγορίθμους διαμέρισμου και φραγής έχουν σαν κύριο θέμα τους την ανέυρεση σφικτού (και όχι πολύ χρονοβόρου στον υπολογισμό του) κάτω φράγματος.

Το κάτω φράγμα προκύπτει πάντα από το άθροισμα δύο όρων, ο πρώτος από τους οποίους είναι το κόστος που επιφέρει η μερική διάταξη των ήδη διατεταγμένων εργασιών, που είναι γνωστό, και ο δεύτερος είναι ένα κάτω φράγμα στο κόστος που θα επιφέρει η μερική διάταξη των αδιάτακτων εργασιών δεδομένου ότι η εκτελεσή τους θα ξεκινήσει τη στιγμή που τελειώνει η εκτέλεση της τελευταίας από τις διατεταγμένες (όταν η κατασκευή του βέλτιστου προγράμματος γίνεται από την αρχή προς το τέλος). Επομένως ο υπολογισμός του κάτω φράγματος ανάγεται στον υπολογισμό του δεύτερου όρου που αναφέρεται στις αδιάτακτες εργασίες. Δυστυχώς στο δικό μας πρόβλημα δεν στάθηκε δυνατή η επινόηση κάποιας μεθόδου που θα οδηγούσε σε σφικτά κάτω φράγματα.

Δύο μέθοδοι υπολογισμού κάτω φράγματος που θα μπορούσαν να χρησιμοποιηθούν στο προβλημά μας είναι η χαλάρωση Lagrange στους περιορισμούς χωρητικότητας της μηχανής που προτείνει ο Fisher [Fish-76] και η χαλάρωση του χώρου καταστάσεων του Δυναμικού Προγραμματισμού που προτείνεται στην εργασία [AbPo-88]. Όπως εξηγήσαμε αναλυτικά στην παράγραφο 3.2 συγκεκριμένοι λόγοι για τη κάθε μέθοδο που επηρεάζουν αρνητικά την ποιότητα του προκύπτοντος κάτω φράγματος (που μπορεί να φτάσουν μέχρι το να καταστήσουν χωρίς νόημα την εφαρμογή της πρώτης ειδικά) σε συνδυασμό με τη μεγάλη υπολογιστική τους πλοκή δεν αφήνουν πολλές ελπίδες για μια συνολικά καλή αποδοσή τους, κάτι που επιβεβαιώνεται και από τα πειραματικά δεδομένα της εργασίας [APW-90] στην οποία οι δύο παραπάνω αλγόριθμοι φέρονται να δίνουν τα λιγότερο καλά αποτελέσματα από όσες μεθόδους δοκιμάστηκαν μεταξύ των οποίων υπήρχαν και αλγόριθμοι Δυναμικού Πραγραμματισμού.

Ένα άλλο κάτω φράγμα, του οποίου ο υπολογισμός δεν έχει μεγάλη πλοκή, είναι αυτό που προκύπτει παραλλάσσοντας κατάλληλα τη ιδέα για υπολογισμό κάτω φράγματος του Γεωργίου [Γεωρ-91]. Συγκεκριμένα, θεωρώντας για απλούστευση ότι οι χρόνοι εξάρμωσης είναι ανεξάρτητοι ακολουθίας, μπορούμε να κατασκευάσουμε ένα πρόγραμμα στο οποίο οι χρόνοι εξάρμωσης των ομάδων εμφανίζονται ακριβώς μία φορά ο καθένας και μάλιστα με τέτοιο τρόπο ώστε να αυξάνει το $\sum C_i$ όσο το δυνατόν λιγότερο (δηλαδή στην αρχή μπαίνει ο μικρότερος χρόνος εξάρμωσης μετά από τόσες εργασίες όσες έχει η πρώτη σε πλήθος ομάδα μπαίνει ο δεύτερος μικρότερος, μετά από τόσες εργασίες όσες έχει η δεύτερη σε πλήθος ομάδα ο τρίτος μικρότερος κ.ο.κ) και με δεδομένη την ύπαρξη αυτών των ανενεργών χρονικών περιόδων να λύσουμε το $1/\text{idle time}/\sum C_i, T_{max}=0$, αγνοώντας δηλαδή τώρα το διαχωρισμό των εργασιών σε ομάδες, για το οποίο αποδεικνύεται όπως ακριβώς στο $1//\sum C_i, T_{max}=0$ ότι λύνεται με τον ίδιο τρόπο με το τελευταίο. Επειδή στον αλγόριθμο αυτή η διάταξη των εργασιών (αφού έχουν τοποθετηθεί σε κάποιες θέσεις οι χρόνοι εξάρμωσης μία φορά ο καθένας) γίνεται αγνοώντας το διαχωρισμό τους σε ομάδες, αγνοώντας δηλαδή το βασικότερο χαρακτηριστικό του προβλήματος που έχει καθοριστική επίδραση στη διαμόρφωση της βέλτιστης διάταξης και της τιμής της, το προκύπτον κάτω φράγμα δεν αναμένεται αρκετά σφικτό, κάτι που επιβεβαιώνεται και από τους Γεωργίου [Γεωρ-91] και Τσατσάκη [Τσατ-93].

Αφού λοιπόν δεν βρήκαμε κάποιο σχετικά γρήγορα υπολογιζόμενο και σχετικά σφικτό κάτω φράγμα δεν έχουμε ελπίδα να μειώσουμε αισθητά τις τουλάχιστο $O(n!\cdot n)$ πράξεις (αφού οι κόμβοι είναι $O(n!)$ και για καθέναν από αυτούς χρειάζονται τουλάχιστο $O(n)$ πράξεις για τη δημιουργία των παιδιών του) που απαιτεί ένας αλγόριθμος διαμερισμού και φραγής κι επομένως να εκτελείται με παραδεκτή ταχύτητα κάποια υλοποίηση του αλγορίθμου. Το δυσμενές αυτό συμπέρασμα δεν μπορεί να ανατραπεί ούτε και αν η στρατηγική έρευνας του σχηματιζόμενου δένδρου είναι η επιλογή κάθε φορά για διακλάδωση του κόμβου με το μικρότερο κάτω φράγμα, που απαιτεί περισσότερη μνήμη από τη συνήθως χρησιμοποιούμενη depth-first-search στρατηγική αλλά δίνει ταχύτερους αλγορίθμους ([Fox-70], [Fox-78], [Ibar-87]). Κι αυτό γιατί το γεγονός ότι το κάτω φράγμα που θα χρησιμοποιήσουμε, εάν υλοποιήσουμε έναν τέτοιο αλγόριθμο, είναι χαλαρό, κάτι που θα είναι τόσο εντονότερο όσο πλησιέστερα στη ρίζα βρίσκεται ο κόμβος για τον οποίο υπολογίζεται το κάτω φράγμα αφού τόσο περισσότερες είναι εκεί οι αδιάτακτες εργασίες, δεν θα επιτρέπει να εμφανίζεται το μικρότερο κάτω φράγμα σε κάποιο κόμβο που απέχει έστω και λίγα επίπεδα από τους κόμβους που βρίσκονται πλησιέστερα στη ρίζα και δεν έχουν ακόμα δημιουργηθεί τα παιδιά τους. Έτσι το τέλος του αλγορίθμου, που έρχεται όταν εμφανιστεί ένα πλήρες πρόγραμμα, σε κάποιο φύλλο του δένδρου, με τιμή της αντικειμενικής συνάρτησης (που μπορεί να ληφθεί και σαν κάτω φράγμα του φύλλου αυτού) μικρότερη από το κάτω φράγμα κάθε κόμβου, θα έρθει όταν όλοι οι κόμβοι των οποίων τα παιδιά δεν έχουν δημιουργηθεί βρίσκονται κοντά στα φύλλα, δηλαδή έχει ερευνηθεί το μεγαλύτερο μέρος του δένρου.

Να σημειώσουμε τελειώνοντας την αναφορά μας στο ενδεχόμενο χρησιμοποίησης αλγόριθμου διαμερισμού και φραγής για το πρόβλημά μας, ότι το ενδεχόμενο να ελέγχει κανείς ποιός από τους κόμβους που αντιστοιχούν σε ένα συγκεκριμένο υποσύνολο εργασιών (στην περίπτωση που υπάρχουν χρόνοι εξάρμωσης θα πρέπει οι κόμβοι να πληρούν και κάποιες άλλες συνθήκες, ανάλογα με την περίπτωση) έχει τη μικρότερη, μέχρι εκείνη τη στιγμή, τιμή στην αντικειμενική συνάρτηση και να "κλαδεύει" τους υπόλοιπους, όταν χρησιμοποιείται breadth-first-search ή και η παραπάνω στρατηγική, ισοδυναμεί με τη χρήση Δυναμικού Προγραμματισμού αφού για κάθε υποσύνολο αποθηκεύεται ένας μόνο κόμβος, αντιστοιχίζοντας ουσιαστικά σε κάθε κόμβο ένα υποσύνολο εργασιών κι όχι μια διάταξη ενός υποσυνόλου, γι’ αυτό και κάτι τέτοιο δεν θα το θεωρούμε ειδική περίπτωση του αλγορίθμου διαμερισμού και φραγής.

5.2. Σχήμα Δυναμικού Προγραμματισμού για το 1/S_{ij}/$\sum C_i$|T_{max}=0

Μία άλλη μέθοδος που χρησιμοποιείται για την επίλυση προβλημάτων Χρονικού Προγραμματισμού είναι, όπως έχουμε πει, ο Δυναμικός Προγραμματισμός. Όπως είπαμε στο προηγούμενο κεφάλαιο, η πλοκή κάθε αλγορίθμου Δυναμικού Προγραμματισμού έχει σαν σημαντικότερο παράγοντα το 2^n αντί του n! κι έτσι είναι πολύ μικρότερη από αυτή των αλγορίθμων διαμερισμού και φραγής. Γι’ αυτό, ειδικά όταν δεν υπάρχει κάποιο καλό κάτω φράγμα να μειώσει σημαντικά τον αριθμό των πράξεων του αλγορίθμου διαμερισμού και φραγής, ένας αλγόριθμος Δυναμικού Προγραμματισμού αναμένεται κατά πολύ ταχύτερος ενός αλγορίθμου διαμερισμού και φραγής που λύνει το ίδιο πρόβλημα. Αυτό προκύπτει και από τις πειραματικές μετρήσεις των εργασιών [ScBa-78], [PoWa-85] και [APW-90] όπου οι αλγόριθμοι Δυναμικού Προγραμματισμού φέρονται σχεδόν πάντα ταχύτεροι από τους αντίστοιχους αλγορίθμους διαμερισμού και φραγής.

Το μεγάλο μειονέκτημα των πρώτων έναντι των δευτέρων είναι η πολύ μεγαλύτερη απαίτηση σε μνήμη, ειδικά όταν οι αλγόριθμοι διαμερισμού και φραγής ακολουθούν μία στρατηγική έρευνας depth-first-search, όπως γίνεται συνήθως, οπότε η απαιτούμενη μνήμη είναι O(n) ενώ των αλγορίθμων Δυναμικού Προγραμματισμού είναι $O(2^n)$. Αυτός είναι και ο βασικός λόγος που οι αλγόριθμοι διαμερισμού και φραγής συνήθως προτιμούνται από το Δυναμικό Προγραμματισμό. Από τη μια όμως η αύξηση της μνήμης των υπολογιστών κατά τα τελευταία 10-15 χρόνια που είναι δυσανάλογα μεγαλύτερη από την αύξηση της υπολογιστικής δύναμης αυτών και από την άλλη η μεγαλύτερη δυσκολία επίλυσης των προβλημάτων Χρονικού Προγραμματισμού όταν αυτά περιέχουν χρόνους εξάρμωσης (ή τουλάχιστον του $1/S_{ij}/\sum C_i$ και πολύ περισσότερο του $1/S_{ij}/\sum C_i, T_{max}=0$) που συνεπάγεται μεγαλύτερους χρόνους εκτέλεσης των προγραμμάτων για ίδιο αριθμό εργασιών μας ωθούν στην υπόθεση ότι ίσως τελικά ο περιοριστικός παράγοντας να είναι ο χρόνος και όχι η μνήμη, εφόσον στα πλαίσια του προγραμματισμού παραγωγής ενδιαφερόμαστε για εκτελέσεις προγραμμάτων σε σχεδόν "πραγματικό χρόνο", υπόθεση που όπως θα δούμε στο επόμενο κεφάλαιο επαληθεύεται.

Αιτιολογώντας τον ισχυρισμό ότι το $1/S_{ij}/\sum C_i$ απαιτεί μεγαλύτερο υπολογιστικό χρόνο από τα προβλήματα Χρονικού Προγραμματισμου χωρίς χρόνους εξάρμωσης, πρέπει να αναφέρουμε ότι οι Ahn-Hyun [AhHy-90] λύνουν προβλήματα μέχρι 25 εργασιών σε χρόνο ενός λεπτού (εκτός και αν ο αριθμός των ομάδων είναι δυσανάλογα μικρότερος από τον αριθμό των εργασιών, π.χ. n=30, b=3) ενώ στις εργασίες [ScBa-78], [PoWa-85] και [APW-90] που χρησιμοποιούν επίσης Δυναμικό Προγραμματισμό για λύση όμως προβήματος χωρίς χρόνους εξάρμωσης φτάνουν μέχρι 50, 30 και 40 εργασίες αντίστοιχα πριν εμφανιστούν προβλήματα με τη μνήμη ενώ οι χρόνοι εκτέλεσης των προγραμμάτων είναι μικρότεροι απο 5 δευτερόλεπτα το πολύ. Αρκετές επίσης από τις εργασίες που έχουμε αναφέρει και λύνουν προβλήματα Χρονικού Προγραμματισμού με τη συνήθως βραδύτερη, όπως έχουμε πει, μέθοδο του διαμερισμού και φραγής επιτυγχάνουν σε χρόνο ενός λεπτού επίλυση προβλημάτων μέχρι 30 περίπου εργασιών χωρίς να λείπουν και περιπτώσεις που φτάνουν τις 50. Είναι λοιπόν φανερό ότι λύνοντας το $1/S_{ij}/\sum C_i | T_{max}=0$ που (για λόγους που αναφέραμε στο δεύτερο και τρίτο κεφάλαιο και θα ξαναθίξουμε αργότερα αλλά και όπως θα φανεί πειραματικά στο επόμενο κεφάλαιο) απαιτεί πολύ περισσότερο χρόνο για την επίλυσή του από το $1/S_{ij}/\sum C_i$, θα περιοριστούμε αναγκαστικά σε σχετικά μικρές τιμές του πλήθους των εργασιών, οπότε ελπίζουμε ότι η διαθέσιμη μνήμη των 64 MBytes (πολύ μεγαλύτερη των 12 MBytes των Ahn-Hyun και των όχι περισσότερο από 200 KBytes των υπολοίπων) θα αποδειχθεί, όπως και πράγματι γίνεται, επαρκής.

Οι παραπάνω σκέψεις μας ώθησαν να προσανατολιστούμε προς μία λύση του προβλήματος με Δυναμικό Προγραμματισμό παρά με έναν αλγόριθμο διαμερισμού και φραγής. Το σχήμα του Δυναμικού Προγραμματισμού που χρησιμοποιήσαμε είναι κάτι μεταξύ του σχήματος των Monma-Potts [MoPo-84] για το $1/S_{ij}/\sum w_i C_i$ (και λιγότερο των Ahn-Hyun [AhHy-90] που παρουσιάσαμε στην παράγραφο 3.1) και του γενικού σχήματος Δυναμικού Προγραμματισμού που χρησιμοποείται για προβλήματα χωρίς χρόνους εξάρμωσης, που παρουσιάσαμε στην παράγραφο 4.1: Υπάρχουν πάλι N+1 στάδια το καθένα από τα οποία αντιστοιχεί στο σύνολο των ημιτελών (γενικά) προγραμμάτων με ένα συγκεκριμένο πλήθος εργασιών (από 0 έως N). Κάθε κατάσταση αντιστοιχεί στο σύνολο των ημιτελών προγραμμάτων που περιέχουν εργασίες από ένα συγκεκριμένο σύνολο R, η τελευταία εργασία τους ανήκει σε μία συγκεκριμένη ομάδα α και αποπερατώνονται μία συγκεκριμένη χρονική στιγμή t. Ο άμεσος ορισμός του συνόλου των εργασιών R, αντί της αναφοράς του πλήθους των εργασιών από κάθε ομάδα που ανήκουν σ' αυτό, όπως γινόταν στο $1/S_{ij}/\sum w_i C_i$ είναι απαραίτητος αφού το πρόβλημά μας δεν είναι πρόβλημα διατεταγμένων ομάδων. Σε κάθε κατάσταση, που καθορίζεται από την τριάδα των μεταβλητών κατάστασης R, α, t, αντιστοιχεί η μικρότερη τιμή του $\sum C_i$ που μπορεί να έχει μία διάταξη των εργασιών του R που τελειώνει τη στιγμή t με κάποια εργασία της ομάδας α (οπότε μπορούμε να θεωρούμε ότι σε κάθε κατάσταση αντιστοιχεί η διάταξη αυτή με την μικρότερη τιμή του $\sum C_i$), έστω $f(R,\alpha,t)$ που δίνεται από τον τύπο:

$$f(R,\alpha,t)=\begin{cases}0, & \text{εάν } R=\varnothing,\ \alpha=0,\ t=0\\ \min\limits_{\substack{i\in Q(R),\ g(i)=\alpha\\ 0\le c\le b}} f(R-\{i\},c,t-S_{c,\alpha}-p_i)+t, & \text{εάν υπάρχει } i\in Q(R) \text{ με } g(i)=\alpha \text{ και } d_i\ge t\\ \infty, & \text{διαφορετικά}\end{cases}$$

όπου με Q(R) συμβολίζουμε, όπως έχουμε πει, το υποσύνολο του R που περιέχει αυτές τις εργασίες που δεν έχουν κάποιο απόγονό τους στο R. (Σχέσεις προτεραιότητας προκύπτουν στο πρόβλημά μας με τους δύο κανόνες που αναφέραμε στην παράγραφο 2.3). Το c στον παραπάνω τύπο παίρνει την τιμή 0 (που αντιστοιχεί, όπως έχουμε πει, στην ανενεργή κατάσταση της μηχανής που προηγείται της έναρξης της εκτέλεσης κάποιου προγράμματος) όταν το R είναι μονομελές. Η λύση του προβλήματος δίνεται από $\min\limits_{\substack{1\le\alpha\le b\\ 0<t<\infty}} f(S,\alpha,t)$ όπου με S συμβολίζουμε το σύνολο όλων των εργασιών.

Εάν συμβολίσουμε με T το μεγαλύτερο πλήθος τιμών που μπορεί να πάρει η μεταβλητή κατάστασης t για όλα τα R έχουμε ότι ένα άνω φράγμα στο πλήθος των καταστάσεων είναι το $2^n \cdot b \cdot T$. Είναι προφανές ότι το μεγαλύτερο πλήθος τιμών μπορεί να το πάρει η t όταν R=S. Τότε ένα άνω φράγμα στο T δίνεται, όπως προτείνουν οι Monma-Potts [MoPo-89] από τη μέγιστη τιμή που μπορεί να πάρει το C_{max}, δηλαδή από $n\cdot(\overline{p}+\overline{S})$. Ένα καλύτερο άνω φράγμα δίνεται από τη διαφορά της μέγιστης τιμής που μπορεί να πάρει το C_{max} από την ελάχιστη τιμή που μπορεί αυτό να πάρει δηλαδή $n\cdot(\overline{p}+\overline{S})-(n\cdot\overline{p}+b\cdot\overline{S})=(n-b)\cdot\overline{S}$. Οι Monma-Potts παρατηρούν ακόμα ότι όταν οι χρόνοι εξάρμωσης μπορούν να πάρουν s διαφορετικές τιμές, ένα άλλο φράγμα του T δίνεται από n^s. Για χρόνους εξάρμωσης εξαρτημένους και ανεξάρτητους ακολουθίας το s μπορεί να είναι μέχρι b^2+b και b αντίστοιχα οπότε το προκύπτον άνω φράγμα είναι, για συνηθισμένες και "φυσιολογικές" τιμές των παραμέτρων n, b, $\overline{S}$, όπως θα πούμε στο επόμενο κεφάλαιο, πολύ μεγαλύτερο του $(n-b)\cdot\overline{S}$. Όταν όμως ο χρόνος εξάρμωσης είναι σταθερός το τελευταίο άνω φράγμα του T γίνεται μόνο n, που είναι βέβαια, πολύ καλύτερο από $(n-b)\cdot\overline{S}$. Συνοψίζοντας λοιπόν, έχουμε ότι ο αριθμός των καταστάσεων είναι $O(2^n \cdot b \cdot T)$ όπου το T φράσσεται από την ποσότητα

$(n-b)\cdot\bar{S}$ και στην ειδική περίπτωση των σταθερών χρόνων εξάρμωσης από n.

Όταν εμφανίζεται το T σε εκφράσεις υπολογιστικής πλοκής ή απαιτούμενης μνήμης θα προτιμούμε να το αφήνουμε έτσι και να μην το αντικαθιστούμε από το άνω φράγμα του $(n-b)\cdot\bar{S}$. Κι αυτό γιατί η ποσότητα $b\cdot(n-b)$ που θα εμφανίζεται δείχνει ότι η πλοκή ή η απαιτούμενη μνήμη γίνεται μέγιστη (ως προς το b) για $b=n/2$, κάτι που όπως θα δούμε στο επόμενο κεφάλαιο δεν ισχύει αφού πλοκή και απαιτούμενη μνήμη αυξανουν με το b. Και δεν υπάρχει λόγος να μην γίνεται αυτό αφού το $(n-b)\cdot\bar{S}$ δεν είναι παρά ένα άνω φράγμα, και συγκεκριμένα το στενότερο από όσα έχουν περιέλθει στην αντίληψή μας, του T. Εάν στη θέση του είχε χρησιμοποιηθεί το n^{b^2+b} θα ήταν και σε πρώτη ματιά φανερή η σχέση της όλης ποσότητας με το b, αλλά δεν θα είχαμε χρησιμοποιήσει το στενότερο άνω φράγμα του T.

5.3. Απαρίθμηση των εφικτών συνόλων με αύξουσα σειρά πληθάριθμου

Η υπολογιστική πλοκή και η απαιτούμενη μνήμη ενός αλγορίθμου που δουλεύει με βάση το παραπάνω σχήμα εξαρτώνται από το συγκεκριμένο αλγόριθμο και συγκεκριμένα από το ποιά από τις τεχνικές που παρουσιάσαμε στο προηγούμενο κεφάλαιο χρησιμοποιείται για την αποθήκευση των σχετικών με τα υποσύνολα εργασιών πληροφοριών. Αρχικά, επειδή δεν ήμασταν σίγουροι ότι ο χρόνος θα αποδεικνυόταν πιο περιοριστικός από τη μνήμη, αποφασίσαμε να χρησιμοποιήσουμε τον αλγόριθμο του Lawler για την αποθήκευση των σχετικών με τα εφικτά υποσύνολα πληροφοριών, αφού ο αλγόριθμος αυτός έχει τη μικρότερη απαίτηση σε μνήμη αν και η υπολογιστική του πλοκή στην περίπτωση που χρησιμοποιείται το γενικό σχήμα Δυναμικού Προγραμματισμού για προβλήματα χωρίς χρόνους εξάρμωσης, που αναφέραμε στο προηγούμενο κεφάλαιο, δεν είναι η μικρότερη δυνατή.

΄Ενα άλλο προτέρημα του αλγορίθμου αυτού έναντι εκείνου των Schrage-Baker (και των Kao-Queyranne και, εν μέρει, των Held-Karp-Shareshian) που χρησιμοποιούν πίνακα για την αποθήκευση των σχετικών με τα υποσύνολα πληροφοριών (αντί της δυναμικής αποθήκευσης του Lawler) και που η απαρίθμηση των συνόλων δεν γίνεται με αύξοντα πληθάριθμο είναι ότι εάν κατά την εξέλιξη του αλγορίθμου διαπιστωθεί ανεπάρκεια μνήμης μπορεί να σταματήσει στο σημείο αυτό ο αλγόριθμος και η έρευνα να συνεχιστεί ξεκινώντας ένας αλγόριθμος διαμερισμού και φραγής από κάθε κατάσταση του Δυναμικού Προγραμματισμού της οποίας οι καταστάσεις απόγονοι δεν έχουν ακόμα δημιουργηθεί. Όταν χρησιμοποιείται πίνακας υπάρχει το ενδεχόμενο να μην είναι αρκετή η μνήμη για την αποθήκευσή του, οπότε με το Δυναμικό Προγραμματισμό δεν ερευνάται κανένα τμήμα του χώρου έρευνας και μένει όλη η δουλειά για τον βραδύτερο αλγόριθμο διαμερισμού και φραγής.

Επίσης όταν υπάρχει το ενδεχόμενο αλλαγής του αλγορίθμου από Δυναμικό Προγραμματισμό σε διαμερισμό και φραγή είναι καλό να δημιουργούνται τα σύνολα με αύξουσα σειρά πληθάριθμου. Κι αυτό γιατί ακόμα κι αν υποθέσουμε ότι έχει δημιουργηθεί ίδιος αριθμός συνόλων με τις δύο μεθόδους τη στιγμή που προκύπτει πρόβλημα μνήμης (κάτι που δεν πρόκειται, βέβαια, να συμβεί αφού για τον ίδιο αριθμό δημιουργημένων συνόλων η μέθοδος των Schrage-Baker απαιτεί περισσότερη μνήμη αφού κρατάει αποθηκεύμενο ολόκληρο των πίνακα αποθήκευσης των συνόλων) το πιο ολιγομελές σύνολο που θα έχει δημιουργηθεί όταν αυτά απαριθμούνται με αύξουσα σειρά πληθάριθμου θα έχει περισσότερες εργασίες από το πιο ολιγομελές σύνολο που θα έχει δημιουργηθεί όταν αυτά απαριθμούνται με λεξικογραφική σειρά, οπότε η πλοκή της έρευνας που απομένει για τον βραδύτερο αλγόριθμο διαμερισμού και φραγής είναι μικρότερη. Εάν, για παράδειγμα, δημιουργηθούν ακριβώς τα μισά σύνολα και όλα τα σύνολα είναι εφικτά, απαριθμώντας τα σύνολα με αύξουσα σειρά πληθάριθμου το βάθος που θα έχουν τα δένδρα έρευνας που θα δημιουργηθούν από τον αλγόριθμο διαμερισμού και φραγής όταν αυτός κληθεί για να συνεχιστεί η έρευνα από τις καταστάσεις του δυναμικού προγραμματισμού που αντιστοιχούν στα σύνολα πληθάριθμου $n/2$, θα είναι $n/2$ (οπότε η πλοκή της έρευνας σε κάθε τέτοιο δένδρο θα είναι $(n/2)!$), ενώ αν τα σύνολα απαριθμούνται λεξικογραφικά θα υπάρχει δένδρο έρευνας βάθους $n-1$, αφού θα υπάρχει ένα μονοσύνολο με το οποίο δεν θα έχει ασχοληθεί ο δυναμικός προγραμματισμός, οπότε η πλοκή της έρευνας θα είναι $(n-1)!$.

Τέλος, όπως θα δούμε στη συνέχεια, όταν χρησιμοποιείται το σχήμα Δυναμικού Προγραμματισμού που περιγράψαμε στο κεφάλαιο αυτό η διαφορά της υπολογιστικής πλοκής του αλγορίθμου του Lawler από αυτόν των Schrage-Baker, που στη γενική περίπτωση χωρίς χρόνους εξάρμωσης που εξετάσαμε στο προηγούμενο κεφάλαιο έχει την μικρότερη πλοκή, περιορίζεται σε δευτερεύοντες όρους των αντίστοιχων εκφράσεων.

Ο αλγόριθμος που υλοποιήσαμε λειτουργεί σε γενικές γραμμές ως εξής: Αρχικά επαναριθμούνται οι εργασίες κατά αύξουσα σειρά των προθεσμιών και με δευτερεύον κριτήριο τους χρόνους επεξεργασίας (σε αύξουσα επίσης σειρά). Αυτό διευκολύνει πολύ το δεύτερο βήμα που είναι η εξαγωγή των κανόνων προτεραιότητας. Οι κανόνες αυτοί προκύπτουν με τη χρήση των δύο θεωρημάτων που αναφέραμε στο δεύτερο κεφάλαιο.

Μετά την εξαγωγή τους ξεκινάει ο αλγόριθμος Δυναμικού Προγαμματισμού που προχωράει επαναληπτικά από το ένα στάδιο στο επόμενο. Ξεκινώντας μία επανάληψη έχουν αποθηκευτεί οι πληροφορίες που σχετίζονται με καταστάσεις που αντιστοιχούν σε σύνολα ενός συγκεκριμένου πληθάριθμου, έστω κ. Συγκεκριμένα, για κάθε σύνολο έχει δημιουργηθεί μία δομή που περιέχει το χαρακτηριστικό του διάνυσμα, B δείκτες, ένα για κάθε ομάδα, απ’ όπου ξεκινάνε λίστες καταστάσεων (δηλαδή κάποιων άλλων δομών που αντιπροσωπεύουν καταστάσεις) που αντιστοιχούν στο συγκεκριμένο σύνολο και ομάδα και διαφέρουν στην τρίτη μεταβλητή κατάστασης που δηλώνει το χρόνο αποπεράτωσης της τελευταίας εργασίας της αντίστοιχης διάταξης και, τέλος, η απαραίτητη πληροφορία για την αποθήκευση της δομής, που στην περίπτωσή μας που οι εν λόγω δομές αποθηκεύονται σε ισοζυγισμένα δυαδικά δένδρα είναι δύο δείκτες προς τα ισάριθμα παιδιά-υποδένδρα της δομής (που αποτελεί πλέον κόμβο σ’ ένα δένδρο) και μία ένδειξη του εάν και ποιό από τα δύο παιδιά-υποδένδρα είναι κατά ένα επίπεδο βαθύτερο του άλλου (που είναι απαραίτητη ώστε κατά την εγγραφή νέων δομών στο δένδρο αυτό να παραμένει ισοζυγισμένο).

Στις δομές που αντιπροσωπεύουν καταστάσεις έχουν αποθηκευτεί η μεταβλητή κατάστασης t που δηλώνει τη στιγμή αποπεράτωσης της αντίστοιχης διάταξης (οι άλλες δύο μεταβλητές κατάστασης, R και α, υποδηλώνονται έμμεσα από το σε ποιά από τις B λίστες (η δεύτερη) της δομής που αντιστοιχεί σε ποιό σύνολο (η πρώτη) βρίσκεται η κατάσταση), η τιμή της συνάρτησης f στην παρούσα κατάσταση (δηλαδή η τιμή του $\sum C_i$ της αντίστοιχης διάταξης), η μέγιστη βραδύτητα της αντίστοιχης διάταξης, ένας δείκτης στο επόμενο στοιχείο της λίστας και η απαραίτητη πληροφορία για την ανακατασκευή της βέλτιστης λύσης.

Όπως είπαμε στο προηγούμενο κεφάλαιο για να είναι η απαιτούμενη μνήμη ανάλογη του SMAX (όπως αυτό ορίστηκε εκεί) και όχι του M (του αριθμού των εφικτών συνόλων) θα πρέπει είτε να χρησιμοποιηθεί δευτερεύουσα μνήμη για την αποθήκευση των πληροφοριών που σχετίζονται με τα εφικτά σύνολα είτε να χρησιμοποιηθεί η μέθοδος των Munro και Ramirez σύμφωνα με την οποία κάθε κατάσταση από το μεσαίο στάδιο κι έπειτα, έχει αποθηκευμένη την πληροφορία του ποιά κατάσταση του μεσαίου σταδίου βρίσκεται στο βέλτιστο μονοπάτι μέχρι τη συγκεκριμένη κατάσταση. Για τους λογους που είχαμε αναφέρει στο κεφάλαιο εκείνο (κυρίως τη συνεπαγόμενη καθυστέρηση από τη χρησιμοποίηση δευτερεύουσας μνήμης) επιλέξαμε τη χρησιμοποίηση της αναδρομικής μεθόδου των Munro-Ramirez. Έτσι, η πληροφορία για την ανακατασκευή της βέλτιστης λύσης που αποθηκεύεται σε κάθε κατάσταση από το μεσαίο στάδιο και έπειτα απαρτίζεται από τις τρεις μεταβλητές κατάστασης (τα σύνολο αντιπροσωπεύεται από το χαρακτηριστικό του διάνυσμα) της κατάστασης που βρίσκεται στο μεσαίο στάδιο (αυτό δηλαδή, με πληθάριθμο των αντίστοιχων συνόλων $\lfloor n/2 \rfloor$) του βέλτιστου μονοπατιού από την αρχή μέχρι τη συγκεκριμένη κατάσταση.

Όταν, λοιπόν, ο αλγόριθμος είναι έτοιμος να ξεκινήσει μία νέα επανάληψη με στόχο τη δημιουργία των καταστάσεων (και τον υπολογισμό σ’ αυτές της συνάρτησης f) του σταδίου κ+1, έχει, από την προηγούμενη επανάληψη, δημιουργηθεί ένα ισοζυγισμένο δένδρο κάθε κόμβος του οποίου αντιστοιχεί σ’ ένα εφικτό σύνολο πληθάριθμου κ. Από κάθε τέτοιο κόμβο ξεκινάνε B (μία για κάθε ομάδα) γραμμικές λίστες καταστάσεων, οι οποίες (καταστάσεις) είναι ταξινομημένες κατά αύξουσα σειρά της τρίτης μεταβλητής κατάστασης t, που είναι ουσιαστικά το C_{max} της αντίστοιχης διάταξης, επομένως, σύμφωνα με την πρόταση 2.5, είναι ταξινομημένες κατά φθίνουσα σειρά του $\sum C_i$. Γιατί εάν για δύο διατάξεις π_1 και π_2 των ίδιων εργασιών που τελειώνουν σε εργασίες της ίδιας ομάδας ισχύει $C_{max}(\pi_1) \leq C_{max}(\pi_2)$ και $\sum C_i(\pi_1) < \sum C_i(\pi_2)$ τότε η π_1 κυριαρχεί της π_2 και δεν υπάρχει λόγος να συνεχιστεί η έρευνα σε διατάξεις που προκύπτουν συμπληρώνοντας την π_2, οπότε η αντίστοιχη κατάσταση μπορεί να διαγραφεί. Γι’ αυτού του είδους τα "κλαδέματα" θα μιλήσουμε αναλυτικότερα λίγο παρακάτω.

Για κάθε εφικτό σύνολο πληθάριθμου κ, που αντιστοιχεί σε κάποιο κόμβο του ισοζυγισμένου δένδρου, υπολογίζεται αρχικά ένα κάτω φράγμα της μέγιστης βραδύτητας, L_{max}, που μπορεί να εμφανιστεί σε οποιαδήποτε διάταξη των αδιάτακτων εργασιών, διατάσσοντάς τες κατά αύξουσα σειρά των προθεσμιών και αγνοώντας τους χρόνους εξάρμωσης. Έπειτα για κάθε εργασία i που δεν ανήκει στο σύνολο και δεν υπάρχει κάποιος προγονός της που δεν ανήκει σε αυτό διατρέχονται οι b γραμμικές λίστες καταστάσεων. Για κάθε κατάσταση η διάταξη π_2 που προκύπτει προσθέτοντας την εργασία j στο τέλος της διάταξης π_1 που αντιστοιχεί στην κατάσταση αντιστοιχεί σε μια υποψήφια κατάσταση του επόμενου σταδίου. Το εάν θα αποθηκευτεί η νέα αυτή κατάσταση στην κατάλληλη λίστα του κατάλληλου συνόλου του επόμενου σταδίου ή αν θα απορριφθεί εξαρτάται από δύο παράγοντες.

Ο ένας είναι το κάτω φράγμα της μέγιστης βραδύτητας στη συνέχεια της π_1, που προκύπτει προσθέτοντας το χρόνο έναρξης της εργασίας j όταν αυτή τοποθετηθεί αμέσως μετά την π_1 στο κάτω φράγμα που είπαμε ότι υπολογίστηκε αρχικά. Εάν το προκύπτον κάτω φράγμα είναι θετικό η νέα κατάσταση απορρίπτεται, γιατί δεν υπάρχει εφικτό πρόγραμμα (δηλαδή πρόγραμμα που να σέβεται τις προθεσμίες) που να ξεκινάει με τη διάταξη π_2 (δηλαδή την π_1 και αμέσως μετά την εργασία j). Το τι ακριβώς συμβαίνει τότε εξαρτάται από το εάν θετική βραδύτητα εμφανίζεται σε εργασία που προηγείται της j στην κατά EDD διάταξη, που έπεται αυτής ή στην ίδια τη j. Ό,τι από τα τρία και να συμβαίνει η νέα κατάσταση απορρίπτεται και δεν ελέγχονται οι υπόλοιπες καταστάσεις της ίδιας λίστας για την εργασία j, γιατί αντιστοιχούν σε διατάξεις με μεγαλύτερο C_{max} οπότε και σε εκείνες η μέγιστη βραδύτητα θα είναι θετική. Εάν όμως συμβαίνει το δεύτερο ή το τρίτο τότε επιπλέον διαγράφεται το τμήμα της λίστας καταστάσεων από την παρούσα και έπειτα γιατί όχι μόνο για την ίδια εργασία j αλλά και για οποιαδήποτε άλλη j' δεν υπάρχει εφικτό πρόγραμμα που να ξεκινάει με τη διάταξη π_1 και να ακολουθεί η j'. Οι κατάλληλες επίσης ενέργειες γίνονται και στην περίπτωση που το κάτω φράγμα της μέγιστης βραδύτητας είναι ακριβώς μηδέν.

Εάν δεν εμφανιστεί πρόβλημα εφικτότητας γίνεται ένας δεύτερος έλεγχος για να διαπιστωθεί εάν έχει νόημα η αποθήκευση της νέας κατάστασης που αντιστοιχεί στη διάταξη π_2. Ο έλεγχος αυτός αποτελεί υλοποίηση της πρότασης 2.5 από την οποία προκύπτει ότι όταν για δύο διατάξεις σ_1 και σ_2 των ίδιων εργασιών που τελειώνουν με εργασία της ίδιας ομάδας ισχύει (1) $C_{max}(\sigma_1) \le C_{max}(\sigma_2)$ και (2) $\sum C_i(\sigma_1) + [n-(k+1)] \cdot C_{max}(\sigma_1) \le \sum C_i(\sigma_2) + [n-(k+1)] \cdot C_{max}(\sigma_2)$ (υπενθυμίζουμε ότι οι καταστάσεις που δημιουργούνται αντιστοιχούν σε διατάξεις με κ+1 εργασίες) η σ_1 κυριαρχεί της σ_2 στο $1/s_{ij}/\sum C_i | T_{max}=0$ (δηλαδή για κάθε διάταξη σ_0 των $n-(k+1)$ αδιάτακτων εργασιών για την οποία το $\sigma_2\sigma_0$ είναι εφικτό πρόγραμμα ισχύει $\sum C_i(\sigma_1\sigma_0) \le \sum C_i(\sigma_2\sigma_0)$). Επειδή, όπως έχουμε πει, ανζητούμε μεταξύ των βέλτιστων λύσεων του $1/s_{ij}/\sum C_i | T_{max}=0$ εκείνη με το μικρότερο L_{max}, εάν η σχέση (2) ισχύει με ισότητα θα πρέπει να ισχύει επιπλέον (3) $L_{max}(\sigma_1) \le L_{max}(\sigma_2)$ (αφού η σχέση μεταξύ των δύο μελών της (2) μεταφέρεται, όπως απλά προκύπτει από την απόδειξη της προκείμενης πρότασης, και μεταξύ των $\sum C_i(\sigma_1\sigma_0)$ και $\sum C_i(\sigma_2\sigma_0)$.

Έτσι λοιπόν, εξετάζοντας εάν και που θα αποθηκευτεί η κατάσταση που αντιστοιχεί στη διάταξη π_2 διατρέχεται η λίστα καταστάσεων που αντιστοιχει στην ομάδα της εργασίας j και στο σύνολο των εργασιών που περιέχονται στην π_2 (εάν έχει προηγουμένως δημιουργηθεί μία δομή που να αντιστοιχεί στο σύνολο αυτό - εάν όχι δημιουργείται τώρα και εισάγεται κατάλληλα στο δένδρο των δομών του συνόλου πληθάριθμου κ+1 ώστε αυτό να μείνει ισοζυγισμένο) μέχρι να βρεθεί μία κατάσταση με μεγαλύτερο C_{max} από την π_2 (να υπενθυμίσουμε ότι οι καταστάσεις είναι ταξινομημένες κατά αύξουσα σειρά του C_{max}). Μέχρι να βρεθεί η πρώτη τέτοια κατάσταση ελέγχουμε αν η διάταξη που αντιστοιχεί σε κάποια από τις καταστάσεις που περνάμε κυριαρχεί της π_2, με βάση την παραπάνω πρόταση. Εάν βρεθεί κάποια τέτοια, η κατάσταση που αντιστοιχεί στην π_2 δεν αποθηκεύεται. Διαφορετικά εισέρχεται στη λίστα και γίνεται ένας νέος έλεγχος να διαπιστωθεί μήπως η π_2, τώρα, κυριαρχεί σε ορισμένες από τις διατάξεις που αντιστοιχούν σε επόμενες (με μεγαλύτερο C_{max}) καταστάσεις. Διατρέχοντας και πάλι τη λίστα από το σημείο που εισήλθε η αντίστοιχη κατάσταση της π_2 κι έπειτα όσο συναντώνται καταστάσεις οι αντίστοιχες διατάξεις των οποίων κυριαρχούνται από την π_2, αυτές διαγράφονται. Όταν βρεθεί η πρώτη κατάσταση που δεν διαγράφεται, ο έλεγχος σταματάει, εφόσον οι ποσότητες $\sum C_i + [n-(k+1)] \cdot C_{max}$ φθίνουν διατρέχοντας τη λίστα (αφού τα C_{max} αυξάνουν και δεν είναι δυνατόν να αυξάνουν και τα δύο γιατί τότε κάποιες καταστάσεις θα είχαν διαγραφεί).

Η πρόταση 2.5 έχει αποδειχθεί για τη γενικότερη περίπτωση όπου οι δύο μερικές διατάξεις, σ_1 και σ_2, δεν τελειώνουν αναγκαστικά με εργασίες της ίδιας ομάδας. Υλοποιήθηκε και η γενικότερη αυτή εκδοχή, ελέγχοντας δηλαδή την ισχύ των σχέσεων (1), (2) (γενικευμένων, όπως δίνονται στην παράγραφο 2.4), και (3) μεταξύ της νεοσχηματιζόμενης διάταξης και των ήδη υπαρχουσών όπως πριν με τη διαφορά ότι ο έλεγχος επεκτεινόταν και στις λίστες καταστάσεων των άλλων ομάδων. Εύκολα παρατηρεί κανείς ότι είναι πιο δύσκολο (δηλαδή λιγότερο πιθανό) να ισχύουν οι (γενικευμένες) σχέσεις (1) και (2) για διατάξεις που τελειώνουν σε εργασίες διαφορετικών ομάδων αφού στο $C_{max}(\sigma_1)$ προστίθεται κάποιος χρόνος εξάρμωσης. Γι' αυτό και οι διαγραφές καταστάσεων που αναμένονται από τέτοιες συγκρίσεις μερικών διατάξεων είναι λιγότερες. Πράγματι, ο χρόνος εκτέλεσης του προγράμματος αυξήθηκε όταν εισάγαμε και τέτοιες συγκρίσεις αντί να μειωθεί, που δείχνει ότι οι επιπλέον διαγραφές καταστάσεων δεν ήταν αρκετές ώστε το κέρδος από τη συνεπαγόμενη ελάττωση του χώρου έρευνας να είναι σημαντικότερο από τον επιπρόσθετο υπολογιστικό φόρτο που εισάγουν αυτές οι συγκρίσεις. Γι' αυτό και στην τελική μορφή του προγράμματος τέτοιες συγκρίσεις δεν γίνονται.

Αφού ολοκληρωθούν οι έλεγχοι που σχετίζονται με μία κατάσταση του σταδίου κ (έλεγχοι που μπορεί να έχουν σαν αποτέλεσμα την αποθήκευση μιας νέας κατάστασης στο στάδιο κ+1 και τη διαγραφή κάποιων ήδη αποθηκευμένων καταστάσεων) γίνεται η ίδια δουλειά για την επόμενη κατάσταση της ίδιας λίστας (εκτός και αν η διάταξη π_2, που προέκυψε από την προσθήκη της εργασίας j - για την οποία διατρέχονται οι b λίστες καταστάσεων κάποιου εφικτού συνόλου πληθάριθμου κ - στο τέλος της διάταξης π_1 που αντιστοιχεί στην εν λόγω κατάσταση, παρουσίασε θετική βραδύτητα οπότε περνάμε στην επόμενη λίστα αφού η προσθήκη της εργασίας j στις διατάξεις που αντιστοιχούν στις επόμενες καταστάσεις θα προκαλέσει πάλι θετική βραδύτητα αφού οι καταστάσεις είναι ταξινομημένες κατά αύξον C_{max}). Μόνο που τώρα ο έλεγχος γαι την ισχύ της πρότασης 2.5 θα συνεχιστεί από το σημείο (της κατάλληλης λίστας καταστάσεων του κατάλληλου συνόλου εργασιών) που είχε σταματήσει όταν γινόταν για την προηγούμενη κατάσταση.

Όταν για κάποιο σύνολο διατρέξουμε τις b λίστες καταστάσεών του για όλες τις εργασίες που δεν ανήκουν σ’ αυτό και δεν έχουν κάποιο πρόγονο που να μην ανήκει σ’ αυτό, δημιουργώντας έτσι μερικές από τις καταστάσεις του επόμενου σταδίου, η δομή που κρατάει τη σχετική με το σύνολο αυτό πληροφορία και οι λίστες καταστάσεων που ξεκινούν απ’ αυτή διαγράφονται για οικονομία μνήμης. Όταν γίνει η δουλειά αυτή για κάθε σύνολο του σταδίου κ, το ισοζυγισμένο δένδρο του οποίου οι κόμβοι αντιστοιχούσαν στα εφικτά σύνολα πληθάριθμου κ έχει διαγραφεί πλήρως ενώ έχει δημιουργηθεί ένα αντίστοιχο δένδρο για το στάδιο κ+1.

΄Ετσι προχωράει ο αλγόριθμος επαναληπτικά μέχρι να κατασκευασθεί το ισοζυγισμένο δένδρο του σταδίου n το οποίο περιέχει μόνο έναν κόμβο, αφού μόνο ένα σύνολο, το πλήρες, υπάρχει με n εργασίες. Οι b λίστες που ξεκινούν από τη δομή που αντιστοιχεί στο σύνολο αυτό περιέχουν καταστάσεις που αντιστοιχούν σε πλήρη προγράμματα. Οι καταστάσεις αυτές είναι, σε καθε λίστα, ταξινομημένες κατά αύξουσα σειρά του C_{max} και κατά φθίνουσα σειρά του $\sum C_i$. Επομένως η βέλτιστη λύση του προβλήματος δίνεται από τη διάταξη που αντιστοιχεί στην κατάσταση εκείνη που από τις b καταστάσεις που είναι τελευταίες στις ισάριθμες λίστες έχει το μικρότερο $\sum C_i$ (λύνοντας, όπως έχουμε, πει τις πιθανές ισοπαλίες διαλέγοντας το πρόγραμμα με το μικρότερο L_{max} που είναι επίσης αποθηκευμένο στη δομή που αντιστοιχεί στην κάθε κατάσταση).

Βέβαια το ποιά είναι η διάταξη αυτή δεν είναι αποθηκευμένο στη δομή που αντιπροσωπεύει την αντίστοιχη κατάσταση γιατί υλοποιώντας το πρόγραμμα αυτό δεν ήμασταν σίγουροι ότι η μνήμη θα αποδικνύετο αρκετή, όχι μόνο για την αποθήκευση σε κάθε κατάσταση της αντίστοιχης διάταξης (κάτι που θα οδηγούσε σε πολλαπλασιασμό του μεγέθους των δομών που αντιπροσωπεύουν καταστάσεις), αλλά ακόμα και για την τελική μορφή της υλοποίησης που απαιτεί σημαντικά λιγότερη μνήμη αφού δεν γίνεται κάτι τέτοιο. Για τον ίδιο ακριβώς λόγο οι καταστάσεις που σχετίζονται με κάποιο σύνολο διαγράφονται όταν δημιουργηθούν οι καταστάσεις του επόμενου σταδίου που προκύπτουν από αυτές (με την προσθήκη στο τέλος της αντίστοιχης διάταξης μιας εργασίας), μην επιτρέποντας έτσι την ανακατασκευή της βέλτιστης λύσης γυρνώντας, αφού τελειώσει ο Δυναμικός Προγραμματισμός, στάδιο-στάδιο προς τα πίσω.

΄Ετσι επιλέξαμε, όπως έχουμε ξαναπεί, τη μέθοδο των Munro-Ramirez σύμφωνα με την οποία σε κάθε κατάσταση αποθηκεύεται το ποια είναι η κατάσταση του μεσαίου σταδίου που βρίσκεται στο βέλτιστο μονοπάτι προς αυτήν και το πρόβλημα λύνεται αναδρομικά. ΄Ετσι λοιπόν στην κατάσταση που αντιστοιχεί στη βέλτιστη λύση, στο δικό μας πρόβλημα, είναι αποθηκευμένη, αντί της βέλτιστης λύσης, η πληροφορία του ποια είναι η κατάσταση του μεσαίου σταδίου στο βέλτιστο μονοπάτι, δηλαδη ποιές εργασίες βρίσκονται στις πρώτες μισές θέσεις (αλλά όχι με ποιά σειρά), σε ποια ομάδα ανήκει η τελευταία από αυτές και πότε η εκτέλεση αυτής αποπερατώνεται. Για να ολοκληρωνόταν η συγκεκριμένη υλοποίηση θα έπρεπε (εάν επαληθεύαμε στην πράξη ότι η μνήμη δεν είναι αρκετή για να αποθηκεύονται όλες οι καταστάσεις ταυτόχρονα ή να αποθηκεύεται σε κάθε κατάσταση η αντίστοιχη διάταξη), όταν όλα αυτά θα ήταν γνωστά, θα είχε δηλαδή τελειώσει ο Δυναμικός Προγραμματισμός, να καλούμε την όλη διαδικασία αναδρομικά για το πρώτο και δεύτερο μισό βρίσκοντας έτσι μία-μία τις καταστάσεις του βέλτιστου μονοπατιού και επομένως τη λύση του προβλήματος. ΄Ετσι κι αλλιώς, αφού ο αλγόριθμος έχει εκθετική πλοκή, όπως θα πούμε αμέσως παρακάτω, ο χρόνος κάθε άλλης εκτέλεσης της αναδρομής αναμενόταν πολύ μικρότερος αυτού της πρώτης δηλαδή αυτή θα καθόριζε τη συνολική ταχύτητα του προγράμματος.

Για να υπολογίσουμε την πλοκή του αλγορίθμου παρατηρούμε ότι για καθένα από τα M (που είναι της τάξης του 2^n) εφικτά σύνολα και καθεμία από τις $O(n)$ εργασίες που δεν ανήκουν σε αυτό και είτε δεν έχουν προγόνους είτε όλοι οι πρόγονοί τους ανήκουν σ' αυτό, αρχικά αναζητείται σ' ένα ισοζυγισμένο δένδρο με κόμβους τα εφικτά σύνολα ενός συγκεκριμένου πληθάριθμου η δομή που αντιπροσωπεύει το σύνολο που προκύπτει από την προσθήκη της συγκεκριμένης εργασίας στο συγκεκριμένο σύνολο. Το πλήθος των κόμβων του δένδρου φράσσεται από το 2^n και αφού η αναζήτηση και η εισαγωγή ενός νέου κόμβου σ' ένα ισοζυγισμένο δένδρο έχει λογαριθμική πλοκή ο αριθμός των συγκρίσεων μεταξύ χαρακτηριστικών διανυσμάτων που απαιτείται είναι $O(n)$ κι επειδή, όπως αναλύσαμε στο προηγούμενο κεφάλαιο, η σύγκριση δύο τέτοιων διανυσμάτων έχει πλοκή $O(n)$ η όλη αναζήτηση έχει πλοκή $O(n^2)$. Αφού βρεθεί ή, αν δεν υπάρχει, εισαχθεί, η κατάλληλη δομή στο δένδρο του επόμενου σταδίου, διατρέχονται οι b λίστες καταστάσεων του αρχικού συνόλου (πληθάριθμου κ) και για κάθε κατάσταση πιθανώς να δημιουργηθεί μία νέα κατάσταση στην κατάλληλη λίστα του νέου συνόλου (πληθάριθμου κ+1) και πιθανώς να διαγραφούν μερικές από τις ήδη υπάρχουσες. Για κάθε λίστα του αρχικού συνόλου, που διατρέχεται μία φορά, διατρέχεται και η κατάλληλη λίστα του νέου συνόλου επίσης μία φορά, εφόσον ο έλεγχος για την ισχύ της πρότασης 2.5 συνεχίζεται για κάθε κατάσταση του αρχικού συνόλου από το σημείο της κατάλληλης λίστας του νέου συνόλου που είχε σταματήσει για την προηγούμενη κατάσταση. Επειδή ακριβώς τα στοιχεία των λιστών αυτών διατρέχονται ένα-ένα όπως αυτά είναι ταξινομημένα δεν μπορεί να υπάρξει πιο συμφέρων, από άποψη πλοκής, τρόπος αποθήκευσης τους από τις γραμμικές λίστες. Η πλοκή, λοιπόν, της δημιουργίας (ή της διαγραφής) νέων καταστάσεων που προκύπτουν από την προσθήκη μίας εργασίας στις διατάξεις που αντιστοιχούν στις καταστάσεις μιας λίστας του αρχικού συνόλου είναι $O(T)$ όπου T είναι, όπως έχουμε πει, το πλήθος των διαφορετικών τιμών που μπορει να πάρει το C_{max} ενός πλήρους προγράμματος (δηλαδή ένα άνω φράγμα στο μήκος κάθε λίστας καταστάσεων) κι επομένως για όλες τις λίστες καταστάσεων η πλοκή είναι $O(b{\cdot}T)$. Η συνολική, λοιπόν, πλοκή του αλγορίθμου είναι $O(M{\cdot}n{\cdot}(n^2+b{\cdot}T))$, όπου το M είναι της τάξης του 2^n. Αυτή είναι, βέβαια, η πλοκή της πρώτης αναδρομής όταν χρησιμοποιείται η μέθοδος των Munro-Ramirez, αλλά επειδή στις υπόλοιπες αναδρομές απλώς αντικαθίσταται το n από $n/2$, $n/4$ κ.ο.κ και η πλοκή είναι, λόγω του M, εκθετική, αυτός είναι ο κυρίαρχος όρος στην έκφραση της πλοκής του ολοκληρωμένου αλγορίθμου, επομένως αποτελεί πλοκή και αυτού.

Για συνηθισμένες και "φυσιολογικές" τιμές των n, b, $\bar{S}$ το n^2 είναι αρκετά μικρότερο του $b{\cdot}T$ (παίρνοντας το T σαν το στενότερο άνω φράγμα που βρήκαμε γι' αυτό, δηλαδή $(n-b){\cdot}\bar{S}$) οπότε η πλοκή εστιάζεται στο $M{\cdot}n{\cdot}b{\cdot}T$. Άλλωστε, όπως εξηγήσαμε στο προηγούμενο κεφάλαιο, ο αριθμός των πράξεων που απαιτούνται για τη σύγκριση δύο χαρακτηριστικών διανυσμάτων είναι $\lfloor n/w \rfloor$ (όπου w το μήκος της λέξης του υπολογιστή) δηλαδή πολύ μικρότερος από n, οπότε η αναζήτηση ενός συνόλου στο αντίστοιχο δένδρο απαιτεί πολύ λιγότερες πράξεις από n^2. Θεωρητικά όμως, που η αναζήτηση ενός συνόλου σ' ένα ισοζυγισμένο δένδρο έχει πλοκή $O(n^2)$ και που πρέπει να καλύπτουμε όλες τις περιπτώσεις τιμών των παραμέτρων, η πλοκή δίνεται από τον παραπάνω τύπο. Όσον αφορά στην απαιτούμενη μνήμη, εφόσον οι αποθηκευμένες καταστάσεις δεν αντιστοιχούν σε περισσότερα από $SMAX$ σύνολα και σε κάθε σύνολο δεν αντιστοιχούν περισσότερες από $b \cdot T$ καταστάσεις, η μνήμη αυτή είναι $O(SMAX \cdot b \cdot T)$.

Έχουμε πει ότι ο κυριότερος λόγος που προτιμήσαμε τον αλγόριθμο του Lawler για την δημιουργία και την αποθήκευση των εφικτών συνόλων ήταν ότι αυτός απαιτούσε τη λιγότερη μνήμη από όσους έχουν περιέλθει στην αντίληψή μας και εξετάσαμε στο προηγούμενο κεφάλαιο, σε συνδυασμό βέβαια με το ότι σχεδόν πάντα οι αλγόριθμοι Δυναμικού Προγραμματισμού που παρουσιάζονται σε διάφορες εργασίες αντιμετωπίζουν προβλήματα μνήμης. Εκτελώντας όμως το πρόγραμμα που υλοποιούσε τον παραπάνω αλγόριθμο (συγκεκριμένα την πρώτη αναδρομή της μεθόδου των Munro-Ramirez - τις υπόλοιπες σκοπεύαμε να τις υπολοιήσουμε αμέσως μετά) διαπιστώσαμε ότι ο χρόνος ήταν πιο περιοριστικός από την μνήμη. Συγκεκριμένα καθώς το πλήθος των εργασιών και των ομάδων αύξανε ο χρόνος (CPU time) του ενός λεπτού που είχαμε θέσει σαν όριο μεταξύ των παραδεκτών και των μη παραδεκτών χρόνων ξεπερνιόταν ενώ ακόμα η διαθέσιμη μνήμη των 64 Mbytes δεν είχε εξαντληθεί. Όπως είπαμε και νωρίτερα στο κεφάλαιο αυτό, το φαινόμενο αυτό μπορεί να ερμηνευτεί αφενός από τη μεγαλύτερη δυσκολία του προβλήματος αυτού έναντι άλλων που είχαν λυθεί με Δυναμικό Προγραμματισμό (κάτι που επιβεβαιώνεται και από τα αποτελέσματα των Ahn-Hyun που, για το ευκολότερο από το δικό μας πρόβλημα, όπως φαίνεται και από την πλοκή του καθενός, $1/S_{ij}/\sum C_i$, εμφανίζουν πολύ μεγαλύτερους χρόνους εκτέλεσης του προγράμματός τους από άλλες εργασίες που ασχολούνται με άλλα προβλήματα - χρόνους που επίσης ξεπερνούν το ένα λεπτό προτού εξαντληθεί η μνήμη) και αφετέρου από τη μεγαλύτερη μνήμη έναντι όλων των άλλων που είχαμε στη διαθεσή μας.

5.4. Απαρίθμηση των εφικτών συνόλων με λεξικογραφική σειρά

Διαπιστώνοντας ότι το πρόβλημα είναι ο χρόνος και όχι η μνήμη, αποφασίσαμε να μην ολοκληρώσουμε το πρόγραμμα που υλοποιούσε τον παραπάνω αλγόριθμο συμπληρώνοντας το τμήμα της ανακατασκευής της βέλτιστης λύσης που έλειπε (είτε ολοκληρώνοντας την αναδρομική διαδικασία των Munro-Ramirez, είτε - αφού μνήμη υπήρχε - αποθηκεύοντας σε κάθε κατάσταση την αντίστοιχη διάταξη, είτε μη διαγράφοντας τα περασμένα στάδια), αλλά να υλοποιήσουμε τον αλγόριθμο δημιουργίας και αποθήκευσης των εφικτών συνόλων με τη μικρότερη πλοκή που όπως είπαμε στο προηγούμενο κεφάλαιο, στη γενική περίπτωση χωρίς όμως χρόνους εξάρμωσης, είναι εκείνος των Schrage και Baker. Στο δικό μας πρόβλημα η διαφορά στην πλοκή είναι, όπως θα δούμε, πιο μικρή (λιγότερο σημαντική) αλλά και πάλι υπαρκτή. Η απαιτούμενη μνήμη είναι βέβαια μεγαλύτερη αλλά ελπίσαμε, όπως πράγματι έγινε, ότι η διαθέσιμη θα αποδεικνυόταν και πάλι αρκετή.

Αρχικά υλοποιήσαμε την παραλλαγή των Kao-Queyranne που χρησιμοποιεί κυκλικά τον πίνακα αποθήκευσης των συνόλων απαιτώντας έτσι μνήμη ανάλογη του μέγιστου label που ανατίθεται σε κάποια εργασία, LMAX, αντί του αθροίσματος των labels, LSUM, όπως γίνεται στον αυθεντικό αλγόριθμο των Schrage-Baker. Το τίμημα γι' αυτό είναι ότι η ανακατασκευή της βέλτιστης λύσης, στο τέλος του Δυναμικού Προγραμματισμού, δεν είναι απλή (όπως, δηλαδή, και στον αλγόριθμο του Lawler) χωρίς όμως, όπως έχουμε πει, να αυξάνεται η υπολογιστική πλοκή κι ότι χάνεται ακόμα κάποιος χρόνος για να διαγράφονται οι λίστες των καταστάσεων που ξεκινούν από θέσεις του πίνακα που θα ξαναχρησιμοποιηθούν για άλλα σύνολα. Προτιμήσαμε όμως να υλοποιήσουμε την παραλλαγή αυτή λόγω της (συνηθισμένης) αβεβαιότητας μας για την επάρκεια της μνήμης που ενισχυόταν από το γεγονός ότι η απαιτούμενη μνήμη για τον αλγόριθμο των Schrage-Baker αναμενόταν αρκετά μεγαλύτερη από εκείνη του αλγορίθμου του Lawler εφόσον η μνήμη που χρησιμοποιείται για αποθήκευση πληροφοριών σχετικών με τα εφικτά σύνολα είναι ανάλογη του LSUM αντί του πολύ μικρότερου (σύμφωνα με τον Steiner [Stei-90]), SMAX που ήταν πριν και εκείνη που χρησιμοποείται για αποθήκευση καταστάσεων είναι ανάλογη του M αντί και πάλι του αρκετά μικρότερου SMAX.

Έτσι κι αλλιώς, αν εξαιρέσουμε το θέμα της ανακατασκευής της βέλτιστης λύσης, οι δύο αλγόριθμοι (ο αυθεντικός του Schrage-Baker και η παραλλαγή των Kao-Queyranne) είναι σχεδόν ίδιοι. Η μόνη ουσιαστική διαφορά είναι ότι στον πρώτο τα σύνολα αποθηκεύονται στη θέση του πίνακα που δηλώνεται από τη διεύθυνσή τους (δηλαδή, το άθροισμα των labels των εργασιών που περιέχουν) ενώ στο δεύτερο αποθηκεύονται στη θέση που δηλώνεται από το υπόλοιπο της διαίρεσης της διεύθυνσής τους με το LMAX. Κι επειδή στον αλγόριθμο των Kao-Queyranne τα σύνολα πρέπει να απαριθμούνται, όπως έχουμε πει, με αύξουσα σειρά των διευθύνσεων τους, ενώ σε εκείνον των Schrage-Baker δεν υπάρχει ανάλογος περιορισμός, είναι πολύ απλό να τροποποιήσει κανείς μια υλοποίηση του πρώτου λαμβάνοντας μια υλοποίηση του δεύτερου (ενώ το αντίστροφο είναι λιγότερο απλό για το λόγο που αναφέραμε).

Έτσι λοιπόν, τροποποιήσαμε το αρχικό πρόγραμμα που δημιουργούσε και αποθήκευε τα εφικτά σύνολα με βάση τον αλγόριθμο του Lawler έτσι ώστε να χρησιμοποιεί για το σκοπό αυτό τον αλγόριθμο των Kao-Queyranne. Όπως ήταν φυσικό το τμήμα του προγράμματος που ασχολείται με την επεξεργασία των λιστών καταστάσεων (δημιουργία νέων καταστάσεων, διαγραφή ορισμένων είτε λόγω μη εφικτότητας είτε λόγω κυριαρχίας από άλλες) έμεινε το ίδιο. Αυτό που άλλαξε ήταν ο τρόπος δημιουργίας και αποθήκευσης συνόλων εργασιών.

Συγκεκριμένα, αφού υπολογιστούν οι σχέσεις προτεραιότητας μεταξύ των εργασιών, ανατίθεται ένα label σε κάθε εργασία. Για να μπορεί η μνήμη να χρησιμοποιηθεί κυκλικά θα πρέπει τα εφικτά σύνολα να απαριθμούνται με αύξουσα σειρά των διευθύνσεών τους. Οι Kao-Queyranne προτείνουν, όπως έχουμε πει, έναν αλγόριθμο πλοκής $O(LSUM \cdot n^2)$ που για ένα σύνολο εργασιών με τα αντιστοιχα labels απαριθμεί τα εφικτά σύνολα με τη σειρά που τα θέλουμε. Προτείνουν επίσης έναν ευρηματικό κανόνα ανάθεσης labels σε εργασίες με στόχο την ελαχιστοποίηση του LMAX, από το οποίο εξαρτάται το μέγεθος του πίνακα αποθήκευσης των σχετικών με τα εφικτά σύνολα πληροφοριών. Παρατηρούν, τέλος, ότι αν οι εργασίες έχουν αριθμηθεί με τρόπο ώστε κάθε μια να έπεται όλων των προγόνων της και τα labels ανατίθενται στις εργασίες με τη σειρά αυτή τότε, εάν τα εφικτά σύνολα απαριθμούνται με λεξικογραφική σειρά, θα απαριθμούνται και με αύξουσα σειρά των διευθύνσεών τους. Επειδή η λεξικογραφική απαρίθμηση των συνόλων γίνεται σε $O(M \cdot n^2)$ (που είναι αρκετά μικρότερο του $O(LSUM \cdot n^2)$ αφού το M, ο αριθμός των εφικτών συνόλων, είναι αρκετά μικρότερο του LSUM) χρησιμοποιήσαμε την παρατήρηση αυτή στη δικιά μας υλοποίηση σε συνδυασμό με τον ευρηματικό αλγόριθμο ανάθεσης labels (σε εργασίες) των Schrage-Baker (οι οποίοι επιδιώκουν ελαχιστοποίηση του LSUM αφού στο δικό τους αλγόριθμο από αυτό εξαρτάται το μέγεθος του πίνακα των συνόλων) σύμφωνα με τον οποίο, label παίρνει κάθε φορά η εργασία που, εάν πάρει εκείνη τη στιγμή, θα πάρει το μικρότερο.

Έτσι λοιπόν, δώσαμε labels στις εργασίες (και ταυτόχρονα τις αριθμήσαμε με την ίδια σειρά) έτσι ώστε κάθε φορά να παίρνει label αυτή που, από όσες δεν έχουν κάποιο πρόγονο που να μην έχει πάρει ακόμα label (και φυσικά δεν έχουν πάρει οι ίδιες), θα πάρει το μικρότερο, εάν πάρει εκείνη τη στιγμή. Απαριθμώντας έπειτα τα εφικτά σύνολα λεξικογραφικά, μπορούμε να χρησιμοποιούμε τη μνήμη κυκλικά αφού, όπως είπαμε, τα σύνολα θα προκύπτουν με αύξουσα σειρά των διευθύνσεών τους. Το τίμημα για τη μείωση του αριθμού των πράξεων που σχετίζονται με τη απαρίθμηση των συνόλων, που αναφέραμε παραπάνω, είναι ότι ίσως έτσι το LMAX προκύψει μεγαλύτερο (δηλαδή, απαιτηθεί περισσότερη μνήμη) αφού ο ευρηματικός κανόνας ανάθεσης labels σε εργασίες των Kao-Queyranne, που δεν χρησιμοποιήσαμε, είχε προταθεί ακριβώς με στόχο την ελαχιστοποίηση του LMAX. Έτσι κι αλλιώς όμως και πάλι, όπως θα πούμε και παρακάτω, ο χρόνος αποδείχθηκε πιο περιοριστικός από τη μνήμη.

Έχοντας λοιπόν αναθέσει ένα label σε κάθε εργασία ο αλγόριθμος συνεχίζει όπως πριν, μόνο που τώρα, οι δομές που αντιστοιχούν σε σύνολα δεν αποθηκεύονται σε ισοζυγισμένα δένδρα, αλλά κυκλικά σε ένα πίνακα. Η επιλογή του συνόλου στο οποίο θα προσθέσουμε εργασίες που δεν ανήκουν σ' αυτό (ουσιαστικά θα τοποθετήσουμε τις τελευταίες στο τέλος των διατάξεων που αντιστοιχούν σ' αυτό) δεν γίνεται όπως πριν, που διαλέγαμε ένα οποιοδήποτε σύνολο από το δένδρο ενός συγκεκριμένου σταδίου, αλλά γίνεται διαλέγοντας το σύνολο που λεξικογραφικά έπεται αυτού που είχε επιλεγεί αμέσως πριν.

Ένα ζήτημα που δεν έχουμε θίξει, και ίσως τώρα είναι η κατάλληλη στιγμή, σχετίζεται με την απαρίθμηση των συνόλων και τη ροή της πληροφορίας από ένα σύνολο σε ένα υπερσύνολό του με μία εργασία περισσότερη. Μέχρι τώρα θεωρούμε ότι η πληροφορία (στην περίπτωσή μας οι λίστες των καταστάσεων) που σχετίζεται με ένα σύνολο που προκύπτει από την απαρίθμηση έχει ήδη υπολογιστεί και, προσθέτοντας στο σύνολο αυτό εργασίες που δεν ανήκουν σ’ αυτό, ενημερώνουμε τις λίστες καταστάσεων των συνόλων που προκύπτουν από αυτές τις προσθήκες. Για κάθε σύνολο, δηλαδή, που προκύπτει από την απαρίθμηση ενημερώνουμε τα σύνολα που έχουν ακριβώς μία εργασία περισσότερη από αυτό. Η προσέγγιση αυτή χαρακτηρίζεται "reaching"-based [DeFo-79]. Μία άλλη προσέγγιση είναι η εξής: κάθε σύνολο που προκύπτει από την απαρίθμηση είναι εκείνο του οποίου οι λίστες καταστάσεων θα κατασκευαστούν από τις αντίστοιχες λίστες των συνόλων που έχουν ακριβώς μία εργασία λιγότερη. Η προσέγγιση αυτή χαρακτηρίζεται "pulling"-based. Όταν τα σύνολα απαριθμούνται με αύξουσα σειρά πληθάριθμου χρησιμοποιείται η πρώτη προσέγγιση. Αυτό συμβαίνει γιατί είναι εύκολο να απαριθμούνται τα σύνολα ενός συγκεκριμένου πληθάριθμου όταν έχουν ήδη δημιουργηθεί, ενώ αν χρησιμοποιούνταν η δεύτερη προσέγγιση, θα έπρεπε να απαριθμούνται αυτά χωρίς να έχουν από πριν δημιουργηθεί, κάτι που θα αύξανε τον υπολογιστικό φόρτο του αλγορίθμου. Όταν τα σύνολα απαριθμούνται με λεξικογραφική σειρά δεν επηρεάζεται ο αριθμός των απαιτούμενων πράξεων από το ποιά από τις δύο παραπάνω προσεγγίσεις θα επιλεγεί, αφού η πλοκή της λεξικογραφικής απαρίθμησης των εφικτών συνόλων είναι ανεξάρτητη από το εάν έχουν ήδη δημιουργηθεί τα σύνολα που προκύπτουν από την απαρίθμηση ή όχι. Στη βιβλιογραφία, πάντως, η λεξικογραφική απαρίθμηση των εφικτών συνόλων συνδυάζεται με τη δεύτερη προσέγγιση, με την οποία την είχαν συνδυάσει οι εμπνευστές της, χωρίς να υπάρχει λόγος να γίνει ή να μη γίνει αυτό. Επειδή εμείς είχαμε χρησιμοποιήσει την πρώτη ("pulling"-based) προσέγγιση στον πρώτο αλγόριθμο (που η απαρίθμηση των συνόλων γινόταν με αύξουσα σειρά πληθάριθμου) χρησιμοποιήσαμε την ίδια και στο δεύτερο, αφού, όπως είπαμε, ο αριθμός των απαιτούμενων πράξεων είναι ακριβώς ο ίδιος.

Με το πέρας του αλγορίθμου Δυναμικού Προγραμματισμού, έχουμε την ίδια πληροφορία που είχαμε και στην προηγούμενη υλοποίηση. Για την ανακατασκευή της βέλτιστης λύσης ισχύουν όσα είχαμε πει εκεί.

Από άποψη πλοκής, δεν απαιτούνται τώρα οι $O(n^2)$ πράξεις που ήταν απαραίτητες πριν για να βρεθεί το σύνολο, που προκύπτει από την προσθήκη μιας εργασίας σ’ ένα άλλο σύνολο, αφού η διεύθυνσή του, που καθορίζει τη θέση του πίνακα που αυτό βρίσκεται, υπολογίζεται προσθέτοντας απλά το label της νέας εργασίας στη διεύθυνση του αρχικού συνόλου. Έτσι για κάθε εργασία, που προστίθεται στις εργασίες ενός συνόλου, απαιτούνται $O(n)$ πράξεις για τον έλεγχο εφικτότητας του νέου συνόλου (εάν, δηλαδή, όλοι οι πρόγονοι της εργασίας ανήκουν στο σύνολο), κάτι που γινόταν βέβαια και στον προηγούμενο αλγόριθμο αλλά υπερκαλυπτόταν από τις $O(n^2)$ πράξεις που απαιτούσε η αναζήτηση του προκύπτοντος συνόλου, κι άλλες $O(b{\cdot}T)$ για την επεξεργασία των λιστών καταστάσεων, όπως και πριν (ουσιαστικά την κατάλληλη συγχώνευσή τους σε μία λίστα του προκύπτοντος συνόλου). Από την άλλη, βέβαια, η πλοκή της επιλογής ενός νέου συνόλου του οποίου τις λίστες καταστάσεων θα επεξεργαστούμε για να προκύψουν νέες καταστάσεις γίνεται τώρα $O(n^2)$ γιατί αυτή είναι η πλοκή της δημιουργίας του χαρακτηριστικού διανύσματος ενός εφικτού συνόλου, όταν έχουμε το αντίστοιχο διάνυσμα του εφικτού συνόλου που προηγείται λεξικογραφικά. Στον προηγούμενο αλγόριθμο αυτό γίνεται σε χρόνο O(n) γιατί επιλέγεται και διαγράφεται ένας κόμβος από ένα δυαδικό δένδρο βάθους όχι μεγαλύτερου από n. Η δουλειά αυτή όμως γίνεται μία φορά για κάθε σύνολο κι όχι μία φορά για κάθε εργασία που προστίθεται στις εργασίες κάποιου συνόλου. Επομένως η συνολική πλοκή του αλγορίθμου (συγκεκριμένα της πρώτης αναδρομής, όταν χρησιμοποιείται η μέθοδος των Munro-Ramirez για την ανακατασκευή της βέλτιστης λύσης, η οποία καθορίζει την πλοκή όλου του αλγορίθμου) είναι $O(M \cdot n \cdot (b \cdot T + n))$. Βλέπουμε λοιπόν ότι η πλοκή του αλγορίθμου αυτού είναι λίγο μικρότερη από την πλοκή του προηγούμενου, αφού το n^2 που προσετίθετο στο $b \cdot T$ έχει τώρα αντικατασταθεί από το n. Όπως λέγαμε όμως και παραπάνω, κυρίαρχος όρος του αθροίσματος, για συνηθισμένες τιμές των παραμέτρων, είναι το $b \cdot T$ οπότε η διαφορά αυτή δεν είναι τόσο σημαντική, όσο φαίνεται σε πρώτη ματιά. Η απαιτούμενη μνήμη είναι, βέβαια, μεγαλύτερη και συγκεκριμένα το τμήμα αυτής που χρησιμοποιείται για την αποθήκευση του πίνακα είναι $O(LMAX \cdot b)$ κι εκείνο που χρησιμοποιείται για την αποθήκευση των καταστάσεων $O(M \cdot b \cdot T)$, που είναι και το μεγαλύτερο, αφού μπορεί το M να είναι μικρότερο του LMAX (γύρω στο 1/4 του, σύμφωνα με τον Steiner [Stei-90]), όμως στη δεύτερη ποσότητα εμφανίζεται και το T που είναι αρκετά μεγαλύτερο της μονάδας και το μέγεθος των δομών που αντιπροσωπεύουν καταστάσεις είναι αρκετά μεγάλυτερο του ενός και μόνο δείκτη που βρίσκεται σε καθεμιά από τις $LMAX \cdot b$ θέσεις του πίνακα.

Εκτελώντας το πρόγραμμα που υλοποιούσε τον αλγόριθμο αυτό (συγκεκριμένα τη πρώτη αναδρομή της μεθόδου των Munro-Ramirez) διαπιστώσαμε, όπως και πριν, ότι ο χρόνος ήταν πιο περιοριστικός από τη μνήμη. Από το μικρό δείγμα πειραμάτων που εκτελέσαμε δοκιμαστικά προέκυψε ότι ο αλγόριθμος αυτός είναι γύρω στο 9% ταχύτερος από τον προηγούμενο, όμως και πάλι το όριο του ενός λεπτού ξεπερνιόταν πριν η απαιτούμενη μνήμη ξεπεράσει τη διαθέσιμη.

Έτσι, λοιπόν, τροποποιήσαμε ελαφρά το τμήμα του αλγορίθμου που ήταν υπεύθυνο για τη δημιουργία (απαρίθμηση) και αποθήκευση των εφικτών συνόλων κατασταλάζοντας στη μέθοδο των Schrage-Baker. Τώρα ο πίνακας έχει LSUM θέσεις, περιέχοντας όλα τα εφικτά σύνολα. Στο τέλος του Δυναμικού Προγραμματισμού από τις θέσεις του πίνακα που αντιστοιχούν σε εφικτά σύνολα ξεκινάνε b λίστες καταστάσεων που περιέχουν όλες τις καταστάσεις που δημιουργήθηκαν, εφόσον η μη κυκλική χρησιμοποίηση του πίνακα δεν μας ανάγκασε να σβήνουμε λίστες καταστάσεων. Έτσι δε χρειάζεται ιδιαίτερη προσπάθεια για την ανακατασκευή της βέλτιστης λύσης (αρκεί σε κάθε κατάσταση να αποθηκεύεται το ποιά είναι η τελευταία εργασία της αντίστοιχης διάταξης) και δεν χάνεται χρόνος στη διαγραφή των λιστών καταστάσεων που ξεκινάνε από θέσεις του πίνακα που πρόκειται να ξαναχρησιμοποιηθούν για άλλα σύνολα. Όταν, βέβαια, η λύση του $1/s_{ij}/\sum C_i | T_{max}=0$ χρησιμοποιηθεί λίγο παρακάτω για να δώσει ένα αποδοτικό σημείο του $1/s_{ij}/\sum C_i, T_{max}$ οι διαγραφές αυτές θα είναι απαραίτητες, όταν υπολογίζεται το ένα αποδοτικό σημείο και προχωράμε για το επόμενο, οπότε θα διαγράφεται ο πίνακας των συνόλων και οι λίστες καταστάσεων που ξεκινάνε από αυτόν, γιατί η μνήμη δεν θα είναι αρκετή για την αποθήκευση όλων αυτών των πληροφοριών για όλα τα αποδοτικά σημεία ταυτόχρονα. Η πλοκή της τελικής μορφής του αλγορίθμου, που χρησιμοποιεί δηλαδή τη μέθοδο των Schrage-Baker, είναι ίδια με την πλοκή όταν χρησιμοποιείται η μέθοδος των Kao-Queyranne για την απαρίθμηση και αποθήκευση των εφικτών συνόλων, δηλαδή $O(M \cdot n \cdot (b \cdot T+n))$, ενώ η απαιτούμενη μνήμη είναι μεγαλύτερη αφού ο πίνακας αποθηκεύεται σε $O(LSUM \cdot b)$ θέσεις και δεν διαγράφονται πλέον λίστες καταστάσεων.

Από το μικρό δείγμα πειραμάτων που, όπως είπαμε, εκτελέσαμε δοκιμαστικά η τελική αυτή μορφή φάνηκε ταχύτερη από την προηγούμενη που χρησιμποποιούσε κυκλικά τον πίνακα (συγκεκριμένα από την πρώτη αναδρομή της μεθόδου των Munro-Ramirez) κατά 12% περίπου, προφανώς λόγω της μη διαγραφής των λιστών καταστάσεων (αφού αυτή ήταν η μόνη επιπλέον δουλειά που γινόταν εκεί), αλλά και πάλι το όριο στην αύξηση του μεγέθους των προβλημάτων έμπαινε από το χρόνο και όχι από τη μνήμη. Στο ίδιο συμπέρασμα οδηγούν και τα εκτενή πειράματα που εκτελέσαμε λύνοντας το $1/s_{ij}/\sum C_i,T_{max}$ και θα παρουσιάσουμε αναλυτικά στο επόμενο κεφάλαιο.

Να σημειώσουμε ότι επειδή η μέθοδος των Kao-Queyranne είναι στην ουσία μία παραλλαγή της μεθόδου των Schrage-Baker και με τον τρόπο που τις υλοποιήσαμε εμείς (με τον τρόπο, συγκεκριμένα, που απαριθμούμε τα εφικτά σύνολα, που είναι κοινός και στις δύο) διαφέρουν ελάχιστα, θα τις θεωρούμε ως δύο εκδοχές του ίδιου αλγορίθμου. Αφού λοιπόν περιοριστικότερος παράγοντας αποδείχτηκε ο χρόνος και ταχύτερος αλγόριθμος είναι αυτός που χρησιμοποιεί για τη δημιουργία και αποθήκευση των εφικτών συνόλων την αυθεντική μέθοδο των Schrage-Baker, αυτόν χρησιμοποιήσαμε επαναληπτικά για να βρούμε ένα-ένα τα αποδοτικά σημεία του $1/s_{ij}/\sum C_i,T_{max}$ και σε αυτόν αναφερόμαστε στο εξής.

5.5. Επίλυση του $1/s_{ij}/\sum C_i,T_{max}$

Για να βρεθεί το πρώτο αποδοτικό σημείο του $1/s_{ij}/\sum C_i,T_{max}$, αυτό δηλαδή με το μικρότερο $\sum C_i$ και το μεγαλύτερο T_{max}, προσθέτουμε στις προθεσμίες των εργασιών τη μέγιστη τιμή που μπορεί να πάρει το C_{max} (θα μπορούσε να είναι οποιοσδήποτε "πολύ μεγάλος" ακέραιος) ώστε κάθε πρόγραμμα να είναι εφικτό και λύνουμε το $1/s_{ij}/\sum C_i|T_{max}=0$ για τις προκύπτουσες προθεσμίες, d_i^1. Όταν βρεθεί το πρώτο αποδοτικό σημείο, έστω π_1, οι προθεσμίες μειώνονται από την προηγούμενη τιμή τους, d_i^1, κατά $L'_{max}(\pi_1)-1$, όπου το $L'_{max}(\pi_1)$ είναι η μέγιστη βραδύτητα του π_1 μετρημένη ως προς τις προθεσμίες d_i^1, κάτι που είναι ισοδύναμο με το να αυξηθούν οι προθεσμίες από τις αρχικές τιμές τους, d_i, κατά $T_{max}(\pi_1)-1$ (τώρα το T_{max} είναι μετρημένο ως προς d_i) και για τις νέες προθεσμίες που προκύπτουν, έστω d_i^2, λύνεται πάλι το $1/s_{ij}/\sum C_i|T_{max}=0$. Η παραπάνω διαδικασία επαναλαμβάνεται συνεχώς δίνοντας σε κάθε επανάληψή της κι ένα νέο αποδοτικό σημείο με μικρότερο T_{max} από το προηγούμενο και μεγαλύτερο $\sum C_i$, όπως λέγαμε στο δεύτερο κεφάλαιο. Οι επαναλήψεις σταματούν όταν δεν βρεθεί, κάποια φορά, εφικτή λύση στο $1/s_{ij}/\sum C_i|T_{max}=0$ (με βάση τις προθεσμίες όπως έχουν διαμορφωθεί εκείνη τη στιγμή, έστω d_i^k εάν πρόκειται για την k-οστή φορά που επιχειρούμε να λύσουμε ένα στιγμιότυπο του $1/s_{ij}/\sum C_i|T_{max}=0$) που σημαίνει ότι το τελευταίο αποδοτικό σημείο, αυτό δηλαδή με το μικρότερο T_{max} και το μεγαλύτερο $\sum C_i$, ήταν εκείνο που βρέθηκε στην τελευταία επανάληψη, ή εάν βρεθεί ένα αποδοτικό σημείο με T_{max} (ως προς d_i) ίσο με μηδέν. Εάν δεν μας ενδιέφερε το T_{max} αλλά το L_{max}, θα σταματούσαμε μόνο στην πρώτη από τις δύο παραπάνω περιπτώσεις.

Να σημειώσουμε ότι όσες προθεσμίες, από αυτές για τις οποίες καλούμαστε σε κάποια επανάληψη να λύσουμε το $1/s_{ij}/\sum C_i|T_{max}=0$, είναι μεγαλύτερες από τη μέγιστη δυνατή τιμή του C_{max}, έστω CMAX, κάτι που ισχύει στην πρώτη επανάληψη για όλες τις προθεσμίες, θεωρούνται για την επανάληψη αυτή ίσες με CMAX. Αυτό δεν αλλοιώνει τα αποτελέσματα γιατί, έτσι κι αλλιώς, αποκλείεται κάποια εργασία να περατωθεί μετά από τη στιγμή αυτή. Αντίθετα, έχει το θεμιτό αποτέλεσμα ότι δημιουργούνται, από την πρόταση 2.3, περισσότερες σχέσεις προτεραιότητας μεταξύ των εργασιών, γιατί για οποιοδήποτε ζεύγος εργασιών της ίδιας ομάδας με προθεσμίες μεγαλύτερες του CMAX δημιουργείται μία τέτοια σχέση, ενώ όταν οι προθεσμίες δύο εργασιών είναι άνισες, θα πρέπει η εργασία με τη μικρότερη προθεσμία να έχει και το μικρότερο χρόνο επεξεργασίας για να δημιουργηθεί μία σχέση προτεραιότητας μεταξύ τους.

Γι αυτό και στην πρώτη ειδικά επανάληψη οποιεσδήποτε δύο εργασίες κάθε ομάδας συνδέονται μεταξύ τους με μια σχέση προτεραιότητας, οπότε οι εργασίες κάθε ομάδας τοποθετούνται σε μία διάταξη που διατηρείται σε κάθε διάταξη του συνόλου των εργασιών που είναι συμβατή με τους κανόνες προτεραιότητας. Αυτό έχει σαν αποτελέσμα τα εφικτά σύνολα να είναι πολύ λιγότερα, όπως και οι εργασίες που μπορούν να προστεθούν σε αυτά, απ' ότι στις επόμενες επαναλήψεις με τελικό αποτέλεσμα, όπως θα δούμε στο επόμενο κεφάλαιο, ο απαιτούμενος υπολογιστικός χρόνος να είναι πολύ μικρότερος. Από θεωρητική άποψη, τα εφικτά σύνολα μειώνονται τώρα από $O(2^n)$ σε $O(n^b)$, κι ακόμα καλύτερα σε $O(n_{max}^b)$ (γίνεται, δηλαδή, το πλήθος τους εκθετικό ως προς τον αριθμό των ομάδων κι όχι ως προς τον αριθμό των εργασιών), και οι εργασίες που μπορούν να προστεθούν σε κάθε σύνολο από O(n) σε O(b), οπότε η πλοκή μειώνεται τώρα από $O(2^n \cdot n \cdot (b \cdot T + n))$ (έχοντας αντικαταστήσει το M με 2^n για σαφήνεια) σε $O(n^b \cdot b \cdot (b \cdot T + n))$.

Θα μπορούσαμε, με κατάλληλη αποθήκευση των εργασιών, να μην ελέγχουμε την εφικτότητα του συνόλου που δημιουργείται από την προσθήκη μιας εργασίας σε κάποιο άλλο σύνολο, γιατί η εργασία αυτή θα ήταν πάντα μία από τις b που θα μπορούσαν να προστεθούν, οπότε θα εξαλειφόταν το n μέσα στην παρένθεση. Κυρίως όμως θα μπορούσαμε να εφαρμόζουμε την πρόταση 2.5 στη μορφή που αυτή δίνεται για το $1/s_{ij}/\sum C_i$, οπότε για οποιεσδήποτε δύο διατάξεις του ίδιου συνόλου που τελειώνουν με εργασία της ίδιας ομάδας η μία θα κυριαρχούσε οπωσδήποτε της άλλης με αποτέλεσμα οι λίστες καταστάσεων να ήταν πάντα μονομελείς εξαλείφοντας έτσι και το T και παίρνοντας σαν πλοκή το $O(n^b \cdot b^2)$ που είναι το ίδιο με την πλοκή του αλγορίθμου των Ahn-Hyun για το πρόβλημα αυτό. Δεν επιχειρήσαμε όμως να ελαττώσουμε κι αλλό το χρόνο που απαιτείται για τον υπολογισμό του πρώτου αποδοτικού σημείου γιατί, όπως θα δούμε στο επόμενο κεφάλαιο, είναι πολύ μικρότερος των αντίστοιχων χρόνων των άλλων αποδοτικών σημείων οπότε η συνολική χρονοβελτίωση θα ήταν ασήμαντη. Να σημειώσουμε, τελειώνοντας την παρατήρηση αυτή, ότι το πρώτο δεν είναι κατ' ανάγκη το μοναδικό αποδοτικό σημείο για τον υπολογισμό του οποίου χρησιμοποιούνται προθεσμίες μεγαλύτερες από CMAX. Απλώς εκεί συμβαίνει αυτό για όλες τις εργασίες.

Μια άλλη ενδιαφέρουσα παρατήρηση είναι ότι όσο μικραίνουν οι προθεσμίες, προχωρώντας από το ένα αποδοτικό σημείο στο άλλο, τόσο περισσότερες σχέσεις προτεραιότητας προκύπτουν από την πρόταση 2.4 η ισχύς της οποίας ευνοείται από τις μικρές προθεσμίες. ΄Ετσι, από κάποιο αποδοτικό σημείο και μετά, οι χρόνοι υπολογισμού τους φθίνουν, όπως θα δούμε στο επόμενο κεφάλαιο, λόγω της μείωσης του αριθμού των εφικτών συνόλων.

Να πούμε τέλος ότι, όταν όλες οι προθεσμίες πέσουν κάτω από CMAX, η κατά EDD ταξινόμηση (στην οποία χρησιμοποιείται, όπως έχουμε πει, ως δευτερεύον κριτήριο ο χρόνος επεξεργασίας) και η εξαγωγή κανόνων προτεραιότητας με βάση την πρόταση 2.3 δίνουν πλέον τα ίδια αποτελέσματα σε κάθε αποδοτικό σημείο, οπότε λαμβάνουν χώρα μία μόνο τελευταία φορά.

6. Πειραματικά αποτελέσματα

Αφού αναλύσαμε στο προηγούμενο κεφάλαιο διάφορους αλγόριθμους επίλυσης του $1/s_{ij}/\sum C_i|T_{max}=0$, επιλέξαμε ανάμεσά τους εκείνον με τη μικρότερη υπολογιστική πλοκή, που συμβαίνει να έχει τη μεγαλύτερη απαίτηση σε μνήμη, η οποία, όμως, όπως θα δούμε αναλυτικότερα στο κεφάλαιο αυτό, παραμένει, σχεδόν πάντα, μικρότερη από τη διαθέσιμη μνήμη (των συγκεκριμένων υπολογιστών που χρησιμοποιήσαμε) όσο το μέγεθος των προβλημάτων είναι τέτοιο που ο απαιτούμενος υπολογιστικός χρόνος να παραμένει μέσα στα όρια που θέτουμε και υλοποιήσαμε ένα πρόγραμμα επίλυσης του $1/s_{ij}/\sum C_i,T_{max}$ βασισμένο στον αλγόριθμο αυτόν, ήρθε η στιγμή να εξετάσουμε κάτι διαφορετικό. Την επίδραση που έχουν στην ταχύτητα εκτέλεσης του παραπάνω προγράμματος (συγκεκριμένα στο μέσο χρόνο υπολογισμού και το πλήθος των αποδοτικών λύσεων) διάφορες παράμετροι που σχετίζονται με τα δεδομένα του προβλήματος και συγκεκριμένα ο αριθμός των εργασιών n, ο αριθμός των ομάδων b (και ειδικότερα πόσο μπορούν να μεγαλώσουν αυτές ώστε ο χρόνος επίλυσης του προβλήματος να μένει σε παραδεκτά όρια), οι τιμές των χρόνων εξάρμωσης (σε σχέση με τους χρόνους επεξεργασίας των εργασιών), η διακύμανσή τους και οι τιμές των προθεσμιών. Ενδιαφέρον παρουσιάζει επίσης και η εξάρτηση του χρόνου υπολογισμού ενός αποδοτικού σημείου από τη σειρά που αυτό κατέχει ανάμεσα στα υπόλοιπα.

Οι απαντήσεις στα παραπάνω ερωτήματα ήρθαν μέσα από την εκτέλεση εκτεταμένων πειραμάτων, δηλαδή εκτελέσεων του προγράμματος που γράψαμε (στη τελική του μορφή) για μια μεγάλη ποικιλία στιγμιότυπων του προβλήματος. Οι εκτελέσεις αυτές έγιναν σε SPARCstations ELC με επεξεργαστή SPARC (τεχνολογίας RISC) στα 33 MHz με απόδοση 17.1 SPECmarks (21 MIPS και 3 MFLOPS), data bus των 32 bits, μνήμη μέχρι 64 MBytes (τόση χρησιμοποιήσαμε εμείς), κρυφή μνήμη 64 KBytes και λειτουργικό σύστημα SunOS-4.1.2. Το πρόγραμμα γράφτηκε σε γλώσσα C που διαθέτει την απαιτούμενη πληρότητα προγραμματιστικών δομών και επιτρέπει δυναμική χρησιμοποίηση της μνήμης και bitwise πράξεις που μας ήταν απαραίτητες. Να σημειώσουμε ότι λόγω της εκθετικής πλοκής του αλγορίθμου η μεταβολή στο μέγεθος των προβλημάτων που λύνονται σ' ένα συγκεκριμένο χρονικό διάστημα (π.χ. ένα λεπτό) θα είναι δυσανάλογα μικρότερη από την ενδεχόμενη διαφορά ταχύτητας που μπορεί να έχουν δύο υπολογιστές στους οποίους δοκιμάζουμε το πρόγραμμά μας.

6.1. Προβλήματα δοκιμών

Στα πειράματα που εκτελέσαμε, οι χρόνοι επεξεργασίας των εργασιών παίρνουν ακέραιες τιμές από την ομοιόμορφη κατανομή στο διάστημα $[1,100]$ και οι προθεσμίες παίρνουν ακέραιες, επίσης, τιμές από την ομοιόμορφη, επίσης, κατανομή στο $[P(1-t-r/2),P(1-t+r/2)]$ όπου $P=\sum_{i=1}^{n} p_i$, το t είναι μία παράμετρος που λέγεται παράγοντας καθυστέρησης και δηλώνει τη σχετική απόσταση του μέσου του διαστήματος όπου παίρνουν τιμές οι προθεσμίες από το P, δίνοντας ταυτόχρονα ένα μέτρο του πόσο "σφικτές" είναι οι προθεσμίες και το r είναι μία ακόμα παράμετρος που λέγεται σχετικό εύρος των προθεσμιών και δηλώνει ακριβώς το σχετικό εύρος του παραπάνω διαστήματος. Αυτός είναι ο συνηθέστερος τρόπος δημιουργίας προβλημάτων (από εδώ κι έπειτα στο κεφάλαιο αυτό με τον όρο "προβλήματα" θα εννοούμε "στιγμιότυπα προβλήματος") σε εργασίες που ασχολούνται με την περιοχή (π.χ. [PoWa-83]). Κάθε εργασία, επίσης, κατανέμεται τυχαία σε μία από τις b ομάδες.

Να σημειώσουμε ότι εάν οι χρόνοι επεξεργασίας λαμβάνονταν στο διάστημα [1,10] (όπως γίνεται σε μερικές εργασίες), παίρνοντας έτσι λιγότερες διαφορετικές τιμές, θα ήταν μεγαλύτερη η πιθανότητα να έχουν δύο εργασίες της ίδιας ομάδας ίσους χρόνους επεξεργασίας οπότε θα προέκυπταν από την πρόταση 2.3 περισσότερες σχέσεις προτεραιότητας μεταξύ των εργασιών ελλατώνοντας έτσι το πλήθος των εφικτών συνόλων. Επειδή λιγότερες τιμές θα παίρνουν τότε και τα άλλα χαρακτηριστικά των εργασιών και των ομάδων, δηλαδή προθεσμίες και χρόνοι εξάρμωσης που, εκτός από την περίπτωση που οι τελευταίοι είναι σταθεροί, λαμβάνονται συνήθως όμοια με τους χρόνους επεξεργασίας (δηλαδή από την ομοιόμορφη κατανομή στο ίδιο διάστημα), οπότε κάθε σχέση μεταξύ τέτοιων ποσοτήτων θα έχει μεγαλύτερη πιθανότητα να ισχύει με ισότητα. Για το λόγο αυτό πιθανόν η πρόταση 2.4 να δώσει λιγότερες σχέσεις προτεραιότητας, αφού κάποιες φορές που η αντίστοιχη ανισότητα ίσχυε με μικρή διαφορά, τώρα (όταν, δηλαδή, οι χρόνοι επεξεργασίας παίρνουν τιμές στο διάστημα [1,10]) μπορεί να ισχύει σαν ισότητα, εφόσον έχει "χαθεί λεπτομέρεια", μη δίνοντας έτσι τις σχέσεις προτεραιότητας που έδινε πριν. Αυτό όμως φαίνεται αρκετά πιο απίθανο από την προηγούμενη παρατήρηση που αναφερόταν στην πρόταση 2.3.

Επειδή, επιπλέον, και οι χρόνοι εξάρμωσης παίρνουν λιγότερες τιμές (εκτός όταν είναι σταθεροί) είναι πιθανότερο τώρα δύο διατάξεις του ίδιου υποσυνόλου εργασιών να έχουν ίδιο C_{max} οπότε, εάν τελειώνουν με εργασία της ίδιας ομάδας, μία από τις δύο θα κυριαρχεί οπωσδήποτε της άλλης, σύμφωνα με την πρόταση 2.5, κάτι που δεν ήταν καθόλου σίγουρο πριν, μειώνοντας έτσι το μέσο μήκος των λιστών καταστάσεων. Αυτό έρχεται σε συμφωνία με το ότι ένα άνω φράγμα στο μήκος των λιστών αυτών είναι το N^s όπου s είναι το πλήθος των διαφορετικών τιμών που μπορούν να πάρουν οι χρόνοι εξάρμωσης. Συνοψίζοντας, λοιπόν, μπορούμε να πούμε ότι, κυρίως στην περίπτωση των μη σταθερών χρόνων εξάρμωσης, η ανάθεση τιμών στους χρόνους επεξεργασίας μέσα από το διάστημα [1,10] αντί του [1,100], με τις συνεπακόλουθες διαφοροποιήσεις στις τιμές των προθεσμιών και των χρόνων εξάρμωσης, αναμένεται να δώσει προβλήματα που θα απαιτούν για την επίλυσή τους μικρότερο αριθμό πράξεων.

Στα πειράματα που εκτελέσαμε το σχετικό εύρος των προθεσμιών, r, πήρε τρεις τιμές και συγκεκριμένα 0.2, 0.5 και 0.8 για να φανεί το πως αυτός επιδρά στους χρόνους επίλυσης των προβλημάτων.

Όπως έχουμε πει τα αποδοτικά σημεία υπολογίζονται ένα-ένα, από εκείνο με το μικρότερο $\sum C_i$ και το μεγαλύτερο T_{max} προς εκείνο με το μικρότερο T_{max} και το μεγαλύτερο $\sum C_i$. Η διαδικασία αυτή σταματάει όταν βρεθεί ένα αποδοτικό σημείο με $T_{max}=0$ ή όταν δεν υπάρχει πρόγραμμα με μικρότερο T_{max} από εκείνο του αποδοτικού σημείου που βρέθηκε τελευταίο. Ο παράγοντας καθυστέρησης t που καθορίζει τη θέση των προθεσμιών επηρεάζει το τι από τα δύο παραπάνω θα συμβεί και πότε. Καθώς μεταβάλλεται το t από κάποια μικρή τιμή προς κάποια μεγαλύτερη, διατηρώντας τα υπόλοιπα μεγέθη του προβλήματος σταθερά, αυτό που αλλάζει είναι ότι οι προθεσμίες μετακινούνται στο χρονικό ορίζοντα προς τα αριστερά, δηλαδή στενεύουν. Αυτό έχει σαν αποτέλεσμα να αυξάνεται σταδιακά ο αριθμός των αποδοτικών σημείων του προβλήματος, αφού ένα αποδοτικό σημείο με $T_{max}=0$ παύει, όταν στενέψουν αρκετά οι προθεσμίες, να έχει $T_{max}=0$, οπότε υπολογίζεται και ένα ακόμα αποδοτικό σημείο με μεγαλύτερο $\sum C_i$, που τώρα έχει αυτό $T_{max}=0$. Αυτό γίνεται καθώς αυξάνεται το t μέχρι να μικρύνουν αρκετά οι προθεσμίες ώστε να μην υπάρχει πρόγραμμα με $T_{max}=0$. Από κει και πέρα όσο και να αυξάνεται το t δεν υπολογίζονται περισσότερα αποδοτικά σημεία, αφού δεν υπάρχουν άλλα τέτοια μετά από εκείνο με το μικρότερο T_{max} (που είναι τώρα θετικό).

Με άλλα λόγια, το σύνολο των αποδοτικών σημείων του $1/s_{ij}/\sum C_i,T_{max}$ είναι ένα υποσύνολο του συνόλου των αποδοτικών σημείων του ίδιου προβλήματος για προθεσμίες που έχουν μεταφερθεί αριστερά τόσο πολύ ώστε όλες οι εργασίες να είναι εκπρόθεσμες ή θεωρώντας ότι το πρόγραμμα ξεκινάει μια χρονική στιγμή ίση με τη μέγιστη προθεσμία ή ακόμα του συνόλου των αποδοτικών σημείων του $1/s_{ij}/\sum C_i,L_{max}$. Υποσύνολο που περιέχει όλα τα αποδοτικά σημεία από το πρώτο (αυτό με το μικρότερο $\sum C_i$) μέχρι κάποιο π^*. Όσο μεγαλύτερες είναι οι προθεσμίες τόσο πιο ολιγομελές είναι το υποσύνολο αυτό, γιατί τόσο συντομότερα συναντάται το π^*, που είναι αυτό με $T_{max}=0$. Όσο μικραίνουν οι προθεσμίες τόσο εισέρχονται και άλλα αποδοτικά σημεία στο σύνολο αυτό ώσπου το τελευταίο να ταυτιστεί με το υπερσύνολό του (το σύνολο, δηλαδή, των αποδοτικών σημείων του $1/s_{ij}/\sum C_i,L_{max}$). Γι’ αυτό, η τιμή του παράγοντα καθυστέρησης δεν επηρεάζει την ταχύτητα υπολογισμού ενός αποδοτικού σημείου αλλά τον αριθμό αυτών. Στα πειράματα που εκτελέσαμε διαλέξαμε δύο τιμές για το t. Μία σχετικά μικρή και συγκεκριμένα 0.1 για την οποία να μην είναι αποδοτικά σημεία του $1/s_{ij}/\sum C_i,T_{max}$ όλα τα αποδοτικά σημεία του $1/s_{ij}/\sum C_i,L_{max}$ και μία αρκετά μεγάλη, συγκεκριμένα 0.5, ώστε το σύνολο των αποδοτικών σημείων του πρώτου να είναι το μέγιστο δυνατό. Έτσι επαληθεύσαμε πειραματικά, τα όσα αναφέραμε παραπάνω θεωρητικά.

Θέλοντας να εξετάσουμε το πώς επιδρά το μέγεθος των χρόνων εξάρμωσης στη δυσκολία επίλυσης των προβλημάτων θεωρήσαμε ότι οι χρόνοι αυτοί είναι σταθεροί, ίσοι με s. Στο s δώσαμε τρεις τιμές, και συκγκεκριμένα 25, 50 και 75 δηλαδή μία μικρή, μία μεσαία και μία μεγάλη σχετικά με τους χρόνους επεξεργασίας. Για να εξετάσουμε επίσης την επίδραση στη δυσκολία του προβλήματος του εάν θα είναι σταθεροί οι χρόνοι εξάρμωσης ή όχι δώσαμε, σαν ένα τέταρτο ενδεχόμενο για τις τιμές των χρόνων εξάρμωσης, στους τελευταίους τιμές από την ομοιόμορφη κατανομή στο $[1,100]$ όπως δηλαδή στους χρόνους επεξεργασίας των εργασιών.

Οι τιμές που δώσαμε στις παραμέτρους n και b (πλήθος εργασιών και ομάδων, αντίστοιχα) ήταν τέτοιες που αφενός να μας δίνουν μία εικόνα του πώς αυξάνει ο χρόνος εκτέλεσης του προγράμματος καθώς αυξάνει κάθε μια από τις δύο και αφετέρου για μερικές ενδεικτικές τιμές του b να μας δείχνουν μέχρι ποιά περίπου τιμή μπορεί να αυξηθεί το n και αντίστροφα για μερικές ενδεικτικές τιμές του n μέχρι ποιά περίπου τιμή μπορεί να αυξηθεί το b, ώστε και στις δύο περιπτώσεις το πρόγραμμα να εκτελείται σε παραδεκτό χρόνο. Επειδή από τις αρχικές δοκιμές που κάναμε φάνηκε ότι για όχι μεγάλες τιμές του b το πρόγραμμα έτρεχε σε παραδεκτό χρόνο για n μέχρι λίγο μεγαλύτερο από 20 στο n δώσαμε τιμές σε εκείνη την περιοχή αριθμών. Συγκεκριμένα το n πήρε τις τιμές 18, 20, 22, 24 κι επειδή για αυτήν την τελευταία από τις διάφορες τιμές του b που δοκιμάσαμε, και θα πούμε αμέσως παρακάτω, το πρόγραμμα έτρεχε (σε παραδεκτό χρόνο) μόνο για b=2, αυξήσαμε το n σε 30 και 35, μια και ήταν φανερό από τους χρόνους εκτέλεσης των προγραμμάτων για τις προηγούμενες τιμές του n (και b=2) ότι αυτό θα μεγάλωνε πολύ ακόμα μέχρι να ξεπεραστούν τα χρονικά όρια που, όπως θα πούμε, θέσαμε. Το βήμα αύξησης του n ήταν 2, σχετικά μικρό δηλαδή, γιατί η εκθετική πλοκή του αλγορίθμου μας προιδέαζε για σημαντική αύξηση του απαιτούμενου χρόνου, όπως και πράγματι γίνεται, για μικρές αυξήσεις του n.

Οι τιμές που δόθηκαν στο b ήταν 2, 5, 8 και 11, για όσες τιμές του n η καθεμιά είχε νόημα να δίνεται. Όταν, δηλαδή, για κάποιες τιμές των n και b προέκυπτε το συμπέρασμα ότι δεν εκτελούνταν το πρόγραμμα σε παραδεκτό χρόνο δεν υπήρχε λόγος να εκτελεστεί πρόγραμμα για το ίδιο b και μεγαλύτερο n ή για το ίδιο n και μεγαλύτερο b. Όπως θα δούμε παρακάτω το b=11 ήταν ήδη πολύ μεγάλο για τις τιμές του n που χρησιμοποιήσαμε, οπότε δεν υπήρχε λόγος να του δοθούν μεγαλύτερες τιμές.

Επειδή το πρόβλημα που μας απασχολεί εμφανίζεται, όπως έχουμε πει, στον Λεπτομερειακό Προγραμματισμό Παραγωγής όπου οι αποφάσεις πρέπει να λαμβάνονται σε σχεδόν πραγματικό χρόνο θεωρήσαμε ότι ο χρόνος εκτέλεσης του προγράμματος είναι παραδεκτός όταν ο μέσος χρόνος υπολογισμού του ενός αποδοτικού σημείου δεν ξεπερνάει το 1 λεπτό. Γι’ αυτό και όταν κατά την εκτέλεση του προγράμματος ο χρόνος υπολογισμού κάποιου αποδοτικού σημείου ξεπέρναγε κάποιο όριο αρκετά μεγαλύτερο του ενός λεπτού χωρίς να έχει ακόμα υπολογισθεί το συγκεκριμένο αποδοτικό σημείο, το πρόγραμμα διακόπτονταν. Το όριο αυτό τέθηκε στα 2.5 λεπτά. Επειδή πέρα από το χρόνο ανά αποδοτικό σημείο σημασία έχει και ο χρόνος εκτέλεσης όλου του προγράμματος, διακόπτεται το τελευταίο και στην περίπτωση που ο χρόνος εκτέλεσης ξεπεράσει τα 20 λεπτά και ακόμα δεν έχουν βρεθεί όλα τα αποδοτικά σημεία. Επειδή μία τυπική τιμή για τον αριθμό των αποδοτικών σημείων ενός προβλήματος είναι όπως θα δούμε γύρω στα 10 ή λιγο μεγαλύτερη, το χρονικό όριο που συνήθως έχει ξεπεραστεί, όταν συμβεί διακοπή της εκτέλεσης του προγράμματος, είναι το πρώτο από τα δύο. Το δεύτερο παραβιάζεται σε περιπτώσεις προβλημάτων με πολύ μεγάλο αριθμό αποδοτικών σημείων και αρκετό χρόνο υπολογισμού του καθενός από αυτά.

Για κάθε συγκεκριμένη τετράδα των παραμέτρων n, b, t, r και κάθε συγκεκριμένο τρόπο ανάθεσης τιμών στους χρόνους εξάρμωσης (εάν, δηλαδή, θα παίρνουν σταθερή τιμή και ποιά ή εάν θα είναι ανεξάρτητοι ακολουθίας με τιμές από το ίδιο διάστημα που παίρνουν και οι χρόνοι επεξεργασίας) κατασκευάσαμε, με τον τρόπο που περιγράψαμε παραπάνω, και λύσαμε με το πρόγραμμά μας 10 προβλήματα. Έφόσον οι χρόνοι εξάρμωσης παίρνουν τιμές με τέσσερεις διαφορετικούς τρόπους (σταθεροί, ίσοι με 25, 50 και 75 και ανεξάρτητοι ακολουθίας με τιμές από το [1,100]), ο παράγοντας καθυστέρησης t παίρνει δύο τιμές (0.1 και 0.5), το σχετικό εύρος των προθεσμιών παίρνει 3 (0.2, 0.5 και 0.8) και για καθορισμένες παραμέτρους λύνουμε 10 προβλήματα έχουμε ότι για κάθε ζεύγος n, b λύνονται $4 \cdot 2 \cdot 3 \cdot 10 = 240$ προβλήματα. Όπως θα δούμε εξετάσαμε 13 διαφορετικά ζεύγη των n και b οπότε ο συνολικός αριθμός προβλημάτων που λύθηκαν είναι $240 \cdot 13 = 3120$.

6.2. Παρουσίαση και ανάλυση αποτελεσμάτων

Στους πίνακες που ακολουθούν παρουσιάζονται, με τρόπο που θα εξηγήσουμε, τα αποτελέματα των πειραμάτων. Ο πίνακας 1 έχει ως στόχο να δείξει την επίδραση των n και b στο χρόνο επίλυσης των προβλημάτων καθώς και τη σχετική ταχύτητα υπολογισμού των διαφόρων αποδοτικών σημείων. Η πληροφορία που βρίσκεται σε κάθε θέση του διδιάστατου αυτού πίνακα, που αντιστοιχεί σε κάποιο ζεύγος των n και b, έχει την εξής ερμηνεία :

- 1η γραμμή: αριθμός προβλημάτων (από τα 240) που λύθηκαν και μέσα σε παρένθεση πόσα δεν λύθηκαν επειδή απαιτούσαν περισσότερη μνήμη από τη διαθέσιμη + πόσα δεν λύθηκαν επειδή ξεπέρασαν ένα από τα δύο χρονικά όρια που αναφέραμε.
- 2η γραμμή: ελάχιστη, μέση και μέγιστη τιμή του μέσου υπολογιστικού χρόνου ανά αποδοτικό σημείο. Για κάθε ένα από τα προβλήματα που λύθηκαν, δηλαδή, υπολογίζεται πόσος χρόνος χρειάστηκε, κατά μέσο όρο, για τον υπολογισμό ενός αποδοτικού σημείου διαιρώντας το συνολικό χρόνο εκτέλεσης του προγράμματος με τον αριθμό των αποδοτικών σημείων και στη γραμμή αυτή σημειώνεται ο ελάχιστος χρόνος από αυτούς, ο μέσος και ο μέγιστος.
- 3η γραμμή : ελάχιστος, μέσος και μέγιστος αριθμός αποδοτικών σημείων.
- 4η γραμμή : ελάχιστος, μέσος και μέγιστος χρόνος επίλυσης του προβλήματος (υπολογισμού δηλαδή όλων των αποδοτικών σημείων).
- 5η γραμμή : ελάχιστος, μέσος και μέγιστος χρόνος υπολογισμού του πρώτου αποδοτικού σημείου.
- 6η γραμμή : ελάχιστος, μέσος και μέγιστος χρόνος υπολογισμού του δεύτερου αποδοτικού σημείου.
- όμοια και οι υπόλοιπες γραμμές για τα υπόλοιπα αποδοτικά σημεία. Οι χρόνοι αυτοί δίνονται για τόσα αποδοτικά σημεία όσα υπάρχουν σε τουλάχιστο μισά από τα προβλήματα που λύθηκαν (για τα συγκεκριμένα n και b).

Όλοι οι χρόνοι είναι σε δευτερόλεπτα.

Για τους συνδυασμούς των n και b που λύθηκαν λιγότερα από τα μισά προβλήματα δεν δίνουμε άλλα στοιχεία πέρα από το πόσα λύθηκαν και το γιατί δε λύθηκαν τα υπόλοιπα (δηλαδή τα στοιχεία της πρώτης γραμμής) γιατί θεωρούμε τα υπόλοιπα στατιστικά στοιχεία μάλλον αναξιόπιστα λόγω του μικρού ποσοστού των λυμένων προβλημάτων, δηλαδή θεωρούμε ότι αυτά θα ήταν, κατά πάσα πιθανότητα, πολύ διαφορετικά εάν δεν είχαμε βάλει τα χρονικά όρια και είχαμε στη διάθεσή μας αρκετά μεγαλύτερη μνήμη οπότε θα λύνονταν όλα τα προβλήματα.

Από τον πίνακα 1 φαίνεται ότι ο μέσος χρόνος υπολογισμού ενός αποδοτικού σημείου αυξάνει σημαντικά καθώς αυξάνει το n όχι όμως και εκθετικά όπως δηλώνει η έκφραση της υπολογιστικής πλοκής. Αυτό μπορεί να αποδοθεί στο ότι ο αριθμός M των εφικτών συνόλων που είναι θεωρητικά της τάξης του 2^n είναι στην πράξη αρκετά μικρότερος από αυτό (λόγω των σχέσεων προτεραιότητας μεταξύ των εργασιών και λιγότερο των διαγραφών καταστάσεων κατά την εξέλιξη του Δυναμικού Προγραμματισμού) και, κυρίως, αυξάνει με μικρότερο ρυθμό. Παρατηρούμε επίσης ότι αυξάνοντας το n αυξάνεται σχεδόν πάντα και ο αριθμός των αποδοτικών σημείων. Αυτό είναι λογικό εφόσον όσο μεγαλύτερο είναι το n τόσο περισσότερες είναι οι δυνατές διατάξεις των εργασιών, που είναι n!, οπότε είναι φυσικό τόσο περισσότερες ανάμεσά τους να είναι αποδοτικές. Μία εξαίρεση στην παρατήρηση αυτή αποτελεί η περίπτωση n=22, b=5 που έχει λιγότερα αποδοτικά σημεία από όσα εμφανίζονται σε μικρότερα n για το ίδιο b. Αυτό εξηγείται πιθανώς από το γεγονός ότι για το συγκεκριμένο συνδυασμό των n και b έχουν τερματίσει τα λιγότερα τρεξίματα (179) και ίσως αρκετές από τις υπόλοιπες δοκιμές που δεν τερμάτισαν να είχαν μεγάλο αριθμό αποδοτικών σημείων και να ξεπέρασαν το όριο του 20λέπτου. Πάντως σε όλες τις άλλες περιπτώσεις αύξηση του n οδηγεί σε αύξηση του αριθμού των αποδοτικών σημείων.

Όσον αφορά στην επίδραση που έχει ο αριθμός των ομάδων b, παρατηρούμε ότι η αύξηση αυτού συνεπάγεται μεγάλη αύξηση του απιτούμενου υπολογιστικού χρόνου για τον υπολογισμό ενός αποδοτικού σημείου, αρκετά μεγαλύτερη από τη γραμμική αύξηση που δηλώνει η υπολογιστική πλοκή. Αυτό οφείλεται στο ότι όσο αυξάνει το b, για συγκεκριμένο n, τόσο λιγοστεύουν οι εργασίες της κάθε ομάδας με αποτέλεσμα τόσο να λιγοστεύουν και οι σχέσεις προτεραιότητας μεταξύ εργασιών που προκύπτουν από την πρόταση 2.3, η οποία δίνει τέτοιες σχέσεις για εργασίες της ίδιας ομάδας. Η μείωση αυτή των σχέσεων προτεραιότητας οδηγεί σε αύξηση του αριθμού των εφικτών συνόλων και άρα του απαιτούμενου υπολογιστικού χρόνου. Το πλήθος των αποδοτικών σημείων δεν φαίνεται να επηρεάζεται από το b.

Μία τελευταία ενδιαφέρουσα παρατήρηση που εξάγεται από τα στοιχεία του πίνακα 1 σχετίζεται με την επίδραση της θέσης ενός αποδοτικού σημείου ανάμεσα στα υπόλοιπα (αν δηλαδή είναι πρώτο, δεύτερο κ.ο.κ) στον απαιτούμενο χρόνο υπολογισμού του. Αυτό που παρατηρεί κανείς αρχικά είναι ότι ο χρόνος υπολογισμού του πρώτου αποδοτικού σημείου είναι πολύ μικρότερος από τους χρόνους υπολογισμού των υπολοίπων σημείων (συνήθως από 15 έως 80 φορές). Έπειτα οι χρόνοι αυξάνονται, καθώς προχωράμε στα επόμενα αποδοτικά σημεία, μέχρι κάποια μέγιστη τιμή (που συνήθως εντοπίζεται στο τρίτο αποδοτικό σημείο) και μετά φθίνουν.

	n = 18			n = 20			n = 22			n = 24		
b = 2	240 (0 + 0)			240 (0 + 0)			240 (0 + 0)			240 (0 + 0)		
	0.0	0.7	3.0	0.0	1.6	12.8	0.1	3.4	17.3	0.1	5.8	38.7
	1	10.0	41	1	11.5	45	1	14.6	54	1	15.2	57
	0.0	7.5	50.5	0.0	22.8	307.6	0.1	59.1	694.7	0.1	102.6	1054.8
	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.1	0.0	0.1	0.1
	0.1	0.8	3.4	0.2	1.8	11.0	0.4	3.7	15.6	0.7	6.0	33.6
	0.1	0.9	4.2	0.2	2.1	13.8	0.4	4.5	18.7	0.6	7.5	40.9
	0.1	0.9	4.1	0.2	2.1	15.2	0.5	4.6	19.0	0.5	7.8	47.6
	0.1	0.9	4.0	0.2	2.1	14.7	0.6	4.6	19.1	0.7	7.9	50.9
	0.1	0.9	4.0	0.2	2.2	15.0	0.5	4.5	18.7	1.0	7.6	49.8
	0.1	0.9	3.8	0.2	2.2	14.9	0.4	4.4	18.4	0.7	7.5	49.8
	0.1	0.9	3.6	0.2	2.2	14.7	0.5	4.6	18.4	0.8	7.6	50.4
				0.2	2.2	14.6	0.5	4.7	18.8	0.9	7.9	47.2
							0.5	4.6	18.6	1.1	8.1	46.6
							0.5	4.6	18.2	0.9	7.8	44.2
										0.7	7.9	44.0
b = 5	240 (0 + 0)			225 (0 + 15)			179 (0 + 61)			68 (6 + 166)		
	0.8	12.9	50.5	1.7	29.9	110.6	1.7	47.1	118.5			
	1	11.6	40	1	11.9	40	1	10.2	35			
	0.8	156.6	1080.2	1.7	351.4	1215.5	1.7	463.2	1248.7			
	0.5	0.9	1.2	0.8	1.4	1.9	1.6	2.2	3.3			
	3.1	13.9	56.1	6.3	30.5	100.5	8.3	50.2	114.4			
	3.3	17.8	68.8	7.2	39.1	138.9	14.6	63.3	144.2			
	2.5	17.6	64.2	6.9	39.0	141.3	13.5	61.7	138.6			
	2.1	16.9	61.1	5.8	39.0	120.6	13.2	59.9	137.1			
	1.9	16.3	58.6	4.5	37.7	119.8	12.3	56.9	135.5			
	1.0	15.4	55.7	6.8	37.0	125.1	14.0	52.4	134.0			
	2.2	14.8	52.2	6.8	34.8	118.2	13.4	49.8	128.8			
	2.2	14.7	52.0	5.7	32.8	115.0						
				4.6	31.6	112.9						
b = 8	225 (0 + 15)			102 (0 + 138)								
	6.1	37.1	103.1									
	1	9.0	36									
	6.3	321.9	1204.2									
	4.4	6.7	10.9									
	9.2	37.8	92.4									
	9.9	48.6	121.0									
	9.5	48.5	125.4									
	7.1	47.2	117.2									
	6.0	43.5	112.0									
	5.7	40.7	109.0									
b = 11	114 (0 + 126)											

Πίνακας 1 (α). Επίδραση του πλήθους των εργασιών και του πλήθους των ομάδων (μέρος Α).

Η συμπεριφορά αυτή μπορεί κάλλιστα να εξηγηθεί από τα όσα έχουμε πει για τη λειτουργία του αλγορίθμου. Για να υπολογιστεί το πρώτο αποδοτικό σημείο, έχουμε πει ότι οι προθεσμίες ολισθαίνουν δεξιά ξεπερνώντας το CMAX, που είναι ένα άνω φράγμα στο C_{max} κάθε διάταξης. Έχουμε επίσης πει ότι όσες

	n = 30			n = 35
	196 (6 + 38)			89 (44 + 107)
	0.1	23.1	95.8	
	1	17.3	65	
	0.1	382.6	1176.4	
	0.1	0.1	0.1	
	3.2	23.4	86.1	
	3.6	28.7	105.8	
	3.5	29.0	131.4	
	3.3	28.6	129.6	
b = 2	3.2	27.7	120.5	
	3.1	27.7	122.2	
	3.0	27.9	113.3	
	2.9	27.8	107.5	
	2.9	26.2	117.0	
	2.8	25.6	115.9	
	2.6	25.3	114.8	
	2.5	25.0	114.5	
	2.5	23.8	113.7	
	3.7	23.0	110.3	

Πίνακας 1 (β). Επίδραση του πλήθους των εργασιών και του πλήθους των ομάδων (μέρος Β).

προθεσμίες ξεπερνούν το CMAX θεωρούνται, για τον υπολογισμό του αποδοτικού σημείου, ίσες με αυτό. Έτσι στο πρώτο αποδοτικό σημείο όλες οι προθεσμίες θεωρούνται ίσες, με αποτέλεσμα, από την πρόταση 2.3, οι εργασίες κάθε ομάδας να μπαίνουν σε μία διάταξη, την οποία θα τηρήσουν σε κάθε διάταξη όλων των εργασιών. Αυτό είναι πολύ σημαντικό γιατί έτσι τα εφικτά σύνολα είναι πολύ λιγότερα από ότι στα υπόλοιπα αποδοτικά σημεία, για τα οποία ισχύουν πολύ λιγότερες σχέσεις προτεραιότητας, με τελικό αποτελέσμα ο υπολογιστικός χρόνος να είναι πολύ μικρότερος.

Προχωρώντας στο επόμενο αποδοτικό σημείο οι προθεσμίες ολισθαίνουν αριστερά, από τις τιμές που είχαν πριν, και πολλές από αυτές (ίσως και όλες) γίνονται μικρότερες από CMAX. Αφού είναι λιγότερες οι ίσες μεταξύ τους προθεσμίες, από την πρόταση 2.3 προκύπτουν λιγότερες σχέσεις προτεραιότητας, αφού από την πρόταση αυτή απορρέει οπωσδήποτε μία σχέση προτεραιότητας για κάθε ζευγάρι εργασιών της ίδιας ομάδας με ίσες προθεσμίες, κάτι που δεν είναι σίγουρο όταν οι προθεσμίες είναι άνισες. Οι λιγότερες σχέσεις προτεραιότητας έχουν, βέβαια, σαν αποτέλεσμα περισσότερα εφικτά σύνολα και άρα μεγαλύτερο χρόνο υπολογισμού του αποδοτικού σημείου. Καθώς προχωράμε από το ένα αποδοτικό σημείο στο επόμενο τόσο λιγοστεύουν οι προθεσμίες που είναι μεγαλύτερες απο CMAX, άρα λιγοστεύουν και οι σχέσεις προτεραιότητας, μέχρι που (γενικά) γίνονται όλες μικρότερες. Στην πράξη βλέπουμε ότι αυτό συμβαίνει συνήθως γρήγορα και συγκεκριμένα στο τρίτο κιόλας αποδοτικό σημείο όλες οι προθεσμίες έχουν γίνει μικρότερες από CMAX.

Όσο όμως μικραίνουν οι προθεσμίες τόσο πιο αποτελεσματική γίνεται η πρόταση 2.4 η οποία ευνοείται στην εξαγωγή σχέσεων προτεραιότητας από τις μικρές προθεσμίες. Επίσης τόσο πληθαίνουν και οι διαγραφές κόμβων του σχήματος Δυναμικού Προγραμματισμού λόγω εντοπισμού διατάξεων χωρίς εφικτή συνέχεια. Έτσι όσο μικραίνουν οι προθεσμίες, προχωρώντας από το ένα αποδοτικό σημείο στο επόμενο, από τη μία γίνεται όλο και λιγότερο αποδοτική η πρόταση 2.3, αλλά από την άλλη αυξάνεται η αποδοτικότητα των δύο τελευταίων μηχανισμών περιορισμού του χώρου έρευνας που αναφέραμε. Έτσι παρατηρείται αυτό που είπαμε, ότι δηλαδή ο χρόνος υπολογισμού ενός αποδοτικού σημείου αυξάνει μέχρι κάποιο σημείο, γιατί όλο και λιγότερες σχέσεις προτεραιότητας δίνει η πρόταση 2.3 χωρίς να έχουν αρχίσει να εφαρμόζονται (ή να βρίσκουν ελάχιστη εφαρμογή) οι άλλοι δύο μηχανισμοί, κι από εκεί και μετά, λόγω της συνεχούς μείωσης των προθεσμιών που έχουν πλέον γίνει αρκετά μικρές ώστε να βρίσκουν εφαρμογή οι μηχανισμοί που βασίζονται σ' αυτές, οι χρόνοι φθίνουν, χωρίς βέβαια να γίνονται τόσο μικροί όσο του πρώτου αποδοτικού σημείου με τα πολύ μικρότερα (εκθετικά ως προς b, όπως έχουμε πει, και όχι ως προς n) εφικτά σύνολα.

Μπορεί να παρατηρήσει, επίσης, κανείς ότι όσο αυξάνει το n τόσο πιο σημαντική γίνεται η σχετική διαφορά μεταξύ του χρόνου υπολογισμού του πρώτου από τα υπόλοιπα αποδοτικά σημεία, ενώ όσο αυξάνει το b η διαφορά αυτή ελαττώνεται. Το φαινόμενο αυτό μπορεί να εξηγηθεί εάν λάβουμε υπόψιν τη σχέση του πλήθους των εφικτών συνόλων με τα n και b, όταν υπολογίζουμε το πρώτο αποδοτικό σημείο ή κάποιο από τα υπόλοιπα. Μπορούμε, κατ' αρχήν, για απλούστευση να θεωρήσουμε ότι το πλήθος αυτών των συνόλων παίρνει τη μεγαλύτερη δυνατή τιμή και στις δύο περιπτώσεις, δηλαδή σ' ένα οποιοδήποτε αποδοτικό σημείο πλην του πρώτου και στο πρώτο. Στην πρώτη περίπτωση τα εφικτά σύνολα μπορούν να φτάσουν μέχρι 2^n, όταν δεν υπάρχουν καθόλου σχέσεις προτεραιότητας. Στη δεύτερη περίπτωση, που οι εργασίες σε κάθε ομάδα είναι διατεταγμένες, η μεγαλύτερη τιμή που μπορεί να πάρει το πλήθος των εφικτών συνόλων εμφανίζεται όταν οι ομάδες περιέχουν ίδιο αριθμό εργασιών και το πλήθος τότε είναι $(n/b+1)^b$. Καθώς αυξάνει το n, το 2^n αυξάνει πολύ ταχύτερα από το $(n/b+1)^b$, γι' αυτό και αυξάνει η σχετική διαφορά στο χρόνο υπολογισμού του πρώτου αποδοτικού σημείου από τα επόμενα. Αντίθετα, καθώς αυξάνει το b το πλήθος των εφικτών συνόλων στα υπόλοιπα αποδοτικά σημεία, εκτός του πρώτου, μένει σταθερός, ενώ το πλήθος των συνόλων αυτών για το πρώτο αποδοτικό σημείο αυξάνει, μειώνοντας έτσι τη διαφορά στους χρόνους υπολογισμού του πρώτου από τα υπόλοιπα αποδοτικά σημεία.

Όπως έχουμε πει, βέβαια, στη γενική περίπτωση που υπάρχουν σχέσεις προτεραιότητας και στα υπόλοιπα αποδοτικά σημεία το πλήθος των εφικτών συνόλων αυξάνει ως προς το n με μικρότερο ρυθμό από 2^n και δεν είναι ανεξάρτητο από το b (συγκεκριμένα φθίνει καθώς το τελευταίο αυξάνει). Να όμως που και στην πράξη επαληθεύονται τα συμπεράσματα των παραπάνω σκέψεων, κατά τις οποίες άλλωστε πήραμε τη χειρότερη περίπτωση και για το πλήθος των εφικτών συνόλων στα υπόλοιπα, πλην του πρώτου, αποδοτικά σημεία, αλλά και για το πλήθος αυτό στο πρώτο αποδοτικό σημείο. Φάνηκε, δηλαδή, στην πράξη ότι ο εκθετικός χαρακτήρας του 2^n μπορεί να είναι λιγότερο ισχυρός απ' ότι φαίνεται, αλλά παραμένει σημαντικότερος του πολυωνυμικού (ως προς n) $(n/b+1)^b$ το οποίο αυξάνει, ως προς b, πραγματικά ταχύτερα από το πλήθος των εφικτών συνόλων στα εκτός του πρώτου στοιχεία, άσχετα αν και το τελευταίο δεν μένει σταθερό.

Στους πίνακες 2, 3 και 4 τα εκτελεσθέντα πειράματα παρουσιάζονται ομαδοποιημένα κατά τιμή των χρόνων εξάρμωσης, παράγοντα καθυστέρησης και σχετικού εύρους προθεσμιών αντίστοιχα με σκοπό τη διερεύνηση της επίδρασης των τριών αυτών παραμέτρων στο χρόνο εκτέλεσης του προγράμματος. Τα στοιχεία που δίνονται σε κάθε θέση των πινάκων αυτών έχουν την ίδια ερμηνεία με εκείνη του πίνακα 1. Τώρα όμως έχουν χρησιμοποιηθεί για την εξαγωγή τους λιγότερα πειράματα και συγκεκριμένα όλα τα πειράματα με συνδυασμούς των n και b για τους οποίους έχουν λυθεί τουλάχιστον τα μισά προβλήματα (δηλαδή, για τους συνδυασμούς αυτούς για τους οποίους παρουσιάζουμε στατιστικά στοιχεία, πέρα από το πόσα προβλήματα λύθηκαν, στον πίνακα 1). Οι συνδυασμοί αυτοί είναι 9 και αφού κάθε τέτοιος αντιστοιχεί σε 240, όπως έχουμε πει, προβλήματα, ο αριθμός των προβλημάτων τα οποία χρησιμοποιήσαμε για την παιρετέρω εξαγωγή στατιστικών στοιχείων είναι 2160.

S = 25			S = 50			S = 75			S ˜ U[1,100]		
499 (6 + 35)			501 (0 + 39)			511 (0 + 29)			514 (0 + 26)		
0.0	16.6	107.9	0.0	17.9	113.8	0.0	16.1	118.5	0.0	16.4	103.1
1	12.3	46	1	12.3	65	1	12.4	59	1	12.4	53
0.0	198.2	1215.5	0.0	202.9	1248.7	0.0	183.2	1164.3	0.0	192.4	1204.2
0.0	1.3	9.4	0.0	1.2	8.6	0.0	1.2	8.3	0.0	1.3	10.9
0.3	16.0	101.9	0.1	18.5	104.2	0.2	17.0	114.4	0.1	17.9	100.5
0.3	20.7	131.8	0.1	23.7	137.7	0.2	21.1	144.2	0.2	21.4	138.9
0.3	21.2	132.3	0.1	23.5	138.3	0.2	19.8	138.6	0.1	21.4	141.3
0.2	21.0	130.8	0.1	22.7	137.1	0.2	19.2	136.7	0.2	20.7	129.6
0.2	20.7	120.8	0.1	21.2	135.5	0.1	17.9	131.6	0.2	19.7	120.5
0.3	20.3	125.1	0.1	20.9	134.0	0.2	17.4	124.0	0.2	17.9	122.2
0.3	19.5	118.2	0.1	20.1	128.8	0.2	16.4	119.2	0.2	17.6	113.3
0.2	19.4	115.0	0.1	17.6	117.8	0.2	15.3	82.2	0.2	17.5	97.1
0.2	17.9	117.0				0.2	14.6	76.1	0.2	17.5	90.8

Πίνακας 2. Επίδραση των χρόνων εξάρμωσης.

Στον πίνακα 2 τα πειράματα έχουν χωριστεί ανάλογα με την τιμή που παίρνουν οι χρόνοι εξάρμωσης. Σε κάθε τέτοια τιμή αντιστοιχούν 540 προβλήματα.

Μπορεί να παρατηρήσει κάποιος ότι για μεγάλους χρόνους εξάρμωσης (S=75) λύνονται περισσότερα προβλήματα και με μικρότερο χρόνο υπολογισμού ανά αποδοτικό σημείο από ότι για μικρούς (S=25) και μεσαίους (S=50) χρόνους εξάρμωσης. Η λογική εξήγηση που μπορεί να δοθεί σ’ αυτό είναι ότι όσο μεγαλύτεροι είναι οι χρόνοι εξάρμωσης τόσο μεγαλύτεροι είναι οι χρόνοι αποπεράτωσης εργασιών και διατάξεων οπότε τόσο πιο αποτελεσματική γίνονται οι μηχανισμοί περιορισμού του χώρου έρευνας που βασίζονται στην παραβίαση προθεσμιών και συγκεκριμένα τόσο περισσότερες σχέσεις προτεραιότητας απορρέουν από την πρόταση 2.4 και τόσο περισσότερες διαγραφές κόμβων γίνονται λόγω μη εφικτών συνεχειών των αντίστοιχων διατάξεων.

Ένας άλλος λόγος που επίσης συντρέχει στην εμφάνιση του παραπάνω φαινομένου είναι ότι όσο μεγαλώνουν οι χρόνοι εξάρμωσης τόσο αυξάνει η διαφορά μεταξύ του C_{max} δύο διαφορετικών διατάξεων, έστω π_1 και π_2, του ίδιου υποσυνόλου εργασιών με διαφορετικό αριθμό αλλαγών κατεργασίας, οπότε τόσο αυξανει η πιθανότητα όταν το $C_{max}(\pi_1)$ είναι μικρότερο του $C_{max}(\pi_2)$ να ισχύει η ίδια ανισότητα και μεταξύ των ποσοτήτων $\sum C_i(\pi_1)+k\cdot C_{max}(\pi_1)$ και $\sum C_i(\pi_2)+k\cdot C_{max}(\pi_2)$ (όταν απομένουν k εργασίες να διαταχθούν) οπότε η π_1 να κυριαρχεί της π_2, βάση της πρότασης 2.5, με αποτέλεσμα η κατάσταση που αντιστοιχεί στην π_2 να διαγράφεται. Για τον ίδιο λόγο, εξάλλου, που αυξάνει περισσότερο το C_{max} της διάταξης με τις περισσότερες αλλαγές κατεργασίας, πιθανώς θα αυξηθεί περισσότερο και το $\sum C_i$ αυτής, ενισχύοντας έτσι το παραπάνω συμπέρασμα.

Τα στοιχεία που παρατίθενται στον πίνακα 2 δεν βοηθούν στο να γίνει μία σύγκριση της ευκολίας του προβλήματος μεταξύ των περιπτώσεων $(S=25)$ και $(S=50)$ γιατί για το ένα λύθηκαν περισσότερα προβλήματα ενώ για το άλλο ο μέσος υπολογιστικός χρόνος ανά αποδοτικό σημείο ήταν μικρότερος.

Παρατηρούμε ακόμα ότι η συμπεριφορά του προγράμματος είναι καλύτερη (λύνονται περισσότερα προβλήματα και σε λιγότερο χρόνο) όταν οι χρόνοι εξάρμωσης δεν είναι σταθεροί, συγκρίνοντας την περίπτωση που οι χρόνοι αυτοί παίρνουν τιμές από την ομοιόμορφη κατανομή στο $[1,100]$ με την "αντίστοιχη" της $(S=50)$. Ανατρέχοντας στον τρόπο λειτουργίας του αλγορίθμου μας, δεν μπορούμε να εντοπίσουμε κάποιο σημείο που να επηρεάζεται από το εάν οι χρόνοι θα είναι σταθεροί ή όχι. Πάντως η παρατήρηση αυτή έρχεται σε συμφωνία με τις παρατηρήσεις των εργασιών [UsSm-75] και [Γεωρ-91] που, όπως έχουμε πει, ασχολούνται με άλλα προβλήματα Χρονικού Προγραμματισμού με χρόνους εξάρμωσης. Να πούμε ακόμα ότι το πλήθος των αποδοτικών σημείων δεν φαίνεται να επηρεάζεται από το αν οι χρόνοι εξάρμωσης έχουν σταθερή τιμή ή όχι και, αν έχουν σταθερή τιμή, από το ποιά είναι αυτή.

Να σημειώσουμε, τέλος, ότι επειδή, όπως είπαμε, τα στατιστικά στοιχεία που παρουσιάζουμε προέκυψαν λαμβάνοντας υπόψιν διάφορες τιμές των n και b με πολύ διαφορετικούς χρόνους υπολογισμού των αποδοτικών τους σημείων και συνολικά εκτέλεσης του προγράμματος, παρατηρούμε μια πολύ σημαντική διαφορά μεταξύ των ελαχίστων και μεγίστων τιμών των διαφόρων χρόνων που παρουσιάσαμε. Αυτό ισχύει, ασφαλώς, και για τις άλλες δύο παραμέτρους που θα μας απασχολήσουν, t και r.

Στον πίνακα 3 παρουσιάζονται τα ίδια στατιστικά στοιχεία όπως στους δύο προηγούμενους πίνακες μόνο που τώρα η ομαδοποίηση του προβλήματος έγινε ανάλογα με τον παράγοντα καθυστέρησης τους. Έτσι τα 2160 προβλήματα που δοκιμάσαμε και πριν χωρίζονται σε δύο ομάδες των 1080 προβλημάτων, μία με $t=0.1$ και μία με $t=0.5$.

Τα αποτελέσματα είναι τα αναμενόμενα. Για λόγους που εξηγήσαμε νωρίτερα στο κεφάλαιο αυτό, τα αποδοτικά σημεία της δεύτερης περίπτωσης είναι αρκετά περισσότερα από εκείνα της πρώτης, αφού για κάθε πρόβλημα με $t=0.1$ το σύνολο των αποδοτικών σημείων είναι ένα αρχικό απόκομμα του συνόλου των αποδοτικών σημείων του αντίστοιχου προβλήματος με $t=0.5$. Για το

t = 0.1					t = 0.5				
1028 (3 + 49)					997 (3 + 80)				
0.0	-	17.6	-	118.5	0.1	-	15.8	-	118.1
1	-	9.1	-	37	1	-	15.7	-	65
0.0	-	170.0	-	1202.3	0.1	-	218.9	-	1248.7
0.0	-	1.2	-	10.9	0.0	-	1.2	-	10.6
0.1	-	17.9	-	114.4	0.1	-	16.8	-	112.0
0.1	-	22.5	-	144.2	0.1	-	20.9	-	142.7
0.1	-	22.6	-	141.3	0.1	-	20.3	-	140.1
0.1	-	22.4	-	137.1	0.1	-	19.6	-	136.7
0.1	-	21.5	-	135.5	0.1	-	18.5	-	120.8
0.1	-	21.3	-	134.0	0.1	-	17.5	-	119.2
0.1	-	20.9	-	128.8	0.1	-	16.6	-	107.9
					0.1	-	15.8	-	99.9
					0.1	-	15.0	-	93.1
					0.1	-	14.4	-	90.9
					0.1	-	13.9	-	88.2
					0.1	-	13.2	-	85.5

Πίνακας 3. Επίδραση του παράγοντα καθυστέρησης.

λόγο αυτό, ο μέσος χρόνος εκτέλεσης ενός προγράμματος με t=0.5 είναι αρκετά μεγαλύτερος από τον αντίστοιχο χρόνο για t=0.1 και το πλήθος των προβλημάτων που τελικά δεν λύνονται επειδή παραβιάζουν κάποιο από τα δύο χρονικά όρια που έχουμε θέσει (και συγκεκριμένα αυτό του 20λέπτου για όλο το πρόγραμμα, γιατί το άλλο των 2.5 λεπτών ανά αποδοτικό σημείο αν δεν παραβιαστεί στα πρώτα αποδοτικά σημεία που υπολογίζονται και για t=0.1 δε θα παραβιαστεί ούτε στα υπόλοιπα αφού τα τελευταία υπολογίζονται, όπως έχουμε πει, ταχύτερα από τα πρώτα) είναι επίσης μεγαλύτερο. Επειδή τα επιπλέον αποδοτικά σημεία που προκύπτουν για t=0.5 υπολογίζονται σε μικρότερους χρόνους, για λόγους που εκθέσαμε νωρίτερα στο κεφάλαιο αυτό, από τα πρώτα που υπολογίζονται και για t=0.1, ο μέσος χρόνος υπολογισμού ενός αποδοτικού σημείου είναι μικρότερος για t=0.5. Να σημειώσουμε ότι για t=0.1 το τελευταίο αποδοτικό σημείο που υπολογίζεται (αυτό με το μικρότερο T_{max}) έχει τις περισσότερες φορες T_{max}=0, ενώ για t=0.5 το T_{max} του τελευταίου σημείου είναι πάντα θετικό (που, όπως έχουμε πει, δείχνει ότι και να μίκραιναν και άλλο οι προθεσμίες δε θα προσετίθετο κάποιο νέο αποδοτικό σημείο στα υπάρχοντα).

Η επίδραση του σχετικού εύρους προθεσμιών, r, φαίνεται από τα στοιχεία που παρατίθενται στον πίνακα 4. Εδώ τα 2160 προβλήματα έχουν χωριστεί σε τρείς ομάδες των 720 προβλημάτων ανάλογα με την τιμή του r.

Παρατηρούμε αρχικά ότι ο μέσος χρόνος υπολογισμού ενός αποδοτικού σημείου φθίνει καθώς το r αυξάνει. Ένας λόγος για τον οποίο συμβαίνει αυτό είναι ότι με το να είναι πιο "απλωμένες" οι προθεσμίες, αργούν περισσότερο να πέσουν όλες κάτω από CMAX (το άνω φράγμα του C_{max}), καθώς προχωράμε από το ένα αποδοτικό σημείο στο επόμενο, δίνοντας έτσι περισσότερες σχέσεις

r = 0.2	r = 0.5	r = 0.8
694 (0 + 26)	681 (0 + 39)	650 (6 + 64)
0.0 - 20.0 - 118.5	0.0 - 17.5 - 110.6	0.0 - 12.5 - 113.8
1 - 7.7 - 25	1 - 12.0 - 54	1 - 17.7 - 65
0.0 - 162.6 - 1164.0	0.0 - 212.4 - 1248.7	0.0 - 208.6 - 1147.7
0.0 - 1.2 - 10.8	0.0 - 1.2 - 10.7	0.0 - 1.2 - 10.9
0.1 - 19.9 - 114.4	0.2 - 18.0 - 103.3	0.1 - 13.9 - 104.2
0.1 - 25.0 - 144.2	0.3 - 22.7 - 133.6	0.1 - 17.2 - 137.7
0.1 - 24.2 - 141.3	0.1 - 22.8 - 138.6	0.1 - 17.3 - 138.3
0.2 - 24.0 - 136.7	0.2 - 22.2 - 132.6	0.1 - 16.7 - 137.1
0.1 - 22.5 - 120.8	0.2 - 21.6 - 131.6	0.1 - 15.9 - 135.5
0.1 - 21.7 - 119.2	0.2 - 21.0 - 124.0	0.1 - 15.5 - 134.0
	0.2 - 20.4 - 119.2	0.1 - 14.8 - 128.8
	0.2 - 19.2 - 99.9	0.1 - 14.2 - 117.8
	0.2 - 18.6 - 93.1	0.1 - 13.7 - 117.0
	0.2 - 17.9 - 90.9	0.1 - 13.1 - 115.9
		0.1 - 12.5 - 114.8
		0.1 - 12.0 - 114.5
		0.1 - 11.6 - 113.7
		0.1 - 11.2 - 110.3
		0.1 - 10.7 - 109.0

Πίνακας 4. Επίδραση του σχετικού εύρους προθεσμιών.

προτεραιότητας για τα αρχικά αποδοτικά σημεία και για περισσότερα τέτοια. Το τελευταίο φαίνεται και από τον πίνακα όπου ο μέγιστος χρόνος για τον υπολογισμό ενός αποδοτικού σημείου εμφανίζεται στο τρίτο σημείο για r=0.2 ενώ για 0.5 και 0.8, κατά μέσο όρο, στο τέταρτο.

Ένας άλλος λόγος είναι ότι εφόσον για μεγάλο r υπάρχουν μικρότερες προθεσμίες από ότι για μικρό, (για το ίδιο t), καθώς πάλι προχωράμε από το ένα αποδοτικό σημείο στο επόμενο, πιο γρήγορα γίνονται αρκετά μικρές οι προθεσμίες ώστε να βρίσκει εφαρμογή η πρόταση 2.4 δημιουργώντας έτσι περισσότερες σχέσεις προτεραιότητας. Για τον ίδιο λόγο και ο μηχανισμός διαγραφής κόμβων λόγω μη εφικτής συνέχειας γίνεται πιο αποτελεσματικός. Το γεγονός , βέβαια, ότι οι πιο μεγάλες προθεσμίες γίνονται ακόμα μεγαλύτερες, για μεγάλο r, αποδυναμώνει τον τελευταίο κάπως αλλά οι διαγραφές που δεν γίνονται λόγω αυτής της αύξησης είναι λιγότερο σημαντικές γιατί γίνονται σε πιο προχωρημένο στάδιο του δυναμικού προγραμματισμού αποσοβώντας έτσι μικρότερο αριθμό πράξεων από τις επιπρόσθετες διαγραφές που γίνονται λόγω της μείωσης των πιό μικρών προθεσμιών.

Παρατηρούμε ακόμα ότι όσο αυξάνει το r αυξάνει και το πλήθος των αποδοτικών σημείων του προβλήματος, κάτι που έχει σαν αποτέλεσμα να αυξάνεται ο αριθμός των προβλημάτων που δεν λύνονται λόγω υπέρβασης του 20λέπτου (όπως γινόταν και με το συντελεστή καθυστέρησης) παρ’ ότι ο χρόνος υπολογισμού του κάθε αποδοτικού σημείου μειώνεται. Ένας λόγος που συμβαίνει αυτό είναι ότι όσο μεγαλύτερο είναι το r τόσο μεγαλύτερες τιμές παίρνουν μερικές (αρχικές) προθεσμίες οπότε καθώς προχωράμε από το ένα αποδοτικό σημείο στο επόμενο και οι προθεσμίες που είχαν μετακινηθεί αρχικά αρκετά δεξιά ολισθαίνουν σταδιακά αριστερά, τόσο περισσότερο αριστερά θα χρειαστεί να μετακινηθούν ώστε να γίνουν και οι μεγαλύτερες από αυτές αρκετά μικρές ώστε να μην υπάρχει εφικτή λύση (όταν, βέβαια, οι αρχικές προθεσμίες είναι αρκετά μικρές ώστε να μη φθάνουν μέχρι αυτές οι ολισθαίνουσες και σταματήσει εκεί το πρόγραμμα) οπότε τόσο περισσότερα αποδοτικά σημεία θα υπολογισθούν κατά τη διαδικασία αυτή. Μία μαθηματική έκφραση του παραπάνω αποτελεί το ότι η μέγιστη (αρχική) προθεσμία είναι φραγμένη από το $(1-t+r/2)\cdot P$ (όπου P είναι, όπως έχουμε πει, το $n\cdot\overline{p}$) και όταν οι προθεσμίες έχουν μετακινηθεί δεξιά κατά Δd από το $(1-t+r/2)\cdot P+\Delta d$. Μία αναγκαία συνθήκη για να υπάρχει εφικτή λύση είναι η παραπάνω ποσότητα να είναι μεγαλύτερη από τη μικρότερη δυνατή τιμή του C_{max} (δηλαδή το $n\cdot\overline{p}+b\cdot\overline{S}$). Όσο μεγαλύτερο είναι το r τόσο μικρότερο μπορεί να γίνει το Δd, δηλαδή τόσο αργότερα, καθώς, μικραίνει το τελευταίο, θα πάψει να ισχύει η αναγκαία αυτή συνθήκη.

Το γεγονός ότι οι προθεσμίες ολισθαίνουν περισσότερο αριστερά για μεγάλες τιμές του r γίνεται πιο σημαντικό λόγω του ότι όσο μικρότερες είναι οι προθεσμίες τόσο μικρότερη, εν γένει, είναι η διαφορά μεταξύ των T_{max} δύο "γειτονικών" αποδοτικών προγραμμάτων, όπως διαπιστώσαμε παρατηρώντας τα αποτελέσματα αρκετών πειραμάτων, δηλαδή τόσο πιο "πυκνή", όσον αφορά τις τιμές των T_{max}, είναι η εμφάνιση των αποδοτικών σημείων. Αυτό οφείλεται στο ότι όσο είναι ακόμα χαλαρές οι προθεσμίες, τόσο περισσότερες επιλογές υπάρχουν κατά τη δημιουργία των αποδοτικών προγραμμάτων με αποτέλεσμα αυτά, και τα T_{max} τους, να διαφέρουν περισσότερο μεταξύ τους, όπως επαληθεύεται και πειραματικά, απ' ότι όταν οι προθεσμίες έχουν στενέψει και οι επιλογές στη δημιουργία των αποδοτικών προγραμμάτων έχουν λιγοστέψει.

Ένας άλλος λόγος που, κάπως λιγότερο, συνεισφέρει κι αυτός στην ύπαρξη περισσότερων αποδοτικών σημείων για μεγάλες τιμές του r είναι ότι το T_{max} του πρώτου αποδοτικού σημείου, που κατά πάσα πιθανότητα εμφανίζεται σε εργασία με μικρή προθεσμία, είναι αρκετά μεγαλύτερο για μεγάλα r αφού η προθεσμία αυτή είναι αρκετά μικρότερη, με αποτέλεσμα, όχι μόνο να σταματάει η ολίσθηση των προθεσμιών πιο αριστερά όπως είπαμε, αλλά και να ξεκινάει (αφού βρεθεί το πρώτο σημείο) από πιο δεξιά επιμηκύνοντας ακόμα περισσότερο το διάστημα που αυτές θα μετακινηθούν.

6.3. Σχόλιο πάνω στην ταχύτητα του προγράμματός μας

Τελειώνοντας το κεφάλαιο αυτό θα θέλαμε να σχολιάσουμε την ταχύτητα του προγράμματός μας. Έχουμε πει ότι το μέγεθος των προβλημάτων που λύνονται σε άλλες εργασίες που ασχολούνται με προβλήματα Χρονικού Προγραμματισμού σε ένα λεπτό είναι συνήθως γύρω στις 30 εργασίες, χωρίς να λείπουν και περιπτώσεις που φθάνουν τις 50. Στη δική μας εργασία φθάσαμε μέχρι τις 20 εργασίες ή και λίγο περισσότερες. Η μεγάλη υπολογιστική πλοκή του προβλήματος (ο παράγοντας $b \cdot T$ δεν εμφανίζεται στην πλοκή προβλημάτων χωρίς χρόνους εξάρμωσης) και η έλλειψη (ή τουλάχιστον η αδυναμία εντοπισμού) κάποιας "κανονικότητας" που θα οδηγούσε σε περισσότερους και αποδοτικότερους μηχανισμούς ελάττωσης του χώρου έρευνας είναι οι βασικοί λόγοι γι' αυτό. Παρ' όλα αυτά, καταφέραμε να υπολογίζουμε το πρώτο αποδοτικό σημείο, και όχι μόνο αυτό πάντα, πολύ ταχύτερα απο τους Ahn και Hyun που ασχολούνται με το $1/s_{ij}/\sum C_i$ και των οποίων η εργασία είναι η μόνη από όσες έχουν πέσει στην αντίληψή μας που ασχολείται με το παραπάνω πρόβλημα παρουσιάζοντας χρόνους υπολογισμού της βέλτιστης λύσης του η οποία, όπως έχουμε πει, είναι το πρώτο αποδοτικό σημείο του δικού μας προβλήματος.

Συγκεκριμένα, ορισμένα από τα αποτελέσματα που εκείνοι παρουσιάζουν είναι (όλοι οι χρόνοι είναι σε δευτερόλεπτα): προβλήματα με $n=10$ και $b=5$ λύνονται κατά μέσο όρο σε 1.4 δευτερόλεπτα, με $n=12$ και $b=4$ σε 1, με $n=15$ και $b=5$ σε 6.5, με $n=16$ και $b=4$ σε 2.5, με $n=20$ και $b=4$ σε 5.4 και με $n=20$ και $b=5$ σε 21. Οι χρόνοι αυτοί είναι αρκετά μεγάλοι αν συγκριθούν με κάποια δικά μας αποτελέσματα: το πρώτο αποδοτικό σημείο προβλημάτων με $n=18$ και $b=5$ υπολογίζεται σε 0.9 δευτερόλεπτα, με $n=20$ και $b=5$ σε 1.4 και με $n=22$ και $b=5$, πάλι, σε 2.2. Οι χρόνοι που παρουσιάζουν εκείνοι είναι δηλαδή μεγαλύτεροι από χρόνους που απαιτεί το δικό μας πρόβλημα για να λύσει μεγαλύτερα προβλήματα. Για $n=20$ και $b=5$ που είναι η μόνη κοινή περίπτωση ο χρόνος που παρουσιάζουν εκείνοι είναι 15 φορές μεγαλύτερος από το χρόνο που απαιτείται για τον υπολογισμό του πρώτου αποδοτικού σημείου (του προβλήματός μας από το πρόγραμμά μας) και (παρατηρώντας τον πίνακα 1) μεγαλύτερος από το μισό του χρόνου υπολογισμού του χρόνου υπολογισμού των υπόλοιπων αποδοτικών σημείων. Τα ίδια συμπεράσματα βγαίνουν και από τα παρακάτω αποτελέσματα που παρουσιάζουν: προβλήματα με $n=14$ και $b=7$ λύνονται σε 27.7 δευτερόλεπτα, με $n=16$ και $b=8$ σε 116 και με $n=18$ και $b=6$ σε 39.6 όταν για $n=18$ και $b=8$ το πρόγραμμά μας απαιτεί μόνο 6.7 δευτερόλεπτα για τον υπολογισμό του πρώτου αποδοτικού σημείου και μέχρι 48.6 για καθένα από τα υπόλοιπα. Βλέπουμε ότι ο χρόνος που παρουσιάζουν εκείνοι για $n=16$ και $b=8$ είναι 17.3 φορές μεγαλύτερος από το χρόνο που χρειάζεται το πρόγραμμά μας για τον υπολογισμό του πρώτου αποδοτικού σημείου για n μεγαλύτερο κατά 2 και ίδιο b και τουλάχιστον 2.4 φορές μεγαλύτερος από το χρόνο υπολογισμού των υπόλοιπων αποδοτικών σημείων (για τα ίδια, πάλι, n και b).

Στη διαφορά της ταχύτητας των προγραμμάτων συντελεί βέβαια και η διαφορά στο περιβάλλον που αυτά εκτελέσθηκαν, αλλά αυτό δεν είναι αρκετό για να αιτιολογήσει το μέγεθός της. Μπορούμε να συμπεράνουμε λοιπόν ότι η ταχύτητα του προγράμματός μας είναι αρκετά καλή αφού οι χρόνοι επίλυσης του $1/s_{ij}/\sum C_i$ που παρουσιάζονται στην πρόσφατα δημοσιευμένη εργασία των Ahn-Hyun είναι μεγαλύτεροι όχι μόνο από τους χρόνους υπολογισμού του πρώτου αποδοτικού σημείου του προβλήματός μας, αλλά, κάποιες φορές, ακόμα και από τους χρόνους υπολογισμού των υπολοίπων.

7. Επίλογος

7.1. Ανασκόπηση - Συμπεράσματα

Στην παρούσα εργασία ασχοληθήκαμε με το πρόβλημα Χρονικού Προγραμματισμού $1/s_{ij}/\sum C_i,T_{max}$ δηλαδή εκείνο στο οποίο δοθείσης μιας μηχανής και n εργασιών χωρισμένων σε b ομάδες ζητούνται τα αποδοτικά προγράμματα με κριτήρια το άθροισμα των χρόνων αποπεράτωσης των εργασιών ($\sum C_i$) και τη μέγιστη καθυστέρηση (T_{max}) όταν μεταξύ των εκτελέσεων δύο εργασιών που ανήκουν σε διαφορετικές ομάδες παρεμβάλλεται ένας χρόνος εξάρμωσης (s_{ij}) κατά τον οποίο η μηχανή μένει ανενεργή (δεν εκτελεί δηλαδή καμία εργασία). Προβλήματα Χρονικού Προγραμματισμού εμφανίζονται κυρίως στο Λεπτομεριακό Προγραμματισμό Παραγωγής ενός βιομηχανικού περιβάλλοντος. Επιθυμητό είναι η λύση να δίνεται σε σχεδόν "πραγματικό χρόνο", γενικά της τάξης του ενός λεπτού.

Εχουν δημοσιευτεί πολλές εργασίες που ασχολούνται με προβλήματα Χρονικού Προγραμματισμού αλλά η συντριπτική τους πλειοψηφία αναφέρεται σε προβλήματα χωρίς χρόνους εξάρμωσης, στα οποία είναι ευκολότερο να βρει κανείς κάποια "κανονικότητα" που θα βοηθήσει στην εξαγωγή μηχανισμών που θα συντελέσουν, συνήθως μέσω του περιορισμού του χώρου έρευνας σε ταχύτερη επίλυση του προβλήματος. Στις λίγες εργασίες που ασχολούνται με προβλήματα Χρονικού Προγραμματισμού με χρόνους εξάρμωσης εξετάζονται, κυρίως, το $1/s_{ij}/\sum C_i$ και το $1/s_{ij}/T_{max}$.

Επειδή στον Λεπτομερειακό Προγραμματισμό Παραγωγής ενδιαφέρεται συνήθως κανείς για προγράμματα (διατάξεις εργασιών) που έχουν μια γενικά καλή συμπεριφορά ως προς διάφορα κριτήρια, τα πολυκριτηριακά προβλήματα Χρονικού Προγραμματισμού αποτελούν μια ρεαλιστικότερη εκδοχή της πραγματικότητας. Η γενικότερη αντιμετώπισή τους είναι ο υπολογισμός του συνόλου των αποδοτικών σημείων αφού είτε τα κριτήρια ιεραρχηθούν είτε συνδυαστούν σε μία αντικειμενική συνάρτηση (ενέργειες που αφενός δεν είναι γενικά εύκολο να γίνουν έτσι ώστε να αντικατοπτρίζουν με ρεαλισμό τις προτιμήσεις του χρήστη και αφετέρου υπόκεινται σε πιθανές αλλαγές αφού και οι προτιμήσεις του χρήστη αλλάζουν) το πρόγραμμα που προκύπτει ως λύση είναι ένα από τα αποδοτικά προγράμματα.

Τα αποδοτικά σημεία του προβλήματος που μας απασχόλησε υπολογίζονται, ακολουθώντας την ιδέα των Van Wassenhove και Gelders [WaGe-80] για το ίδιο πρόβλημα χωρίς χρόνους εξάρμωσης, με επαναληπτική επίλυση του $1/s_{ij}/\sum C_i|T_{max}=0$ για κατάλληλα μετατοπισμένες δεξιά προθεσμίες οι οποίες μετά τον υπολογισμό κάθε αποδοτικού σημείου, ολισθαίνουν αριστερότερα έτσι ώστε το αποδοτικό σημείο που μόλις υπολογίστηκε να έχει $T_{max}=1$ (θεωρώντας ότι όλα τα μεγέθη του προβλήματος είναι ακέραιοι αριθμοί). Το πρώτο αποδοτικό σημείο που υπολογίζεται με τον παραπάνω τρόπο αποτελεί λύση του $1/s_{ij}/\sum C_i$ και το τελευταίο του $1/s_{ij}/T_{max}$. Η παραπάνω αναγωγή μας οδήγησε στο να στραφούν οι προσπαθειές μας στην επίλυση του $1/s_{ij}/\sum C_i|T_{max}=0$.

Για τα $1/s_{ij}/\sum C_i$ και $1/s_{ij}/T_{max}$ που έχουν, κυρίως το πρώτο, μελετηθεί, όπως είπαμε, περισσότερο έχει αποδειχθεί ([MoPo-89]) ότι είναι προβλήματα διατεταγμένων ομάδων, δηλαδή υπάρχει βέλτιστο πρόγραμμα στο οποίο η διάταξη σε κάθε ομάδα είναι από την αρχή γνωστή. Αυτό είναι πολύ σημαντικό γιατί ο αριθμός των εφικτών συνόλων, δηλαδή εκείνων που κάποια διάταξη των εργασιών τους μπορεί να βρίσκεται στην αρχή ενός βέλτιστου προγράμματος (αριθμός που παίζει καθοριστικό ρόλο στην πλοκή, άρα και στην ταχύτητα, ενός αλγορίθμου επίλυσης του προβλήματος) είναι εκθετικός ως προς b, και συγκεκριμένα όχι μεγαλύτερος από $(n/b+1)^b$ (που εμφανίζεται όταν οι εργασίες είναι ομοιόμορφα κατανεμημένες στις ομάδες). Για τα προβλήματα που δεν έχουν αυτή την ιδιότητα ο αντίστοιχος αριθμός είναι 2^n, εκθετικός, δηλαδή, ως προς n και γενικά πολύ μεγαλύτερος του $(n/b+1)^b$. Δυστυχώς και το δικό μας πρόβλημα φαίνεται να ανήκει στην κατηγορία αυτή αφού οι βέλτιστες λύσεις του προβλήματος όταν αυτό λυθεί σε κάθε ομάδα ξεχωριστά επηρεάζονται ελάχιστα (ή καθόλου) από τις προθεσμίες αφού τα C_i είναι αρκετά μικρά με αποτέλεσμα να μην είναι, γενικά, ούτε καν εφικτή η συγχώνευσή τους.

Η μέθοδος που συνήθως χρησιμοποιείται για την επίλυση προβλημάτων Χρονικού Προγραμματισμού είναι ένας αλγόριθμος διαμερισμού και φραγής που σε κάθε κόμβο του σχηματιζόμενου δένδρου αντιστοιχεί μια μερική διάταξη των εργασιών, που έχει προκύψει προσθέτοντας μια εργασία στο τέλος της διάταξης που αντιστοιχεί στον κόμβο-πατέρα του συγκεκριμένου κόμβου. Αρκετά λιγότερες εργασίες από όσες χρησιμοποιούν το παραπάνω γενικό σχήμα, ακολουθούν ένα σχήμα Δυναμικού Προγραμματισμού κάθε στάδιο του οποίου αντιστοιχεί στο σύνολο των εφικτών συνόλων με ένα συγκεκριμένο πληθάριθμο και, όταν δεν υπάρχουν χρόνοι εξάρμωσης, κάθε κατάσταση αντιστοιχεί σε ένα συγκεκριμένο εφικτό σύνολο. Οταν υπάρχουν χρόνοι εξάρμωσης οι καταστάσεις αντιστοιχούν στο σύνολο των διατάξεων ενός συνόλου που επιπλέον τελειώνουν με εργασία της ίδιας ομάδας και, γενικά, την ίδια χρονική στιγμή, εφόσον τα δύο τελευταία στοιχεία επηρεάζουν το κόστος του οποιουδήποτε συμπληρώματος της διάταξης και το τελευταίο (η στιγμή περάτωσης της τελευταίας εργασίας) δεν εξάγεται από τη γνώση της ταυτότητας του συνόλου των εργασιών. Σπάνια χρησιμοποιείται και 0-1 Γραμμικός Προγραμματισμός χωρίς όμως ενθαρρυντικά αποτελέσματα λόγω του μεγάλου αριθμού μεταβλητών που έχει η διαμόρφωση ενός προβλήματος Χρονικού Προγραμματισμούσε σε πρόβλημα 0-1 Γραμμικού Προγραμματισμού και της εκθετικής, ως προς τον αριθμό των μεταβλητών, πλοκής της επίλυσης του προβλήματος Γραμμικού Προγραμματισμού. Υπάρχουν βέβαια και εργασίες που αναπτύσσουν αλγορίθμους επίλυσης για συγκεκριμένα προβλήματα Χρονικού Προγραμματισμού, που δεν εντάσσονται σε κάποια από τις παραπάνω κατηγορίες.

Τα δύο παραπάνω σχήματα διαμερισμού και φραγής και δυναμικού προγραμματισμού είναι συγγενή μεταξύ τους παρόλο που μπορεί να διαφέρουν σημαντικά στις επιδόσεις τους. Και στα δύο επιχεειρείται μία έμμεση απαρίθμηση του συνόλου των (εφικτών) διατάξεων. Η διαφορά βρίσκεται κυρίως στον τρόπο περιορισμού του χώρου έρευνας. Στο σχήμα διαμερισμού και φραγής που σε κάθε κόμβο αντιστοιχεί μια διάταξη ενός υποσυνόλου, ο χώρος έρευνας περιορίζεται κυρίως με το γνωστό μηχανισμό κάτω φράγματος, ενώ στο σχήμα δυναμικού προγραμματισμού, που σε κάθε κατάσταση αντιστοιχεί ένα υποσύνολο εργασιών, ο περιορισμός αυτός επιτυγχάνεται μέσω του γεγονότος ότι για κάθε υποσύνολο εργασιών η έρευνα συνεχίζεται για τις διατάξεις που προκύπτουν συμπληρώνοντας μία μόνο, όταν δεν υπάρχουν χρόνοι εξάρμωσης, διάταξη των εργασιών. Και χρόνοι εξάρμωσης να υπάρχουν, οι διατάξεις για τις οποίες συνεχίζεται η έρευνα είναι πολύ λιγότερες (φραγμένες από το $b \cdot (k-b) \cdot \bar{S}$, όταν το σύνολο έχει k εργασίες) από τις $k!$ διατάξεις που παραμένουν στο σχήμα διαμερισμού και φραγής, αν δεν έχουν διαγραφεί από το μηχανισμό του κάτω φράγματος. Ο αριθμός, δηλαδή, των κόμβων και η υπολογιστική πλοκή στον δυναμικό προγραμματισμό έχουν σαν κύριο παράγοντα το 2^n (άνω φράγμα στο πλήθος των εφικτών συνόλων) ενώ οι αντίστοιχες εκφράσεις για το σχήμα διαμερισμού και φραγής το $n!$ (άνω φράγμα στο πλήθος των δυνατών διατάξεων). Αυτός είναι και ο λόγος που η ταχύτητα αλγορίθμων που ακολουθούν το πρώτο σχήμα είναι τις περισσότερες φορές σημαντικά μεγαλύτερη από την ταχύτητα αλγορίθμων που ακολουθούν το δεύτερο (όπως προκύπτει από τις συγκρίσεις που έχουν γίνει στις εργασίες [ScBa-780], [PoWa-85] και [APW-90]).

Η ταχύτητα των αλγορίθμων διαμερισμού και φραγής εξαρτάται κυρίως από το κάτω φράγμα που χρησιμοποιείται. Οταν υπολογίζεται σχετικά γρήγορα και είναι σχετικά σφικτό, γίνεται μεγάλος αριθμός διαγραφών κόμβων χωρίς να χάνεται γι’ αυτό σημαντικός χρόνος οπότε η ταχύτητα του αλγορίθμου είναι συγκρίσιμη με αυτή ενός αλγορίθμου που ακολουθεί το σχήμα δυναμικού προγραμματισμού. Λίγες όμως φορές γίνεται κατορθωτή η επινόηση ενός κάτω φράγματος με τα παραπάνω χαρακτηριστικά. Να σημειώσουμε ότι και στο σχήμα δυναμικού προγραμματισμού μπορεί να ενσωματωθεί μηχανισμός κάτω φράγματος, όπως προτείνουν οι Moris και Marsten ([MoMa-76]) αρκεί να υπάρχει κάποιο γνωστό άνω φράγμα (εκτός, βέβαια, από τον αλγόριθμο υπολογισμού κάτω φράγματος σε κάθε κατάσταση). Στο σχήμα διαμερισμού και φραγής, αυτό δεν είναι απαραίτητο γιατί άνω φράγματα αποτελούν οι υποβέλτιστες λύσεις που υπολογίζονται κατά την εξέλιξη του αλγορίθμου όταν η στρατηγική έρευνας είναι depth-first-search, όπως γίνεται συνήθως. Σαν αρχικό άνω φράγμα, στον δυναμικό προγραμματισμό, μπορεί να ληφθεί η τιμή κάποιας εφικτής λύσης. Στο δικό μας πρόβλημα όμως κάτι τέτοιο δεν έχει πρακτική εφαρμογή, αφού η εύρεση μιας εφικτής λύσης είναι NP-complete πρόβλημα.

Ο λόγος που συνήθως χρησιμοποιείται το σχήμα διαμερισμού και φραγής για την επίλυση προβλημάτων Χρονικού Προγραμματισμού παρ’ ότι είναι, γενικά, αργότερο από το σχήμα Δυναμικού Πραγραμματισμού είναι ότι το πρώτο έχει ελάχιστες απαιτήσεις σε μνήμη (όταν η στρατηγική έρευνας είναι depth-first-search) ενώ το δεύτερο έχει εκθετική απαίτηση και δημιουργούνται προβλήματα μνήμης πριν οι χρόνοι επίλυσης πάρουν μεγάλες τιμές. Μελετώντας διάφορες εργασίες διαπιστώσαμε ότι οι μέθοδοι που χρησιμοποιούνταν εκεί, συνήθως για υπολογισμό κάτω φράγματος, δεν μπορούσαν να τροποποιηθούν κατάλληλα ώστε να εφαρμόζονται και στη δική μας περίπτωση. Μερικά από τα βασικότερα και πιο συνηθισμένα αίτια γι’ αυτό, που κι αυτά με τη σειρά τους πηγάζουν από την ύπαρξη χρόνων εξάρμωσης, είναι η άγνοια του C_{max} ενός συνόλου εργασιών, το ενδεχόμενο μεταβολής αυτού από την ανταλλαγή δύο εργασιών, το γεγονός ότι η εύρεση εφικτής λύσης είναι NP-complete πρόβλημα και η ανυπαρξία ενός πολυωνυμικού αλγορίθμου για το ίδιο πρόβλημα χωρίς προθεσμίες και μιας ικανής συνθήκης βελτίστου του $1/s_{ij}/\sum w_i C_i$.

Δύο μέθοδοι υπολογισμού κάτω φράγματος που θα μπορούσαν να εφαρμοσθούν στο πρόβλημά μας ([Fish-76], [AbPo-88]) είχαν μεγάλη υπολογιστική πλοκή λόγω της οποίας είχαν τις λιγότερο καλές επιδόσεις από διάφορες μεθόδους που συγκρίθηκαν στην εργασία [APW-90]. Επίσης συγκεκριμένοι λόγοι για την καθεμία τις καθιστούσαν λιγότερο (μέχρι και καθόλου, τη μία από τις δύο, κάτω από κατάλληλες συνθήκες) αποτελεσματικές για το πρόβλημα μας, από ότι στην περίπτωση που εφαρμόστηκαν. Το κάτω φργμα, επίσης, που θα μπορούσε να προκύψει παραλλάσοντας κατάλληλα την ιδέα του Γεωργίου [Γεωρ-91] για το $1/s_{ij}/T_{max}$, αναμενόταν πολύ χαλαρό λόγω της μη ουσιαστικής παρέμβασης των χρόνων εξάρμωσης στον υπολογισμό του.

Αφού δεν στάθηκε δυνατή η επινόηση κάποιου σχετικά γρήγορα υπολογιζόμενου και σχετικά σφιχτού κάτω φράγματος για το πρόβλημά μας κι εφόσον η διαθέσιμη μνήμη των 64 MBytes ήταν πολύ μεγάλυτερη από αυτές που αναφέρονται στις εργασίες που είχαμε υπόψιν μας, υλοποιήσαμε ένα αλγόριθμο Δυναμικού Προγραμματισμού για την επίλυση του προβλήματος μας. Άλλωστε το μέγεθος των προβλημάτων που λύνονται σε χρόνο ενός λεπτού, ακόμα και για το, ευκολότερο από το δικό μας, $1/s_{ij}/\sum C_i$ (όπως φαίνεται στην εργασία [AhHy-90]), είναι σχετικά μικρό, οπότε, όπως φάνηκε από τα πειράματα που εκτελέσαμε, δεν εμφανίζονται σχεδόν καθόλου προβλήματα μνήμης.

Ο αλγόριθμος Δυναμικού Προγραμματισμού που υλοποιήσαμε ακολουθεί το γενικό σχήμα που δώσαμε παραπάνω κι έχει πλοκή $O(2^n \cdot n \cdot (b \cdot T + n))$. Το T δηλώνει το πλήθος των διαφορετικών τιμών που μπορεί να πάρει το C_{max} και για σταθερούς χρόνους εξάρμωσης φράσσεται από το $n-b$, ενώ διαφορετικά από το $(n-b)\cdot\bar{S}$. Εφόσον το πρόβλημα δεν είναι διατεταγμένων ομάδων δεν αρκούσε η αποθήκευση του πλήθους των εργασιών κάθε ομάδας που ανήκουν σ' ένα εφικτό σύνολο για να καθοριστεί η ταυτότητα του συνόλου αυτού, αλλά χρειαζόταν η άμεση περιγραφή του, οπότε ο αριθμός των εφικτών συνόλων, άρα και των καταστάσεων και τελικά η συνολική πλοκή του αλγορίθμου δεν είναι εκθετική ως προς b αλλά ως προς n.

Το πλήθος των εφικτών συνόλων περιορίζεται όπως γίνεται συνήθως, από κανόνες προτεραιότητας μεταξύ των εργασιών. Σχέσεις κυριαρχίας μεταξύ καταστάσεων βοηθούν επίσης στον περιορισμό του χώρου έρευνας. Διαγραφές καταστάσεων γίνονται ακόμα και όταν διαπιστωθεί ότι κάποια τέτοια δεν έχει εφικτή συνέχεια. Να σημειώσουμε ότι η πρόταση παραγωγής σχέσεων κυριαρχίας αποδείχθηκε για καταστάσεις που απλά αντιστοιχούν στο ίδιο εφικτό σύνολο, αλλά χρησιμοποιήθηκε όταν επιπλέον οι αντίστοιχες διατάξεις τελειώνουν σε εργασίες της ίδιας ομάδας, γιατί από τα πειράματα που έγιναν προέκυψε ότι ο χρόνος, που απαιτείται για τον έλεγχο της ισχύος της πρότασης είναι σημαντικότερος από τη μείωση του χρόνου, που οφείλεται στον περιορισμό του χώρου έρευνας λόγω κυριαρχίας διατάξεων με τελευταία εργασία από διαφορετική ομάδα, επειδή κάτι τέτοιο δεν συμβαίνει αρκετά συχνά.

Για την απαρίθμηση και αποθήκευση των εφικτών συνόλων και των καταστάσεων που αντιστοιχούν σε καθένα από αυτά χρησιμοποιήσαμε δύο αλγορίθμους από τους οποίους ο πρώτος ([Lawl-79]), που δημιουργεί τα σύνολα με αύξουσα σειρά πληθάριθμου, απαιτεί λιγότερη μνήμη και ο δεύτερος ([ScBa-78]), που τα δημιουργεί με λεξικογραφική σειρά, είναι λίγο ταχύτερος. Από τα πειράματα που εκτελέσαμε αποδείχθηκε ότι ο χρόνος είναι περιοριστικότερος παράγοντας από τη μνήμη, κάτι που λογικά οφείλεται στο μεγάλο μέγεθος της διαθέσιμης μνήμης, στον περιορισμένο χρόνο μέσα στον οποίον απαιτούμε να λύνονται τα προβλήματα και στην ιδιαίτερη δυσκολία (που μεταφράζεται σε μεγάλο υπολογιστικό χρόνο) του προβλήματος κάτι που φαίνεται και απο τα αποτελέσματα της εργασίας [AhHy-90] που ασχολείται, όπως είπαμε, με το ευκολότερο πρόβλημα $1/s_{ij}/\sum C_i$. Γι αυτό και επιλέξαμε στην τελική μορφή του προγράμματος τον αλγόριθμο με την καλύτερη επίδοση στην ταχύτητα.

Από τα πειράματα που εκτελέσαμε διαπιστώσαμε ότι ο υπολογιστικός χρόνος αυξάνει σημαντικά καθώς αυξάνουν τα n και b, παρόλο που η πλοκή ως προς το δεύτερο είναι απλώς γραμμική. Αυτό οφείλεται στο ότι αύξηση του b οδηγεί σε μείωση των σχέσεων προτεραιότητας κι επομένως αύξηση του πλήθους των εφικτών συνόλων που σχετίζεται άμεσα με την ταχύτητα του αλγορίθμου.

Διαπιστώσαμε επίσης ότι ο χρόνος που απαιτείται για τον υπολογισμό του πρώτου αποδοτικού σημείου (της λύσης, δηλαδή, του $1/s_{ij}/\sum C_i$) είναι πολύ μικρότερος από τους αντίστοιχους χρόνους των άλλων αποδοτικών σημείων. Αυτό οφείλεται στο ότι επειδή οι προθεσμίες αγνοούνται, ουσιαστικά, κατά τον υπολογισμό του πρώτου αποδοτικού σημείου οι εργασίες σε κάθε ομάδα είναι από την αρχή διατεταγμένες σε μία βέλτιστη διάταξη, όπως γίνεται στο $1/s_{ij}/\sum C_i$, οπότε το πλήθος των εφικτών συνόλων είναι πολύ μικρότερο από τη γενική περίπτωση (φραγμένο από $(n/b+1)^b$, όπως είπαμε, κι όχι από 2^n). Προχωρώντας από το ένα αποδοτικό σημείο στο επόμενο ο χρόνος υπολογισμού του αυξάνει για λίγο και μετά σταδιακά φθίνει, χωρίς να πλησιάσει την πολύ μικρότερη τιμή που παίρνει στο πρώτο αποδοτικό σημείο. Αυτό οφείλεται στο ότι όταν οι προθεσμίες είναι ακόμα αρκετά μεγάλες μερικές από αυτές (όσες είναι μεγαλύτερες από ένα άνω φράγμα του C_{max}) θεωρούνται ίσες, κάτι που βοηθάει στην εξαγωγή κανόνων προτεραιότητας. Όσο πάλι μικραίνουν, μπορεί το παραπάνω να μην ισχύει αλλά γίνονται αποτελεσματικότεροι οι μηχανισμοί περιορισμού του χώρου έρευνας που βασίζονται στο θέμα της εφικτότητας και ενισχύονται με τις μικρές προθεσμίες.

Το πρόγραμμα που υλοποιεί τον αλγόριθμό μας δεν καταφέρνει να λύσει αρκετά μεγάλα προβλήματα σε παραδεκτό χρόνο, αφού φτάνει μέχρι τις λίγο περισότερες από 20 εργασίες, ενώ άλλες εργασίες που λύνουν άλλα προβλήματα Χρονικού Προγραμματισμού λύνουν προβλήματα μέχρι περίπου 30 εργασιών χωρίς να λείπουν και εργασίες που φτάνουν τις 50. Αυτό οφείλεται στην μεγάλη υπολογιστική πλοκή του προβλήματος και στην έλλειψη κάποιας "κανονικότητας" που θα βοηθούσε στην εξαγωγή περισσότερων και αποτελεσματικότερων μηχανισμών περιορισμού του χώρου έρευνας, που με τη σειρά τους οφείλονται στην ύπαρξη των χρόνων εξάρμωσης και στο γεγονός ότι το πρόβλημα δεν είναι διατεταγμένων ομάδων. Παρ' όλα αυτά το πρόγραμμά μας υπολογίζει το πρώτο αποδοτικό σημείο και μερικές φορές και τα υπόλοιπα πολύ ταχύτερα απ' ότι οι Ahn-Hyun λύνουν το $1/s_{ij}/\sum C_i$ που, όπως είπαμε, αντιστοιχεί στο πρώτο αποδοτικό μας σημείο. Επομένως μπορούμε να ισχυριστούμε ότι η ταχύτητα του προγράμματός μας είναι αρκετά καλή για το πρόβλημα που λύνει.

Ένα γενικό συμπέρασμα που εξάγεται από την παρούσα εργασία είναι ότι αντιμετωπίζοντας κανείς ένα δύσκολο πρόβλημα Χρονικού Προγραμματισμού (ή ίσως και γενικώτερα κάποιο συνδυαστικό πρόβλημα που λύνεται με έμμεση απαρίθμηση) που καλείται να το λύσει σε (σχεδόν) πραγματικό χρόνο σε ένα σύγχρονο σταθμό εργασίας με την πολύ μεγάλη μνήμη που αυτός μπορεί να διαθέτει, ο χρόνος αποδεικνύεται περιοριστικότερος παράγοντας από τη μνήμη. Γι αυτό και οι αλγόριθμοι Δυναμικού Προγραμματισμού που στο παρελθόν δεν είχαν χρησιμοποιηθεί αρκετά, μοιάζουν να γίνονται πιο αποτελεσματικοί από τους πιο φειδωλούς σε μνήμη αλλά, γενικά, βραδύτερους αλγόριθμους διαμερισμού και φραγής, εκτός κι αν καταφέρει κανείς να επινοήσει ένα αρκετά γρήγορο και αρκετά σφιχτό κάτω φράγμα για τους τελευταίους. Πάντως ο περιορισμός που τίθεται από τη μνήμη στο μέγεθος των προβλημάτων που λύνονται σε παραδεκτό χρόνο, που αποτελούσε το βασικό λόγο αποφυγής χρησιμοποίησης Δυναμικού Προγραμματισμού, αποκτάει δευτερεύουσα σημασία.

7.2. Πιθανές προεκτάσεις

Πολλές φορές το πρόβλημα Χρονικού Προγραμματισμού που διαμορφώνει και λύνει κανείς δεν αποτελεί παρά μια απλούστευση της πραγματικότητας που συναντάται στο βιομηχανικό περιβάλλον η οποία είναι πολύ σύνθετη και ίσως ασαφής (κυρίως οι στόχοι), ώστε να μην είναι δυνατόν όχι μόνο να λυθεί σε λογικό χρόνο κάποιο πρόβλημα που την παριστάνει με ακρίβεια αλλά ούτε καν να διατυπωθεί αυστηρά μαθηματικά. Για το λόγο αυτό δεν είναι απαραίτητο να καταβάλει κανείς κάθε δυνατή προσπάθεια να λύσει το πρόβλημα με ακρίβεια, αφού και αυτό δεν αποτελεί ακριβή απεικόνιση της πραγματικότητας. Επειδή, ακόμα, τα προβλήματα που εμφανίζονται στην πράξη έχουν συχνά πολύ μεγαλύτερο αριθμό εργασιών (μπορεί και δεκαπλάσιο) από αυτά που λύνονται σε χρόνο τάξης ενός λεπτού από τους (ακριβείς) αλγορίθμους που δημοσιεύονται σε διάφορες εργασίες, γίνεται αρκετές φορές απαραίτητη η χρησιμοποίηση προσεγγιστικών, γενικά ευρηματικών, αλγορίθμων που σε σύντομο χρονικό διάστημα βρίσκουν μια προσεγγιστική λύση του προβλήματος. Η ποιότητα ενός τέτοιου αλγορίθμου εξαρτάται από την ταχύτητά του, που συνήθως είναι μεγάλη, και κυρίως την ποιότητα της προσέγγισης.

Δεν είναι εύκολο να βρεθεί προσεγγιστικός αλγόριθμος για το $1/s_{ij}/\sum C_i|T_{max}=0$ αφού η εύρεση απλώς μιας εφικτής λύσης είναι NP-complete πρόβλημα. Μία ιδέα για την ανάπτυξη ενός ευρηματικού αλγορίθμου για την προσεγγιστική επίλυση του $1/s_{ij}/\sum C_i,T_{max}$ είναι η εξής: Χρησιμοποιώντας τον ευρηματικό κανόνα του

Τσατσάκη [Τσατ-93] μπορούμε να βρούμε μια προσέγγιση (με μεγάλη μάλιστα πιθανότητα επίτευξης βελτίστου) του πρώτου αποδοτικού σημείου (αυτού, δηλαδή, που αποτελεί λύση του $1/s_{ij}/\sum C_i$). Ελέγχοντας μετά όλους τους δυνατούς συνδυασμούς αντιμετάθεσης δύο συνεχόμενων τμημάτων μπορούμε σε χρόνο $O(n^3)$ να βρούμε τα συνεχόμενα εκείνα τμήματα που όταν αντιμετατεθούν δίνουν πρόγραμμα με το μικρότερο δυνατό $\sum C_i$ από όσα προγράματα έχουν T_{max} μικρότερο από εκείνο του αρχικού προγράμματος και προκύπτουν από αυτό με τον ίδιο τρόπο (δηλαδή με αντιμετάθεση δύο συνεχόμενων τμημάτων). Το πρόγραμμα που προκύπτει αποτελεί το δεύτερο αποδοτικό σημείο του ευρηματικού αλγορίθμου. Η συνέχεια γίνεται ακριβώς με τον ίδιο τρόπο, δημιουργώντας το κάθε ψεύδο-αποδοτικό σημείο (αν ονομασουμε έτσι τα σημεία που προτείνει σαν αποδοτικά ο ευρηματικός αλγόριθμος) αντιμεταθέτοντας δύο συνεχόμενα τμήματα του προηγούμενου ψεύδο-αποδοτικού σημείου. Το τέλος, βέβαια, έρχεται όταν για κάποιο ψεύδο-αποδοτικό σημείο δεν μπορεί να βρεθεί πρόγραμμα που να προκύπτει με τον παραπάνω τρόπο που να έχει T_{max} μικρότερο από το T_{max} αυτού του ψεύδο-αποδοτικού σημείου, ή όταν βρεθεί ένα τέτοιο σημείο με $T_{max}=0$.

Μία συμμετρική εκδοχή είναι να ξεκινάμε από τη λύση που ο ευρηματικός κανόνας του Γεωργίου [Γεωρ-91] υπολογίζει για το $1/s_{ij}/T_{max}$ (που έχει επίσης μεγάλη πιθανότητα να συμπίπτει με το βέλτιστο) και να προχωράμε από το ένα ψεύδο-αποδοτικό σημείο στο άλλο με παρόμοιο τρόπο, δηλαδή από όσα προγράμματα προκύπτουν από ένα τέτοιο σημείο με την αντιμετάθεση δύο συνεχόμενων τμημάτων του και έχουν μικρότερο $\sum C_i$ από αυτό να κρατιέται σα νέο ψεύδο-αποδοτικό σημείο αυτό με το μικρότερο T_{max}. Μπορούν να εκτελούνται και οι δύο παραπάνω εκδοχές και να κρατούνται τελικά ως λύση του ευρηματικού αλγορίθμου τα ψευδο-αποδοτικά σημεία που βρέθηκαν με τους δύο παραπάνω τρόπους για τα οποία δεν υπάρχει κάποιο ανάμεσά τους με μικρότερη τιμή στο ένα κριτήριο και όχι μεγάλυτερη στο άλλο.

Εάν διαπιστωθεί ότι η ταχύτητα του αλγορίθμου είναι μικρή τότε η πρώτη εκδοχή ίσως να μπορεί να επιταχυνθεί αν θέσουμε σαν επιπλέον περιορισμό να διατηρείται ο SPT στις παρτίδες, δηλαδή στα μέγιστα σύνολα συνεχόμενων εργασιών από την ίδια ομάδα. Κι αυτό γιατί μπορεί να αποδειχθεί ότι τότε η τελαυταία εργασία του πρώτου τμήματος και η πρώτη εργασία του δεύτερου τμήματος θα ανήκουν σε διαφορετικές ομάδες ελαττώνοντας έτσι σημαντικά το πλήθος των δυνατών συνδυασμών, κάτι, βέβαια, που μόνο αρνητικά μπορεί να επηρεάσει την ποιότητα του αλγορίθμου. Εάν όμως υλοποιηθεί αυτός ο περιορισμός θα χρειάζονται αρκετοί έλεγχοι για την επιβεβαίωση αφενός της διατήρησης του SPT στις παρτίδες μετά την αντιμετάθεση και αφετέρου του ότι η αντιμετάθεση που τελικά θα γίνει οδηγεί πράγματι στο πρόγραμμα με το μικρότερο $\sum C_i$ από όσα προκύπτουν με τον τρόπο αυτό (διατηρώντας και τον SPT στις παρτίδες) οπότε δεν αποκλείεται να μην υπάρξει, αξιόλογη τουλάχιστον, βελτίωση της ταχύτητας.

Εάν, αντίθετα, διαπιστωθεί ότι ο αλγόριθμος είναι αρκετά γρήγορος και θα μπορούσε να κάνει λίγο περισσότερη δουλειά για να προσεγγίσει καλύτερα τη βέλτιστη λύση μπορούμε να μην περιοριζόμαστε σε αντιμεταθέσεις συνεχόμενων μόνο τμημάτων, αυξάνοντας έτσι την πλοκή του σε $O(n^4)$.

Ανεξάρτητα από το εάν θα γίνεται αυτό ή όχι μπορούμε να ενεργήσουμε παρόμοια με τον ευρηματικό κανόνα των Ahn-Hyun για το $1/s_{ij}/\sum C_i$: Έστω ότι ξεκινάμε από κάποιο ψεύδο-αποδοτικό σημείο, π_1, και ελέγχουμε τις δυνατές αντιμεταθέσεις (συνεχόμενων) τμημάτων για να βρούμε το επόμενο, π_2, κι έστω ότι χρησιμοποιούμε την πρώτη από τις δύο εκδοχές που αναφέραμε αρχικά. Όταν βρεθεί το πρώτο ζεύγος τμημάτων που αν αντιμετατεθεί οδηγούμαστε σε πρόγραμμα με μικρότερο T_{max} από του π_1 εκτελείται η αντιμετάθεση και η έρευνα συνεχίζεται. Όταν βρεθεί κι άλλο ζεύγος τμημάτων η αντιμετάθεση του οποίου οδηγεί σε πρόγραμμα με T_{max} μικρότερο του π_1 και $\sum C_i$ μικρότερο αυτού που βρέθηκε πριν, η αντιμετάθεση πάλι εκτελείται και έτσι συνεχίζουμε. Το π_2 είναι αυτό που προκύπτει αφού τελειώσουν οι όποιες αντιμεταθέσεις γίνουν.

Ανάλογα με το σχετικό βάρος που δίνει κανείς στην ταχύτητα και στην ποιότητα της προσεγγιστικής λύσης μπορεί ματά από κάθε αντιμετάθεση να ξαναρχίζει ο έλεγχος των δυνατών αντιμεταθέσεων από την αρχή ή από κάποιο ενδιάμεσο σημείο της έρευνας (όπως κάνουν οι Ahn-Hyun). Όταν δεν ξαναρχίζει από την αρχή μπορεί όταν τελειώσει και βρεθεί κάποιο π_2, να επαναλαμβάνεται η ίδια διαδικασία μέχρι να μη βελτιώνεται άλλο το π_2.

Πιστεύουμε ότι ένας ευρηματικός αλγόριθμος που χρησιμοποιεί κάποια ή κάποιες από τις παραπάνω ιδέες έχει ελπίδες να δώσει μια καλή προσέγγιση γιατί, παρατηρώντας τα αποδοτικά σημεία διαφόρων προβλημάτων, διαπιστώσαμε ότι, συνήθως, καθένα από αυτά μπορεί να προκύψει με λίγες (ίσως και μόνο μία) αντιμεταθέσεις κάποιων τμημάτων του προηγούμενου. Πιθανώς, λοιπόν, να απέδιδε κάτι καλό μια έρευνα προς αυτήν την κατεύθυνση.

Τελειώνοντας θα θέλαμε να κάνουμε κάποια σχόλια σχετικά με τον αλγόριθμο που χρησιμοποιήσαμε για την επίλυση του $1/s_{ij}/\sum C_i|T_{max}=0$. Παρατηρούμε καταρχήν ότι επειδή κάποια διάταξη π_1 κυριαρχεί κάποιας άλλης διάταξης π_2 των ίδιων $n-k$ εργασιών που τελειώνει και σε εργασία της ίδιας ομάδας, μόνο αν ισχύει $C_{max}(\pi_1)\leq C_{max}(\pi_2)$ και $\sum C_i(\pi_1)+k\cdot C_{max}(\pi_1)\leq\sum C_i(\pi_2)+k\cdot C_{max}(\pi_2)$ δεν διαγράφεται κανένας κόμβος η αντίστοιχη διάταξη του οποίου αν συμπληρωθεί κατάλληλα οδηγεί σε αποδοτικό σημείο του $1/s_{ij}/C_{max},\sum C_i|T_{max}=0$. Κι αυτό γιατί κάθε πρόγραμμα που προκύπτει συμπληρώνοντας με κάποιο τρόπο το π_2, εάν είναι εφικτό, θα έχει μεγαλύτερο ή ίσο C_{max} και $\sum C_i$ από το πρόγραμμα που προκύπτει συμπληρώνοντας με τον ίδιο τρόπο την π^1 οπότε δεν θα είναι αποδοτική λύση. Έτσι όταν περατωθεί ο Δυναμικός Πραγραμματισμός οι καταστάσεις που αντιστοιχούν στο πλήρες σύνολο περιέχουν όλες αυτές που αντιστιχούν σε αποδοτικά σημεία του $1/s_{ij}/C_{max},\sum C_i|T_{max}=0$ οι οποίες μπορουν σε $O(b\cdot T)$ να εντοπισθούν, με τρόπο όμοιο με τη συγχώνευση των λιστών καταστάσεων που γινόταν κατά τη διάρκεια του Δυναμικού Πραγραμματισμού. Επομένως ο αλγόριθμος που χρησιμοποιήσαμε για την επίλυση του $1/s_{ij}/\sum C_i|T_{max}=0$ λύνει και το $1/s_{ij}/C_{max},\sum C_i|T_{max}=0$.

Να παρατηρήσουμε ακόμα ότι μπορούμε να θεωρήσουμε ως καταστάσεις τα σύνολα των διατάξεων ίδιων εργασιών που τελειώνουν σε εργασίες της ίδιας ομάδας, ανεξάρτητα από το C_{max} της κάθε διάταξης. Με τη θεώρηση αυτή, που διαφέρει βέβαια μόνο θεωρητικά από αυτήν που χρησιμοποιήσαμε εμείς, σε κάθε τέτοια κατάσταση (που είναι ένα σύνολο - συγκεκριμένα μία λίστα -

καταστάσεων όπως τις ορίζουμε εμείς) δεν αντιστοιχεί απλώς η τιμή μιας συνάρησης κόστους, όπως γίνεται στον κλασσικό Δυναμικό Πραγραμματισμό, αλλά το σύνολο των αποδοτικών σημείων του $1/S_{ij}/C_{max},\sum C_i|T_{max}=0$ που αντιστοιχούν στο συγκεκριμένο σύνολο και τη συγκεκριμένη ομάδα, δηλαδή ένα σύνολο από διατεταγμένα ζεύγη αριθμών (οι τιμές των δύο παραπάνω κριτηρίων). Η θεώρηση αυτή είναι όμοια με κείνη της εργασίας [ViKa-81] για το γενικό πολυκριτηριακό πρόβλημα ακέραιου προγραμματισμου.

Μια τελευταία παρατήρηση είναι ότι το σχήμα Δυναμικού Πραγραμματισμού που χρησιμοποιήσαμε μπορεί με κατάλληλο ορισμό της συνάρτησης κόστους και κατάλληλους μηχανισμούς παραγωγής κανόνων προτεραιότητας και σχέσεων κυριαρχίας να λύσει οποιδήποτε πρόβλημα Χρονικού Προγραμματισμού μιας μηχανής με ή χωρίς χρόνους εξάρμωσης, με ένα ή περισσότερα αθροιστικά κριτήρια ή κριτήρια μεγίστου. Αυτό ισχύει γιατί με το συγκεκριμένο σχήμα δημιουργούνται όλες οι δυνατές διατάξεις όλων των εφικτών υποσυνόλων που δεν έχουν μεγαλύτερη τιμή στο προς ελαχιστοποίηση κριτήριο (ή κριτήρια) και στο C_{max} από κάποια άλλη διάταξη των ίδιων εργασιών με τελευταία εργασία από την ίδια ομάδα, δηλαδή που δεν αποκλείεται να οδηγήσουν σε βέλτιστη λύση. (Εμείς, βέβαια, ελέγχουμε τις ποσότητες $\sum C_i+k\cdot C_{max}$ αντί των $\sum C_i$, απλώς επειδή αποδικνύεται ότι με τον τρόπο αυτό επιτυγχάνονται περισσότερες διαγραφές καταστάσεων.)

Όταν το πρόβλημα έχει περισσότερα του ενός κριτήρια και υπάρχει κάποιο (ή κάποια) κριτήρια μεγίστου αναμεσά τους, εκτός από την περίπτωση που τα κριτήρια είναι ιεραρχημένα και το κριτήριο μεγίστου είναι το λιγότερο σημαντικό, το πρόβλημα αναγεται στην εύρεση των αποδοτικών σημείων, ανεξάρτητα από το εάν δίνεται μία αντικειμενική συνάρτηση που συνδυάζει τα κριτήρια ή αν αυτά είναι ιεραρχημένα. Κι αυτό γιατί μπορεί δύο μερικές διατάξεις να διαφέρουν, πολύ ίσως, στο κριτήριο μεγίστου αλλά δεν αποκλείεται σε κάποια κοινή συμπλήρωσή τους να εμφανίζεται μία εργασία με μεγαλύτερη τιμή σ' αυτό από ό,τι υπήρχε μέχρι τότε και παραπλήσια, ή ίδια, στις δύο συμπληρωμένες διατάξεις οπότε τελικά η προτίμηση μεταξύ των δύο διατάξεων να αντιστρέφεται, εάν αυτή που είχε τη μεγαλύτερη τιμή στο κριτήριο μεγίστου είχε μικρότερη στα υπόλοιπα. Όταν όλα τα κριτήρια είναι αθροιστικά δε χρειάζεται καμία αναγωγή και οι διαγραφές των κόμβων γίνονται ακολουθώντας τη γενική ιδέα που αναφέραμε παραπάνω.

Όπως και στο δικό μας πρόβλημα, έτσι και στη γενική περίπτωση η λύση του προβλήματος περιέχει τα αποδοτικά σημεία και ως προς C_{max}. Ακόμα, οι καταστάσεις μπορεί να αντιστοιχούν σε συγκεκριμένα υποσύνολα εργασιών και ομάδες όπου ανήκουν οι τελευταίες εργασίες των διατάξεων ή να ζητείται επιπλέον η ισότητα στα C_{max} (όπως τις θεωρήσαμε εμείς) ή και άλλων κριτηρίων στην περίπτωση των πολυκριτηριακών προβλημάτων.

Αφού λοιπόν μπορεί να λυθεί με το σχήμα αυτό οποιοδήποτε πρόβλημα Χρονικού Προγραμματισμού μιας μηχανής μπορεί και η άμεση γενίκευση του $1/S_{ij}/\sum C_i|T_{max}=0$ όταν υπάρχουν βάρη, δηλαδη το $1/S_{ij}/\sum w_iC_i|T_{max}=0$. Τότε προκύπτουν λιγότερες σχέσεις προτεραιότητας γιατί στην ικανή συνθήκη $p_i\leq p_j$ και $d_i\leq d_j$ για την εξαγωγή του συμπεράσματος ότι η εργασία i είναι πρόγονος της εργασίας j προστίθεται και το $w_i\geq w_j$, όμοια με την περίπτωση χωρίς χρόνους εξάρμωσης ([PoWa-81]). Αντικαθιστώντας το k με $\sum_{i \in R'} w_i$, όταν μένουν να διαταχθούν οι k εργασίες του συνόλου R′, στη σχέση που συνδέει τις ποσότητες $\sum C_i + k \cdot C_{max}$ στην πρόταση 2.5, η τελευταία ισχύει και για το $1/s_{ij}/\sum w_i C_i | T_{max}=0$.

Το πιο ενδιαφέρον, πιστεύουμε, είναι ότι εφόσον το γενικό αυτό σχήμα μπορεί να χρησιμοποιηθεί για οποιοδήποτε πρόβλημα μιας μηχανής, μπορεί να χρησιμοποιηθεί κατευθείαν και για το $1/s_{ij}/\sum C_i, T_{max}$. Εάν οι καταστάσεις ορισθούν όπως πριν, σε καθεμιά θα αντιστοιχεί το σύνολο των λύσεων αυτού του δικριτηριακού, με δεδομένο βέβαια ένα συγκεκριμένο σύνολο εργασιών, την ομάδα που ανήκει η τελευταία εργασία και το C_{max} όπως και πριν. Εναλλακτικά, σε κάθε κατάσταση μπορεί να αντιστοιχεί το υποσύνολο των παραπάνω διατάξεων που επιπλέον έχουν την ίδια τιμή σε ένα από τα δύο κριτήρια και θα αποθηκεύεται μόνο η τιμή του άλλου κριτηρίου του προγράμματος που έχει τη μικρότερη τέτοια τιμή. Μία τρίτη εκδοχή είναι να θεωρούμε ότι σε κάθε κατάσταση αντιστοιχούν όλες οι διατάξεις ενός συγκεκριμένου συνόλου εργασιών που τελειώνουν σε εργασία μίας συγκεκριμένης ομάδας ανεξάρτητα από C_{max}, $\sum C_i$ και T_{max} και θα αποθηκεύονται γι’ αυτήν τα αποδοτικά σημεία του τρικριτηριακού $1/s_{ij}/C_{max}, \sum C_i, T_{max}$. Η διαφορά μεταξύ των παραπάνω θεωρήσεων είναι, όπως έχουμε πει, καθαρά θεωρητική και δεν επηρεάζει καθόλου την υλοποίηση.

Τώρα (λύνοντας το $1/s_{ij}/\sum C_i, T_{max}$) δε θα λειτουργούν οι μηχανισμοί περιορισμού του χώρου έρευνας που βασίζονται στις προθεσμίες (πρόταση 2.4, διαγραφή κόμβων λόγω μη εφικτής συνέχειας) γιατί δε θα υπάρχουν μη εφικτές διατάξεις. Επίσης, στις ποσότητες δύο διατάξεων που συγκρίνονται μεταξύ τους για να ελεγθεί εάν κάποια ποσότητα κυριαρχεί κάποιας άλλης (C_{max}, $\sum C_i + k \cdot C_{max}$) προστίθεται τώρα και το T_{max} οπότε οι διαγραφές που προκύπτουν έτσι γίνονται σπανιότερες. Μέχρι τώρα η συγχώνευση δύο λιστών καταστάσεων είχε πλοκή $O(T)$ γιατί κάθε τέτοια λίστα ήταν ταξινομημένη κατά αύξουσα τάξη του ενός κριτηρίου και κατά φθίνουσα του άλλου (C_{max}, $\sum C_i$). Τώρα που προστίθεται και το T_{max} κι έχουμε τρία κριτήρια η ταξινόμηση μπορεί να παραμείνει μόνο ως προς το ένα κριτήριο. Αυτό έχει σαν αποτέλεσμα μετά την εισαγωγή ενός νέου κόμβου σε μία λίστα και το πέρας των ελέγχων για πιθανές διαγραφές που ακολουθούν, ο έλεγχος για την εισαγωγή του επόμενου κόμβου να μη συνεχίζεται από εκεί που είχε σταματήσει για τον προηγούμενο αλλά από εκεί που είχε εισαχθεί ο προηγούμενος, που με τη σειρά του έχει σαν αποτέλεσμα την αύξηση της πλοκής συγχώνευσης δύο λιστών σε $O(T^2)$. Έτσι συνολικά η πλοκή του αλγορίθμου αυξάνεται σε $O(2^n \cdot n \cdot (b \cdot T^2 + n))$. Για τους παραπάνω λόγους γίνεται αμφίβολο το εάν θα βρεθούν τα αποδοτικά σημεία του $1/s_{ij}/\sum C_i, T_{max}$ ταχύτερα, λύνοντάς το κατεθείαν όπως είπαμε. Πιστεύουμε πάντως ότι αξίζει να ερευνηθεί και το ενδεχόμενο αυτό.

Βιβλιογραφία

[AhHy-90]

Ahn B. και Hyun J., 1990, "Single Facility Multi-Class Job Scheduling", Comput. and Opns. Res. 17, 265-272.

[APW-90]

Abdul-Razaq T., Potts C. και Van Wassenhove L., 1990, "A survey of algorithms for the single machine total weighted tardiness scheduling problem", Discrete Applied Mathematics 26, 235-253.

[BaAh-87]

Baghi U. και Ahmadi R., 1987, "An Improved Lower Bound for Minimizing Weighted Completion Times with deadlines", Opns. Res. 35, 311-313.

[Bagh-89]

Baghi U., 1989, "Simultaneous minimization of mean and variation of flow time and waiting time in single machine systems", Opns. Res. 37, 118-125.

[Bans-80]

Bansal S., 1980, "Single Machine Scheduling to Minimize Weighted Sum of Completion Times with Secondary Criteria", Eur. J. Opns. Res. 5, 177-181.

[BaSc-78]

Baker K. και Schrage L., 1978, "Finding an optimal sequence by dynamic programming: An extension to precedence-related tasks", Opns. Res. 26, 111-120.

[BaVa-81]

Barnes J. και Vanston L., 1981, "Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs", Opns. Res. 29, 146-160.

[BrDo-78]

Bruno J. και Downey P., 1978, "Complexity of Task Sequencing with Deadlines, Set-Up Times and Changeover Costs", SIAM Journal on Computing 7, 393-404.

[Burn-76]

Burns R., 1976, "Scheduling to Minimize the Weighted Sum of Completion Times with Secondary Criteria", Naval Res. Logist. Quart. 23, 125-129.

[BuSt-81]

Burns R. και Steiner G., 1981, "Single machine scheduling with series-parallel precedence constraints", Opns. Res. 29, 1195-1207.

[CCMM-92]

Carraway R., Chambers R., Morin T. και Moskowitz H., 1992, "Single machine sequencing with nonlinear multicriteria cost functions: an application of generalized dynamic programming", Comput. Ops. Res. 19, 69-77.

[Γεωρ-91]

Γεωργίου Θ., 1991, "Το Πρόβλημα της Μεγιστης Καθυστέρησης σε μία Μηχανή με Χρόνους Εξάρμωσης", Μεταπτυχιακή Εργασία, Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης.

[ChBu-90]

Chen C. και Bulfin R., 1990, "Scheduling unit processing time jobs on a single machine with multiple criteria", Comp. Opns. Res. 17, 1-7.

[CNY-89]

Coffman E., Nozari A. και Yannakakis M., 1989, "Optimal Scheduling of Products with two Subassemblies on a Single Machine", Opns. Res. 37, 426-436.

[DeFo-79]

Denardo E. και Fox B., 1979, "Shortest-Route Methods: 1.Reaching, Pruning and Buckets", Opns. Res. 27, 161-186.

[DiSe-88]

Dileepan P. και Sen T., 1988, "Bicriterion Static Scheduling Research for a Single Machine", Omega 16, 53-59.

[DyWo-90]

Dyer M. και Wolsey L., 1990, "Formulating the single machine sequencing problem with release dates as a mixed integer program", Discrete Applied Mathematics 26, 255-270.

[Emmo-75a]

Emmons H., 1975, "One machine sequencing to minimize mean flowtime with minimum number tardy", Naval Res. Logist. Quart. 22, 585-592.

[Emmo-75b]

Emmons H., 1975, "A Note on a Scheduling Problem with Dual Criteria", Naval Res. Logist. Quart. 22, 615-616.

[Evan-84]

Evans G., 1984, "An Overview of Techniques for Solving Multiobjective Mathematical Programs", Management Science 30, 1268-1282.

[Fish-76]

Fisher M., 1976, "A dual algorithm for the one-machine scheduling problem", Mathematical Programming 11, 229-251.

[FLLR-83]

Fisher M., Lageweg B., Lenstra J. και Rinnooy Kan A., 1983, "Surrogate duality relaxation for job shop scheduling", Discr. Appl. Math. 5, 65-75.

[Fox-70]

Fox B., 1970, "Accelerating List Processing in Discrete Programming", J. Assoc. Comput. Mach. 17, 383-384.

[Fox-78]

Fox B., 1978, "Data Structures and Computer Science Techniques in Opns. Res.," Opns. Res. 26, 686-717.

[FrLe-87]

Fry T. και Leong G., 1987, "A bi-criterion approach to minimizing inventory costs on a single machine when early shipments are forbidden", Comput. Opns. Res. 14, 363-368.

[GeKl-74]

Gelders R. και Kleindorfer P., 1974, "Coordinating Aggregate and Detailed Scheduling in the One-Machine Job Shop: I-Theory", Opns. Res. 22, 46-60.

[GeKl-75]

Gelders R. και Kleindorfer P., 1975, "Coordinating Aggregate and Detailed Schedu ling in the One-Machine Job Shop: II-Computation and Structure", Opns. Res. 23, 312-324.

[Grav-81]

Graves S., 1981, "A Review of Production Scheduling", Opns. Res. 29, 646-675.

[GuSe-84]

Gupta S. και Sen T., 1984, "Minimizing the range of lateness on a single machine", J. Oper. Res. Soc. 35, 853-857.

[HaPo-83]

Hariri A. και Potts C., 1983, "An Algorithm for Single Machine Sequencing with Release Dates to Minimize Total Weighted Completion Time", Discr. Appl. Math. 5, 99-109.

[HeKa-62]

Held M. και Karp R., 1962, "A dynamic programming approach to sequencing problems", J. Soc. Indust. Appl. Math. 10, 196-210.

[HeRo-72]

Heck H. και Roberts S., 1972, "A Note on the Extension of a Result on Scheduling with Secondary Criteria", Naval Res. Logist. Quart. 19, 403-405.

[HKS-63]

Held M., Karp R. και Shareshian R., 1963, "Assembly line balancing - Dynamic programming with precedence constraints", Opns. Res. 11, 442-459.

[HoVe-92]

Hoogeveen J., Van de Velde S., 1992, "A new lower bound approach for single-machine multicriteria scheduling", Opns. Res. Letters 11, 39-44.

[HRRW-80]

Huckert K., Rhode R., Roglin O. και Weber R., 1980, "On the Interactive Solution to a Multicriteria Scheduling Problem", Z. Oper. Res. 24, 47-60.

[Ibar-87]

Ibaraki T., 1987, "Enumerative approaches to combinatorial optimization, Parts I,II", Annals of Opns. Res. 10,11.

[John-89]

John T., 1989, "Tradeoff solutions in single machine production scheduling for minimizing flow time and maximum penalty", Comput. and Opns. Res. 16, 471-479.

[Kao-80]

Kao E., 1980, "A Multiple Objective Decision Theoretic Aproach to One-Machine Scheduling Problems", Comput. and Opns. Res. 7, 251-259.

[KaQu-82]

Kao E. και Queyranne M., 1982, "On dynamic programmic methods for assembly line balancing", Opns. Res. 30, 375-390.

[Lawl-64]

Lawler E., 1964, "On scheduling problems with deferral costs", Management Sci. 11, 280-288.

[Lawl-79]

Lawler E., 1979, "Efficient implementation of dynamic programming algorithms for sequencing problems", Report BW 106/79, Stichting Mathematisch Centrum, Amsterdam.

[LHT-92]

Liao C., Huang R. και Tseng S., 1992, "Use of variable range in solving multiple criteria scheduling problems", Comput. and Opns. Res. 19, 453-460.

[LRB-77]

Lenstra J., Rinnooy Kan A. και Brucker P., 1977, "Complexity of machine scheduling problems", Ann. Discrete Math. 1, 343-362.

[Mats-88]

Matsuo H., 1988, "The Weighted Total Tardiness Problem with Fixed Shipping Times and Overtime Utilization", Opns. Res. 36, 293-307.

[Miya-81]

Miyazaki S., 1981, "One Machine Scheduling with Dual Criteria", J. Opns. Res. Soc. Jpn. 24, 31-50.

[MoMa-76]

Morin T. και Marsten R., 1976, "Branch-and-Bound Strategies for Dynamic Programming", Opns. Res. 24, 611-627.

[MoPo-89]

Monma C. και Potts C., 1989, "On the Complexity of Scheduling with Batch Setup Times", Opns. Res. 37, 798-804.

[MuRa-82]

Munro J. και Ramirez R., 1982, "Reducing space requirements for shortest path problems", Opns. Res. 30, 1009-1013.

[NaSa-88]

Naddef D. και Santos C., 1988, "One-Pass Batching Algorithms for the One-Machine Problem", Discrete Applied Mathematics 21, 133-145.

[NSD-86]

Nelson R., Sarin R. και Daniels R., 1986, "Scheduling with multiple performance measures: The one-machine case", Management Sci. 32, 464-479.

[Posn-85]

Posner M., 1985, "Minimizing Weighted Completion times With Deadlines", Opns. Res. 33, 562-574.

[Posn-86]

Posner M., 1986, "A sequencing problem with release dates and clustered jobs", Management Sci. 32, 731-738.

[Posn-88]

Posner M., 1988, "The Deadline Constrained Weighted Completion time Problem: Analysis of a Heuristic", Opns. Res. 36, 742-746.

[PoWa-83]

Potts C. και Van Wassenhove L., 1983, "An algorithm for single machine sequencing with deadlines to minimize total weighted completion time", Eur. J. Oper. Res. 28, 379-387.

[PoWa-85]

Potts C. και Van Wassenhove L., 1985, "A Branch and Bound Algorithm for the Total Weighted Tardiness Problem", Opns. Res. 33, 363-377.

[PoWa-87]

Potts C. και Van Wassenhove L., 1987, "Dynamic programming and decomposition approaches for the single machine total tardiness problem", Eur. J. Oper. Res. 32, 405-414.

[Psar-80]

Psaraftis H., 1980, "A Dynamic Programming Approach for Sequencing Groups of Identical Jobs", Opns. Res. 28, 1347-1359.

[RLL-75]

Rinnooy Kan A., Lageweg B. και Lenstra J., 1975, "Minimizing Total Costs in One-Machine Scheduling", Opns. Res. 23, 908-927.

[RoWh-88]

Rodammer F. και White P., 1988, "A Recent Survey of Production Scheduling", IEEE Transactions on Systems, Man and Cybernetics 18, 841-851.

[Sahn-72]

Sahney V., 1972, "Single-Server, Two-Machine Sequencing with Switching Time", Opns. Res. 20, 24-36.

[Sahn-76]

Sahni S., 1976, "Algorithms for Scheduling Independent Tasks", Journal of the Association for Computing Machinery 23, 116-127.

[SaMa-85]

Santos C. και Magazine M., 1985, "Batching in single operation manufacturing systems", Opns. Res. Letters 4, 99-103.

[ScBa-78]

Schrage L. και Baker K., 1978, "Dynamic programming solution of sequencing problems with precedence constraints", Opns. Res. 26, 444-449.

[SeGu-83]

Sen T. και Gupta S., 1983, "A branch-and-bound procedure to solve a bicriterion scheduling problem", IIE Trans. 15, 84-88.

[Shan-83]

Shanthikumar J., 1983, "Scheduling n jobs on one machine to minimize th maximum tardiness with minimum number tardy", Comput. and Ops. Res. 10, 255-266.

[ShBu-82]

Shanthikumar J. και Buzacott J., 1982, "On the Use of Decomposition Approaches in a Single Machine Scheduling Problem", J. Oper. Res. Soc. Jap. 25, 29-47.

[Smit-56]

Smith W., 1956, "Various Optimizers For Single-Stage Production", Naval Res. Logist. Quart. 3, 59-66.

[SRD-88]

Sen T., Raiszadeh F. και Dileepan P., 1988, "A branch-and-bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness", Management Sci. 34, 254-260.

[Stei-90]

Steiner G., 1990, "On the complexity of dynamic programming for sequencing problems with precedence constraints", Ann. Opns. Res. 26, 103-123.

[Ters-85]

Tersine R., 1985, "Production/Operations Management: concepts, structure & analysis", Second Edition, North-Holland.

[TeVl-88]

Tegze M. και Vlach M., 1988, "Improved bounds for the range of lateness on a single machine", J. Oper. Res. Soc. 39, 675-680.

[Τσατ-93]

Τσατσάκης Ν., 1993, "Το πρόβλημα του συνολικού χρόνου περάτωσης Ν εργασιών σε μια μηχανή με χρόνους εξάρμωσης", Μεταπτυχιακή Εργασία, Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης.

[UsSm-75]

Uskup E. και Smith S., 1975, "A Branch-and-Bound Algorithm for Two-Stage Production-Sequencing Problems", Opns. Res. 23, 118-136.

[Vick-80]

Vickson R., 1980, "Choosing the Job Sequence and Processing Times to Minimize Total Processing Plus Flow Cost on a Single Machine", Opns. Res. 28, 1155-1167.

[ViKa-81]

Villarreal B. και Karwan M., 1981, "Multicriteria Integer Programming: A (Hybrid) Dynamic Programming Recurcive Approach", Math. Prog. 21, 204-223.

[WaGe-80]

Van Wassenhove L. και Gelders F., 1980, "Solving a bicriterion scheduling problem", Europ. J. Oper. Res. 4, 42-48.